

Image Processing
(Lecture notes for EECS 556)
(Student Version)

Jeff Fessler
University of Michigan

January 4, 2018

Chapter 0

Image Processing: Introduction

Contents (student version)

| | |
|---|-------------|
| 0.1 Course logistics | 0.3 |
| Exams | 0.3 |
| Team Projects | 0.4 |
| Homework / collaboration policy | 0.5 |
| Engaged learning | 0.7 |
| 0.2 Introduction to image processing | 0.13 |
| Image | 0.13 |
| Image Processing | 0.13 |
| Applications | 0.14 |
| Audience / scope | 0.14 |
| Software: Octave or MATLAB or JULIA | 0.14 |
| Introduction to major topics | 0.15 |
| Examples: a graphical overview | 0.15 |
| Imaging systems and image formation | 0.31 |
| 0.3 Resources | 0.32 |
| Image processing journals | 0.32 |
| Reference books | 0.32 |
| Web sites with image processing tools | 0.32 |

Acknowledgment

These notes have benefited from modifications and suggestions by Prof. David Neuhoff, Prof. Mike Wakin, Prof. Silvio Savarese, and Dr. Boklye Kim. A few figures were inspired by Prof. Doug Noll's course notes.

The notes have also benefited tremendously from numerous comments by former students in EECS 556, particularly in W15 and W16 when I taught the course in a “flipped” style, and in W17 when Boklye Kim did the same.

0.1 Course logistics

EECS 556: Image Processing

3 credits

Class: Tue & Thu 1:30-3:00PM, 224 GFL

Instructor: Prof. Jeff Fessler fessler@umich.edu <https://web.eecs.umich.edu/~fessler/>

Office hours: Tue & Thu 3-4PM, 4431 EECS

- Course web site: <http://web.eecs.umich.edu/~fessler/course/556>
course notes, homework, code templates, ...
- **Canvas**
quizzes, homework solutions, exam solutions, ...

No textbook to purchase; over 600 pages of course notes.

Prerequisites: EECS 551 (matrix methods), EECS 501 (random processes)

Signals and systems background (EECS 216, EECS 351) helpful.

Objective: Fundamentals of imaging and image processing

Topics: image formation, sampling, interpolation, representation, enhancement, restoration, analysis, and compression.

Exams

Exam 1: Wed. Feb. 21, 6-9 PM, Room **1200/1012 EECS** (seats 78/26)

(more feedback,

Exam 2: Wed. Mar. 28, 6-9 PM, Rooms **1200/1012 EECS** (seats 78/26)

less pressure per exam)

No final exam. Project presentations during **final exam time: Thu. Apr. 19, 4-6 PM** (probably later too), Room TBA

All students must take all exams during the scheduled times. In all work, legibility counts.

Scores may be standardized before computing the final score if the means and standard deviations vary.

The exams will be a combination of “**open notes no electronics**” questions and “open notes with electronics” take-home questions; the latter will include software problems. **Both portions must be completed individually initially (more later).**

Requests for re-grades of exams must be submitted in writing within one week of exam return. All questions may be re-graded.

Letter grades will be assigned using a curve. The cutoff for an A- will be 90% or lower.

Evolving class-by-class topic list:

<http://web.eecs.umich.edu/~fessler/course/556/a/topic.txt>

Team Projects

In lieu of a final exam, students will work in small groups on image processing projects that apply the tools learned in the course as well as using ideas from the contemporary literature.

Project Milestones:

- Fri. Feb. 09 5 PM: form your project teams (at the latest)
- Fri. Feb. 23 5 PM: written project proposal (upload pdf to Canvas)
- Fri. Mar. 23 5 PM: written progress report (upload pdf to Canvas)
- Fri. Apr. 13 5 PM: practice oral presentation with other team (optional)
- Thu. Apr. 19 1 PM: oral presentation files (upload pdf to Canvas)
- Thu. Apr. 19 4-? PM: team oral presentations to entire class (Room TBA)
- Tue. Apr. 24 5 PM: written project reports (upload pdf to Canvas)
includes peer evaluation form (at end of report)
- Tue. Apr. 24 5 PM: individual project self-evaluation (google form)

Related topics

- The treatment of image coding may be brief. Students with primary interests in compression should take EECS 553 (Theory and Practice of Data Compression). JPEG 2000 is a wavelet-based coding method but neither wavelets nor 553 is a prerequisite for this course, so a detailed treatment of the JPEG 2000 standard is infeasible. We will cover the general ideas of transform coding.
- The computer vision and image processing fields have substantial overlap. UM has excellent computer vision courses (EECS 442, 504, 542) so topics *and projects* in EECS 556 need to use image processing material beyond what is covered in the computer vision courses.
- The same goes for machine learning: projects in EECS 556 need to use image processing material, not just material that is covered in machine learning courses.

Homework / collaboration policy

Solutions will be provided for all homework problems.

No late homework assignments will be accepted.

Homework/computer project policy. You must attempt to solve all homework problems by yourself, and implement all computer programs by yourself. Copying homework solutions or code from another student, or from solutions from previous semesters, will be considered violations of the engineering honor code. However, after making a genuine attempt to solve the homework problems, you are encouraged to discuss the answers with other students currently enrolled in 556 to check the answers and compare solution approaches. (Indeed this will be a formal part of the engaged learning process.) After such a discussion, you may rewrite your answer as long as you do so individually, without copying the solutions of other students and without referring to solutions from previous terms. Basically, the answers you turn in should reflect your own level of understanding, not someone else's. All solutions and code submitted must be generated by the person whose name appears on the assignment.

Some homework problems may be due before class based on the reading assignments. Some homework problems may be in-class work, possibly in teams, and graded in part based on class participation.

Homework grading

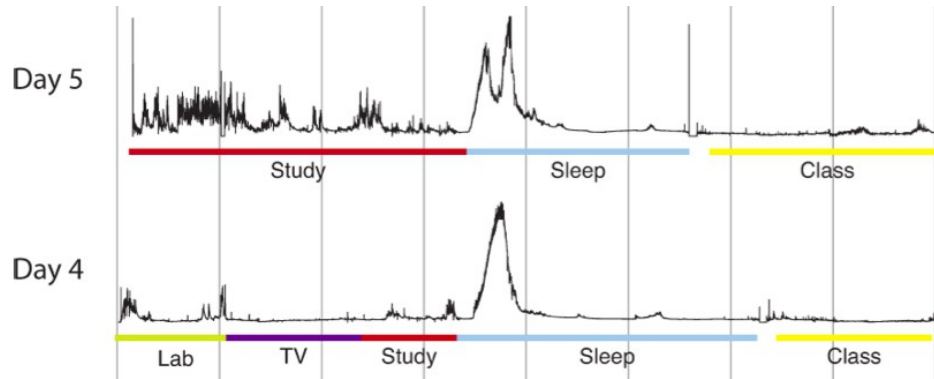
The GEO contract requires that a GSI title be given to any graduate student who grades papers “in a manner that requires subjective evaluation above and beyond the mechanical or routine comparison of submitted papers or examination with answers, responses, or elements predetermined as correct or acceptable.” Past use of graders has led to grievances. In compliance with this GEO contract, rather than evaluating solution accuracy, the grader (and I) will be assessing effort and honesty (more on that later).

Substantial portions of the homework assignment points will be coding problems (Matlab/Octave/Julia). The points assigned to some of these problems may far outweigh those of other homework problems. The goal is to develop both analytical skills and computational skills related to imaging. Hopefully many of those problems will be submitted to an autograder and full points will be earned for a correct solution (with unlimited attempts), otherwise zero points.

From M Poh, M Swenson, R Picard: "A wearable sensor for unobtrusive, long-term assessment of electrodermal activity."

IEEE Tr. on Biomed. Engin., 57(5):1243-52, May 2010.

<http://dx.doi.org/10.1109/TBME.2009.2038487>



14th-century manuscript illustration showing Henry of Germany lecturing to university students in Bologna.

Artist: Laurentius de Voltolina ([source](#))



Engaged learning

Traditional approach in humanities:

- Students read materials *before* class
- Class time: interactive discussion with professor and classmates (peer instruction)
- Assessment based on class participation, written essays and possibly exams

Traditional approach in engineering:

- Week 1: Professor lectures on topic (typically students are passive)
 - Possibly a couple questions are asked by students during class
 - Maybe the professor asks to students a few questions
 - Typically just a few students answer most of those questions
- Week 2: Student reads text, works on homework problems, submits solutions
- Week 3: Grader grades homework
- Week 4: Graded homework returned to student
Undesirably long delay between student action and feedback

Engaged learning approach:

- Students read *and interact* with “textbook” before class
 - NB (Nota bene = “note well”) interactive PDF annotation: nb.engin.umich.edu
 - Students engage with material by posting questions that show thoughtful reading
 - Students engage with material by answering questions (peer instruction) that show thoughtful reading
 - Students answer some “reading questions” to assess comprehension and provide accountability (e.g., via **Canvas** “quizzes”)
 - Students attempt homework problems individually *before* class
- Class time:
 - Facilitated discussion about any remaining areas of confusion in reading (inquiry-driven, no wasted time on simple topics)
 - Address issues arising from answers to “reading questions”
 - Team-based final solutions to homework problems
(Teams will be assigned and changed many times over the term; you will interact with most other students.)
 - Other team-based derivations and problem solving
 - Immediate feedback on correct homework solutions

Engagement

25% of grade is “homework” (a broad category covering everything but exams and the final project.)

- 5% for NB annotations (thoughtful questions and/or answers); see p. 0.10
- 5% for “reading questions” (RQ), due at **11AM** the morning before each class (via **Canvas** “quizzes” with no time limit)

Each RQ is worth “5 points” by default in **Canvas**, but:

approximately 120 RQs so each is worth $5\%/120 = 0.042\%$ of overall score.

Ideally you’d earn partial credit for each attempted quiz question, but **Canvas** does not support this feature.

- 15% for homework problems / in-class work / etc. **HW Process:**
 - Answer problems yourself prior to Thu. class as completely as possible
 - Scan and upload draft (individual) solutions to **gradescope** by **9:00AM** each Thu., with automatic extension to 1:30PM. No later submissions accepted.
 - Grader checks effort / timeliness / completeness on **gradescope**, *not* correctness
 - Bring paper copy of your solutions to Thu. class.
 - Rewrite / correct your answers based on group work in Thu. class. (Self grading.)
 - Correct solutions revealed during Thu. class, and released on **Canvas** after class.
 - Write short self reflection (**form on course web site**)
(Did I make a minor arithmetic error? Did I have a conceptual misunderstanding?)
 - Scan and upload your self reflection *and* self-graded / corrected solution to **gradescope** by **1PM** each Fri., with automatic extension to 5PM. No later submissions accepted.
 - HW scores for **gradescope** portion based on effort and honesty - focus on learning not assessment
HW grading rubric on web page
- Some HW problems may be graded traditionally, including auto-graded coding problems.

Action:

- Create account on **gradescope**
- Use course entry code **93Y5WN** and use your UM email and UM Student ID and your given name and family name
Correct **gradescope** registration required to earn HW points!
- Look for confirmation email from `team@gradescope.com`
- Follow email link to create your password.
- Review **gradescope scan/pdf submission process**.

For exams:

- Partly in-class evening exam (paper materials only); partly take home
- 75% of score based on individual work (submitted to **gradescope** by deadline given *before* class)
- 25% of score based on group answer completed during class the day after the exam. (Why? See [6].)
(If you miss that class for any reason your exam score will be 100% based on individual work.)
- Answers given during class after group answers completed.

YouTube videos of Prof. Eric Mazur discussing engaged learning: [overview](#) [why you](#) [NB / in-class](#) [homework](#)

Participation

Because of the emphasis on teamwork, it is important that all team members attend and participate in class. Due to the collaborative nature of the activities, it is impossible to make up any team activities, particularly homework solving / self-reflection and group exam answers. (The same, incidentally, is true in the professional world.)

If you miss class during which we discuss a problem set, you will miss the benefit of the discussion and the opportunity to learn from (and teach) others. You also will miss being able to accurately assess your own learning. You may turn in your individual homework answers *in advance* (by the 9AM deadline) for half credit. If you fail to hand in a problem set altogether, your problem set score will be zero. One “half credit” answer set will be excused for a documented medical reason or for presenting research at a conference.

No “low score” will be dropped because HW here is mostly scored based on effort, not correctness.

Caution. **Canvas** has a dumb default where ungraded assignments (even unsubmitted ones) are not included in the total score. Unsubmitted assignments earn 0 points, so to see your score properly you must you the **Canvas** “WhatIf” feature:

<https://community.canvaslms.com/docs/DOC-10631-421255065>

Annotations: why

Portions of this text by Prof. Steven Yalisove who in turn borrowed from Prof. Eric Mazur.

Overview

Students will be assigned reading weekly. We will post a pdf version on the course web site and an annotatable version on nb.engin.umich.edu. Students will annotate sections of the text there. The annotations will be visible to the entire class and instructor. Students can rephrase difficult to understand concepts, fill in confusing steps in derivations, identify errors, provide better ways to illustrate the ideas than the examples in the book, ask great questions, and answer (correctly) other's questions. Students will be graded based on the quantity, quality and timeliness of their annotations. One annotation per reading assignment is too little and more than 20 annotations is probably too much. Providing 5 excellent annotations, including correct answers to another student's good questions, will earn full credit for the week as long as they are completed in time and cover the material reasonably completely (*i.e.*, are not just all in the first section).

Why annotate?

Annotating the text helps you and us. First, you get practice reading technical material. Once you graduate, books (and papers) will be your primary vehicle for learning and learning does not stop when you graduate. If you can learn from books, you have mastered an important lifelong skill. Second, by reading with attention and with an inquiring mind, you take ownership of your learning. That skill too will be useful for your whole life (you may want to start reading ahead in some of your other classes to get more out of them; you'll have to read those books at some point anyway!). Third, by annotating the text, you reverse the roles of student and teacher: for a change you are the one determining what's wrong or confusing. In a traditional class, the teacher tells you what is wrong or confusing about your work. When you annotate the text because you are confused, you have identified a problem in the text: you are right and the author is wrong! By communicating that confusion to others, you create an opportunity to address the confusion and learn. If many people in the class express confusion about a particular topic, we will know that we need to address that confusion in class or online.

How (much) should I annotate? (what we expect)

Without lectures, the reading is your initial (and in some sense primary) exposure to the content of this class. It is therefore essential that you study each chapter with an inquisitive mind. Your annotations can either be queries, comments, or answers/reactions to queries or comments posted by others. When we look at your annotations we want them to reflect the effort you put in your study of the text. It is unlikely that that effort will be reflected by just one or two annotations per chapter, unless your annotations are unusually thoughtful and stimulate a deep discussion. On the other extreme, 20 per chapter is probably too many to be practical. Somewhere in between these two extremes is about right.

Annotations: how

Your annotations will be evaluated based on quality, quantity, and timeliness.

Your goal is demonstrating substantive, thorough, timely, and thoughtful reading of the text.

- Insufficient: “This confuses me”
- Better: “This equation appears to contradict (previous equation) or seems counter-intuitive because ...”
- Insufficient: Yes/No answers to questions without explanation.
- For more examples, see [\[this link\]](#)

We calculate the score as follows: Final score (per assignment) =

Average of the top 5 quality × timeliness scores - 1 point for lack of reasonably even coverage (of questions).

| Quality score | Description and criteria |
|---------------|---|
| 6 | Demonstrates thorough and thoughtful reading and insightful interpretation of the text. |
| 2 | Demonstrates reading, but little or only superficial interpretation of the text. |
| 0 | Does not demonstrate any thoughtful reading of the text. |

Quantity

Each student must enter a minimum of 5 annotations per reading assignment. (*This number is subject to change.*)

Submit 5-10 annotations for each reading assignment. We will average the top 5 scores for your grade. If it appears you did not read most of the sections you will lose 1 point from your final score.

Timeliness: a multiplicative weighting factor applied to the Quality score for each individual annotation

| Factor | Question or comment (responses get an additional 3 days beyond indicated deadline) |
|--------|---|
| 1 | Submitted by Reading deadline: by 11AM each class day. Or submitted by end of week (Sunday at midnight) IF the annotation is the answer to someone else’s question that was not answered in class. |
| 0.5 | Submitted by end of week (Sunday by midnight) |
| 0 | If submitted after end of week (Sunday by midnight) you will not earn credit for the annotation (but answers from others may help you learn) |

The “timeliness” score is machine-computed and if you miss a deadline by just one second, your score decreases. There are no resources to evaluate this criterion by hand or to make individual adjustments, so the computer will be the unbiased judge of time!

Course evaluation comments from W15

(From the brave pioneers.)

- “I was skeptical about the ‘engaged learning’ thing; however, after the one full day of lecture we had, while very good, I concluded that group-work was far more useful and engaging than sitting in a chair staring at the front of the room.”
- “I found the group work (as well as annotations) guided by the instructor to be particularly helpful in gaining a deeper understanding of the material.”
- “I couldn’t be happier with this engaged method. Sure, it is a lot more work than the average engineering lecture-based course. However, rather than semi- learning material throughout the term and cramming at the end for a final (in which I will likely forget some of the material later), I am really gaining a deep understanding of everything presented each time we meet and know that I will understand/maintain the material from this course much better than with a traditional teaching method.”
- “This class was a lot of work. Far more than either of my other 3-credit classes this semester. The workload was really more appropriate for a 4- credit class, and adding a 4th credit would mean an extra hour of lecture, so it would really be a win-win. I’m sure it’s a bureaucratic nightmare to get the credit level changed, but it might be worth looking into.”

0.2 Introduction to image processing

Image

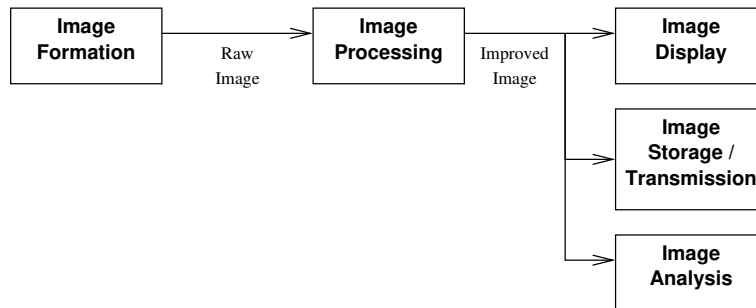
American Heritage Dictionary, 3rd edition:

- A reproduction of the form of a person or an object, especially a sculptured likeness.
- Physics. An optically formed duplicate, counterpart, or other **representative reproduction** of an object, especially an optical reproduction of an object formed by a lens or mirror.
- One that closely or exactly resembles another; a double: He is the image of his uncle.
- Photographer **Diane Arbus** said: “A picture is a secret about a secret, the more it tells you the less you know.”
- ...

Image Processing

Image processing is the application of 2D signal processing methods to images.

In one sense, it is anything that goes in the box labeled “image processing” in diagrams like those below.



More broadly, judging by the topics at the IEEE International Conference on Image Processing (ICIP), and in the IEEE Transactions on Image Processing (T-IP), the field of image processing also includes many aspects of **image formation**, **image display**, **image compression**, and **image analysis**.

Applications

- Medicine (radiological diagnoses, microscopy)
- Defense (radar, sonar, infrared, satellites, etc.)
- Robotics / machine vision (*e.g.*, “intelligent” vehicles)
- Human / computer interfaces (face / fingerprint “recognition” for security, character recognition)
- Compression for storage, transmission from space probes, etc.
- Entertainment industry
- Manufacturing (*e.g.*, part inspection)

The course emphasis will be fundamental methods and principles that apply across many applications.

Audience / scope

- This course targets graduate students who want to understand the mathematical foundations underlying advanced image processing algorithms and then eventually develop *new* algorithms.
- It is not a “how to” course on using image processing software like **GIMP**.
- Understanding the principles of existing methods is essential for developing new methods.

Software: Octave or MATLAB or JULIA

All of the examples shown later in this chapter were generated with MATLAB or with the freeware program **Octave**
<http://www.gnu.org/software/octave/>

Eventually I hope to use JULIA with Jupyter notebooks instead.

Students may use MATLAB or Octave or JULIA for assignments this term.

Autograder problems will be designed to accept both MATLAB and JULIA files this term.

Introduction to major topics

- **Image formation**

How are images created in the first place?

Includes **image reconstruction** from line-integral projections (**computed tomography**).

And reconstruction from Fourier samples (MRI, Radar, ...)

Continuous-space 2D signals and systems theory

- **Image interpolation (resizing, rotation, warping)**

- **Image registration or alignment**

- **Image enhancement**

Accentuate certain desired features for subsequent analysis or display.

- **Image restoration / image filtering**

Remove or minimize known degradations (*e.g.*, image blur due to faulty Hubble telescope optics)

(However, the repair of the Hubble mirror reminds us that analog solutions can sometimes prevail over digital ones. We will see why in our discussion of the limitations of image restoration.)

- **Image analysis / computer vision**

Extracting information from images, can be quantitative (*e.g.*, object dimensions or position) or symbolic (*e.g.*, character recognition).

- **Image compression** for efficient storage, transmission.

Entire courses could be devoted to each of these topics. Rather than attempting to cover as many methods for image processing as is possible, we will discuss the fundamentals in each of the above categories, hopefully in sufficient depth to facilitate subsequent independent reading of the image processing literature.

The topics above are the *applications*; much of the course will focus on the *methods*:

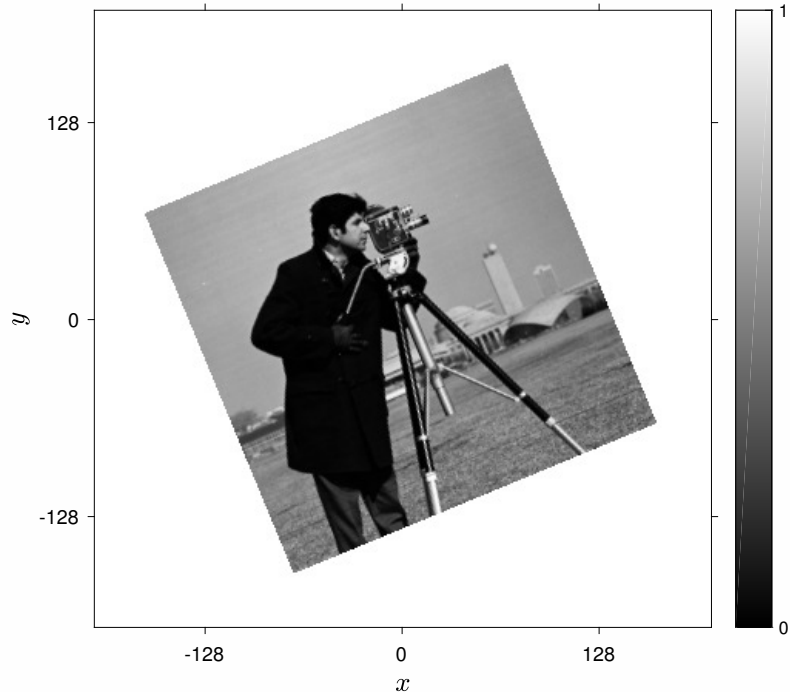
Fourier transforms, filtering, sampling, wavelets, random processes, etc.

Examples: a graphical overview

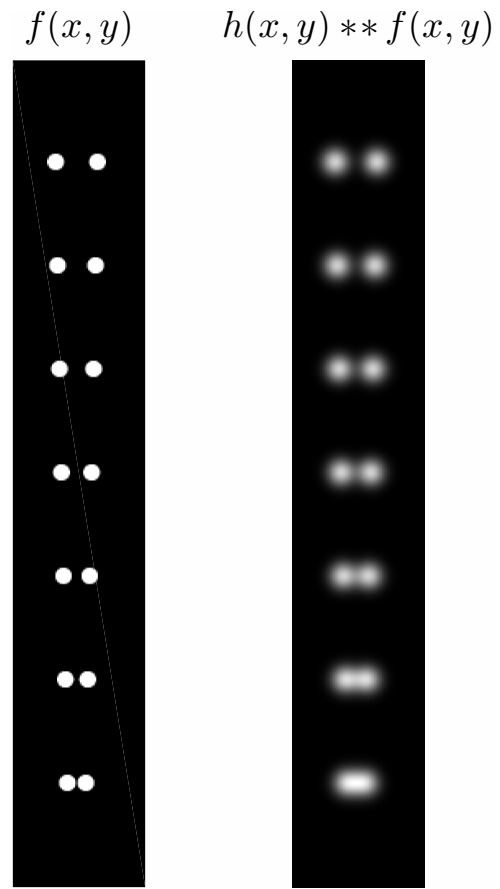
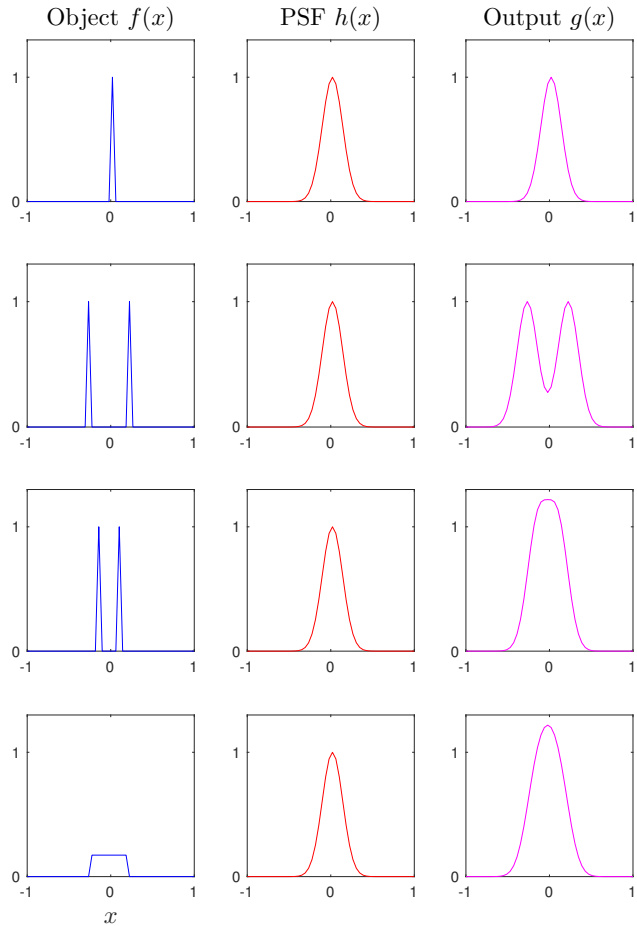
A fun aspect of image processing is that one can *see* the topics!

Spatial transformations of images (e.g., rotation)

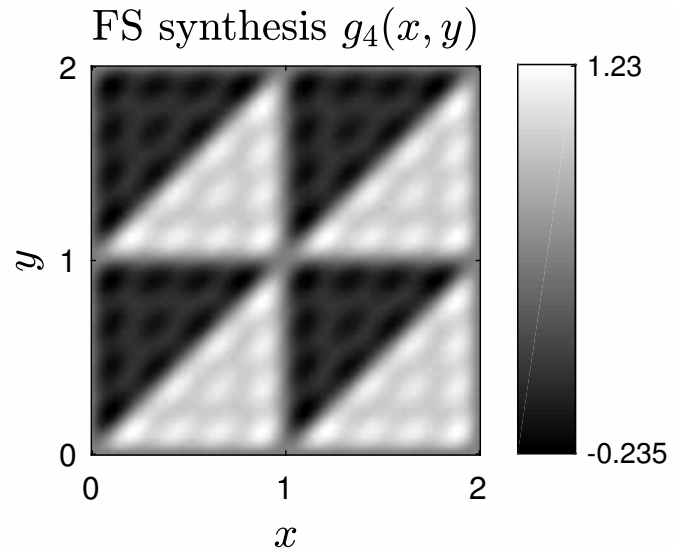
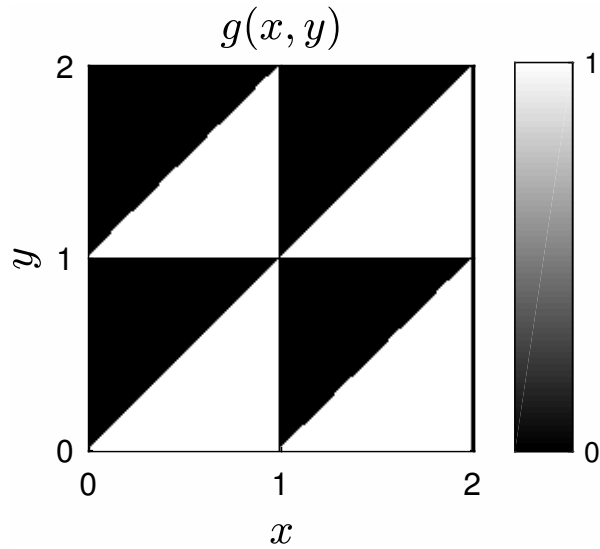
$$g(x, y) = f(x \cos(\pi/8) + y \sin(\pi/8), -x \sin(\pi/8) + y \cos(\pi/8))$$



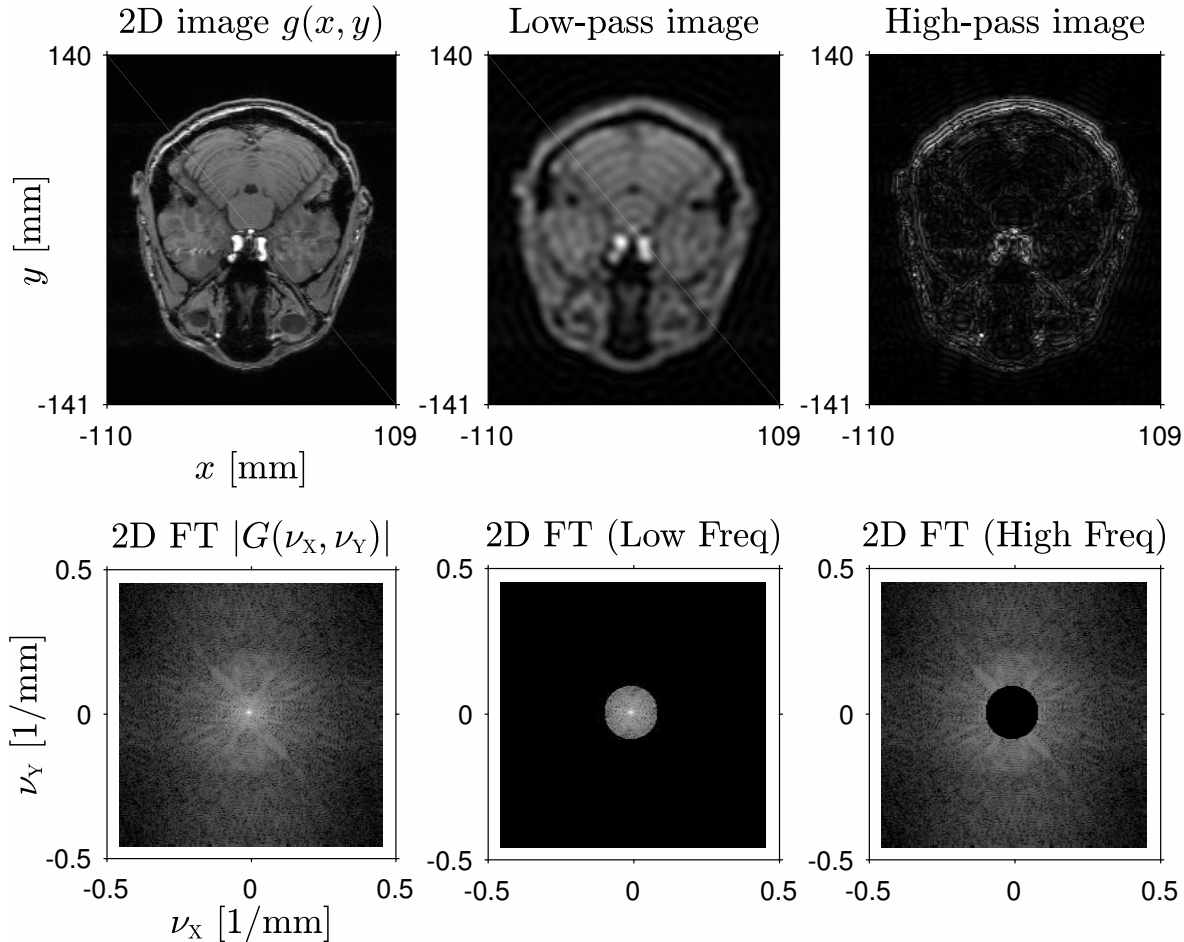
Convolution / blur / PSF



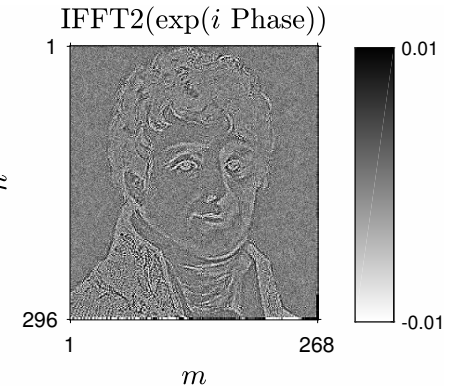
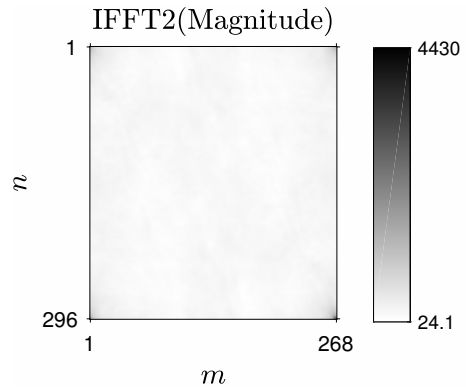
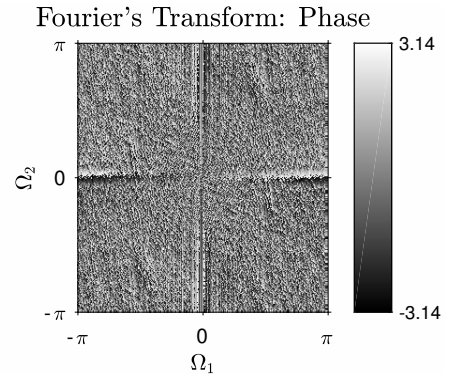
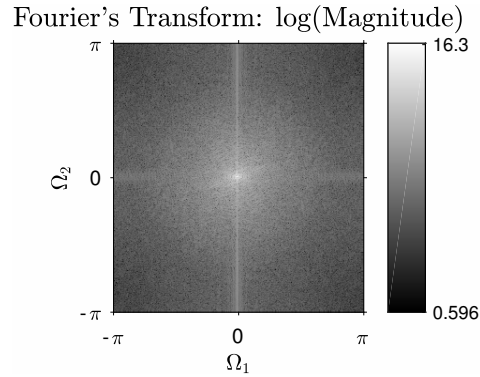
2D Fourier series

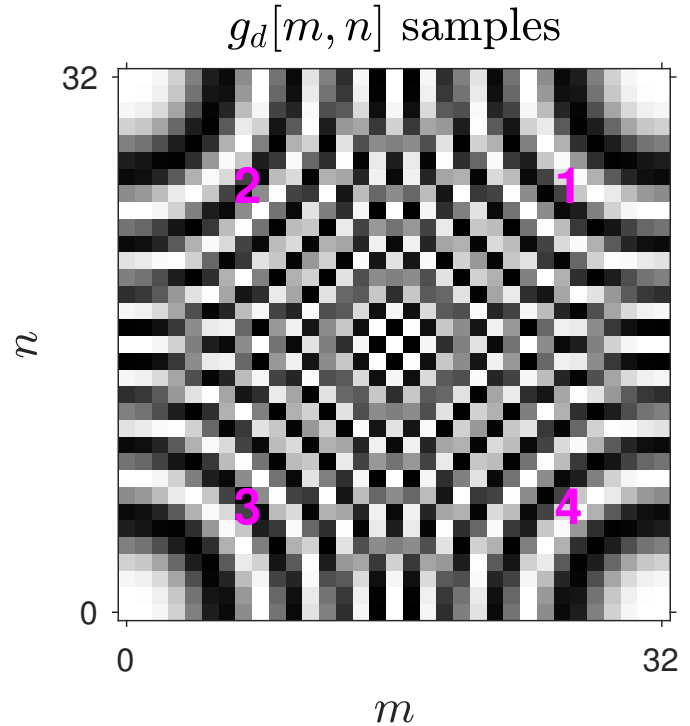
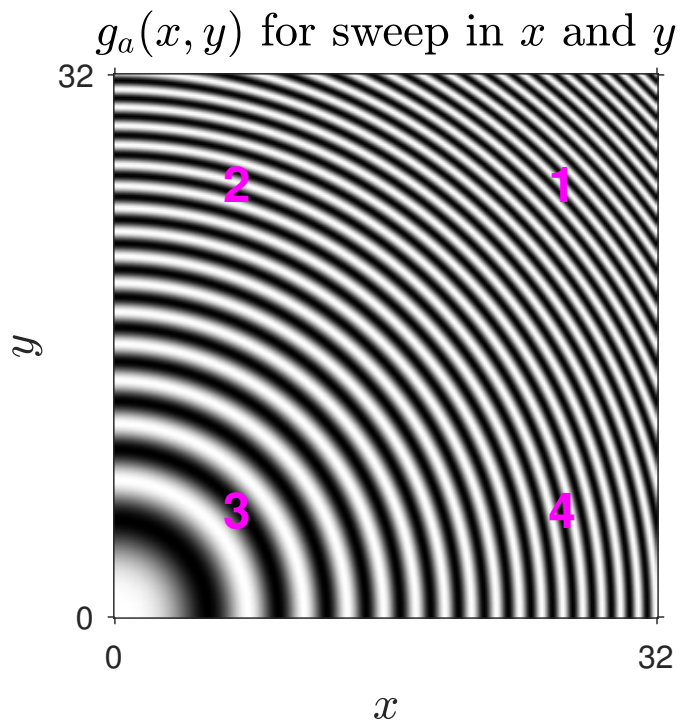


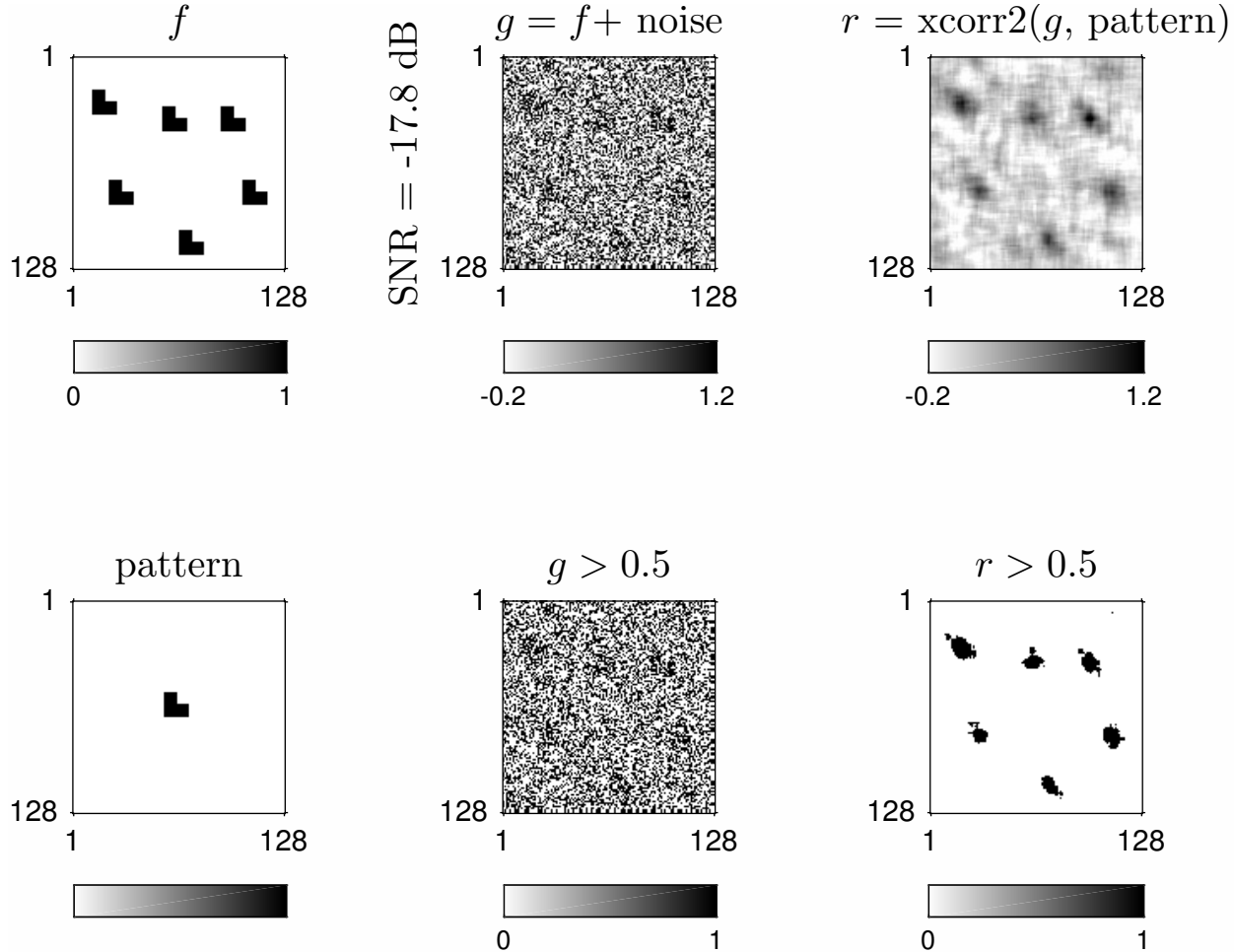
2D Fourier transform: low and high spatial frequencies



2D Fourier transform: magnitude and phase



2D sampling / aliasing

2D correlation / matched filter / threshold-based image segmentation

2D filter design

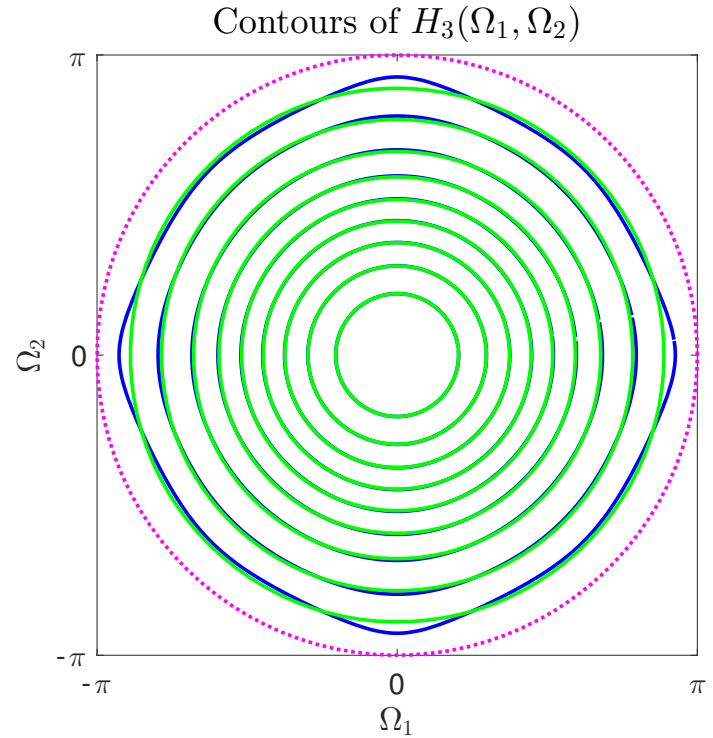
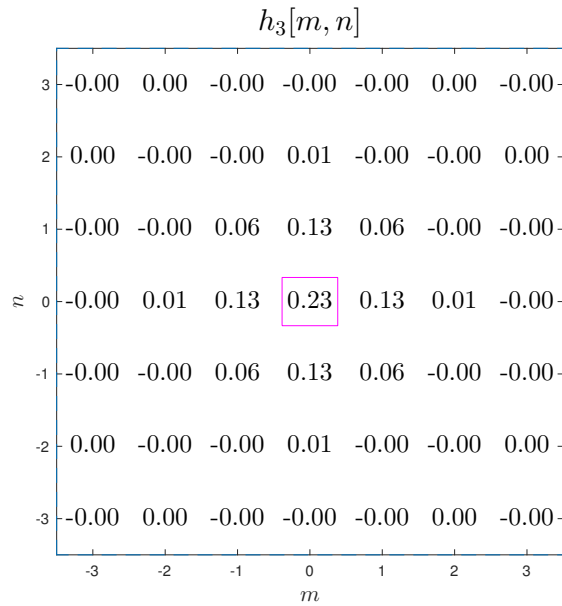
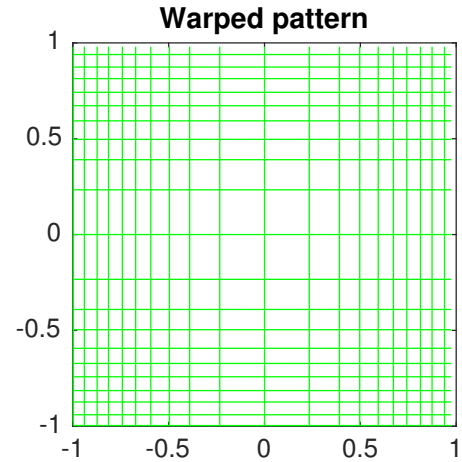
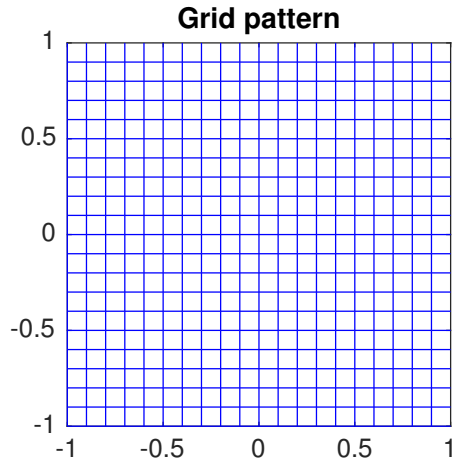
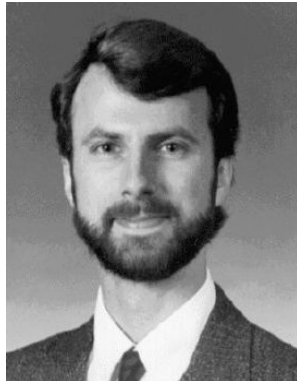


Image interpolation



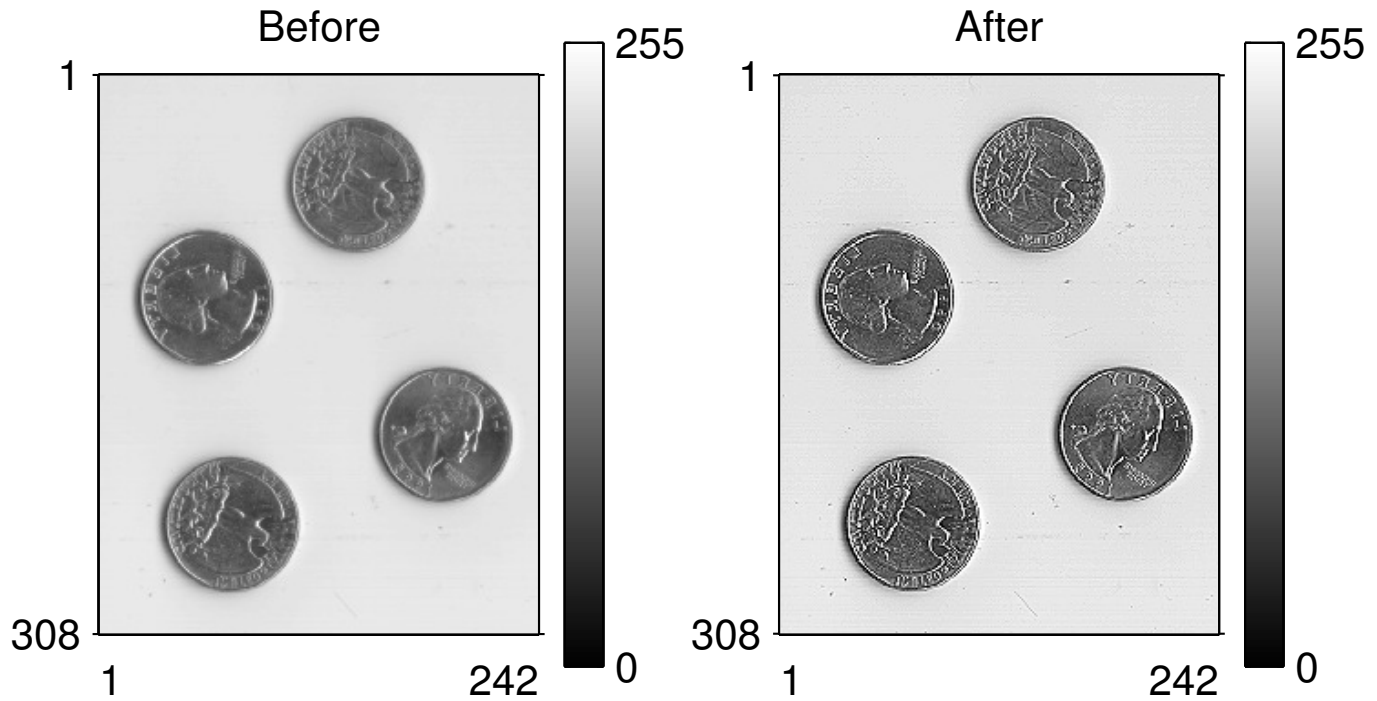
A professor



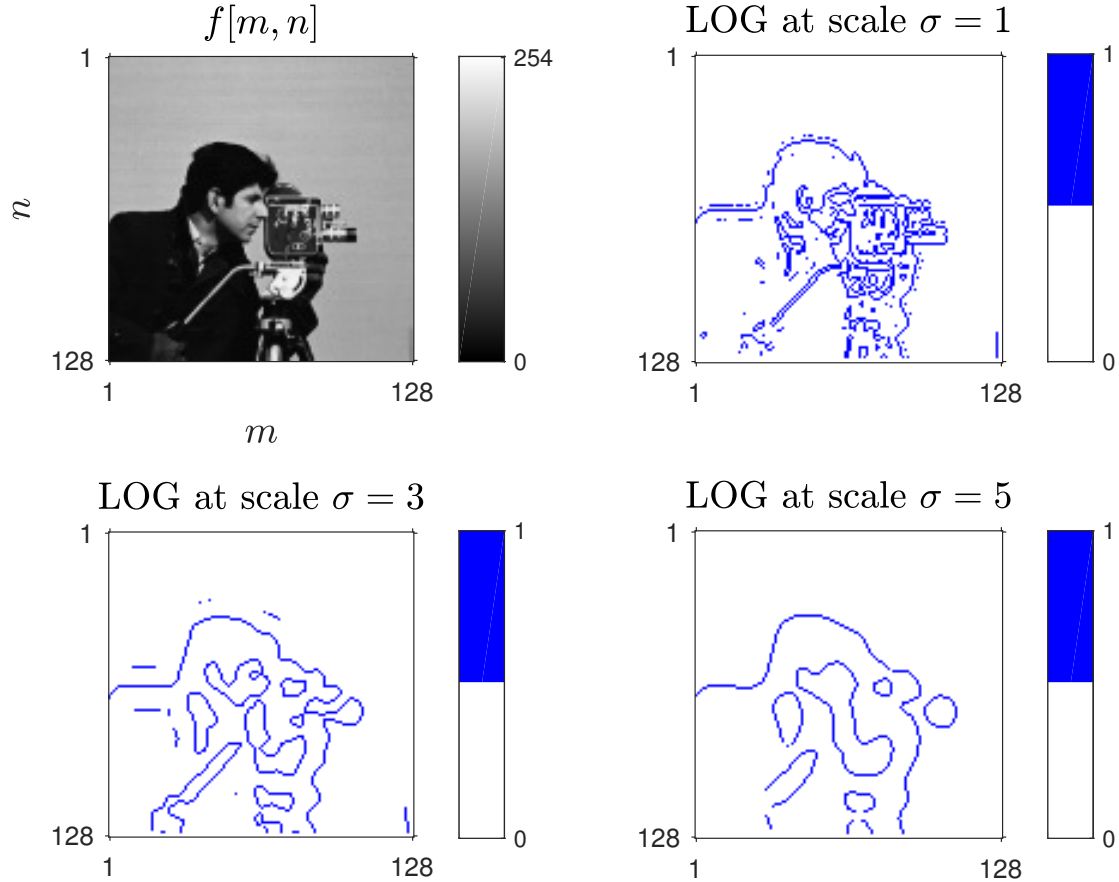
A warped professor



Image enhancement



Edge detection / scale



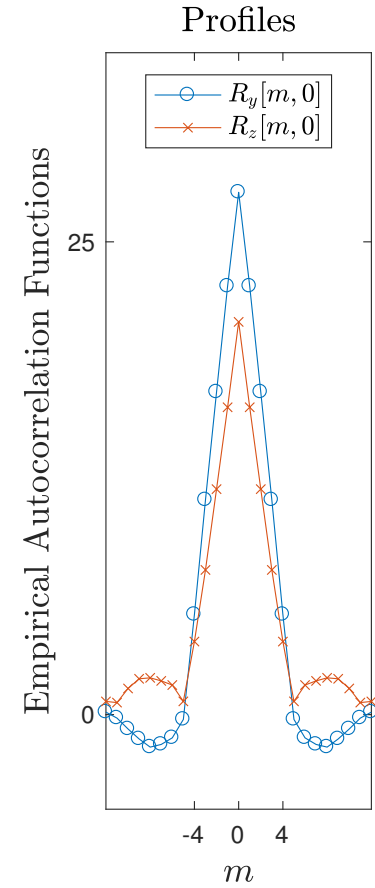
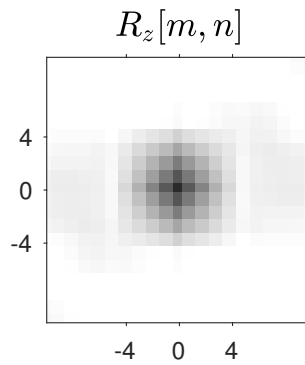
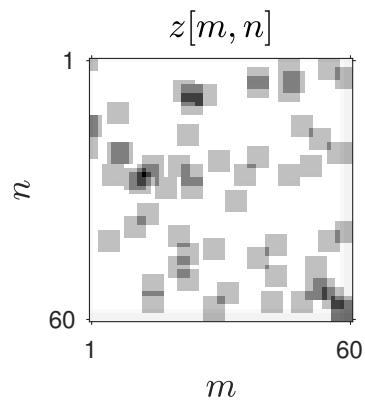
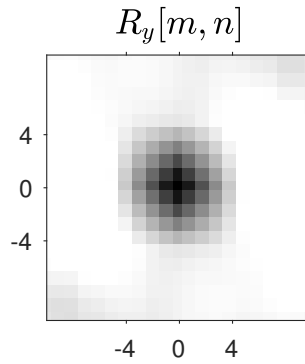
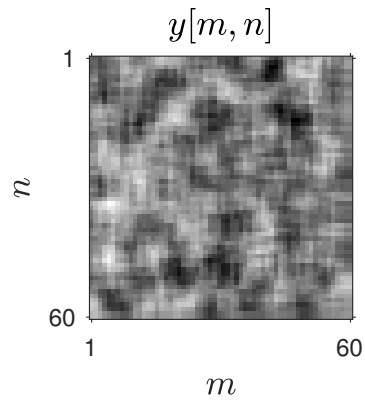
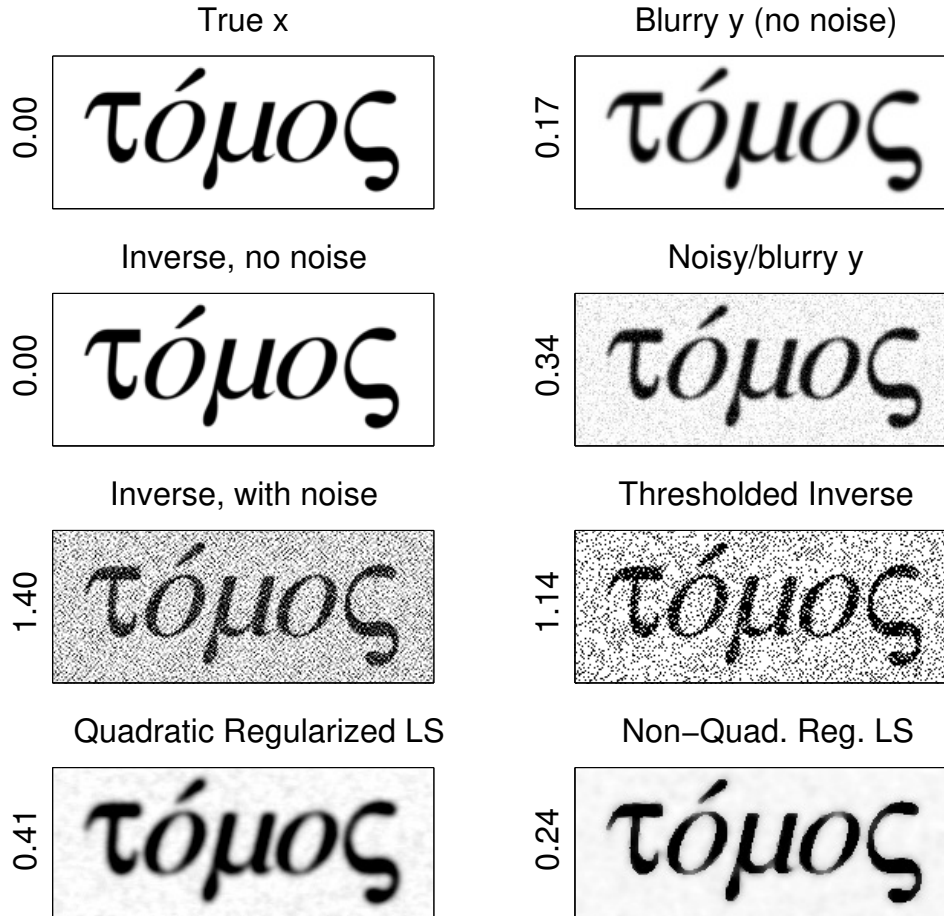
2D random processes / autocorrelation

Image restoration (deblurring / denoising)



e.g., Hubble space telescope v1.

Image comparison

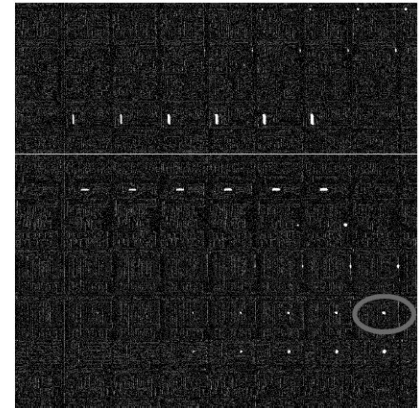
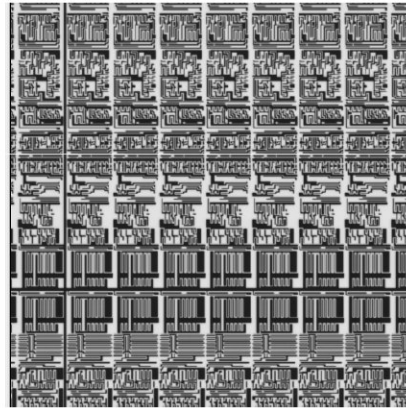
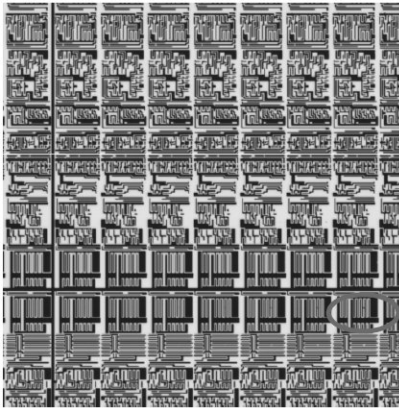


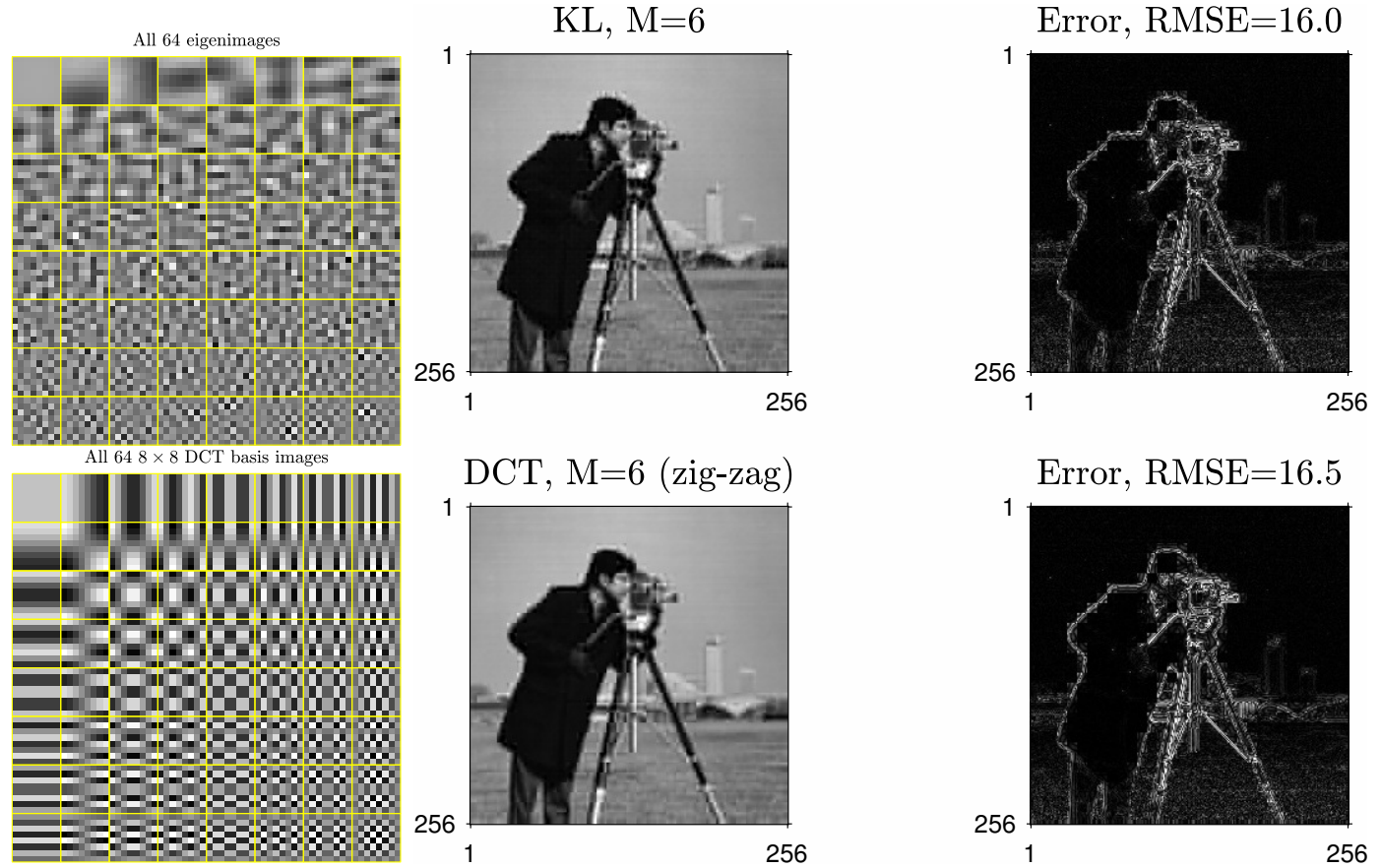
Image data from KLA-Tencor. Two high-resolution images of silicon wafers for semiconductor defect detection.



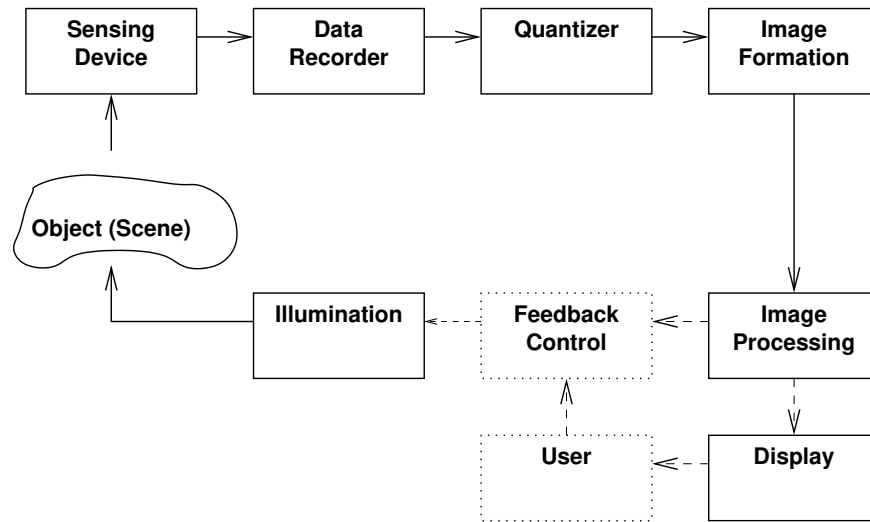
In recent years, KLA-Tencor and/or Apple have sponsored prizes for best team project(s).

News about past EECS 556 course project winning teams: [2011](#) [2013](#) [2014](#) [2015](#) [2016](#) [2017](#)

Image compression / transform coding (KLT, DCT)



Imaging systems and image formation



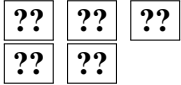
The **illumination** can be of many forms, each of which associated with corresponding applications.

- optical: coherent, incoherent, structured (*e.g.*, microscopes, telescopes, infrared systems, etc.)
- electromagnetic waves (radar, microwave, etc.)
- acoustic (medical ultrasound, sonar)
- ionizing radiation (x-rays, gamma rays, protons, neutrons, etc.)

(In cases like astronomical imaging, the scene itself is the illumination.)

From [8, p. 296], “A common function of an imaging system is to provide an observer with visual information that is more detailed and/or more accurate than could be obtained with the unaided eye.”

What are some characteristics of good imaging systems?



Note: these course notes contain many such embedded questions that usually will be answered in class. If you miss an answer, you are always welcome to come ask in office hours.

0.3 Resources

Image processing journals

Image processing papers appear in (too) many journals, especially in the list below, but also in other SIAM journals, Inverse Problems, Physics in Medicine and Biology, etc., because anyone can easily “process images” these days. Well-informed image processing researchers skim the table of contents of some of these journals regularly.

- IEEE Trans. Image Processing
- IEEE Trans. Signal Processing
- SIAM J. on Imaging Science
- IEEE Trans. Computational Imaging
- IEEE Trans. Medical Imaging
- IEEE Trans. Pattern Analysis and Machine Intelligence
- Journal of the Optical Society of America
- Optical Engineering
- Medical Image Analysis

Reference books

- Texts that were previously used / recommended for this course were [1] [2] (but students reported that they were unnecessary).
- References I use frequently are [3] [4] [5].
- Some classic image processing books are [9] [10].
- Some other interesting books are [11, 12] [13] [14] [15] [16] [8].
- Some books are available online: [17] [18].

Web sites with image processing tools

- <http://fiji.sc/Fiji>
- <http://developer.imagej.net/>
- <http://www.gimp.org/>

Bibliography

- [1] A. K. Jain. *Fundamentals of digital image processing*. New Jersey: Prentice-Hall, 1989. ISBN: 978-0133361650.
- [2] J. S. Lim. *Two-dimensional signal and image processing*. New York: Prentice-Hall, 1990. ISBN: 978-0139353222.
- [3] R. N. Bracewell. *Two-dimensional imaging*. New York: Prentice-Hall, 1995.
- [4] R. Bracewell. *The Fourier transform and its applications*. New York: McGraw-Hill, 1978.
- [5] J. W. Goodman. *Introduction to Fourier optics*. New York: McGraw-Hill, 1968. ISBN: 978-0974707723.
- [6] C. E. Wieman, G. W. Rieger, and C. E. Heiner. “Physics exams that promote collaborative learning”. In: *The Physics Teacher* 52.1 (2014), 51–3.
- [7] L. P. Yaroslavsky. “Compression, restoration, resampling, ‘compressive sensing’: fast transforms in digital imaging”. In: *J. of Optics* 17.7 (2015), p. 073001.
- [8] J. W. Goodman. *Statistical optics*. New York: Wiley, 1985.
- [9] R. C. Gonzalez and R. C. Woods. *Digital image processing*. Reading, Mass.: Addison-Wesley, 1992.
- [10] A. Rosenfeld and A. C. Kak. *Digital picture processing*. 2nd ed. Vol. 2. New York: Academic, 1982.
- [11] S. L. Tanimoto. *An interdisciplinary introduction to image processing: Pixels, numbers, and programs*. MIT Press, 2012.
- [12] A. McAndrew. *A computational introduction to digital image processing*. 2nd Edition. CRC, 2016.
- [13] D. Marr. *Vision*. San Francisco: Freeman, 1982.
- [14] M. Bertero and P. Boccacci. *Introduction to inverse problems in imaging*. London: IoP, 1998.
- [15] J. C. Russ. *The image processing handbook*. Boca Raton, FL: CRC Press, 1999.
- [16] M. Sonka, V. Hlavac, and R. Boyle. *Image processing, analysis, and machine vision*. 2nd ed. Pacific Grove, CA: PWS Pub., 1999.
- [17] W. K. Pratt. *Digital image processing*. New York: Wiley, 2007.
- [18] C. A. Bouman. *Model based image processing [A guide to the tools of]*. ., 2013.

Chapter 1

2D “Continuous Space” Signals and Systems

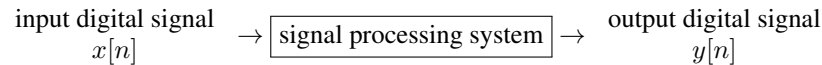
Contents (student version)

| | |
|---|-------------|
| 1.0 Introduction | 1.2 |
| 1.1 Signals | 1.4 |
| Definition of signals/images | 1.4 |
| Periodic images | 1.8 |
| Simple image transformations | 1.16 |
| Important 2D signals | 1.23 |
| Simple image classes | 1.28 |
| 2D Dirac impulse | 1.31 |
| 1.2 Systems in 2D (Imaging systems) | 1.40 |
| Classification of systems | 1.42 |
| Image rotation is a separable operation | 1.46 |
| Impulse response | 1.51 |
| Shift invariance | 1.57 |
| 1.3 LSI systems and convolution | 1.59 |
| LSI system properties via PSF | 1.61 |
| Magnification and LSI systems | 1.65 |
| Convolution examples | 1.67 |
| Summary | 1.68 |
| 1.4 Appendix: Vector Spaces and Linear Operators (<i>skip</i>) | 1.68 |

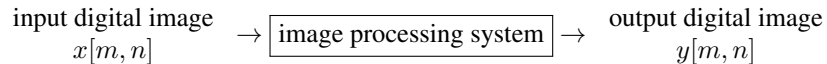
1.0 Introduction

This chapter begins to lay the foundation for 2D signal processing, *i.e.*, image processing.

In an undergraduate introductory digital signal processing course, signal processing often is represented using the following (somewhat) general block diagram.

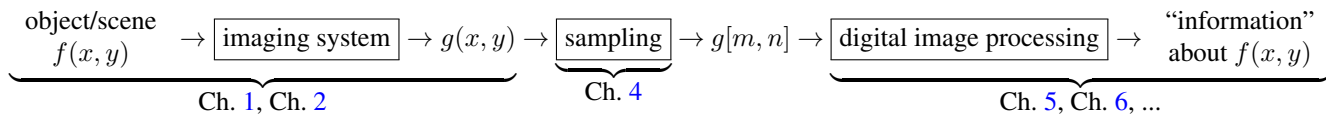


One might approach digital image processing by simply generalizing the signals to two dimensions as follows.



However, except for computer-generated images (like digital animation), a digital image $g[m, n]$ nearly always originates from some (continuous) real-world scene, say $f(x, y)$, recorded by an imaging system that includes “analog” components like lenses. Sometimes our goal is to process $g[m, n]$ to make a different (often “better”) image, but equally often our goal is to extract some “information” from $g[m, n]$ that relates to the original scene $f(x, y)$. Thus, for a complete view of image processing, one must be familiar with the basic principles of such imaging systems. Many such systems are (approximately) linear and shift-invariant, so many of the classical “signals and systems” concepts apply, albeit generalized to 2D (or 3D) rather than in 1D.

So our first goal is a brief review of 2D “continuous” or “analog” signals and systems. (Essentially EECS 216 in 2D).



Overview

The following outline summarizes the main topics that will be covered in Ch. 1 through Ch. 4.

- **2D signals / images**

- Definitions
- Signal classes (even, odd, periodic, ...).
- Simple signal transformations (shift, scale).
- Important signals (Dirac impulses, sinusoids)

New: circular symmetry, separable.
New: rotation, warping.

- **2D systems**

- System classes: (linear, stable, invertible, causal, static, shift-invariant).
- Impulse response (**point-spread function**)
- Linear shift-invariant (**LSI**) systems
- Superposition integral, **convolution**, and properties.
- LSI system properties in terms of PSF

New: rotation invariant.

New: magnification.

- **2D signal representation** (Ch. 2)

- Orthogonal bases
- Fourier series / eigenfunctions / properties

- **2D Fourier transform** (Ch. 2)

- 2D FT properties (convolution etc.)
- Hankel transform

New: rotation, separability, circular symmetry.

- **2D sampling** / recovery via interpolation (Ch. 8)

New: non-Cartesian sampling.

Reflect on your 1D signals and systems background; what major topics from a 1D course are absent above?

- ?? ; ??

[RQ]

This review will emphasize the similarities and *differences* between the 1D and 2D cases. This treatment serves to reinforce signals and systems concepts learned previously, while simultaneously introducing a few new concepts and properties that are unique to 2D (and higher) problems. For further reference, see [1, 2].

Advanced topics that are not essential reading are marked with the “dangerous bend” symbol in the margin. [wiki]



1.1 Signals

Definition of signals/images

Mathematically, a **signal** is a *function* of one or more independent variables. An **image** is a function of two (or perhaps 3) variables.

A **function** is a mapping from one set of values, called the **domain**, to another set called the **range**. When introducing a new function, such as one called g , one generally writes $g : \mathcal{D} \mapsto \mathcal{R}$ and then describes the domain \mathcal{D} , the range \mathcal{R} , and the value $g(d) \in \mathcal{R}$ that the function assigns to an arbitrary member d of the domain \mathcal{D} .

Saying that an image is a function of two or three variables is rather broad. To get more specific we make some *classifications*!

Domain dimension

One way to classify signals is by the **dimension of the domain** of the function, *i.e.*, how many arguments the function has.

- A **one-dimensional (1D)** signal is a function of a single variable, *e.g.*, time or elevation above sea level. In this case the domain is a subset of $\mathbb{R} = (-\infty, \infty)$, the set of all real numbers.
- An **M -dimensional** signal is a function of M independent variables. In this case, the domain is a subset of $\mathbb{R}^M = (-\infty, \infty)^M$, the set of all M -tuples of real numbers.

Example. A sequence of “black and white” (really: grayscale) TV pictures $I(x, y, t)$ is a scalar-valued function of three arguments: spatial coordinates x and y , and time index t , so it is a **three-dimensional (3D)** signal.

We focus on **two-dimensional (2D)** signals, *i.e.*, images $g(x, y)$; generally the independent variables (x, y) denote spatial position.

Range dimension

Another way to classify signals is by the **dimension of the range** of the function, *i.e.*, the dimension of the set of values the function can take.

- **scalar** or **single-channel** signals have range dimension = 1
- **multichannel** signals have a vector of range values for each domain value.

Example. A BW TV picture is scalar valued, whereas a color TV picture can be described as a three-channel signal, where the components of the signal represent red, green, and blue (RGB). Here $\vec{g}(x, y) = [g_R(x, y) \ g_G(x, y) \ g_B(x, y)]$. We will consider both **scalar** (grayscale) images and multichannel (*e.g.*, color) images.

Discrete-space vs. continuous-space images

- **Continuous-space images**, *i.e.*, (CS) images

Example. $g(x, y) = e^{-(x^2+y^2)}$, $-\infty < x, y < \infty$.

Defined for all locations/coordinates $(x, y) \in \mathbb{R}^2 = (-\infty, \infty)^2$, or at least in some (usually rectangular) subset of \mathbb{R}^2 , *i.e.*, the domain is this subset. Often we focus on such images for mathematical analysis. The default domain is \mathbb{R}^2 unless stated otherwise.

- **Discrete-space images**

Example. $g[m, n] = (-1)^{m+n}$ (somewhat similar to a checkerboard)

Defined only for integer locations/coordinates $[m, n] \in \mathbb{Z}^2$, or at least in some (usually rectangular) subset of \mathbb{Z}^2 , *i.e.*, the domain is this subset. Note that $\mathbb{Z} \triangleq \{\dots, -2, -1, 0, 1, 2, \dots\}$.

- Often, $g[m, n]$ denotes the value (*i.e.*, **sample**, see Ch. 4) of some continuous-space image $g(x, y)$ at some discrete location (x_m, y_n) , *i.e.*, $g[m, n] = g(x_m, y_n)$, but not always.
- Often $g[m, n]$ is called the **pixel value** at index $[m, n]$ or location (x_m, y_n) , particularly when the discrete locations are spaced equally on a rectangular grid, *e.g.*, when $x_m = m\Delta_x$, and $y_n = n\Delta_y$, where Δ_x and Δ_y are horizontal and vertical pixel spacings.
- The term “pixel value” is slightly risky because we usually think of a pixel as having a finite size (Δ_x by Δ_y) whereas $g[m, n]$ has no area. The term is used because typical displays interpolate using rectangular kernels (see Ch. 8).
- It is *incorrect* to think of $g[m, n]$ as being zero for non-integer values of $[m, n]$.

Some of the common ways in which discrete-space images arise are as follows.

- Sampling a continuous image at discrete positions. (See Ch. 4.)
- Integrating a continuous image over each of a set of pixel regions. (See Ch. 4.)
- Processing another discrete-space image. (See Ch. 5.)

Focus on continuous-space images

As this chapter focuses on continuous-space images; hereafter, unless stated otherwise, all images in this chapter are considered to be continuous-space images.

Image support

The **support** or **support region** of an image $g(x, y)$ is the subset of its domain \mathcal{D} for which the signal value is non-zero: $\text{support}(g) \triangleq \{(x, y) \in \mathcal{D} : g(x, y) \neq 0\}$.

An image g is said to have **finite** or **bounded support** if there is a real number $S < \infty$ such that $\text{support}(g) \subseteq [-S, S]^2$.

Value characteristics

- For a **continuous-valued image**, each value $g(x, y)$ lies in an uncountable set, typically an interval of real numbers, *e.g.*, $[g_{\min}, g_{\max}]$.
- For a **discrete-valued image**, each value $g(x, y)$ lies in a *countable* set, *e.g.*, a finite set as in $\{0, 1, 2, \dots, 255\}$ for an 8-bit image, or a countably infinite set, as in \mathbb{Z} .
Discrete-valued images typically arise from **quantization** of continuous-valued images (*e.g.*, A-to-D conversion), or from counting (*e.g.*, nuclear imaging).
- A **binary** image has pixels that take only two values, typically 0 or 1.
Binary images arise from **thresholding** continuous-valued or discrete-valued images, or forming **halftone** grayscale images.
- For generality, we will consider both **real-valued** and **complex-valued** images. A real-valued image has $g(x, y) \in \mathbb{R}$. A complex-valued image has $g(x, y) \in \mathbb{C}$, where \mathbb{C} denotes the set of all complex numbers of the form $u + iv$, with $u, v \in \mathbb{R}$ and $i = \sqrt{-1}$.
- Some image values have meaningful physical units, such as lumens. But often the image values are just relative “intensities.”

Analog and digital images

- An **analog image** is typically a **continuous-space, continuous-valued** image
- An **digital image** is typically a **discrete-space, discrete-valued** image

Notation

- We will (usually) use parentheses for the arguments of continuous-space images, *e.g.*, $g(x, y)$, and square brackets for the arguments of discrete-space images, *e.g.*, $g[m, n]$. When it is helpful to further distinguish the two, we will add the subscripts a and d , as in $g_a(x, y)$ and $g_d[m, n]$.
- In these notes, as throughout much of engineering, when we write $g(x, y)$, there are two possible interpretations: $g(x, y)$ might mean the value of the image at location (x, y) or it might refer to the entire image. The correct interpretation will usually be clear from context. Another approach is to simply write g or $g(\cdot, \cdot)$ when referring to the entire image.

Deterministic vs random images

Whether a given set of images is considered to be deterministic or random is a matter of the philosophy of whatever mathematical or statistical models that we assume. We will consider both deterministic and stochastic models for images.

Energy and power images


The **energy** of an image $g(x, y)$ with domain \mathbb{R}^2 is defined as

$$E \triangleq \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} |g(x, y)|^2 dx dy. \quad (1.1)$$

The **average power** of an image $g(x, y)$ with domain \mathbb{R}^2 is defined as

$$P \triangleq \lim_{T \rightarrow \infty} \frac{1}{(2T)^2} \int_{-T}^T \int_{-T}^T |g(x, y)|^2 dx dy \quad (1.2)$$

$$\stackrel{?}{=} \lim_{R \rightarrow \infty} \frac{1}{\pi R^2} \iint_{\{(x, y) : x^2 + y^2 < R\}} |g(x, y)|^2 dx dy. \quad (1.3)$$

Challenge. Use circles of radius T and $\sqrt{2}T$ that inscribe and surround the square $[-T, T] \times [-T, T]$ to show that the above two definitions of power are equivalent, possibly with additional assumptions on $g(x, y)$, or show that they differ. 

- If E is finite ($E < \infty$), then $g(x, y)$ is called an **energy image**, or is said to be **square integrable**, and $P = 0$.
- If E is infinite, then P can be either finite or infinite. If P is finite and nonzero, then $g(x, y)$ is called a **power image**.
- Some images are neither energy images nor power images, such as $g(x, y) = x^2$, for which $E = \infty$ and $P = \infty$. Such signals are generally of little practical engineering importance.
- When an image has a finite domain, the integrals defining energy and power are taken only over that domain, and instead of taking a limit in the power definition, the integral is simply divided by the area of the domain. In any event, power is energy per unit area.
- Using notation from functional analysis, $\mathcal{L}_2(\mathcal{S})$ denotes the set of all real-valued or complex-valued functions with domain \mathcal{S} and finite energy. For example, we often consider $\mathcal{L}_2(\mathbb{R}^2)$, the collection of all finite energy images over the Cartesian plane.
- For complex-valued images, $|g(x, y)|^2 = g(x, y)g^*(x, y)$, where $g^*(x, y)$ denotes the complex conjugate of $g(x, y)$, where $(u + v)^* = u - v$ for $u, v \in \mathbb{R}$.

Periodic images

A 1D signal $g(t)$ is called **periodic** with **period** $T > 0$ iff $g(t) = g(t + T)$ for all $t \in \mathbb{R}$. How to generalize this concept to 2D?

It would be tempting (but incorrect) to say that a 2D signal (image) $g(x, y)$ is periodic if $g(x + T_x, y + T_y) = g(x, y)$, $\forall x, y \in \mathbb{R}$ for some $T_x, T_y > 0$. However, “aperiodic” signals such as $g(x, y) = e^{-|x-y|}$ satisfy this condition (for any $T_x = T_y > 0$). So we need a better condition.

- An image $g(x, y)$ is called **periodic** with **period** $(T_x, T_y) > 0$ iff

$$g(x + T_x, y) = g(x, y + T_y) = g(x, y), \quad \forall x, y \in \mathbb{R}. \quad (1.4)$$

- Otherwise $g(x, y)$ is called **aperiodic**.
- Ordinarily, periodic images are presumed to have domain equal to \mathbb{R}^2 , *i.e.*, infinite domain. One could also consider an image with finite domain to be periodic, if the above periodicity property (1.4) holds over the image domain.
- If $g(x, y)$ is periodic with period (T_x, T_y) , then

$$g(x, y) = g(x + T_x, y) = g((x + T_x) + T_x, y) = g(x + 2T_x, y) = \cdots = g(x + mT_x, y + nT_y), \quad \forall m, n \in \mathbb{Z}.$$

- The following fact is readily verified.

An image that is periodic with period (T_x, T_y) is also periodic with period (mT_x, nT_y) , for any $m, n \in \mathbb{N}$.

Example. The following 2D sine wave image is periodic with period $(T_x, T_y) = (k/\nu_x, l/\nu_y)$ for any $k, l \in \mathbb{N}$:

$$g(x, y) = \sin(2\pi[\nu_x x + \nu_y y]). \quad (1.5)$$

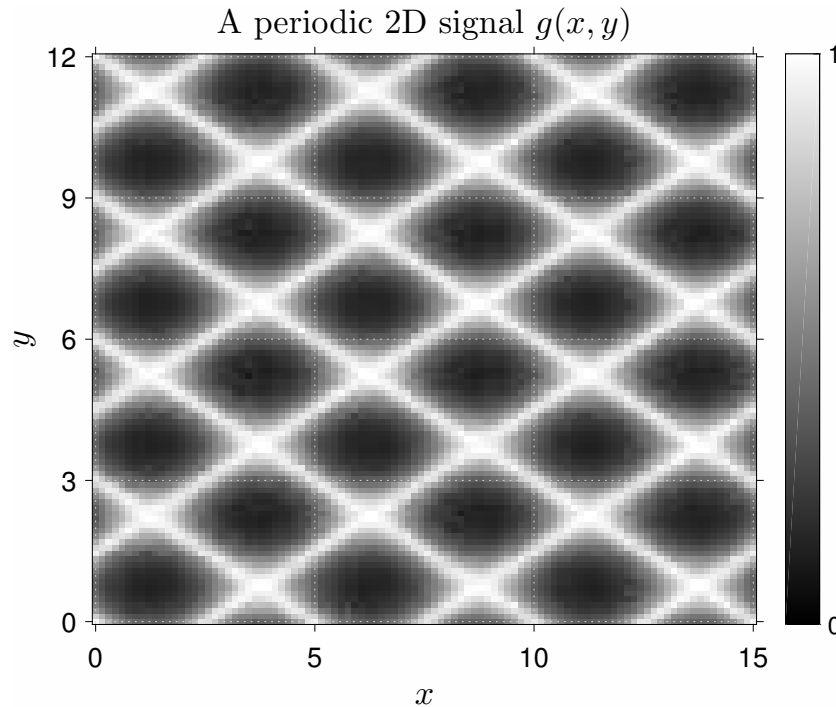
Examining a **line profile** through this image at angle θ along the line described parametrically by $(x(t), y(t)) = (t \cos \theta, t \sin \theta)$, with t ranging from $-\infty$ to ∞ , leads to the 1D signal

$$g_\theta(t) = \sin(2\pi[\nu_x \cos \theta + \nu_y \sin \theta]t)$$

that is periodic with period $1/(\nu_x \cos \theta + \nu_y \sin \theta)$. Profiles in different directions can yield 1D signals with different periods!

Exercise. Give an example of a (simple) periodic 2D function $g(x, y)$ for which the 1D profiles $g_\theta(t) \triangleq g(t \cos \theta, t \sin \theta)$ are periodic for some, but not all, values of θ . In other words, 1D profiles through 2D periodic signals are not always periodic.

Example. The signal $g(x, y) = e^{-|\sin(2\pi x/5) + \sin(2\pi y/3)|}$ is periodic with period $(5, 3)$. The following figure shows (part of) this signal using a (hopefully self explanatory) **gray scale** display.



Fact. Using (1.2), periodic signals are **power signals** with

$$P = \frac{1}{T_x T_y} \int_{\langle T_x \rangle} \int_{\langle T_y \rangle} |g(x, y)|^2 dx dy, \quad (1.6)$$

where $\int_{\langle T \rangle}$ denotes an integral over any interval of length T .

Exercise. Determine the period of the signal $g(x, y) = \sin^2(2\pi x/3) e^{-\text{mod}(y, 5)}$. ??

[RQ]

Coordinate systems?**(skim)**

The definition above seems “constrained” arbitrarily by the choice of the (x, y) coordinate system. If we have an image that is periodic (per the above definition) and we rotate it by $\pi/6$, it will still look “periodic” to our eyes, but will not satisfy the above definition of periodicity.

Here is a more flexible definition [3, p. 51]. An image is “ $\begin{bmatrix} T_{11} & T_{21} \\ T_{12} & T_{22} \end{bmatrix}$ periodic” iff $\exists T_{11}, T_{12}, T_{21}, T_{22}$ such that

$$g(x, y) = g(x + T_{11}, y + T_{12}) = g(x + T_{21}, y + T_{22}), \quad \forall x, y \quad (1.7)$$

and where the translation vectors (T_{11}, T_{12}) and (T_{21}, T_{22}) are linearly independent, *i.e.*, nonzero determinant:

$$|T_{11}T_{22} - T_{12}T_{21}| \neq 0.$$

Such images have a repeating pattern based on the content within the parallelogram having corners at

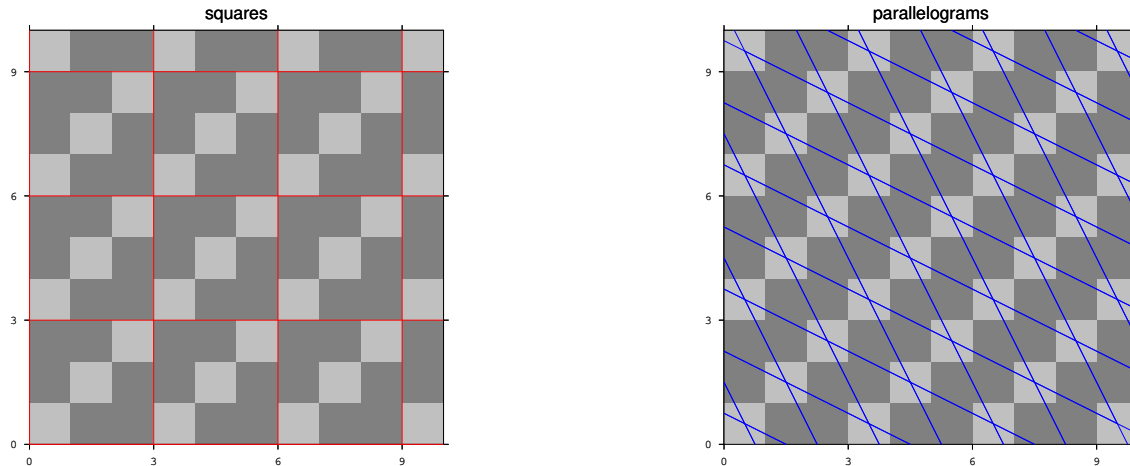
$$(0, 0), (T_{11}, T_{12}), (T_{21}, T_{22}), (T_{11} + T_{21}, T_{12} + T_{22}).$$

Clearly, an image that is (T_x, T_y) periodic is $\begin{bmatrix} T_x & 0 \\ 0 & T_y \end{bmatrix}$ periodic.

Challenge. If an image is $\begin{bmatrix} T_{11} & T_{21} \\ T_{12} & T_{22} \end{bmatrix}$ periodic, when does there exist T'_x and T'_y such that the image is (T'_x, T'_y) periodic? One uninteresting case is when $T_{21} = T_{12} = 0$. Find more general conditions.

Example. The following figure shows an image that is clearly $(3, 3)$ periodic as illustrated by the red squares on the left.

This same image is also $\begin{bmatrix} -2 & -1 \\ 1 & 2 \end{bmatrix}$ -periodic, as illustrated by the blue parallelograms on the right.



The area of each blue parallelogram is the determinant $|(-2)(2) - 1(-1)| = 3$, which clearly is much less than the area of each red square which is 9. Despite this possible “efficiency,” we will not use the more general definition (1.7) because the 2D Fourier series is easier using (1.4). We mention (1.7) primarily as an example of the additional richness that arises beyond 1D. Also, some 2D sampling theorems use ideas related to (1.7).

Fundamental period?*(skim)*

In 1D, at this point we usually define the **fundamental period** to be the “smallest value” of the period satisfying (1.4).

How do we define “smallest value” in 2D?

Any definition of “fundamental period” should be useful, should always exist, and should be unique.

As we will see, there seems not to be a unique way to define a smallest period in 2D.

- Suppose an image $g(x, y)$ is periodic with some period $(T_x, T_y) > 0$, not necessarily the fundamental period. Define $T_{X,0}$ to be the smallest value of T for which (T, T_y) is a period of $g(x, y)$. Define $T_{Y,0}$ to be the smallest value of T for which (T_x, T) is a period of $g(x, y)$. Then it would be natural to define $(T_{X,0}, T_{Y,0})$ as the **fundamental period** of $g(x, y)$. But such a definition would only be sensible if it were independent of whether we minimized over x or y first.

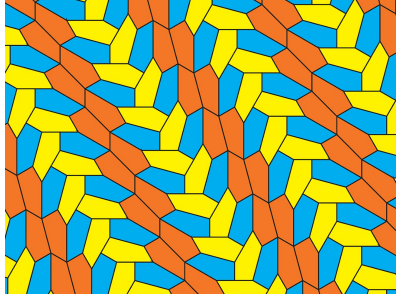
Challenge: Is the preceding definition invariant to ordering of minimization?

(Questions noted as “challenges” are usually ones for which I do not know the answer.)

- Another possible approach would be to define the **fundamental period** to be the smallest period of any periodic 1D signal produced by a profile through the image. For example, for the sinusoidal signal in (1.5) the profile with smallest period has angle $\theta = \tan^{-1}(\nu_y/\nu_x)$ and period $1/\sqrt{\nu_x^2 + \nu_y^2}$. However, it is unclear how this definition that gives a scalar quantity, would correspond to our original definition of **period** that involves a tuple.
 - Yet another approach would be to consider any **undominated period** (T_x, T_y) to be **fundamental**, where *undominated* means there exists no other period (T'_x, T'_y) such that $T'_x \leq T_x$ and $T'_y \leq T_y$. With this definition, there may be a set of fundamental periods, rather than just one.
 - For the more general definition of period in (1.7), a natural definition of the “smallest” period would correspond to the smallest parallelogram area, *i.e.*, the set of $T_{11}, T_{12}, T_{21}, T_{22}$ values that minimize the determinant $|T_{11}T_{22} - T_{12}T_{21}|$. [\[wiki\]](#)
- Challenge:** Is there a unique minimizer of that area subject to the constraints in (1.7)?

- The study of crystal lattices (which are periodic structures) uses related quantities called **primitive cells** [\[wiki\]](#) [4].

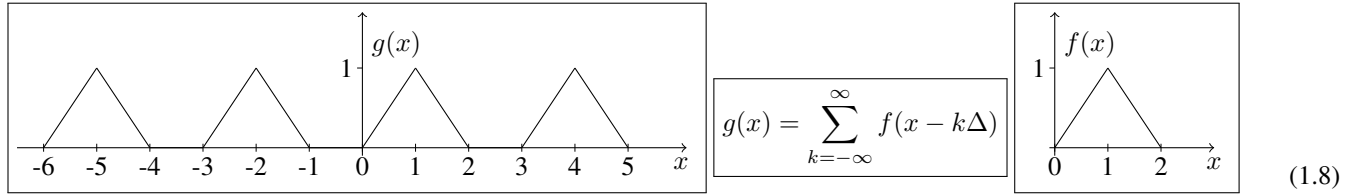
- Consider the following figure that is evidently periodic; defining its fundamental period seems nontrivial. siam.org



There seems not to be a canonical definition of fundamental period for 2D (or higher) signals. Fortunately, defining the fundamental period is not terribly important, because the main place we use the concept of period is with the Fourier series (FS), and for FS it is not essential to use the fundamental period—any period will suffice. (See Ch. 2.)

Periodic functions and modulo

Suppose $f(x)$ is a 1D signal with support $[a, b]$ and we define a periodic signal using superimposed replicates as follows:



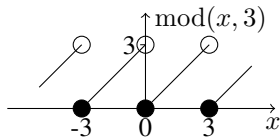
If $\Delta \geq b - a$ so there is *no overlap* of the replicates, then an alternative equivalent expression that is often more convenient for numerical implementation is:

$$g(x) = f(\text{mod}(x - a, \Delta) + a), \quad (1.9)$$

where the **modulo** function is defined (for $y \neq 0$) by

$$\text{mod}(x, y) \triangleq x - \lfloor x/y \rfloor y,$$

and the **floor** function $\lfloor t \rfloor$ denotes the largest integer that is $\leq t$. Note that $\text{mod}(x, y) \in [0, y)$.



Caution: MATLAB's `rem` (remainder) function differs from `mod` for negative arguments.

Exercise. To understand the importance of “no overlap,” determine and sketch $g(x)$ for (1.8) and (1.9) for the case $\Delta = 1$.

Simple image transformations

Now we describe some simple mathematical operations, *i.e.*, transformations, that transform an image into another image. Some of these operate directly on the value of an image, while others operate directly on the independent space variables x and y , and only indirectly on the image values. In other words, some operate on the **range** and some on the **domain** of the signal.

Amplitude transformations

- Affine amplitude transformation:

$$g(x, y) = a f(x, y) + b.$$

- Nonlinear amplitude transformation (we will define linearity formally later):

$$g(x, y) = \begin{cases} 255, & a f(x, y) + b > g_{\max} \\ a f(x, y) + b, & g_{\min} \leq a f(x, y) + b \leq g_{\max} \\ 0, & a f(x, y) + b < g_{\min} \end{cases}$$

This is often used to map an image to the range $[0, 255]$ for display. Performed with MATLAB's `imagesc` function.

Spatial transformations

- Space shifting or translation (*e.g.*, camera panning):

$$g(x, y) = f(x - x_0, y - y_0)$$

- Mirroring or reflection (space reversal):

$$g(x, y) = f(-x, -y), \text{ or } g(x, y) = f(-x, y), \text{ or } g(x, y) = f(x, -y)$$

MATLAB commands: `transpose`, `fliplr`, `flipud`, `flipdim`

- Space scaling (**zooming**, **magnification**, **minification**, or shrinking):

$$g(x, y) = f(ax, ay)$$

Ordinarily, we take $a > 0$, because if $a < 0$, then mirroring occurs in addition to space scaling. Zooming/magnification corresponds to $a < 1$, and shrinking/minification corresponds to $a > 1$.

- **Rotation** (counterclockwise by ϕ radians):

$$g(x, y) = f\left(\underbrace{x \cos \phi + y \sin \phi}_{x'}, \underbrace{-x \sin \phi + y \cos \phi}_{y'}\right). \quad (1.10)$$

MATLAB command: `imrotate` (in discrete-space)

This operation is perhaps easiest expressed in **polar coordinates**. We write an image $g(x, y)$ in polar coordinates as

$$g_{\circ}(r, \theta) = g(r \cos \theta, r \sin \theta),$$

in other words: $(x, y) = (r \cos \theta, r \sin \theta)$. Usually we drop the subscript and just write $g(r, \theta)$.

In polar coordinates, image rotation is simply a translation in the angular coordinate:

$$g_{\circ}(r, \theta) = f_{\circ}(r, \theta - \phi). \quad (1.11)$$

Exercise. Verify that (1.10) and (1.11) are equivalent.

- General **spatial transformation** (e.g., **morphing** or **warping** an image)

$$g(x, y) = f(T_X(x, y), T_Y(x, y)),$$

where here $T_X : \mathbb{R}^2 \rightarrow \mathbb{R}$ and likewise for T_Y . Used, for example, in video coding to approximately match one frame to the next account for scene motion between frames.

MATLAB commands: `interp2`, `interp`, `griddata`

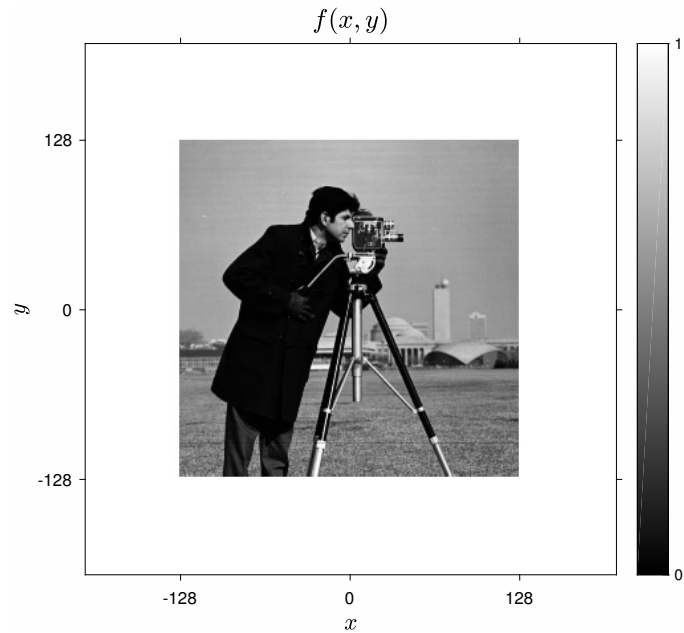
Popular methods for parameterizing general spatial transformations include **thin-plate splines** [wiki] and **tensor-product B-splines**.

There are numerous additional “transformations” that arise in image processing, e.g., **histogram equalization** (Ch. 10), the **gradient magnitude operation**:

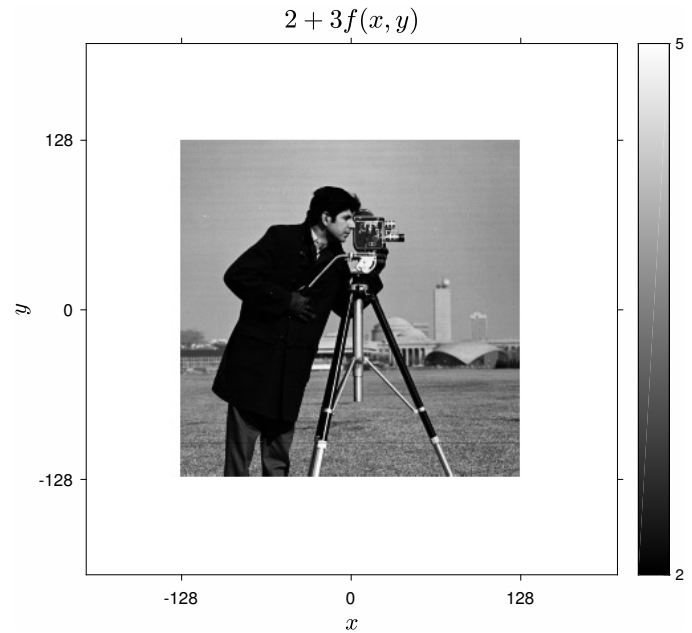
$$g(x, y) = \sqrt{\left(\frac{\partial}{\partial x} f(x, y)\right)^2 + \left(\frac{\partial}{\partial y} f(x, y)\right)^2}, \text{ etc.}$$

Example. These figures illustrate several of the amplitude and spatial transformations.

Original:



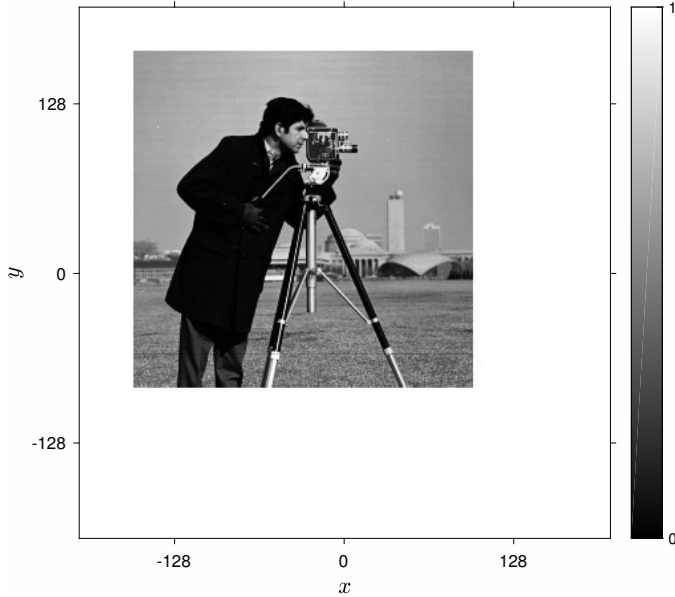
Affine amplitude transform:



The two images look identical because of how the `imagesc` function works. But the `colorbar` differs for the image on the right.

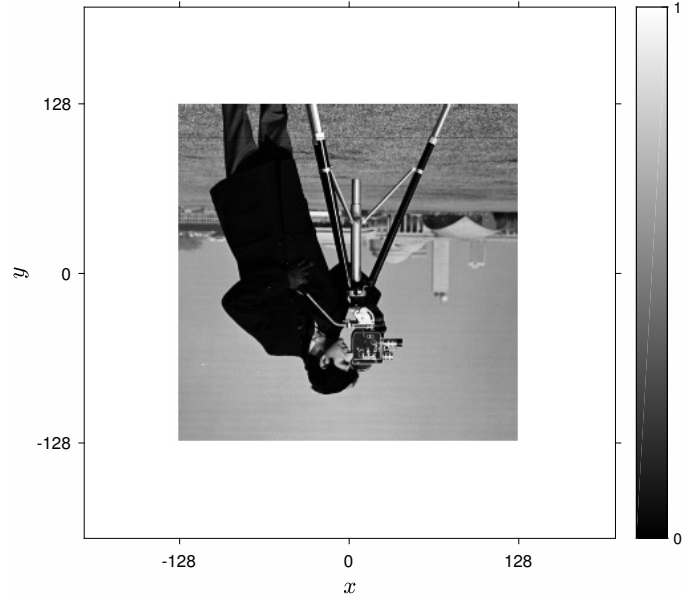
Spatial shift:

$$g(x, y) = f(x + 30, y - 40)$$



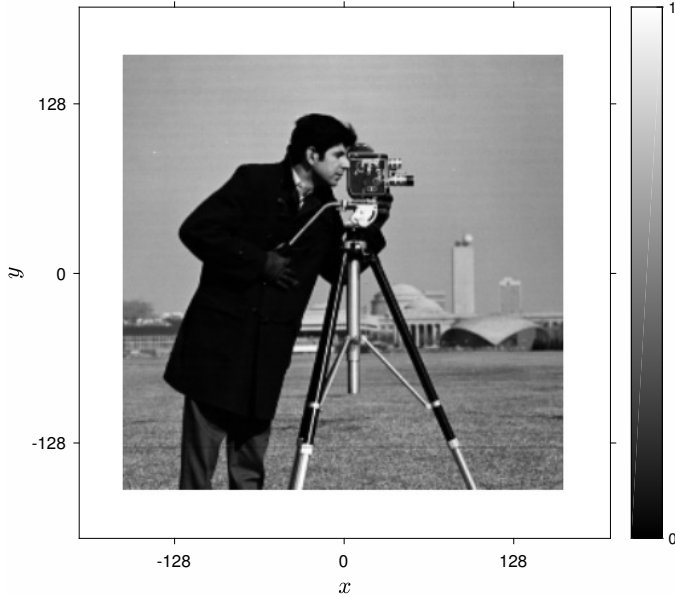
Mirroring or reflecting along y axis:

$$g(x, y) = f(x, -y)$$



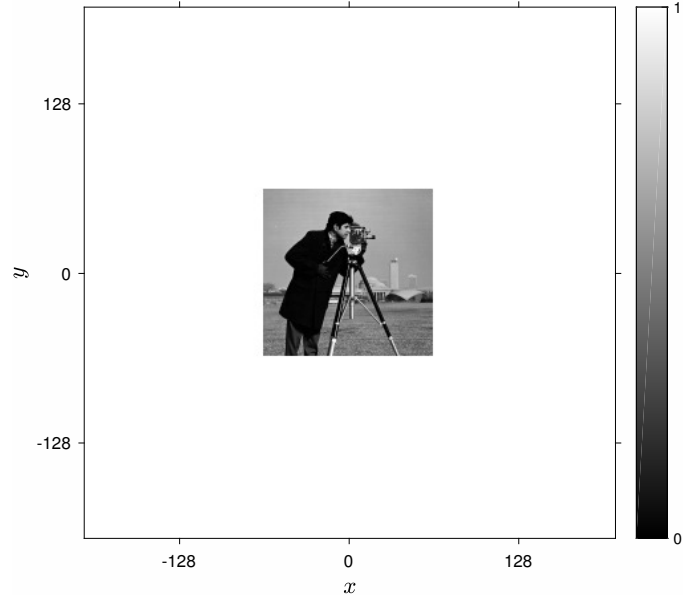
Magnify:

$$g(x, y) = f(x/1.3, y/1.3)$$



Minify:

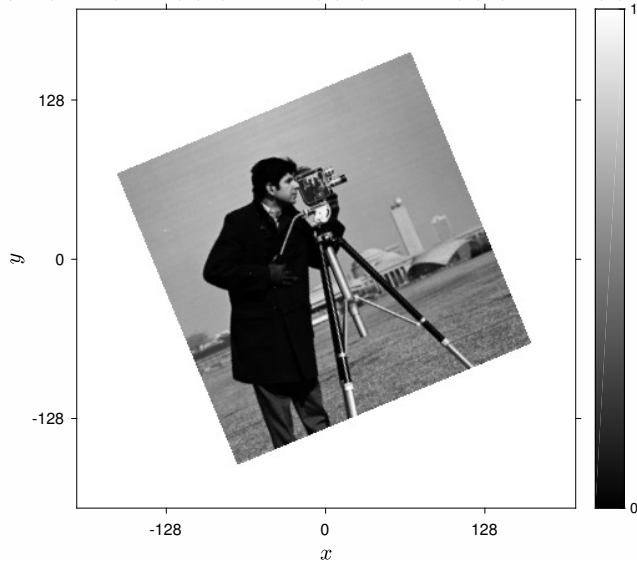
$$g(x, y) = f(2x, 2y)$$



Make sure you understand why the image on the right looks smaller than the original!

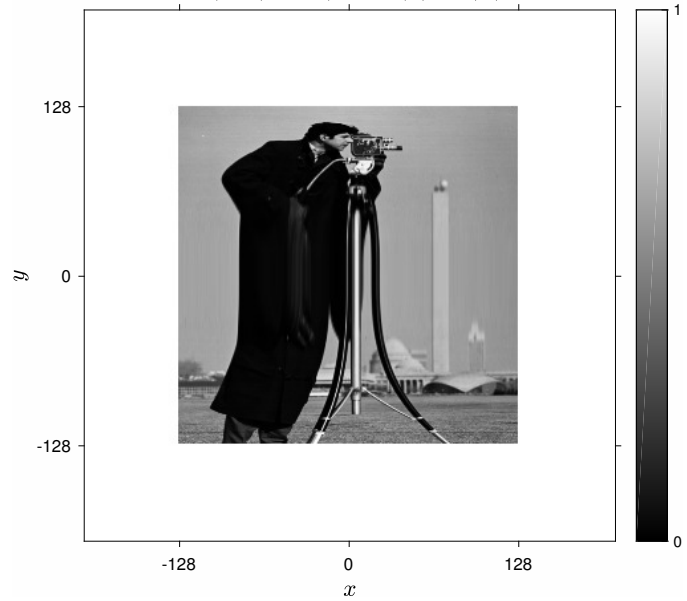
Rotate:

$$g(x, y) = f(x \cos(\pi/8) + y \sin(\pi/8), -x \sin(\pi/8) + y \cos(\pi/8))$$



Warp:

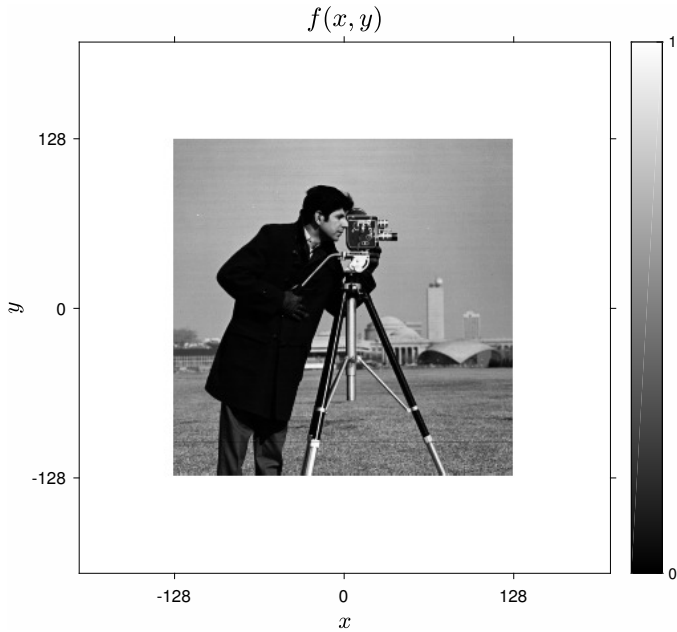
$$g(x, y) = f(x, 128(y/128)^3)$$



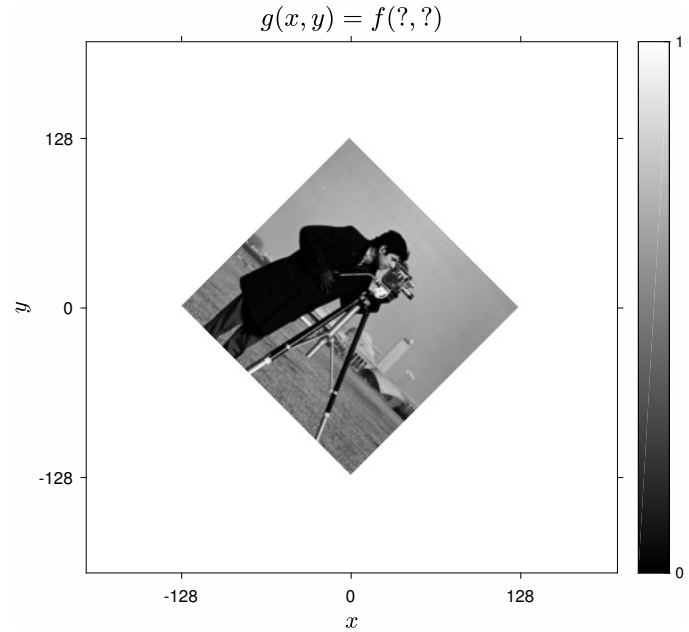
Exercise. Determine the (unique?) transformation of the image $f(x, y)$ that yields the following image $g(x, y)$.

??

Original:



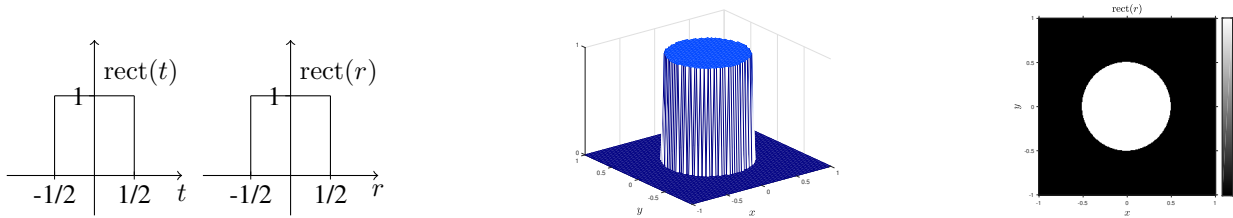
Mystery:



Important 2D signals

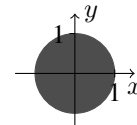
- The **pillbox** image or **disk** image is unity inside a disk of radius $1/2$:

$$g(x, y) \triangleq \text{rect}\left(\sqrt{x^2 + y^2}\right) = \text{rect}(r), \text{ where } \text{rect}(t) \triangleq \begin{cases} 1, & |t| < 1/2 \\ 0, & \text{otherwise.} \end{cases}$$



- A closely-related image is the **circ** function, which is unity over the **unit disk**:

$$g(x, y) = \text{circ}\left(\sqrt{x^2 + y^2}\right) = \text{circ}(r) = \text{rect}\left(\frac{r}{2}\right).$$



(1.12)

- A **box** image or **2D rect** is defined by

$$g(x, y) = \text{rect}_2\left(\frac{x - x_0}{w_x}, \frac{y - y_0}{w_y}\right) \triangleq \text{rect}\left(\frac{x - x_0}{w_x}\right) \text{rect}\left(\frac{y - y_0}{w_y}\right). \quad (1.13)$$

This function is unity within a box of size w_x by w_y centered at (x_0, y_0) , and is zero elsewhere.



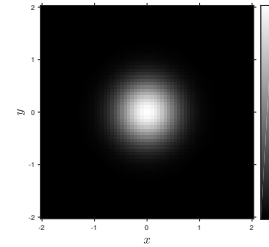
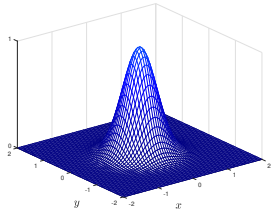
- The 2D **step function** is

$$\text{step}_2(x, y) = \text{step}(x) \text{step}(y), \quad \text{where } \text{step}(x) \triangleq \mathbb{I}_{\{x \geq 0\}} = \begin{cases} 1, & x \geq 0 \\ 0, & \text{otherwise.} \end{cases}$$

Exercise. Sketch the 2D unit step function. ??

- The 2D **gaussian** image is

$$g(x, y) = e^{-\pi r^2} = e^{-\pi x^2} e^{-\pi y^2}. \quad (1.14)$$



- The class of **2D sinusoidal images** is given by

$$g(x, y) = \alpha \cos(2\pi[\nu_x x + \nu_y y] + \phi), \quad \text{where } \nu_x, \nu_y \in \mathbb{R} \text{ and } \alpha \in \mathbb{R}.$$

ν_x and ν_y are called the **spatial frequencies** of the sinusoid. What are their units? ??

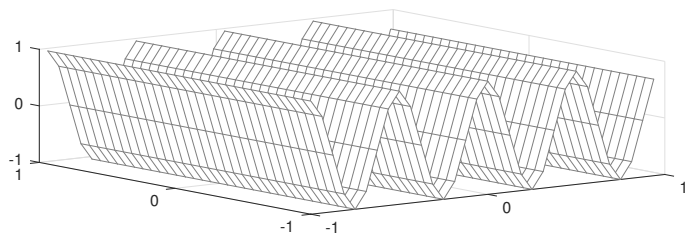
[RQ]

- The class of **2D complex exponential signals** is given by

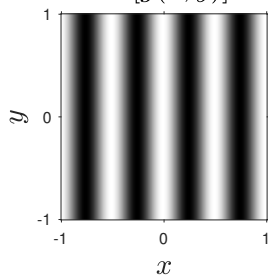
$$g(x, y) = \alpha e^{j2\pi(\nu_x x + \nu_y y)}, \text{ where } \nu_x, \nu_y \in \mathbb{R} \text{ and } \alpha \in \mathbb{C}.$$

These are complex signals, so we visualize them by displaying their real and imaginary parts, which are sinusoidal signals.

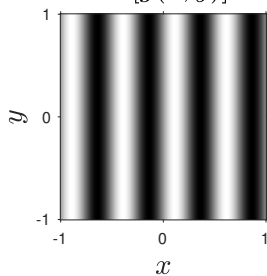
$$(\nu_x, \nu_y) = (2, 0)$$



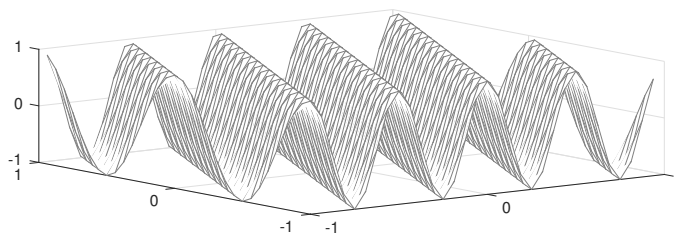
$$\text{Re}[g(x, y)]$$



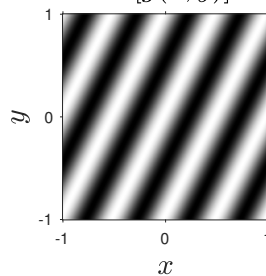
$$\text{Im}[g(x, y)]$$



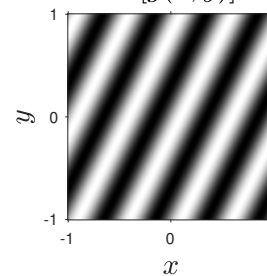
$$(\nu_x, \nu_y) = (2, -1)$$



$$\text{Re}[g(x, y)]$$

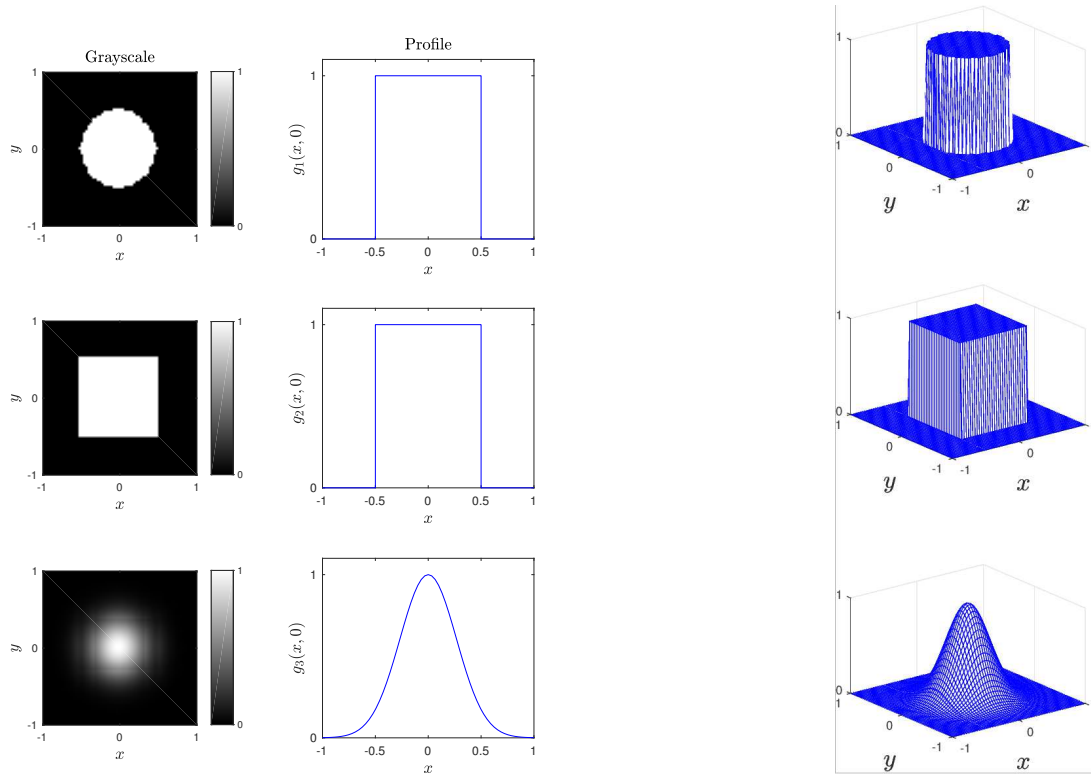


$$\text{Im}[g(x, y)]$$



Displaying images in MATLAB

To efficiently generate (samples of) 2D signals in MATLAB and to display them as images, the `ndgrid` in conjunction with `imagesc` are very useful. Study the following example that shows the 2D images $g(x, y)$ both using a **gray scale** display and as a **mesh plot** and horizontal **profiles** $g(x, 0)$.




```

% fig_signal2a.m illustrate some 2d signals
% uses some plot commands from Michigan Image Recon Toolbox (MIRT)
% xlabel, ylabel, title, cbar, ztick

nx = 2^6; x = [-nx/2:nx/2-1]/nx * 2.1;
x = sort([x -0.5-eps -0.5+eps 0.5-eps 0.5+eps]); % plot trick for rect edges
ny = nx; y = [-ny/2:ny/2-1]/ny * 2.1;
[xx, yy] = ndgrid(x,y); % useful trick for x,y grid!
rect = @(t) double(abs(t) < 1/2); % anonymous: useful trick for short functions
g1f = @(x,y) rect(sqrt(x.^2 + y.^2)); g1 = g1f(xx,yy);
g2f = @(x,y) rect(x) .* rect(y); g2 = g2f(xx,yy);
g3f = @(x,y) exp(-7 * (x.^2+y.^2)); g3 = g3f(xx,yy);
clf, colormap(gray(256))
subplot(321), i_fun(x, y, g1), titlef 'Grayscale' % titlef('g_1(x,y)')
subplot(323), i_fun(x, y, g2)
subplot(325), i_fun(x, y, g3)

subplot(322), plot(x, g1f(x,0), 'b-'), p_fun, ylabel '$g_1(x,0)$', titlef('Profile')
subplot(324), plot(x, g2f(x,0), 'b-'), p_fun, ylabel '$g_2(x,0)$'
subplot(326), plot(x, g3f(x,0), 'b-'), p_fun, ylabel '$g_3(x,0)$'
%orient tall, print('fig_signal2a', '-depsc')

clf, c = ones(size(g1)); colormap([0 0 1]) % a single mesh color
subplot(321), mesh(xx, yy, g1, c), m_fun
subplot(323), mesh(xx, yy, g2, c), m_fun
subplot(325), mesh(xx, yy, g3, c), m_fun
%orient tall, print('fig_signal2b', '-depsc')

function i_fun(x, y, g, t) % function within script added in matlab 2016b
imagesc(x, y, g', [0 1]), axis xy square, cbar
set(gca, 'xtick', -1:1), set(gca, 'ytick', -1:1)
axis([-1 1 -1 1]), xlabel('$x$'), ylabel('$y$')
end

function p_fun
axis([-1 1 0 1.1]), ytick([0 1]), xlabel('$x$')
end

function m_fun
axis([-1 1 -1 1 0 1]), xlabel('$x$'), ylabel('$y$'), ztick([0 1])
end

```

Simple image classes

Symmetric images

There are several ways to define symmetry properties of 2D images. For example, in examining axial brain images, we expect left-right symmetry. For establishing properties of the 2D FT, however, the most useful symmetry properties are the following.

- $g(x, y)$ has **even symmetry** iff $\forall x, y : g(-x, -y) = g(x, y)$
- $g(x, y)$ has **odd symmetry** iff $\forall x, y : g(-x, -y) = -g(x, y)$
- $g(x, y)$ is **Hermitian symmetric** iff $\forall x, y : g(-x, -y) = \boxed{??}$

[RQ]

We can decompose any image $g(x, y)$ into its even and odd parts, just as in the 1D case:

$$g(x, y) = \text{Ev} \{g(x, y)\} + \text{Od} \{g(x, y)\},$$

where the **even part** and **odd part** of an image are defined as follows:

$$\text{Ev} \{g(x, y)\} \triangleq \frac{1}{2} [g(x, y) + g(-x, -y)], \quad \text{Od} \{g(x, y)\} \triangleq \frac{1}{2} [g(x, y) - g(-x, -y)].$$

An alternate way to describe a symmetry property is to define a simple image operation and then to consider the set of images that are left unchanged for by the operation. For example, an image has even symmetry iff mirroring (*i.e.*, space reversal) produces the same image. That is, an image has even symmetry iff it is *invariant* to mirroring. As another example, an image $f(x, y)$ has odd symmetry iff the transformation $g(x, y) = -f(-x, -y)$ produces the same image.

Circular symmetry _____ (No 1D analog!)

Another symmetry condition is particularly important for imaging problems.

We say an image $g(x, y)$ is **circularly symmetric** or **radially symmetric** iff it is invariant to any rotation, or equivalently, iff

$$g(x, y) = g(s, t) \text{ whenever } \sqrt{x^2 + y^2} = \sqrt{s^2 + t^2}$$

or equivalently, iff

$$g(x, y) = g_R\left(\sqrt{x^2 + y^2}\right), \quad \forall x, y$$

for some 1D function $g_R(\cdot)$. Often we just write $g(r)$ rather than $g_R(r)$, and the reader must remember that if we consider r to be a scalar, then $g(r)$ is a 1D function, but if we consider $r = \sqrt{x^2 + y^2}$, where x and y are both variables, then $g(r)$ is a 2D function.

Example: see **pillbox** and 2D **gaussian** images shown above.

How would we define the “radially symmetric component” of an image? ??

Can we decompose any image into a radially symmetric component plus some other meaningful component?
(We can do a decomposition but as far as I can tell the other component is not particularly interesting.)

n-fold rotational symmetry _____ (*skim*)

An image $g(x, y)$ is said to have **n-fold rotational symmetry** iff it is invariant to a rotation by angle $2\pi/n$, equivalently, iff

$$g(x, y) = g\left(\cos\left(\frac{2\pi}{n}\right)x - \sin\left(\frac{2\pi}{n}\right)y, \sin\left(\frac{2\pi}{n}\right)x + \cos\left(\frac{2\pi}{n}\right)y\right), \quad \forall x, y.$$

This property is used far less frequently than circular symmetry.

Example: $\text{rect}_2(x, y)$ has four-fold rotational symmetry.

Fact. All circularly symmetric images have n -fold rotational symmetry for every $n \in \mathbb{N} \triangleq \{1, 2, \dots\}$.

Exercise. Show that the converse of this fact is not true.

Separable images _____ (No 1D analog!)

For convolution, Fourier transforms (Ch. 2), and other analyses, we often simplify results by exploiting separability in one of the two following forms.

- An image $g(x, y)$ is called **separable in Cartesian coordinates** (or just **separable**) iff

$$g(x, y) = g_x(x)g_y(y), \text{ for some 1D functions } g_x(\cdot) \text{ and } g_y(\cdot).$$

- An image $g(x, y)$ is called **separable in polar coordinates** (or **polar separable**) iff

$$g(x, y) = g_R(\sqrt{x^2 + y^2})g_\Theta(\tan^{-1}(y/x)),$$

or equivalently: iff

$$g_\circ(r, \theta) = g_R(r)g_\Theta(\theta),$$

for some 1D functions $g_R(\cdot)$ and $g_\Theta(\cdot)$, where $r = \sqrt{x^2 + y^2}$, $\theta = \tan^{-1}(y/x)$.

Occasionally we might also have use for functions that are **additively separable**:

$$g(x, y) = g_x(x) + g_y(y).$$

Exercise. Find a (non-zero) example image $g(x, y)$ that is both separable and additively separable. ??

Exercise. What images are *both* (Cartesian) separable and radially symmetric? (Be as general as possible.)

Exercise. What images are *both* additively separable and radially symmetric?

Does circular symmetry imply polar separability, or vice versa? ??

[RQ]

What kind of symmetry does a 2D complex exponential signal have? ??

[RQ]

Which of the “important 2D signals” are circularly symmetric? ??

[RQ]

Which of them are separable? ??

[RQ]

Is the 2D function $\text{rect}(r/3 - 1)$ circularly symmetric? ??

[RQ]

2D Dirac impulse

The 1D Dirac impulse $\delta(t)$ has a central role in the analysis of 1D systems.

The **2D Dirac impulse** $\delta_2(x, y)$ has an equally important role for analysis of 2D systems.

Although the Dirac impulse is sometimes called the “**Dirac delta function**” in the engineering literature, it is not really a **function**. In the branch of mathematics that deals rigorously with such entities, they are called **distributions**. [wiki]

We “define” a 2D Dirac impulse by specifying its *properties*, rather than by giving an explicit formula for $\delta_2(x, y)$ itself. Basically, it is defined by what it does as an integrand, although there are also a few other useful properties.

Major properties

- **Sifting property** $\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y) \delta_2(x - x_0, y - y_0) dx dy = g(x_0, y_0)$ if $g(x, y)$ is continuous at (x_0, y_0) .

This is by far the most important property, and it is the key to simplifying the analysis of linear systems.

- More generally, the above property holds if the integration limits are replaced by an open set that contains the point (x_0, y_0) , e.g., the set $(x_0 - \varepsilon, x_0 + \varepsilon) \times (y_0 - \varepsilon, y_0 + \varepsilon)$ for some $\varepsilon > 0$.
- The sifting integral is zero if the integral is over any closed set that does not contain the point (x_0, y_0) .
- The integral is *undefined* if the point (x_0, y_0) is on the boundary of the region of integration.
- **Sampling property** $g(x, y) \delta_2(x - x_0, y - y_0) = g(x_0, y_0) \delta_2(x - x_0, y - y_0)$ if $g(x, y)$ is continuous at (x_0, y_0) .

Minor properties (these all follow from the sifting property!)

- **unit area property** $\iint \delta_2(x, y) dx dy = 1$
(The integral can be over any open set containing the origin.)
- **scaling property** $\delta_2(ax + b, cy + d) = \boxed{??}$
- **symmetry property** $\delta_2(-x, -y) = \delta_2(x, y)$
- **support property** $\delta_2(x - x_0, y - y_0) = 0$ if $x \neq x_0$ or $y \neq y_0$.
- relationship with unit step function: $\delta_2(x, y) = \frac{\partial^2}{\partial x \partial y} \text{step}(x) \text{step}(y)$
- Separability: $\delta_2(x, y) = \delta(x) \delta(y)$ (not essential)

What are the units of $\delta_2(x, y)$? ??

[RQ]

What are the units of $\delta_2(x/\Delta_x, y/\Delta_y)$? ??


[RQ]

How do we unify these answers? ??

No ordinary function can have all the above properties. For example, any ordinary function that is zero everywhere except at $(x, y) = (0, 0)$ will integrate to zero, rather than to unity.

However, there are many functions *approximate* the properties of a Dirac impulse, for example, many tall and narrow pulse functions that integrate to unity. The approximation improves as the pulse becomes taller and narrower. As an example, for intuition one can think of the Dirac impulse as the limit of a unit-volume 2D function, *e.g.*, a 2D Gaussian:

$$\delta_2(x, y) \text{ “} = \text{” } \lim_{\alpha \rightarrow \infty} \delta_2(x, y; \alpha), \quad \delta_2(x, y; \alpha) = \alpha^2 e^{-\pi\alpha^2(x^2+y^2)}.$$

The preceding limit is not rigorous. However, what is really meant by such expressions is that for points (x, y) at which $g(x, y)$ is continuous, in the limit, integrating $g(x', y')$ times $\delta_2(x', y'; \alpha)$ over an interval containing (x, y) has the same effect as integrating $g(x', y')$ times $\delta_2(x', y')$: 

$$\lim_{\alpha \rightarrow \infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x', y') \delta_2(x' - x, y' - y; \alpha) dx' dy' = g(x, y). \quad (1.15)$$

In other words, $\delta_2(x, y)$ is just an idealization of a tall narrow pulse like $\delta_2(x, y; \alpha)$. The type of limit shown above can be studied rigorously, and holds for many pulse-like functions $\delta_2(x, y; \alpha)$, not just for Gaussians.

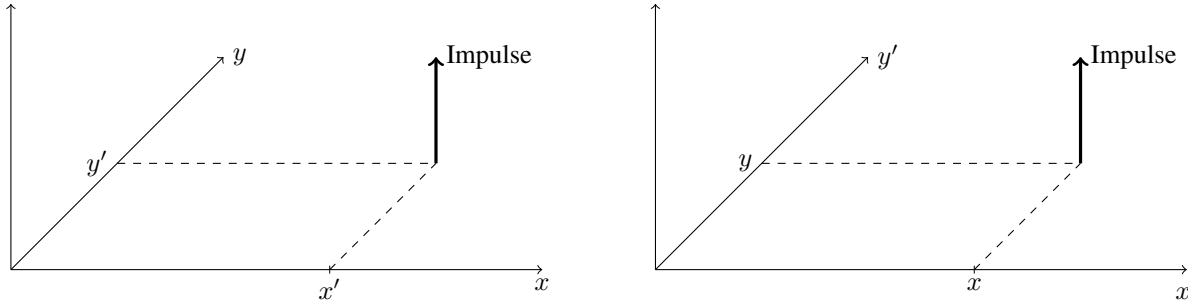
The Gaussian choice is useful when (many) derivatives of $\delta_2(x, y)$ are needed. More frequently, no derivatives are needed, and the rect choice $\delta_2(x, y; \alpha) = \alpha^2 \text{rect}_2(\alpha x, \alpha y)$ can be more convenient.

Sifting property

The sifting property written above can also be viewed as inducing a decomposition of $g(x, y)$ into (weighted) elementary functions, namely off-center Dirac impulses. Define

$$\delta_2(x, y; x', y') \triangleq \delta_2(x - x', y - y'),$$

which one can visualize as follows.



The usual 2D Dirac impulse located at the origin is simply

$$\delta_2(x, y; 0, 0) = \delta_2(x, y),$$

whereas $\delta_2(x, y; x', y')$ is a Dirac impulse located at (x', y') .

We can then rewrite the sifting property as follows (the first equation corresponds to the left diagram above):

$$g(x', y') = \iint g(x, y) \delta_2(x, y; x', y') dx dy. \quad \text{(Picture)}$$

$$g(x, y) = \iint g(x', y') \delta_2(x, y; x', y') dx' dy' \quad \text{“superposition” version for deriving convolution later}$$

Comb “functions”

The 1D **comb** “function” or **shah** “function” or **impulse train** is defined as follows:

$$\text{comb}(x) \triangleq \sum_{n=-\infty}^{\infty} \delta(x - n). \quad \begin{array}{c} \uparrow \text{comb}(x) \\ \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \dots \\ -3 \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3 \quad x \end{array} \quad (1.16)$$

We often need non-integer spacing; note the scale factor:

$$\text{comb}\left(\frac{x}{\Delta_x}\right) = \sum_{n=-\infty}^{\infty} \delta\left(\frac{x}{\Delta_x} - n\right) = \Delta_x \sum_{n=-\infty}^{\infty} \delta(x - n\Delta_x). \quad (1.17)$$

The 2D comb “function” is defined as a separable product that makes a “**bed of nails**.”

$$\begin{aligned} \text{comb}_2\left(\frac{x}{\Delta_x}, \frac{y}{\Delta_y}\right) &\triangleq \text{comb}\left(\frac{x}{\Delta_x}\right) \text{comb}\left(\frac{y}{\Delta_y}\right) \\ &= \Delta_x \Delta_y \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \delta_2(x - m\Delta_x, y - n\Delta_y). \end{aligned} \quad \begin{array}{c} \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \dots \\ \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \dots \\ \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \dots \\ \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \dots \\ \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \dots \\ -1 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad x \end{array} \quad (1.18)$$

These “functions” are useful for *analyzing* idealized sampling. (See Ch. 4.)

Exercise. Sketch a “proof” of the (2D) **sifting property** for $\text{comb}_2(x/\Delta_x, y/\Delta_y)$:

$$\iint f(x, y) \text{comb}\left(\frac{x}{\Delta_x}\right) \text{comb}\left(\frac{y}{\Delta_y}\right) dx dy = \Delta_x \Delta_y \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(m\Delta_x, n\Delta_y).$$

Dirac impulses with nonlinear arguments

What does $\delta(\sin(x))$ or $\delta_2(\sin(x), y^2)$ mean? Study the descriptions on the next pages. (This is optional reading.)



1D Dirac impulses with nonlinear arguments*(skim)*

For the 1D Dirac impulse, one can argue by simple change-of-variables that

$$\delta(f(x)) = \sum_{\{x_n : f(x_n)=0\}} \frac{\delta(x - x_n)}{|\dot{f}(x_n)|}, \quad (1.19)$$

where the x_n values are the roots of the equation $f(x) = 0$ and $\dot{f} \triangleq \frac{d}{dx}f$, provided $\dot{f}(x_n) \neq 0$ for each root.

As can be seen from the derivation below, this formula stems from the fact that at each root x_n , the derivative becomes a local scale factor. This formula is “valid” if the set of roots is finite (or perhaps even countable), if f is continuously differentiable and one-to-one in the neighborhood of each root, and if \dot{f} is nonzero at each root.

Derivation.

Let $\{B_n\}$ be a set of disjoint open intervals such that $x_n \in B_n$ but $x_m \notin B_n$ for $n \neq m$. By the support property of the Dirac impulse:

$$\int_{-\infty}^{\infty} \delta(f(x)) q(x) dx = \sum_n \int_{B_n} \delta(f(x)) q(x) dx.$$

Making the change of variables $x' = f(x)$ we have

$$\begin{aligned} \int_{B_n} \delta(f(x)) q(x) dx &= \int_{f(B_n)} \delta(x') q(f^{-1}(x')) \frac{dx'}{|\dot{f}(f^{-1}(x'))|} = \frac{q(f^{-1}(0))}{|\dot{f}(f^{-1}(0))|} \\ &= \frac{q(x_n)}{|\dot{f}(x_n)|} = \int_{-\infty}^{\infty} \frac{\delta(x - x_n)}{|\dot{f}(x_n)|} q(x) dx, \end{aligned}$$

where we have applied the usual sifting property twice and we assume $q(x)$ is continuous at each x_n .

Example. The scaling property $\delta(ax) = \frac{1}{|a|} \delta(x)$ for $a \neq 0$ is a simple special case of the general result above.

Example. To understand $\delta(x^2 - 9)$, consider what happens to $\frac{1}{\alpha} \text{rect}\left(\frac{x^2-9}{\alpha}\right)$ as $\alpha \rightarrow 0$.

To examine this formally, let $F(x) = \int_{-\infty}^x f(t) dt$, so that

$$\begin{aligned}
 \int f(x) \delta(x^2 - 9) dx &= \lim_{\alpha \rightarrow 0} \int f(x) \frac{1}{\alpha} \text{rect}\left(\frac{x^2 - 9}{\alpha}\right) dx \\
 &= \frac{1}{\alpha} \int_{\sqrt{9-\alpha/2}}^{\sqrt{9+\alpha/2}} f(x) dx + \frac{1}{\alpha} \int_{-\sqrt{9+\alpha/2}}^{-\sqrt{9-\alpha/2}} f(x) dx \\
 &= \frac{1}{\alpha} \left[F(\sqrt{9+\alpha/2}) - F(\sqrt{9-\alpha/2}) \right] + \frac{1}{\alpha} \left[F(-\sqrt{9-\alpha/2}) - F(-\sqrt{9+\alpha/2}) \right] \\
 &\rightarrow \dot{F}(3) \left[\frac{1/4}{\sqrt{9+\alpha/2}} - \frac{-1/4}{\sqrt{9-\alpha/2}} \right] + \dot{F}(-3) \left[\frac{1/4}{\sqrt{9-\alpha/2}} - \frac{-1/4}{\sqrt{9+\alpha/2}} \right] \Bigg|_{\alpha=0} \\
 &= \frac{1}{6} f(3) + \frac{1}{6} f(-3) = \int f(x) \left[\frac{1}{6} \delta(x-3) + \frac{1}{6} \delta(x+3) \right] dx
 \end{aligned}$$

using the chain rule for derivatives. Thus we conclude, consistent with (1.19), that

$$\delta(x^2 - 9) = \frac{1}{6} \delta(x - 3) + \frac{1}{6} \delta(x + 3).$$

Another explanation is that the support of $\text{rect}\left(\frac{x^2-9}{\alpha}\right)$ is $-\alpha/2 < x^2 - 9 < \alpha/2$ or equivalently $\sqrt{9-\alpha/2} < |x| < \sqrt{9+\alpha/2}$.

For small α , that support is approximately $3 - \frac{\alpha}{12} < |x| < 3 + \frac{\alpha}{12}$ or equivalently $-\frac{1}{2} < \frac{|x|-3}{\alpha/6} < \frac{1}{2}$.

Thus $\text{rect}\left(\frac{x^2-9}{\alpha}\right) \approx \text{rect}\left(\frac{x-3}{\alpha/6}\right) + \text{rect}\left(\frac{x+3}{\alpha/6}\right)$ or equivalently $\frac{1}{\alpha} \text{rect}\left(\frac{x^2-9}{\alpha}\right) \approx \frac{1}{6} \frac{1}{\alpha} \text{rect}\left(\frac{x-3}{\alpha/6}\right) + \frac{1}{6} \frac{1}{\alpha} \text{rect}\left(\frac{x+3}{\alpha/6}\right)$.

2D Dirac impulses with nonlinear arguments**(skim)**

What about in 2D? What is $\delta_2(f(x, y), g(x, y))$?

Assume that the set of roots that simultaneously solve $f(x, y) = 0$ and $g(x, y) = 0$ is finite, and that f and g are differentiable near their zeros. Let (x_n, y_n) denote the coordinate pairs where both $f(x_n, y_n) = 0$ and $g(x_n, y_n) = 0$. Then

$$\delta_2(f(x, y), g(x, y)) = \sum_n \frac{\delta_2(x - x_n, y - y_n)}{|d_n|},$$

where d_n is the (assumed nonzero) Jacobian determinant of $\begin{bmatrix} f(x, y) \\ g(x, y) \end{bmatrix}$ at (x_n, y_n) , i.e., $d_n = \det \begin{bmatrix} \frac{\partial}{\partial x} f(x, y) & \frac{\partial}{\partial y} f(x, y) \\ \frac{\partial}{\partial x} g(x, y) & \frac{\partial}{\partial y} g(x, y) \end{bmatrix}$.

“Proof.” Let $\{B_n\}$ be a set of small disjoint open disks such that $(x_n, y_n) \in B_n$. Then for any continuous function $q(x, y)$:

$$\begin{aligned} \iint \delta_2(f(x, y), g(x, y)) q(x, y) dx dy &= \sum_n \iint_{B_n} \delta_2(f(x, y), g(x, y)) q(x, y) dx dy \\ &= \sum_n \iint_{T(B_n)} \delta_2(u, v) q(x(u, v), y(u, v)) \frac{1}{|J(u, v)|} du dv = \sum_n \frac{1}{|d_n|} q(x_n, y_n) \\ &= \sum_n \iint \frac{\delta_2(x - x_n, y - y_n)}{|d_n|} q(x, y) dx dy = \iint \left[\sum_n \frac{\delta_2(x - x_n, y - y_n)}{|d_n|} \right] q(x, y) dx dy \end{aligned}$$

where we made the change of variables $u = f(x, y)$, $v = g(x, y)$, and applied multivariate calculus. [\[wiki\]](#)

(skim)

The above proof also works if there is countable set $\{(x_n, y_n)\}$ of simultaneous zeros of f and g , as long as those zeros are “spaced apart” i.e. $\inf_m \sqrt{(x_n - x_m)^2 + (y_n - y_m)^2} > 0$ for all n .

Impulse “ribbons”**(skip)**

In some applications we need 1D Dirac impulses in the context of a 2D problem, which requires different sifting properties than the usual formula. One can always determine the appropriate sifting property by returning to a first-principles expression like (1.15). Typically a 2D integral containing a 1D Dirac impulse “sifts” to a 1D integral.

For example, in tomography, two ways to express a line integral at angle θ and radial distance r_0 are as follows:

$$\begin{aligned} \iint \delta(x \cos \theta + y \sin \theta - r_0) g(x, y) dx dy &= \iint \delta(s - r_0) g(s \cos \theta - l \sin \theta, s \sin \theta + l \cos \theta) ds dl \\ &= \int g(r_0 \cos \theta - l \sin \theta, r_0 \sin \theta + l \cos \theta) dl, \end{aligned}$$

by making the change of variables

$$\begin{bmatrix} s \\ l \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

Curvilinear Dirac impulses**(skip)**

Bracewell [2, p. 130] shows that a 1D curvilinear Dirac impulse of the form $\delta(f(x, y))$ has “strength”

$$S(x, y) = \begin{cases} \frac{1}{|\text{grad } f(x, y)|}, & f(x, y) = 0 \\ 0, & \text{otherwise,} \end{cases}$$

where the slope in the direction normal to the contour where $f(x, y) = 0$ is $|\text{grad } f(x, y)| = \sqrt{(\frac{\partial}{\partial x} f)^2 + (\frac{\partial}{\partial y} f)^2}$. This leads to the following sifting property [2, p. 132]:

$$\iint g(x, y) \delta(f(x, y)) dx dy = \int_C S(s) g(x, y) ds,$$

where C is the curve where $f(x, y) = 0$, and ds is the element of arc length along the curve.

Transformation of variables _____ (skim)

The following result from multivariable calculus [5, 6] was used above. [wiki]

Suppose $T : \mathbb{R}^n \rightarrow \mathbb{R}^n$ satisfies the following:

- T is continuously differentiable.
- T is one-to-one on a set $\mathcal{R} \subseteq \mathbb{R}^n$, where \mathcal{R} has a boundary consisting of finitely many smooth sets and where \mathcal{R} and its boundary are contained in the interior of the domain of T .
- $\det\{T\}$, the Jacobian determinant of T , is nonzero on \mathcal{R} .

Then if f is bounded and continuous on $T(\mathcal{R})$ then

$$\int_{T(\mathcal{R})} f(x_1, \dots, x_n) dx_1 \cdots dx_n = \int_{\mathcal{R}} f(T(x_1, \dots, x_n)) |\det\{T(x_1, \dots, x_n)\}| dx_1 \cdots dx_n .$$

Local image phase _____ (skip)

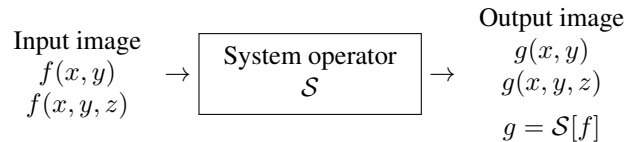
The concept of **local phase** of a signal or image can be important in certain applications. It is defined in 1D via the **Hilbert transform**. See [7] for a discussion of local phase for 1D and 2D signals in the context of image registration.

1.2 Systems in 2D (Imaging systems)

An **imaging system**, or just **system**, operates on a 2D signal called the **input image** and produces a 2D signal called the **output image**.

For continuous-space systems, the input signal $f(x, y)$ is **transformed** by the system into the output signal $g(x, y)$. (We focus on 2D-to-2D systems here, but of course 3D-to-3D and 3D-to-2D systems are also of considerable interest. Extensions to 3D are straightforward.)

This mapping can be depicted as follows.



Roughly speaking, there are two broad types of imaging systems: **image capture systems** and **image processing systems**. A camera and an X-ray scanner are examples of image capture systems. Zoom lenses, optical color filters, photographic film, and analog photocopiers are examples of continuous-space image processing systems. A scanner / digitizer, a contrast enhancer, a denoiser, an image compressor and an edge detector are examples of image processing systems where the output is a discrete-space image.

In an image capture system, $f(x, y)$ is typically the **true image**, or true spatial distribution of some physical quantity of interest, often called the **object**, whereas $g(x, y)$ is the **observed image** or **recorded image**. The goal is often for $g(x, y)$ to be as similar as possible to $f(x, y)$.

(In image restoration problems the goal is to recover $f(x, y)$ from $g(x, y)$.)

In an image processing system, $f(x, y)$ is typically the image produced by some image capture system, which might be noisy or blurry for example, and $g(x, y)$ is some enhanced version of $f(x, y)$. (Typically this is done digitally and thus with a discrete-space system, although optical / analog image enhancement is also possible.)

The **input-output relationship** for an imaging system is described by a **system operator** or **system function** \mathcal{S} and is written

$$g = \mathcal{S}[f], \text{ where } \mathcal{S} : \mathcal{L}_2(\mathbb{R}^2) \rightarrow \mathcal{L}_2(\mathbb{R}^2).$$

In other words, \mathcal{S} maps one function to another function, *i.e.*, an image to another image. The relationship is very often expressed using the following somewhat dangerously imprecise notation:

$$“ g(x, y) = \mathcal{S}[f(x, y)] ”.$$

The problem with that notation is that it suggests that the value of $g(x, y)$ at any point (x, y) depends only on the input image at that same spatial location. This is rarely the case. More precise, but somewhat cumbersome, notation is:

$$g(x, y) = (\mathcal{S}[f])(x, y).$$

For convenience, we will often simply write one of the following:

$$f \xrightarrow{\mathcal{S}} g \text{ or } f(x, y) \xrightarrow{\mathcal{S}} g(x, y) \text{ or } f(x, y) \rightarrow \boxed{\mathcal{S}} \rightarrow g(x, y).$$

Example. Overhead projector...

What is the system function \mathcal{S} ? For most image capture systems, it is a blurring function. Often we can model the image capture system operator as the convolution of the object with the 2D (or 3D) point spread function.

Classification of systems

To begin to analyze 2D imaging systems, we first divide system characteristics into the following two categories.

- Amplitude properties
 - A-1 linearity
 - A-2 stability
 - A-3 invertibility
- Spatial properties
 - S-1 causality
 - S-2 memory
 - S-3 **separability**
 - S-4 shift invariance
 - S-5 **rotation invariance**

Most of these class are also defined for 1D systems, except for **separability** and **rotation invariance**.

We will save linearity (the most important?) for last.

A-2 Stability

- A system is **bounded-input bounded-output (BIBO) stable** iff every bounded input produces a bounded output.

$\forall f$, if $\exists M_f$ s.t. $|f(x, y)| \leq M_f < \infty$, $\forall x, y$, then there must exist an M_g s.t. $|g(x, y)| \leq M_g < \infty$, $\forall x, y$.

Usually M_g will depend on M_f .

- Otherwise the system is called **unstable**, and it is possible that a bounded input signal will make the output “blow up.”

Example: **moving average filter** with $\Delta_x, \Delta_y > 0$

$$\begin{aligned} g(x, y) &= \frac{1}{\Delta_x \Delta_y} \int_{y-\Delta_y/2}^{y+\Delta_y/2} \int_{x-\Delta_x/2}^{x+\Delta_x/2} f(x', y') dx' dy' \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \frac{1}{\Delta_x \Delta_y} \text{rect}_2\left(\frac{x-x'}{\Delta_x}, \frac{y-y'}{\Delta_y}\right) f(x', y') dx' dy'. \end{aligned} \quad (1.20)$$

Suppose $|f(x, y)| \leq M_f < \infty$, $\forall x, y$, so $f(x, y)$ is a bounded input. Then by the **triangle inequality** for integration¹:

$$\begin{aligned} |g(x, y)| &= \left| \frac{1}{\Delta_x \Delta_y} \int_{y-\Delta_y/2}^{y+\Delta_y/2} \int_{x-\Delta_x/2}^{x+\Delta_x/2} f(x', y') dx' dy' \right| \\ &\leq \frac{1}{\Delta_x \Delta_y} \int_{y-\Delta_y/2}^{y+\Delta_y/2} \int_{x-\Delta_x/2}^{x+\Delta_x/2} |f(x', y')| dx' dy' \\ &\leq \frac{1}{\Delta_x \Delta_y} \int_{y-\Delta_y/2}^{y+\Delta_y/2} \int_{x-\Delta_x/2}^{x+\Delta_x/2} M_f dx_0 dy_0 \\ &= M_f, \end{aligned}$$

so $|g(x, y)| \leq M_f$. Thus the output signal is also bounded for a bounded input, so this system is BIBO stable.

We will derive a simple test for BIBO stability shortly that applies to LSI systems (but not more generally).

It is easy to postulate *hypothetical* systems that are unstable, like $g(x, y) = 1/f(x, y)$,

or the ideal integrator: $g(x, y) = \int_{-\infty}^x \int_{-\infty}^y f(x', y') dx' dy'$.

But it is difficult to think of *real* continuous-space systems that are unstable.

The issue of stability is particularly relevant when designing *discrete-space* filters, where unstable examples abound.

¹ The usual triangle inequality is $|a + b| \leq |a| + |b|$. Generalizing to summation and integration yields: $|\sum_n f_n| \leq \sum_n |f_n|$ and $|\int f| \leq \int |f|$.

A-3 Invertibility

- A system \mathcal{S} is called **invertible** iff each (possible) output signal is the response to only *one* input signal.
- Otherwise \mathcal{S} is not invertible.

If a system \mathcal{S} is invertible, then there exists a system \mathcal{S}^{-1} such that

$$f(x, y) \rightarrow \boxed{\mathcal{S}} \rightarrow g(x, y) \rightarrow \boxed{\mathcal{S}^{-1}} \rightarrow f(x, y).$$

Design of \mathcal{S}^{-1} is important in many image processing applications.

Mathematically:

$$\mathcal{S}^{-1}[\mathcal{S}[f]] = f.$$

Invertibility is a particularly important concept in imaging because often we would like to “undo” degrading effects like optical blurring (*e.g.*, in images from the infamous Hubble telescope).

Example: Is the moving average filter (1.20) an invertible system?

[RQ]

??

Hint. If two different input images produce the same output image, then the system is not invertible.

S-1 Causal systems

The concept of causality is very important in 1D systems (when the independent variable is time). A 1D system is causal if the output at any time t_0 depends only on the values of the input signal for times $t \leq t_0$.

In imaging systems, the independent variables are spatial position, not time, so causality is usually unimportant. (Of course in “real time” systems like video, where the independent variables are both space *and* time, one must consider causality with respect to the time variable.)

There are some (mostly older) papers in the image processing literature that address raster-scan based image processing methods, where the image is filtered, for example, by starting in the upper left hand corner and working lexicographically down towards the lower right hand corner. Such processing could be described as “causal” (whereas the moving average filter above would be “noncausal” by such a definition).

S-2 Memory

A system whose output $g(x, y)$ at any point (x, y) only depends on the input image $f(x, y)$ at the same location (x, y) could be called **memoryless**, if we stretch the English usage of “memory” somewhat. For example, a detector that converts amplitude to intensity (with no other effects) via the relation $g(x, y) = |f(x, y)|^2$ is memoryless. Another term that is perhaps more apt in imaging is **pointwise**.

Continuous-space image capture systems are virtually never memoryless, due to the spreading of light when propagating through space. But discrete-space image processing operations can be categorized by those that use pixel neighborhoods versus those that work independently pixel-by-pixel. So the memoryless concept is more relevant to digital image processing than to image capture systems.

S-3 Separable systems

The above definition of causality is a restriction on the how the system function \mathcal{S} behaves in terms of the domain of the input signal. The concept of a **separable system** is also a restriction on the behavior in terms of the domain of the input signal. Roughly speaking, a separable system \mathcal{S} processes the “rows” and “columns” of an image independently, and can be written as

$$\mathcal{S} = \mathcal{S}_h \circ \mathcal{S}_v \text{ or } \mathcal{S} = \mathcal{S}_v \circ \mathcal{S}_h,$$

where \circ denotes function composition, *i.e.*, $\mathcal{S}[f] = \mathcal{S}_h[\mathcal{S}_v[f]]$. Here, \mathcal{S}_h denotes a “horizontal acting” operator where if $g = \mathcal{S}_h[f]$, then for any given y_0 , the output values $g(x, y_0)$ depend only on the input values $f(\cdot, y_0)$, *i.e.*, the input values in the same row. Likewise, \mathcal{S}_v denotes a “vertical acting” operator where if $g = \mathcal{S}_v[f]$, then for any given x_0 , the output values $g(x_0, y)$ depend only on the input values $f(x_0, \cdot)$, *i.e.*, the input values in the same column.

Separable systems essentially require only 1D operations, so they are often used in image processing to minimize computation.

Example. Is the moving average filter (1.20) a separable system? [RQ]

??

Challenge: prove or disprove: if $\mathcal{S} = \mathcal{S}_h \circ \mathcal{S}_v$ then does there always exist systems \mathcal{S}'_h and \mathcal{S}'_v such that $\mathcal{S} = \mathcal{S}'_v \circ \mathcal{S}'_h$, even if \mathcal{S} is nonlinear? If not, what if \mathcal{S} is linear?

Example. It is not obvious, but image rotation is in fact a separable operation, as explained next.



Image rotation is a separable operation

(skim)



Image rotation is used in applications such as digital image stabilization for hand-held cameras.

The obvious approach is to use 2D interpolation (8.2) in Ch. 8, as used in MATLAB's `imrotate` command.

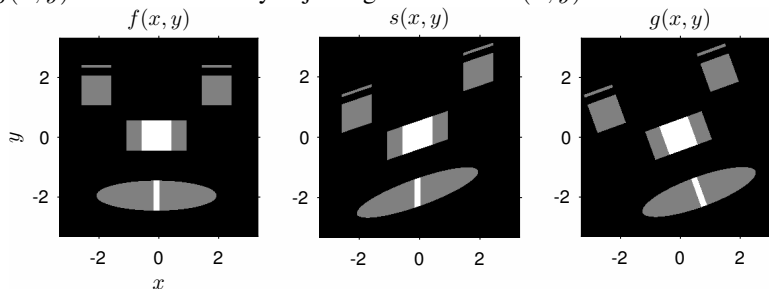
To save computation, one can use a **separable** implementation with the following factorization of a coordinate rotation matrix [8]:

$$\begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\tan \phi & 1/\cos \phi \end{bmatrix} \begin{bmatrix} \cos \phi & \sin \phi \\ 0 & 1 \end{bmatrix}.$$

This factorization allows us to implement image rotation using the following two-pass approach; each pass involves 1D operations:

$$\begin{aligned} f(x, y) &\rightarrow \boxed{\mathcal{S}_v} \rightarrow s(x, y) = f(x, -x \tan \phi + y / \cos \phi) \quad (\text{acts along } y \text{ only}) \\ s(x, y) &\rightarrow \boxed{\mathcal{S}_h} \rightarrow g(x, y) = s(x \cos \phi + y \sin \phi, y) \quad (\text{acts along } x \text{ only}) \\ &= f(x \cos \phi + y \sin \phi, -(x \cos \phi + y \sin \phi) \tan \phi + y / \cos \phi) \\ &= f(x \cos \phi + y \sin \phi, -x \sin \phi + y \cos \phi). \end{aligned}$$

Example. The figure below shows the initial image $f(x, y)$, the intermediate image $s(x, y)$ after the first pass that acts along y only, and the final rotated image $g(x, y)$ that is obtained by adjusting the rows of $s(x, y)$.



There is also a 3-pass factorization where each term has unity determinant (reducing aliasing in a discrete-space implementation):

$$\begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} = \begin{bmatrix} 1 & \tan(\phi/2) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -\sin \phi & 1 \end{bmatrix} \begin{bmatrix} 1 & \tan(\phi/2) \\ 0 & 1 \end{bmatrix}.$$

These factorizations allow one to implement image rotation using a set of 1D interpolation operations, saving computation.

Example. Is the imaging zooming system defined by $g(x, y) = f(x/2, y/2)$ shift invariant? ??

[RQ]

Example: Is the ideal mirror system defined by $g(x, y) = f(-x, -y)$ shift invariant? ??

[RQ]

S-4 Rotation-invariance

Usually the object being imaged has an orientation that is unknown relative to the recording device's coordinate system. Often we would like both the imaging system and the image processing operations applied to the recorded image to be independent of the orientation of the object.

A system \mathcal{S} is called **rotationally invariant** iff

$$f(x, y) \xrightarrow{\mathcal{S}} g(x, y) \quad \text{implies that} \quad f_{\theta}(x, y) \xrightarrow{\mathcal{S}} g_{\theta}(x, y) \quad \text{where} \quad f_{\theta}(x, y) \triangleq f(x \cos \theta + y \sin \theta, -x \sin \theta + y \cos \theta) \quad (1.22)$$

and $g_{\theta}(x, y)$ is defined similarly, for *every* input image $f(x, y)$ and rotation θ .

Fact. A system \mathcal{S} is **rotationally invariant** iff the following two operations produce the same result (*i.e.*, $g_1 = g_2$) for any input image f and any rotation angle θ :

$$\begin{array}{c} f(x, y) \rightarrow \boxed{\mathcal{S}} \xrightarrow{g(x, y)} \boxed{\begin{array}{c} \text{rotate} \\ \theta \end{array}} \rightarrow g_1(x, y) \\ f(x, y) \rightarrow \boxed{\begin{array}{c} \text{rotate} \\ \theta \end{array}} \xrightarrow{f_2(x, y)} \boxed{\mathcal{S}} \rightarrow g_2(x, y) \end{array}$$

Example: Is the moving average system above rotationally invariant? ??

[RQ]

How could we modify the moving-average system to make it rotationally invariant?

??

A-3 Linear systems

We often focus on **linear systems** because linearity is **desirable** for ensuring *representative reproductions*. For example, if one tumor is using glucose at twice the rate of a second tumor, it would be nice for the image intensity (or “gray level”) to be twice as high for the first tumor. Although many imaging system components (such as photographic film) do have some form of nonlinearity, many imaging systems are intentionally **designed** to *minimize nonlinearities* by designing the system to operate over the range that is most linear, avoiding the extreme cases (*e.g.*, over-saturation).

Also, linear systems are *easier to analyze* than nonlinear ones. For *didactic purposes* one must walk before running, *i.e.*, a thorough understanding of the linear case is needed to prepare one for studying any nonlinearities.

In particular, we can often find linear shift-invariant (LSI) approximations to imaging systems, which, through the convolution property of Fourier transforms (see Ch. 2), enables use of frequency-domain concepts as a unifying framework for understanding system properties.

Finally, if a system can be modeled by a system function $\mathcal{S}[f]$ that is only “slightly” nonlinear, then we can linearize it by considering the first-order Taylor expansion about some nominal image f_0 :

$$\mathcal{S}[f] \approx \mathcal{S}[f_0] + \delta_f \mathcal{S}[f_0](f - f_0).$$

(This requires a Gateaux **differential**.) Thus a linear systems analysis is sometimes called a “*first-order*” analysis.

In the context of 2D continuous-space systems, we will say \mathcal{S} is a **linear system** (function) iff it is a linear operator on the vector space of functions $\mathcal{L}_2(\mathbb{R}^2)$ described in the Appendix, *i.e.*, iff

$$\mathcal{S}[\alpha f_1 + \beta f_2] = \alpha \mathcal{S}[f_1] + \beta \mathcal{S}[f_2] \quad \text{superposition property}$$

for all images f_1, f_2 and all constants α, β , considering both real *and* complex constants.

Example: Is the moving average system above linear? ??

[RQ]

Two special cases of the above condition for linearity are particularly important in the context of linear systems.

- If the input to a linear system is scaled by a constant, then the output is scaled by that same constant.

$$\mathcal{S}[\alpha f] = \alpha \mathcal{S}[f] \quad \text{scaling property.}$$

- If the sum of two inputs is presented to the system, then the output will be simply the sum of the outputs that would have resulted from the inputs individually.

$$\mathcal{S}[f_1 + f_2] = \mathcal{S}[f_1] + \mathcal{S}[f_2] \quad \text{additivity property.}$$

We can extend the superposition property to finite sums: by applying proof-by-induction:

$$\mathcal{S} \left[\sum_{k=1}^K \alpha_k f_k \right] = \sum_{k=1}^K \alpha_k \mathcal{S}[f_k].$$

For a system to be called linear, we require the superposition property to hold *even for complex input signals and complex scaling constants*, assuming that the system is defined to permit both real and complex inputs.



Example: Is $g(x, y) = \text{real}\{f(x, y)\}$ a linear system? ??

[RQ]

Thus, both the additivity *and* scaling properties are required for a system to be linear.

Impulse response (point spread function) (PSF) of a linear system

The PSF is perhaps the most important tool in the analysis of imaging systems. It is analogous to the impulse response used in 1D signals and systems.

The superposition property of a linear system enables a very simple method to characterize the system function.

- Decompose the input into some elementary functions.
- Compute the response (output) of the system to each elementary function.
- Determine the total response of the system to the desired (complex) input by simply summing the individual outputs.

A “simple” and powerful function for decomposition is the **Dirac impulse** $\delta_2(x, y)$.

The key property of the Dirac impulse is the **sifting property**:

$$f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x', y') \delta_2(x - x', y - y') dx' dy' = \iint f(x', y') \delta_2(x, y; x', y') dx' dy',$$

where the latter form expresses the image as a decomposition of an image into (weighted) elementary functions, namely off-center Dirac impulses, and where $\delta_2(x, y; x', y') = \delta_2(x - x', y - y')$, was defined earlier.

PSF

Suppose the input to a system \mathcal{S} is an impulse centered at (x', y') , *i.e.*, the input is $\delta_2(x, y; x', y')$, where x' and y' are considered “fixed,” and x, y are the independent variables of the input image. Let $h(x, y; x', y')$ denote the corresponding output signal:

$$\delta_2(x, y; x', y') \xrightarrow{\mathcal{S}} h(x, y; x', y') \quad .$$

The function $h(x, y; x', y')$, the output of the system \mathcal{S} when the input is a Dirac impulse located at (x', y') , is called the **point spread function (PSF)** of the system, or (less frequently in the context of imaging systems) the **impulse response** of the system. In some fields, particular for systems described by partial differential equations (**PDE**)s, the PSF is called the **Green’s function** [wiki].

Example. Out-of-focus overhead projector.

Example: What is the PSF of an ideal mirror?

The input-output relationship is $g(x, y) = f(-x, -y)$, so (using the Dirac symmetry property):

$$\delta_2(x, y; x', y') \xrightarrow{\mathcal{S}} h(x, y; x', y') = \delta_2(-x, -y; x', y') = \delta_2(-x - x', -y - y') = \delta_2(x + x', y + y').$$

Think about this graphically!

Example: What is the PSF of an ideal magnifying lens? For a magnification of 2, the ideal input-output relationship is $g(x, y) = \alpha f(x/2, y/2)$, for some $0 < \alpha < 1/4$ related to the absorption of the lens and the energy spreading. Thus

$$\delta_2(x - x', y - y') \xrightarrow{\mathcal{S}} h(x, y; x', y') = \boxed{??}$$

More generally, the impulse response of the magnifying (or minifying) system $f(x, y) \xrightarrow{\mathcal{S}} g(x, y) = \alpha f(x/a, y/b)$ is $\boxed{??}$

Example: What is the impulse response of the moving average filter (1.20)?

Substitute $f(x, y) = \delta_2(x - x', y - y')$ in (1.20), and we see

$$\begin{aligned} h(x, y; x', y') &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \frac{1}{\Delta_x \Delta_y} \text{rect}_2\left(\frac{x - x''}{\Delta_x}, \frac{y - y''}{\Delta_y}\right) \delta_2(x'' - x', y'' - y') dx'' dy'' \\ &= \frac{1}{\Delta_x \Delta_y} \text{rect}_2\left(\frac{x - x'}{\Delta_x}, \frac{y - y'}{\Delta_y}\right). \end{aligned}$$

The first two examples above also illustrate that linearity is distinct from shift-invariance!

Example. For a camera image with linear motion blur of length L along the x direction, a reasonable PSF model is $h(x, y; x', y') = \delta(y - y') \frac{1}{L} \text{rect}\left(\frac{x - x' - L/2}{L}\right)$. Is this PSF separable? $\boxed{??}$

What if camera is rotated by, say, 30 degrees? $\boxed{??}$

Superposition integral _____ for any linear system

Consider a linear system with system function \mathcal{S} operating on input $f(x, y)$. We want to find a simple expression for the output image $g(x, y)$ in terms of the input image $f(x, y)$. We would like an expression that is more informative than $g = \mathcal{S}[f]$. The PSF, the impulse decomposition, and the superposition property are the key ingredients.

To aid interpretation, first consider a finite weighted sum of input impulses:

$$f(x, y) = \sum_n \alpha_n \delta_2(x - x_n, y - y_n) \xrightarrow{\mathcal{S}} g(x, y) = \sum_n \alpha_n h(x, y; x_n, y_n).$$

Now generalize to a continuum using the “strong” superposition property (see Appendix):

$$\begin{aligned} f(x, y) = \iint f(x', y') \delta_2(x, y; x', y') dx' dy' &\xrightarrow{\mathcal{S}} g(x, y) = \iint f(x', y') \mathcal{S}[\delta_2(\cdot, \cdot; x', y')](x, y) dx' dy' \\ &= \iint f(x', y') \mathcal{S}[\delta_2(x, y; x', y')] dx' dy' \\ &= \iint f(x', y') h(x, y; x', y') dx' dy', \end{aligned}$$

where we have substituted in the definition of h . The quoted equals warns you of the dangerous notation discussed on p. 1.40.

Thus we have derived the following input-output relationship for *linear* systems, known as the **superposition integral**:

$$\boxed{g(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x', y') h(x, y; x', y') dx' dy' .} \quad (1.23)$$

This is the general superposition integral for the case where we have decomposed the input image into Dirac impulses.

The superposition integral tells us that once we have found the PSF of the imaging system for all input coordinates, then the output is *fully* determined for *any* input. Thus, a linear system function \mathcal{S} is characterized fully by its PSF h .

Thus we will rarely (if ever) explicitly work with \mathcal{S} , for linear systems.

Example. Previously we considered an “ideal mirror” system where $g(x, y) = f(-x, -y)$. A more realistic mathematical model would account for the finite size of the mirror, *e.g.*, if the mirror size is $\text{FOV}_X \times \text{FOV}_Y$ then a better model could be

$$g(x, y) = \begin{cases} f(-x, -y) & |x| \leq \text{FOV}_X \text{ and } |y| \leq \text{FOV}_Y \\ 0, & \text{otherwise} \end{cases} = f(-x, -y) \text{rect}_2\left(\frac{x}{\text{FOV}_X}, \frac{y}{\text{FOV}_Y}\right).$$

This system is clearly linear, but it is not shift invariant due to both the mirror reflection *and* the finite **field of view (FOV)**.

Because the system is linear, it can be written using the superposition integral (1.23), provided we define correctly the PSF $h(x, y; x', y')$. By examining the output of this system when the input is $\delta_2(x, y; x', y')$, the PSF is

$$h(x, y; x', y') = \delta_2(x + x', y + y') \text{rect}_2\left(\frac{x'}{\text{FOV}_X}, \frac{y'}{\text{FOV}_Y}\right).$$

One can verify that this PSF expression is correct by substituting it into (1.23) and simplifying. Thus the input-output relationship for this system in superposition integral form is:

$$g(x, y) = \iint f(x', y') \delta_2(x + x', y + y') \text{rect}_2\left(\frac{x'}{\text{FOV}_X}, \frac{y'}{\text{FOV}_Y}\right) dx' dy'.$$

For an image capture system, the ideal PSF would be simply a Dirac impulse:

$$h(x, y; x', y') = \delta_2(x - x', y - y'),$$

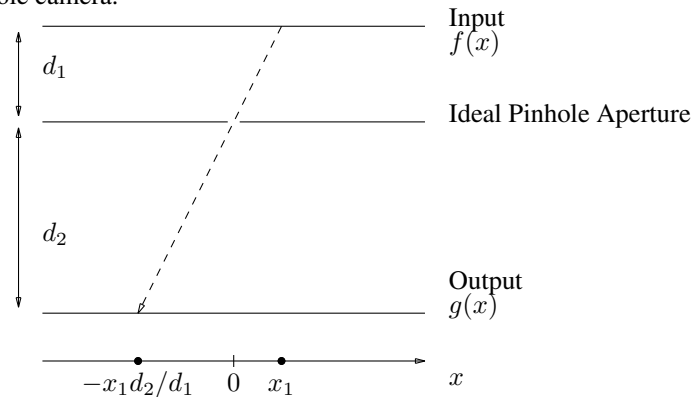
so that the system output would exactly equal the input: $g = \mathcal{S}[f] = f$.

In real systems there is always some blurring (imperfect resolution), if not other distortions as well. This blurring means that an input Dirac impulse gets “spread out” at the output, hence the term.

Analysis of imaging systems is often firstly concerned with:

- finding h for the system,
- and then understanding how h is influenced by the various system parameters (such as detector spacing, etc.).
- Then we can design system parameters to “improve” $h(x, y)$.
- Particularly useful as a rule-of-thumb for examining h is the width of the PSF, often specified by the **full-width at half maximum (FWHM)** [wiki].

Example: Consider the following 1D pinhole camera.



How do we find the PSF? It varies by imaging system. For this system we can use simple geometric reasoning. For incoherent illumination, this system is linear (in intensity).

If the input were a Dirac impulse at $x = x_1$, then for an ideal (infinitesimal) pinhole the output would be an Dirac impulse at $x = -x_1 d_2 / d_1 = m x_1$, where $m \triangleq -d_2 / d_1$ is called the **source magnification factor** [9, p. 44]. (For now we ignore the fact that light intensity decreases with propagation distance.)

Thus by simple ray-tracing we see that (for an ideal pinhole) the PSF would be given by:

$$h(x; x') = \delta(x - m x').$$

Substituting into the 1D superposition integral (cf. (1.23)) and simplifying by the sifting property yields the input-output relation:

$$g(x) = \frac{1}{|m|} f(x/m).$$

Note that the system is highly shift-variant, due to the mirroring and (spatial) scaling / magnification. On the other hand, as will be discussed later, with a simple coordinate change for the input image, the system can be considered to be shift-invariant!

Example. If the input is $f(x) = e^{-(x-3)^2}$ (a Gaussian bump centered at $x = 3$), then we can use the superposition integral to compute the output:

$$g(x) = \int_{-\infty}^{\infty} f(x')h(x; x') dx' = \int_{-\infty}^{\infty} e^{-(x'-3)^2} \delta(x - mx') dx' = \frac{1}{|m|} e^{-\left(\frac{x-3m}{m}\right)^2}.$$

Thus the output is a Gaussian bump centered at $-3|m|$ and “stretched out” (magnified) by the factor $|m|$. The superposition integral is easier than a “direct” attempt to find $g(x)$.

If the pinhole has finite width w , then the PSF is not an Dirac impulse, but rather a rectangular function of width $w(d_1 + d_2)/d_1$ centered at $-x'd_2/d_1$:

$$h(x; x') = \text{rect}\left(\frac{x - mx'}{wM}\right),$$

where $M = 1 + d_2/d_1$ is the **aperture magnification factor**. If the aperture plane has finite thickness, then the width of the rectangle will vary with x' , which causes further shift-variance.

If we account for the $1/r^2$ falloff with distance of the intensity of incoherent radiation, then the PSF of an ideal pinhole will be:

$$h(x; x') = \delta(x - mx') \frac{1}{r^2(x)},$$

where the distance between the source point and the detector point is

$$r(x) = \sqrt{(d_1 + d_2)^2 + (x + xd_1/d_2)^2} = (d_1 + d_2)\sqrt{1 + (x/d_2)^2}.$$

This intensity effect adds an additional form of shift-variance, although we can often ignore it for points near the axis (paraxial approximation).

The effect of a finite detector aperture of width W could be described by truncating the PSF by multiplying by $\text{rect}(x/W)$. This also makes the system shift-varying.

The PSF depends on system “design” parameters: d_2 , d_1 , and w . Specifically, the ratio d_2/d_1 determines a magnification factor. When is magnification useful?

- ??
- ??

Shift invariance or **space invariance** of **linear systems**

In any *linear* system where the point spread function is the same for all input points (except for a shift), we can greatly simplify general superposition integral.

Fact. A linear system satisfies the **shift invariant** or **space invariant** condition (1.21) iff the point spread function satisfies

$$\boxed{h(x, y; x', y') = h(x - x', y - y'; 0, 0), \quad \forall x, y, x', y'.} \quad (1.24)$$

Proof. We must show that (1.21) holds iff (1.24) holds.

(Sufficiency.) Suppose (1.24) holds. Then (1.23) becomes

$$f \xrightarrow{\mathcal{S}} g \iff g(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x, y; x', y') f(x', y') dx' dy' = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x - x', y - y') f(x', y') dx' dy'.$$

Making the change of variables $(x'', y'') = (x' + x_1, y' + y_1)$ we have

$$\begin{aligned} g(x - x_1, y - y_1) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x - x_1 - x', y - y_1 - y') f(x', y') dx' dy' \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x - x'', y - y'') f(x'' - x_1, y'' - y_1) dx'' dy'', \end{aligned}$$

which clearly shows that the general shift invariance condition (1.21) holds.

(Necessity.) If (1.24) does not hold for any input location (x', y') , then we can consider the impulse input image $f(x, y) = \delta_2(x, y)$ and another (translated) input image, $\delta_2(x - x', y - y')$, and the corresponding output images will not satisfy (1.21). \square

.....
 For shift-invariant systems, the $(0, 0)$ notation in (1.24) is superfluous. Therefore, to simplify notation we define

$$h(x, y) \triangleq h(x, y; 0, 0),$$

so that (with a little notation recycling) we can simply write

$$h(x, y; x', y') = h(x - x', y - y').$$

Because $h(x, y)$ is the response of the system to an impulse at the origin, if the system is linear and shift invariant, then we can determine the PSF by measuring the response to a single point-source input.

Example. X-ray CT phantom with tungsten wire...

To test for shift invariance of a linear system, first determine the system output for an impulse at the origin. Then shift or translate the input impulse by some arbitrary distance. Does the system output shift by the same distance, the same direction, and retain the same shape? If the answer yes (for any shift), then the system is shift invariant.

Example. Is the mirror system $g(x, y) = f(-x, -y)$ shift invariant?

No. As a specific counter-example, consider $\delta_2(x, y) \xrightarrow{S} \delta_2(x, y)$, but $\delta_2(x - 1, y) \xrightarrow{S} \delta_2(x + 1, y)$.

Example. Are the magnification and 1D pinhole systems shift invariant? ??

[RQ]

Example. Is the moving average filter (1.20) shift invariant? ??

[RQ]

Exercise. To show that a given linear system is shift-varying, does it suffice to check for counter-examples using impulse input functions (*e.g.*, as used in the mirror example above)? If so, prove it. If not, explain why.

1.3 LSI systems and convolution

For **linear shift-invariant (LSI)** systems, the input-output relationship simplifies from the general superposition integral (1.23) for linear systems to the following 2D **convolution** integral:

$$g(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x', y') h(x - x', y - y') dx' dy'. \quad (1.25)$$

We have the following notations for convolution:

$$g = f ** h, \text{ or } g(x, y) = (f ** h)(x, y), \text{ or dangerously imprecisely: } g(x, y) = f(x, y) ** h(x, y).$$

The notation `**` indicates 2D convolution. [See this link for an applet that illustrates convolution.](#)

LSI systems are easier to describe than linear shift-variant systems, because we only need the two-argument PSF $h(x, y)$, rather than the general PSF $h(x, y; x', y')$ with its 4 arguments. Furthermore, for LSI systems we can simplify the convolution using Fourier methods. (See Ch. 2.)

Unfortunately, *in most image capture systems, the PSF is not exactly shift invariant!* For example, in phased-array ultrasound the PSF in the region of the transmit focus is vastly different from the PSF near the face of the array.

However, the PSF of most imaging systems varies *slowly* over space. (If this were not the case then the images might be very difficult to interpret visually.) Consequently, for the purposes of system design and analysis, the PSF can often be considered shift invariant over local regions of the image.

Linearity alone does not imply shift-invariance, as in the mirror system example above.

One rarely performs the 2D convolution integral (1.25) analytically. Whenever possible we use properties instead.

Technicalities _____ **(skip)**

Note² `**` : $(\mathcal{L}_2(\mathbb{R}^2) \times \mathcal{L}_1(\mathbb{R}^2)) \cup (\mathcal{L}_2(\mathbb{R}^2) \times \mathcal{L}_1(\mathbb{R}^2)) \rightarrow \mathcal{L}_2(\mathbb{R}^2)$.

Challenge. Prove or disprove that `**` : $\mathcal{L}_2(\mathbb{R}^2) \times \mathcal{L}_2(\mathbb{R}^2) \rightarrow \mathcal{L}_2(\mathbb{R}^2)$.

² For a more precise conditions under which convolution is defined rigorously, see EECS 600.

2D convolution properties (relevant to LSI systems)

The following properties of 2D convolution are easily verified directly from the 2D convolution integral by manipulations that are essentially identical to those for 1D convolution.

- **Commutative** property: $f ** h = h ** f$
- **Associative** property: $[f ** h_1] ** h_2 = f ** [h_1 ** h_2]$
- **Distributive** property: $f ** [h_1 + h_2] = [f ** h_1] + [f ** h_2]$
- The order of serial connection of LSI systems does not affect the overall impulse response: $h_1 ** h_2 = h_2 ** h_1$.
- $f(x, y) ** \delta_2(x, y) = f(x, y)$
- **Shift** property: $f(x, y) ** \delta_2(x - x', y - y') = f(x - x', y - y')$
- **Shift-invariance**: If $g(x, y) = f(x, y) ** h(x, y)$, then $f(x - x_1, y - y_1) ** h(x - x_2, y - y_2) = g(x - x_1 - x_2, y - y_1 - y_2)$
- **Separability** property: $[f_1(x)f_2(y)] ** [h_1(x)h_2(y)] = [f_1(x) * h_1(x)] \cdot [f_2(y) * h_2(y)]$. (1.26)

Convoluting two separable functions yields a separable function.

What is $\delta(x) ** \delta(y)$? Hint: $\delta(y) = 1 \delta(y)$. ??

[RQ]

- **Circular symmetry** property: if $f(x, y)$ and $h(x, y)$ are circularly symmetric, then so is $f(x, y) ** h(x, y)$.
- **Scaling** property.
- **Exercise**. If $g(x, y) = f(x, y) ** h(x, y)$, then what is $f(2x, 2y) ** h(2x, 2y)$?

- **Support** property

If $f(x, y)$ has support $[0, a] \times [0, b]$ and $h(x, y)$ has support $[0, c] \times [0, d]$ then what is the support of $g = f ** h$? ??

LSI system properties via PSF

Because an LSI system is characterized completely by its PSF, we should be able to express all the other five system properties in terms of conditions on its PSF $h(x, y)$.

- causal: $h(x, y) = 0$ whenever $x < 0$ or $y < 0$?

That would be the natural analog to the 1D case, but it is essentially irrelevant for imaging.

- memoryless (static): ??

- stable: ??

Exercise: state and prove the appropriate condition for stability.

- rotational invariance.

Under what conditions on the PSF $h(x, y)$ is an LSI system rotationally invariant? ??

[HW1]

- invertible: For what type of $h(x, y)$ is an LSI system invertible? ??

Fact. If $h(x, y) ** f(x, y) = 0_{\text{signal}}$ for some nonzero signal $f(x, y)$, then the system with PSF $h(x, y)$ is not invertible.

Exercise. Prove or disprove the converse of the above Fact. If every non-zero input signal $f(x, y)$ produces a non-zero output signal then the system is invertible.

??

Separability

Fact. An LSI imaging system \mathcal{S} is a **separable system** if and only if its PSF $h(x, y)$ is a **separable function** (note terminology!), *i.e.*, $h(x, y) = h_1(x)h_2(y)$ for some 1D functions h_1 and h_2 .

For a separable PSF, 2D convolution simplifies into two sets of 1D convolution operations:

$$\begin{aligned} f(x, y) ** h(x, y) &= f(x, y) ** (h_1(x)h_2(y)) = f(x, y) ** ((h_1(x) \delta(y)) ** (\delta(x) h_2(y))) \\ &= (f(x, y) ** (h_1(x) \delta(y))) ** (\delta(x) h_2(y)), \end{aligned}$$

$$\text{where } f(x, y) ** (h_1(x) \delta(y)) = \iint f(x - x', y - y') h_1(x') \delta(y') dx' dy' = \int f(x - x', y) h_1(x') dx'.$$

Example. The moving average filter (1.20) is separable because its PSF is $h(x, y) = \text{rect}_2\left(\frac{x}{\Delta_x}, \frac{y}{\Delta_y}\right) = \text{rect}\left(\frac{x}{\Delta_x}\right) \text{rect}\left(\frac{y}{\Delta_y}\right)$, which is separable. In fact, we can see that this system is separable by rewriting slightly the original input-output relation (1.20):

$$g(x, y) = \frac{1}{\Delta_y} \int_{y-\Delta_y/2}^{y+\Delta_y/2} \left[\frac{1}{\Delta_x} \int_{x-\Delta_x/2}^{x+\Delta_x/2} f(x', y') dx' \right] dy'. \quad (1.27)$$

The inner integral acts only along the x direction, *i.e.*, it is a “horizontal acting” operation, whereas the outer integral acts only in the y direction, *i.e.*, it is a “vertical acting” operation.

The 2D convolution integral for any LSI system with a separable PSF can be decomposed similarly:

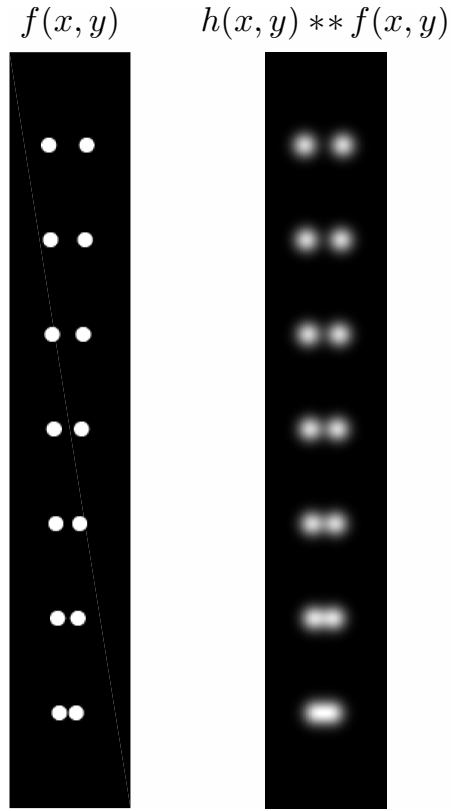
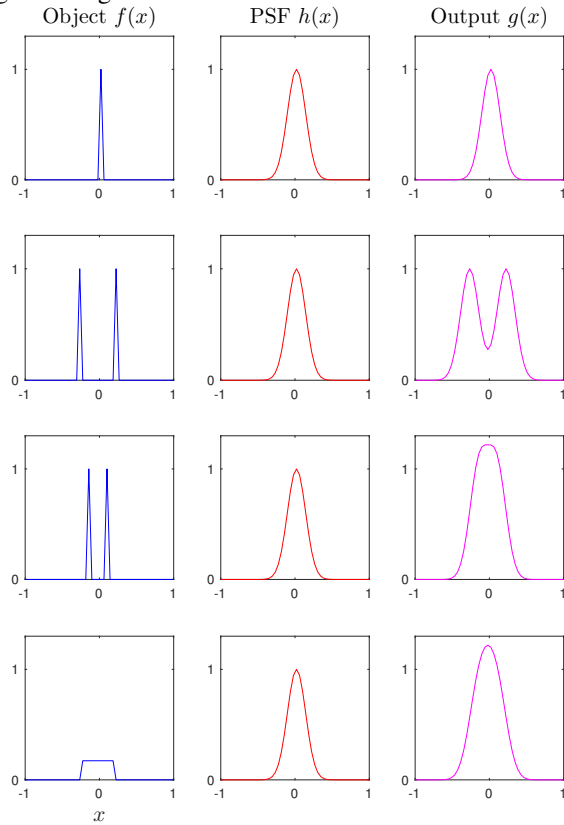
$$\begin{aligned} h(x, y) = h_1(x)h_2(y) \implies g(x, y) = h(x, y) ** f(x, y) &= \int \left[\int h_1(x - x') f(x', y') dx' \right] h_2(y - y') dy' \\ &= \int \left[\int h_2(y - y') f(x', y') dy' \right] h_1(x - x') dx'. \end{aligned}$$

Differential equations?

At this point in a 1D signals and systems course, most texts address the specific case of systems described by linear constant coefficient differential equations, such as RLC circuits. Indeed, the analysis of such systems occupies a substantial portion of the effort in such courses. Although there are certainly some important differential equations relevant to imaging, such as the wave equation, there is not a clear analog to RLC systems. This is a significant difference between 1D (time) signals and systems and multidimensional signal processing.

Effect of PSF on spatial resolution

Example. The following figures illustrate PSF and spatial resolution in 1D and 2D. For the 1D figures, note how similar the bottom right plot is to the one above it, so it is difficult to tell from the output signal whether the input was a pair of closely spaced impulses or a single rect signal.



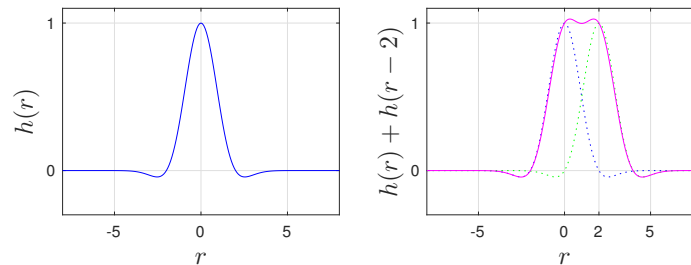
Resolution criteria

Sometimes an image contains the superposition of two closely spaced impulse-like features, *i.e.*, two closely spaced “dots.” If their spacing is sufficiently close then their appearance will be indistinguishable from, simply, one dot. If they are distinguishable, then one says they are **resolved**. It is sometimes important to develop a mathematical criteria for determining when this will happen.

For example, from [10, p. 326]:

the question of when two closely spaced point sources are barely resolved is a complex one and lends itself to a variety of rather subjective answers. According to the so-called **Rayleigh resolution criterion**, two equally bright points are barely resolved when the first zero of the Airy pattern [PSF] of the image of one point exactly coincide with the central maximum of the [PSF] of the image of the second point. ... An alternative definition is the so-called **Sparrow criterion**, which states that two point sources are just resolved if the second derivative of the image intensity pattern vanishes at the point midway between the [sum of the two PSFs].

Example. If the PSF of a hypothetical imaging system is $h(r) = \exp(-\pi(r/3)^2) \cos(2\pi r/8)$, then the first zero is at $r = 2$, so the “just resolved distance” is 2 (in whatever unit is associated with “8”).

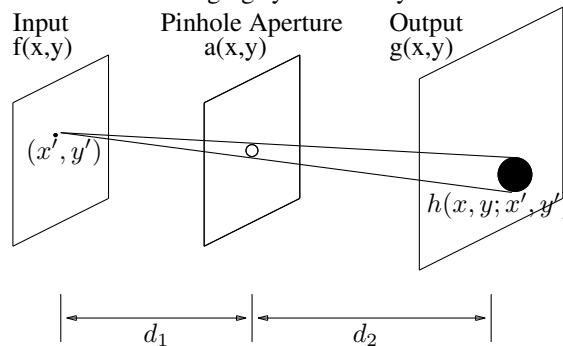


Magnification and LSI systems

As illustrated by the 1D pinhole example above, any system with magnification or mirroring is, strictly speaking, a shift-variant system, because the PSF does not depend solely on the difference between input and output coordinates.

However, we can recast such systems into equivalent shift-invariant systems by suitable coordinate changes for the input signal. This is reasonable in the context of imaging systems because the coordinates that we use are somewhat arbitrary. (In contrast, in the context of time-domain systems, “coordinate changes” are nontrivial.)

We will show the required coordinate change for the example of a 2D pinhole camera. Among other things, this example will illustrate the type of manipulations that are useful for imaging systems analysis.



Consider the simple pinhole camera system shown above, assumed linear, with a pinhole opening described by a real **aperture function** $0 \leq a(x, y) \leq 1$, where $a(x, y)$ denotes the fraction of light incident on the point (x, y) that passes through.

Example. For a hole of diameter r_0 , the aperture is $a(x, y) = a(r) = \text{rect}(r/r_0)$.

If the thickness of the aperture plane is infinitesimal (and ignoring $1/r^2$ intensity falloff), then by geometry, the PSF is

$$h(x, y; x', y') = h(x - mx', y - my'), \quad \text{where } h(x, y) = a(x/M, y/M),$$

where $m = -d_2/d_1$ is the **source magnification factor**, and $M = \frac{d_1+d_2}{d_1} = 1 - m$ is called the **object magnification factor** (but might better be called the **aperture magnification factor**).

What does the minus sign in the definition of m indicate? ??

Assuming this is a linear system, the output intensity is written in terms of the PSF by the superposition integral:

$$g(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x', y') h(x, y; x', y') dx' dy' = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x', y') h(x - mx', y - my') dx' dy'.$$

We can now rewrite the system equation using the simple change of variables $(x_m, y_m) = (mx', my')$, leading to:

$$g(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f_m(x_m, y_m) h(x - x_m, y - y_m) dx_m dy_m = (f_m ** h)(x, y). \quad (1.28)$$

This is now a simple convolution of the modified image $f_m(x, y) \triangleq \frac{1}{m^2} f(x/m, y/m)$, which is a magnified, scaled and mirrored version of the input image, obtained via the coordinate change above, and the PSF $h(x, y)$, which is a magnified version of the aperture function.

To understand the $1/m^2$ in the definition of f_m , consider that for an infinitesimal pinhole, the system magnifies the image by the factor m in each dimension. Thus the spatial extent of the input image is extended by m^2 in the output. But because the total radiation striking the output plane does not change (*i.e.*, passive camera system), the intensity of the output image is reduced by the factor $1/m^2$.

The magnified image $f_m(x, y)$ is what the system output would be for an ideal pinhole aperture, *i.e.*, when $a(x, y) = \delta_2(x, y)$.

Somewhere above we ignored the finite detector size, which is another source of shift variance.

Where? ??

How would we modify (1.28) to account for finite detector size?

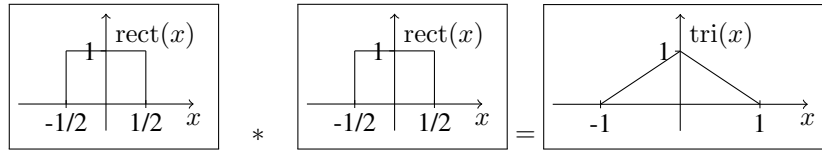
??

Convolution examples

Example. In 1D, it is well known that the convolution of two rect functions of equal width yields a triangle function:

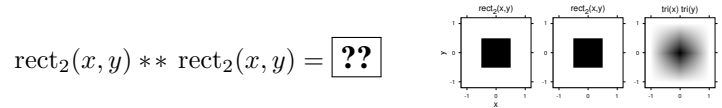
$$\text{rect}(x) * \text{rect}(x) = \text{tri}(x) \triangleq (1 - |x|) \text{rect}\left(\frac{x}{2}\right).$$

(Any reader unfamiliar with this should verify from the definition of convolution. Graphically:



In 2D, using the separability property (1.26) of 2D convolution:

[RQ]



For 1D convolution, the variable name is unimportant: $\text{rect}(t) * \text{rect}(t) = \text{tri}(t)$ and $\text{rect}(r) * \text{rect}(r) = \text{tri}(r)$.

In 2D we must remember our convention that $\text{rect}(r)$ is a shorthand for the 2D circularly symmetric function $\text{rect}(\sqrt{x^2 + y^2})$. In particular, 2D convolution of $\text{rect}(r)$ with itself does *not* yield $\text{tri}(r)$. Using the area of a circular segment [wiki], one can show by 2D convolution that

$$\text{rect}(r) ** \text{rect}(r) = \begin{cases} \frac{1}{2} (\arccos(|r|) - |r| \sqrt{1 - r^2}), & |r| \leq 1 \\ 0, & \text{otherwise.} \end{cases}$$

This function is called the **chinese hat** function. It is not the same shape as a triangle.

Summary

This chapter has introduced 2D signals and 2D systems.

The majority of the concepts in 2D generalize readily from those in 1D. New concepts include **separability** and **rotation**.

What part of this chapter did you find most confusing? ??

[RQ]

1.4 Appendix: Vector Spaces and Linear Operators (skip)

Linear systems are represented mathematically by linear transformations from one vector space to another.

For completeness, we give below the definition of a vector space (in more generality than we will actually use).

That definition uses the concept of a field of scalars, so we first review that.

Field of scalars

A **field of scalars** \mathcal{F} is a collection of elements $\alpha, \beta, \gamma, \dots$ along with an “addition” and a “multiplication” operator [11].

To every pair of scalars α, β in \mathcal{F} , there must correspond a scalar $\alpha + \beta$ in \mathcal{F} , called the **sum** of α and β , such that

- Addition is commutative: $\alpha + \beta = \beta + \alpha$
- Addition is associative: $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$
- There exists a unique element $0 \in \mathcal{F}$, called **zero**, for which $\alpha + 0 = \alpha, \forall \alpha \in \mathcal{F}$
- For every $\alpha \in \mathcal{F}$, there corresponds a unique scalar $(-\alpha) \in \mathcal{F}$ for which $\alpha + (-\alpha) = 0$.

To every pair of scalars α, β in \mathcal{F} , there must correspond a scalar $\alpha\beta$ in \mathcal{F} , called the **product** of α and β , such that

- Multiplication is commutative: $\alpha\beta = \beta\alpha$
- Multiplication is associative: $\alpha(\beta\gamma) = (\alpha\beta)\gamma$
- Multiplication distributes over addition: $\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$
- There exists a unique element $1 \in \mathcal{F}$, called **one**, or **unity**, or the **identity** element, for which $1\alpha = \alpha, \forall \alpha \in \mathcal{F}$
- For every nonzero $\alpha \in \mathcal{F}$, there corresponds a unique scalar $\alpha^{-1} \in \mathcal{F}$, called the **inverse** of α for which $\alpha\alpha^{-1} = 1$.

Simple facts for fields:

- $0 + 0 = 0$
- $-0 = 0$

One example of a field is the set \mathbb{Q} of rational numbers (with the usual definitions of addition and multiplication).

The only fields that we will need are the field of real numbers \mathbb{R} and the field of complex numbers \mathbb{C} .

Vector spaces

A **vector space** or **linear space** consists of

- A field \mathcal{F} of scalars.
- A set \mathcal{V} of entities called **vectors**
- An operation called **vector addition** that associates a **sum** $\mathbf{x} + \mathbf{y} \in \mathcal{V}$ with each pair of vectors $\mathbf{x}, \mathbf{y} \in \mathcal{V}$ such that
 - Addition is commutative: $\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$
 - Addition is associative: $\mathbf{x} + (\mathbf{y} + \mathbf{z}) = (\mathbf{x} + \mathbf{y}) + \mathbf{z}$
 - There exists a unique element $\mathbf{0} \in \mathcal{V}$, called the **zero vector**, for which $\mathbf{x} + \mathbf{0} = \mathbf{x}, \forall \mathbf{x} \in \mathcal{V}$
 - For every $\mathbf{x} \in \mathcal{V}$, there corresponds a unique vector $(-\mathbf{x}) \in \mathcal{V}$ for which $\mathbf{x} + (-\mathbf{x}) = \mathbf{0}$.
- An operation called **multiplication by a scalar** that associates with each scalar $\alpha \in \mathcal{F}$ and vector $\mathbf{x} \in \mathcal{V}$ a vector $\alpha\mathbf{x} \in \mathcal{V}$, called the **product** of α and \mathbf{x} , such that:
 - Associative: $\alpha(\beta\mathbf{x}) = (\alpha\beta)\mathbf{x}$
 - Distributive $\alpha(\mathbf{x} + \mathbf{y}) = \alpha\mathbf{x} + \alpha\mathbf{y}$
 - Distributive $(\alpha + \beta)\mathbf{x} = \alpha\mathbf{x} + \beta\mathbf{x}$
 - If 1 is the identify element of \mathcal{F} , then $1\mathbf{x} = \mathbf{x}, \forall \mathbf{x} \in \mathcal{V}$.
- No operations are presumed to be defined for multiplying two vectors or adding a vector and a scalar.

Simple facts for vector spaces:

- $0\mathbf{x} = \mathbf{0}$ for $\mathbf{x} \in \mathcal{V}$. Proof: $\mathbf{x} = 1\mathbf{x} = (1 + 0)\mathbf{x} = 1\mathbf{x} + 0\mathbf{x} = \mathbf{x} + 0\mathbf{x}, \forall \mathbf{x} \in \mathcal{V}$. Thus $0\mathbf{x} = \mathbf{0}$
- $(-1)\mathbf{x} = -\mathbf{x}$ for $\mathbf{x} \in \mathcal{V}$. Proof: $\mathbf{x} + (-1)\mathbf{x} = 1\mathbf{x} + (-1)\mathbf{x} = (1 + (-1))\mathbf{x} = 0\mathbf{x} = \mathbf{0}$.
- $\alpha\mathbf{0} = \mathbf{0}$ for $\alpha \in \mathcal{F}$. Proof: $\alpha\mathbf{0} = \alpha\mathbf{0} + \mathbf{0} = \alpha\mathbf{0} + [\alpha\mathbf{0} + (-\alpha\mathbf{0})] = [\alpha\mathbf{0} + \alpha\mathbf{0}] + (-\alpha\mathbf{0}) = \alpha(\mathbf{0} + \mathbf{0}) + (-\alpha\mathbf{0}) = \alpha\mathbf{0} + (-\alpha\mathbf{0}) = \mathbf{0}$.

Examples of important vector spaces

- **Euclidean n -dimensional space** or **n -tuple space**: $\mathcal{V} = \mathbb{R}^n$. If $\mathbf{x} \in \mathcal{V}$, then $\mathbf{x} = (x_1, x_2, \dots, x_n)$ where $x_i \in \mathbb{R}$. The field of scalars is $\mathcal{F} = \mathbb{R}$. Of course $\mathbf{x} + \mathbf{y} = (x_1 + y_1, x_2 + y_2, \dots, x_n + y_n)$, and $\alpha\mathbf{x} = (\alpha x_1, \dots, \alpha x_n)$.
- **Complex Euclidean n -dimensional space**: $\mathcal{V} = \mathbb{C}^n$. If $\mathbf{x} \in \mathcal{V}$, then $\mathbf{x} = (x_1, x_2, \dots, x_n)$ where $x_i \in \mathbb{C}$. The field of scalars is $\mathcal{F} = \mathbb{C}$.
Of course $\mathbf{x} + \mathbf{y} = (x_1 + y_1, x_2 + y_2, \dots, x_n + y_n)$, and $\alpha\mathbf{x} = (\alpha x_1, \dots, \alpha x_n)$.
- $\mathcal{V} = \mathcal{L}_2(\mathbb{R}^3)$. The set of functions $f : \mathbb{R}^3 \rightarrow \mathbb{C}$ that are **square integrable**: $\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} |f(x, y, z)|^2 dx dy dz < \infty$. The field is $\mathcal{F} = \mathbb{C}$. Addition and scalar multiplication are defined in the natural way.
To show show that $f, g \in \mathcal{L}_2(\mathbb{R}^3)$ implies $f + g \in \mathcal{L}_2(\mathbb{R}^3)$, one can apply the triangle inequality: $\|f + g\| \leq \|f\| + \|g\|$ where $\|f\| = \langle f, f \rangle$, and where it can be shown that $\langle f, g \rangle = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y, z)g^*(x, y, z) dx dy dz$ is a valid inner product.
- The set of functions on the plane \mathbb{R}^2 that are zero outside of the unit square.
- The set of solutions to a homogeneous linear system of equations $A\mathbf{x} = \mathbf{0}$.

2.1 Linear transformations and linear operators

Let \mathcal{U} and \mathcal{V} be two vector spaces over a common field \mathcal{F} .

A function $A : \mathcal{U} \rightarrow \mathcal{V}$ is called a **linear transformation** or **linear mapping** from \mathcal{U} into \mathcal{V} iff $\forall \mathbf{u}_1, \mathbf{u}_2 \in \mathcal{U}$ and all scalars $\alpha, \beta \in \mathcal{F}$:

$$A(\alpha\mathbf{u}_1 + \beta\mathbf{u}_2) = \alpha A(\mathbf{u}_1) + \beta A(\mathbf{u}_2).$$

Example:

- Let $\mathcal{F} = \mathbb{R}$ and let \mathcal{V} be the space of continuous functions on \mathbb{R} . Define the linear transformation A by: if $F = A(f)$ then $F(x) = \int_0^x f(t) dt$. Thus integration (with suitable limits) is linear.

If A is a linear transformation from \mathcal{V} into \mathcal{V} , then we say A is a **linear operator**. However, the terminology distinguishing linear transformations from linear operators is not universal, and the two terms are often used interchangeably.

Simple fact for linear transformations:

- $A[\mathbf{0}] = \mathbf{0}$. Proof: $A[\mathbf{0}] = A[0\mathbf{0}] = 0A[\mathbf{0}] = \mathbf{0}$. This is called the “zero in, zero out” property.

Caution! (From Linear Systems by T. Kailath.) By induction it follows that

$$A \left(\sum_{i=1}^n \alpha_i \mathbf{u}_i \right) = \sum_{i=1}^n \alpha_i A(\mathbf{u}_i)$$

for any finite n , but the above *does not* imply in general that linearity holds for infinite summations or integrals.

Further assumptions about “smoothness” or “regularity” or “continuity” of A are needed for that.

This is usually no problem for real imaging systems, so we will use infinite sums whenever needed without further ado.

Although additional regularity conditions on the linear operator A are necessary to ensure this, we assume hereafter that the superposition property holds even for infinite summations, including integrals. That is, if for each Ω in a set Ω , the function f_Ω is square integrable, then we assume

$$A \left[\int_{\Omega} a(\Omega) f_\Omega \, d\Omega \right] = \int_{\Omega} a(\Omega) A[f_\Omega] \, d\Omega \quad \text{“strong” superposition property.}$$

Thus we will exchange the order of integration and linear operators freely in our analysis.

Bibliography

- [1] R. Bracewell. *The Fourier transform and its applications*. New York: McGraw-Hill, 1978.
- [2] R. N. Bracewell. *Two-dimensional imaging*. New York: Prentice-Hall, 1995.
- [3] J. S. Lim. *Two-dimensional signal and image processing*. New York: Prentice-Hall, 1990. ISBN: 978-0139353222.
- [4] C. Kittel. *Introduction to solid state physics*. New York: Wiley, 1976.
- [5] P. E. Varberg. “Change of variables in multiple integrals”. In: *Amer. Math. Monthly* 78.1 (Jan. 1971), 42–5.
- [6] R. E. Williamson, R. H. Crowell, and H. F. Trotter. *Calculus of vector functions*. New Jersey: Prentice-Hall, 1972.
- [7] M. Mellor and M. Brady. “Phase mutual information as a similarity measure for registration”. In: *Med. Im. Anal.* 9.4 (Aug. 2005), 330–43.
- [8] M. Unser, P. Thevenaz, and L. Yaroslavsky. “Convolution-based interpolation for fast, high quality rotation of images”. In: *IEEE Trans. Im. Proc.* 4.10 (Oct. 1995), 1371–81.
- [9] A. Macovski. *Medical imaging systems*. New Jersey: Prentice-Hall, 1983.
- [10] J. W. Goodman. *Statistical optics*. New York: Wiley, 1985.
- [11] B. Noble and J. W. Daniel. *Applied linear algebra*. 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1977.

Chapter 2

2D Fourier Transforms

Contents (student version)

| | |
|---|-------------|
| 2.0 Introduction | 2.2 |
| 2.1 Representation of signals by orthogonal bases | 2.2 |
| Fourier series | 2.14 |
| 2.2 Fourier transforms | 2.21 |
| Two-dimensional Fourier transform | 2.22 |
| Relation to 1D Fourier transform | 2.23 |
| 2D Fourier transform properties that match 1D FT properties | 2.27 |
| Properties that are analogous to 1D FT properties | 2.27 |
| Generalized Fourier transforms | 2.30 |
| 2D Fourier transform and LSI systems | 2.31 |
| Properties that are new for the 2D FT | 2.32 |
| Using lenses for 2D Fourier transforms | 2.35 |
| Periodic signals: relating 2D FT and 2D Fourier series | 2.36 |
| Hankel transform | 2.38 |
| Other 2D transforms | 2.46 |
| Bandwidth and time-bandwidth products | 2.48 |
| Summary | 2.50 |

2.0 Introduction

This chapter discusses 2D Fourier series, 2D Fourier transforms, and related transforms for 2D continuous-space signals. New concepts (compared to 1D Fourier transforms) are related to separability and circular symmetry.

2.1 Representation of signals by orthogonal bases

The next topic usually discussed in a 1D signals and systems text is **Fourier series**. Although Fourier methods are very important in imaging, we are also often interested in other representations of 2D signals, such as wavelets. We first describe the general representation of signals by orthogonal bases, and then consider Fourier series using sinusoids as an important special case.

A lot of 1D signal processing is motivated by **audio** examples where periodic phenomena are prevalent, so the Fourier basis is a natural choice. In contrast, natural images are less likely to contain much periodic components, so the Fourier basis is often suboptimal. This is why we must consider alternative bases for imaging problems.

In engineering it is often very useful to represent signals by a linear combination of “simpler” signals:

$$g(x, y) = \sum_k c_k \phi_k(x, y). \quad (2.1)$$

This process of **signal analysis** is particularly convenient if we choose a set of signals $\{\phi_k\}$ that are **orthogonal**.

Review of inner products and orthogonality _____ (skim)

The concept of orthogonality is an extension of the geometric concept of perpendicular lines. Suppose two lines in the plane are perpendicular and intersect at the origin. Let \vec{u} and \vec{v} denote two vectors pointing along those lines. Because the lines are perpendicular, we say \vec{u} and \vec{v} are **orthogonal vectors**, meaning their **inner product** or **dot product** is zero. In Euclidean 2-space, the standard inner product of \vec{u} and \vec{v} is defined by

$$\langle \vec{u}, \vec{v} \rangle = \vec{u} \cdot \vec{v} = \vec{v}^T \vec{u} = u_1 v_1 + u_2 v_2.$$

More generally, for n -dimensional complex vectors $\mathbf{x}, \mathbf{y} \in \mathbb{C}^n$, the standard inner product is

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{y}' \mathbf{x} = \sum_{i=1}^n x_i y_i^*,$$

where \mathbf{y}' denotes the Hermitian transpose of a vector (or matrix).

To generalize the concept of **orthogonality** to another setting (*i.e.*, another vector space) beyond ordinary Euclidean space, the key is to define an appropriate inner product. Any **inner product** must satisfy the following conditions for all \mathbf{x} and \mathbf{y} in the space:

1. $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle^*$ (**Hermitian symmetry**), where $*$ denotes complex conjugate.
2. $\langle \mathbf{x} + \mathbf{y}, \mathbf{z} \rangle = \langle \mathbf{x}, \mathbf{z} \rangle + \langle \mathbf{y}, \mathbf{z} \rangle$ (**additivity**)
3. $\langle \alpha \mathbf{x}, \mathbf{y} \rangle = \alpha \langle \mathbf{x}, \mathbf{y} \rangle$ for all $\alpha \in \mathbb{C}$ (**scaling**)
4. $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$, and $\langle \mathbf{x}, \mathbf{x} \rangle = 0$ iff $\mathbf{x} = \mathbf{0}$. (**positive definite**)

Using the above conditions, one can show that any inner product is a **continuous** function [1, p. 49], and is a **bilinear form** [wiki] (or **sesquilinear form** [wiki] for complex signals) that satisfies:

$$\left\langle \sum_n \alpha_n \mathbf{x}_n, \sum_m \beta_m \mathbf{y}_m \right\rangle = \sum_n \sum_m \alpha_n \beta_m^* \langle \mathbf{x}_n, \mathbf{y}_m \rangle, \quad (2.2)$$

for any complex constants $\{\alpha_n\}$ and $\{\beta_m\}$ if $\sum_n \alpha_n \mathbf{x}_n$ and $\sum_m \beta_m \mathbf{y}_m$ are convergent series. (Whether the conjugate goes with the first or second argument is a matter of convention that differs between fields.)

Related to the concept of inner product is the concept of the **norm** of a vector, defined in terms of an inner product as follows:

$$\|\mathbf{x}\| \triangleq \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}.$$

In Euclidean space this measures the **length** of a vector, *i.e.*, $\|\vec{u}\| = \sqrt{\langle \vec{u}, \vec{u} \rangle} = \sqrt{\sum_{i=1}^n u_i u_i^*} = \sqrt{\sum_{i=1}^n |u_i|^2}$.

Also related is the concept of the **distance** between two vectors:

$$\|\mathbf{x} - \mathbf{y}\|.$$

In MATLAB, to compute the norm of a column vector \mathbf{x} , use `norm(x)` or `sqrt(sum(abs(x).^2, 'double'))`. Do not use `sqrt(x' * x)` because in finite precision computing, $\mathbf{x}' * \mathbf{x}$ is not always exactly real!

Inner products and orthogonality for 2D images

Here we are working with 2D continuous-space images; these are functions defined on \mathbb{R}^2 , rather than simple vectors in \mathbb{R}^n or \mathbb{C}^n . For images with domain $\mathcal{D} \subset \mathbb{R}^2$, the natural definitions of **inner product**, **norm** (squared) and **distance** are:

$$\begin{aligned}\langle f, g \rangle &= \iint_{\mathcal{D}} f(x, y) g^*(x, y) \, dx \, dy \\ \|f\|^2 &= \langle f, f \rangle = \iint_{\mathcal{D}} |f(x, y)|^2 \, dx \, dy \\ \|f - g\| &= \left(\iint_{\mathcal{D}} |f(x, y) - g(x, y)|^2 \, dx \, dy \right)^{1/2}.\end{aligned}$$

The norm of f is the square root of its energy.

Two images f and g are called **orthogonal** if $\langle f, g \rangle = 0$.

A set of 2D signals $\{\phi_k\}$ is called **orthogonal** iff the **inner product** of each pair of distinct signals in the set is zero:

$$\langle \phi_k, \phi_l \rangle = \iint_{\mathcal{D}} \phi_k(x, y) \phi_l^*(x, y) \, dx \, dy = \begin{cases} \mathcal{E}_k, & l = k \\ 0, & l \neq k \end{cases} = \mathcal{E}_k \delta[l - k],$$

where $\mathcal{E}_k = \|\phi_k\|^2$ is the energy of $\phi_k(x, y)$.

The 1D **Kronecker impulse** or **Kronecker delta function** (this one is a function!) is defined by $\delta[l] = \begin{cases} 1, & l = 0 \\ 0, & l \neq 0 \end{cases}$.

If $\mathcal{E}_k = 1$ for all k , then we call the signals **orthonormal**, because they are **normalized** to have unit energy (and thus unit norm).

For specific 2D examples, we will often want to write:

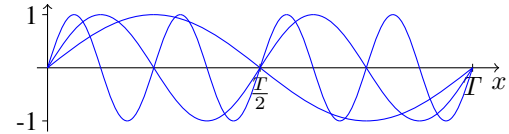
$$g(x, y) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} c_{k,l} \phi_{k,l}(x, y) \tag{2.3}$$

rather than (2.1), *i.e.*, we will often index the basis signals with double subscripts in 2D.

In this case, we say $\{\phi_{k_1, l_1}\}$ is an orthogonal set of signals if $\langle \phi_{k_1, l_1}, \phi_{k_2, l_2} \rangle = 0$ whenever $k_1 \neq k_2$ or $l_1 \neq l_2$.

In the following examples, we will use the following useful fact:

$$\int_0^T \sin(2\pi kx/T + \phi) dx = \begin{cases} 0, & k = \pm 1, \pm 2, \dots \\ T \sin \phi, & k = 0 \end{cases} = T \sin(\phi) \delta[k].$$

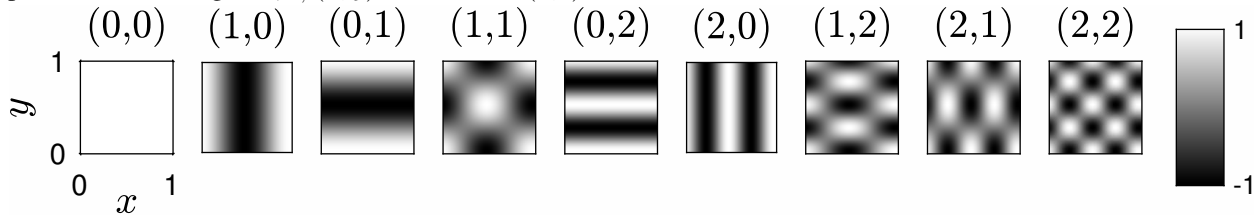


In other words, the 1D **harmonic sinusoidal signals** $\{\sin(2\pi kx/T)\}$ are an orthogonal set over the interval $[0, T]$.

Example. The (separable) harmonic sinusoids $\phi_{k,l}(x, y) = \cos(2\pi kx/T_x) \cos(2\pi ly/T_y)$ for $k, l = 1, 2, \dots$ are orthogonal on the set $\mathcal{D} = [0, T_x] \times [0, T_y]$ because

$$\langle \phi_{k,l}, \phi_{m,n} \rangle = \int_0^{T_x} \int_0^{T_y} \phi_{k,l}(x, y) \phi_{m,n}^*(x, y) dx dy = \dots = \frac{T_x T_y}{4} \delta[k - m] \delta[l - n].$$

The tuple above each image of $\phi_{k,l}(x, y)$ below shows (k, l) :



Example. The (separable) harmonic complex exponentials $\phi_{k,l}(x, y) = e^{i2\pi(kx/T_x + ly/T_y)}$ for $k, l \in \mathbb{Z}$ are orthogonal over the set $\mathcal{D} = [0, T_x] \times [0, T_y]$ because

$$\begin{aligned} \langle \phi_{k,l}, \phi_{m,n} \rangle &= \int_0^{T_x} \int_0^{T_y} \phi_{k,l}(x, y) \phi_{m,n}^*(x, y) dx dy = \int_0^{T_x} \int_0^{T_y} e^{i2\pi(k-m)x/T_x} e^{i2\pi(l-n)y/T_y} dx dy \\ &= T_x T_y \delta[k - m] \delta[l - n]. \end{aligned}$$

Generalized Fourier series

In image processing we are often interested in basis functions besides sinusoids and complex exponentials. In general, we will be interested in sets of orthogonal signals $\{\phi_k\}$ on some set \mathcal{D} such that if $g(x, y)$ has finite energy over that set, then we can represent $g(x, y)$ either exactly or approximately in terms of the ϕ_k functions. Because these concepts generalize the usual choice of sinusoids used in Fourier series, they are called **Generalized Fourier series** [wiki].

We start with the approximation perspective. If $\{\phi_k\}$ is a set of signals that are orthogonal on a set \mathcal{D} , we often want to find the “best” approximation of $g(x, y)$ over the set \mathcal{D} using a linear combination of just a finite subset of the functions $\{\phi_k\}$. Without loss of generality, we choose ϕ_k for $k = -N, \dots, N$, meaning that we want:

$$g(x, y) \approx g_N(x, y) \triangleq \sum_{k=-N}^N c_k \phi_k(x, y). \quad (2.4)$$

There are two issues: how to define the “best” approximation and how to determine the coefficients c_k .

The approximation error is the difference between the actual signal $g(x, y)$ and its finite series approximation:

$$e_N(x, y) \triangleq g(x, y) - g_N(x, y) = g(x, y) - \sum_{k=-N}^N c_k \phi_k(x, y).$$

One reasonable way to define the “best” approximation is to minimize the **energy** of the approximation error, *i.e.*, the **energy** of the error signal. (This criterion is reasonable because it is analytically tractable for one thing.) Thus we would like to choose the c_k values to minimize $\|e_N(x, y)\|^2$. This is essentially one form of “least squares” fitting. The solution, *i.e.*, the best choice of $\{c_k\}$ is:

$$c_k = \frac{\langle g, \phi_k \rangle}{\|\phi_k\|^2} = \frac{1}{\mathcal{E}_k} \iint_{\mathcal{D}} g(x, y) \phi_k^*(x, y) dx dy. \quad (2.5)$$

It is nice that the solution to this error minimization problem does not depend on N ! It is also nice that we can compute each c_k independently for each k ; if the signals ϕ_k were not orthogonal, then there would be a coupled system of equations to solve.

The c_k values are called **Fourier coefficients** with respect to the orthogonal set $\{\phi_k\}$, which is called an **orthogonal basis**. The series is called an **orthogonal series representation** for g or simply an **orthogonal series** or **orthogonal representation**.

Proof of (2.5):

$$\begin{aligned}
\|e_N(x, y)\|^2 &= \left\| g - \sum_{k=-N}^N c_k \phi_k \right\|^2 = \left\langle g - \sum_{k=-N}^N c_k \phi_k, g - \sum_{l=-N}^N c_l \phi_l \right\rangle && \text{apply bilinearity (2.2):} \\
&= \|g\|^2 - \sum_{l=-N}^N c_l \langle g, \phi_l \rangle - \sum_{k=-N}^N c_k \langle \phi_k, g \rangle + \sum_{k=-N}^N \sum_{l=-N}^N c_k c_l^* \langle \phi_k, \phi_l \rangle && \text{use orthogonality:} \\
&= \|g\|^2 + \sum_{k=-N}^N |c_k|^2 \mathcal{E}_k - \sum_{l=-N}^N c_l \langle g, \phi_l \rangle - \sum_{k=-N}^N c_k \langle \phi_k, g \rangle && \text{completing the square:} \\
&= \|g\|^2 + \sum_{k=-N}^N \mathcal{E}_k \left| c_k - \frac{1}{\mathcal{E}_k} \langle g, \phi_k \rangle \right|^2 - \sum_{k=-N}^N \frac{1}{\mathcal{E}_k} |\langle g, \phi_k \rangle|^2.
\end{aligned}$$

The first and third terms are independent of c_k , so the error signal energy is minimized when the middle term is equal to zero, which happens when and only when c_k is given by (2.5). Geometrically, using (2.4) with this choice of $\{c_k\}$ is called a **projection** of $g(x, y)$ onto the $(2N + 1)$ -dimensional subspace spanned by $\{\phi_k(x, y)\}_{k=-N}^N$.

In general the finite series (2.4) will be an approximation, and we would like to know whether the approximation improves as $N \rightarrow \infty$ and whether the approximation error goes to zero.

A set of orthogonal functions $\{\phi_k\}$ on \mathcal{D} is called **complete**, if, for every $g \in \mathcal{L}_2(\mathcal{D})$, i.e., for all signals with finite energy over \mathcal{D} , the error goes to zero when we use (2.5) and let N grow:

$$\|g - g_N\|_2^2 \rightarrow 0 \text{ as } N \rightarrow \infty.$$

For a complete orthogonal basis in $\mathcal{L}_2(\mathcal{D})$, we often write the following convergent series¹ where c_k was defined in (2.5):

$$g(x, y) = \sum_{k=-\infty}^{\infty} c_k \phi_k(x, y), \quad (x, y) \in \mathcal{D}. \quad (2.6)$$

The fact that (2.5) is optimal regardless of N is a result of measuring error by energy. If we instead had used, for example, the L_1 norm: $\iint_{\mathcal{D}} |e_N(x, y)| dx dy$ (integrated absolute error) as the measure of error, then the optimal choice for the c_k values is not (2.5), and is not independent of N !

From the above formula for the approximation error energy, we can deduce that when we choose the c_k values according to (2.5), then the error energy is

$$\|e_N(x, y)\|^2 = \|g\|^2 - \sum_{k=-N}^N \mathcal{E}_k |c_k|^2 = \sum_{|n|>N} \mathcal{E}_k |c_k|^2.$$

This shows that the energy of the approximation error goes to zero, *i.e.*, the finite series converges to the true image $g(x, y)$, iff

$$\lim_{N \rightarrow \infty} \sum_{k=-N}^N \mathcal{E}_k |c_k|^2 = \|g\|^2.$$

It follows that the above is a necessary and sufficient condition for a set of orthogonal functions to be complete. Unfortunately, neither this condition, nor any other condition for completeness, is particularly easy to check. (See [1, p. 61] for an example of proving completeness of polynomials over intervals in 1D.)

Finally, we note that when we have an orthogonal series representation as in (2.6), the sense in which equality holds is that the difference between $g(x, y)$ and the summation while not necessarily zero for all (x, y) has zero energy. This means they can differ on a finite or countably infinite set of points, or even on a “set of measure zero”. Such differences are, generally speaking, unobservable. For example, the output of any LSI system will not be influenced by such differences.

¹ The equality in (2.6) is not quite the usual equality, but rather that the two sides are equivalent in the L_2 sense, *i.e.*, that the norm of the difference of the two sides is zero.

Parseval's theorem

The relationship that we know as **Parseval's theorem** for Fourier series (see (2.14)) also holds for general orthogonal expansions. If $g = \sum_{k=-\infty}^{\infty} c_k \phi_k$ and $h = \sum_{k=-\infty}^{\infty} d_k \phi_k$ then by the bilinearity of inner products (2.2):

$$\langle g, h \rangle = \left\langle \sum_{k=-\infty}^{\infty} c_k \phi_k, \sum_{l=-\infty}^{\infty} d_l \phi_l \right\rangle = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} c_k d_l^* \langle \phi_k, \phi_l \rangle = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} c_k d_l^* \mathcal{E}_k \delta[k-l] = \sum_{k=-\infty}^{\infty} \mathcal{E}_k c_k d_k^*,$$

using the orthogonality of the ϕ_k functions. (A rigorous proof requires more care with the infinite sums.) In other words:

$$\iint_{\mathcal{D}} g(x, y) h^*(x, y) dx dy = \sum_{k=-\infty}^{\infty} \mathcal{E}_k c_k d_k^*. \quad (2.7)$$

Taking $h = g$ as a special case yields **Rayleigh's theorem**:

$$\|g\|^2 = \iint_{\mathcal{D}} |g(x, y)|^2 dx dy = \sum_{k=-\infty}^{\infty} \mathcal{E}_k |c_k|^2. \quad (2.8)$$

In the engineering literature, both of these equalities are often called **Parseval's theorem** [2] [wiki], even for non-Fourier (orthogonal) bases. However, in mathematics the general version (2.8) is called the **Plancherel theorem** [3] [wiki].

Gram-Schmidt _____ (skim)

We can orthogonalize any set of signals by applying the **Gram-Schmidt** orthogonalization procedure. The procedure for signals is completely analogous to that for vectors, except that we use the inner products and norms that are appropriate for signals.

Separable orthogonal bases in 2D _____ (important!)

There are many well known orthogonal sets of 1D signals, such as harmonic complex exponentials, the Haar basis, etc. We can easily construct sets of 2D orthogonal signals from such 1D sets.

Suppose $\{\phi_k\}$ is an orthogonal set of 1D signals over the interval $[a, b]$. A simple approach to constructing orthogonal sets of 2D signals uses separability:

$$\phi_{k,l}(x, y) = \phi_k(x)\phi_l(y). \quad (2.9)$$

Over what set \mathcal{D} is this set of functions $\{\phi_{k,l}\}$ orthogonal? **??** [RQ]

Separable 2D signals are simple and convenient, but may not always be the best choice for a given application.

Example. Harmonic complex exponentials are separable, because $\phi_{k,l}(x, y) = e^{i2\pi(kx/T_X + ly/T_Y)} = e^{i2\pi kx/T_X} e^{i2\pi ly/T_Y}$.

Fact. If $\{\phi_k(x)\}$ is an orthonormal basis over $[a, b]$ and $\{\psi_l(y)\}$ is an orthonormal basis over $[c, d]$, then $\{\phi_k(x)\psi_l(y)\}$ is an orthonormal basis over $[a, b] \times [c, d]$ [4, p. 50], where “ \times ” denotes the Cartesian product for sets [wiki].

So separability is quite useful in designing 2D bases.

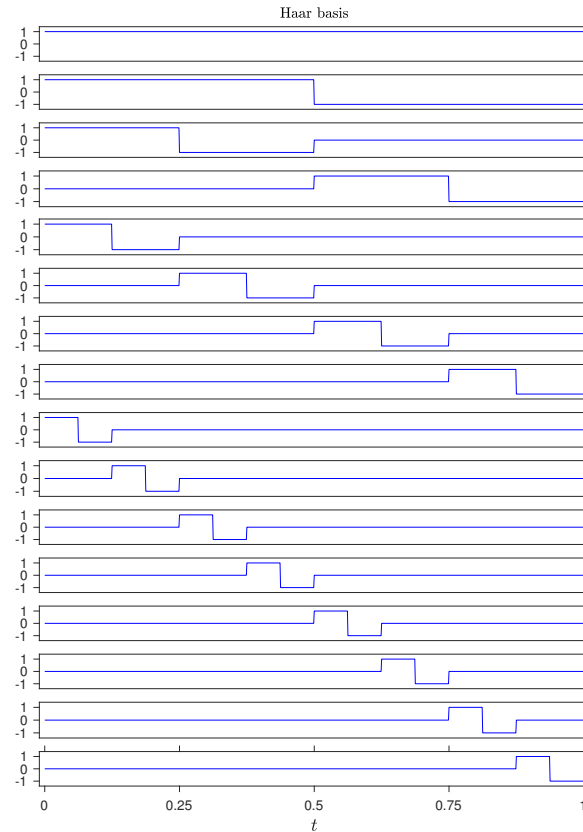
Uses of orthogonal representations _____

We use orthogonal representations for at least three distinct classes of signals, as follows.

- Finite support signals, where $g(x, y) = 0$ except for $(x, y) \in \mathcal{D}$
- Periodic signals. Apply representation over one period. If the representation is accurate over one period and the ϕ_k functions have the same period, then the representation is equally good over all periods.
- Finite energy signals with infinite support.

Less frequently we use orthogonal representations for finite energy signals with infinite support.

Example: The **Haar basis** is a set of 1D signals that is orthogonal and complete on the interval (0,1) and that consists of pieces of square waves as illustrated below:

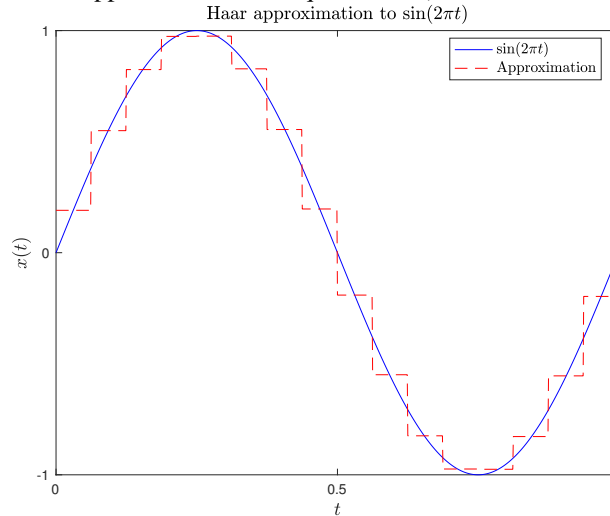


Each signal takes values 1, -1, and 0.

Due to its completeness, any (real, finite-energy) signal can be approximated arbitrarily well (in terms of the \mathcal{L}_2 norm) using the Haar basis with a sufficiently large number of terms N .

Example. Here is a Haar approximation to $x(t) = \sin(2\pi t)$.

(Contrast this with the usual Fourier series approximation to a square wave.)



These plots used a discretized approximation to the integral (2.5) to compute the c_k values, *i.e.*, for large N :

$$c_k = \frac{\langle x, \phi_k \rangle}{\|\phi_k\|^2} = \frac{\int_0^1 x(t) \phi_k^*(t) dt}{\int_0^1 |\phi_k^*(t)|^2 dt} \approx \frac{\frac{1}{N} \sum_{i=0}^{N-1} x(i/N) \phi_k^*(i/N)}{\frac{1}{N} \sum_{i=0}^{N-1} |\phi_k(i/N)|^2}.$$

A natural corresponding 2D basis consists of the separable form (2.9). However, 2D discrete Haar wavelets use a different 2D basis; see Ch. 13.

There are many possible complete sets of orthogonal signals, all of which can approximate all finite-energy signals to arbitrary accuracy. One large family is called **wavelets**, of which the Haar basis is a member.

Why do traditional signal processing texts focus on the Fourier series rather than the Haar basis?

?? ; ?? ; ??

Eigenfunctions of LSI systems

Of all the many orthogonal bases, the set of harmonic complex exponentials is particularly important.

The reason is simple: complex exponentials are **eigenfunctions** of LSI systems as shown by the following:

$$\underbrace{e^{i2\pi(\nu_x x + \nu_y y)}}_{\text{in}} \rightarrow \boxed{\text{LSI } h(x, y)} \rightarrow g(x, y) = e^{i2\pi(\nu_x x + \nu_y y)} ** h(x, y)$$

$$= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{i2\pi[(x-x')\nu_x + (y-y')\nu_y]} h(x', y') dx' dy' = \underbrace{e^{i2\pi(\nu_x x + \nu_y y)}}_{\text{out}} H(\nu_x, \nu_y)$$

where $H(\nu_x, \nu_y)$ is the **2D Fourier transform** of $h(x, y)$:

$$H(\nu_x, \nu_y) \triangleq \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x, y) e^{-i2\pi(\nu_x x + \nu_y y)} dx dy. \quad (2.10)$$

So the response due to a complex exponential input signal is just that same complex exponential input signal scaled by $H(\nu_x, \nu_y)$, which is why the complex exponentials are called eigenfunctions of LSI systems.

So if we can decompose a signal into complex exponential components, it is easy to determine the output of an LSI system by applying superposition.

(skip)

Are there other finite-power signals besides complex exponentials that are eigenfunction of *all* LSI systems?

No! Consider $g(x) = \int_{-\infty}^x f(x') dx'$ so $\frac{d}{dx}g(x) = f(x)$. Thus an eigenfunction of this system must satisfy $\lambda \frac{d}{dx}f(x) = f(x)$, so $f(x)$ must be of the form $f(x) = e^{x/\lambda}$. Because we are considering only f with finite power, λ must be purely imaginary.

Thus, complex exponentials are the only eigenfunctions for this system.

Challenge. Is there an example involving a **stable** system?

Fourier series (the classical 2D case developed by Jean-Baptiste Joseph Fourier [5] [wiki])

If $g(x, y)$ is periodic with period (T_x, T_y) , then we can represent $g(x, y)$ using a 2D **Fourier series (2D FS)**:

$$g(x, y) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} c_{k,l} e^{i2\pi(xk/T_x + yl/T_y)} \quad (\text{synthesis}) \quad (2.11)$$

$$c_{k,l} = \frac{1}{T_x T_y} \int_0^{T_y} \int_0^{T_x} g(x, y) e^{-i2\pi(xk/T_x + yl/T_y)} dx dy. \quad (\text{analysis}) \quad (2.12)$$

This series uses basis functions $\{\phi_{k,l}\}$ with $\phi_{k,l} = e^{i2\pi(xk/T_x + yl/T_y)}$. These are orthogonal and complete on any support rectangle whose width is a multiple of T_x and height is a multiple of T_y . Because the formula for the coefficients $c_{k,l}$ does not depend on which rectangle we choose, the Fourier series expansion applies to all such rectangles, *i.e.*, it applies to all x, y . (Note that the summation formula in (2.11) is periodic with period (T_x, T_y) . Note also that the integration in (2.12) can be taken over any rectangle of width T_x and height T_y . Finally, notice that each $\phi_{k,l}$ has energy $T_x T_y$ over any $T_x \times T_y$ rectangle. Therefore, they each have unity power.) What are the units of the FS coefficients $c_{k,l}$ if $g(x, y)$ is unitless? ?? [RQ]

One of the most important applications of the Fourier series is in the context of deriving sampling theorems.

Convergence _____ **(skim)**

- Because the harmonic complex exponentials are **complete**, equality in (2.11) holds in the sense that the difference between the left and right-hand sides has zero energy. Equivalently, the error in the difference between $g(x, y)$ and the summation of terms from $-N$ to N converges to zero as $N \rightarrow \infty$, i.e., $\|g - g_N\| \rightarrow 0$ as $N \rightarrow \infty$ for any square integrable g , where we define the finite sum:

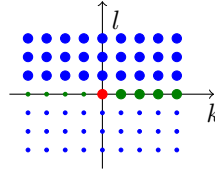
$$g_N(x, y) \triangleq \sum_{k=-N}^N \sum_{l=-N}^N c_{k,l} e^{j2\pi(xk/T_X + yl/T_Y)}.$$

- As in the 1D case, if $g(x, y)$ satisfies additional conditions, then the right-hand side of (2.11) equals $g(x, y)$ at all points x, y where $g(x, y)$ is continuous.
- Certainly if $g(x, y)$ is separable, then all convergence properties are inherited directly from the 1D case.
- The **Gibbs phenomena** [wiki] will also be present for signals with discontinuities.
- If $g(x, y)$ has continuous 1st and 2nd derivatives, then the 2D FS is pointwise [wiki] and uniformly convergent. This means that $g_N(x, y) \rightarrow g(x, y)$ for any x, y as $N \rightarrow \infty$. (Not true for discontinuous images!)

Properties

- If $g(x, y)$ is real, then the coefficients are **Hermitian symmetric**: $c_{-k, -l} = c_{k, l}^*$, and we can write:

$$g(x, y) = c_{00} + \sum_{k=1}^{\infty} 2|c_{k,0}| \cos(2\pi(k/T_x)x + \angle c_{k,0}) + \sum_{k=-\infty}^{\infty} \sum_{l=1}^{\infty} 2|c_{k,l}| \cos(2\pi[xk/T_x + yl/T_y] + \angle c_{k,l}). \quad (2.13)$$



- The other usual 1D properties (time shift, conjugation, etc.) also all carry over directly.

Example. If $g(x, y)$ has 2D FS coefficients $c_{k,l}$, then $g(x - x_0, y - y_0)$ has 2D FS coefficients $c_{k,l} e^{-i2\pi(x_0k/T_x + y_0l/T_y)}$.

- If $g(x, y)$ is **separable**, i.e., $g(x, y) = g_x(x)g_y(y)$, where $g_x(x)$ has period T_x and $g_y(y)$ has period T_y , then $c_{k,l}$ is the product of the 1D Fourier series coefficients of $g_x(x)$ and $g_y(y)$, i.e.,

$$c_{k,l} = \left[\frac{1}{T_x} \int_0^{T_x} g_x(x) e^{-i2\pi xk/T_x} dx \right] \left[\frac{1}{T_y} \int_0^{T_y} g_y(y) e^{-i2\pi yl/T_y} dy \right].$$

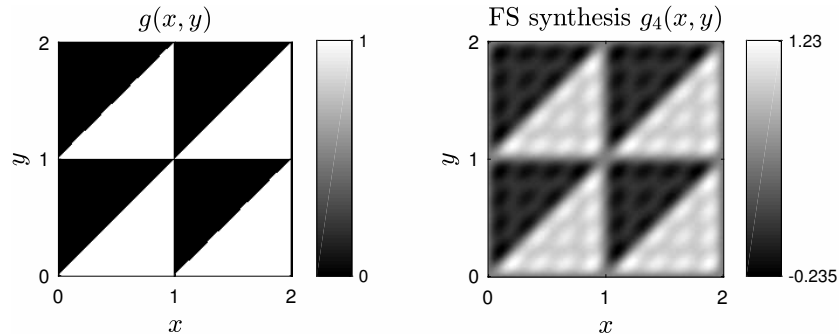
- Parseval's theorem** [2] [wiki] and **Rayleigh's theorem** are special cases of (2.7) and (2.8), due to the orthogonality. If $g(x, y)$ has 2D FS coefficients $\{c_{k,l}\}$ and $h(x, y)$ has 2D FS coefficients $\{d_{k,l}\}$ then

$$\frac{1}{T_x T_y} \int_{T_y} \int_{T_x} g(x, y) h^*(x, y) dx dy = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} c_{k,l} d_{k,l}^*,$$

$$\frac{1}{T_x T_y} \int_{T_y} \int_{T_x} |g(x, y)|^2 dx dy = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} |c_{k,l}|^2. \quad (2.14)$$

In words, the **power** in the space domain equals the sum of the powers of each frequency component.

Example. Consider the 2D signal $g(x, y)$ shown below left, defined to be unity where white, and zero where black.

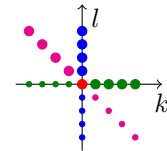


Is this signal separable? ?? What is c_{00} ? ?? What are T_x and T_y ? ??

[RQ]

We calculate the coefficients using (2.12):

$$\begin{aligned}
 c_{k,l} &= \int_0^1 \int_0^x e^{-i2\pi(kx+ly)} dy dx = \begin{cases} \int_0^1 x e^{-i2\pi kx} dx, & l = 0 \\ \int_0^1 \frac{1}{-i2\pi l} e^{-i2\pi(kx+ly)} \Big|_{y=0}^x dx, & l \neq 0 \end{cases} \\
 &= \begin{cases} 1/2, & k = l = 0 \\ \frac{i}{k2\pi}, & l = 0, k \neq 0 \\ \int_0^1 \frac{1}{i2\pi l} [e^{-i2\pi kx} - e^{-i2\pi(k+l)x}] dx, & l \neq 0 \end{cases} = \begin{cases} 1/2, & k = l = 0 \\ \frac{i}{k2\pi}, & l = 0, k \neq 0 \\ \frac{-i}{l2\pi}, & k = 0, l \neq 0 \\ \frac{i}{l2\pi}, & k = -l \neq 0 \\ 0, & \text{otherwise.} \end{cases}
 \end{aligned}$$



So most of the coefficients are zero, except where $l = 0$ or $k = 0$ or $k = -l$.

Are the coefficients separable? ??

[RQ]

Thus by (2.11) or (2.13):

$$g(x, y) = \frac{1}{2} + \sum_{k=1}^{\infty} \frac{1}{k\pi} [-\sin(2\pi kx) + \sin(2\pi ky) + \sin(2\pi k(x - y))].$$

The right figure above shows a FS approximation to this signal (with $1 + 4 \times 3 = 13$ terms), as generated by the following code.

Note that this code uses arrays to “vectorize” over x and y , but uses a loop over the Fourier series coefficients. In this example a single loop sufficed, but typically we will need a double loop akin to (2.12) or (2.13).

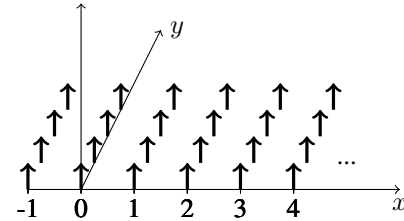
```
% fig_fs1.m Illustrate 2D FS
x = linspace(0,2,201); y = x; tx = 1; ty = 1;
[xx yy] = ndgrid(x,y);
xm = mod(xx,tx); ym = mod(yy,ty); % handy trick for periodic functions
gxy = ym < xm; % think about this trick!

g = 1/2 * ones(size(xx)); % DC term
for k=1:4
    g = g + 1/(k*pi) * (-sin(2*pi*k*xx) + sin(2*pi*k*yy) + sin(2*pi*k*(xx-yy)));
end

clf, subplot(221), imagesc(x,y,gxy'), axis xy, axis square, cbar
subplot(222), imagesc(x,y,g'), axis xy, axis square, colormap gray, cbar
```

Example. Consider the 2D comb (“bed of nails”) defined in (1.18):

$$g(x, y) = \text{comb}_2\left(\frac{x}{T_X}, \frac{y}{T_Y}\right) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} T_X T_Y \delta_2(x - mT_X, y - nT_Y).$$



Using the analysis formula (2.12), this $g(x, y)$ has FS coefficients $c_{k,l} = 1$, so we use the synthesis formula to write:

$$\text{comb}_2\left(\frac{x}{T_X}, \frac{y}{T_Y}\right) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} e^{i2\pi(xk/T_X + yl/T_Y)}. \quad (2.15)$$

(However, this $g(x, y)$ is not square integrable, so the FS does not converge so the above equality is just an engineering convention.)

This is a 2D generalization of the 1D Fourier series:

$$\text{comb}\left(\frac{x}{T_X}\right) = \sum_{n=-\infty}^{\infty} T_X \delta(x - nT_X) = \sum_{k=-\infty}^{\infty} e^{i2\pi xk/T_X}.$$

These “equalities” can be useful in deriving sampling theorems.

Note that because the 2D comb $\text{comb}_2\left(\frac{x}{T_X}, \frac{y}{T_Y}\right) = \text{comb}\left(\frac{x}{T_X}\right) \text{comb}\left(\frac{y}{T_Y}\right)$ is separable, so is its Fourier series:

$$\sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} e^{i2\pi(xk/T_X + yl/T_Y)} = \left[\sum_{k=-\infty}^{\infty} e^{i2\pi xk/T_X} \right] \left[\sum_{l=-\infty}^{\infty} e^{i2\pi yl/T_Y} \right].$$

Fourier series for finite support functions (skim)

The preceding pages describe how to expand a *periodic* function using Fourier series. Using the above analysis, it is easy to see that one can also use Fourier series to describe a **finite support** function. Indeed, mathematical treatments of Fourier series usually focus on finite support cases.

If $g(x, y)$ is a 2D function with support $[a, a + T_x] \times [b, b + T_y]$, then we can represent $g(x, y)$ using a 2D Fourier series as follows:

$$g(x, y) = \begin{cases} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} c_{k,l} e^{i2\pi(xk/T_x + yl/T_y)}, & a \leq x \leq a + T_x \text{ \& } b \leq y \leq b + T_y \\ 0, & \text{otherwise} \end{cases}$$

$$c_{k,l} = \frac{1}{T_x T_y} \int_a^{a+T_x} \int_b^{b+T_y} g(x, y) e^{-i2\pi(xk/T_x + yl/T_y)} dx dy.$$

Notice that the above synthesis expression is almost identical to (2.11) except for the braces to limit the support, and the analysis expression is similar to (2.12) except that the region of integration covers the support.

If we were given only the Fourier series coefficients $\{c_{k,l}\}$ and asked to synthesize $g(x, y)$, we would need to ask for additional (support) information to determine whether to use (2.11) or the above version with braces. Of course the two formula are identical over the support, so the distinction between the two is important only if one might evaluate the synthesis expression for arguments outside the support.

This discussion is a preview of the **discrete Fourier series (DFS)** and the **discrete Fourier transform (DFT)**. Those two transforms also involve synthesis formulae that are identical except for support considerations.

2.2 Fourier transforms

There are many transforms that are useful for **image analysis**, such as wavelets and cosine transforms, but of all of these, the 2D **Fourier transform** is particularly important for analyzing **imaging systems**. Why?

- **Reason 1: Convolution property**

A shift-invariant PSF reduces the entire system equation to a convolution integral. Such a system is easier to analyze in the Fourier domain, because convolutions in the space domain become simply multiplications in the Fourier domain. That is, the transform of the output function is the transform of the input function multiplied by the system transfer function H , the transform of the PSF.

The preceding concept is the most important property of Fourier transforms for our purposes.

- **Reason 2: Eigenfunctions of LSI systems**

As shown previously, complex exponential signals are the eigenfunctions of LSI systems, and the eigenvalues corresponding to those eigenfunctions are exactly the values of the FT of the impulse response at the frequency of the signal:

$$e^{i2\pi(\nu_x x + \nu_y y)} \rightarrow \boxed{\text{LSI } h(x, y)} \rightarrow H(\nu_x, \nu_y) e^{i2\pi(\nu_x x + \nu_y y)},$$

where $h(x, y) \xleftrightarrow{\mathcal{F}_2} H(\nu_x, \nu_y)$. (See derivation preceding (2.10).) The fact that the FT formula (2.10) fell out of the convolution with the complex exponential signal is by itself a compelling motivation to study that integral further.

The above two reasons are why Fourier methods are so important for analyzing LSI systems.

Two-dimensional Fourier transform

The continuous-space 2D **Fourier transform (2D FT)** of a function $g(x, y)$ is another function $G(\nu_x, \nu_y)$ defined as

$$G(\nu_x, \nu_y) = (\mathcal{F}_2[g])(\nu_x, \nu_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y) e^{-i2\pi(\nu_x x + \nu_y y)} dx dy. \quad (\text{analysis equation})$$

The **inverse Fourier transform** of $G(\nu_x, \nu_y)$ is defined by

$$g(x, y) = (\mathcal{F}_2^{-1}[G])(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} G(\nu_x, \nu_y) e^{i2\pi(\nu_x x + \nu_y y)} d\nu_x d\nu_y. \quad (\text{synthesis equation})$$

Like the Fourier series, the above expression is a decomposition of $g(x, y)$ into complex exponentials, except that now there are a continuum of exponentials, combined with an integral, rather than a sum.

Other notation:

$$\begin{aligned} G &= \mathcal{F}_2[g], & G(\nu_x, \nu_y) &= (\mathcal{F}_2[g])(\nu_x, \nu_y), \\ g &= \mathcal{F}_2^{-1}[G], & g(x, y) &= (\mathcal{F}_2^{-1}[G])(x, y), \\ & & g(x, y) &\stackrel{\mathcal{F}_2}{\longleftrightarrow} G(\nu_x, \nu_y). \end{aligned}$$

- We refer to ν_x and ν_y as **spatial frequencies** by analogy to 1D transforms where $\omega = 2\pi f$ where f is frequency.
- What are the units of ν_x and ν_y ? Hint: the units of x and y are distances, for example centimeters.

??

- Physically, each component $G(\nu_x, \nu_y)$ can be considered that component of the function $g(x, y)$ resulting from a plane wave of wavevector $K = 2\pi\sqrt{\nu_x^2 + \nu_y^2}$ and propagating along direction θ , where $\theta = \tan^{-1}(\nu_y/\nu_x)$.
- What are the units of $G(\nu_x, \nu_y)$? ?? [RQ]
- For sets of sufficient conditions under which the Fourier transform exists see [wiki] and the **Dirichlet conditions** [wiki].
- For a rigorous proof of the inverse Fourier transform, see [wiki].
- One can motivate the 2D FT from the Fourier series by letting the period(s) go to infinity, in analogy to the usual 1D presentation.

Relation to 1D Fourier transform

One can think of the 2D Fourier transform as resulting from first applying the 1D Fourier transform to $g(x, y)$ with x as the independent variable and y as a constant, to obtain a function of ν_x and y , and then applying the 1D Fourier transform to this function with y as the independent variable and ν_x as a constant. That is,

$$G = \mathcal{F}_{1D,y} \{ \mathcal{F}_{1D,x} \{ g \} \}$$

$$G(\nu_x, \nu_y) = \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} g(x, y) e^{-i2\pi\nu_x x} dx \right] e^{-i2\pi\nu_y y} dy.$$

where $\mathcal{F}_{1D,x}$ and $\mathcal{F}_{1D,y}$ denote the 1D Fourier transforms with respect to x and y , respectively. This two-stage approach is useful for numerical evaluation; see see Ch. 7.

Relation to Fourier series (skim)

One can motivate the 2D FT from the Fourier series by letting the period(s) go to infinity, in analogy to the usual 1D presentation.

Here is a pseudo-proof that $\mathcal{F}_2^{-1}[\mathcal{F}_2[g]] = g$. (For a rigorous proof, see [\[wiki\]](#).) If $G = \mathcal{F}_2[g]$ and g is continuous at (x, y) , then:

$$\begin{aligned} (\mathcal{F}_2^{-1}[\mathcal{F}_2[g]])(x, y) &= \lim_{T \rightarrow \infty} \int_{-T/2}^{T/2} \int_{-T/2}^{T/2} G(\nu_x, \nu_y) e^{i2\pi(\nu_x x + \nu_y y)} d\nu_x d\nu_y \\ &= \lim_{T \rightarrow \infty} \int_{-T/2}^{T/2} \int_{-T/2}^{T/2} \left[\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x', y') e^{-i2\pi(x'\nu_x + y'\nu_y)} dx' dy' \right] e^{i2\pi(\nu_x x + \nu_y y)} d\nu_x d\nu_y \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x', y') \left[\lim_{T \rightarrow \infty} \int_{-T/2}^{T/2} \int_{-T/2}^{T/2} e^{i2\pi[(x-x')\nu_x + (y-y')\nu_y]} d\nu_x d\nu_y \right] dx' dy' \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x', y') \left[\lim_{T \rightarrow \infty} T^2 \text{sinc}_2(T(x-x'), T(y-y')) \right] dx' dy' \quad (\text{because } \text{rect} \xrightarrow{\mathcal{F}} \text{sinc}; \text{ see 2.33}) \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x', y') \delta_2(x-x', y-y') dx' dy' = g(x, y). \end{aligned}$$

(This is typical of how FT properties are shown, by exchanging order of integration, changes of variables, etc.) To make this rigorous, one must identify regularity conditions on $g(x, y)$ that allow the exchange of order of limits and integration.

This “proof” does not work at points where $g(x, y)$ is discontinuous. See subsequent pages.

Existence **(skim)**

An integral with infinite limits represents a shorthand for a limit of an integral with endpoints that go to infinity, as seen above. Anytime one works with such integrals, for a mathematically rigorous treatment one should pause and consider when the limit (e.g., $\lim_{T \rightarrow \infty}$ above) actually exists.

There are two common sets of sufficient conditions that ensure that the 2D Fourier transform of an image $g(x, y)$ exists and that the inverse transform relation also holds.

1. $g(x, y)$ has finite energy, *i.e.*, it is square integrable.

In this case, the synthesis equation says that $g(x, y)$ equals the synthesis integral in the sense that their difference has zero energy.

2. Each of the following holds (**Dirichlet conditions**) [\[wiki\]](#):

- $g(x, y)$ is absolutely integrable (over all of \mathbb{R}).
- $g(x, y)$ has only a finite number of discontinuities and a finite number of maxima and minima in any finite region.
- $g(x, y)$ has no infinite discontinuities.
- $g(x, y)$ is bounded

In this case, the Fourier transform exists and the synthesis integral holds exactly at all x, y except where there is a discontinuity in $g(x, y)$.

We will also apply \mathcal{F}_2 to functions that satisfy neither of these conditions, such as Dirac impulses.

Null functions _____ (a cautionary note)

Taking a FT of a function g and then taking the inverse FT may not give you back *exactly* the original function g . Consider the function

$$g(x, y) = \begin{cases} 1, & x = 0, y = 0 \\ 0, & \text{otherwise.} \end{cases}$$

Because this function (called a “null function”) has no “area,” $G(\nu_x, \nu_y) = 0$, so the inverse FT $\tilde{g} = \mathcal{F}_2^{-1}[G]$ is simply $\tilde{g} = 0$, which does not equal g exactly! However, g and \tilde{g} are equivalent in the $\mathcal{L}_2(\mathbb{R}^2)$ sense that $\|g - \tilde{g}\|^2 = \iint (g - \tilde{g})^2 = 0$, which is more than adequate for any practical problem.

If we restrict attention to *continuous* functions g (that are absolutely integrable), then it will be true that $g = \mathcal{F}_2^{-1}[\mathcal{F}_2[g]]$. Most physical functions are continuous, or at least do not have type of isolated points that the function g above has. Moreover, most physical systems are insensitive to the addition of a null function to its input. Therefore the above mathematical subtleties need not deter our use of transform methods.

In practice, we can safely restrict attention to functions g for which $g = \mathcal{F}_2^{-1}[\mathcal{F}_2[g]]$ or at least to functions such that $g = \mathcal{F}_2^{-1}[\mathcal{F}_2[g]] + n$, where n is a null function.

Lerch’s theorem [6]: if f and g have the same Fourier transform, then $f - g$ is a null function, i.e., $\int_{-\infty}^{\infty} |f(x) - g(x)| dx = 0$.

In 1D, when there is a (finite) discontinuity in the signal, taking the FT and then the inverse FT recovers the midpoint of the discontinuity. Mathematically: $(\mathcal{F}^{-1}[\mathcal{F}[g]])(x) = \lim_{\varepsilon \rightarrow 0} \frac{g(x+\varepsilon) + g(x-\varepsilon)}{2} = \frac{g(x^+) + g(x^-)}{2}$.

Example. Consider $g(x) = \text{rect}(x) \xrightarrow{\mathcal{F}} G(\nu) = \text{sinc}(\nu)$. If $h = \mathcal{F}^{-1}[G]$ then $h(x) = \begin{cases} 1, & |x| < 1/2 \\ 1/2, & x = \pm 1/2 \\ 0, & |x| > 1/2. \end{cases}$

Note that $h(x) - g(x)$ is a **null function**.

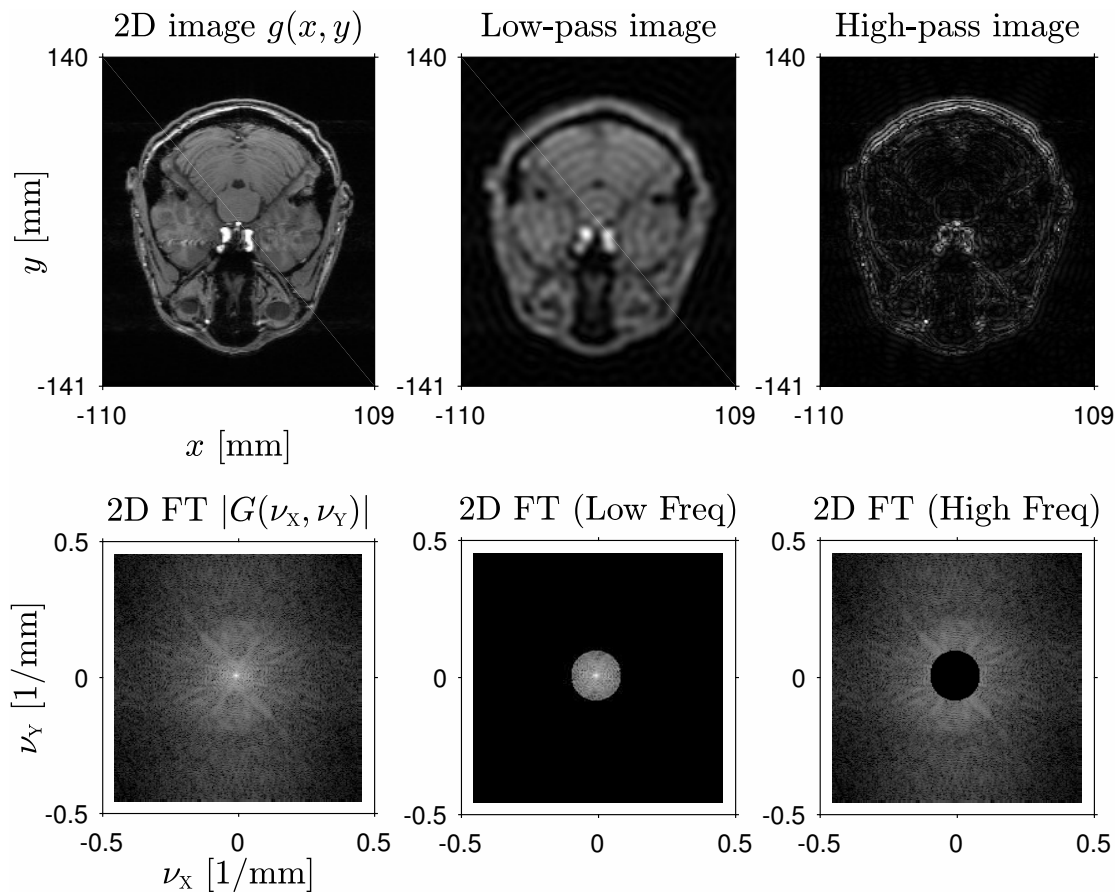
The corresponding result in 2D is the *angular average* [7, p. 438]:

$$(\mathcal{F}_2^{-1}[\mathcal{F}_2[g]])(x, y) = \lim_{\varepsilon \rightarrow 0} \frac{1}{2\pi} \int_0^{2\pi} g(x + \varepsilon \cos \phi, y + \varepsilon \sin \phi) d\phi.$$

Exercise. For $G(\nu_x, \nu_y) = \text{sinc}_2(\nu_x, \nu_y)$, determine $h = \mathcal{F}_2^{-1}[G]$. Hint. $h(x, y) = \begin{cases} 1, & |x| < 1/2 \text{ and } |y| < 1/2 \\ \boxed{??}, & |x| = 1/2 \text{ and } |y| = 1/2 \\ \boxed{??}, & \text{otherwise} \\ 0, & |x| > 1/2 \text{ or } |y| > 1/2. \end{cases}$

[RQ]

Example. Here is the 2D FT of an MR head image (do you recognize this person?) and the corresponding images synthesized from only the low spatial frequencies and from only the high spatial frequencies.



2D Fourier transform properties that match 1D FT properties

We use the notation $g(x, y) \xleftrightarrow{\mathcal{F}_2} G(\nu_x, \nu_y)$ to indicate that G is the FT of g , i.e., $G = \mathcal{F}_2[g]$.

Properties that are analogous to 1D FT properties

Linearity The 2D Fourier transform operation is linear:

$$\sum_k \alpha_k g_k(x, y) \xleftrightarrow{\mathcal{F}_2} \sum_k \alpha_k G_k(\nu_x, \nu_y).$$

Convolution

$$(g ** h)(x, y) \xleftrightarrow{\mathcal{F}_2} G(\nu_x, \nu_y) H(\nu_x, \nu_y)$$

The convolution of two images has a spectrum that is the product of the spectra of the two images. This is a very important relation for imaging.

Cross correlation [wiki] [8, p. 47]

$$(g \star \star h)(x, y) \triangleq \begin{aligned} & \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g^*(x', y') h(x' + x, y' + y) dx' dy' \\ & \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g^*(x' - x, y' - y) h(x', y') dx' dy' \end{aligned} \xleftrightarrow{\mathcal{F}_2} G^*(\nu_x, \nu_y) H(\nu_x, \nu_y).$$

The complex conjugate product of the spectra of two signals is the FT of their complex cross-correlation function.

Cross-correlation is used for object detection via matched filtering. Discrete-space implementation is MATLAB's `xcorr2` command.

An important special case is image **autocorrelation**:

$$(g \star \star g)(x, y) \triangleq \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g^*(x', y') g(x' + x, y' + y) dx' dy' \xleftrightarrow{\mathcal{F}_2} |G(\nu_x, \nu_y)|^2.$$

Although $|G(\nu_x, \nu_y)|^2$ is often called the **power spectral density**, the term **energy density spectrum** is more appropriate.

Magnification (scaling)

For $\alpha, \beta \neq 0$, $g(\alpha x, \beta y) \xleftrightarrow{\mathcal{F}_2} \boxed{??}$

Stretching in one domain proportionally contracts in the other domain, together with a constant weighting factor.

Shift (in space)

$$g(x - a, y - b) \xleftrightarrow{\mathcal{F}_2} G(\nu_x, \nu_y) e^{-i2\pi(\nu_x a + \nu_y b)}$$

Translation of a function in one space introduces a linear phase shift in the transform domain.

Is \mathcal{F}_2 a shift invariant operator? $\boxed{??}$

[RQ]

Derivative

$$\frac{d}{dx} g(x, y) \xleftrightarrow{\mathcal{F}_2} i2\pi\nu_x G(\nu_x, \nu_y)$$

DC value (historical term, not necessarily any “current” here.)

$$G(0, 0) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y) dx dy.$$

Multiplication (dual of convolution property)

$$g(x, y) h(x, y) \xleftrightarrow{\mathcal{F}_2} G(\nu_x, \nu_y) ** H(\nu_x, \nu_y).$$

Modulation (dual of shift property)

$$g(x, y) e^{i2\pi(u x + v y)} \xleftrightarrow{\mathcal{F}_2} G(\nu_x - u, \nu_y - v).$$

This property is at the heart of how **holography** works, and is important in deriving sampling theorems.

Self similarity

If $g(x, y) \xleftrightarrow{\mathcal{F}_2} G(\nu_x, \nu_y)$, then $g(x, y) + g(-x, -y) + G(x, y) + G(-x, -y)$ is its own Fourier transform!

Example: Gaussian, comb, derivative of Gaussian all have 2D Fourier transforms of the same functional form,

$$e.g., g(x, y) = e^{-\pi(x^2 + y^2)} \xleftrightarrow{\mathcal{F}_2} G(\nu_x, \nu_y) = e^{-\pi(\nu_x^2 + \nu_y^2)}.$$

Duality

If $g(x, y) \xleftrightarrow{\mathcal{F}_2} G(\nu_x, \nu_y)$, then $G^*(x, y) \xleftrightarrow{\mathcal{F}_2} g^*(\nu_x, \nu_y)$.

Parseval's Theorem or **Power Theorem**

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y) h^*(x, y) dx dy = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} G(\nu_x, \nu_y) H^*(\nu_x, \nu_y) d\nu_x d\nu_y$$

in particular: Rayleigh's theorem (1889):

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} |g(x, y)|^2 dx dy = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} |G(\nu_x, \nu_y)|^2 d\nu_x d\nu_y.$$

Hermitian symmetry properties**Conjugation**

$$g^*(x, y) \xleftrightarrow{\mathcal{F}_2} G^*(-\nu_x, -\nu_y).$$

Proof. $\iint g^*(x, y) e^{-i2\pi(\nu_x x + \nu_y y)} dx dy = [\iint g(x, y) e^{-i2\pi[(-\nu_x)x + (-\nu_y)y]} dx dy]^* = G^*(-\nu_x, -\nu_y)$.

Hermitian symmetry

If $g(-x, -y) = g^*(x, y)$, then $G(\nu_x, \nu_y)$ is real.

(This follows readily from the **conjugation property**.)

By duality, if $g(x, y)$ is real (which is usually the case), then $G(\nu_x, \nu_y)$ is Hermitian symmetric.

There are many other such relationships, such as: if $g(x, y) = g(-x, -y)$, *i.e.*, g is even, then $G(\nu_x, \nu_y)$ is even.

If g is odd, is $G(\nu_x, \nu_y)$ odd? ??

[RQ]

If g is Hermitian symmetric, is $G(\nu_x, \nu_y)$ Hermitian symmetric? ??

[RQ]

Example. One application of the Hermitian property is in “half k-space” MRI, where one acquires only half of the Fourier samples of an (assumed real) object, and “fills in” the other half using Hermitian symmetry [9, 10].

Generalized Fourier transforms

There are many important images for which we would like to use Fourier theory, even they are not square or absolutely integrable. These include constants, exponentials, sinusoids, and periodic functions. In addition, we would like to apply Fourier theory to the Dirac impulse and its cousin the “comb function.” In all such cases, the Dirac impulse and its defining properties play a principal role. Here are several examples of such Fourier transform pairs, most of which follow from the definition of the FT and its inverse and the sifting property of the Dirac impulse. One could make these pairs rigorous using the theory of **distributions**. [\[wiki\]](#)

Constant function

$$1 \xleftrightarrow{\mathcal{F}_2} \delta_2(\nu_x, \nu_y).$$

2D step function

$$\text{step}(x)\text{step}(y) \xleftrightarrow{\mathcal{F}_2} \left(\frac{1}{i2\pi\nu_x} + \delta(\nu_x) \right) \left(\frac{1}{i2\pi\nu_y} + \delta(\nu_y) \right)$$

2D Dirac impulse

$$\delta_2(x, y) \xleftrightarrow{\mathcal{F}_2} 1, \quad \delta_2(x - a, y - b) \xleftrightarrow{\mathcal{F}_2} e^{-i2\pi(\nu_x a + \nu_y b)}.$$

2D complex exponential

$$e^{i2\pi(xu_0 + yv_0)} \xleftrightarrow{\mathcal{F}_2} \delta_2(\nu_x - u_0, \nu_y - v_0).$$

2D sinusoid

$$\cos(2\pi[xu_0 + yv_0]) \xleftrightarrow{\mathcal{F}_2} \frac{1}{2} [\delta_2(\nu_x - u_0, \nu_y - v_0) + \delta_2(\nu_x + u_0, \nu_y + v_0)].$$

2D comb

Combining the 2D complex exponential above with linearity leads to:

$$\sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} e^{i2\pi(xk + yl)} \xleftrightarrow{\mathcal{F}_2} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \delta_2(\nu_x - k, \nu_y - l).$$

So using the comb Fourier series (2.15) and the comb definition (1.18) leads to the following curious (generalized) 2D FT pair:

$$\text{comb}_2(x, y) \xleftrightarrow{\mathcal{F}_2} \text{comb}_2(\nu_x, \nu_y).$$

2D Fourier transform and LSI systems

The convolution property of the 2D FT is particularly important for understanding 2D LSI systems.

The space-domain perspective is:

$$f(x, y) \rightarrow \boxed{\text{LSI } h(x, y)} \rightarrow g(x, y) = h(x, y) ** f(x, y)$$

The frequency-domain perspective is:

$$F(\nu_x, \nu_y) \rightarrow \boxed{\text{LSI } H(\nu_x, \nu_y)} \rightarrow G(\nu_x, \nu_y) = H(\nu_x, \nu_y) F(\nu_x, \nu_y)$$

where $h(x, y) \xleftrightarrow{\mathcal{F}_2} H(\nu_x, \nu_y)$.

We call $H(\nu_x, \nu_y)$ the **frequency response** or **transfer function** of the system.

OTF and MTF

(skim)

For LSI imaging systems, the **optical transfer function (OTF)** is defined as the normalized frequency response

$$\text{OTF}(\nu_x, \nu_y) \triangleq \frac{H(\nu_x, \nu_y)}{H(0, 0)},$$

and the **modulation transfer function (MTF)** is defined as its magnitude:

$$\text{MTF}(\nu_x, \nu_y) \triangleq |\text{OTF}(\nu_x, \nu_y)| = \frac{|H(\nu_x, \nu_y)|}{|H(0, 0)|}.$$

Why? See [11, p. 98].

Properties that are new for the 2D FT

The following properties of the 2D FT have no analog for the 1D FT. These are all important in imaging and image processing!

- **Rotation**

If $g(x, y) \xleftrightarrow{\mathcal{F}_2} G(\nu_x, \nu_y)$, then (see (1.22)):

$$g_\theta(x, y) \xleftrightarrow{\mathcal{F}_2} G_\theta(\nu_x, \nu_y) = G(\nu_x \cos \theta + \nu_y \sin \theta, -\nu_x \sin \theta + \nu_y \cos \theta). \quad (2.16)$$

- **Rotational symmetry**

If an image $g(x, y)$ has ***n*-fold rotational symmetry**, then so does its 2D FT $G(\nu_x, \nu_y)$. This is a trivial consequence of the preceding property.

- **Circular symmetry**

If $g(x, y) = g(r)$ is circularly symmetric, then so is its 2D FT: $G(\nu_x, \nu_y) = G(\rho)$, $\rho = \sqrt{\nu_x^2 + \nu_y^2}$.

- **Separability (Cartesian)**

Images that are (Cartesian) separable have (Cartesian) separable Fourier transforms:

$$g(x, y) = g_x(x)g_y(y) \xleftrightarrow{\mathcal{F}_2} G(\nu_x, \nu_y) = G_x(\nu_x)G_y(\nu_y),$$

where G_x and G_y are the 1D Fourier transforms of $g_x(x)$ and $g_y(y)$ respectively.

Separable functions are often used in filtering, where the filtering operation is a convolution such that

$$(g ** h)(x, y) = g(x, y) ** [h_x(x)h_y(y)] \xleftrightarrow{\mathcal{F}_2} G(\nu_x, \nu_y) H(\nu_x, \nu_y) = G(\nu_x, \nu_y) H_x(\nu_x)H_y(\nu_y).$$

With these properties in hand, especially separability, we can give some concrete examples by building on known 1D FT pairs.

2D FT Examples: Separable functions

Separable functions

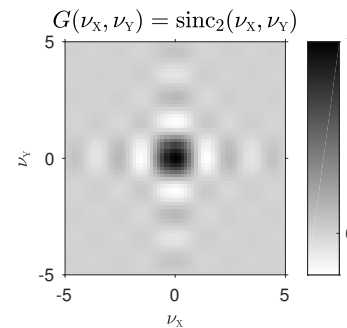
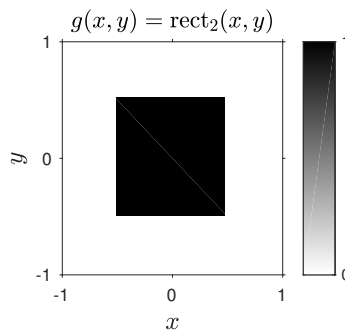
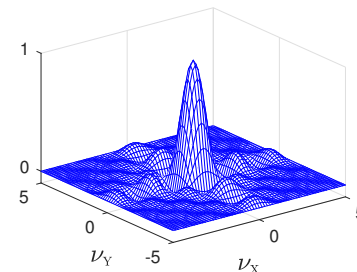
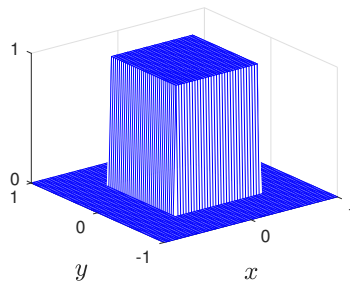
| Function | Spectrum (2D FT) |
|--|---|
| $\delta_2(x, y) = \frac{1}{\pi r} \delta(r)$ | 1 |
| 1 | $\delta_2(\nu_x, \nu_y)$ |
| $\text{comb}_2(x, y)$ | $\text{comb}_2(\nu_x, \nu_y)$ |
| $\text{rect}_2(x, y)$ | $\text{sinc}_2(\nu_x, \nu_y)$ |
| $e^{-\pi x^2} e^{-\pi y^2}$ | $e^{-\pi \nu_x^2} e^{-\pi \nu_y^2}$ |
| $e^{- x } e^{- y }$ | $\frac{2}{1 + (2\pi \nu_x)^2} \frac{2}{1 + (2\pi \nu_y)^2}$ |

$$\text{rect}_2(x, y) \triangleq \text{rect}(x) \text{rect}(y)$$

$$\text{rect}(t) \triangleq \begin{cases} 1, & |t| < 1/2 \\ 0, & \text{otherwise} \end{cases}$$

$$\text{sinc}_2(\nu_x, \nu_y) \triangleq \text{sinc}(\nu_x) \text{sinc}(\nu_y)$$

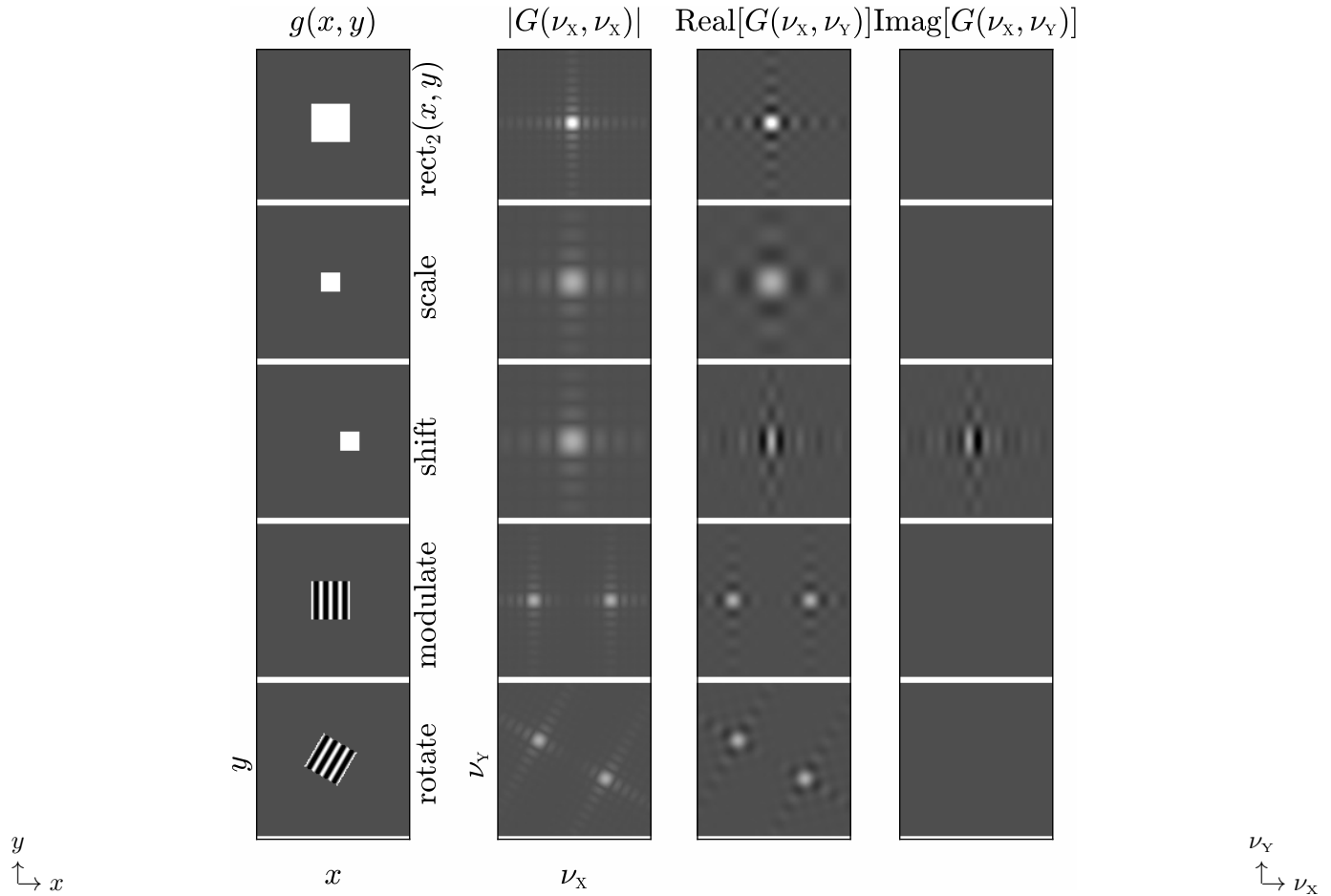
$$\text{sinc}(t) \triangleq \begin{cases} \frac{\sin \pi t}{\pi t}, & t \neq 0 \\ 1, & t = 0 \end{cases}$$



Caution. This definition of $\text{sinc}(t)$ may differ from your 1D signal processing class.

But this definition does match the `sinc` function in MATLAB's Signal Processing Toolbox and in GNU Octave.

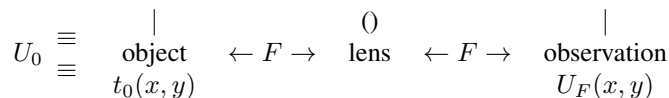
Example. The following figure illustrates some of the 2D FT properties.



Using lenses for 2D Fourier transforms

Usually a 2D Fourier transform is evaluated either analytically using tables and properties, or numerically using the 2D FFT (see Ch. 7). However, it is also possible to evaluate the 2D FT of a scene using a lens, and this property has been used in optical information processing systems.

Consider a lens having focal length F that is placed a distance F from a thin object (e.g., cells on a microscope slide) with transmittance $t_0(x, y)$ and distance F from a sensor, as illustrated below.



Let the object be illuminated by a monochromatic plane wave with wavelength λ , i.e., where the field amplitude $U_0(x, y) = U_0$ is a constant. Ignoring the finite extent of the lens, one can show [12, eqn. (5-19)] that the field amplitude at the observation plane is:

$$U_F(x, y) = \frac{U_0}{i\lambda F} T_0\left(\frac{x}{\lambda F}, \frac{y}{\lambda F}\right) = \frac{U_0}{i\lambda F} T_0(\nu_x, \nu_y) \Big|_{\nu_x = \frac{x}{\lambda F}, \nu_y = \frac{y}{\lambda F}}, \quad (2.17)$$

where $t_0 \xrightarrow{\mathcal{F}_2} T_0$. In other words, to within a largely irrelevant constant, the field at the observation plane is the 2DFT of the field at the object plane for this geometry.

Is this 2D imaging system shift invariant? ??

[RQ]

Example. For the following transmittance function, which corresponds to a pinhole in an otherwise absorbing screen:

$$t_0(x, y) = \delta_2(x - x_0, y - y_0) \xrightarrow{\mathcal{F}_2} T_0(\nu_x, \nu_y) = e^{-i2\pi(x_0\nu_x + y_0\nu_y)},$$

the field at the observation plane corresponds to a plane wave: $U_F(x, y) \propto e^{-ik(x x_0 + y y_0)/F}$.

An optical recorder at the observation plane will only record the incident *intensity*, $I(x, y) = |U_F(x, y)|^2$, so we only get to measure the *magnitude* of the Fourier transform of the object. To recover the object transmittance $t_0(x, y)$ from such measurements we also need to somehow determine (an estimate of) the phase of the FT. This is known as the **phase retrieval** problem [13].

For the preceding pinhole example, what is the recorded intensity $I(x, y)$? ??

[RQ]

Periodic signals: relating 2D FT and 2D Fourier series

One approach to finding the 2D FT of a periodic signal uses the 2D FS:

$$g(x, y) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} c_{k,l} e^{i2\pi(xk/T_X + yl/T_Y)} \xleftrightarrow{\mathcal{F}_2} G(\nu_X, \nu_Y) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} c_{k,l} \delta_2(\nu_X - k/T_X, \nu_Y - l/T_Y). \quad (2.18)$$

So the spectrum of a periodic signal is a kind of line spectrum that looks like a “bed of nails” of different “lengths.”

Often periodic signals arise as the superposition of shifted replicates of a basic pattern $f(x, y)$:

$$\begin{aligned} g(x, y) &= \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(x - mT_X, y - nT_Y) \\ &= f(x, y) ** \left[\sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \delta_2(x - mT_X, y - nT_Y) \right] = f(x, y) ** \left[\frac{1}{T_X} \frac{1}{T_Y} \text{comb}_2\left(\frac{x}{T_X}, \frac{y}{T_Y}\right) \right] \\ \xleftrightarrow{\mathcal{F}_2} G(\nu_X, \nu_Y) &= F(\nu_X, \nu_Y) \text{comb}_2(T_X\nu_X, T_Y\nu_Y) \quad (\text{now apply sampling property of Dirac impulse}) \\ &= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \frac{1}{T_X T_Y} F\left(\frac{k}{T_X}, \frac{l}{T_Y}\right) \delta_2\left(\nu_X - \frac{k}{T_X}, \nu_Y - \frac{l}{T_Y}\right). \end{aligned} \quad (2.20)$$

Such a “replicated pattern” signal is periodic with period (T_X, T_Y) .

Comparing the right hand side of (2.18) and (2.20), we see that the FS coefficients $\{c_{k,l}\}$ of $g(x, y)$ are simply samples of the FT of the “pattern” f :

$$\boxed{c_{k,l} = \frac{1}{T_X T_Y} F\left(\frac{k}{T_X}, \frac{l}{T_Y}\right)}. \quad (2.21)$$

This expression is usually the easiest way to find 2D FS coefficients of a periodic signal of the “replicated pattern” form (2.19).

Using (1.9), an alternative way to write (2.19) is

$$g(x, y) = f(\text{mod}(x + T_X/2, T_X) - T_X/2, \text{mod}(y + T_Y/2, T_Y) - T_Y/2).$$

For this form, the expression (2.21) is valid if $f(x, y)$ is supported on $[-T_X/2, T_X/2] \times [-T_Y/2, T_Y/2]$.

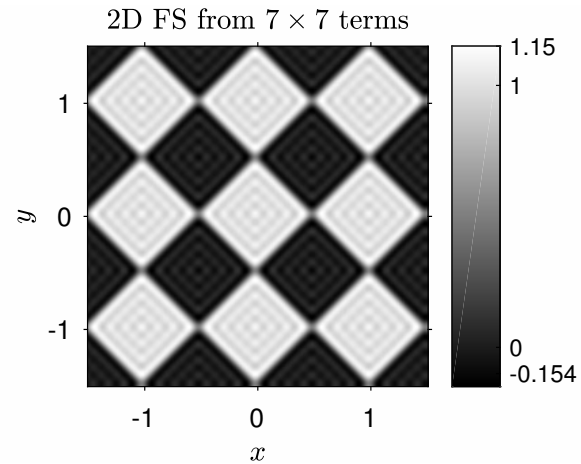
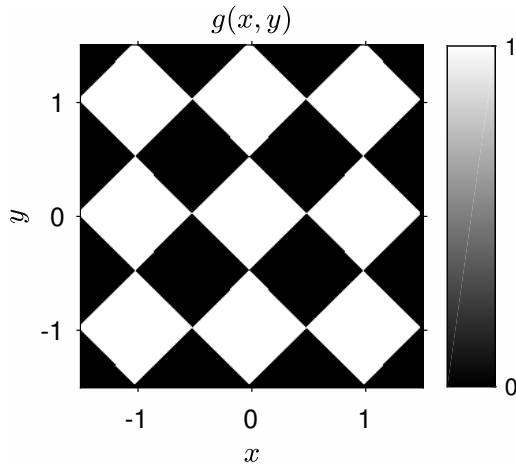
Example. A (rotated) “checker board” image can be written $g(x, y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(x - m, y - n)$ where for $\theta = \pi/4$:

$$f(x, y) = \text{rect}_2(x + y, x - y) = \text{rect}_2\left(\sqrt{2}\frac{x+y}{\sqrt{2}}, \sqrt{2}\frac{x-y}{\sqrt{2}}\right) = \text{rect}_2(\sqrt{2}(x \cos \theta + y \sin \theta), \sqrt{2}(-x \sin \theta + y \cos \theta)).$$

Using the scaling and rotation properties of the 2D FT:

$$F(\nu_x, \nu_y) = \frac{1}{\sqrt{2}\sqrt{2}} \text{sinc}_2\left(\frac{1}{\sqrt{2}}(\nu_x \cos \theta + \nu_y \sin \theta), \frac{1}{\sqrt{2}}(-\nu_x \sin \theta + \nu_y \cos \theta)\right) = \frac{1}{2} \text{sinc}_2\left(\frac{\nu_x + \nu_y}{2}, \frac{\nu_x - \nu_y}{2}\right),$$

so the 2D FS coefficients for this periodic signal are $c_{k,l} = \frac{1}{2} \text{sinc}_2\left(\frac{k+l}{2}, \frac{k-l}{2}\right)$.



What is c_{11} for this checker board image?

[RQ]

Hankel transform

We would also like a 2D FT property for **polar separable** images. For this topic we must first introduce the Hankel transform.

On [14, p. 360: 9.1.21] of Abramowitz and Stegun, the k th-order Bessel function (of the 1st kind) J_k is expressed:

$$J_k(z) = \frac{i^{-k}}{\pi} \int_0^\pi e^{iz \cos \phi} \cos(k\phi) d\phi = \frac{(-i)^{-k}}{2\pi} \int_0^{2\pi} e^{-iz \cos \phi} e^{ik\phi} d\phi = \frac{1}{2\pi} \int_0^{2\pi} e^{-iz \sin \phi} e^{-ik\phi} d\phi. \quad (2.22)$$

(Note that $J_{-k}(z) = (-1)^k J_k(z)$. And J_k solves $z^2 \ddot{f} + z\dot{f} + (z^2 - k^2)f = 0$.)

Bessel functions satisfy the following orthogonality property (called the “closure” property on Wikipedia): [\[wiki\]](#)

$$\int_0^\infty J_k(ar) J_k(br) r dr = \frac{1}{a} \delta(a - b), \quad \text{for } a \neq 0 \text{ and } b \neq 0. \quad (2.23)$$

The **k th-order Hankel transform** of the 1D function $g(r)$ is defined by²:

$$g(r) \xleftrightarrow{\mathcal{H}_k} G_k(\rho) = 2\pi \int_0^\infty g(r) J_k(2\pi r \rho) r dr. \quad (2.24)$$

The inverse k th-order Hankel transform is given by (assuming $g(r)$ is continuous at r) the remarkably similar expression:

$$g(r) = 2\pi \int_0^\infty G_k(\rho) J_k(2\pi r \rho) \rho d\rho \xleftrightarrow{\mathcal{H}_k} G_k(\rho).$$

To verify this inverse, we use the orthogonality property (2.23) above:

$$\begin{aligned} 2\pi \int_0^\infty G_k(\rho) J_k(2\pi r \rho) \rho d\rho &= 2\pi \int_0^\infty \left[2\pi \int_0^\infty g(r') J_k(2\pi r' \rho) r' dr' \right] J_k(2\pi r \rho) \rho d\rho \\ &= 2\pi \int_0^\infty g(r') \left[2\pi \int_0^\infty J_k(2\pi r' \rho) J_k(2\pi r \rho) \rho d\rho \right] r' dr' \\ &= 2\pi \int_0^\infty g(r') \left[2\pi \frac{1}{2\pi r} \delta(2\pi(r' - r)) \right] r' dr' = g(r). \end{aligned}$$

² Caution. As of Jan. 2016, Wikipedia uses a different definition without the 2π factors [\[wiki\]](#).

2D FT in polar coordinates

To understand the importance of Hankel transforms, we must first consider the 2D FT in polar coordinates.

We write the spectrum $F(\nu_x, \nu_y)$ of a 2D image $f(x, y)$ in polar coordinates as follows:

$$F_o(\rho, \phi) \triangleq F(\rho \cos \phi, \rho \sin \phi) = F(\nu_x, \nu_y) \Big|_{\nu_x = \rho \cos \phi, \nu_y = \rho \sin \phi}$$

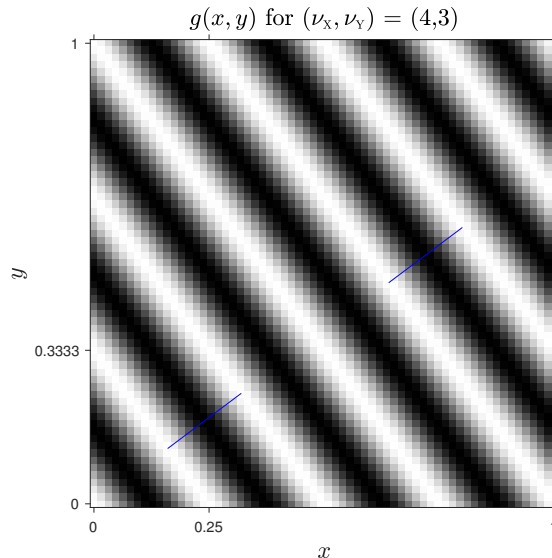
$$\rho = \sqrt{\nu_x^2 + \nu_y^2}, \quad \phi = \tan^{-1}(\nu_y/\nu_x), \quad \nu_x = \rho \cos \phi, \quad \nu_y = \rho \sin \phi.$$

What are the units of ρ ? **??**

[RQ]

The pair (ν_x, ν_y) is known as a **wave vector** and its magnitude ρ is also known as the **wavenumber**. [wiki]

Example. The following figure helps interpret 2D “plane waves” in polar coordinates.



Because $\nu_x = 4$ cycles/m in this example, the crests are spaced by $1/\nu_x = 1/4$ m along the x -axis.

Likewise $\nu_y = 3$ cycles/m and the crests are spaced by $1/\nu_y = 1/3$ m along the y -axis.

But the orientation of the two axes is somewhat arbitrary.

What is the “natural” orientation of the coordinate system here?

??

The spacing between the crests of the waves is $1/\rho$, shown by the blue line segments, *i.e.*, the wave oscillates at rate ρ cycles per unit distance along its direction of travel. The blue line segments make an angle ϕ with respect to the x axis.

What is ρ in this example? ??

[RQ]

2D Fourier synthesis in polar coordinates

Using a simple change of variables, we can rewrite the inverse 2D Fourier transform in terms of the spectrum in polar coordinates as follows:

$$\begin{aligned} f(x, y) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(\nu_x, \nu_y) e^{i2\pi(x\nu_x + y\nu_y)} d\nu_x d\nu_y \\ &= \int_0^{2\pi} \int_0^{\infty} F_o(\rho, \phi) e^{i2\pi(x\rho \cos \phi + y\rho \sin \phi)} \rho d\rho d\phi. \end{aligned}$$

Each “basis function” in this synthesis expression is a plane wave like that illustrated in the preceding figure.

Relationship between Hankel and 2D FT

The most important property of the k th-order Hankel transform is that it is related to the **2D FT** of an “angularly phase modulated” circularly symmetric function as follows:

$$\boxed{f(r, \theta) = g(r) e^{ik\theta} \xleftrightarrow{\mathcal{F}_2} F_o(\rho, \phi) = e^{ik\phi} (-i)^k G_k(\rho),} \quad (2.25)$$

where $g(r) \xleftrightarrow{\mathcal{H}_k} G_k(\rho)$.

Proof:

$$\begin{aligned} F_o(\rho, \phi) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-i2\pi(x\rho \cos \phi + y\rho \sin \phi)} dx dy &= \int_0^{\infty} \int_0^{2\pi} f(r, \theta) e^{-i2\pi r\rho(\cos \theta \cos \phi + \sin \theta \sin \phi)} r d\theta dr \\ &= \int_0^{\infty} g(r) \left[\int_0^{2\pi} e^{ik\theta} e^{-i2\pi r\rho \cos(\theta-\phi)} d\theta \right] r dr &= e^{ik\phi} \int_0^{\infty} g(r) \left[\int_0^{2\pi} e^{ik(\theta-\phi)} e^{-i2\pi r\rho \cos(\theta-\phi)} d\theta \right] r dr \\ &= e^{ik\phi} \int_0^{\infty} g(r) \left[\int_0^{2\pi} e^{ik\theta} e^{-i2\pi r\rho \cos(\theta)} d\theta \right] r dr &= e^{ik\phi} (-i)^k 2\pi \int_0^{\infty} g(r) J_k(2\pi r\rho) r dr = e^{ik\phi} (-i)^k G_k(\rho). \end{aligned}$$

A function having the form $f(r, \theta) = g(r) e^{ik\theta}$ is called **circularly harmonic**, and the result (2.25) shows that circularly harmonic functions have circularly harmonic 2D Fourier transforms.

Separability (Polar)

If $g(x, y)$ is **separable in polar coordinates**, i.e., if $g(r, \theta) = g_R(r)g_\Theta(\theta)$, where $g_\Theta(\theta)$ is 2π -periodic, then it is natural to express the 2D FT in polar coordinates (although in general the 2D FT is not quite separable in those coordinates).

Because $g_\Theta(\theta)$ is 2π -periodic, we use the 1D Fourier series to express it in terms of its angular harmonics:

$$g_\Theta(\theta) = \sum_{k=-\infty}^{\infty} c_k e^{ik\theta}, \quad \text{where } c_k = \frac{1}{2\pi} \int_0^{2\pi} g_\Theta(\theta) e^{-ik\theta} d\theta.$$

Thus a polar separable function g has the following 2D FT:

$$g(r, \theta) = \sum_{k=-\infty}^{\infty} c_k [g_R(r) e^{ik\theta}] \xrightarrow{\mathcal{F}_2} G(\rho, \phi) = \sum_{k=-\infty}^{\infty} c_k e^{ik\phi} (-i)^k G_k(\rho),$$

where $g_R(r) \xrightarrow{\mathcal{H}_k} G_k(\rho)$. This expression is the sum of polar separable functions, but is not itself polar separable.

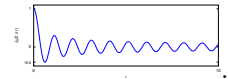
Circular symmetry

By far the most frequent use of polar separability is in problems where $g(x, y)$ has **circular symmetry**, so $g_\Theta(\theta) = 1$ and $g(x, y) = g(r)$. In this case only the $k = 0$ term above is nonzero, so the 2D FT is also circularly symmetric:

$$g(x, y) = g(r) \xrightarrow{\mathcal{F}_2} G(\rho, \phi) = G(\rho) = 2\pi \int_0^{\infty} g(r) J_0(2\pi\rho r) r dr,$$

where J_0 is the 0th-order Bessel function, cf (2.22), defined by [14, p. 360: 9.1.18]:

$$J_0(z) = \frac{1}{\pi} \int_0^{\pi} \cos(z \sin \phi) d\phi = \frac{1}{\pi} \int_0^{\pi} \cos(z \cos \phi) d\phi = \frac{1}{2\pi} \int_0^{2\pi} e^{\pm iz \cos \phi} d\phi = \frac{1}{2\pi} \int_0^{2\pi} e^{\pm iz \sin \phi} d\phi$$



Note that we wrote $g(r) \xrightarrow{\mathcal{F}_2} G(\rho)$, thinking 2D, but we could also write $g(r) \xrightarrow{\mathcal{H}_0} G(\rho)$, thinking 1D. Thus

$$\boxed{G(\rho) = 2\pi \int_0^{\infty} g(r) J_0(2\pi\rho r) r dr.} \quad \text{Similarly:} \quad \boxed{g(r) = 2\pi \int_0^{\infty} G(\rho) J_0(2\pi\rho r) \rho d\rho.} \quad (2.26)$$

This is called the **Hankel Transform** (of zero order). It is often called the **Fourier-Bessel transform** in lens/diffraction theory.

Example. The functions $g_\alpha(r) = 1/r^\alpha$ for $\alpha \in (0, 2)$ are important in tomography [15] [16, p. 414]. Their Hankel transforms are

$$G_\alpha(\rho) = 2\pi \int_0^\infty \frac{1}{r^\alpha} J_0(2\pi\rho r) r \, dr = 2\pi \int_0^\infty \frac{1}{(r'/\rho)^\alpha} J_0(2\pi r') \frac{r'}{\rho} \frac{dr'}{\rho} = \frac{c_\alpha}{\rho^{2-\alpha}},$$

where $r' = \rho r$ and $c_\alpha \triangleq 2\pi \int_0^\infty J_0(2\pi r) r^{1-\alpha} \, dr = \frac{\pi^\alpha}{\Gamma^2(\alpha/2) \sin(\pi\alpha/2)}$. Note that $c_1 = 1$ because $\Gamma(1/2) = \sqrt{\pi}$.

In MATLAB, the command `besselj(k, z)` evaluates $J_k(z)$.

Properties

Some, but not all, of the Fourier transform properties apply to the Hankel transform.

Duality _____ (for circularly symmetric functions)

$$g(r) \xleftrightarrow{\mathcal{F}_2} G(\rho) \iff G^*(r) \xleftrightarrow{\mathcal{F}_2} g^*(\rho).$$

Symmetry: Because J_0 is real-valued, in the usual case where $g(r)$ is real then $G(\rho)$ is real!

Scaling: What is the scaling property for the Hankel transform? $g(ar) \xleftrightarrow{\mathcal{F}_2} \boxed{??}$ [RQ]

List at least one other important property of the 2D FT that also holds for the Hankel transform. [RQ]

Can you think of any other basic properties of the 2D FT that hold for the Hankel transform?

??

There are **fast Hankel transform** methods [17] that facilitate the numerical use of (2.24).

Hankel transform in higher dimensions _____ (skim)

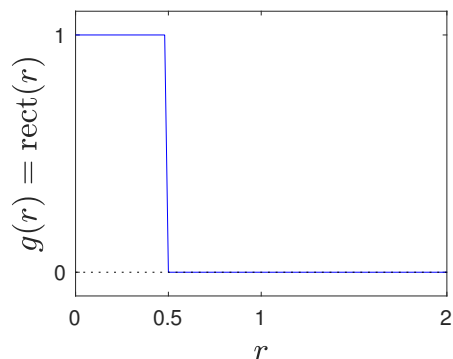
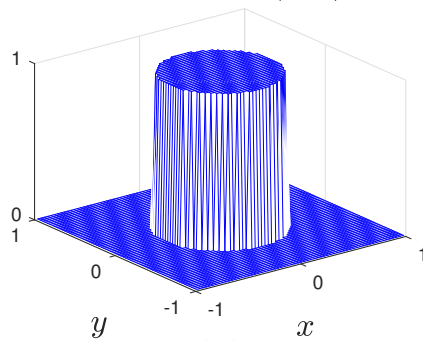
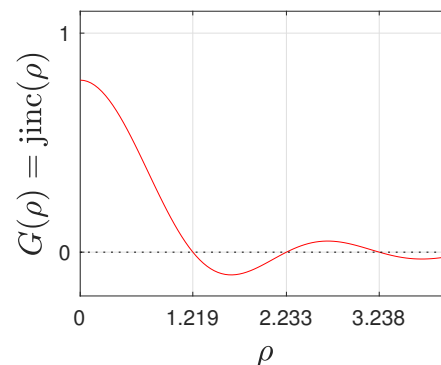
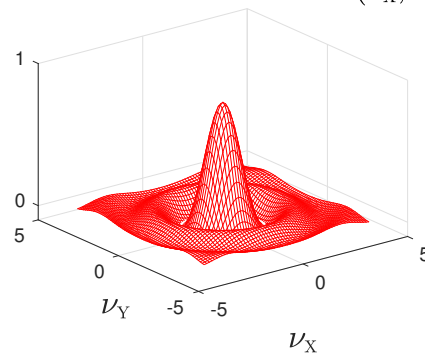
In dimensions $d > 2$, if a function has **spherical symmetry**, i.e., $f(x_1, x_2, \dots, x_d) = f_0(\sqrt{x_1^2 + x_2^2 + \dots + x_d^2})$, then so will its Fourier transform $F(\nu_1, \nu_2, \dots, \nu_d) = F_0(\sqrt{\nu_1^2 + \nu_2^2 + \dots + \nu_d^2})$, and [18, p. 1836]:

$$F_0(\rho) = \frac{2\pi}{\rho^{d/2-1}} \int_0^\infty f_0(r) J_{d/2-1}(2\pi\rho r) r^{d/2} \, dr.$$

Example. These are two particularly important Hankel transform pairs:

$$\text{Pillbox: } \text{rect}(r) \xleftrightarrow{\mathcal{F}_2} \text{jinc}(\rho) \triangleq \begin{cases} \frac{J_1(\pi\rho)}{2\rho}, & \rho \neq 0 \\ \pi/4, & \rho = 0, \end{cases} \quad \text{Gaussian: } e^{-\pi r^2} \xleftrightarrow{\mathcal{F}_2} e^{-\pi\rho^2}$$

1D Function

2D image: $g(x, y)$ Hankel Transform: $G(\rho)$ 2D Fourier Transform: $G(\nu_x, \nu_y)$ 

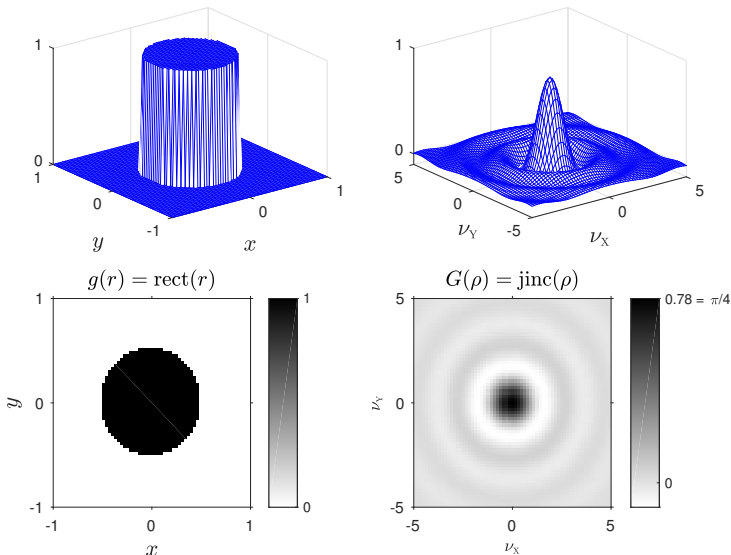
The Hankel transform and the 2D FT are intimately related.

Caution: $\text{rect}(x) \xleftrightarrow{\mathcal{F}} \text{sinc}(\nu_x)$ but $\text{rect}(r) \xleftrightarrow{\mathcal{F}_2} \text{jinc}(\rho)$.

2D FT Examples: Circularly symmetric functions

Circularly symmetric functions

| Function | Spectrum (2D FT or Hankel) |
|--------------------------------------|--|
| $\frac{1}{\pi r} \delta(r)$ | 1 |
| $\delta(r - r_0), r_0 > 0$ | $2\pi r_0 J_0(2\pi r_0 \rho)$ |
| $\frac{1}{r}$ | $\frac{1}{\rho}$ |
| $\frac{1}{r^\alpha}, 0 < \alpha < 2$ | $\frac{\pi^\alpha}{\Gamma^2(\alpha/2) \sin(\pi\alpha/2)} \frac{1}{\rho^{2-\alpha}}$ |
| $\frac{-1}{4\pi^2 r^3}$ | ρ |
| $e^{-\pi r^2}$ | $e^{-\pi \rho^2}$ |
| $\text{rect}(r)$ | $\text{jinc}(\rho) = \begin{cases} \frac{J_1(\pi\rho)}{2\rho}, & \rho \neq 0 \\ \pi/4, & \rho = 0 \end{cases}$ |
| $\text{circ}(r) = \text{rect}(r/2)$ | $4 \text{jinc}(2\rho) = \frac{J_1(2\pi\rho)}{\rho}$ |
| e^{-ar} | $\frac{2\pi a}{[(2\pi\rho)^2 + a^2]^{3/2}}$ |
| $\frac{1}{r} e^{-ar}$ | ? |
| $e^{i\pi r^2}$ | $i e^{-i\pi\rho^2}$ [19, p. 274] |



Exercise. Determine the 2D FT of the function that is unity in a disk of radius 2 centered at (5,0) and zero elsewhere. ??

Exercise. Determine the 2D FT of the circularly symmetric function $g(r) = \text{rect}\left(\frac{r-8}{4}\right)$. ??

Other 2D transforms

A number of other 2D continuous-space transforms exist; these are included for completeness but are less important for this course so can be skimmed.

2D Laplace transform

One could define a (bilateral) **2D Laplace transform** as follows:

$$G(s_1, s_2) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y) e^{-(s_1 x + s_2 y)} dx dy,$$

where s_1 and s_2 are both complex numbers.

A primary use of 1D Laplace transforms is in the solution of 1D differential-equations (and analysis of LTI systems described by differential-equations). Because 2D LSI systems are rarely described by differential equations, the 2D Laplace transform is used much less frequently.

2D Hartley transform

The 2D **Hartley transform** [\[wiki\]](#) and its inverse are defined by the linear operations:

$$H(\nu_x, \nu_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y) \text{cas}(2\pi(\nu_x x + \nu_y y)) dx dy$$

$$g(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} H(\nu_x, \nu_y) \text{cas}(2\pi(\nu_x x + \nu_y y)) d\nu_x d\nu_y,$$

where $\text{cas}(\phi) \triangleq \cos \phi + \sin \phi$. The Hartley transform of a real image is itself real, and fast algorithms exist, *e.g.*, [\[17, 20\]](#).

Radon transform

The **Radon transform** of a 2D image $f(x, y)$ is the collection of all “line integral projections” through that image, defined as:

$$p_\phi(r) = \int_{-\infty}^{\infty} f(r \cos \phi - \ell \sin \phi, r \sin \phi + \ell \cos \phi) d\ell.$$

This transform is the foundation for tomographic imaging like **X-ray computed tomography (CT)**.

Projection slice theorem

Because $p_\phi(r)$ is a 1D function of r (for fixed ϕ), it has a 1D FT $P_\theta(\nu)$. The 2D image $f(x, y)$ has a 2D Fourier transform $F(\nu_x, \nu_y) = F(\rho, \phi)$. The **projection slice theorem** states that the 1D FT of $p_\theta(r)$ is a “slice” through the 2D FT of $f(x, y)$:

$$P_\theta(\nu) = F(\rho, \theta) \Big|_{\rho=\nu}.$$

The field of tomographic imaging essentially is built upon this property, because it shows that the set of line-integral projections (at all angles ϕ) contains sufficient information to describe the 2D FT of an image $f(x, y)$, and hence $f(x, y)$ itself.

Frequency distance principle

This is an interesting topic related to what happens if you take the 2D FT of a sinogram $p_\theta(r)$. See [21–23].

1D Hilbert transform

The **Hilbert transform** of a 1D function $f(t)$ [wiki] is defined (using **Cauchy principal values**) by³ [16, p. 248]:

$$f_{\text{Hilbert}}(t) = \frac{1}{\pi} \int_{-\infty}^{\infty} \frac{1}{t-s} f(s) ds = \frac{1}{\pi t} * f(t). \quad (2.27)$$

Note that this “transform” returns another function of t . It is used frequently in tomographic imaging [26, 27]. See [28] for **2D Hilbert transform** definitions, e.g., convolution with $1/(\pi^2 xy)$.

The corresponding relationship in the frequency domain is

$$F_{\text{Hilbert}}(\nu) = -i \operatorname{sgn}(\nu) F(\nu). \quad (2.28)$$

Example. The Hilbert transform of the **rect function** $f(t) = \operatorname{rect}(t) = \mathbb{I}_{\{|t| \leq 1/2\}}$ is [16, p. 249]: $f_{\text{Hilbert}}(t) = \frac{1}{\pi} \log \left| \frac{t+1/2}{t-1/2} \right|$.

³ Note that some texts use the opposite sign, e.g., [24, p. 359] [25, p. 194].

Bandwidth and time-bandwidth products

The scaling property of the Fourier transform implies that as a signal becomes narrower in the time or space domain, its spectrum becomes wider in the frequency domain. There are fundamental limits that prevent a signal from being too narrow in both domains simultaneously, called time-bandwidth products. The upper bound on time-bandwidth products extends nicely from 1D to n -D.

1D bandwidth relationships

There are many definitions of **bandwidth** (frequency domain) and **pulse width** (time domain).

For symmetric spectra, **root mean-squared bandwidth** or **RMS bandwidth** is a natural definition:

$$\nu_{\text{rms}} \triangleq \sqrt{\frac{\int_{-\infty}^{\infty} \nu^2 |G(\nu)|^2 d\nu}{\int_{-\infty}^{\infty} |G(\nu)|^2 d\nu}}. \quad (2.29)$$

Using **Parseval's theorem** we can express the RMS bandwidth of differentiable signals as follows:

$$(2\pi\nu_{\text{rms}})^2 = \frac{\int_{-\infty}^{\infty} (2\pi\nu)^2 |G(\nu)|^2 d\nu}{\int_{-\infty}^{\infty} |G(\nu)|^2 d\nu} = \frac{\int_{-\infty}^{\infty} |i2\pi\nu G(\nu)|^2 d\nu}{\int_{-\infty}^{\infty} |g(t)|^2 dt} = \frac{\int_{-\infty}^{\infty} \left|\frac{d}{dt}g(t)\right|^2 dt}{\int_{-\infty}^{\infty} |g(t)|^2 dt}. \quad (2.30)$$

Example. Gaussian signal. For $t_0 > 0$:

$$g(t) = e^{-\pi(t/t_0)^2} \xleftrightarrow{\mathcal{F}} G(\nu) = t_0 e^{-\pi(\nu t_0)^2}. \quad (2.31)$$

The RMS bandwidth is:

(Use change of variables $x = \nu t_0$.)

$$\nu_{\text{rms}} = \sqrt{\frac{\int_{-\infty}^{\infty} \nu^2 |G(\nu)|^2 d\nu}{\int_{-\infty}^{\infty} |G(\nu)|^2 d\nu}} = \sqrt{\frac{\int_{-\infty}^{\infty} \nu^2 t_0^2 e^{-2\pi(\nu t_0)^2} d\nu}{\int_{-\infty}^{\infty} t_0^2 e^{-2\pi(\nu t_0)^2} d\nu}} = \sqrt{\frac{\frac{1}{t_0} \int_{-\infty}^{\infty} x^2 e^{-2\pi x^2} dx}{t_0 \int_{-\infty}^{\infty} e^{-2\pi x^2} dx}} = \frac{1}{t_0} \sqrt{\frac{\sqrt{2}/(8\pi)}{\sqrt{2}/2}} = \frac{1}{t_0} \frac{1}{2\sqrt{\pi}}.$$

For symmetric signals, **root mean-squared pulse width** or **RMS pulse width** is the natural measure:

$$t_{\text{rms}} \triangleq \sqrt{\frac{\int_{-\infty}^{\infty} t^2 |g(t)|^2 dt}{\int_{-\infty}^{\infty} |g(t)|^2 dt}}. \quad (2.32)$$

Example. Gaussian signal in (2.31) above. The RMS pulse width is:

(Use change of variables $x = t/t_0$.)

$$t_{\text{rms}} = \sqrt{\frac{\int_{-\infty}^{\infty} t^2 |g(t)|^2 dt}{\int_{-\infty}^{\infty} |g(t)|^2 dt}} = \sqrt{\frac{\int_{-\infty}^{\infty} t^2 e^{-2\pi(t/t_0)^2} dt}{\int_{-\infty}^{\infty} e^{-2\pi(t/t_0)^2} dt}} = \sqrt{\frac{t_0^3 \int_{-\infty}^{\infty} x^2 e^{-2\pi x^2} dx}{t_0 \int_{-\infty}^{\infty} e^{-2\pi x^2} dx}} = t_0 \frac{1}{2\sqrt{\pi}}.$$

Time-bandwidth product

Using the Cauchy-Schwarz inequality, one can show [29] [24, p. 178] (for signals satisfying certain regularity conditions):

$$\nu_{\text{rms}} t_{\text{rms}} \geq \frac{1}{4\pi}. \quad (2.33)$$

This **uncertainty relation** is similar to the Heisenberg uncertainty principle. For discrete-time and other extensions, see [30].

Because of the uncertainty relation (2.33), a signal cannot be “localized” in both time and in frequency. Extreme cases:

- Dirac impulse: narrow in time, wide spectrum
- sinusoid signal: periodic (infinite duration) in time, narrow spectrum
- gaussian signal: time-bandwidth product is smallest possible.

Example. For a Gaussian signal: $2\pi\nu_{\text{rms}}t_{\text{rms}} = 1/2$, which is “optimal” in a time-bandwidth sense.

Example. $g(t) = \text{tri}(t) \xleftrightarrow{\mathcal{F}} G(\nu) = \text{sinc}^2(\nu)$.

$$t_{\text{rms}}^2 = \frac{\int_0^1 t^2 (1-t)^2 dt}{\int_0^1 (1-t)^2 dt} = \frac{1/30}{1/3} = \frac{1}{10}, \text{ so } t_{\text{rms}} = 1/\sqrt{10}.$$

$$\text{Using (2.30): } (2\pi\nu_{\text{rms}})^2 = \frac{\int_{-\infty}^{\infty} \left| \frac{d}{dt} \text{tri}(t) \right|^2 dt}{\int_{-\infty}^{\infty} |\text{tri}(\nu)|^2 dt} = \frac{2}{2/3} = 3.$$

Thus $2\pi\nu_{\text{rms}}t_{\text{rms}} = \sqrt{3/10} \approx 0.548 \geq 1/2$. So tri exceeds the minimum possible time-bandwidth product by only about 10%.

Exercise. Show that for a **cubic B-spline** (see (8.25) in Ch. 8), the time-bandwidth product is $2\pi\nu_{\text{rms}}t_{\text{rms}} \approx 0.5012$. In other words, the cubic B-spline has a time-bandwidth product that is only about 1% larger than the minimum possible.

***n*-dimensional case**

For n -dimensional signals that are possibly asymmetric, natural definitions of width are:

$$\nu_{\text{rms}} \triangleq \inf_{\vec{\mu} \in \mathbb{R}^n} \sqrt{\frac{\int_{-\infty}^{\infty} \|\vec{\nu} - \vec{\mu}\|^2 |G(\vec{\nu})|^2 d\vec{\nu}}{\int_{-\infty}^{\infty} |G(\vec{\nu})|^2 d\vec{\nu}}}, \quad t_{\text{rms}} \triangleq \inf_{\vec{\mu} \in \mathbb{R}^n} \sqrt{\frac{\int_{-\infty}^{\infty} \|\vec{x} - \vec{\mu}\|^2 |g(\vec{x})|^2 d\vec{x}}{\int_{-\infty}^{\infty} |g(\vec{x})|^2 d\vec{x}}}$$

where, for $n = 2$, $\vec{x} = (x, y)$ and $\vec{\nu} = (\nu_x, \nu_y)$. And one can show [16, p. 108] the following uncertainty relation:

$$\nu_{\text{rms}} t_{\text{rms}} \geq \frac{n}{4\pi}.$$

One can also show that a **space-limited signal** cannot be **band-limited** and vice-versa, except for the zero signal, using complex analysis [wiki]. See also the **Paley Wiener Theorem** [wiki].

Summary

This chapter has introduced 2D Fourier series and 2D Fourier transforms.

The majority of the concepts in 2D generalize readily from those in 1D. New concepts include **separability** and **rotation**, the latter leading to the Hankel transform.

What part of this chapter did you find most confusing? ??

[RQ]

Bibliography

- [1] D. G. Luenberger. *Optimization by vector space methods*. New York: Wiley, 1969.
- [2] M-A. Parseval des Chênes. *Mémoire sur les séries et sur l'intégration complète d'une équation aux différences partielles linéaire du second ordre, à coefficients constants*. Presented before the Académie des Sciences (Paris) on 5 April 1799. Published in *Mémoires présentés à l'Institut des Sciences, Lettres et Arts, par divers savans, et lus dans ses assemblées. Sciences, mathématiques et physiques. (Savans étrangers.)*, vol. 1, pages 638-648 (1806). 1806.
- [3] M. Plancherel. "Contribution à l'étude de la représentation d'une fonction arbitraire par des intégrales définies". In: *Rendiconti del Circolo Matematico di Palermo* 30.1 (1910), 289–335.
- [4] M. Reed and B. Simon. *Methods of modern mathematical physics, Vol. I: Functional analysis*. New York: Academic Press, 1972.
- [5] A. Dominguez. "Highlights in the history of the Fourier transform [Retrospectroscope]". In: *IEEE Pulse* 7.1 (Jan. 2016), 53–61.
- [6] M. Lerch. "Sur un point de la théorie des fonctions génératrices d'Abel". In: *Acta Mathematica* 27.1 (1903), 339–351.
- [7] J. W. Goodman. *Introduction to Fourier optics*. 3rd edition. Greenwood Village, CO: Roberts, 2005.
- [8] R. N. Bracewell. *Two-dimensional imaging*. New York: Prentice-Hall, 1995.
- [9] G. McGibney, M. R. Smith, S. T. Nichols, and A. Crawley. "Quantitative evaluation of several partial Fourier reconstruction algorithms used in MRI". In: *Mag. Res. Med.* 30.1 (July 1993), 51–9.
- [10] Z. P. Liang, F. E. Boada, R. T. Constable, E. M. Haacke, P. C. Lauterbur, and M. R. Smith. "Constrained reconstruction methods in MR imaging". In: *Reviews of Magnetic Resonance in Medicine* 4 (1992), 67–185.
- [11] I. A. Cunningham. "Applied linear-systems theory". In: *Handbook of Medical Imaging, Volume 1. Physics and Psychophysics*. Ed. by J. Beutel, H. L. Kundel, and R. L. Van Metter. Bellingham: SPIE, 2000, pp. 79–160.
- [12] J. W. Goodman. *Introduction to Fourier optics*. New York: McGraw-Hill, 1968. ISBN: 978-0974707723.
- [13] A. E. Yagle and H. Ahn. "Partitioning algorithms for 1-D and 2-D discrete phase-retrieval problems with disconnected support". In: *IEEE Trans. Sig. Proc.* 45.9 (Sept. 1997), 2220–30.
- [14] M. Abramowitz and I. A. Stegun. *Handbook of mathematical functions*. New York: Dover, 1964.
- [15] K. T. Smith and F. Keiner. "Mathematical foundations of computed tomography". In: *Appl. Optics* 24.23 (1985), 3950–7.
- [16] L. Grafakos. *Classical and modern Fourier analysis*. NJ: Pearson, 2004.
- [17] J. A. Ferrari, D. Perciante, and A. Dubra. "Fast Hankel transform of nth order". In: *J. Opt. Soc. Am. A* 16.10 (Oct. 1999), 2581–2.
- [18] R. M. Lewitt. "Multidimensional digital image representations using generalized Kaiser-Bessel window functions". In: *J. Opt. Soc. Am. A* 7.10 (Oct. 1990), 1834–46.

- [19] R. Bracewell. *The Fourier transform and its applications*. New York: McGraw-Hill, 1978.
- [20] C. H. Paik and M. D. Fox. “Fast Hartley transforms for image processing”. In: *IEEE Trans. Med. Imag.* 7.2 (June 1988), 149–53.
- [21] R. M. Lewitt, P. R. Edholm, and W. Xia. “Fourier method for correction of depth-dependent collimator blurring”. In: *Proc. SPIE 1092 Med. Im. III: Im. Proc.* 1989, 232–43.
- [22] S. J. Glick, B. C. Penney, M. A. King, and C. L. Byrne. “Noniterative compensation for the distance-dependent detector response and photon attenuation in SPECT”. In: *IEEE Trans. Med. Imag.* 13.2 (June 1994), 363–74.
- [23] W. Xia, R. M. Lewitt, and P. R. Edholm. “Fourier correction for spatially variant collimator blurring in SPECT”. In: *IEEE Trans. Med. Imag.* 14.1 (Mar. 1995), 100–15.
- [24] R. Bracewell. *The Fourier transform and its applications*. 3rd ed. New York: McGraw-Hill, 2000.
- [25] H. H. Barrett and K. J. Myers. *Foundations of image science*. New York: Wiley, 2003.
- [26] F. Noo, R. Clackdoyle, and J. D. Pack. “A two-step Hilbert transform method for 2D image reconstruction”. In: *Phys. Med. Biol.* 49.17 (Sept. 2004), 3903–24.
- [27] A. Katsevich. “Singular value decomposition for the truncated Hilbert transform”. In: *Inverse Prob.* 26.11 (Nov. 2010), p. 115011.
- [28] J. Valentí Lorenzo-Ginori. *An approach to the 2D Hilbert transform for image processing applications*. 2007.
- [29] D. Gabor. “Theory of communication”. In: *J Inst Elec Eng* 93.26 (Nov. 1946), 429–57.
- [30] D. L. Donoho and P. B. Stark. “Uncertainty principles and signal recovery”. In: *SIAM J. Appl. Math.* 49.3 (June 1989), 906–31.

Chapter 4

2D Sampling

Contents (class version)

| | |
|---|-------------|
| 4.0 Introduction | 4.2 |
| 4.1 Ideal 2D sampling | 4.2 |
| 4.2 Signal reconstruction/interpolation | 4.13 |
| Frequency-domain reconstruction “method” | 4.13 |
| Space-domain reconstruction method | 4.14 |
| Sub-Nyquist sampling in 2D | 4.19 |
| 4.3 Aliasing | 4.20 |
| Imaging properties of lenses and anti-aliasing | 4.20 |
| Sampling images that are not band-limited | 4.23 |
| Digital displays: Aliasing in signal reconstruction | 4.24 |
| 4.4 Summary | 4.31 |

4.0 Introduction

This chapter considers 2D sampling using continuous-space 2D Fourier transforms. Ideal sinc interpolation is also introduced. A later chapter considers practical interpolation methods in much more detail. A later chapter considers how sampling relates to other Fourier transforms (DSFT and DFT) in more detail. Sampling is a crucial step as we transition to digital image processing topics. Anyone who has used a camera in a mobile phone has first-hand experience with 2D sampling, whether they know it or not.

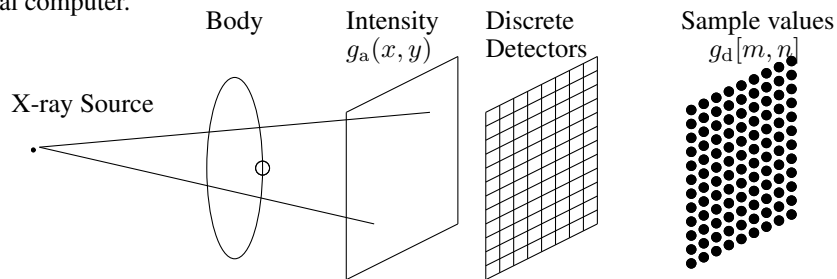
4.1 Ideal 2D sampling

Thus far we have described an image, such as the output of an image capture system, as a continuous function $g(x, y)$. This is convenient for analysis, but in any digital imaging system, we can only record a (finite) 2D array of numbers $\{g_d[m, n]\}$ that should somehow represent $g(x, y)$.

A word about notation in this section is in order. When focusing on continuous-space images, we write $g(x, y)$ rather than the more cumbersome $g_a(x, y)$. When focusing on discrete-space images (later on), we will write $g[m, n]$ rather than the more cumbersome $g_d[m, n]$. But chapter requires both continuous-space and discrete-space images, so we include the subscripts for clarity. The following table summarizes the various functions considered.

| Signal domain | signal | spectrum | transform type |
|---------------------------|-------------|-------------------------|-----------------|
| Continuous space (analog) | $g_a(x, y)$ | $G_a(\nu_x, \nu_y)$ | 2D FT |
| Impulse synthesized | $g_s(x, y)$ | $G_s(\nu_x, \nu_y)$ | 2D FT |
| Discrete space (digital) | $g_d[m, n]$ | $G(\Omega_1, \Omega_2)$ | 2D DSFT (Ch. 5) |
| DS (finite domain) | | $G[k, l]$ | 2D DFT (Ch. 7) |

Example. In some digital mammography systems, the intensity of a transmitted X-ray beam is recorded in “analog” format on some continuous detector (e.g., photographic film, or a rewritable equivalent). The intensity that is recorded is our $g_a(x, y)$. Then a digitization device (such as a laser-scanning system or CCD camera) samples $g_a(x, y)$ over a regular grid, and sends (quantized) values $g_d[m, n]$ to the digital computer.



Example. An optical scanner is another device that converts a continuous, analog image (picture) into a sampled, digital form.

Ideal uniform rectilinear sampling

The most common¹ form of 2D sampling uses a 2D uniformly sampled grid (matching raster-scan displays and printers) and is convenient for storage. (It is usually not the most efficient sampling scheme however, as shown in [2, Pr. 1.35]. See HW.)

Let Δ_x and Δ_y denote the **sampling intervals** in the x and y directions respectively. We define **ideal 2D sampling** by the model:

$$g_d[m, n] = g_a(m\Delta_x, n\Delta_y), \quad \forall [m, n] \in \mathbb{Z}. \quad (4.1)$$

Note there are no Dirac impulses here!

In what ways is this sampling model idealized?.

[RQ]

??

The ratios $1/\Delta_x$ and $1/\Delta_y$ can be called the **sampling rates** in the x and y directions, respectively.

What are the units of these sampling rates? ??

[RQ]

When considering non-Cartesian sampling, it is more relevant to quote the **sampling density** of a given sampling pattern, which has units of samples per unit area. For rectilinear sampling, the sampling density is $1/(\Delta_x\Delta_y)$.

¹That is, the most common in digital systems. In human retinas, the “sampling” of rods and cones is systematically nonuniform (higher concentration in fovea), and some semiconductor sensors have been developed that mimic this nonuniform sampling property [1].

Non-ideal (realistic) sampling

A more realistic sampling model accounts for finite detector element size:

$$g_d[m, n] = \frac{1}{\Delta_x \Delta_y} \int_{(n-1/2)\Delta_y}^{(n+1/2)\Delta_y} \int_{(m-1/2)\Delta_x}^{(m+1/2)\Delta_x} g_a(x, y) \, dx \, dy. \quad (4.2)$$

(Note: even further generalizations are possible, *e.g.*, [3].)

(Analyzing this is a HW problem!)

Overview of sampling analysis

What are the questions to answer when analyzing sampling?

- Q1. Can we recover $g_a(x, y)$ from $g_d[m, n]$? (Yes, if $g_a(x, y)$ is band-limited.)
- Q2. What Δ_x and Δ_y values allow recovery? (Need $\Delta_x < 1/(2\nu_x^{\max})$, likewise for Δ_y)
- Q3. How do we recover $g_a(x, y)$ from $g_d[m, n]$? (Rect window in frequency domain, sinc interpolation in space domain.)
- Q4. What happens if the sampling conditions are violated? (Aliasing)

These questions and their answers are called **Nyquist-Shannon sampling theory**.

For *analysis* of sampling, it is convenient to use a 2D **Dirac comb** (“bed of nails”) to synthesize (on paper) the following continuous-space “function” $g_s(x, y)$ from the samples $\{g_d[m, n]\}$. This manipulation is purely analytical—no such impulses exist in *any* real-world A/D converter. The hypothetical “sample carrying” continuous-space image is defined by

$$g_s(x, y) \triangleq \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g_d[m, n] \Delta_x \Delta_y \delta_2(x - m\Delta_x, y - n\Delta_y) \quad (4.3)$$

$$= g_a(x, y) \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \Delta_x \Delta_y \delta_2(x - m\Delta_x, y - n\Delta_y) \quad \text{using (4.1), by sampling property of } \delta_2(x, y)$$

$$= g_a(x, y) \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \delta_2(x/\Delta_x - m, y/\Delta_y - n) \quad \text{by scaling property of } \delta_2(x, y)$$

$$= g_a(x, y) \left[\text{comb}_2 \left(\frac{x}{\Delta_x}, \frac{y}{\Delta_y} \right) \right]. \quad (4.4)$$

Using the **sampling property of the Dirac impulse** depends on some assumptions, a subtle point often ignored. Here we assume that $g_a(x, y)$ is **continuous** (at least at all the sample points) and that its samples $g_d[m, n]$ are **absolutely summable**.

Because $\text{comb}_2(x, y) \xleftrightarrow{\mathcal{F}_2} \text{comb}_2(\nu_x, \nu_y)$, it follows from the modulation and scaling properties of the 2D FT that the spectrum $G_s(\nu_x, \nu_y)$ of the hypothetical impulse sampled image $g_s(x, y)$ is related to the spectrum $G_a(\nu_x, \nu_y)$ of the original image $g_a(x, y)$ as follows:

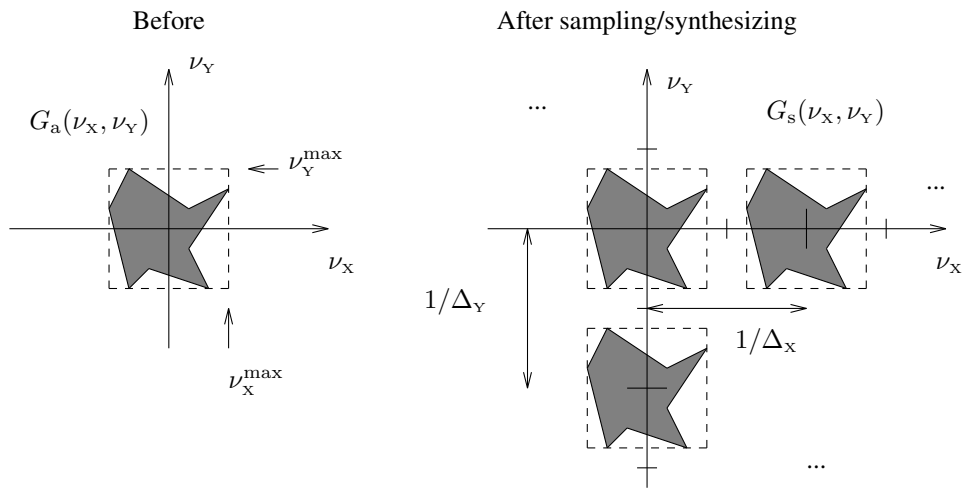
$$\begin{aligned} g_s(x, y) = g_a(x, y) \left[\text{comb}_2 \left(\frac{x}{\Delta_x}, \frac{y}{\Delta_y} \right) \right] &\xleftrightarrow{\mathcal{F}_2} G_s(\nu_x, \nu_y) = G_a(\nu_x, \nu_y) ** [\Delta_x \Delta_y \text{comb}_2(\nu_x \Delta_x, \nu_y \Delta_y)] \\ &= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} G_a(\nu_x - k/\Delta_x, \nu_y - l/\Delta_y), \end{aligned} \quad (4.5)$$

because (see (1.18))

$$\Delta_x \Delta_y \text{comb}_2(\nu_x \Delta_x, \nu_y \Delta_y) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \delta_2(\nu_x - k/\Delta_x, \nu_y - l/\Delta_y).$$

Thus $G_s(\nu_x, \nu_y)$ consists of the sum of shifted copies of the spectrum $G_a(\nu_x, \nu_y)$ as illustrated below.

$$g_s(x, y) \xleftrightarrow{\mathcal{F}_2} G_s(\nu_x, \nu_y) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} G_a(\nu_x - k/\Delta_x, \nu_y - l/\Delta_y) \quad (4.6)$$



Now we can answer the questions posed above.

Q1. Can we recover $g_a(x, y)$ from $g_d[m, n]$?

In general one *cannot* recover $g_a(x, y)$ from the samples $\{g_d[m, n]\}$, no matter how closely spaced. However, there are classes of functions for which it *is* possible in theory. One such class of particular importance is the class of band-limited functions. (For a review of other formulations see [4, 5].)

We say $g_a(x, y)$ is **band-limited** to $(\nu_x^{\max}, \nu_y^{\max})$ iff $G_a(\nu_x, \nu_y) = 0$ for $|\nu_x| > \nu_x^{\max}$ or $|\nu_y| > \nu_y^{\max}$, as illustrated above.

Q2. How finely must we sample?

If $G_a(\nu_x, \nu_y)$ is **band-limited** to $(\nu_x^{\max}, \nu_y^{\max})$ and if $1/\Delta_x > 2\nu_x^{\max}$ and $1/\Delta_y > 2\nu_y^{\max}$, then there is no overlap of the replicates of $G_a(\nu_x, \nu_y)$ in $G_s(\nu_x, \nu_y)$. Spectral overlap is called **aliasing** (higher spatial frequencies will “alias” as lower spatial frequencies).

If $\Delta_x < \frac{1}{2\nu_x^{\max}}$ and $\Delta_y < \frac{1}{2\nu_y^{\max}}$ then we say the image (or system) is **adequately sampled**.

Otherwise we say the image is **under sampled**.

If $\Delta_x \ll 1/(2\nu_x^{\max})$ and $\Delta_y \ll 1/(2\nu_y^{\max})$ then we say the image is **over sampled**.

To see why it is important to sample finely enough, consider the following example.

Example. Consider sampling the (band-limited!) image $g_a(x, y) = \cos\left(2\pi\left(\frac{1+\epsilon_X}{2\Delta_X}\right)x + 2\pi\left(\frac{1+\epsilon_Y}{2\Delta_Y}\right)y\right)$ with sampling intervals (Δ_X, Δ_Y) , where $0 < \epsilon_X < 1$ and $0 < \epsilon_Y < 1$. Then, $g_a(x, y)$ is undersampled, producing samples

$$\begin{aligned} g_d[m, n] &= \cos\left(2\pi\left(\frac{1+\epsilon_X}{2\Delta_X}\right)m\Delta_X + 2\pi\left(\frac{1+\epsilon_Y}{2\Delta_Y}\right)n\Delta_Y\right) \\ &= \cos(\pi(1+\epsilon_X)m + \pi(1+\epsilon_Y)n). \end{aligned}$$

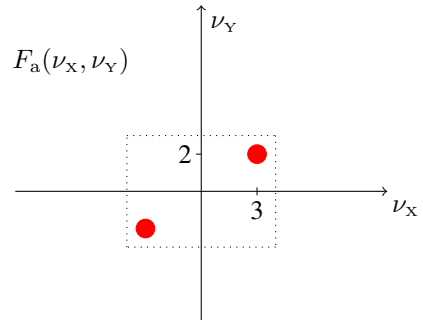
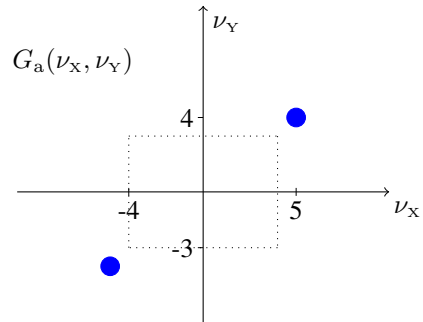
For these sampling intervals, a different (lower frequency) image has exactly the same samples.

Specifically, consider the signal (also band-limited) $f_a(x, y) = \cos\left(2\pi\left(\frac{1-\epsilon_X}{2\Delta_X}\right)x + 2\pi\left(\frac{1-\epsilon_Y}{2\Delta_Y}\right)y\right)$, whose samples are

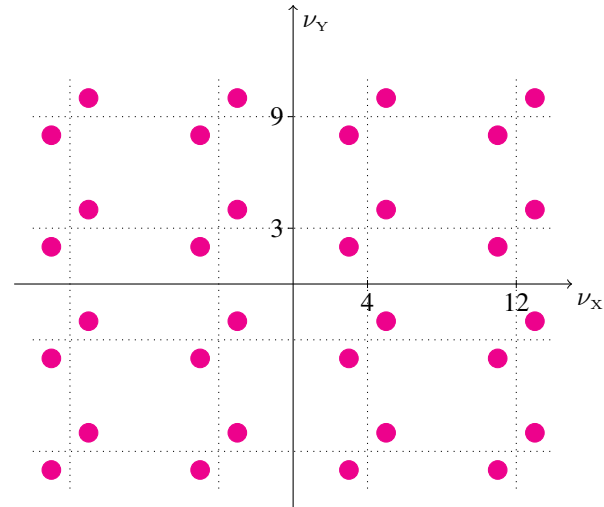
$$\begin{aligned} f_d[m, n] &= \cos\left(2\pi\left(\frac{1-\epsilon_X}{2\Delta_X}\right)m\Delta_X + 2\pi\left(\frac{1-\epsilon_Y}{2\Delta_Y}\right)n\Delta_Y\right) \\ &= \cos(\pi(1-\epsilon_X)m + \pi(1-\epsilon_Y)n) \\ &= \cos(2\pi m - \pi m + \pi\epsilon_X m + 2\pi n - \pi n + \pi\epsilon_Y n) \quad \text{because } \cos(2\pi k - \theta) = \cos(\theta) \text{ for } k \in \mathbb{Z} \\ &= \cos(\pi(1+\epsilon_X)m + \pi(1+\epsilon_Y)n) \\ &= g_d[m, n]. \end{aligned}$$

Because $f_a(x, y)$ and $g_a(x, y)$ have the same samples, they are said to be **aliases** of each other after sampling.

Example. If $(\Delta_x, \Delta_y) = (1/8, 1/6)$ and $(\epsilon_x, \epsilon_y) = (1/4, 1/3)$, then for the preceding example the signals are $g_a(x, y) = \cos(2\pi(5x + 4y))$ and $f_a(x, y) = \cos(2\pi(3x + 2y))$. The following diagrams illustrate their spectra.



$$F_s(\nu_x, \nu_y) = G_s(\nu_x, \nu_y)$$



Nyquist sampling “rate”

In 2D, there are a few more possible definitions of a scalar value for **Nyquist sampling “rate”** than in 1D.

One possible definition is twice the highest spatial frequency of the rectangle bounding the spectra support:

$$\nu_{\max} \triangleq 2 \max \{ \nu_x^{\max}, \nu_y^{\max} \}. \quad (4.7)$$

Alternatively we could use $\nu_{\max} = 2\rho_{\max}$, *i.e.*, twice the radius of the smallest disk containing the support of the signal spectrum. In either case, if we sample with $\Delta_x < \frac{1}{2\nu_{\max}}$ and $\Delta_y < \frac{1}{2\nu_{\max}}$ then we certainly will be adequately sampled, but we may be sampling more finely in one of the two dimensions than needed if $\nu_x^{\max} \neq \nu_y^{\max}$.

Another definition of sampling rate is the sampling density (number of samples per unit area needed):

$$4\nu_x^{\max}\nu_y^{\max}.$$

In practice, usually $\nu_x^{\max} = \nu_y^{\max}$ (the band-limit of real-world images usually is independent of the coordinate system) in which case (4.7) is a suitable definition for a scalar-valued Nyquist sampling rate.

If a band-limited image satisfies $\nu_x^{\max} = \nu_y^{\max}$ and a suitable sampling density is 100 samples / mm² then which of these are a possible values for ν_x^{\max} ? ?? [RQ]

Non-band-limited images

If $g_a(x, y)$ is not band-limited, then in general no algorithm can recover $g_a(x, y)$ from $g_d[m, n]$ perfectly without more assumptions. Other information (such as nonnegativity) would be required. For example, an active area of research in signal processing is to consider signal models based on assumptions of **sparsity** with respect to some representation. In these **compressive sensing** formulations, one usually still uses linear measurements, but uses a nonlinear signal recovery method, e.g., [6].

However, for “reasonably behaved” images $g_a(x, y)$, one can *approximately* recover $g_a(x, y)$ from its samples by taking Δ_x and Δ_y sufficiently small. In fact, the energy between $g_a(x, y)$ and the recovered approximation from the samples can be made arbitrarily small by making Δ_x and Δ_y small enough. This is the subject of **approximation theory** and it is relevant to signal and image processing because rarely are signals exactly band-limited.

Avoiding aliasing

How can we avoid aliasing of non-band-limited images?

Use a prefilter to limit the spatial frequencies impinging on the recorder. For example, with the pupil function of a lens.

Does that solution allow perfect image recovery from the samples?

Although the pre-filter eliminates aliasing, one can recover only the filtered image from the samples.

4.2 Signal reconstruction/interpolation

Q3. How can we “recover” a (band-limited) image $g_a(x, y)$ from its samples $\{g_d[m, n]\}$? _____

(Of course we cannot store *all* of $g_a(x, y)$ on a computer, but we might want values of $g_a(x, y)$ on a grid finer than the original sampling. If the function is band-limited and adequately sampled then this can be done!) Or we might like to design a suitable 2D D/A converter (*e.g.*, a computer monitor) that takes a digital input signal $g_d[m, n]$ and recreates the analog $g_a(x, y)$ at its output.

The key to methods for recovering $g_a(x, y)$ from $\{g_d[m, n]\}$ is the following fact.

If $G_a(\nu_x, \nu_y)$ is band-limited to $(1/(2\Delta_x), 1/(2\Delta_y))$ (and hence adequately sampled), then

$$G_a(\nu_x, \nu_y) = G_s(\nu_x, \nu_y) \text{rect}_2(\nu_x \Delta_x, \nu_y \Delta_y). \quad (4.8)$$

This relationship follows directly from the form of $G_s(\nu_x, \nu_y)$ given in (4.6) above.

The ideal lowpass filter $H(\nu_x, \nu_y) = \text{rect}_2(\nu_x \Delta_x, \nu_y \Delta_y)$ selects out the central replicate. This replicate is precisely the spectrum $G_a(\nu_x, \nu_y)$ of the original signal. This is easily seen in the spectrum, but is perhaps puzzling in the space domain!

Frequency-domain reconstruction “method”

- Construct $g_s(x, y)$ from samples $\{g_d[m, n]\}$. (This is hypothetical!)
- Take 2D FT of $g_s(x, y)$ to get $G_s(\nu_x, \nu_y)$
- Truncate $G_s(\nu_x, \nu_y)$ to the band-limited region (set everything outside to zero) to get $G_a(\nu_x, \nu_y)$.
- Take inverse 2D Fourier transform of $G_a(\nu_x, \nu_y)$ to get $g_a(x, y)$.

This **Fourier interpolation** method is impractical because $G_a(\nu_x, \nu_y)$ is continuous; on a computer one could store only a finite number of samples of the spectra, so one would not recover $g_a(x, y)$ exactly *at any coordinates*. (However, a discrete approximation to this approach is sometimes applied; see MATLAB’s `interpft` routine, discussed in Ch. 8.)

Space-domain reconstruction method

Taking the inverse 2D FT of key property (4.8) above yields:

$$\begin{aligned}
 g_a(x, y) &= g_s(x, y) ** \left[\frac{1}{\Delta_x \Delta_y} \text{sinc}_2 \left(\frac{x}{\Delta_x}, \frac{y}{\Delta_y} \right) \right] \\
 &= \left[\sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g_d[m, n] \Delta_x \Delta_y \delta_2(x - m\Delta_x, y - n\Delta_y) \right] ** \left[\frac{1}{\Delta_x \Delta_y} \text{sinc}_2 \left(\frac{x}{\Delta_x}, \frac{y}{\Delta_y} \right) \right] \\
 \implies g_a(x, y) &= \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g_d[m, n] \text{sinc}_2 \left(\frac{x - m\Delta_x}{\Delta_x}, \frac{y - n\Delta_y}{\Delta_y} \right), \tag{4.9}
 \end{aligned}$$

using the shift property for convolution with a Dirac impulse.

Thus we can recover $g_a(x, y)$ by **interpolating** the samples $g_d[m, n]$ using sinc functions. This is called **sinc interpolation**.

The sinc function has infinite extent, so in practice approximate interpolators are used more frequently (such as B-splines, discussed in Ch. 8). See MATLAB's `interp2` routine.

Example. The following figures compare sinc interpolation (4.9), of signal samples $g_d[m, n]$ shown in the upper left, with some practical alternatives available in MATLAB's `interp2` routine. (We will discuss these interpolators in Ch. 8.)

The abbreviation “**flop**” in the figure stands for **floating point operations**, *i.e.*, the number of adds and multiplies needed to perform each of those interpolation methods. Note that sinc interpolation requires the most computation.

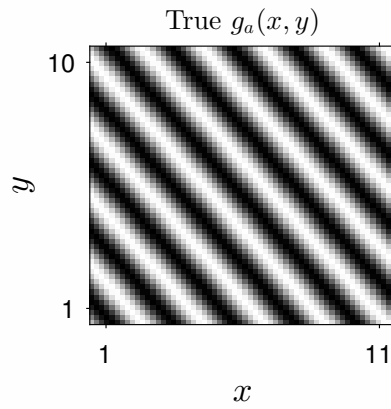
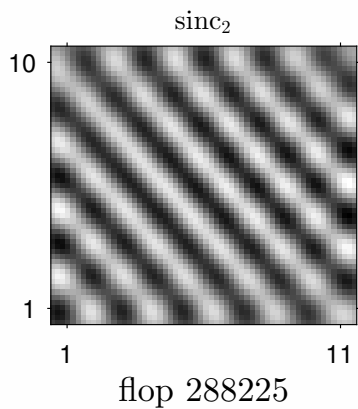
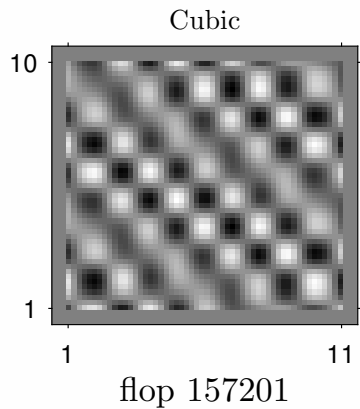
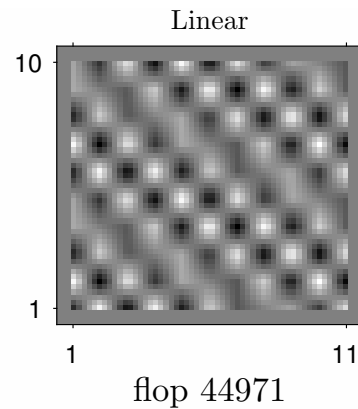
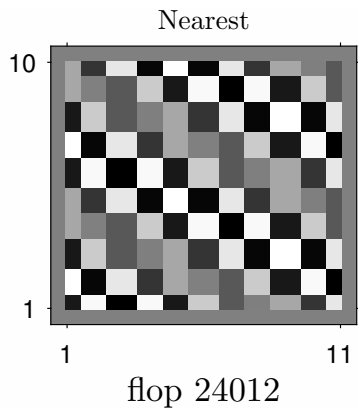
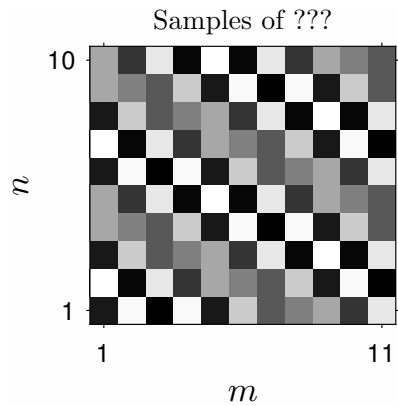
Why are MATLAB's choices so poor in this example?

Because the signal is a sinusoid whose frequency is very close to half of the sampling rate.

What causes the remaining artifacts in the sinc interpolated image?

Finite set of of object samples, *i.e.*, either the original object is band-limited but we have only a finite number of samples, or the object is space limited and thus not band-limited.

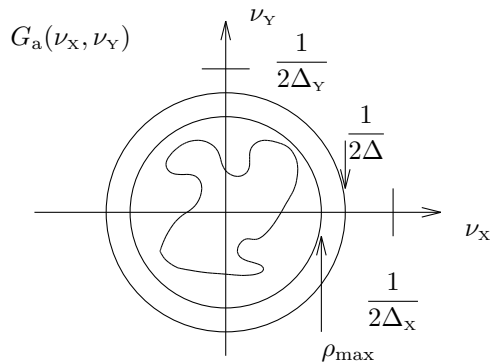
One should not conclude from this example that sinc interpolation is always preferable in practice!



Non-rectangular spectral support (over sampling)

If $G_a(\nu_x, \nu_y)$ is band-limited with a support region that is smaller than $[-1/(2\Delta_x), 1/(2\Delta_x)] \times [-1/(2\Delta_y), 1/(2\Delta_y)]$, the usual rectangle, then (4.8) is not the only approach that will work in the frequency domain. In fact, there will be *many* ideal interpolators! In a sense there are “more options” in 2D than in 1D.

Example. If $G_a(\nu_x, \nu_y)$ is zero when $\rho = \sqrt{\nu_x^2 + \nu_y^2} > \rho_{\max}$, i.e., $g_a(x, y)$ is band-limited to a *circular* region in the frequency domain, and if $\Delta_x < \frac{1}{2\rho_{\max}}$ and $\Delta_y < \frac{1}{2\rho_{\max}}$, then we could choose to replace the 2D rect function in (4.8) with a circ function $\text{rect}(\rho\Delta)$ using any Δ such that $\max(\Delta_x, \Delta_y) \leq \Delta < \frac{1}{2\rho_{\max}}$.



The space-domain reconstruction formula corresponding to (4.9) becomes

$$g_a(x, y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g_d[m, n] \boxed{??} \text{jinc} \left(\frac{\sqrt{(x - m\Delta_x)^2 + (y - n\Delta_y)^2}}{\Delta} \right), \quad (4.10)$$

because $\frac{1}{\Delta^2} \text{jinc}(r/\Delta) \xleftrightarrow{\mathcal{F}_2} \text{rect}(\rho\Delta)$. This is called **jinc interpolation**. It is used rarely because it is computationally expensive, like $\text{sinc}_2(\cdot, \cdot)$, but even more so because it uses Bessel functions.

Find the value of the “??” in the jinc interpolator above when $\Delta_x = 0.4$, $\Delta_y = 0.6$, $\Delta = 2$. **??** [RQ]

Alternative reconstruction methods

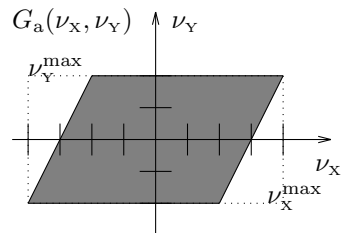
Other reconstruction methods, generally referred to as **interpolation** methods, will be discussed later in EECS 556 in Ch. 8.

Sub-Nyquist sampling in 2D

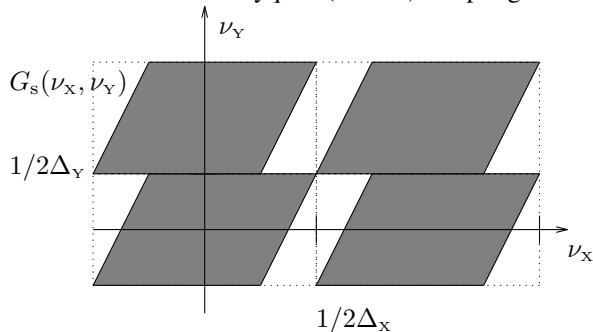
Example.

Suppose the 2D frequency-domain support of a signal lies within a parallelogram as shown to the right.

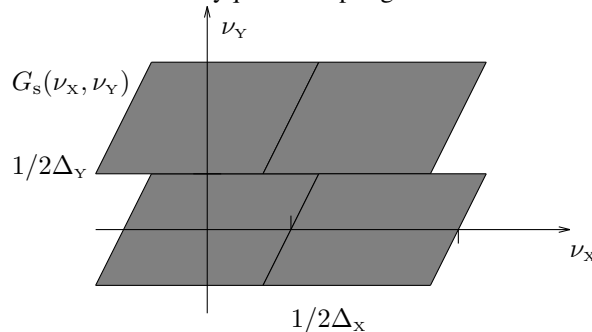
One application of this type of spectrum/sampling is in certain dynamic imaging problems where one of the axes is time [7].



With Nyquist (critical) sampling:



With “Sub-Nyquist” sampling:



Can we sample at less than the “usual Nyquist rate” in this parallelogram case and still recover $g(x, y)$? ??

[RQ]

If so, will we use the usual sinc interpolator?

??

[RQ]

4.3 Aliasing**What happens if $g_a(x, y)$ is not band-limited?**

If $g_a(x, y)$ is *not* band-limited or is sampled below the Nyquist rate, then **aliasing** will occur.

Aliasing means more than one frequency component contributes to a single component of the Fourier transform.

Many imaging systems (*e.g.*, a lens in an optical system, see below) are intrinsically nearly band-limited, so aliasing can be eliminated approximately by appropriate sampling. (The finite band-limit of practical imaging systems is fortunate from an aliasing perspective, but unfortunate from a resolution perspective.) Nevertheless, the cost in terms of number of detector elements may be high. For example, digital mammography systems can have 4K by 4K pixels, presenting a nontrivial data storage problem alone, as well as a challenge for fast image processing. (The aggressive lossy JPEG compression used in consumer digital cameras is usually unsuitable for medical images.)

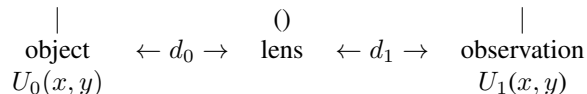
In electrical systems we make anti-alias filters out of RLC circuits that approximate an ideal low-pass. For optical systems, we can make anti-alias filters out of lenses, as described next.

Imaging properties of lenses and anti-aliasing (Coherent case)

Ch. 2 described how one particular configuration of an (ideal thin) lens can provide a 2D FT of an image. Now we turn to the more typical use of a lens: to make an **image** of the object. For simplicity, assume monochromatic illumination (a coherent system) with wavelength λ . For imaging, we need the distance d_0 between the object and the lens and the distance d_1 between the lens and the detector to satisfy the **lens law** or **thin lens formula**

$$1/d_0 + 1/d_1 = 1/F, \quad (4.11)$$

where F is the **focal length** of the lens. This relationship ensures that rays from a point in the object plane cross at a point on the observation plane.



Defining the **magnification factor** $M = d_1/d_0$, from geometric optics, ideally the image at the observation plane would be

$$U_g(x, y) = \frac{1}{M} U_0\left(-\frac{x}{M}, -\frac{y}{M}\right).$$

However, due to the finite **pupil function** $p(x, y)$ of the lens, the field amplitude at the observation plane is [8, eqn. (5-44)]:

$$U_1(x, y) = (h ** U_g)(x, y), \quad h(x, y) = (\mathcal{F}_2^{-1} \{p(-\lambda d_1 \nu_x, -\lambda d_1 \nu_y)\})(x, y). \quad (4.12)$$

Thus, under coherent illumination, an optical system is **LSI** in complex field amplitude, and the frequency response is

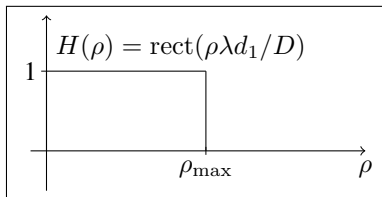
$$\boxed{H(\nu_x, \nu_y) = p(-\lambda d_1 \nu_x, -\lambda d_1 \nu_y) = p(x, y) \Big|_{x = -\lambda d_1 \nu_x, y = -\lambda d_1 \nu_y}.} \quad (4.13)$$

Ignoring aberrations, the shape of the pupil function (typically circular) determines the frequency response of the imaging system.

Usually the pupil is circularly symmetric, *i.e.*, $p(r)$, in which case the frequency response is

$$\boxed{H(\rho) = p(\lambda d_1 \rho) = p(r) \Big|_{r = \lambda d_1 \rho}.}$$

Example. For a usual circular lens: $p(r) = \text{rect}(r/D)$, where D is the lens diameter. Thus $H(\rho) = \text{rect}\left(\frac{\rho\lambda d_1}{D}\right)$.



What is the cutoff frequency? $\rho_{\max} = \boxed{??}$

[RQ]

Larger lenses pass higher frequency components. (Think astronomy.) This result requires Fourier optics to explain; geometric optics is insufficient because it disregards spot size, treating rays as infinitesimal.

Smaller wavelengths also increase cutoff frequency.

What is the PSF? $h(r) = \boxed{??}$

[RQ]

Out-of-focus imaging

If a camera is focused at infinity, *i.e.*, $d_1 = F$ in (4.11), and the scene is reasonably far from the lens (but not infinitely far), then the image will be somewhat out of focus [9] and researchers have used the “uniform disk” PSF model $h(r) = \text{rect}\left(\frac{r}{R}\right)$, where R is a parameter that depends on the geometry.

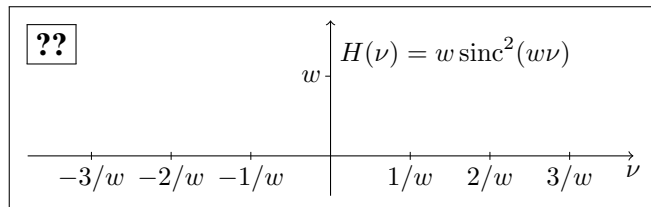
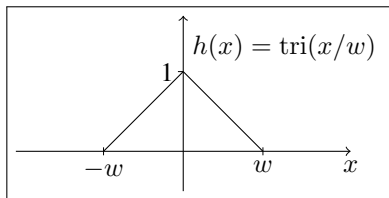
In this imaging configuration, does the lens act as an ideal low-pass filter?

[RQ] ??

Sampling images that are not band-limited

Physical objects are space-limited, so they cannot be band-limited! Aliasing is unavoidable for images that are not band-limited. A practical rule-of-thumb that is sometimes used is to use a sample spacing that is half the FWHM of the PSF.

Example. A 1D system with triangular PSF $h(x) = \text{tri}(x/w)$ having FWHM = w and frequency response $H(\nu) = w \text{sinc}^2(w\nu)$.



Here $\nu_{\max} \approx 1/w$, the location of first zero. So a reasonable sampling interval is $\Delta_x = \frac{1}{2\nu_{\max}} = w/2$, so $\Delta_x = \text{FWHM}/2$. This is not exact Nyquist sampling because $H(\nu)$ is not band-limiting, but it is a reasonable starting point in practice. The peak side-lobe of sinc^2 is about $0.047w$, but often the amount of aliasing will be less because the object spectrum usually peaks near zero and then decreases for higher $|\nu|$.

Super-resolution

One can combine multiple aliased low-resolution images to form high-resolution images through a process known as **super-resolution** [10–13]. In this application, aliasing is an important ingredient! There are also single image super-resolution methods [14], including many CNN-based methods such as that of **Entropix**.

Digital displays: Aliasing in signal reconstruction

Standard digital displays “reconstruct” an analog image from discrete-space data $g_d[m, n]$, but they do not use the ideal sinc interpolator (4.9). Typically these displays use a 2D version of “sample and hold,” which is mathematically equivalent to convolving $g_s(x, y)$ with $\frac{1}{\Delta_x \Delta_y} \text{rect}_2\left(\frac{x}{\Delta_x}, \frac{y}{\Delta_y}\right)$, where $\Delta_x \times \Delta_y$ is the pixel size. Thus the reconstructed image is

$$\begin{aligned} g_r(x, y) &= g_s(x, y) ** \frac{1}{\Delta_x \Delta_y} \text{rect}_2\left(\frac{x}{\Delta_x}, \frac{y}{\Delta_y}\right) \\ &= \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g_d[m, n] \text{rect}_2\left(\frac{x - m\Delta_x}{\Delta_x}, \frac{y - n\Delta_y}{\Delta_y}\right). \end{aligned}$$

e.g., display (Of course the sum is finite in practice.) So the spectrum of the reconstructed image is:

$$G_r(\nu_x, \nu_y) = \left[\sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} G_a(\nu_x - k/\Delta_x, \nu_y - l/\Delta_y) \right] \text{sinc}_2(\Delta_x \nu_x, \Delta_y \nu_y). \quad (4.14)$$

The figure on the next page illustrates this spectrum in 1D.

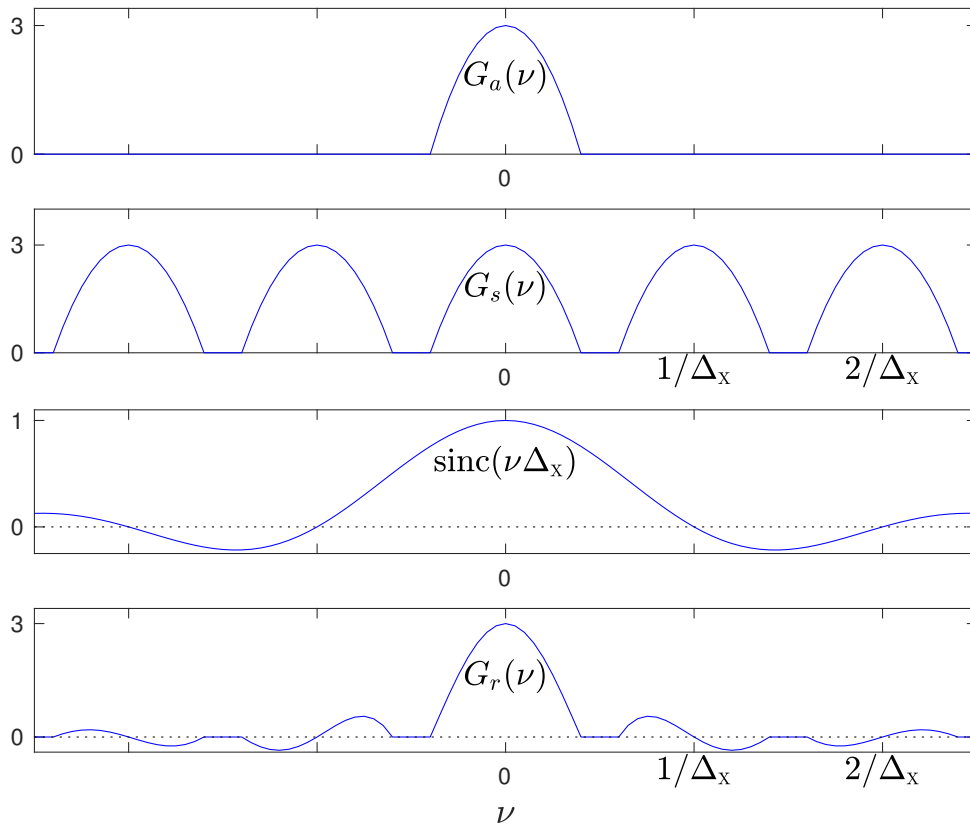
The important conclusion here is that there can be some “aliasing” (imperfect reconstruction) in such a display device, *even if the original signal is band-limited and over-sampled.*

Example. Demonstration of aliasing.

demo_alias1.m

http://web.eecs.umich.edu/~fessler/course/556/r/test_display_alias.gif

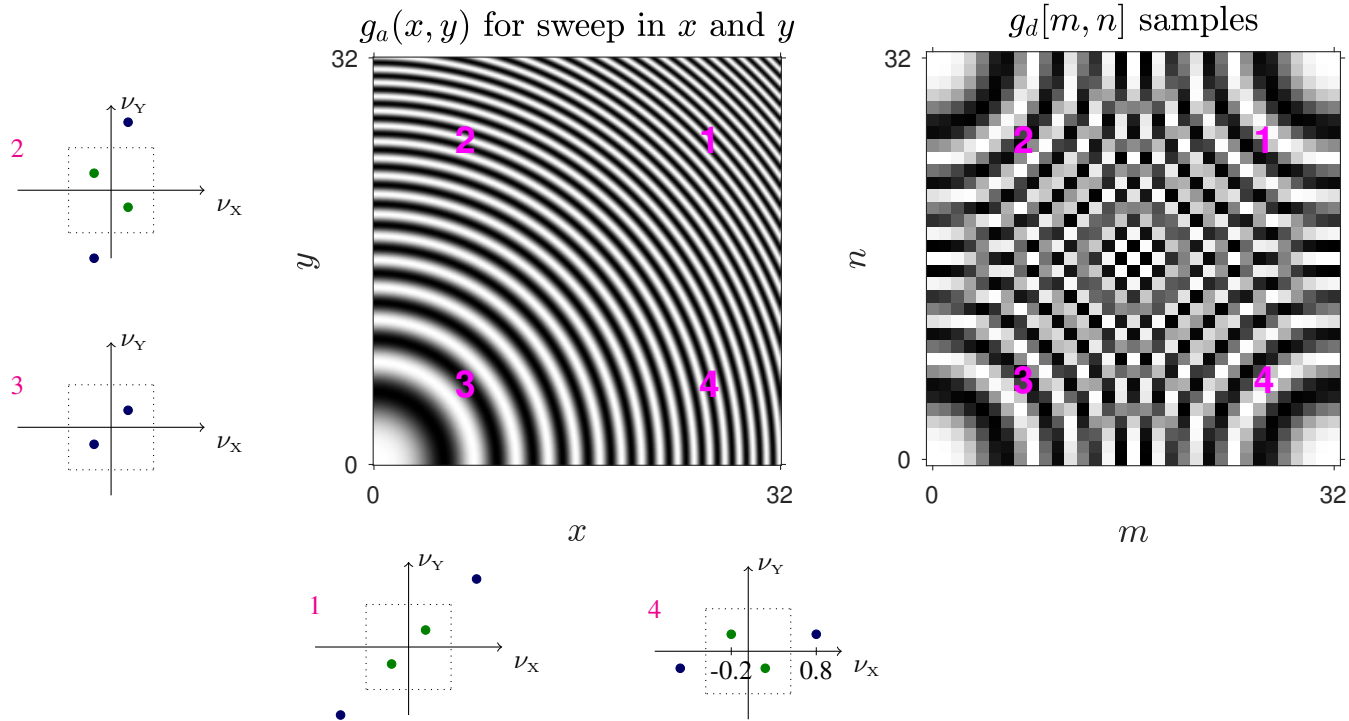
Example. The figure on the next page illustrates the spectra associated with a digital display (1D case); see (4.14).



Aliasing illustration

In the following figure, both the x and y frequency components sweep linearly from 0 to 1 cycles/m, *i.e.*, $g_a(r) = \cos(2\pi r^2/64)$. The left image is (approximately) the original signal $g_a(x, y)$ and the right image is the signal sampled with $\Delta_x = \Delta_y = 1$ m.

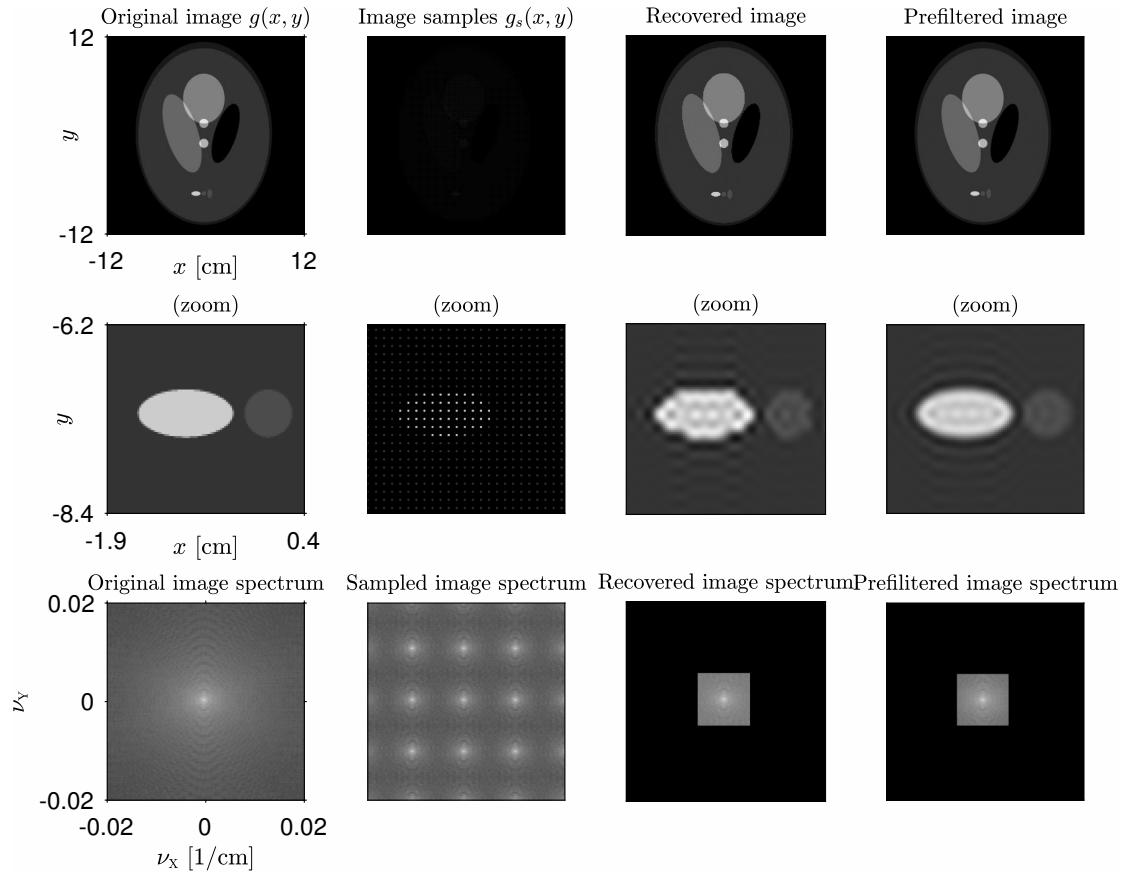
In the local spectra, the dashed lines mark the ± 0.5 cycles/m that corresponds the Nyquist limit. Only the spectral components in the central box are adequately sampled. Blue dots correspond to the true spatial frequencies. The green dots show the spectral replicates after sampling. Only the quadrant 3 has no aliasing. After sampling, the high spatial frequencies in quadrant 3 alias to the same components as in quadrant 1. The upper right and lower left also appear to have the same frequencies, but with altered orientation. This change in orientation due to aliasing is new in 2D compared to 1D.

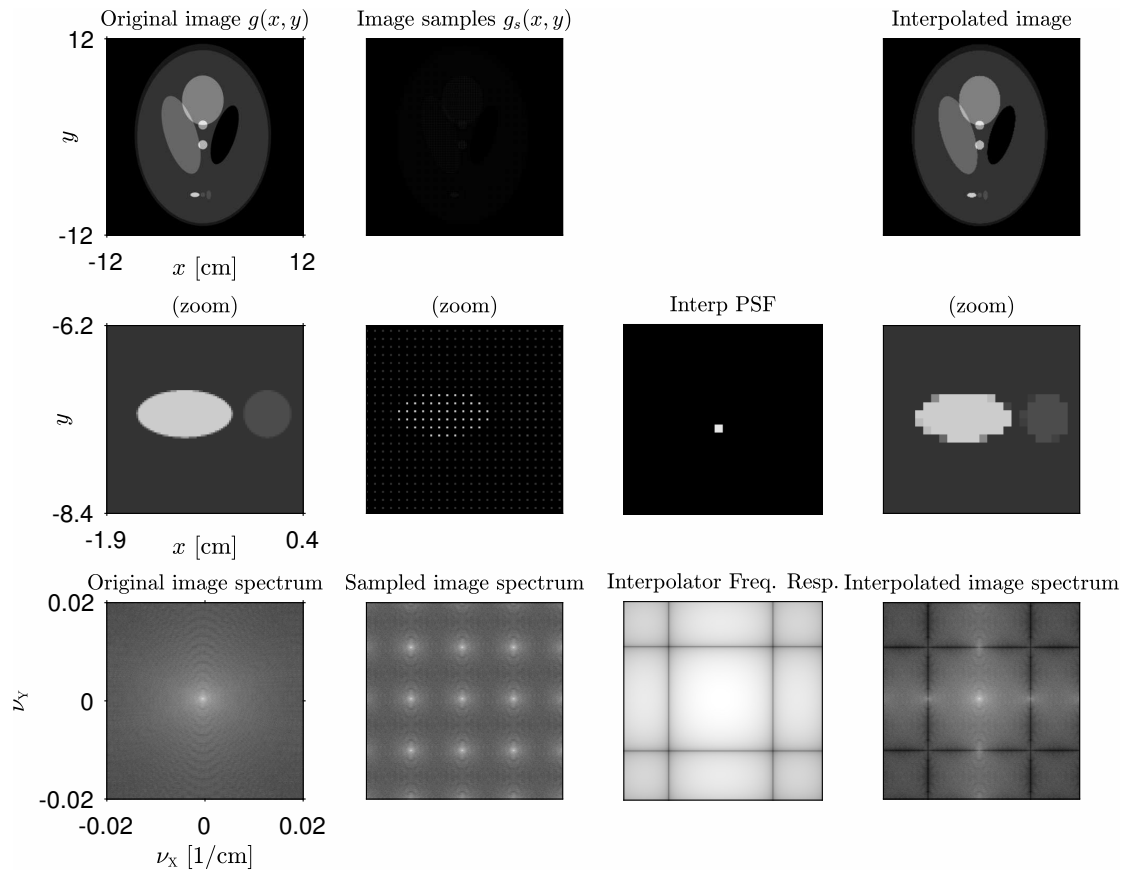


Sampling, aliasing and reconstruction illustration

Example. The following figure illustrates sampling and the effect of band-limiting an image before sampling. The first column shows an image $g(x, y)$, a zoom into a small part of the image, and its spectrum $G(\nu_x, \nu_y)$. The second column illustrates the synthetic sampled signal $g_s(x, y)$ and its spectrum. This signal is not bandlimited, so if we extract the central part of the spectrum and perform an inverse Fourier transform, we get the image shown in the 3rd column that suffers from aliasing artifacts. On the other hand, if we prefilter the image by an ideal (square) anti-alias filter, as illustrated in the 4th column, then we recover an image that has ringing due to the $\text{sinc}_2(i, m)$ pulse response but does not suffer from aliasing.

The subsequent figure shows the spectrum of an image that was interpolated using a **sample-and-hold** or **nearest-neighbor** interpolator. Clearly the original signal spectrum is not recovered perfectly.





4.4 Summary

- Shannon’s sampling theory generalizes readily from 1D to 2D.
- The main difference is the following: for cases where the signal is band-limited to a region of 2D Fourier space that is smaller than the rectangle $[-\nu_X^{\max}, \nu_X^{\max}] \times [-\nu_Y^{\max}, \nu_Y^{\max}]$ there is a “larger” family of interpolators (such as the jinc interpolator) than in 1D problems.
- The possible shapes of the spectral support in 2D lead to interesting alternative sampling patterns.
- This generality has been exploited successfully in dynamic imaging problems (2D + time and 3D + time) [7].
- There have also been extensions of 2D rectilinear sampling theory to uniform sampling in polar coordinates [15], and to other sampling patterns including Manhattan grids [16].
- Pixels shaped like hexagons are used commercially, e.g., for **digital mammography**. Examining hexagonal sampling is a HW problem.
- The “**Super CCD**” uses octagonal pixels.

What part of this chapter did you find most confusing? ??

[RQ]

Bibliography

- [1] B. A. Wandell, A. E. Gamal, and B. Girod. “Common principles of image acquisition systems and biological vision”. In: *Proc. IEEE* 90.1 (Jan. 2002), 5–17.
- [2] J. S. Lim. *Two-dimensional signal and image processing*. New York: Prentice-Hall, 1990. ISBN: 978-0139353222.
- [3] W. Sun and X. Zhou. “Reconstruction of band-limited signals from local averages”. In: *IEEE Trans. Info. Theory* 48.11 (Nov. 2002), 2955–63.
- [4] T. G. Dvorkind, Y. C. Eldar, and E. Matusiak. “Nonlinear and nonideal sampling: theory and methods”. In: *IEEE Trans. Sig. Proc.* 56.12 (Dec. 2008), 5874–90.
- [5] Y. C. Eldar and T. Michaeli. “Beyond bandlimited sampling”. In: *IEEE Sig. Proc. Mag.* 26.3 (May 2009), 48–68.

- [6] R. G. Baraniuk. “Compressive sensing”. In: *IEEE Sig. Proc. Mag.* 24.4 (2007), 118–21.
- [7] N. P. Willis and Y. Bresler. “Lattice-theoretic analysis of time-sequential sampling of spatiotemporal signals. II. Large space-bandwidth product asymptotics”. In: *IEEE Trans. Info. Theory* 43.1 (Jan. 1997), 208–20.
- [8] J. W. Goodman. *Introduction to Fourier optics*. New York: McGraw-Hill, 1968. ISBN: 978-0974707723.
- [9] J. P. Oliveira, M. A. T. Figueiredo, and J. M. Bioucas-Dias. “Parametric blur estimation for blind restoration of natural images: linear motion and out-of-focus”. In: *IEEE Trans. Im. Proc.* 23.1 (Jan. 2014), 466–77.
- [10] S. Farsiu, M. D. Robinson, M. Elad, and P. Milanfar. “Fast and robust multiframe super resolution”. In: *IEEE Trans. Im. Proc.* 13.10 (Oct. 2004), 1327–44.
- [11] S. Farsiu, D. Robinson, M. Elad, and P. Milanfar. “Advances and challenges in super-resolution”. In: *Intl. J. Imaging Sys. and Tech.* 14.2 (2004), 47–57.
- [12] S. Farsiu, M. Elad, and P. Milanfar. “Multiframe demosaicing and super-resolution of color images”. In: *IEEE Trans. Im. Proc.* 15.1 (Jan. 2006), 141–59.
- [13] J. Yang, J. Wright, T. S. Huang, and Y. Ma. “Image super-resolution via sparse representation”. In: *IEEE Trans. Im. Proc.* 19.11 (Nov. 2010), 2861–73.
- [14] Y. Romano, J. Isidoro, and P. Milanfar. “RAISR: rapid and accurate image super resolution”. In: *IEEE Trans. Computational Imaging* 3.1 (Mar. 2017), 110–25.
- [15] H. Stark. “Sampling theorems in polar coordinates”. In: *J. Opt. Soc. Am.* 69.11 (Nov. 1979), 1519–25.
- [16] M. Prelee and D. L. Neuhoff. “A sampling theorem for Manhattan grids”. In: *Proc. IEEE Conf. Acoust. Speech Sig. Proc.* 2012, 3797–800.

Chapter 5

2D discrete-space signals and systems

Contents (class version)

| | |
|---|-------------|
| 5.0 Introduction | 5.2 |
| 5.1 Discrete-space signals | 5.4 |
| Classification of discrete-space signals | 5.9 |
| 5.2 2D discrete-space systems | 5.17 |
| Classification of systems | 5.21 |
| LSI systems | 5.29 |
| 5.3 2D convolution | 5.30 |
| 5.4 2D discrete-space Fourier transform | 5.42 |
| Properties of the DSFT | 5.47 |
| Transforms of elementary functions | 5.52 |
| Sampling revisited with the DSFT | 5.55 |
| Methods for inverse DSFT | 5.58 |
| Frequency-domain characteristics of LSI systems | 5.64 |
| 5.5 Filter design introduction | 5.66 |
| 5.6 Summary | 5.78 |

5.0 Introduction

Using optical devices like lenses, gratings, transparencies, etc., one can perform a very wide variety of *real-time* analog image processing operations. However, there are many benefits of *digital* approaches to image processing, the most important of which is *flexibility*. (It is usually easier to reprogram an image processing algorithm than to change the design of an optical system.)

Having addressed the issues of image formation and sampling in previous sections, we now turn to the world of **discrete-space (DS)** images: $g[m, n]$ for $m, n \in \mathbb{Z}$. A discrete-space image is defined *only* on the integers. (It is *incorrect* to think of $g[m, n]$ as being zero for non-integer values of m, n ; instead $g[m, n]$ is *undefined* for non-integer values.)

In this chapter we start to consider the middle box of the following type of imaging system:



For this chapter we continue to consider **continuous-valued** (but **discrete-space**) 2D signals, saving issues of quantization for later. Nevertheless, we will still often call $g[m, n]$ a **digital image** for simplicity.

Our first goal is a brief review of 2D discrete-space signals and systems. (Essentially EECS 451 in 2D.)

References: [1, Ch. 1], [2].

*The majority of this chapter very straightforward extensions of 1D DSP to 2D signals, so will be reviewed quickly. The primary new concepts (compared to 1D DSP) are **separable signals, separable systems**, subtleties about **circular symmetry** and **rotation invariance**, and edge effects (boundary conditions) for 2D convolution.*

Overview

- **2D discrete-space signals** (digital images)
 - Kronecker impulse function
 - Notation and coordinate systems
 - Signal classes: even, odd, periodic, *separable*, *rotational symmetry*, *circular symmetry*
- **2D discrete-space systems**
 - Simple operations: down-sampling, up-sampling
 - System classes: linear, stable, invertible, causal, static, shift-invariant, *separable*, *rotation invariant*
 - Impulse response
 - Linear shift-invariant (**LSI**) systems
 - **convolution sum**, cross-correlation sum, properties of 2D convolution.
 - LSI System properties in terms of impulse response
- **2D discrete-space Fourier transform** (DSFT)
 - eigenfunctions of 2D DS LSI systems
 - DSFT properties (convolution etc.).
 - Sampling and DSFT vs FT
 - Magnitude and phase of DSFT
 - Frequency response of 2D LSI systems
- **2D filter design**

What (two) major topics from 1D DSP (related to DS systems) are absent?

[RQ]

- ??
- ??

This review emphasizes similarities and *differences* between 1D and 2D. This treatment reinforces signals and systems concepts learned previously, while also introducing a few new concepts and properties that are unique to 2D (and higher) problems.

5.1 Discrete-space signals

Unit impulse function

The 2D **Kronecker impulse function** or **delta function** is defined by:

$$\delta_2[m, n] \triangleq \begin{cases} 1, & m = 0, n = 0 \\ 0, & \text{otherwise.} \end{cases}$$

Unlike the Dirac impulse, which is not a function, the Kronecker impulse really is a (discrete-space) function: $\delta_2 : \mathbb{Z}^2 \mapsto \{0, 1\}$.

Properties:

- **Unity sum:** $\sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \delta_2[m, n] = 1$, which is analogous to the unity integral property of the Dirac impulse.
- **Sampling property:** $\delta_2[m - m_0, n - n_0] f[m, n] = \delta_2[m - m_0, n - n_0] f[m_0, n_0]$
- **Sifting property:** $\sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \delta_2[m - m_0, n - n_0] f[m, n] = f[m_0, n_0]$
- **Symmetry property:** $\delta_2[-m, -n] = \delta_2[m, n]$

- **Scaling property:** $\delta_2[2m, 2n] = \boxed{??}$

[RQ]

Note that this property is quite different from that of the **Dirac impulse**.

Should we discuss a scaling property for “ $\delta_2[m/2, n/2]$ ” **No, it is meaningless.**

We will discuss this issue in more detail later.

- **Separability property:** $\delta_2[m, n] = \delta[m] \delta[n]$, where $\delta[n]$ is the 1D Kronecker impulse function.

In the literature one often sees identical or almost identical notation for the Kronecker delta function $\delta_2[m, n]$ and the Dirac impulse $\delta_2(x, y)$. The reader must determine from context which form is meant (*i.e.*, whether the arguments are discrete or continuous).

MATLAB's "anonymous" functions are handy for creating the 2D **Kronecker** impulse function:

```
delta = @(m,n) (m==0) & (n==0);
disp([delta(0,0) delta(0,1) delta(-7,9) ])
```

The 2D **unit-step** image is given by

$$\text{step}_2[m, n] \triangleq \begin{cases} 1, & m \geq 0, n \geq 0 \\ 0, & \text{otherwise} \end{cases} = \sum_{k=0}^{\infty} \sum_{l=0}^{\infty} \delta_2[m - k, n - l].$$

We can also write

$$\text{step}_2[m, n] = \text{step}[m] \text{step}[n], \text{ where } \text{step}[n] \triangleq \begin{cases} 1, & n \geq 0 \\ 0, & \text{otherwise.} \end{cases}$$

Note that $\delta[n] = \text{step}[n] - \text{step}[n - 1]$ and

$$\delta_2[m, n] = (\text{step}_2[m, n] - \text{step}_2[m - 1, n]) (\text{step}_2[m, n] - \text{step}_2[m, n - 1]),$$

which is analogous to the 2D continuous-space (**CS**) formula $\delta_2(x, y) = \frac{\partial^2}{\partial x \partial y} \text{step}_2(x, y) = \left(\frac{\partial}{\partial x} \text{step}(x)\right) \left(\frac{\partial}{\partial y} \text{step}(y)\right)$.

Furthermore, in DS the following equality holds:

$$\delta_2[m, n] = \text{step}_2[m, n] - \text{step}_2[m - 1, n] - \text{step}_2[m, n - 1] + \text{step}_2[m - 1, n - 1].$$

Discrete-space sinusoidal signals

2D DS sinusoids:

$$A \cos(\Omega_1 m + \Omega_2 n + \theta)$$

2D DS complex exponential signals:

$$e^{i(\Omega_1 m + \Omega_2 n)}$$

What are the units of Ω_1 and Ω_2 ? ??

[RQ]

We sometimes refer to Ω_1 and Ω_2 as “digital frequencies.”**Signal notation**

Here are three equivalent methods for denoting the 2D discrete-space signal that is zero everywhere except is unity in the 3 by 3 neighborhood of the origin:

$$f[m, n] = \begin{cases} 1, & |m| \leq 1, |n| \leq 1 \\ 0, & \text{otherwise} \end{cases} = \sum_{k=-1}^1 \sum_{l=-1}^1 \delta_2[m - k, n - l] = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

All three expressions are useful. The underline denotes the origin where $[m, n] = [0, 0]$.

Coordinate systems

For 1D signals, by convention we almost always display negative values of t or n to the left, and positive values to the right.

For 2D signals, there is more variability in how the two coordinates are used and displayed, depending on the situation.

For theoretical analysis, it is usually convenient to consider an infinite lattice coordinate system centered at the origin $(0, 0)$, with the first index, say m increasing from left to right, and the second index, say n increasing from bottom to top, as follows:

$$\begin{array}{c} n \\ \uparrow \\ \leftarrow m \end{array} \quad f[m, n] = \begin{bmatrix} \ddots & & \vdots & & \\ & f[-1, 1] & f[0, 1] & f[1, 1] & \\ \dots & f[-1, 0] & \underline{f[0, 0]} & f[1, 0] & \dots \\ & f[-1, -1] & f[0, -1] & f[1, -1] & \\ & & \vdots & & \ddots \end{bmatrix}. \quad (5.1)$$

This convention is particularly natural when we consider the values $f[m, n]$ to be samples of a continuous-space (**CS**) signal $f(x, y)$, because it matches the usual Cartesian coordinates (x, y) .

These notes will use the convention (5.1) whenever a 2D DS signal is displayed in the text.

Example.

$$\begin{bmatrix} 0 & 0 & 7 \\ 2 & \underline{3} & 0 \\ 0 & 4 & 0 \end{bmatrix} = 7\delta_2[m-1, n-1] + 2\delta_2[m+1, n] + 3\delta_2[m, n] + 4\delta_2[m, n+1].$$

However, when a digital image is stored in a computer, only a finite number of samples can be stored. In MATLAB and Fortran, the usual indexing for a $M \times N$ 2D **array** \mathbf{A} is as follows:

$$\mathbf{A} = \begin{bmatrix} a_{11} & \dots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{M1} & \dots & a_{MN} \end{bmatrix}.$$

The relationship between array indexes and image indexes is entirely arbitrary, and up to the discretion of the programmer. Whenever FFT operations are going to be used, the following relationship is often appropriate:

$$a_{ij} = f(i - M/2 - 1, j - N/2 - 1)$$

(assuming that M and N are even), so that the $(0, 0)$ value of the image is stored in the $M/2 + 1, N/2 + 1$ location of the array, which is just the right place so that `fftshift` will relocate it to the $1, 1$ array position. (See Ch. 7.)

Unfortunately, MATLAB's `imagesc` command displays the array columns as image columns, even though the column index is more naturally associated with the x coordinate. So it is often helpful to include a `transpose` operation when calling `imagesc` in MATLAB. There are also commands like `axis image`, `axis xy`, and `axis ij` in MATLAB to deal with the variety of options for coordinate systems.

Of course in programming languages like ANSI C the array indices begin from 0 rather than 1. And in fact it is often most efficient to store 2D images as a 1D array rather than using double indexing in C.

In summary, one must always think carefully about relationships between array indices and image coordinates when programming! There is no single “right answer” but inconsistencies will usually cause errors.

Classification of discrete-space signals

The **energy** of a 2D DS signal is defined as

$$E_g \triangleq \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} |g[m, n]|^2.$$

The **average power** of a 2D signal is defined as

$$P_g \triangleq \lim_{N \rightarrow \infty} \frac{1}{(2N+1)^2} \sum_{m=-N}^N \sum_{n=-N}^N |g[m, n]|^2.$$

- If E is finite ($E < \infty$) then $g[m, n]$ is called an **energy signal** and $P = 0$.
- If E is infinite, then P can be either finite or infinite. If P is finite and nonzero, then $g[m, n]$ is called a **power signal**.
- More generally, we compute the energy of a signal only over its domain, which could be a subset of \mathbb{Z}^2 .

Periodic signals

A 2D discrete-space signal $g[m, n]$ is **periodic** with **period** (M, N) if and only if

$$g[m, n] = g[m + M, n] = g[m, n + N], \forall m, n \in \mathbb{Z}.$$

Again we forgo any attempt to define a **fundamental period** for 2D DS periodic signals.

To find a period of $g[m, n] = \cos(\Omega_1 m + \Omega_2 n + \theta)$, write $\Omega_1/2\pi = k/M$ where k and M are integers with no common divisors, and likewise write $\Omega_2/2\pi = l/N$. Then (M, N) will be a period of $g[m, n]$. If no such ratios exist, then $g[m, n]$ is **aperiodic**.

Big differences compared to continuous-space sinusoids:

- A DS sinusoidal signal $g[m, n] = \cos(\Omega_1 m + \Omega_2 n + \theta)$ is periodic if and only if $\Omega_1/2\pi$ and $\Omega_2/2\pi$ are **rational**.
In contrast, a CS sinusoidal signal $\cos(2\pi(\nu_x x + \nu_y y) + \theta)$ is periodic for *any* values ν_x and ν_y .
- Two DS sinusoidal signals with frequencies separated by integer multiples of 2π are identical (indistinguishable).
For example, $g[m, n] = \cos(\Omega m)$ and $f[m, n] = \cos((\Omega + 2\pi)m + 4\pi n)$ are identical. This is a form of **aliasing**.
For such reasons, when considering sinusoids, we usually focus on frequencies in the interval $[0, 2\pi)$ or $[-\pi, \pi)$.

If $g[m, n]$ is (M, N) periodic, then it is a power signal with power $P_g = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} |g[m, n]|^2$.

Separability

A DS signal $g[m, n]$ is **separable** iff we can write

$$g[m, n] = g_1[m] g_2[n].$$

Example. $\delta_2[m, n] = \delta[m] \delta[n]$

Example. $\text{step}_2[m, n] = \text{step}[m] \text{step}[n]$

To check if a finite-support image is separable, simply verify that every row is a multiple of a single row.

Symmetry

- $g[m, n]$ is **symmetric** or **even** iff $g[-m, -n] = g[m, n]$
- $g[m, n]$ is **antisymmetric** or **odd** iff $g[-m, -n] = -g[m, n]$
- $g[m, n]$ is **Hermitian symmetric** iff $g[-m, -n] = g^*[m, n]$

We can decompose any signal into even and odd components:

$$\begin{aligned}g[m, n] &= \text{Ev} \{g[m, n]\} + \text{Od} \{g[m, n]\} \\ \text{Ev} \{g[m, n]\} &\triangleq \frac{1}{2} (g[m, n] + g[-m, -n]) \\ \text{Od} \{g[m, n]\} &\triangleq \frac{1}{2} (g[m, n] - g[-m, -n])\end{aligned}$$

One can also define some types of n -fold **rotational symmetry** but they are used less for DS signals.

Circular symmetry

In continuous-space analysis, we said a signal $g(x, y)$ has **circular symmetry** iff it is a function solely of the square root of the sum-of-squares of its arguments, *i.e.*, $g(x, y) = g_R\left(\sqrt{x^2 + y^2}\right)$ for some function $g_R : [0, \infty) \mapsto \mathbb{C}$.

An equivalent definition is to say that $g(x, y)$ has **circular symmetry** iff $g(x, y)$ is *invariant to rotations*, *i.e.*, iff

$$g(x, y) = g(x \cos \theta + y \sin \theta, -x \sin \theta + y \cos \theta), \quad \forall x, y, \theta. \quad (5.2)$$

For discrete-space signals, there is no canonical rotation formula. In fact, MATLAB's `imrotate` command has several different options (for interpolation). Furthermore, although MATLAB's `imrotate` command can “rotate” a DS image by any specified angle, if you rotate an image (by any angle other than a multiple of $\pi/2$) and then rotate it back, you will not get back the original image exactly, due to the interpolations made in the rotation process. This property can range from being a minor inconvenience to being a serious nuisance in various applications.

Nevertheless, it seems desirable to have some definition of circular symmetry for DS signals, because rotation invariance is a very desirable property in many (if not most) applications. I propose the following two definitions. I have never seen them in any textbooks, but I do not claim that they are new. They are natural definitions, so it seems likely that someone will have thought about similar things before, but for whatever reason they have never made it into the image processing books that I have seen.

Definition. I will say that a DS signal $g[m, n]$ is **weak-sense circularly symmetric** iff it satisfies

$$g[m, n] = g_R\left(\sqrt{m^2 + n^2}\right), \quad (5.3)$$

for some function $g_R(r) : [0, \infty) \mapsto \mathbb{C}$. Equivalently, $g[m, n]$ corresponds to samples of some CS circularly symmetric image.

Example. Sampled-disk signals of the form $g[m, n] = \text{rect}\left(\frac{\sqrt{m^2+n^2}}{d_0}\right)$ are weak-sense circularly symmetric.

The DS impulse signal $g[m, n] = \delta_2[m, n]$ also falls into this class by choosing any $0 < d_0 < 2$.

The reason I call this “weak sense,” is that such DS signals are not necessarily invariant to sinc-based rotations, *i.e.*, they will not (exactly) satisfy a condition analogous to (5.2).

Definition. I will say a DS signal $g[m, n]$ is **strong-sense circularly symmetric** iff $g[m, n]$ satisfies (5.3) for a function $g_R(r)$ whose Hankel transform is zero for $\rho > 1/2$, i.e., an appropriately bandlimited 2D circularly symmetric analog image.

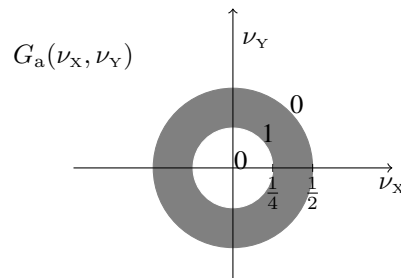
This definition is equivalent to requiring that if we form an analog signal from $g[m, n]$ by jinc-based interpolation, rotate that signal by any angle θ , and then resample the result, then we get back the original $g[m, n]$; this property is analogous to (5.2).

Given a 2D DS signal $g[m, n]$, how can we determine whether it is strong-sense circularly symmetric? In principle, one could verify that the following equality holds for all $m, n \in \mathbb{Z}$ and for all θ :

$$\begin{aligned} g[m, n] &= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} g[k, l] \text{jinc}\left(\sqrt{(x-k)^2 + (y-l)^2}\right) \Bigg|_{\substack{x = m \cos \theta + n \sin \theta \\ y = -m \sin \theta + n \cos \theta}} \\ &= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} g[k, l] \text{jinc}\left(\sqrt{(m \cos \theta + n \sin \theta - k)^2 + (-m \sin \theta + n \cos \theta - l)^2}\right). \end{aligned}$$

The right-hand expression contains jinc-interpolation (using $\Delta_x = \Delta_y = 1$), continuous-space rotation, then resampling.

Example. The signal $g[m, n] = \text{jinc}(\sqrt{m^2 + n^2}) - \frac{1}{4} \text{jinc}(\frac{1}{2}\sqrt{m^2 + n^2})$ is strong-sense circularly symmetric, because it corresponds to samples of the bandlimited circularly symmetric function $g_a(x, y) = \text{jinc}(r) - \frac{1}{4} \text{jinc}(r/2)$ with spectrum $G_a(\nu_x, \nu_y) = \text{rect}(\rho) - \text{rect}(2\rho)$, which has maximum frequency $\rho_{\max} = 1/2$.



Challenge. Is $g[m, n] = \delta_2[m, n]$ strong-sense circularly symmetric?

Example. The figure below provides an example that uses jinc-based interpolation to rotate (by 60°) and resample two types of images: a circ function, and a jinc function.

The 3rd row shows the error images after rotation. As expected, there is much more error with the sampled circ function.

Why? ??

However, the error is small in both cases, interestingly. Something like samples of $g_R(r) = \text{rect}\left(\frac{r-10}{0.4}\right)$ (a thin ring) would presumably show more dramatically the problem with attempting to rotate samples of a non-bandlimited image.

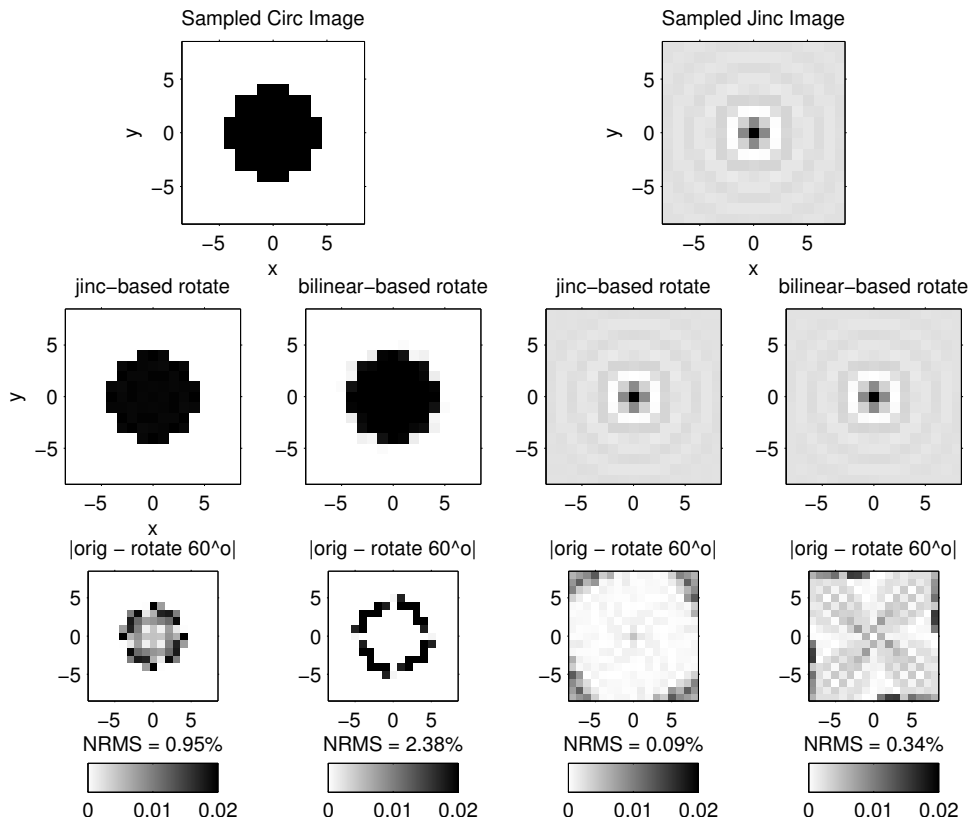
What causes the error when rotating the circ function? **It is not band-limited.**

What causes the error when rotating the jinc function? **It has been truncated in the computation.**

In the figure, “NRMSE” stands for **normalize root mean-squared error**, defined as

$$\text{NRMSE} \triangleq \frac{\sum_{m,n} |f[m,n] - g[m,n]|^2}{\sum_{m,n} |f[m,n]|^2} \cdot 100\%,$$

where $f[m,n]$ is the “original” image and $g[m,n]$ is the “corrupted” image (the rotated one here).



5.2 2D discrete-space systems

A 2D discrete-space (DS) system, or just **system**, operates on a 2D DS signal called the **input image** to form a 2D DS signal called the **output image**.

$$f[m, n] \rightarrow \boxed{\text{System } \mathcal{S}} \rightarrow g[m, n].$$

Notation options:

$$g = \mathcal{S}\{f\}, \quad g[m, n] = \mathcal{S}\{f[m, n]\}, \quad g[m, n] = (\mathcal{S}\{f\})[m, n], \quad f[m, n] \xrightarrow{\mathcal{S}} g[m, n].$$

Example. 2D DS **moving average** over a 3×3 **window** or **neighborhood**:

$$g[m, n] = \frac{1}{9} \sum_{k=m-1}^{m+1} \sum_{l=n-1}^{n+1} f[k, l] = \frac{1}{9} \sum_{k=-1}^1 \sum_{l=-1}^1 f[m-k, n-l]. \quad (5.4)$$

Example. Image **segmentation** based on a simple **threshold** (7 in this case):

$$g[m, n] = \begin{cases} 1, & f[m, n] > 7 \\ 0, & \text{otherwise.} \end{cases} \quad (5.5)$$

Example. The **down-sampling** operation¹ (trivial form of image compression):

$$g[m, n] = f_{\downarrow 2}[m, n] \triangleq f[2m, 2n] = \begin{bmatrix} \ddots & & & \vdots & & & \\ & f[-2, 2] & f[0, 2] & f[2, 2] & & & \\ \dots & f[-2, 0] & f[0, 0] & f[2, 0] & \dots & & \\ & f[-2, -2] & f[0, -2] & f[2, -2] & & & \\ & & \vdots & & & \ddots & \end{bmatrix}. \quad (5.6)$$

Example. The **up-sampling** operation (by zero insertion):

$$g[m, n] = \begin{cases} f[m/2, n/2], & m \text{ even and } n \text{ even} \\ 0, & \text{otherwise} \end{cases} = \begin{bmatrix} \ddots & & & \vdots & & & \\ & 0 & 0 & 0 & 0 & 0 & 0 \\ & 0 & f[-1, 1] & 0 & f[0, 1] & 0 & f[1, 1] & 0 \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \dots & 0 & f[-1, 0] & 0 & \underline{f[0, 0]} & 0 & f[1, 0] & 0 & \dots \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & 0 & f[-1, -1] & 0 & f[0, -1] & 0 & f[1, -1] & 0 \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & & \vdots & & & & \ddots \end{bmatrix}. \quad (5.7)$$

The expression “ $g[m, n] = f[m/2, n/2]$ ” is mathematically meaningless because $f[m, n]$ is undefined for non-integer values! In contrast, the bracketed expression in (5.7) is perfectly rigorous, because $m/2$ and $n/2$ are invoked only for even values of m, n .

¹ Also called **subsampling** or sometimes **decimation**, a term that originated in the Roman punishment consisting of killing one in ten soldiers [3, p. 66].

Often the following notation is used to indicate up-sampling (by zero insertion):

$$f_{\uparrow 2}[m, n] \triangleq \begin{cases} f[m/2, n/2], & m \text{ even and } n \text{ even} \\ 0, & \text{otherwise.} \end{cases} \quad (5.8)$$

Sometimes the following expression can be helpful in derivations because it has no “braces” in it:

$$f_{\uparrow 2}[m, n] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] \delta_2[m - 2k, n - 2l]. \quad (5.9)$$

There are other mathematically rigorous ways to implement **up-sampling**, such as using **replication**:

$$f\left[\left\lfloor \frac{m}{2} \right\rfloor, \left\lfloor \frac{n}{2} \right\rfloor\right] = \begin{cases} f\left[\frac{m}{2}, \frac{n}{2}\right], & m \text{ even, } n \text{ even} \\ f\left[\frac{m-1}{2}, \frac{n}{2}\right], & m \text{ odd, } n \text{ even} \\ f\left[\frac{m}{2}, \frac{n-1}{2}\right], & m \text{ even, } n \text{ odd} \\ f\left[\frac{m-1}{2}, \frac{n-1}{2}\right], & m \text{ odd, } n \text{ odd} \end{cases} = \begin{bmatrix} \ddots & & & \vdots & & & & \\ & f[-1, 1] & f[-1, 1] & f[0, 1] & f[0, 1] & f[1, 1] & f[1, 1] & \\ & f[-1, 1] & f[-1, 1] & f[0, 1] & f[0, 1] & f[1, 1] & f[1, 1] & \\ & f[-1, 0] & f[-1, 0] & f[0, 0] & f[0, 0] & f[1, 0] & f[1, 0] & \\ \dots & f[-1, 0] & f[-1, 0] & f[0, 0] & f[0, 0] & f[1, 0] & f[1, 0] & \dots \\ & f[-1, -1] & f[-1, -1] & f[0, -1] & f[0, -1] & f[1, -1] & f[1, -1] & \\ & f[-1, -1] & f[-1, -1] & f[0, -1] & f[0, -1] & f[1, -1] & f[1, -1] & \\ & & & \vdots & & & & \\ & & & & & & & \ddots \end{bmatrix}, \quad (5.10)$$

where $\lfloor \cdot \rfloor$ denotes the **floor** function.

Practical implementation

Implementing 2D down-sampling in MATLAB requires just one command:

```
g = f(1:2:end,1:2:end);
```

The code in JULIA looks even more like the math thanks to the square braces:

```
g = f[1:2:end,1:2:end]
```

However, to avoid aliasing it is often better to low-pass filter $f[m, n]$ prior to down-sampling. This will be discussed further later in Ch. 6.

To implement 2D up-sampling (by zero insertion) in MATLAB requires just two commands:

```
g = zeros(2*size(f), class(f)); g(1:2:end,1:2:end) = f;
```

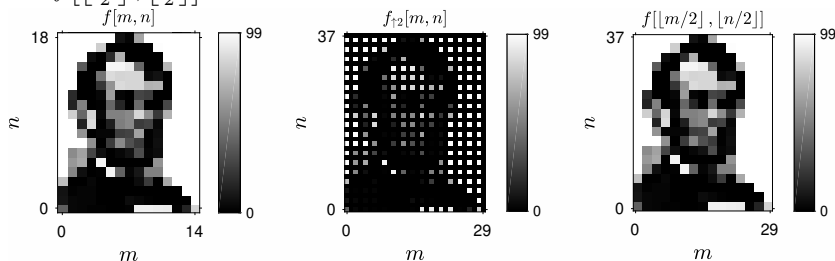
In JULIA:

```
g = zeros(eltype(f), 2*collect(size(f)...)); g[1:2:end,1:2:end] = f]
```

Usually one follows zero insertion with some filtering operation to replace the zeros with more useful values. See Ch. 8.

Example. The following figure illustrates the two types of upsampling operations.

Think about why $f[m, n]$ and $f[\lfloor \frac{m}{2} \rfloor, \lfloor \frac{n}{2} \rfloor]$ “look” almost the same.



Classification of systems

We can organize system characteristics into the following two categories.

- Amplitude properties
 - linearity
 - stability
 - invertibility
- Spatial properties
 - causality
 - separability
 - memory
 - shift invariance
 - rotation invariance

We will save linearity (the most important?) for last.

Stability

- A system is **bounded-input bounded-output (BIBO) stable** iff every bounded input produces a bounded output.
 $\forall f$, if $\exists M_f$ s.t. $|f[m, n]| \leq M_f < \infty \forall m, n$, then there must exist an M_g s.t. $|g[m, n]| \leq M_g < \infty \forall m, n$.
Usually M_g will depend on M_f .
- Otherwise the system is called **unstable**, and it is possible that a bounded input signal will make the output “blow up.”

Example. The down-sampler and up-sampler are both BIBO **stable**.

Is the moving average filter (5.4) BIBO stable?

[RQ]

Example. Consider the following recursively defined filter: $g[m, n] = g[m - 1, n - 1] + f[m, n]$. This system is **unstable** because the response to a unit-step input signal is an unbounded ramp.

We will derive a simple test for BIBO stability shortly that applies to LSI systems (but not more generally).

Invertibility

- A system \mathcal{S} is called **invertible** iff each (possible) output signal is the response to only *one* input signal.
- Otherwise \mathcal{S} is not invertible.

If a system \mathcal{S} is invertible, then there exists a system \mathcal{S}^{-1} such that for any input signal $f[m, n]$:

$$f[m, n] \rightarrow \boxed{\mathcal{S}} \rightarrow g[m, n] \rightarrow \boxed{\mathcal{S}^{-1}} \rightarrow f[m, n].$$

Design of \mathcal{S}^{-1} is important in many image processing applications.

Mathematically:

$$\mathcal{S}^{-1}[\mathcal{S}[f]] = f.$$

Invertibility is a particularly important concept in imaging because often we would like to “undo” degrading effects like blurring.

Example. The 3×3 moving average filter (5.4) is not invertible because the response to the (nonzero) input signal $f[m, n] = \cos(2\pi n/3)$ is zero, as is the response to $f[m, n] = 0$.

Is the down-sampling operation (5.6) invertible?

[RQ]

Is the up-sampling operation (5.8) invertible?

[RQ]

Causal systems

(skim)

The concept of causality is very important in 1D signal processing (when the independent variable usually corresponds to time samples). In 2D signal processing, the independent variables usually correspond to spatial positions, not time, so causality is usually unimportant. (Of course in “real time” systems like video, where the independent variables are both space *and* time, one must consider causality with respect to the time variable.)

There are some (mostly older) papers in the image processing literature that address raster-scan based image processing methods, where the image is filtered, for example, by starting in the upper left hand corner and working lexicographically down towards the lower right hand corner. Such processing could be described as “causal” (whereas the moving average filter above would be “noncausal” by such a definition).

Separable systems

A system is called **separable** if it first acts independently on each image column, and then independently on each row, or vice versa. Separable operations are collections of 1D actions, so are computationally attractive.

Example. The 3×3 moving average (5.4) is separable:

$$g[m, n] = \frac{1}{9} \sum_{k=-1}^1 \sum_{l=-1}^1 f[m-k, n-l] = \frac{1}{9} \underbrace{\sum_{k=-1}^1 s[m-k, n]}_{\text{acts along } m \text{ for each } n}, \quad s[m, n] \triangleq \underbrace{\sum_{l=-1}^1 f[m, n-l]}_{\text{acts along } n \text{ for each } m}.$$

Is down-sampling a separable operation?

[RQ]

Is up-sampling a separable operation?

[RQ]

Memory

A system whose output $g[m, n]$ at any point $[m, n]$ only depends on the input image $f[m, n]$ at the same location $[m, n]$ could be called **memoryless**, if we stretch the English usage of “memory” somewhat. A memoryless system applies a **point-wise** operation to each image pixel. There are many simple, but important and useful, image processing operations that are memoryless.

Example. To compensate for nonlinearities in I/O devices like photographic film, scanners, printers, monitors, and the eye, one often applies **gamma correction** to an image prior to display:

$$g[m, n] = |f[m, n]|^\gamma .$$

MATLAB's `cmgamma` is related to this memoryless operation.

Shift-invariance

A system \mathcal{S} is called **shift invariant** or **space invariant** iff

$$f[m, n] \xrightarrow{\mathcal{S}} g[m, n] \text{ implies that } f[m - m_0, n - n_0] \xrightarrow{\mathcal{S}} g[m - m_0, n - n_0] \quad (5.11)$$

for every input image $f[m, n]$ and shifts $m_0, n_0 \in \mathbb{Z}$. Note that only integer shifts are allowed and required!

Otherwise the system is called **shift variant** or **space variant**.

Example. The moving average system is shift invariant. Proof:

$$\begin{aligned} f[m, n] \xrightarrow{\mathcal{S}} g[m, n] &= \frac{1}{9} \sum_{k=-1}^1 \sum_{l=-1}^1 f[m - k, n - l], \\ f[m - m_0, n - n_0] \xrightarrow{\mathcal{S}} \frac{1}{9} \sum_{k=-1}^1 \sum_{l=-1}^1 f[(m - k) - m_0, (n - l) - n_0] \\ &= \frac{1}{9} \sum_{k=-1}^1 \sum_{l=-1}^1 f[(m - m_0) - k, (n - n_0) - l] = g[m - m_0, n - n_0]. \end{aligned}$$

Is the thresholding system (5.5) shift invariant?

[RQ]

Is up-sampling (5.8) a shift-invariant operation?

[RQ]

Is down-sampling (5.6) a shift-invariant operation?

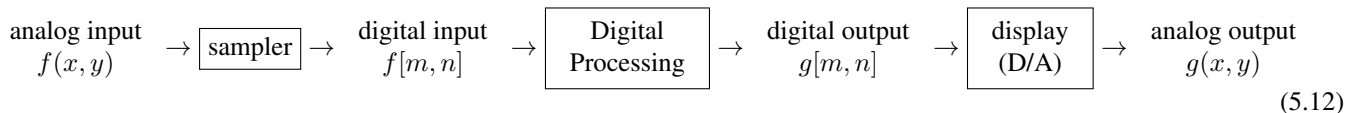
[RQ]

Rotation invariance **(skim)**

We defined continuous-space systems to be **rotation invariant** if rotating any input image caused the output image to rotate by the same amount. This definition is inconvenient for discrete-space analysis, because “rotating” discrete-space images is nontrivial.

An alternative weaker definition for continuous-space systems would be to require that the response of the system to every circularly symmetric input signal would be some circularly symmetric output signal. This alternative definition would more naturally extend to discrete-space systems, but it does not seem general enough.

In practical terms, what we would really like is something like the following. Consider the following image processing system.



What we would like is to specify properties that the digital processing system must have to ensure that the overall end-to-end system is rotationally invariant in the CS sense, *i.e.*, rotating the analog input image causes the output image to simply rotate (except of course for edge effects in a real system with finite numbers of samples). Only certain digital systems will provide this property.

We will return to this topic after discussing the 2D DSFT, where we will see that certain forms of rotation invariance are possible (theoretically at least) under certain conditions.

A-3 Linear Systems

Many image processing operations are linear (but many of the more interesting operations are not). In the context of 2D discrete-space systems, we will say \mathcal{S} is a **linear system** (function) iff it is a linear operator on² $\ell_2(\mathbb{Z}^2)$, i.e., iff \mathcal{S} satisfies

$$\mathcal{S}[\alpha f_1 + \beta f_2] = \alpha \mathcal{S}[f_1] + \beta \mathcal{S}[f_2] \quad \text{superposition property}$$

for all images $f_1[m, n]$, $f_2[m, n]$ and all real and complex constants α, β .

Example. The moving average system above is linear.

Example. The thresholding system above is nonlinear.

Are the up-sampling and down-sampling operations linear? ??

[RQ]

² $\ell_2(\mathbb{Z}^2)$ denotes the vector space of square summable 2D images, i.e., $\{f : \mathbb{Z}^2 \mapsto \mathbb{C} : \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} |f[m, n]|^2 < \infty\}$.

LSI systems

In 2D CS systems, we derived the general **superposition integral** for linear systems, and then derived the **convolution integral** as a special case for linear *shift invariant* (**LSI**) systems. We did this because many 2D CS systems (*e.g.*, mirrors) are linear but not shift invariant.

In contrast, most (if not all) 2D DS systems are “human made” to our specifications, rather than being intrinsically physical systems that must obey natural laws like Maxwell’s equations. So we can deliberately choose to design 2D DS systems to be LSI, and we usually do! Thus we proceed directly to the LSI case and derive the **convolution sum**. We could define a **superposition sum** but it is less useful for DS systems.

Impulse response

For a 2D DS LSI system, let $h[m, n]$, called the **impulse response**, denote the response to a unit impulse signal:

$$\delta_2[m, n] \rightarrow \boxed{\mathcal{S}} \rightarrow h[m, n].$$

Assuming the system is shift invariant, the response to a shifted impulse input will be simply the impulse response shifted:

$$\delta_2[m - k, n - l] \rightarrow \boxed{\mathcal{S} \text{ (SI)}} \rightarrow h[m - k, n - l].$$

Example. The impulse response of the 3 by 3 moving average filter (our only example of an LSI system thus far!) is

$$h[m, n] = \frac{1}{9} \sum_{k=m-1}^{m+1} \sum_{l=n-1}^{n+1} \delta_2[k, l] = \frac{1}{9} \sum_{k=-1}^1 \sum_{l=-1}^1 \delta_2[m - k, n - l] = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}.$$

5.3 2D convolution

We can use the sifting property of the Kronecker impulse function to represent any 2D DS signal as a sum:

$$f[m, n] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] \delta_2[m - k, n - l].$$

For a linear, shift-invariant (**LSI**) system, we can use the **superposition property** to see that

$$f[m, n] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] \delta_2[m - k, n - l] \rightarrow \boxed{\mathcal{S} \text{ LSI}} \rightarrow \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] h[m - k, n - l].$$

Thus we have derived the following general **input-output relationship** for 2D DS LSI systems:

$$f[m, n] \rightarrow \boxed{\text{LSI } h[m, n]} \rightarrow g[m, n] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] h[m - k, n - l] = h[m, n] ** f[m, n],$$

which is called the 2D **convolution sum**.

An LSI system is completely specified by its impulse response.

2D convolution recipe

- Mirror $h[m, n]$ about origin to form $h[-k, -l]$
- Shift the mirrored result by m, n to form $h[m - k, n - l]$
- Multiply $h[m - k, n - l]$ by $f[k, l]$
- Sum over all k, l
- Repeat for every m, n

Example. The following figure illustrates 2D convolution.

Most of the work here could be done using the MATLAB command `conv2` as follows:

```
h = [1 2 3; 2 4 6]', f = zeros(5,4); f(2:4,2) = 1; f(5,4) = 1; g = conv2(h, f)
```

What part of convolution is not performed by the MATLAB `conv2` command? [Identifying the origin \[0, 0\]](#)

$h[m, n]$

| | | | | | |
|-----------|-----------|-----------------|---|-----------------|---|
| | | $\rightarrow n$ | | | |
| | | | 2 | 4 | 6 |
| $N_1 = 2$ | 1 | | 2 | 3 | |
| | $M_1 = 3$ | | | $\rightarrow m$ | |

 $f[k, l]$

| | | | | | | | |
|-----------|-----------|-----------------|---|---|---|---|-----------------|
| | | $\rightarrow l$ | | | | | |
| | | | 0 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 |
| $N_2 = 4$ | | | 0 | 1 | 1 | 1 | 0 |
| | $M_2 = 5$ | | | | | | $\rightarrow k$ |

flipped: $h[-k, -l]$

| | | |
|---|---|---|
| 3 | 2 | 1 |
| 6 | 4 | 2 |

 $g[m, n] = (h ** f)[m, n]$

| | | | | | | | | | |
|---------------------|---------------------|-----------------|---|---|---|----|----|---|-----------------|
| | | $\rightarrow n$ | | | | | | | |
| | | | 0 | 0 | 0 | 0 | 2 | 4 | 6 |
| | | | 0 | 0 | 0 | 0 | 1 | 2 | 3 |
| | | | 0 | 2 | 6 | 12 | 10 | 6 | 0 |
| $N_1 + N_2 - 1 = 5$ | | | 0 | 1 | 3 | 6 | 5 | 3 | 0 |
| | $M_1 + M_2 - 1 = 7$ | | | | | | | | $\rightarrow m$ |

Example. Find the response of the 3 by 3 moving average system to the 2D input signal

$$f[m, n] = \cos(\pi m) \cos(\pi n) = (-1)^{m+n}.$$

What does this signal look like? [checkerboard with \$\pm 1\$](#) $\begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 & \dots \\ 1 & -1 & 1 \end{bmatrix}$

The output is $g[m, n] = \frac{1}{9}(-1)^{m+n} = \frac{1}{9} f[m, n]$.

Image support

On paper we can convolve infinite images. A computer will only convolve **finite support** images, *i.e.*, images that are (implicitly) zero for m, n outside some rectangular region.

MATLAB's `conv2` routine performs 2D DS convolution of finite-support signals. However, the user must keep track of the origin (if it is important).

Convoluting a $M_1 \times N_1$ image with a $M_2 \times N_2$ image results in a $(M_1 + M_2 - 1) \times (N_1 + N_2 - 1)$ output image. Sometimes this “expansion” is inconvenient, so MATLAB's `conv2` routine includes a 'same' option that discards the “extra” pixels at the edge so that the output is the same size as the first input argument.

Computation and separability**(skim)**

The computational cost of 2D convolution of a $M_1 \times N_1$ image with a $M_2 \times N_2$ image is approximately $(M_1 + M_2 - 1)(N_1 + N_2 - 1) \min(M_1 N_1, M_2 N_2)$ adds/multiplies, or about $M^2 N^2$ if $N_1 = M_1 = N \gg N_2 = M_2 = M$.

Using a **separable filter** can greatly reduce computation. If $h[m, n] = h_1[m] h_2[n]$, then we can rewrite the convolution sum as follows:

$$\sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] h_1[m - k] h_2[n - l] = \sum_{k=-\infty}^{\infty} h_1[m - k] \left(\sum_{l=-\infty}^{\infty} f[k, l] h_2[n - l] \right).$$

The inner sum is simply 1D convolution of each column of the image with the 1D kernel h_2 . The outer sum is then the 1D convolution of each row of that intermediate result with the 1D kernel h_1 . For convolving a large $N \times N$ image with a small separable $M \times M$ kernel, the computational requirement is about $N^2(M + M) = 2N^2M$, so we save about a factor of $M/2$ computation. Because M is at least 3 usually, that means at least a 33% reduction in computation, and often much more.

Example. The filter $h[m, n]$ in the preceding example is separable. We can convolve “along n ” with `[1 2 3]`, and then convolve the result “along m ” with `[1 2]`.

In MATLAB this can be done as follows: `tmp = conv2(f, [1 2 3]'); g = conv2(tmp, [1 2])`

Separable filtering is so common that MATLAB’s `conv2` routine now includes an option of the form `conv2(h1, h2, x)` where h_1 is a 1D filter applied to each column and h_2 is a 1D filter applied to each row of x .

To be more precise, 1D convolution (if implemented carefully) of a N -point signal with a M -point signal, with $N \geq M$, requires

$$1 + 2 + \cdots + (M - 1) + (N - M + 1)M + (M - 1) + \cdots + 2 + 1 = M(M - 1) + (N - M + 1)M = MN$$

multiplications, and nearly the same number of additions.

Edge effects / boundary conditions

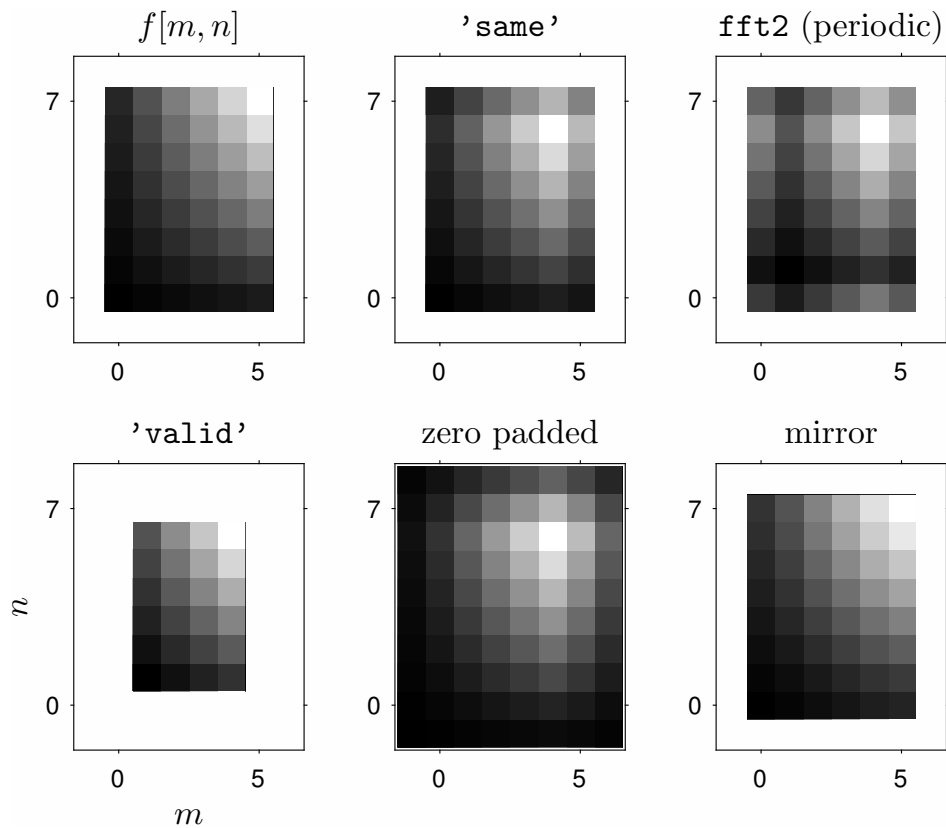
In 1D audio signal processing, usually there are enough zeros at the start (and perhaps end) of a signal that edge effects are relatively unimportant. Furthermore, audio signals have a zero DC value, so **zero padding** (adding extra zeros to start and/or end of a signal) is the natural choice when edge effects are relevant.

Although any image stored in a computer has finite support, most images come from samples of analog images of real-world scenes that have essentially unbounded support (like a photograph of part of a mountain). When processing such images using convolution (e.g., filtering), one must think about edge effects [4].

MATLAB's `conv2` routine assumes that the unavailable values outside the image are zero when computing convolution. This is only one of several options.

- zero “padding” (implicit or explicit). MATLAB's `edges` command is relevant here.
- toroidal extension or circular extension (2D wrap around, happens with 2D FFT)
- edge value replication
- mirror extension aka **mirror boundary conditions** (see **DCT** later)
- truncating and renormalizing the filter kernel near the edges (e.g., for a 5×5 moving average, when close to an edge you can use 5×4 or 5×3 etc. and adjust the denominator factor accordingly.) This is not “true” convolution of course, because it is a space-varying operation.
- Acquire more input samples than actually needed and discard the output values near the edges where the (shifted) impulse response would overlap with the unavailable samples. MATLAB's `conv2` routine includes a `valid` option precisely for this purpose. The `valid` returns an smaller image consisting only of those pixels where the shifted impulse response does not overlap beyond the input image edges.

Example. The following figure illustrates the effects of various boundary conditions when convolving an input signal $f[m, n]$ having 6×8 support with a 3×3 moving average filter.



2D convolution sum properties

The following properties of 2D convolution are easily verified directly from the 2D convolution sum by manipulations that are essentially identical to those for the 1D convolution sum.

- **Commutative property:**

$$h ** f = f ** h$$

- **Associative property:**

$$(h_2 ** h_1) ** f = h_2 ** (h_1 ** f)$$

- **Distributive property:**

$$(h_1 + h_2) ** f = (h_1 ** f) + (h_2 ** f).$$

- The order of serial connection of LSI systems does not affect the overall impulse response, because $h_1 ** h_2 = h_2 ** h_1$.
- Identity property: $f[m, n] ** \delta_2[m, n] = f[m, n]$
- Shift property: $f[m, n] ** \delta_2[m - n_0, n - m_0] = f[m - n_0, n - m_0]$
- **Shift-invariance:**

$$g[m, n] = h[m, n] ** f[m, n] \implies h[m - k_1, n - l_1] ** f[m - k_2, n - l_2] = g[m - k_1 - k_2, n - l_1 - l_2]$$

- **Separability property:** $(h_1[m] h_2[n]) ** (f_1[m] f_2[n]) = (h_1[m] * f_1[m]) \cdot (h_2[n] * f_2[n])$.
- A “scaling” property?

If $g[m, n] = h[m, n] ** f[m, n]$, then is $h[2m, 2n] ** f[2m, 2n] = h_{\downarrow 2}[m, n] ** f_{\downarrow 2}[m, n] \stackrel{?}{=} \frac{1}{4} g[2m, 2n]$?

??

[RQ]

LSI system properties via impulse response

- causal: $h[m, n] = 0$ whenever $m < 0$ or $n < 0$? That would be the natural analog to the 1D case, but it is essentially irrelevant for imaging.
- memoryless (static): $h[m, n] = \alpha \delta_2[m, n]$, otherwise not memoryless
- BIBO stable: $\sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} |h[m, n]| < \infty$
- invertible: For what type of $h[m, n]$ is an LSI system invertible? $h[m, n] ** h_i[m, n] = \delta_2[m, n]$ for some h_i
If $h[m, n] ** f[m, n] = 0_{\text{signal}}$ for some nonzero signal $f[m, n]$, then the system with impulse response $h[m, n]$ is not invertible.

2D correlation

The (deterministic) **cross-correlation** of two 2D signals $f[m, n]$ and $g[m, n]$ is defined by³:

$$r_{fg}[k, l] \triangleq \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f^*[m, n] g[m - k, n - l] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f^*[m + k, n + l] g[m, n], \quad k, l \in \mathbb{Z},$$

where (k, l) is called the **lag**. This operation is very useful for finding features of known shape but unknown position in images.

Properties of cross-correlation

- Hermitian symmetry: $r_{fg}[-k, -l] = r_{gf}^*[k, l]$
- $r_{fg}[m, n] = f^*[-m, -n] ** g[m, n]$ (convolution without the mirroring)
- **Schwarz inequality**: $|r_{fg}[k, l]| \leq \sqrt{E_f E_g}$ where $E_f = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} |f[m, n]|^2$ is the signal energy.
- **Bilinearity** (for complex-valued signals, the technical term is **sesquilinear form**): If $f = \alpha_1 f_1 + \alpha_2 f_2$ and $g = \beta_1 g_1 + \beta_2 g_2$ then $r_{fg}[k, l] = \alpha_1^* \beta_1 r_{f_1 g_1}[k, l] + \alpha_1^* \beta_2 r_{f_1 g_2}[k, l] + \alpha_2^* \beta_1 r_{f_2 g_1}[k, l] + \alpha_2^* \beta_2 r_{f_2 g_2}[k, l]$.

³ The literature is inconsistent on whether the conjugate goes on the first or second argument. Wikipedia's **cross-correlation** page and some books use the first argument, whereas MATLAB's `xcorr` function uses the second argument. Pay attention when using complex data.

The **autocorrelation** of a signal is the cross-correlation of the signal with itself:

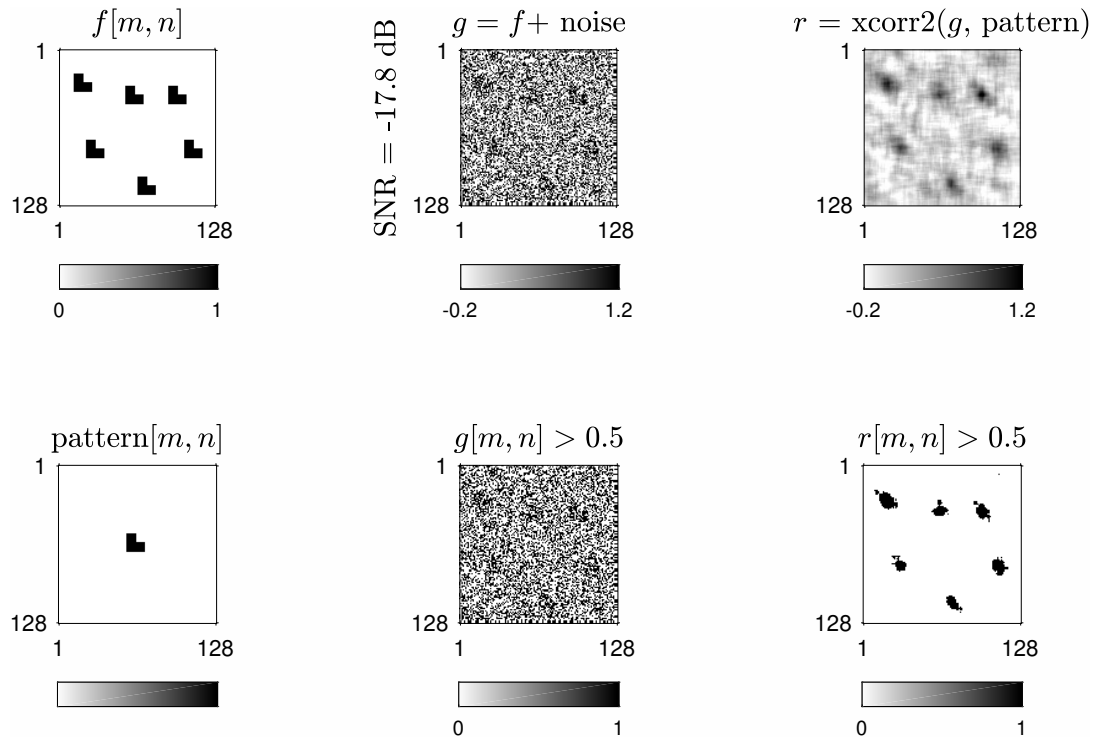
$$r_{ff}[k, l] \triangleq \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f^*[m, n] f[m - k, n - l] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f^*[m + k, n + l] f[m, n].$$

It inherits the properties of cross-correlation. In addition:

- $|r_{ff}[k, l]| \leq r_{ff}[0, 0]$
- $r_{ff}[0, 0] = E_f$

Example. The figure below illustrates an example application using MATLAB's `xcorr2` function to help detect a pattern of known size, orientation, and shape ("L") but unknown location(s) in a noisy image. Here, the pattern image p was normalized to have unit energy so that the cross-correlation values (in the absence of noise) would lie in the range $r_{fp} \in [0, 1]$ so that a threshold of 0.5 would work reasonably well. The Schwarz inequality above is the rationale for this normalization. `fig_xcorr_pattern.m`

This pattern detection technique is closely related to the **matched filter** discussed in detection theory (e.g., EECS 564).



Eigenfunctions of 2D DS systems

2D DS complex exponential signals are **eigenfunctions** of 2D DS LSI systems, *even if they are aperiodic!*

$$\begin{aligned}
 f[m, n] = e^{i(\Omega_1 m + \Omega_2 n)} &\rightarrow \boxed{\text{LSI } h[m, n]} \rightarrow g[m, n] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} h[k, l] f[m - k, n - l] \\
 &= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} h[k, l] e^{i[\Omega_1(m-k) + \Omega_2(n-l)]} \\
 &= e^{i(\Omega_1 m + \Omega_2 n)} H(\Omega_1, \Omega_2),
 \end{aligned}$$

where the 2D DSFT of $h[m, n]$ (aka the **frequency response** of the system) is given by:

$$H(\Omega_1, \Omega_2) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} h[m, n] e^{-i(\Omega_1 m + \Omega_2 n)}.$$

Thus complex exponential signals are particularly important (for studying LSI systems) and the formula for frequency response is also inherently important.

5.4 2D discrete-space Fourier transform

The **2D discrete-space Fourier transform (DSFT)** of a 2D discrete-space signal $g[m, n]$ is defined as follows:

$$G(\Omega_1, \Omega_2) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g[m, n] e^{-i(\Omega_1 m + \Omega_2 n)} \quad (\text{analysis}).$$

(This is the 2D space domain analog of the **discrete-time Fourier transform (DTFT)**).

What are the units of $G(\Omega_1, \Omega_2)$ if $g[m, n]$ is unitless? ??

[RQ]

The **inverse 2D DSFT** is given by

$$g[m, n] = \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} G(\Omega_1, \Omega_2) e^{i(\Omega_1 m + \Omega_2 n)} d\Omega_1 d\Omega_2 \quad (\text{synthesis}). \quad (5.13)$$

Engineer's proof of the inverse 2D DSFT relationship:

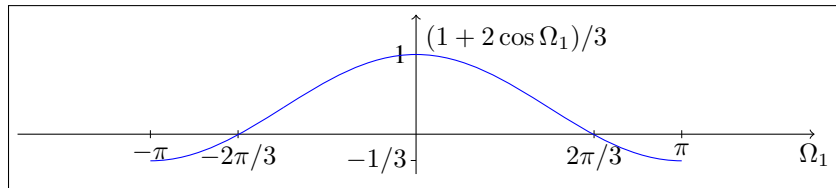
$$\begin{aligned} & \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} G(\Omega_1, \Omega_2) e^{i(\Omega_1 m + \Omega_2 n)} d\Omega_1 d\Omega_2 \\ &= \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \left[\sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} g[k, l] e^{-i(\Omega_1 k + \Omega_2 l)} \right] e^{i(\Omega_1 m + \Omega_2 n)} d\Omega_1 d\Omega_2 \\ &= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} g[k, l] \left[\frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} e^{i[\Omega_1(m-k) + \Omega_2(n-l)]} d\Omega_1 d\Omega_2 \right] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} g[k, l] \delta_2[m-k, n-l] = g[m, n]. \end{aligned}$$

The proof is valid, *i.e.*, the Fourier transform exists and the synthesis formula holds, if $g[m, n]$ has finite energy or is absolutely summable, *i.e.*, $\sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} |g[m, n]| < \infty$.

Example. The frequency response of the (separable!) 3×3 moving average filter is given by

$$H(\Omega_1, \Omega_2) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} h[m, n] e^{-i(\Omega_1 m + \Omega_2 n)} = \frac{1}{9} \sum_{m=-1}^1 \sum_{n=-1}^1 e^{-i\Omega_1 m} e^{-i\Omega_2 n} = \frac{1}{9} (1 + 2 \cos \Omega_1) (1 + 2 \cos \Omega_2).$$

Is this a good low-pass filter? **Not very, because it is not zero at $\pm\pi$.**



Convergence



Any time one encounters infinite sums or integrals, the subject of **convergence** should be considered.

We say the FT of $g[m, n]$ **converges (pointwise)** iff $G(\Omega_1, \Omega_2)$ is everywhere finite and for all Ω_1 and Ω_2 :

$$\lim_{N \rightarrow \infty} G^{(N)}(\Omega_1, \Omega_2) = G(\Omega_1, \Omega_2), \quad \text{where } G^{(N)}(\Omega_1, \Omega_2) \triangleq \sum_{n=-N}^N \sum_{m=-N}^N g[m, n] e^{-i(\Omega_1 m + \Omega_2 n)}.$$

Sufficient condition: if $g[m, n]$ is **absolutely summable**, i.e., if $\sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} |g[m, n]| < \infty$, then the 2D DSFT of $g[m, n]$ converges. Furthermore, $G(\Omega_1, \Omega_2)$ is an **analytic function** and is infinitely differentiable with respect to Ω_1 and Ω_2 .

Alternatively, if $g[m, n]$ is **square summable** (an energy signal), then the 2D DSFT converges in a **mean-square** sense:

$$\int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \left| G^{(N)}(\Omega_1, \Omega_2) - G(\Omega_1, \Omega_2) \right|^2 d\Omega_1 d\Omega_2 \rightarrow 0$$

as $N \rightarrow \infty$, where $G^{(N)}(\Omega_1, \Omega_2)$ is the truncated sum given above. This property is theoretically weaker than pointwise convergence, but it means that the error energy diminishes with increasing N , which usually suffices for practical purposes.

Example. The signal $g[m, n] = \text{sinc}_2\left(\frac{m}{2}, \frac{n}{2}\right)$ is square summable but not absolutely summable because $1 + 1/3 + 1/5 + \dots = \infty$. Its DSFT has “ripples” near $\pi/2$ that persist in the pointwise sense but not in an mean-square sense as $N \rightarrow \infty$.

Convergence of DSFT - proof sketch



Consider the 1D case for simplicity.

If $g[n]$ is absolutely summable, then for all $\varepsilon > 0$ there exists $N_\varepsilon \in \mathbb{N}$ such that $\sum_{|n|>N_\varepsilon} |g[n]| < \varepsilon$. By the triangle inequality:

$$|G(\Omega)| = \left| \sum_{n=-\infty}^{\infty} g[n] e^{-i\Omega n} \right| \leq \sum_{n=-\infty}^{\infty} |g[n] e^{-i\Omega n}| = \sum_{n=-\infty}^{\infty} |g[n]| < \infty,$$

and similarly

$$|G_{N_\varepsilon}(\Omega) - G(\Omega)| = \left| \sum_{n=-N}^N g[n] e^{-i\Omega n} - \sum_{n=-\infty}^{\infty} g[n] e^{-i\Omega n} \right| = \left| \sum_{|n|>N} g[n] e^{-i\Omega n} \right| \leq \sum_{|n|>N_\varepsilon} |g[n]| < \varepsilon.$$

This inequality holds for any ε and any Ω , so we have **point-wise convergence** of $G_N(\Omega)$ to $G(\Omega)$.

Example. Consider the LSI DS system defined by the following block diagram with two sub-systems having recursively defined input-output relationships for parameters $|a| < 1$ and $|b| < 1$:

$$f[m, n] \rightarrow \boxed{s[m, n] = a s[m-1, n] + f[m, n]} \xrightarrow{s[m, n]} \boxed{g[m, n] = b s[m, n-1] + s[m, n]} \rightarrow g[m, n].$$

The corresponding impulse response and frequency response is:

$$h[m, n] = a^m \text{step}[m] b^n \text{step}[n] \xleftrightarrow{\text{DSFT}} H(\Omega_1, \Omega_2) = \frac{1}{1 - a e^{-i\Omega_1}} \frac{1}{1 - b e^{-i\Omega_2}},$$

because if $|a| < 1$ then

$$\sum_{n=-\infty}^{\infty} a^n \text{step}[n] e^{-i\Omega n} = \sum_{n=0}^{\infty} (a e^{-i\Omega})^n = \frac{1}{1 - a e^{-i\Omega}}.$$

The support of this impulse response $h[m, n]$ is infinite, so convergence of the DSFT is a relevant question. Because $|a| < 1$ and $|b| < 1$, the impulse response $h[m, n]$ is absolutely summable:

$$\sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} |h[m, n]| = \left(\sum_{m=0}^{\infty} |a^m| \right) \left(\sum_{n=0}^{\infty} |b^n| \right) = \frac{1}{1 - |a|} \frac{1}{1 - |b|} < \infty.$$

Properties of the DSFT

The table on page 5.50 summarizes the properties of the 2D DSFT. Most of these are straightforwardly derivable and very much like the analogous property of the 2D continuous-space FT. In the following we emphasize properties of the DSFT that *differ* from those of the continuous-space FT.

Periodicity

The 2D DSFT is **periodic** with period $(2\pi, 2\pi)$:

$$G(\Omega_1 + 2\pi, \Omega_2) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g_d[m, n] e^{-i[(\Omega_1+2\pi)m+\Omega_2n]} = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g_d[m, n] e^{-i[\Omega_1 m+\Omega_2 n]} = G(\Omega_1, \Omega_2),$$

and likewise $G(\Omega_1, \Omega_2 + 2\pi) = G(\Omega_1, \Omega_2)$.

Thus we usually display $G(\Omega_1, \Omega_2)$ only over the region $[-\pi, \pi] \times [-\pi, \pi]$.

Multiplication property

Because the 2D DSFT is periodic, the space-domain **multiplication property** is a little more complicated than simply “convolution” in the frequency domain.

$$f[m, n] g[m, n] \stackrel{\text{DSFT}}{\longleftrightarrow} \frac{1}{(2\pi)^2} F(\Omega_1, \Omega_2) \boxed{*}_{2\pi} G(\Omega_1, \Omega_2) = \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} F(\lambda_1, \lambda_2) G(\Omega_1 - \lambda_1, \Omega_2 - \lambda_2) d\lambda_1 d\lambda_2,$$

which is (2π) **periodic convolution**. (Notice that the integrand is periodic with period 2π .) This can be derived by substituting the summations for $F(\Omega_1, \Omega_2)$ and $G(\Omega_1, \Omega_2)$ into the convolution sum and reducing the expression to an integral that can be recognized as the DSFT of $f[m, n] g[m, n]$.

Duality

Is there a **duality** property for the DSFT? No, because the arguments $[m, n]$ and (Ω_1, Ω_2) are fundamentally different.

Transpose property

$$g[m, n] \stackrel{\text{DSFT}}{\longleftrightarrow} G(\Omega_1, \Omega_2) \implies g[n, m] \stackrel{\text{DSFT}}{\longleftrightarrow} G(\Omega_2, \Omega_1)$$

Reflection properties

$$g[-m, n] \stackrel{\text{DSFT}}{\longleftrightarrow} G(-\Omega_1, \Omega_2), \quad g[m, -n] \stackrel{\text{DSFT}}{\longleftrightarrow} G(\Omega_1, -\Omega_2), \quad g[-m, -n] \stackrel{\text{DSFT}}{\longleftrightarrow} G(-\Omega_1, -\Omega_2)$$

Circular symmetry properties

If a signal $g[m, n]$ is **weak-sense circularly symmetric**, what can we say about its spectrum $G(\Omega_1, \Omega_2)$? **Not much!**

Exercise. If a signal $g[m, n]$ is **strong-sense circularly symmetric**, what can we say about its spectrum $G(\Omega_1, \Omega_2)$?
Revisit this question *after* studying the sampling relationship (5.15).

Scaling properties?

In continuous space we have the simple scaling property $g(\alpha x, \beta y) \xleftrightarrow{\mathcal{F}_2} \frac{1}{|\alpha\beta|} G(\nu_x/\alpha, \nu_y/\beta)$.

The corresponding properties in discrete space are more complicated!

- **Exercise.** Find a FT property for **downsampling**: $g[m, n] = f[2m, 2n]$. (HW!)

Hint. Due to possible aliasing, the answer is *not* simply $G(\Omega_1, \Omega_2) \stackrel{\text{no!}}{=} \frac{1}{4} F(\Omega_1/2, \Omega_2/2)$.

- **Exercise.** Find a FT property for **up-sampling**: $g[m, n] = \begin{cases} f[m/2, n/2], & m \text{ even and } n \text{ even} \\ 0, & \text{otherwise.} \end{cases}$ (HW!)

Up-sampling and down-sampling are very common operations in image processing, so their effects on the spectrum are important.

The following table summarizes properties of the 2D DSFT.

The table after that gives a few examples of DSFT pairs.

This DS table is noticeably shorter than our table of 2D CS FT pairs. In discrete space we usually either work numerically (using FFT) or focus on simple functions (typically the impulse response of a 2D filter) as follows:

$$h[m, n] = \sum_l \alpha_l \delta_2[m - m_l, n - n_l] \xleftrightarrow{\text{DSFT}} H(\Omega_1, \Omega_2) = \sum_l \alpha_l e^{-i(\Omega_1 m_l + \Omega_2 n_l)}. \quad (5.14)$$

| | Space | Fourier (DSFT) |
|------------------------------|---|--|
| Synthesis | | $g[m, n] = \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} G(\Omega_1, \Omega_2) e^{i(\Omega_1 m + \Omega_2 n)} d\Omega_1 d\Omega_2$ |
| Analysis | | $G(\Omega_1, \Omega_2) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g[m, n] e^{-i(\Omega_1 m + \Omega_2 n)}$ |
| Eigenfunction | $h[m, n] ** e^{i(am+bn)} = H(a, b) e^{i(am+bn)}$ | $H(\Omega_1, \Omega_2) (2\pi)^2 \delta_2((\Omega_1 - a, \Omega_2 - b))_{(2\pi, 2\pi)}$ $= H(a, b) (2\pi)^2 \delta_2((\Omega_1 - a, \Omega_2 - b))_{(2\pi, 2\pi)}$ |
| Periodicity | | $H(\Omega_1 + k2\pi, \Omega_2 + l2\pi) = H(\Omega_1, \Omega_2), \forall k, l \in \mathbb{Z}$ |
| Linearity | $a_1 f_1[m, n] + a_2 f_2[m, n]$ | $a_1 F_1(\Omega_1, \Omega_2) + a_2 F_2(\Omega_1, \Omega_2)$ |
| Shift | $g[m - m_0, n - n_0]$ | $G(\Omega_1, \Omega_2) e^{-i(\Omega_1 m_0 + \Omega_2 n_0)}$ |
| Reversal | $g[-m, -n]$ | $G(-\Omega_1, -\Omega_2)$ |
| Convolution | $h[m, n] ** f[m, n]$ | $H(\Omega_1, \Omega_2) F(\Omega_1, \Omega_2)$ |
| Multiplication (windowing) | $f[m, n] g[m, n]$ | $\frac{1}{(2\pi)^2} F(\Omega_1, \Omega_2) \boxed{*}_{2\pi} G(\Omega_1, \Omega_2)$ |
| Cross-Correlation | $f[m, n] \star\star g[m, n] = f^*[-m, -n] ** g[m, n]$ | $F^*(\Omega_1, \Omega_2) G(\Omega_1, \Omega_2)$ |
| Frequency shift (modulation) | $g[m, n] e^{i(am+bn)}$ | $G(\Omega_1 - a, \Omega_2 - b)$ |
| Freq. Differentiation | $-i m g[m, n]$ | $\frac{\partial}{\partial \Omega_1} G(\Omega_1, \Omega_2)$ |
| Conjugation | $g^*[m, n]$ | $G^*(-\Omega_1, -\Omega_2)$ |
| Symmetry properties | $g[m, n]$ real $g[m, n] = g^*[-m, -n]$ (Hermitian) | $G(\Omega_1, \Omega_2) = G^*(-\Omega_1, -\Omega_2)$ (Hermitian) $G(\Omega_1, \Omega_2)$ real |
| Separability | $g[m, n] = g_1[m] g_2[n]$ | $G(\Omega_1, \Omega_2) = G_1(\Omega_1) G_2(\Omega_2)$ |
| Parseval's Theorem | | $\sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f[m, n] g^*[m, n] = \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} F(\Omega_1, \Omega_2) G^*(\Omega_1, \Omega_2) d\Omega_1 d\Omega_2$ |
| Parseval/Rayleigh Theorem | | $E = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g[m, n] ^2 = \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} G(\Omega_1, \Omega_2) ^2 d\Omega_1 d\Omega_2$ |
| DC Value | | $\sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g[m, n] = G(0, 0)$ |

Examples of 2D Discrete-Space Fourier Transforms

| Space | Fourier |
|--|---|
| $\delta_2[m, n]$ | 1 |
| 1 | $(2\pi)^2 \delta_2((\Omega_1, \Omega_2))_{(2\pi, 2\pi)}$ |
| $e^{i(am+bn)}$ | $(2\pi)^2 \delta_2((\Omega_1 - a, \Omega_2 - b))_{(2\pi, 2\pi)}$ |
| $\begin{cases} 1, & m \leq M, n \leq N \\ 0, & \text{otherwise} \end{cases} \quad N, M \in \{0, 1, 2, \dots\}$ | $\sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} (2\pi)^2 \delta_2(\Omega_1 - a - 2\pi k, \Omega_2 - b - 2\pi l)$ |
| | $\frac{\sin(\Omega_1(M + 1/2))}{\sin(\Omega_1/2)} \frac{\sin(\Omega_2(N + 1/2))}{\sin(\Omega_2/2)}$ |
| $\frac{\Omega_X^{\max} \Omega_Y^{\max}}{\pi^2} \text{sinc}_2\left(\frac{\Omega_X^{\max}}{\pi} m, \frac{\Omega_Y^{\max}}{\pi} n\right), \quad \Omega_X^{\max} \leq \pi, \Omega_Y^{\max} \leq \pi$ | “ $\text{rect}_2\left(\frac{\Omega_1}{2\Omega_X^{\max}}, \frac{\Omega_2}{2\Omega_Y^{\max}}\right)$ ” |

The function $\frac{\sin((N + 1/2)\Omega)}{\sin(\Omega/2)}$ is sometimes called the **Dirichlet kernel** or the **periodic sinc** function. [\[wiki\]](#)

Its value is $2N + 1$ for $\Omega = 0$.

The MATLAB function `diric` implements something close to this function.

Note that the DSFT of “ $\text{rect}\left(\frac{n}{N}\right)$ ” is **not** $N \text{sinc}(N\Omega)$. This is a common mistake!

Exercise. Derive the rect/Dirichlet pair above.

The sinc/rect pair follows from the sampling relationship (5.15) below.



Transforms of elementary functions

Impulse

$$\delta_2[m, n] \stackrel{\text{DSFT}}{\longleftrightarrow} 1$$

Complex exponential and constant

In CS, we have the FT pair $e^{i2\pi(ax+by)} \stackrel{\mathcal{F}_2}{\longleftrightarrow} \delta_2(\nu_x - a, \nu_y - b)$.

What about in DS? Can the spectrum of a DS signal consist of a single impulse? **No, because $G_d(\Omega_1, \Omega_2)$ is periodic.**

Consider the following periodic spectrum:

$$G(\Omega_1, \Omega_2) = (2\pi)^2 \delta_2((\Omega_1 - a, \Omega_2 - b))_{(2\pi, 2\pi)} \triangleq \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} (2\pi)^2 \delta_2(\Omega_1 - a - k2\pi, \Omega_2 - b - l2\pi)$$

(using Dirac impulses), then by the inverse FT (assuming $a, b \in (-\pi, \pi]$):

$$\begin{aligned} g[m, n] &= \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} (2\pi)^2 \delta_2(\Omega_1 - a, \Omega_2 - b) e^{i(\Omega_1 m + \Omega_2 n)} d\Omega_1 d\Omega_2 \\ &= \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \delta_2(\Omega_1 - a, \Omega_2 - b) e^{i(\Omega_1 m + \Omega_2 n)} d\Omega_1 d\Omega_2 = e^{i(am + bn)}. \end{aligned}$$

So we have shown

$$e^{i(am+bn)} \xleftrightarrow{\text{DSFT}} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} (2\pi)^2 \delta_2(\Omega_1 - a - k2\pi, \Omega_2 - b - l2\pi) = (2\pi)^2 \delta_2((\Omega_1 - a, \Omega_2 - b))_{(2\pi, 2\pi)},$$

where the subscript $(2\pi, 2\pi)$ means, as indicated by the summation formula, that the stated function is repeated periodically with horizontal period 2π and vertical period 2π .

One special case of this pair is a constant function ($a = b = 0$), for which we obtain

$$1 \xleftrightarrow{\text{DSFT}} (2\pi)^2 \delta_2((\Omega_1, \Omega_2))_{(2\pi, 2\pi)}.$$

Another special case is when $a = b = \pm\pi$:

$$(-1)^m (-1)^n = \cos(\pi m) \cos(\pi n) = e^{i\pi m} e^{i\pi n} \xleftrightarrow{\text{DSFT}} (2\pi)^2 \delta_2((\Omega_1 - \pi, \Omega_2 - \pi))_{(2\pi, 2\pi)}.$$

When using the DSFT synthesis formula (5.13) for this signal, it is preferable to replace each $\int_{-\pi}^{\pi}$ with $\int_0^{2\pi}$ to avoid having Dirac impulses at the ends of the integration interval.

Periodic signals **(skim)**

If $g[m, n]$ is periodic with period (M, N) , then using the 2D **DS Fourier series (DFS)** (discussed in more detail in Ch. 7) we can write $g[m, n]$ in terms of complex exponentials:

$$g[m, n] = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} G[k, l] e^{i2\pi(km/M + ln/N)},$$

where $G[k, l]$ denotes the (M, N) -point **(DFS)** (see Ch. 7) of $g[m, n]$:

$$G[k, l] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} g[m, n] e^{-i2\pi\left(\frac{k}{M}m + \frac{l}{N}n\right)}.$$

Thus the Fourier transform of any periodic signal $g[m, n]$ is impulsive, *i.e.*, has a **line spectrum**:

$$\begin{aligned} g[m, n] &\stackrel{\text{DSFT}}{\longleftrightarrow} \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} G[k, l] (2\pi)^2 \delta_2 \left(\left(\Omega_1 - \frac{2\pi k}{M}, \Omega_2 - \frac{2\pi l}{N} \right) \right)_{(2\pi, 2\pi)} \\ &= \left(\frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} G[k, l] (2\pi)^2 \delta_2 \left(\Omega_1 - \frac{2\pi k}{M}, \Omega_2 - \frac{2\pi l}{N} \right) \right)_{(2\pi, 2\pi)}. \end{aligned}$$

Sampling revisited with the DSFT

Using the 2D DSFT, we can now (re)derive 2D sampling theory *without any mention of Dirac impulses*.

Consider ideal rectilinear point sampling model: $g_d[m, n] = g_a(m\Delta_x, n\Delta_y)$.

Using the inverse 2D FT we have

$$\begin{aligned}
 g_d[m, n] &= g_a(m\Delta_x, n\Delta_y) \\
 &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} G_a(\nu_x, \nu_y) e^{i2\pi[m\Delta_x\nu_x + n\Delta_y\nu_y]} d\nu_x d\nu_y \\
 &= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \int_{(k-1/2)/\Delta_x}^{(k+1/2)/\Delta_x} \int_{(l-1/2)/\Delta_y}^{(l+1/2)/\Delta_y} G_a(\nu_x, \nu_y) e^{i2\pi[m\Delta_x\nu_x + n\Delta_y\nu_y]} d\nu_x d\nu_y \\
 &\text{let } \Omega_1 = 2\pi(\nu_x\Delta_x - k) \text{ and } \Omega_2 = 2\pi(\nu_y\Delta_y - l) \\
 &= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} G_a\left(\frac{\Omega_1/(2\pi) + k}{\Delta_x}, \frac{\Omega_2/(2\pi) + l}{\Delta_y}\right) e^{i[(\Omega_1 + 2\pi k)m + (\Omega_2 + 2\pi l)n]} \frac{d\Omega_1 d\Omega_2}{(2\pi\Delta_x)(2\pi\Delta_y)} \\
 &= \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \left[\frac{1}{\Delta_x\Delta_y} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} G_a\left(\frac{\Omega_1/(2\pi) + k}{\Delta_x}, \frac{\Omega_2/(2\pi) + l}{\Delta_y}\right) \right] e^{i(\Omega_1 m + \Omega_2 n)} d\Omega_1 d\Omega_2.
 \end{aligned}$$

This final expression is in the form of the inverse 2D DSFT. Thus, the bracketed expression must be the 2D DSFT of $g_d[m, n]$. In other words, we have shown the following relationship between the spectrum $G_d(\Omega_1, \Omega_2)$ of the sampled signal $g_d[m, n]$ and the spectrum $G_a(\nu_x, \nu_y)$ of the original CS signal $g_a(x, y)$:

$$\boxed{G_d(\Omega_1, \Omega_2) = \frac{1}{\Delta_x\Delta_y} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} G_a\left(\frac{\Omega_1/(2\pi) + k}{\Delta_x}, \frac{\Omega_2/(2\pi) + l}{\Delta_y}\right)}. \quad (5.15)$$

This expressions holds *regardless* of whether the CS signal $g_a(x, y)$ is band-limited!

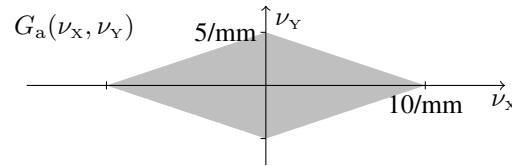
This relationship is useful for DS **filter design** based on sampling CS functions. (See (6.2) in Ch. 6.)

How would we (and when can we) recover the CS signal spectrum $G_a(\nu_x, \nu_y)$ from the DS signal spectrum $G_d(\Omega_1, \Omega_2)$?

??

Example.

Determine the spectrum of samples $g_d[m, n]$ of the CS signal $g_a(x, y)$ having the following spectrum.

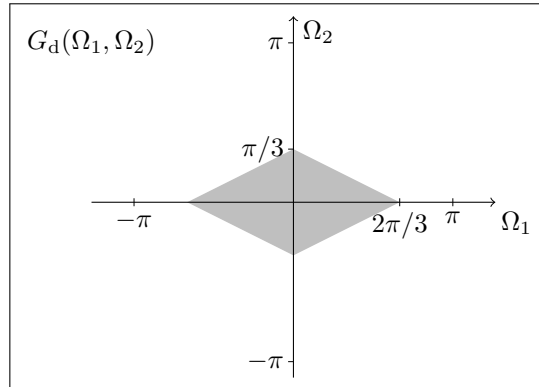


To avoid aliasing, what is maximum sample spacing? ??

[RQ]

Suppose we sample with $\Delta_x = \Delta_y = 1/30$ mm. Find spectrum of $g_d[m, n]$.

Using (5.15) above, we have that $G_d(\Omega_1, \Omega_2)$ is (replicates of) the following, where the amplitude will be scaled by $1/\Delta^2 = 30^2$.



Note. From (5.15): $\Omega_1 = 2\pi\Delta_x\nu_x$ and $\Omega_2 = 2\pi\Delta_y\nu_y$ for $|\Omega_1|, |\Omega_2| \leq \pi$, provided there is no aliasing.

Determine the DSFT of $f[m, n] = \text{sinc}^2(m) \delta[n]$. Recall that $\text{sinc}^2(x) \xleftrightarrow{\mathcal{F}} \text{tri}(\nu)$.

??

(do in class)

Determine the DSFT of $g[m, n] = \text{sinc}^2(m/3) \delta[n]$.

??

(do in class)

Methods for inverse DSFT

- Double integration (as a last resort)
- Table lookup, combined with properties
- Expansion into complex exponentials and then by inspection using (5.14).

Example. Find the inverse DSFT of $H(\Omega_1, \Omega_2) = 12 \sin^2(\Omega_1) + 8 \cos(3\Omega_2)$.

$$H(\Omega_1, \Omega_2) = 6 - 3e^{i2\Omega_1} - 3e^{-i2\Omega_1} + 4e^{i3\Omega_2} + 4e^{-i3\Omega_2}$$

$$\implies h[m, n] = 6\delta_2[m, n] - 3\delta_2[m+2, n] - 3\delta_2[m-2, n] + 4\delta_2[m, n+3] + 4\delta_2[m, n-3]$$

- Use sampling result given above in (5.15).

$$\begin{array}{ccc} h_a(x, y) & \xleftrightarrow{\mathcal{F}} & H_a(\nu_x, \nu_y) \\ \text{Diagram: } \downarrow \text{sample} & & \downarrow (5.15) \\ h[m, n] & \xleftrightarrow{\text{DSFT}} & H(\Omega_1, \Omega_2) \end{array}$$

Example. $H(\Omega_1, \Omega_2) = \text{rect}_2\left(\frac{\Omega_1}{2\Omega_c}, \frac{\Omega_2}{2\Omega_c}\right)$ where $|\Omega_c| < \pi$.

WLOG we can use $\Delta_x = \Delta_y = 1$ in (5.15). Considering just the central replicate in (5.15):

$$H(\Omega_1, \Omega_2) = H_a(\nu_x, \nu_y) \Big|_{\nu_x=\Omega_1/2\pi, \nu_y=\Omega_2/2\pi} \quad \text{so } H_a(\nu_x, \nu_y) = H(\Omega_1, \Omega_2) \Big|_{\Omega_1=2\pi\nu_x, \Omega_2=2\pi\nu_y} = \text{rect}_2\left(\frac{2\pi\nu_x}{2\Omega_c}, \frac{2\pi\nu_y}{2\Omega_c}\right),$$

which has corresponding inverse 2D FT $h_a(x, y) = \left(\frac{\Omega_c}{\pi}\right)^2 \text{sinc}_2(x\Omega_c/\pi, y\Omega_c/\pi)$. Sampling this yields the inverse 2D DSFT:

$h[m, n] = h_a(x, y) \Big|_{x=n, y=m} = \left(\frac{\Omega_c}{\pi}\right)^2 \text{sinc}_2(n\Omega_c/\pi, m\Omega_c/\pi)$. So we have established the following 2D DSFT pair:

$$\left(\frac{\Omega_c}{\pi}\right)^2 \text{sinc}_2(n\Omega_c/\pi, m\Omega_c/\pi) \xleftrightarrow{\text{DSFT}} \text{“rect}_2\left(\frac{\Omega_1}{2\Omega_c}, \frac{\Omega_2}{2\Omega_c}\right)\text{”} = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \text{rect}_2\left(\frac{\Omega_1 - 2\pi k}{2\Omega_c}, \frac{\Omega_2 - 2\pi l}{2\Omega_c}\right), \quad |\Omega_c| < \pi.$$

Magnitude and phase of the DSFT (an introduction)

The 2D DSFT is generally complex, which can be expressed using the real and imaging parts, or in terms of magnitude and phase:

$$G(\Omega_1, \Omega_2) = \text{real}\{G(\Omega_1, \Omega_2)\} + i \text{imag}\{G(\Omega_1, \Omega_2)\} = |G(\Omega_1, \Omega_2)| e^{i\theta(\Omega_1, \Omega_2)} \quad \text{where } \theta(\Omega_1, \Omega_2) \triangleq \angle G(\Omega_1, \Omega_2).$$

In applications such as transform image coding, it is useful to think about the relative importance of the magnitude and phase components. Furthermore, in some imaging contexts one can measure only the magnitude of the Fourier transform of an object, and the phase terms must be estimated somehow from the magnitude using prior information. This latter problem is known as **phase retrieval**, and we might discuss it in more detail along with image restoration later.

The basic question is: can we recover an image from just the magnitude or phase of its 2D DSFT? In general of course the answer is no. But if the object is *real*, or belongs to some other constrained class, then it may be possible [5, 6].

Example. Fig. 5.1 shows an image of Fourier and of the magnitude and phase of Fourier's transform. I displayed the magnitude on logarithmic scale so that the details are more clear.

The naive approach to trying to determine $g[m, n]$ from its magnitude or phase alone would be simply to compute the inverse FT:

$$\int_{-\pi}^{\pi} \int_{-\pi}^{\pi} |G(\Omega_1, \Omega_2)| e^{i(\Omega_1 m + \Omega_2 n)} d\Omega_1 d\Omega_2, \quad \text{or} \quad \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} e^{i\angle G(\Omega_1, \Omega_2)} e^{i(\Omega_1 m + \Omega_2 n)} d\Omega_1 d\Omega_2.$$

Fig. 5.1 shows that this simple approach fails completely for the magnitude, but some recognizable information is seen in the inverse FT of the phase. The importance of the phase in imaging is well known [7].

We can get some insight into why the magnitude fails by thinking about the magnitude squared (which is closely related to the magnitude). What is the inverse 2D DSFT of $|G(\Omega_1, \Omega_2)|^2$? The autocorrelation of $g[m, n]$, which, as illustrated in Fig. 5.1, seems to contain very little information about the original image.

Attempting image recovery via a simple inverse FT of magnitude or phase alone clearly is suboptimal. There are better approaches!

Algebraic approach to signal recovery from Fourier phase only _____ (skip) ♦♦

Suppose the image $g[m, n]$ is known to be **real** and has known **support** $\mathcal{D} \subset \mathbb{Z}^2$. Then

$$\tan \theta(\Omega_1, \Omega_2) = \frac{\text{imag}\{G(\Omega_1, \Omega_2)\}}{\text{real}\{G(\Omega_1, \Omega_2)\}} = \frac{\sum_{[m,n] \in \mathcal{D}} g[m, n] \sin(\Omega_1 m + \Omega_2 n)}{\sum_{[m,n] \in \mathcal{D}} g[m, n] \cos(\Omega_1 m + \Omega_2 n)},$$

where the sums are over the support set. Cross multiplying and subtracting yields the following equality for all Ω_1, Ω_2 :

$$0 = \sum_{[m,n] \in \mathcal{D}} [\sin(\Omega_1 m + \Omega_2 n) - \tan \theta(\Omega_1, \Omega_2) \cos(\Omega_1 m + \Omega_2 n)] g[m, n].$$

If we have measured the phase $\tan \theta(\Omega_1, \Omega_2)$ for K pairs of Ω_1, Ω_2 values, where $K \geq |\mathcal{D}|$, *i.e.*, at least as many frequency locations as there are elements in the support set, then we can (try to) solve the above set of linear equations for $g[m, n]$. However, $g[m, n] = 0$ is one trivial solution to this family of equations (whether noise is present or absent), so we need some other approach. Any of the (vectorized) images in the null space of the $K \times |\mathcal{D}|$ matrix is a non-trivial candidate solution. Clearly some form of prior information or regularization (discussed later) is needed to select from among the many candidate images in the null space.

Later we may discuss **iterative methods** for image recovery from either magnitude or phase information, including noise control. See [8] for examples of reconstruction from phase-only data.

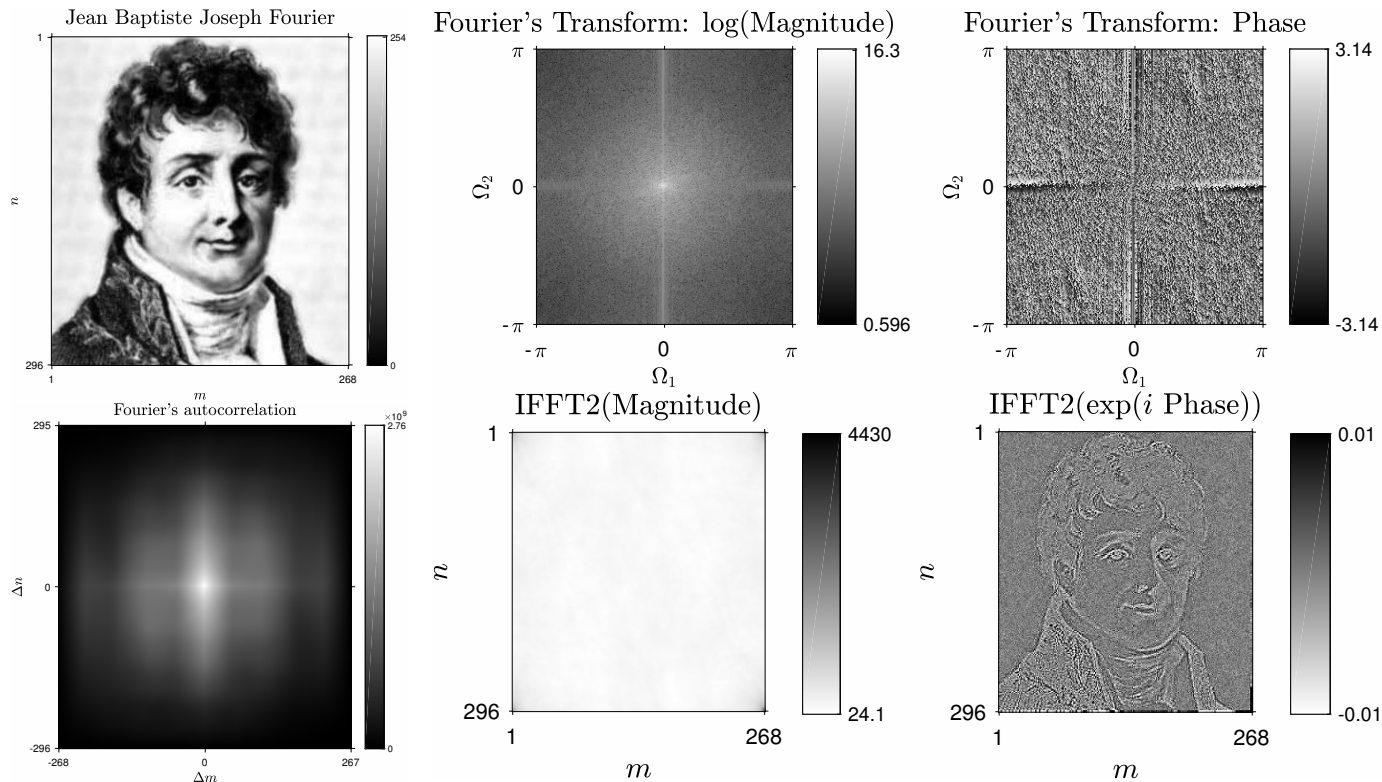


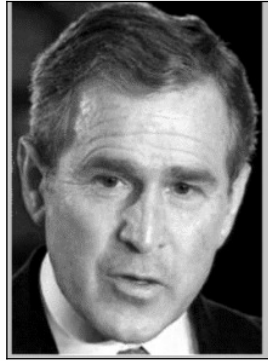
Figure 5.1: Fourier and his transforms.

Example. The following figure illustrates the effect of swapping the phase of two drastically different images.

Fourier $f[m, n]$



Bush $g[m, n]$

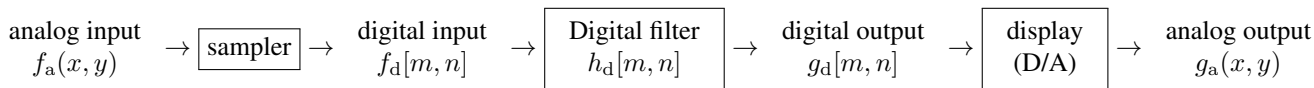


$|\text{ifft2}(|F|e^{i\angle G})|$ $|\text{ifft2}(|G|e^{i\angle F})|$



Digital processing of analog images

Consider again the following typical image processing system.



When is a system like this **LSI** from end-to-end in the analog (continuous-space) sense?

Make the following idealized assumptions:

- The sampling is ideal with sampling interval (Δ_x, Δ_y)
- The sampler includes an ideal anti-aliasing filter
- The digital processing is LSI with frequency response $H_d(\Omega_1, \Omega_2)$
- The D/A converter uses ideal 2D sinc interpolation
- Everything has infinite extent, *i.e.*, $f_d[m, n]$ and $g_d[m, n]$ are defined $\forall m, n \in \mathbb{Z}$.

Under these sufficient conditions, from end-to-end this is a **LSI** *continuous-space* system. Thus the output spectrum is related to the input spectrum by some filter:

$$G_a(\nu_x, \nu_y) = H_a(\nu_x, \nu_y) F_a(\nu_x, \nu_y).$$

What is the frequency response $H_a(\nu_x, \nu_y)$ of the equivalent analog filter? One can show:

$$\begin{aligned}
 H_a(\nu_x, \nu_y) &= H_d(\Omega_1, \Omega_2) \Big|_{\Omega_1=2\pi\Delta_x\nu_x, \Omega_2=2\pi\Delta_y\nu_y} \text{rect}_2(\nu_x\Delta_x, \nu_y\Delta_y) \\
 &= \begin{cases} H_d(2\pi\Delta_x\nu_x, 2\pi\Delta_y\nu_y), & |\nu_x| < 1/(2\Delta_x), |\nu_y| < 1/(2\Delta_y) \\ 0, & \text{otherwise.} \end{cases} \quad (5.16)
 \end{aligned}$$

The formula $\Omega_1 = 2\pi\Delta_x\nu_x$ is important for relating an “analog frequency” ν_x to the corresponding “digital frequency” Ω_1 .

Frequency-domain characteristics of LSI systems

A LSI system is characterized completely by its **frequency response** $H(\Omega_1, \Omega_2)$ (provided it exists).

By the convolution property, if the input signal $f[m, n]$ has transform $F(\Omega_1, \Omega_2)$, then

$$F(\Omega_1, \Omega_2) \rightarrow \boxed{\text{LSI } H(\Omega_1, \Omega_2)} \rightarrow G(\Omega_1, \Omega_2) = F(\Omega_1, \Omega_2) H(\Omega_1, \Omega_2).$$

This property is the foundation for **filtering**.

- The output of an LSI system will contain only frequency components that are present in the input signal.

Stability

- Recall a LSI is BIBO **stable** iff its impulse response $h[m, n]$ is absolutely summable. Fortunately, the DSFT always exists for absolutely summable signals, so the frequency response $H(\Omega_1, \Omega_2)$ always exists for stable LSI systems.
- In terms of the frequency response $H(\Omega_1, \Omega_2)$, when is a LSI system invertible? **??** [RQ]

Example. For what a values is the system with impulse response $h[m, n] = \begin{bmatrix} 0 & a & 0 \\ a & 1 & a \\ 0 & a & 0 \end{bmatrix}$ invertible? (do in class) **??**

- **Rotation invariance**

Consider the end-to-end system described on p. 5.63 (cf. (5.12)) with a LSI 2D DS system in the middle.

For what type of impulse response $h_d[m, n]$ or frequency response $H_d(\Omega_1, \Omega_2)$ is the end-to-end system rotation invariant?

The answer depends on what kind of (idealized) anti-alias filter is used prior to sampling.

- If $\Delta_x = \Delta_y$ and the anti-alias filter has the “square” frequency response $\text{rect}_2(\nu_x \Delta_x, \nu_y \Delta_y)$, then $h_d[m, n]$ should be **strict-sense circularly symmetric**, which implies that its frequency response must be *zero* outside the disk of radius π .
- If $\Delta_x = \Delta_y$ and the anti-alias filter has the “circular” frequency response $\text{rect}(\rho \Delta_x)$, then $H_d(\Omega_1, \Omega_2)$ must be “circularly symmetric” within the central disk of radius π , but its values in the “corners” of the central $(-\pi, \pi) \times (-\pi, \pi)$ square of the (Ω_1, Ω_2) plane are irrelevant because the corners are eliminated by the anti-alias filter.
- These are just two sets of sufficient conditions and there are other possibilities.

Generally speaking, for the end-to-end system to be rotation invariant, the frequency response $H(\Omega_1, \Omega_2)$ of the digital filter must be “circularly symmetric” where the proper interpretation of circular symmetry here must respect the periodicity of $H(\Omega_1, \Omega_2)$.

5.5 Filter design introduction

A common image processing task is to filter an image, *e.g.*, to remove noise (lowpass filtering) or enhance edges (highpass filtering). Perhaps filtering is not the most glamorous image processing operation, but it is often essential as a preprocessing step before more exciting image processing.

Many of the filter design principles for images parallel those of filter design in 1D DSP, *e.g.*, tradeoffs between transition bandwidth and width of the impulse response. Differences in 2D include the desirability of **rotation invariance**, our indifference to **causality**, the computational convenience of **separability**, and the lack of very useful **Z-transform** tools.

Introduction

We treat the subject of filter design using a running example.

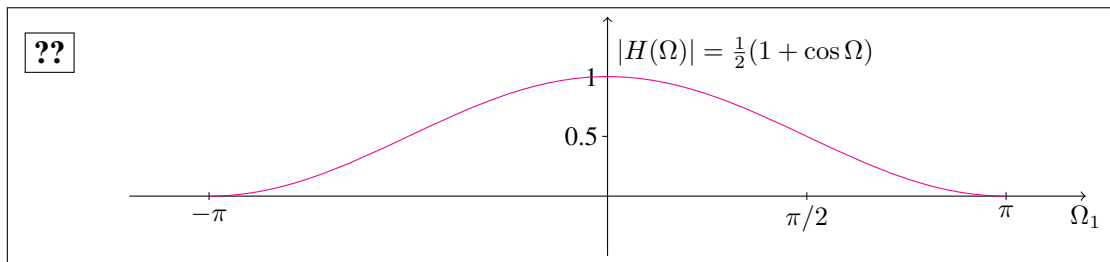
Consider the 1D filter with impulse response $h[n] = [1/4 \ 1/2 \ 1/4]$. What is the frequency response of this filter?

$$H(\Omega) = \sum_{n=-\infty}^{\infty} h[n] e^{-i\Omega n} = (1/4) + (1/2) e^{-i\Omega} + (1/4) e^{-i2\Omega} = (1/4) e^{-i\Omega} (e^{i\Omega} + 2 + e^{-i\Omega}) = (1/2) e^{-i\Omega} (1 + \cos \Omega).$$

Because $1 + \cos \Omega \geq 0$, the phase response is

$$\angle H(\Omega) = -\Omega,$$

so there is a **unit delay** or **shift** associated with this filter. The magnitude response is:



which is a reasonable simple lowpass filter.

This filter eliminates the highest discrete-space frequency ($\pm\pi$).

In that sense it is a better design than the moving average $[1/3 \ 1/3 \ 1/3]$.

How do we make a simple 2D lowpass filter with similar characteristics?

Design 1

A first design attempt makes $h[m, n]$ look like $h[n]$ along each axis: $h_1[m, n] \triangleq \begin{bmatrix} 0 & 1/4 & 0 \\ 1/4 & 1/2 & 1/4 \\ 0 & 1/4 & 0 \end{bmatrix}$.

Is this separable? No. So we must use the full 2D formula to compute the frequency response:

$$H_1(\Omega_1, \Omega_2) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} h[m, n] e^{-i(\Omega_1 m + \Omega_2 n)} = \frac{1}{2} + \frac{1}{4} (e^{-i\Omega_1} + e^{i\Omega_1} + e^{-i\Omega_2} + e^{i\Omega_2}) = \frac{1}{2} (1 + \cos \Omega_1 + \cos \Omega_2).$$

$H_1(\Omega_1, \Omega_2)$ is real (due to Hermitian symmetry of $h_1[m, n]$). Is the phase response zero? Almost:

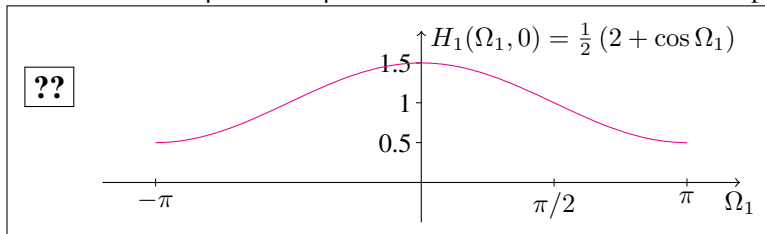
$$\angle H_1(\Omega_1, \Omega_2) = \begin{cases} \pm\pi, & 1 + \cos \Omega_1 + \cos \Omega_2 < 0 \\ 0, & \text{otherwise.} \end{cases}$$

So there is no **shift**! This is one pleasant difference between image processing and 1D signal processing: **zero phase** filters are trivial to design, and feasible to implement in 2D, because “causality” is generally unimportant.

The magnitude response is

$$|H_1(\Omega_1, \Omega_2)| = (1/2) |1 + \cos \Omega_1 + \cos \Omega_2|.$$

Is this a lowpass filter? Where does the response drop to zero? Consider the slice of this response where $\Omega_2 = 0$:



Furthermore: $H_1(\pi, \pi) = -1/2$. So this filter behaves quite differently than the 1D design. In particular, the response to the checkerboard input image $f[m, n] = (-1)^{n+m}$ with frequency (π, π) is nonzero.

We probably would prefer a lowpass design that is zero along the edge of the $\pm\pi, \pm\pi$ box, *i.e.*,

$$H(\pm\pi, \Omega_2) = H(\Omega_1, \pm\pi) = 0, \quad \forall \Omega_1, \Omega_2.$$

Design 2

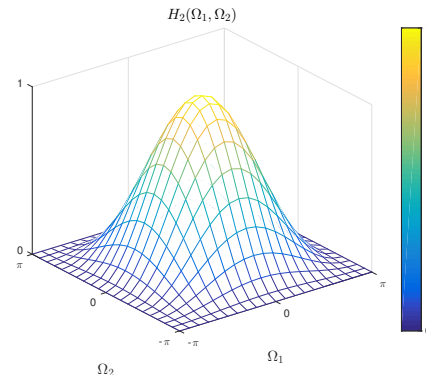
Next we try a **separable** filter design, based on the 1D filter:

$$h_2[m, n] \triangleq h[m+1]h[n+1] = \begin{bmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{bmatrix}.$$

Because this is separable, the corresponding frequency response is simply the product of the 1D frequency response (without any phase shift):

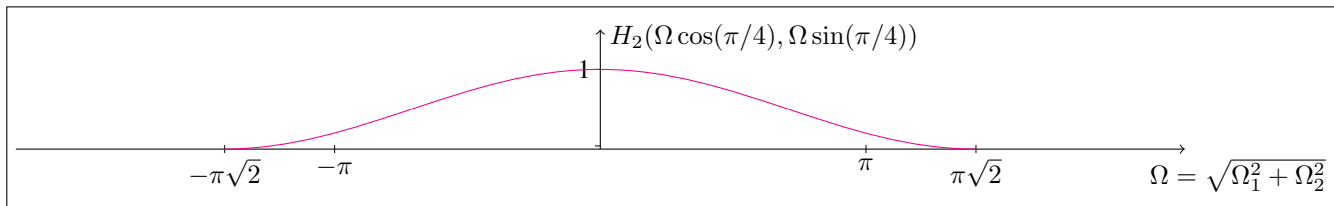
$$H_2(\Omega_1, \Omega_2) = (1/4)(1 + \cos \Omega_1)(1 + \cos \Omega_2).$$

Because it is real and nonnegative, we just plot $H(\Omega_1, \Omega_2)$ rather than $|H(\Omega_1, \Omega_2)|$. Note that $H_2(0, 0) = 1$ and $H_2(\pm\pi, \Omega_2) = H_2(\Omega_1, \pm\pi) = 0$.



Design 2 looks good so far. But what if we look at the slice along the 45° line where $\Omega_1 = \Omega_2$?

Note that $H_2(\pi/\sqrt{2}, \pi/\sqrt{2}) = (1/4)[1 + \cos(\pi/\sqrt{2})]^2 \approx 0.04$.



This filter design is not circularly symmetric! So rotation of the analog input image, followed by sampling and then filtering with this filter will yield different results (not just a rotation of the output).

Example. To demonstrate this lack of rotation invariance we sampled the following circularly symmetric analog image (shown on the next page):

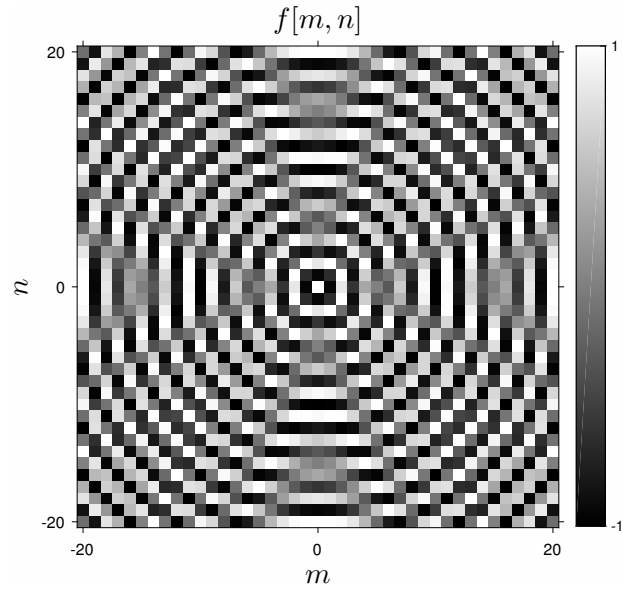
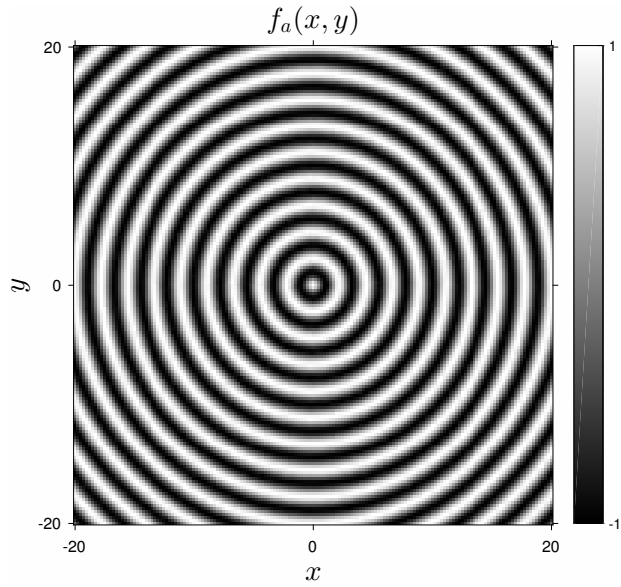
$$f[m, n] = f_a(x, y) \Big|_{x=m, y=n} \quad \text{where} \quad f_a(x, y) = \cos\left(0.9\pi\sqrt{x^2 + y^2}\right).$$

Filtering this $f[m, n]$ with the above impulse response $h_2[m, n]$ yields the output image shown on the bottom left in Fig. 5.2.

After filtering, more of the sinusoidal signal components in the diagonal directions remain than in the horizontal and vertical directions, even though the spatial frequency is essentially 0.9π in all directions.

Is the above signal $f[m, n]$ weak-sense circularly symmetric? ??

Challenge. Is $f[m, n]$ strong-sense circularly symmetric?



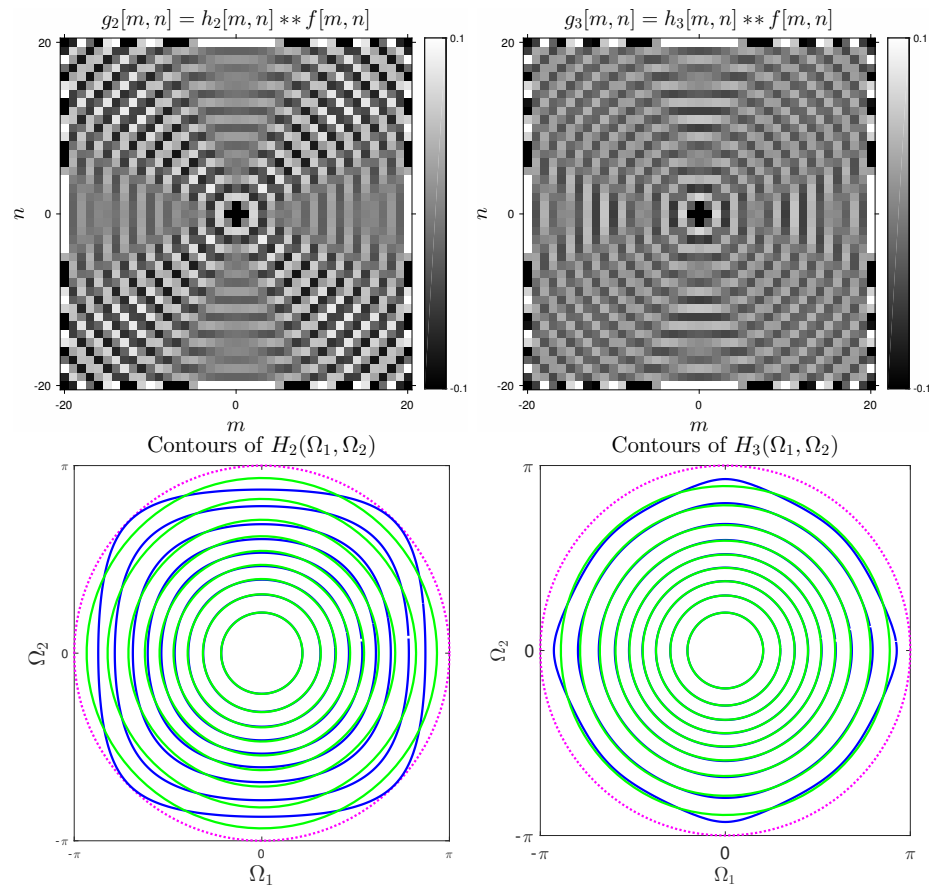


Figure 5.2: Result of filtering a sampled circularly symmetric image. The design $h_3[m, n]$ (right) has a frequency response that is more invariant to the orientation of the “sinusoidal” components than that of $h_2[m, n]$ (left). The bottom figures show contours of $H_2(\Omega_1, \Omega_2)$ and $H_3(\Omega_1, \Omega_2)$ in blue, compared to ideal in green. (The magenta line is circle of radius π .)

Design 3

Idea: start in the frequency domain, then work backwards to find the impulse response. Suppose we like the 1D behavior of

$$H(\Omega) = \frac{1}{2} (1 + \cos \Omega).$$

Then it would be reasonable to consider a 2D filter with the following circularly symmetric frequency response:

$$H_3(\Omega_1, \Omega_2) \triangleq \frac{1}{2} \left(\left(1 + \cos \sqrt{\Omega_1^2 + \Omega_2^2} \right) \text{rect} \left(\frac{\sqrt{\Omega_1^2 + \Omega_2^2}}{2\pi} \right) \right)_{(2\pi, 2\pi)}, \quad (5.17)$$

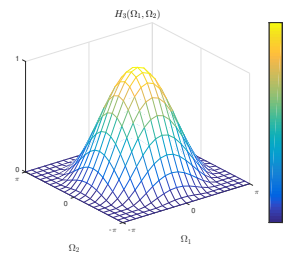
where the inner expression shows the central replicate, and the subscript reminds us that $H_3(\Omega_1, \Omega_2)$ is $(2\pi, 2\pi)$ -periodic. An alternate and equivalent expression that is clearly $(2\pi, 2\pi)$ -periodic is:

$$H_3(\Omega_1, \Omega_2) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \frac{1}{2} \left(1 + \cos \sqrt{(\Omega_1 - 2\pi k)^2 + (\Omega_2 - 2\pi l)^2} \right) \text{rect} \left(\frac{\sqrt{(\Omega_1 - 2\pi k)^2 + (\Omega_2 - 2\pi l)^2}}{2\pi} \right).$$

However, this form is probably harder to grasp quickly than the preceding “shorthand” formula.

This frequency response $H_3(\Omega_1, \Omega_2)$ is zero outside a disk of radius π (modulo 2π).

This figure illustrates $H_3(\Omega_1, \Omega_2)$.



Unfortunately, the filter with frequency response (5.17) has an infinite impulse response (**IIR**). To my knowledge, there is no analytical solution for the inverse 2D DSFT of this spectrum.

However, it is simple to use MATLAB to compute $h_3[m, n]$ approximately using an **FFT** (see Ch. 7), as illustrated in Fig. 5.3.

Exercise. After studying the FFT in Ch. 7, return to this example and think about why there appear to be no factors corresponding to $d\Omega_1 d\Omega_2$ in (5.13) in the code in Fig. 5.3.

??

For practical use we would prefer a **finite impulse response (FIR)** filter. For example, we can take the central 5×5 block of this impulse response, and quantize the coefficients to one decimal place. Fig. 5.2 shows contour plots of the resulting frequency response of this practical filter (blue lines) compared to the ideal circular contours (green lines). The practical frequency response is remarkably circularly symmetric. Although *exact* rotation invariance is impossible to achieve with an FIR filter, this example shows that a reasonable approximation is achievable with only a 5×5 mask, even with the filter coefficients quantized to two base 10 digits or about 5 bits.

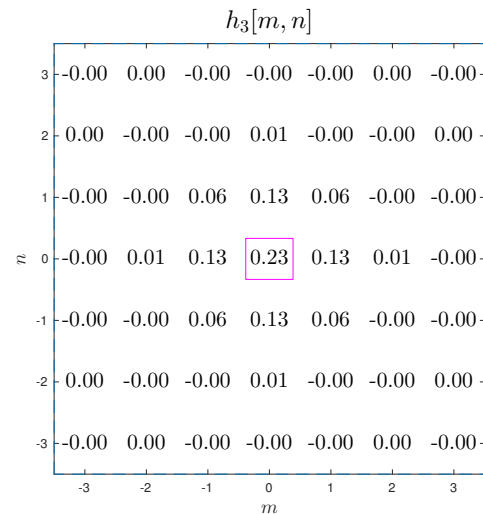
Fig. 5.2 shows the result of applying this truncated, quantized filter to the circles image. The response is much more uniform in all directions.

```

% fig_fir3a
N = 128;
ox = [-N/2:N/2-1]/N*2*pi;
oy = [-N/2:N/2-1]/N*2*pi;
[oox ooy] = ndgrid(ox,oy);
rect = @(t) abs(t) < 1/2;
om = sqrt(oox.^2 + ooy.^2);
H = (1/2) * (1 + cos(om)) .* rect(om/2/pi);
im(ox, oy, H), axis_pipi

h = fftshift(iff2(iff2shift(H)));
h = reale(h); % in case iff2 is imperfect
disp(h(N/2+1+[-3:3],N/2+1+[-3:3]))

```

Figure 5.3: Approximate impulse response $h_3[m, n]$.

Design 4

If we are going to abandon FIR considerations for specifying the frequency response, then perhaps we should just be optimistic and consider the **ideal lowpass** frequency response: for some cutoff frequency $0 < \Omega_c \leq \pi$:

$$H_4(\Omega_1, \Omega_2) = \text{rect} \left(\frac{\sqrt{\Omega_1^2 + \Omega_2^2}}{2\Omega_c} \right)_{(2\pi, 2\pi)} = \text{rect} \left(\frac{\Omega_o}{2\Omega_c} \right), \quad \Omega_o \triangleq \sqrt{(\text{mod}(\Omega_1 + \pi, 2\pi) - \pi)^2 + (\text{mod}(\Omega_2 + \pi, 2\pi) - \pi)^2}.$$

What is the impulse response of this filter?

We could attempt to integrate to find the inverse FT. An easier method uses the fact that we already know that $\text{jinc}(r) \xleftrightarrow{\mathcal{F}_2} \text{rect}(\rho)$, so $h[m, n]$ is samples of some $h_a(x, y)$ where, by the sampling relation (5.15):

$$h_4[m, n] \xleftrightarrow{\text{DSFT}} H_4(\Omega_1, \Omega_2) = \frac{1}{\Delta^2} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} H_a \left(\frac{\Omega_1/2\pi - k}{\Delta}, \frac{\Omega_2/2\pi - l}{\Delta} \right).$$

using $\Delta_x = \Delta_y = \Delta$ for simplicity. By the **scaling property** of the 2D CS FT:

$$h_a(x, y) = (2\rho_c)^2 \text{jinc}(2\rho_c r) \xleftrightarrow{\mathcal{F}_2} H_a(\nu_x, \nu_y) = \text{rect} \left(\frac{\rho}{2\rho_c} \right).$$

Define $\Omega = \sqrt{\Omega_1^2 + \Omega_2^2}$ as the digital radial frequency component. (Really it should be Ω_o given above to be rigorous.) Considering the central replicate, we need to make $\text{rect} \left(\frac{\Omega}{2\Omega_c} \right)$ match with $\frac{1}{\Delta^2} H_a \left(\frac{\Omega}{2\pi\Delta} \right) = \frac{1}{\Delta^2} \text{rect} \left(\frac{\Omega}{4\pi\Delta\rho_c} \right)$. Thus we need $\rho_c = \frac{\Omega_c}{2\pi\Delta}$.

Choosing $\Delta = 1$ for simplicity, the impulse response is:

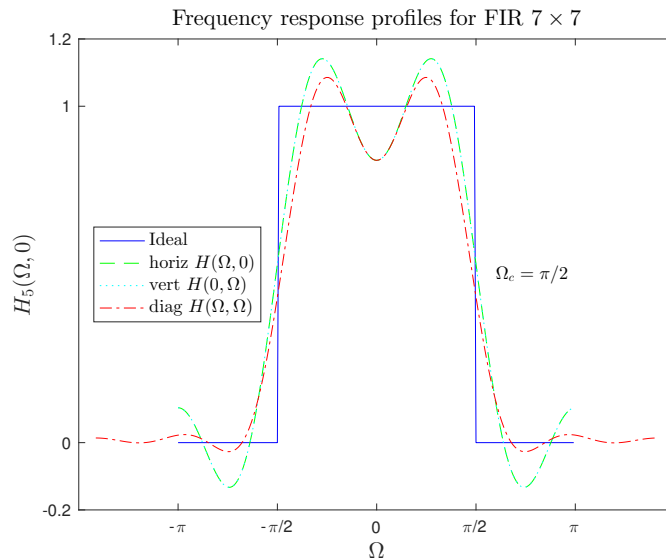
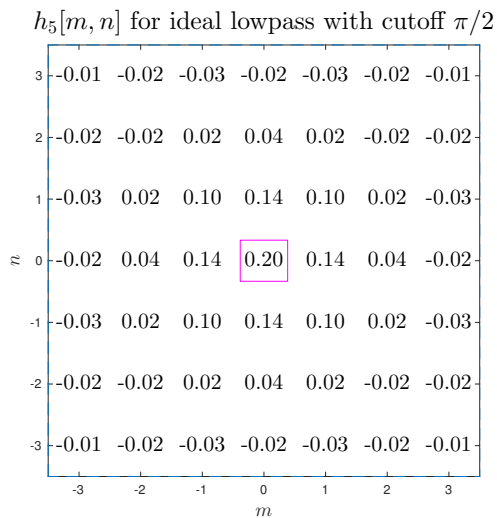
$$h_4[m, n] = h_a(m, n) = h_a \left(\sqrt{m^2 + n^2} \right) = \left(\frac{\Omega_c}{\pi} \right)^2 \text{jinc} \left(\frac{\Omega_c}{\pi} \sqrt{m^2 + n^2} \right). \quad (5.18)$$

Clearly this **circularly symmetric, ideal lowpass filter** has an IIR impulse response, as expected from similar results in 1D.

Example. The following figure shows the rectangularly windowed (FIR) version of jinc filter (5.18), i.e.,

$$h_5[m, n] \triangleq \left(\frac{\Omega_c}{\pi}\right)^2 \text{jinc}\left(\frac{\Omega_c}{\pi} \sqrt{m^2 + n^2}\right) \text{rect}_2\left(\frac{m}{7}, \frac{n}{7}\right),$$

and some profiles through its frequency response $H_5(\Omega_1, \Omega_2)$, for $\Omega_c = \pi/2$.



Is the frequency response of this windowed jinc filter circularly symmetric? **??**

[RQ]

5.6 Summary

This chapter has emphasized 2D discrete-space LSI systems and frequency analysis.

Many image processing operations, such as median filtering (to be discussed soon) are *nonlinear*, so the LSI analysis is inapplicable.

What part of this chapter did you find most confusing?

[RQ]

Bibliography

- [1] J. S. Lim. *Two-dimensional signal and image processing*. New York: Prentice-Hall, 1990. ISBN: 978-0139353222.
- [2] A. K. Jain. *Fundamentals of digital image processing*. New Jersey: Prentice-Hall, 1989. ISBN: 978-0133361650.
- [3] M. Vetterli and J. Kovacevic. *Wavelets and subband coding*. New York: Prentice-Hall, 1995.
- [4] A. Matakos, S. Ramani, and J. A. Fessler. “Accelerated edge-preserving image restoration without boundary artifacts”. In: *IEEE Trans. Im. Proc.* 22.5 (May 2013), 2019–29.
- [5] J. R. Fienup. “Phase retrieval algorithms: a comparison”. In: *Appl. Optics* 21.15 (Aug. 1982), 2758–69.
- [6] M. L. Moravec, J. K. Romberg, and R. G. Baraniuk. “Compressive phase retrieval”. In: *Proc. SPIE 6701 Wavelets XII*. 2007, p. 670120.
- [7] A. V. Oppenheim and J. S. Lim. “The importance of phase in signals”. In: *Proc. IEEE* 69.5 (May 1981), 529–41.
- [8] S. Urieli, M. Porat, and N. Cohen. “Optimal reconstruction of images from localized phase”. In: *IEEE Trans. Im. Proc.* 7.6 (June 1998), 838–53.

Chapter 6

Filters for 2D signals (*i.e.*, images)

Contents (class version)

| | |
|--|-------------|
| 6.0 Introduction | 6.2 |
| Implementation | 6.5 |
| Signal support | 6.6 |
| FIR vs IIR tradeoffs | 6.7 |
| 6.1 Ideal filter specifications | 6.8 |
| Zero-phase filters | 6.18 |
| 6.2 IIR filters | 6.23 |
| Difference equations | 6.23 |
| Z-transforms in 2D | 6.24 |

6.0 Introduction

Digital filters are used extensively in image processing for tasks such as reducing noise, reducing blur caused by the imperfect PSF or an image formation system, enhancing certain features like edges, etc.

Outline

- Overview of filter design
- Ideal frequency response specifications
- Brief discussion of IIR filters, Z-transforms, difference equations
- FIR filter design basics
 - phase response, frequency response
 - symmetries

Because there are many more exciting topics in the field of image processing, our treatment here is brief. However, many such exciting applications use filters in intermediate steps so the topic of filter design is important.

Importance

In 1D signal processing, the subject of filter design via frequency response specifications is very important, and there are numerous compelling applications where one truly must carefully consider the fundamental tradeoffs between frequency response characteristics, like passband ripple and transition bandwidth, temporal characteristics (transient response time), and economic/complexity aspects (filter order) in the design of systems.

Examples? [Anti-alias filters for sound cards and audio devices](#). [Band selection for frequency-division multiplexing](#).

In 2D, it is more difficult to think of applications where (digital) filters such as the ideal lowpass are required. In fact, neither Ch. 4 nor Ch. 5 of [1] mention a single application! This is not to say that filtering is unimportant; filters are used *all the time* in image processing. But the specifications of those filters is not necessarily the kind of idealized frequency-domain characteristics considered in 1D and in [1, Ch. 4]. Often we are quite interested in the effects of the filter in the *spatial domain*, so FWHM of the impulse response may be more useful than knowing the cutoff frequency.

Of course, for anti-alias filtering in 2D we still want analog filters that are as close as possible to the ideal lowpass filter; fortunately, an ideal lens provides an ideal lowpass filter (for optical imaging) as discussed in Ch. 4.

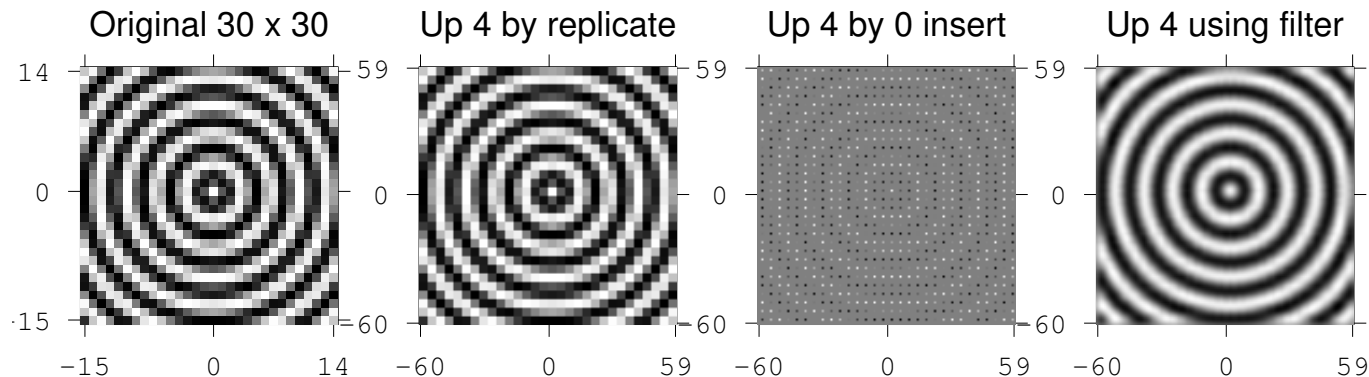
Applications: up-sampling and down-sampling

Cases where good 2D digital filters are important include up-sampling or down-sampling an image.

If we want to down-sample an image by a factor of 2 in each dimension, then, to avoid aliasing, ideally we should first apply a filter having frequency response $H(\Omega_1, \Omega_2) = \text{rect}_2\left(\frac{\Omega_1}{\pi}, \frac{\Omega_2}{\pi}\right)_{(2\pi, 2\pi)}$ prior to down sampling. To understand the need for such a filter, study the HW problem on the DSFT property for down-sampling.

Similarly, if we up-sample an image by a factor of 2 in each dimension using zero insertion, then afterwards, to remove the “aliased” replicates, ideally we should apply a filter having frequency response $H(\Omega_1, \Omega_2) = 4 \text{rect}_2\left(\frac{\Omega_1}{\pi}, \frac{\Omega_2}{\pi}\right)_{(2\pi, 2\pi)}$.

Example. The following figure shows the DS signal $g[m, n] = \sin(0.6\pi\sqrt{m^2 + n^2})$ followed by 4-fold up-sampled versions using replication and by zero insertion. The rightmost figure is the result of convolving $g_{\uparrow 4}[m, n]$ with (a 15×15 truncated version of) the low-pass filter having frequency response $H(\Omega_1, \Omega_2) = 4^2 \text{rect}_2\left(\frac{\Omega_1}{\pi/2}, \frac{\Omega_2}{\pi/2}\right)_{(2\pi, 2\pi)}$. For this (nearly band-limited) signal, clearly using a low-pass filter that is close to the ideal one produces a result that is preferable to basic pixel replication.



This is one of many examples where the *theory* is essential to get the right *practical* results. The “theory” here is analyzing the DSFT for up-sampling (a HW problem) and then (using the convolution property of the DSFT) determining what filter frequency response is needed to eliminate undesired spectral replicates (a kind of aliasing).

Implementation

One can implement filters using

- direct convolution (practical for FIR filters where double sum is finite):

$$g[m, n] = \sum_k \sum_l h[k, l] f[m - k, n - l]$$

- difference equations (very popular for 1D IIR filters; far less common in 2D)

$$g[m, n] = \sum_k \sum_l a_{kl} g[m - k, n - l] + \sum_k \sum_l b_{kl} f[m - k, n - l] \quad (6.1)$$

- **FFT** (See Ch. 7)

$$x[m, n] \rightarrow \boxed{\text{FFT}} \rightarrow X[k, l] \rightarrow \begin{array}{c} \otimes \\ \uparrow \\ H[k, l] \end{array} \rightarrow Y[k, l] \rightarrow \boxed{\text{iFFT}} \rightarrow y[m, n]$$

- **DSFT**, which looks fine on paper but is impractical. Why? ??

[RQ]

$$f[m, n] \rightarrow \boxed{\text{DSFT}} \rightarrow F(\Omega_1, \Omega_2) \rightarrow \begin{array}{c} \otimes \\ \uparrow \\ H(\Omega_1, \Omega_2) \end{array} \rightarrow G(\Omega_1, \Omega_2) \rightarrow \boxed{\text{inverse DSFT}} \rightarrow g[m, n]$$

Signal support

In 1D signal processing, filters are often operated “perpetually” on real-time signals like audio, *i.e.*, signals that are essentially of “infinite duration” (or at least indefinite duration) from the perspective of the filter.

Example. Cathedral reverberation effect in a home audio system. This effect is inherently IIR.

In contrast, the (spatial) support of digital images is almost always inherently finite. This characteristic changes some of the “rules” that one learns in 1D filter design. For example, zero-phase IIR filters are noncausal and therefore impractical for 1D real-time applications like audio. In contrast, for image processing one can apply an IIR filter recursively from left to right and then back from right to left, and the resulting overall response will be zero phase. This technique is used frequently for signal and image processing using **spline** methods [2]. (See Ch. 8.)

FIR vs IIR tradeoffs

Filters can be **finite impulse response (FIR)** or **infinite impulse response (IIR)**.

- FIR filters are always (BIBO) **stable**.
- FIR filters are easily made **zero phase** simply by designing them to be **symmetric**: $h[m, n] = h[-m, -n]$.
Such designs will be “noncausal” but that property is acceptable for image processing.
- Many techniques for 1D FIR filter design can be extended to 2D FIR filter design, including the windowing method and the frequency sampling method.
- Extending the Remez algorithm for min-max filter design is less straightforward.
- For detailed discussion of FIR filter design, see [1, Ch. 4].

Practical IIR filters are implemented using simple recursions (6.1) (*not* by convolution). For given specifications on the frequency response (passband ripple, etc.), an efficiently implemented IIR filter requires fewer operations (additions and multiplications) than the corresponding FIR filter. However, zero phase design of IIR filters is more challenging, and testing for stability is quite complicated in 2D.

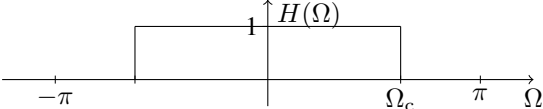
Often 2D IIR filters are **separable**, *i.e.*, $h[m, n] = h_1[m] h_2[n]$.

In these cases often one can use conventional 1D IIR filter design techniques for designing $h_1[m]$ and $h_2[n]$.

6.1 Ideal filter specifications

Review of 1D ideal lowpass filter

In 1D, an **ideal digital lowpass filter** (also called a **brick-wall filter**) has frequency response (6 different notations here!)

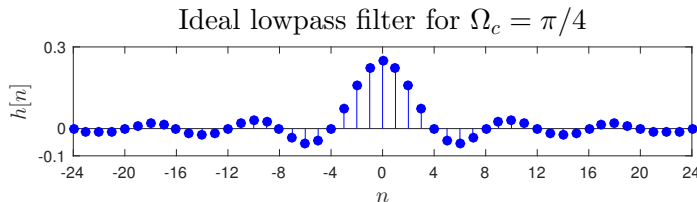
$$H(\Omega) = \begin{cases} 1, & |\Omega| < \Omega_c \\ 0, & \Omega_c \leq |\Omega| \leq \pi \\ \text{periodic,} & \text{otherwise} \end{cases} = \sum_{k=-\infty}^{\infty} \text{rect}\left(\frac{\Omega - 2\pi k}{2\Omega_c}\right)$$


Assuming $|\Omega_c| < \pi$, often people write this simply as “ $H(\Omega) = \text{rect}\left(\frac{\Omega}{2\Omega_c}\right)$,” but to be precise we must remember that this formula is correct only for $-\pi \leq \Omega \leq \pi$ because $H(\Omega)$ must be 2π periodic. To remind us of that, we sometimes use the shorthand $H(\Omega) = \text{rect}\left(\frac{\Omega}{2\Omega_c}\right)_{(2\pi)}$ to avoid writing the precise but cumbersome form $H(\Omega) = \text{rect}\left(\frac{\text{mod}(\Omega + \pi, 2\pi) - \pi}{2\Omega_c}\right)$.

The corresponding impulse response of this ideal lowpass filter is

$$h[n] = \frac{\Omega_c}{\pi} \text{sinc}\left(\frac{\Omega_c}{\pi} n\right).$$

The following figure illustrates (part of) this IIR filter for the case $\Omega_c = \pi/4$.



The corresponding ideal highpass filter has frequency response $H_{\text{hp}}(\Omega) = 1 - H(\Omega)$ and has corresponding impulse response $h_{\text{hp}}[n] = \delta[n] - h[n] = \delta[n] - \frac{\Omega_c}{\pi} \text{sinc}\left(\frac{\Omega_c}{\pi} n\right)$.

These filters are impractical because they are infinitely long, but they provide a starting point for other 1D filter design methods.

2D ideal filters

Now we generalize such ideal filters to 2D [1, Section 4.2]. Because the frequency response $H(\Omega_1, \Omega_2)$ of a 2D DS filter is $(2\pi, 2\pi)$ -periodic, it suffices to specify the desired frequency response $H(\Omega_1, \Omega_2)$ over the region $[-\pi, \pi) \times [-\pi, \pi)$.

In fact, in the usual case where the impulse response $h[m, n]$ is real, the frequency response is Hermitian symmetric: $H(\Omega_1, \Omega_2) = H^*(-\Omega_1, -\Omega_2)$, so it suffices to specify the desired response over the region $[-\pi, \pi) \times [0, \pi)$.

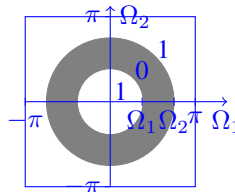
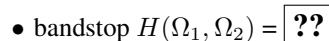
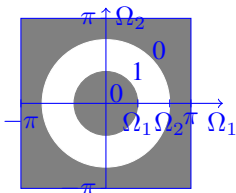
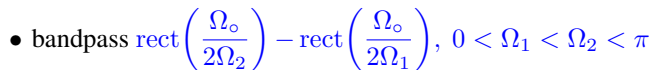
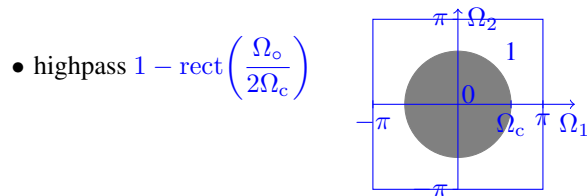
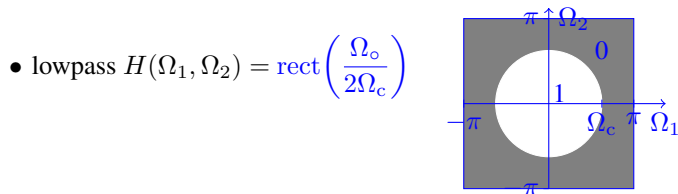
Circularly symmetric filters

In 2D, often we would like filters with **circularly symmetric** frequency responses, so that digital filtering is **rotationally invariant**.

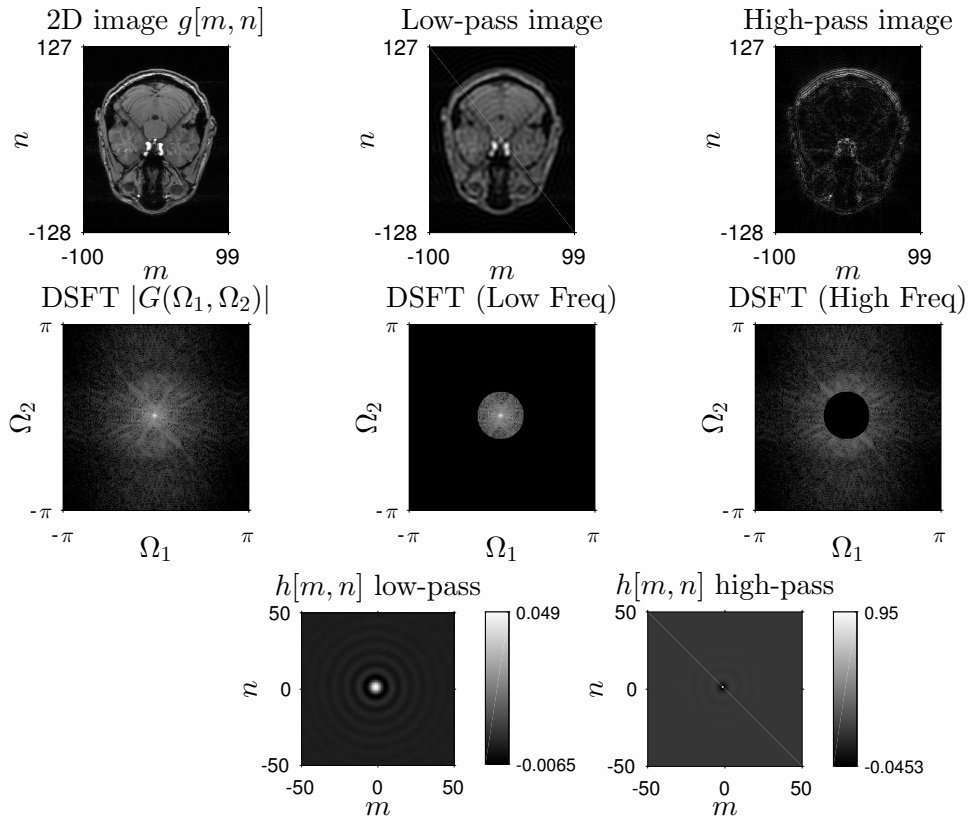
Recall from page 5.65 that the requirements on $H(\Omega_1, \Omega_2)$ for rotation invariance (in an end-to-end analog system) depend on the anti-alias filter used in the A/D converter. For simplicity here, assume that $\Delta_x = \Delta_y$ in the sampler on page 5.63 and that the anti-alias filter has “circular” frequency response $\text{rect}(\rho\Delta_x)$ (like an ideal lens) so that the frequency response $H(\Omega_1, \Omega_2)$ of the digital filter is irrelevant outside the circle of radius π .

What are the frequency responses of the following ideal, circularly symmetric filters?

Throughout, define the “radial coordinate in $[-\pi, \pi) \times [-\pi, \pi)$ ” as $\Omega_o \triangleq \sqrt{(\text{mod}(\Omega_1 + \pi, 2\pi) - \pi)^2 + (\text{mod}(\Omega_2 + \pi, 2\pi) - \pi)^2}$.



This figure illustrates the effects of ideal low-pass and high-pass filters on a MR brain image.



To determine the ideal impulse responses of such filters, recall the CS Hankel transform pair:

$$g(r) = 4\rho_0^2 \text{jinc}(2\rho_0 r) \xleftrightarrow{\mathcal{F}_2} G(\rho) = \text{rect}\left(\frac{\rho}{2\rho_0}\right),$$

where we have used the scaling property of the FT. So by the sampling theory relationship between CS FT and DS FT (5.15):

$$\begin{aligned} h[m, n] &= 4\rho_0^2 \text{jinc}\left(2\rho_0 \sqrt{m^2 + n^2}\right) \xleftrightarrow{\text{DSFT}} H(\Omega_1, \Omega_2) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} G\left(\frac{\Omega_1}{2\pi} - k, \frac{\Omega_2}{2\pi} - l\right) \\ &= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \text{rect}\left(\frac{\sqrt{(\Omega_1 - 2\pi k)^2 + (\Omega_2 - 2\pi l)^2}}{2\Omega_c}\right), \end{aligned}$$

where $\Omega_c = 2\pi\rho_0$ (using $\Delta = 1$). If $|\rho_0| < 1/2$ (cycles / sample), which we assume hereafter, then the replicates in the double sum do not overlap and for $(\Omega_1, \Omega_2) \in [-\pi, \pi) \times [-\pi, \pi)$ we can write the following more concise expression for the central replicate:

$$4\rho_0^2 \text{jinc}\left(2\rho_0 \sqrt{m^2 + n^2}\right) \xleftrightarrow{\text{DSFT}} G\left(\frac{\Omega_o}{2\pi}\right) = \text{rect}\left(\frac{\Omega_o}{2\Omega_c}\right), \quad \Omega_o \triangleq \sqrt{\Omega_1^2 + \Omega_2^2}.$$

Thus, letting $\rho_0 = \Omega_c/(2\pi)$, the impulse responses of the ideal filters described above are

- lowpass

$$h[m, n] = \left(\frac{\Omega_c}{\pi}\right)^2 \text{jinc}\left(\frac{\Omega_c}{\pi} \sqrt{m^2 + n^2}\right) \xleftrightarrow{\text{DSFT}} H(\Omega_1, \Omega_2) = \text{rect}\left(\frac{\Omega_o}{2\Omega_c}\right) \quad (6.2)$$

- highpass (assuming A/D anti-alias filter has circular frequency response)

$$\delta_2[m, n] - \left(\frac{\Omega_c}{\pi}\right)^2 \text{jinc}\left(\frac{\Omega_c}{\pi} \sqrt{m^2 + n^2}\right) \xleftrightarrow{\text{DSFT}} 1 - \text{rect}\left(\frac{\Omega_o}{2\Omega_c}\right)$$

- bandpass $0 \leq \Omega_l \leq \Omega \leq \Omega_h \leq \pi$

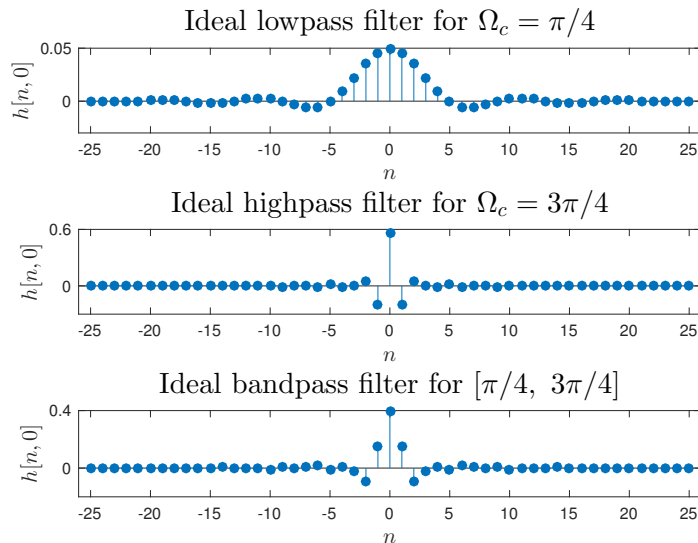
$$\left(\frac{\Omega_h}{\pi}\right)^2 \text{jinc}\left(\frac{\Omega_h}{\pi} \sqrt{m^2 + n^2}\right) - \left(\frac{\Omega_l}{\pi}\right)^2 \text{jinc}\left(\frac{\Omega_l}{\pi} \sqrt{m^2 + n^2}\right) \stackrel{\text{DSFT}}{\longleftrightarrow} \text{rect}\left(\frac{\Omega_o}{2\Omega_h}\right) - \text{rect}\left(\frac{\Omega_o}{2\Omega_l}\right).$$

- If the anti-alias filter has the “square” frequency response $\text{rect}_2(\nu_x \Delta_x, \nu_y \Delta_y)$, then replacing the highpass filter with a bandpass filter having high-frequency cutoff $\Omega_h = \pi$ (removing the “corners” of the $[-\pi, \pi) \times [-\pi, \pi)$ square) ensures rotation invariance.

Unfortunately, just as in 1D, all of these ideal impulse response functions are IIR. So we must make approximations in practice. For example, we can **apodize** these filters by applying a **window** to taper and truncate them. See MATLAB’s `fwind2`.

See [1, Ch. 4] for discussion of filter specifications allowing passband ripple, transition bands, etc.

The following figure shows profiles $h[n, 0]$ through each of these ideal filters.



Practical use of circular symmetry

Because ideal circularly symmetric filters are IIR, in practice one often applies such filters “in the Fourier domain,” using the 2D FFT as discussed in Ch. 7. *This does not completely circumvent the problem of an IIR impulse response however!* As discussed in Ch. 7, sampling the DSFT of an IIR signal and then computing the inverse DFT results in a **spatially aliased** version of the signal. Similarly, using samples of the DSFT as DFT-based filter coefficients results in a final output signal $y[m, n]$ that is the **circular convolution** of the input $x[m, n]$ with the spatially aliased impulse response $h[[m, n]]_{(M, N)}$.

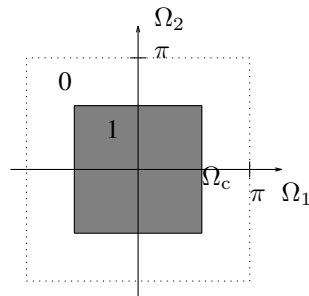
Nevertheless, we do it all the time! We simply choose N and M large enough to make spatial aliasing (wrap around) negligible.

Separable filters

If we are willing to abandon the goal of rotation invariance, then we can reduce computation by using **separable** filters.

Example. An “ideal” separable lowpass filter with cutoff frequency Ω_c has impulse response:

$$h[m, n] = \left(\frac{\Omega_c}{\pi}\right)^2 \operatorname{sinc}\left(\frac{\Omega_c}{\pi}m\right) \operatorname{sinc}\left(\frac{\Omega_c}{\pi}n\right) \stackrel{\text{DSFT}}{\longleftrightarrow} H(\Omega_1, \Omega_2) = \operatorname{rect}\left(\frac{\Omega_1}{2\Omega_c}\right) \operatorname{rect}\left(\frac{\Omega_2}{2\Omega_c}\right).$$



As usual, the above expression for $H(\Omega_1, \Omega_2)$ is a shorthand that is valid only for $(\Omega_1, \Omega_2) \in [-\pi, \pi) \times [-\pi, \pi)$ and we assume that $|\Omega_c| < \pi$ so that the replicates do not overlap.

What is the frequency response $H(\Omega_1, \Omega_2)$ of the corresponding highpass filter?

??

[RQ]

What is its corresponding impulse response?

$$h[m, n] = \delta_2[m, n] - \left(\frac{\Omega_c}{\pi}\right)^2 \operatorname{sinc}\left(\frac{\Omega_c}{\pi}m\right) \operatorname{sinc}\left(\frac{\Omega_c}{\pi}n\right).$$

Is this highpass filter separable? ??

[RQ]

A possible alternative “highpass” filter is: $h[m, n] = \left(\delta[m] - \left(\frac{\Omega_c}{\pi}\right) \operatorname{sinc}\left(\frac{\Omega_c}{\pi} m\right)\right) \left(\delta[n] - \left(\frac{\Omega_c}{\pi}\right) \operatorname{sinc}\left(\frac{\Omega_c}{\pi} n\right)\right)$.

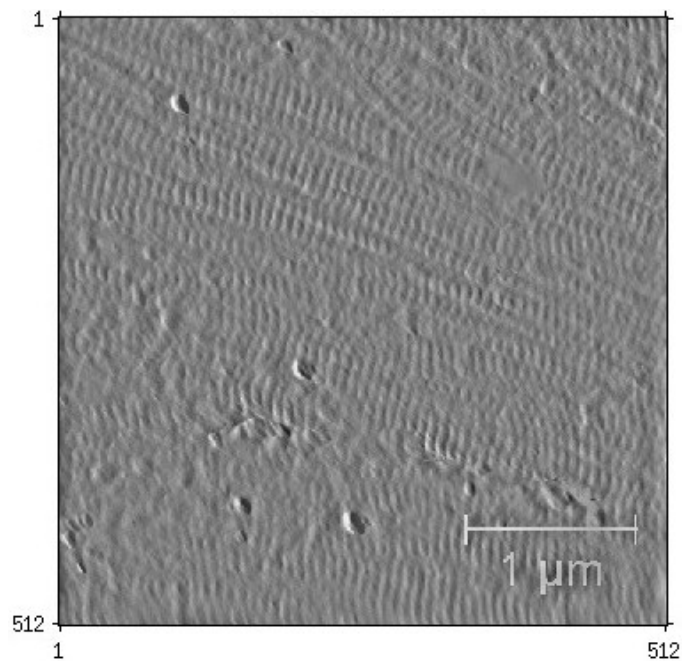
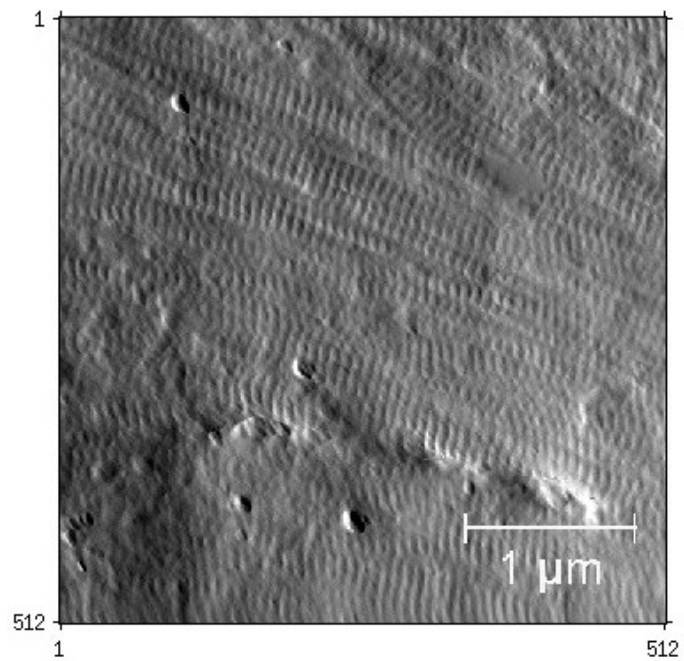
What is the frequency response of this filter? $H(\Omega_1, \Omega_2) = \left(1 - \operatorname{rect}\left(\frac{\Omega_1}{2\Omega_c}\right)\right) \left(1 - \operatorname{rect}\left(\frac{\Omega_2}{2\Omega_c}\right)\right)$.

Can one make a **separable** bandpass filter? ??

Although the expressions here are for ideal sinc-type filters, the issues of separability also apply to practical FIR filters.

High-pass filtering example (skim)

Example. The following pair of figures illustrates one of many applications of image filtering. The left image is of collagen using atomic force microscopy (AFM), courtesy of Prof. Banaszak-Holl in Chemistry. There are low-frequency variations (darker and lighter shaded regions) that make it difficult to detect the fiber spacing. Applying a high-pass filter yields the image on the right, where those low-frequency variations are removed. It is easier to extract the quasi-periodic collagen microstructure features from this processed image.



Zero-phase filters (See [1, Section 4.1].)

The frequency response $H(\Omega_1, \Omega_2)$ of a filter has both a **magnitude response** component $|H(\Omega_1, \Omega_2)|$ and a **phase response** component $\angle H(\Omega_1, \Omega_2)$.

For 1D monophonic speech processing, filter phase response is relatively unimportant compared to the magnitude response [1]. However, phase is very important for *stereo* effects (two ears), *e.g.*, the ability to perform spatial localization.

For image processing, if different frequency components have different shifts (*i.e.*, become mis-aligned after filtering), then the image appears quite distorted to the human eye (even though the magnitude response may equal that of another image that is much more interpretable) [1, Fig. 4.1].

So it is highly desirable to design and use **linear phase** filters in image processing. In fact, because causality is not an issue in image processing, we can even work with so-called **zero-phase** filters, defined as filters whose magnitude response is *real*:

$$H(\Omega_1, \Omega_2) = H^*(\Omega_1, \Omega_2).$$

Calling this zero phase is slightly imprecise, because $H(\Omega_1, \Omega_2)$ can be negative so its phase could of course be $\pm\pi$. But over the passband of the filter the frequency response will typically be positive, so the term sticks.

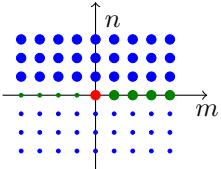
The above condition in turn implies that the impulse response be Hermitian symmetric:

$$h[-m, -n] = h^*[m, n].$$

We usually restrict attention to real filters, so the condition for zero phase reduces to **even symmetry** of the impulse response

$$h[-m, -n] = h[m, n].$$

If the impulse response has **even symmetry**, then we can “simplify” the expression for its **frequency response** by writing it in terms of cosine functions instead of complex exponentials, by splitting up the sum in ways analogous to (2.13) as follows:



$$\begin{aligned}
 H(\Omega_1, \Omega_2) &= \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} h[m, n] e^{-i(\Omega_1 m + \Omega_2 n)} \\
 &= h[0, 0] + \sum_{m=1}^{\infty} h[m, 0] e^{-i\Omega_1 m} + \sum_{m=-\infty}^{-1} h[m, 0] e^{-i\Omega_1 m} \\
 &+ \sum_{m=-\infty}^{\infty} \sum_{n=1}^{\infty} h[m, n] e^{-i(\Omega_1 m + \Omega_2 n)} + \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{-1} h[m, n] e^{-i(\Omega_1 m + \Omega_2 n)} \\
 &= h[0, 0] + \sum_{m=1}^{\infty} 2h[m, 0] \cos(\Omega_1 m) + \sum_{m=-\infty}^{\infty} \sum_{n=1}^{\infty} 2h[m, n] \cos(\Omega_1 m + \Omega_2 n).
 \end{aligned} \tag{6.3}$$

We have seen this “sum of cosines” form of $H(\Omega_1, \Omega_2)$ in almost all of the filter examples considered thus far. The exception was the differentiator—it has odd symmetry.

Example. If $h[m, n]$ is any FIR filter, then its frequency response is a finite sum of cos terms like in (6.3) above.

For example, if $h[m, n] = \text{rect}_2\left(\frac{m}{5}, \frac{n}{5}\right)$ then $H(\Omega_1, \Omega_2) = 1 + \sum_{m=1}^2 2 \cos(\Omega_1 m) + \sum_{m=1}^2 \sum_{n=-2}^2 2 \cos(\Omega_1 m + \Omega_2 n)$, which is a sum of 13 terms all involving cos with arguments Ω_1 and/or Ω_2 multiplied by integers in the set $\{0, \pm 1, \pm 2\}$ corresponding to the support of the filter.

Example. Find the impulse response $h[m, n]$ of the filter having the (additively separable) frequency response $H(\Omega_1, \Omega_2) = 2 \cos(5\Omega_1) + 4 \cos(7\Omega_2)$.

(Caution: there is no “additive separability” property of the DSFT.)

Using linearity and separability: $h[m, n] = \delta_2[m - 5, n] + \delta_2[m + 5, n] + 2 \delta_2[m, n - 7] + 2 \delta_2[m, n + 7]$.

Computational saving using symmetries

One can exploit filter symmetry $h[-n] = h[n]$ to reduce computation when implementing direct convolution. The basic idea is easily illustrated in 1D:

$$\begin{aligned}
 y[n] &= h[n] * x[n] \\
 &= \sum_{k=-\infty}^{\infty} h[k] x[n-k] \\
 &= h[0] x[n] + \sum_{k=1}^{\infty} h[k] x[n-k] + \sum_{k=-\infty}^{-1} h[k] x[n-k] \\
 &= h[0] x[n] + \sum_{k=1}^{\infty} h[k] x[n-k] + \sum_{k=1}^{\infty} h[-k] x[n+k] \\
 &= h[0] x[n] + \sum_{k=1}^{\infty} h[k] x[n-k] + \sum_{k=1}^{\infty} h[k] x[n+k] \\
 &= h[0] x[n] + \sum_{k=1}^{\infty} h[k] (x[n-k] + x[n+k]).
 \end{aligned} \tag{6.4}$$

Because we usually use zero-phase filters in image processing, this trick is often worth the slightly increased programming effort.

Example. To illustrate the savings concretely, consider when a $h[n]$ is a symmetric 9-tap filter $h[-4], \dots, h[4]$. The conventional convolution implementation would be

$$y[n] = \sum_{k=-4}^4 h[k] x[n-k].$$

For each output sample, we must perform 9 multiplications and 8 or 9 additions. In contrast, the efficient version in (6.4) is

$$y[n] = h[0] x[n] + \sum_{k=1}^4 h[k] (x[n-k] + x[n+k])$$

which requires only 5 multiplies and 9 adds.

What part of this chapter did you find most confusing?

[RQ]

List two possible course project topics of interest to you.

[RQ]

6.2 IIR filters

See [1, Ch. 2 and 5].

Difference equations**(skim)**

IIR filters are characterized by difference equations. This is a very important topic in 1D signal processing.

One can represent a 2D IIR filter by a 2D DS linear, constant-coefficient **difference equation**:

$$\sum_{(k,l) \in \mathcal{R}_a} a_{k,l} g[m-k, n-l] = \sum_{(k,l) \in \mathcal{R}_b} b_{k,l} f[m-k, n-l],$$

where \mathcal{R}_a and \mathcal{R}_b are subsets of \mathbb{Z}^2 , and $a_{k,l}$ and $b_{k,l}$ are coefficients.

Typically $a_{0,0} = 1$ and $(0,0) \in \mathcal{R}_a$. When is such a filter FIR? **When $\mathcal{R}_a = \{(0,0)\}$.**

Such systems are inherently **linear** and **shift-invariant**. However, IIR 2D difference equations are used only infrequently in image processing.

Z-transforms in 2D**(skim)**

The primary tool for analyzing a **difference equation** is the **Z-transform**.

By the fundamental theorem of algebra, any 1D polynomial can be factored in terms of its roots. So to check for **stability** we simply verify that the roots of the characteristic polynomial associated with the difference equation all lie within the unit circle.

In 2D, there is not a corresponding guarantee that one can factor a 2D polynomial! Furthermore, the “roots” are surfaces, not points. So verifying stability of a general 2D IIR filter is much more challenging [1, Ch. 2].

From [1, p. 124]

...testing the stability of a 2-D system is considerably more complex than testing the stability of a 1-D system. The complexity of testing a 2-D system’s stability explains, in part, why 2-D FIR digital filters, which are always stable, are much preferred over 2-D IIR digital filters in practice. The preference for FIR filters over IIR filters is much more marked in 2-D than in 1-D signal processing applications.

For certain specific forms of IIR filters, checking stability is easy. For example, the stability of any **separable** IIR filter $h[m, n] = h_1[m] h_2[n]$ reduces to checking separately the stability of the 1D filters $h_1[m]$ and $h_2[n]$, which is easy.

But separable DS filters are not circularly symmetric...

Caution: an expert colleague claims there are mistakes in [1, Ch. 2].

Bibliography

- [1] J. S. Lim. *Two-dimensional signal and image processing*. New York: Prentice-Hall, 1990. ISBN: 978-0139353222.
- [2] M. Unser. “Splines: A perfect fit for signal and image processing”. In: *IEEE Sig. Proc. Mag.* 16.6 (Nov. 1999), 22–38.

Chapter 7

2D DFT

Contents (class version)

| | |
|---|-------------|
| 7.0 Introduction to discrete transforms | 7.2 |
| 7.1 2D discrete-space orthogonal representation | 7.4 |
| 7.2 2D discrete Fourier series (DFS) | 7.6 |
| 7.3 2D Discrete Fourier transform (DFT) | 7.18 |
| Properties of the DFT | 7.25 |
| Sampling the DSFT and “aliasing” | 7.39 |
| Matrix representations of DFT in 1D and 2D | 7.45 |
| 7.4 Fast Fourier transforms (FFT) in 2D | 7.54 |
| 7.5 Relations between various Fourier transforms | 7.56 |
| 2D Fourier transform summary | 7.57 |
| Using FFT to approximate analog spectra | 7.61 |
| FFT and <code>fftshift</code> | 7.64 |
| Summary of key relationships | 7.65 |
| 7.6 Numerical evaluation of Fourier transforms via the FFT | 7.69 |
| Numerical evaluation of CSFT via the FFT | 7.70 |
| 7.7 Design of FIR filters by frequency sampling | 7.78 |
| 7.8 Discrete cosine transform (DCT) | 7.85 |
| 1D DCT | 7.86 |

| | |
|--------------------------|-------------|
| 2D DCT | 7.91 |
| 7.9 Summary | 7.94 |

7.0 Introduction to discrete transforms

This chapter continues our “EECS 351 in 2D” coverage. See [1, Ch. 3] and [2].

- The 1D DFT and 2D DFT are very similar concepts.
- Other than separability, there is little new in 2D.
- It is rarely the case that the DFT is *really* the operation we want!

Usually we want to do things like:

- perform (linear) convolution (*i.e.*, filter an image), or
- determine the spectrum of an analog image that was sampled.
- We use the DFT to perform these tasks because we can do so quickly via an FFT.
- We will focus on how we adapt the DFT to these types of problems, overcoming its limitations approximately or exactly.

This is a long chapter because the FFT is a particularly important tool in signal and image processing, having numerous applications.

A basic equality that we will use repeatedly here is the following geometric series for complex exponentials, sometimes called a **trigonometric identity**:

$$\sum_{n=0}^{N-1} e^{i\Omega n} = \sum_{n=0}^{N-1} (e^{i\Omega})^n = \begin{cases} N, & e^{i\Omega} = 1 \\ \frac{1 - (e^{i\Omega})^N}{1 - e^{i\Omega}}, & e^{i\Omega} \neq 1 \end{cases} = \begin{cases} N, & \Omega = 0, \pm 2\pi, \pm 4\pi, \dots \\ \frac{1 - e^{i\Omega N}}{1 - e^{i\Omega}}, & \text{otherwise.} \end{cases}$$

A particularly important special case of this general formula is when $\Omega = 2\pi \frac{k}{N}$ for $k \in \mathbb{Z}$ for which we have:

$$\sum_{n=0}^{N-1} e^{i \frac{2\pi}{N} kn} = \begin{cases} N, & k = 0, \pm N, \pm 2N, \dots \\ 0, & \text{otherwise} \end{cases} = N \sum_{l=-\infty}^{\infty} \delta[k - lN] = N \delta[k \bmod N]. \quad (7.1)$$

Note that the following ratio is related to the **Dirichlet kernel**:

$$\frac{1 - e^{i\Omega N}}{1 - e^{i\Omega}} = \frac{e^{i\Omega N} - 1}{e^{i\Omega} - 1} = \frac{e^{i\Omega N/2} e^{i\Omega N/2} - e^{-i\Omega N/2}}{e^{i\Omega/2} e^{i\Omega/2} - e^{-i\Omega/2}} = e^{i\Omega(N-1)/2} \frac{\sin(\Omega N/2)}{\sin(\Omega/2)}.$$

Caution. Wikipedia and MATLAB use slightly different definitions of the **Dirichlet kernel**. In MATLAB:

$$\text{diric}(x, N) = \begin{cases} \frac{\sin(xN/2)}{N \sin(x/2)}, & x \text{ not a multiple of } 2\pi \\ \pm 1, & \text{otherwise (depends on limit).} \end{cases}$$

7.1 2D discrete-space orthogonal representation

To represent 2D DS signals with an orthogonal basis, we need an appropriate definition of **inner product**.

For discrete-space signals that are either

- periodic with period (M, N) ,
- or finite-extent, *i.e.*, nonzero only for $[m, n] \in \mathcal{D}_{M,N}$, where we define the domain notation

$$\mathcal{D}_{M,N} \triangleq \{0, \dots, M-1\} \times \{0, \dots, N-1\}, \quad (7.2)$$

the following **inner product** is appropriate:

$$\langle f, g \rangle = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f[m, n] g^*[m, n]. \quad (7.3)$$

There are many **orthogonal bases** with respect to this inner product.

Example. $\phi_{k,l}[m, n] = \delta_2[m - k, n - l]$, for $k = 0, \dots, M-1$, $l = 0, \dots, N-1$, is a trivial orthonormal basis on $\mathcal{D}_{M,N}$.

However, this basis does not provide any “new information” about the signal because the coefficients in the corresponding “synthesis” expression are just the signal samples:

$$f[m, n] = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} f[k, l] \delta_2[m - k, n - l].$$

Complex exponential signals seem to be a more desirable choice of basis because they are eigenfunctions of LSI systems. On the other hand, using complex signals to represent real-valued images can be inconvenient.

Of course there are many other bases, many of which are separable and some have even been put in hardware [3].

Discrete Fourier series basis

The discrete Fourier series basis for $\mathcal{D}_{M,N}$ is given as follows. We scale by $1/MN$ to match the DFT convention:

$$\phi_{k,l}[m,n] = \frac{1}{MN} e^{i2\pi\left(\frac{k}{M}m + \frac{l}{N}n\right)} = \frac{1}{MN} e^{i\frac{2\pi k}{M}m} e^{i\frac{2\pi l}{N}n}, \quad (k,l) \in \mathcal{D}_{M,N}.$$

This is a **separable** 2D basis.

We use (7.1) to verify that the members of this set of signals are orthogonal for $(k,l) \in \mathcal{D}_{M,N}$ and $(k',l') \in \mathcal{D}_{M,N}$:

$$\begin{aligned} \langle \phi_{k,l}, \phi_{k',l'} \rangle &= \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \phi_{k,l}[m,n] \phi_{k',l'}^*[m,n] \\ &= \frac{1}{(MN)^2} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} e^{i\frac{2\pi}{M}(k-k')m} e^{i\frac{2\pi}{N}(l-l')n} = \frac{1}{MN} \delta_2[k-k', l-l']. \end{aligned} \quad (7.4)$$

If we allow any $(k,l) \in \mathbb{Z}^2$ and $(k',l') \in \mathbb{Z}^2$ then instead we find that

$$\langle \phi_{k,l}, \phi_{k',l'} \rangle = \frac{1}{MN} \delta_2[[k-k', l-l']]_{(M,N)},$$

where we introduce the shorthand for a 2D DS “bed of nails” or **Kronecker comb** function:

$$\delta_2[[m,n]]_{(M,N)} \triangleq \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \delta_2[m-kM, n-lN] = \delta_2[m \bmod M, n \bmod N]. \quad (7.5)$$

In other words, the countable set $\{\phi_{k,l} : k,l \in \mathbb{Z}\}$ is *not* an orthogonal basis, but the finite set $\{\phi_{k,l} : (k,l) \in \mathcal{D}_{M,N}\}$ is an orthogonal basis on $\mathcal{D}_{M,N}$, with each $\phi_{k,l}$ having the same energy or norm squared value:

$$E_{k,l} = \|\phi_{k,l}\|^2 = \langle \phi_{k,l}, \phi_{k,l} \rangle = \frac{1}{MN}. \quad (7.6)$$

7.2 2D discrete Fourier series (DFS)

(skim)

Given a periodic signal $\tilde{x}[m, n]$ with period (M, N) , that need not be a fundamental period, we would like to find its 2D **discrete Fourier series (DFS)** representation:

$$\tilde{x}[m, n] = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} \tilde{X}[k, l] \phi_{k,l}[m, n] = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} \tilde{X}[k, l] e^{i2\pi(\frac{k}{M}m + \frac{l}{N}n)}.$$

That is, we want to represent $\tilde{x}[m, n]$ as a sum of MN harmonically related complex exponentials having frequencies

$$(\Omega_1, \Omega_2) \in \left\{ (0, 0), \left(\frac{2\pi}{M}, 0 \right), \dots, \left(\frac{2\pi}{M}(M-1), \frac{2\pi}{N}(N-1) \right) \right\}. \quad (7.7)$$

For similarity to the DFT notation used later, we use $\tilde{X}[k, l]$ to denote the (k, l) th **DFS coefficient**, *i.e.*, the value that multiplies the exponential having frequency $(\frac{2\pi}{M}k, \frac{2\pi}{N}l)$,

From the earlier discussion of general orthogonal representations of signals and by (7.6), the DFS coefficients are given by

$$\tilde{X}[k, l] = \frac{\langle \tilde{x}[\cdot], \phi_{k,l} \rangle}{E_{k,l}} = \underbrace{MN}_{1/E_{k,l}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \tilde{x}[m, n] \underbrace{\frac{1}{MN} e^{-i2\pi(\frac{k}{M}m + \frac{l}{N}n)}}_{\phi_{k,l}^*[m, n]}.$$

Summarizing then, for a periodic signal $\tilde{x}[m, n]$ with period (M, N) , we have the following DFS representation:

$$\tilde{X}[k, l] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \tilde{x}[m, n] e^{-i2\pi\left(\frac{k}{M}m + \frac{l}{N}n\right)}, \quad (\text{analysis}) \quad (7.8)$$

$$\tilde{x}[m, n] = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} \tilde{X}[k, l] e^{i2\pi\left(\frac{k}{M}m + \frac{l}{N}n\right)}. \quad (\text{synthesis}) \quad (7.9)$$

The synthesis formula, as mentioned before, represents $\tilde{x}[m, n]$ as a linear combination of MN harmonically related complex exponential signals.

Exact equalities hold in both the analysis and synthesis formulas (as long as all signal values are finite). There are no questions of convergence here like we had for the DSFT because the sums are finite.

Generalizations**(skip)**

Due to the periodicity of $\tilde{x}[m, n]$ and the complex exponentials, we could equally well evaluate the analysis formula by summing over *any* $M \times N$ rectangular region. That is for any integers m_0, n_0 ,

$$\tilde{X}[k, l] = \sum_{m=m_0}^{m_0+M-1} \sum_{n=n_0}^{n_0+N-1} \tilde{x}[m, n] e^{-i2\pi(\frac{k}{M}m + \frac{l}{N}n)}.$$

Although the synthesis formula above represents $\tilde{x}[m, n]$ in terms of coefficients $\tilde{X}[k, l]$, $k = 0, \dots, M-1$, $l = 0, \dots, N-1$, and corresponding complex exponentials having frequencies (7.7), one could equally well represent $\tilde{x}[m, n]$ in terms of complex coefficients and complex exponentials in any $M \times N$ “box” in frequency space. That is, for any integers k_o, l_o , one could compute $\tilde{x}[m, n]$ as follows:

$$\tilde{x}[m, n] = \frac{1}{MN} \sum_{k=k_o}^{k_o+N-1} \sum_{l=l_o}^{l_o+M-1} \tilde{X}[k, l] e^{i2\pi(\frac{k}{M}m + \frac{l}{N}n)}.$$

Mathematically, we consider the DFS coefficients $\tilde{X}[k, l]$ to be defined for *all* $k, l \in \mathbb{Z}$. Note, however, that they form a periodic 2D function with period (M, N) , because $\tilde{X}[k+M, l] = \tilde{X}[k, l]$ and $\tilde{X}[k, l+N] = \tilde{X}[k, l]$. In contrast, the DFT, to be defined later, produces only a finite number of coefficients.

Because equalities hold in both the analysis and synthesis formulas, we can say there is a one-to-one correspondence between periodic signals with period (M, N) and periodic sequences of Fourier coefficients with period (M, N) .

Example. Consider the following signal

$$\begin{aligned}\tilde{x}[m, n] &= 4 \cos(\pi m/3) \cos(\pi n/4) = \left[e^{i2\pi m/6} + e^{-i2\pi m/6} \right] \left[e^{i2\pi n/8} + e^{-i2\pi n/8} \right] \\ &= e^{i2\pi(m/6+n/8)} + e^{i2\pi(m/6-n/8)} + e^{i2\pi(-m/6+n/8)} + e^{i2\pi(-m/6-n/8)}.\end{aligned}$$

What is the period? **??**

[RQ]

In this case, we need not apply the formula for $\tilde{X}[k, l]$ above, because we have directly represented $\tilde{x}[m, n]$ as a sum of complex exponentials. The coefficients in this sum are the $\tilde{X}[k, l]$ values:

$$\begin{aligned}\tilde{X}[k, l] &= \begin{cases} 48, & k \bmod 6 = \pm 1, l \bmod 8 = \pm 1 \\ 0, & \text{otherwise} \end{cases} \\ &= 48 \left(\delta_2[[k-1, l-1]]_{(6,8)} + \delta_2[[k+1, l-1]]_{(6,8)} + \delta_2[[k-1, l+1]]_{(6,8)} + \delta_2[[k+1, l+1]]_{(6,8)} \right).\end{aligned}$$

Because of the one-to-one correspondence mentioned previously, there can be one and only one representation of the signal in terms of complex exponential signals having the given range of harmonic frequencies. Therefore, the above $\tilde{X}[k, l]$ values are the one and only possible $\tilde{X}[k, l]$ values for this representation.

Example. What signal has constant coefficients? *i.e.*, $\tilde{X}[k, l] = 1$. Presumably some sort of impulse-like signal. But we are working with periodic signals, so it must be a periodic impulse-like signal. In fact it is a 2D “**impulse train**” or **Kronecker comb**:

$$\tilde{x}[m, n] = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} e^{i2\pi(\frac{k}{M}m + \frac{l}{N}n)} = \delta_2[[m, n]]_{(M,N)}.$$

See (7.1). This signal is useful for deriving properties of DFS.

Properties of the DFS **(skim)**

Most properties are analogous to those of the 2D CS FS, except the **scaling property** is absent, because scaling changes the period. (Presumably there is some such property, but it probably differs significantly from the usual scaling properties.)

- **linearity** if $\tilde{x}[m, n]$ and $\tilde{y}[m, n]$ have the same period:

$$\alpha \tilde{x}[m, n] + \beta \tilde{y}[m, n] \stackrel{\text{DFS}}{\longleftrightarrow} \alpha \tilde{X}[k, l] + \beta \tilde{Y}[k, l]$$

- **separability**

$$\tilde{x}[m, n] = \tilde{x}_1[m] \tilde{x}_2[n] \stackrel{\text{DFS}}{\longleftrightarrow} \tilde{X}_1[k] \tilde{X}_2[l],$$

where $\tilde{X}[k]$ denotes the usual 1D DFS.

- **shift**

$$\tilde{x}[m - m_0, n - n_0] \stackrel{\text{DFS}}{\longleftrightarrow} e^{-i2\pi(km_0/M + ln_0/N)} \tilde{X}[k, l]$$

- **Average value** (Caution: $\tilde{X}[0, 0]$ is often called the DC value, but it is the sum, not the average value!)

$$\frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \tilde{x}[m, n] = \frac{1}{MN} \tilde{X}[0, 0]$$

- **Parseval's theorem**

In general, if $f = \sum_{k=-\infty}^{\infty} c_k \phi_k$ and $g = \sum_{k=-\infty}^{\infty} d_k \phi_k$ where $\{\phi_k\}$ are orthogonal, then $\langle f, g \rangle = \sum_{k=-\infty}^{\infty} \mathcal{E}_k c_k d_k^*$. Thus, for the DFS:

$$\sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \tilde{x}[m, n] \tilde{y}^*[m, n] = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} \tilde{X}[k, l] \tilde{Y}^*[k, l]$$

- **Symmetry properties**

$$\tilde{x}^*[m, n] \xleftrightarrow{\text{DFS}} \tilde{X}^*[-k, -l]$$

If $\tilde{x}[m, n]$ is real, then $\tilde{X}[k, l]$ is Hermitian symmetric. And vice-versa.

- **Space reversal (mirror)**

$$\tilde{x}[-m, -n] \xleftrightarrow{\text{DFS}} \tilde{X}[-k, -l] \quad (7.10)$$

- **Duality**

(HW!)

$$\text{If } \tilde{x}[m, n] \xleftrightarrow{\text{DFS}} \tilde{X}[k, l], \text{ then } \tilde{X}[m, n] \xleftrightarrow{\text{DFS}} \boxed{??} \text{ \& } \tilde{X}^*[m, n] \xleftrightarrow{\text{DFS}} \boxed{??}$$

- **Change of period**

If a signal $\tilde{x}[m, n]$ is periodic with period (M, N) , then it is also periodic with period (m_0M, n_0N) for any positive integers m_0, n_0 . Therefore, it can also be represented with a DFS in terms of $m_0M \times n_0N$ complex exponentials with frequencies $(0, 0)$, $(\frac{2\pi}{m_0M}, 0)$, \dots , $(\frac{2\pi}{m_0M}(m_0M - 1), \frac{2\pi}{n_0N}(n_0N - 1))$. With $\tilde{X}[k, l]$ representing the usual DFS coefficients, the “new” DFS coefficients are

$$\hat{X}[k, l] = \begin{cases} \tilde{X}[k/m_0, l/n_0], & \text{if } k/m_0 \text{ and } l/n_0 \text{ are integers} \\ 0, & \text{otherwise} \end{cases} = \tilde{X}_{\uparrow(m_0, n_0)}[k, l].$$

Thus, the “new” DFS coefficients are just the original ones with zeros in between.

- Pairs of Hermitian DFS terms form sinusoids for real signals:

$$\tilde{X}[k, l] e^{i2\pi(\frac{k}{M}m + \frac{l}{N}n)} + \tilde{X}[-k, -l] e^{-i2\pi(\frac{k}{M}m + \frac{l}{N}n)} = 2 \left| \tilde{X}[k, l] \right| \cos\left(2\pi\left(\frac{k}{M}m + \frac{l}{N}n\right) + \angle \tilde{X}[k, l]\right).$$

“Convolution” property of DFS (skim)

For the DSFT, we know that $h[m, n] ** x[m, n] \xleftrightarrow{\text{DSFT}} H(\Omega_1, \Omega_2) X(\Omega_1, \Omega_2)$, i.e., space-domain convolution corresponds to frequency-domain multiplication, which is very useful for filtering without performing convolution.

What is the corresponding result for the DFS?

(The corresponding result *cannot* be “ $\tilde{h}[m, n] ** \tilde{x}[m, n]$ ” because linear convolution of two nonzero periodic signals results in infinite or undefined values.)

Suppose we have two periodic signals $\tilde{h}[m, n]$ and $\tilde{x}[m, n]$ with the same period (M, N) . If we multiply their DFS coefficients to form $\tilde{Y}[k, l] = \tilde{H}[k, l] \tilde{X}[k, l]$ and then compute the inverse DFS to get a signal $\tilde{y}[m, n]$, what is the relationship between $\tilde{y}[m, n]$ and the original $\tilde{h}[m, n]$ and $\tilde{x}[m, n]$? The following derivation shows that the result is (M, N) -point **periodic convolution**, just as in the 1D case:

$$\begin{aligned}
 \tilde{y}[m, n] &= \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} \tilde{Y}[k, l] e^{i2\pi(\frac{k}{M}m + \frac{l}{N}n)} \\
 &= \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} \tilde{H}[k, l] \tilde{X}[k, l] e^{i2\pi(\frac{k}{M}m + \frac{l}{N}n)} \\
 &= \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} \left[\sum_{m'=0}^{M-1} \sum_{n'=0}^{N-1} \tilde{h}[m', n'] e^{-i2\pi(\frac{k}{M}m' + \frac{l}{N}n')} \right] \\
 &\quad \cdot \left[\sum_{m''=0}^{M-1} \sum_{n''=0}^{N-1} \tilde{x}[m'', n''] e^{-i2\pi(\frac{k}{M}m'' + \frac{l}{N}n'')} \right] e^{i2\pi(\frac{k}{M}m + \frac{l}{N}n)} \\
 &= \sum_{m'=0}^{M-1} \sum_{n'=0}^{N-1} \sum_{m''=0}^{M-1} \sum_{n''=0}^{N-1} \tilde{h}[m', n'] \tilde{x}[m'', n''] \left[\frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} e^{-i2\pi(\frac{k}{M}(m'+m''-m) + \frac{l}{N}(n'+n''-n))} \right]
 \end{aligned}$$

$$\begin{aligned}
&= \sum_{m'=0}^{M-1} \sum_{n'=0}^{N-1} \tilde{h}[m', n'] \left[\sum_{m''=0}^{M-1} \sum_{n''=0}^{N-1} \tilde{x}[m'', n''] \delta_2[[m' + m'' - m, n' + n'' - n]]_{(M,N)} \right] \\
&= \sum_{m'=0}^{M-1} \sum_{n'=0}^{N-1} \tilde{h}[m', n'] \tilde{x}[m - m', n - n'],
\end{aligned}$$

where the last equality follows from the fact that the sum over m'', n'' has only MN terms, but $\delta_2[[\cdot]]_{(M,N)}$ is zero for only one of every MN terms, and the periodicity of $\tilde{h}[m, n]$.

Thus we have the **convolution property** of the DFS:

$$\tilde{y}[m, n] = \tilde{h}[m, n] \widetilde{\otimes}_{M,N} \tilde{x}[m, n] \triangleq \sum_{m'=0}^{M-1} \sum_{n'=0}^{N-1} \tilde{h}[m - m', n - n'] \tilde{x}[m', n'] \stackrel{\text{DFS}}{\longleftrightarrow} \tilde{Y}[k, l] = \tilde{H}[k, l] \tilde{X}[k, l]. \quad (7.11)$$

This is called **periodic convolution**¹, because all signals involved are periodic. Note that the summations above are *finite*. In ordinary **linear convolution**, the summations are infinite (and at most only one of the two signals can be periodic). These notes denote (M, N) -point **periodic convolution** using the symbol $\widetilde{\otimes}_{M,N}$ or simply $\widetilde{\otimes}$.

¹ The literature uses the terms **periodic convolution**, **circular convolution** and **cyclic convolution** largely synonymously [wiki]. In 2D, a more “geometrically accurate” term might be “**toroidal convolution**” but this term has been used only rarely [4, 5]. Here, generally we use the term **periodic convolution** when all signals involved are periodic, and **circular convolution** when all signals involved have finite support.

In summary, the (periodic) convolution property of the DFS is:

$$\tilde{h}[m, n] \otimes \widetilde{\tilde{x}[m, n]} \xleftrightarrow{\text{DFS}} \tilde{H}[k, l] \tilde{X}[k, l].$$

Although it was stated earlier that one could use the DFS for either periodic or finite-support signals, the above property holds *only* for periodic signals.

Multiplication property _____ (skim)

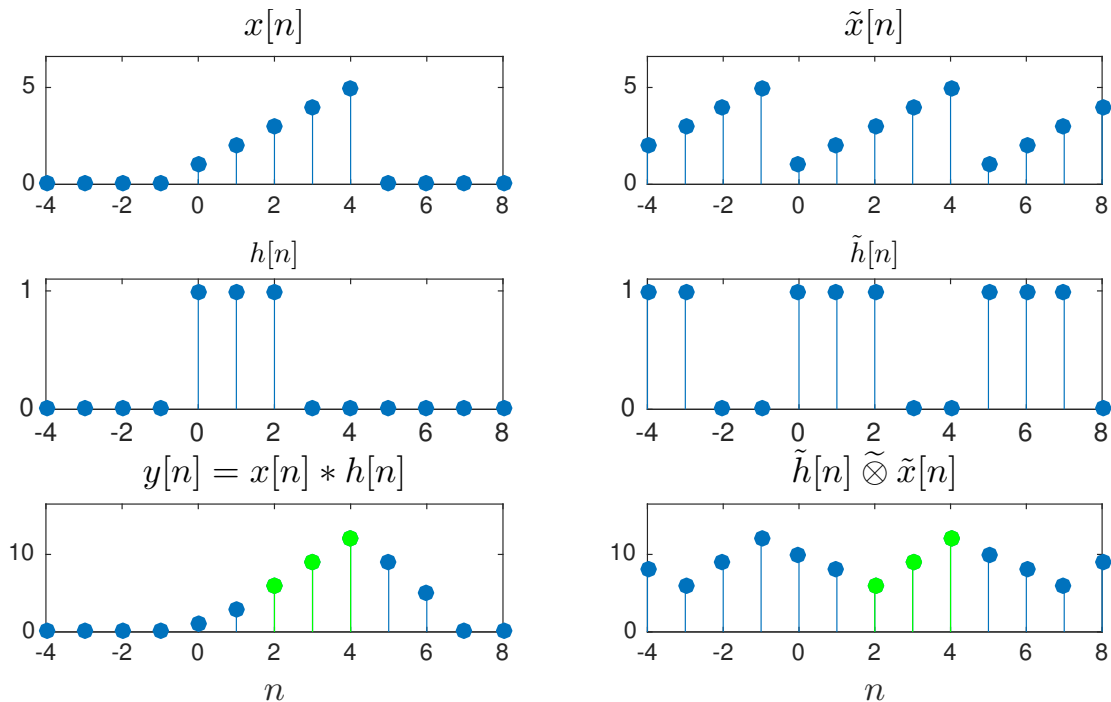
Using duality, one can show that if $\tilde{x}[m, n]$ and $\tilde{y}[m, n]$ are both (M, N) -periodic, then

$$\tilde{x}[m, n] \tilde{y}[m, n] \xleftrightarrow{\text{DFS}} \frac{1}{MN} \tilde{X}[k, l] \otimes_{M, N} \widetilde{\tilde{Y}[k, l]}.$$

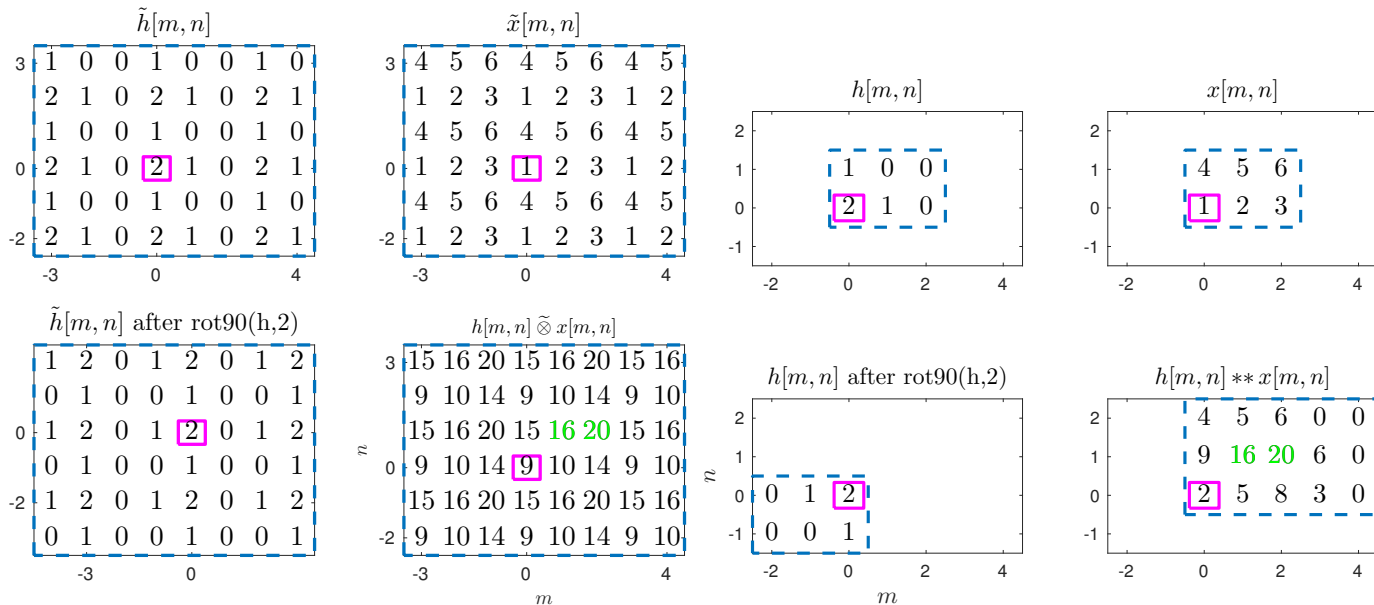
This property is used less often than the convolution property.

Example. Here is a 1D illustration comparing **linear convolution** versus 5-point **periodic convolution** in 1D.

- Linear convolution of a 5-point sequence with a 3-point sequence yields a $5+3-1 = 7$ -point sequence.
- Periodic convolution of two 5-point sequences is still 5-periodic.



Example. 2D illustration comparing **linear convolution** versus (8,6)-point **periodic convolution**. A magenta square denotes the origin. The two results agree only in a few locations in this example.



Application: DSFT property for down-sampling

An earlier HW problem examined down-sampling by two and how that affects the spectrum of a signal. Here we generalize this to down-sampling by arbitrary (integer) factors.

Suppose $f[m, n] \stackrel{\text{DSFT}}{\longleftrightarrow} F(\Omega_1, \Omega_2)$ and we define the down-sampled version $f_{\downarrow L}[m, n] \triangleq f[Lm, Ln]$ for $L \in \mathbb{N}$. How does $F_{\downarrow L}(\Omega_1, \Omega_2)$ relate to $F(\Omega_1, \Omega_2)$? Hint: use the DFS representation of an impulse train given in (7.1).

$$\begin{aligned}
 F_{\downarrow L}(\Omega_1, \Omega_2) &= \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g[m, n] e^{-i(\Omega_1 m + \Omega_2 n)} = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f[Lm, Ln] e^{-i(\Omega_1 m + \Omega_2 n)} \\
 &= \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \delta[m \bmod L] \delta[n \bmod L] f[m, n] e^{-i(\Omega_1 m/L + \Omega_2 n/L)} \\
 &= \frac{1}{L^2} \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \sum_{k=0}^{L-1} \sum_{l=0}^{L-1} e^{i \frac{2\pi}{L} (km + ln)} f[m, n] e^{-i(\Omega_1 m/L + \Omega_2 n/L)} \\
 &= \frac{1}{L^2} \sum_{k=0}^{L-1} \sum_{l=0}^{L-1} \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f[m, n] \exp\left(-i \left(\frac{\Omega_1 - 2\pi k}{L} m + \frac{\Omega_2 - 2\pi l}{L} n \right)\right) \\
 &= \frac{1}{L^2} \sum_{k=0}^{L-1} \sum_{l=0}^{L-1} F\left(\frac{\Omega_1 - 2\pi k}{L}, \frac{\Omega_2 - 2\pi l}{L}\right).
 \end{aligned}$$

In particular, for $L = 2$:

$$f_{\downarrow 2}[m, n] \stackrel{\text{DSFT}}{\longleftrightarrow} \frac{1}{4} \left[F\left(\frac{\Omega_1}{2}, \frac{\Omega_2}{2}\right) + F\left(\frac{\Omega_1}{2} - \pi, \frac{\Omega_2}{2}\right) + F\left(\frac{\Omega_1}{2}, \frac{\Omega_2}{2} - \pi\right) + F\left(\frac{\Omega_1}{2} - \pi, \frac{\Omega_2}{2} - \pi\right) \right]. \quad (7.12)$$

7.3 2D Discrete Fourier transform (DFT)

Now we switch from the “extreme” of 2D periodic sequences $\tilde{x}[m, n]$ defined on \mathbb{Z}^2 to the other extreme of **finite-domain** images $x[m, n]$ (also called **finite-extent** signals), defined on $\mathcal{D}_{M,N} \triangleq \{0, \dots, M-1\} \times \{0, \dots, N-1\}$. The 2D **DFT** is the most appropriate member of the Fourier transform family for such images.

Mathematically speaking, the 2D DFT is also well-defined for **finite-support** signals whose domain includes $\mathcal{D}_{M,N}$ and that are supported on $\mathcal{D}_{M,N}$. But practically speaking, real-world digital images have some finite size $M \times N$ and for computation it is most natural to consider the domain of such signals to be $\mathcal{D}_{M,N}$.

One way to introduce the 2D DFT (but not the only way) is to relate it to the 2D DFS by defining (on paper) a related periodic signal. Given a 2D DS signal $x[m, n]$, we can form periodic signals from $x[m, n]$ in two distinct ways.

- If $x[m, n]$ has domain $\mathcal{D}_{M,N}$ or domain that contains $\mathcal{D}_{M,N}$, then we can define a periodic signal (that has domain \mathbb{Z}^2) called the **(M, N) -point circular extension** of $x[m, n]$ as follows:

$$\tilde{x}[m, n] \triangleq x[m \bmod M, n \bmod N]. \quad (7.13)$$

Notice that this **circular extension** depends only on the values of $x[m, n]$ for $m = 0, \dots, M-1$ and $n = 0, \dots, N-1$, regardless of what the support of $x[m, n]$ is. Clearly the domain of $x[m, n]$ must include $\mathcal{D}_{M,N}$ for this to be well defined.

- If $x[m, n]$ has domain \mathbb{Z}^2 and has finite energy, then another way to construct a periodic signal (that has domain \mathbb{Z}^2) is called the **(M, N) -point periodic superposition** of $x[m, n]$:

$$x_{\text{ps}}[m, n] = x[[m, n]]_{(M,N)} = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} x[m - kM, n - lN]. \quad (7.14)$$

This resulting periodic signal $x_{\text{ps}}[m, n]$ depends on *all* of the values of the original signal $x[m, n]$. Note that the sum in (7.14) is well-defined only for certain signals $x[m, n]$ like those that have finite energy. (The sum diverges if $x[m, n]$ is periodic.)

Many books do not distinguish between the above two ways of creating a periodic signal from $x[m, n]$, and refer to one or the other (or both) as simply the “**periodic extension**” of $x[m, n]$. In general the two methods are distinct. However, there is an important “special case” where the two methods yield identical periodic signals. If $x[m, n]$ has domain \mathbb{Z}^2 yet has **finite-support** on $\mathcal{D}_{M,N}$, then $\tilde{x}[m, n]$ and $x_{\text{ps}}[m, n]$ are identical. So in this “usual” case the generic term “periodic extension” is unambiguous.

For derivations, it is useful to consider when we can “recover” $x[m, n]$ from $\tilde{x}[m, n]$ or $x_{\text{ps}}[m, n]$.

- If $x[m, n]$ has finite domain $\mathcal{D}_{M,N}$, then we can recover it from either $\tilde{x}[m, n]$ or $x_{\text{ps}}[m, n]$ by “extracting” the relevant values:

$$x[m, n] = \begin{cases} \tilde{x}[m, n], & [m, n] \in \mathcal{D}_{M,N} \\ \text{undefined}, & \text{otherwise.} \end{cases} \quad (7.15)$$

In this case the same relationship holds for $x_{\text{ps}}[m, n]$.

- If $x[m, n]$ has domain \mathbb{Z}^2 but has finite support on $\mathcal{D}_{M,N}$, then we can recover it from either $\tilde{x}[m, n]$ or $x_{\text{ps}}[m, n]$ by applying rectangular window in the space domain:

$$x[m, n] = \tilde{x}[m, n] R_{MN}[m, n] = \begin{cases} \tilde{x}[m, n], & [m, n] \in \mathcal{D}_{M,N} \\ 0, & \text{otherwise,} \end{cases} \quad \text{where } R_{MN}[m, n] \triangleq \begin{cases} 1, & [m, n] \in \mathcal{D}_{M,N} \\ 0, & \text{otherwise.} \end{cases} \quad (7.16)$$

Likewise, $x[m, n] = x_{\text{ps}}[m, n] R_{MN}[m, n]$ in this case.

- If the support of $x[m, n]$ is larger than $\mathcal{D}_{M,N}$, then in general it is impossible to recover $x[m, n]$ from $x_{\text{ps}}[m, n]$. However, we can recover the part of $x[m, n]$ defined over $\mathcal{D}_{M,N}$ from $\tilde{x}[m, n]$ using (7.15).

The distinction between (7.15) and (7.16) is largely a matter of convention, *i.e.*, whether one treats $x[m, n]$ as 0 or as undefined (or as periodic) outside of $\mathcal{D}_{M,N}$, and books have different conventions. Here we generally take the “undefined” convention because that choice best matches practical computing methods where one has a finite sized array of signal values.

2D DFT definition

Focusing now on the case of signals $x[m, n]$ having finite-domain $\mathcal{D}_{M,N}$, what is the DFS of the periodic extension signal $\tilde{x}[m, n]$? Applying the analysis formula (7.8) to $\tilde{x}[m, n]$ yields:

$$\tilde{X}[k, l] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \tilde{x}[m, n] e^{-i2\pi(\frac{k}{M}m + \frac{l}{N}n)} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m, n] e^{-i2\pi(\frac{k}{M}m + \frac{l}{N}n)}.$$

(The equality follows from the limits on the sum.) The DFS coefficients are *defined and generally nonzero* for all $k, l \in \mathbb{Z}$.

We truncate this periodic set of coefficients to define a finite-domain 2D function, called the ***MN*-point discrete Fourier transform (DFT)** of $x[m, n]$:

$$X[k, l] \triangleq \begin{cases} \tilde{X}[k, l], & (k, l) \in \mathcal{D}_{M,N} \\ \text{undefined}, & \text{otherwise} \end{cases} \\ = \begin{cases} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m, n] e^{-i2\pi(\frac{k}{M}m + \frac{l}{N}n)}, & (k, l) \in \mathcal{D}_{M,N} \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

- For practical purposes, it is safest to consider $X[k, l]$ to be defined only for $(k, l) \in \mathcal{D}_{M,N}$.
- Alternatively, some books consider $X[k, l]$ to be *defined* for all $k, l \in \mathbb{Z}$, but to be *nonzero* only for $(k, l) \in \mathcal{D}_{M,N}$.
- Alternatively, some books choose to define $X[k, l]$ to be a periodic function of k and l , just like $\tilde{X}[k, l]$ is.

The disadvantage of these latter two conventions is that tools for computing DFTs work only with finite arrays. Trying to evaluate $X[-k, l]$ will cause a program fault usually. To avoid such problems, it is safest to have the theory match practice as closely as possible, so we choose $X[k, l]$ to be *undefined* except for $(k, l) \in \mathcal{D}_{M,N}$. In other words, the **domain** of $X[k, l]$ is simply $\mathcal{D}_{M,N}$.

Note the somewhat similar notation for the DFT $X[k, l]$ and the DSFT $X(\Omega_1, \Omega_2)$, because both use a capital X . Which transform is meant should always be obvious from context.

Of course we can always “recover” the entire periodic sequence $\tilde{X}[k, l]$ (defined on \mathbb{Z}^2) from the DFT $X[k, l]$ by applying a (M, N) -point circular extension as follows:

$$\tilde{X}[k, l] = X[k \bmod M, l \bmod N].$$

Inverse 2D DFT definition

Because the inverse DFS formula (7.9) depends on $\tilde{X}[k, l]$ only for $(k, l) \in \mathcal{D}_{M,N}$, clearly we can recover $x[m, n]$ from $X[k, l]$ using either (7.15) and (7.16) depending on the assumed domain of $x[m, n]$. For practical purposes we focus on the finite-domain case (7.15), so

$$x[m, n] = \begin{cases} \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} X[k, l] e^{i2\pi(\frac{k}{M}m + \frac{l}{N}n)}, & [m, n] \in \mathcal{D}_{M,N} \\ \text{undefined,} & \text{otherwise.} \end{cases}$$

This is the **inverse (M, N) -point DFT (iDFT)** formula in 2D, also known as the **synthesis** formula.

In summary then, for a 2D signal $x[m, n]$ having domain $\mathcal{D}_{M,N}$, the (M, N) -point DFT/iDFT pair is given as follows.

$$\text{analysis: } X[k, l] = \begin{cases} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m, n] e^{-i2\pi(\frac{k}{M}m + \frac{l}{N}n)}, & (k, l) \in \mathcal{D}_{M,N} \\ \text{undefined,} & \text{otherwise,} \end{cases} \quad (7.17)$$

$$\text{synthesis: } x[m, n] = \begin{cases} \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} X[k, l] e^{i2\pi(\frac{k}{M}m + \frac{l}{N}n)}, & [m, n] \in \mathcal{D}_{M,N} \\ \text{undefined,} & \text{otherwise,} \end{cases} \quad (7.18)$$

Some books use zero or the circular extension, but in these notes we will generally treat $x[m, n]$ and $X[k, l]$ as *undefined* outside $\mathcal{D}_{M,N}$ because that interpretation best matches computer software where both $x[m, n]$ and $X[k, l]$ are stored as $M \times N$ arrays. In other words, typically we consider the **domain** of both $x[m, n]$ and $X[k, l]$ to be $\mathcal{D}_{M,N}$.

If signal $x[m, n]$ is unitless, what are the units of its DFT $X[k, l]$? ?? [RQ]

If $M = 3$ and $N = 2$ and $x[m, n] = \delta_2[m, n]$, what are the values of $X[1, 0]$ and $X[2, 1]$? ?? [RQ]

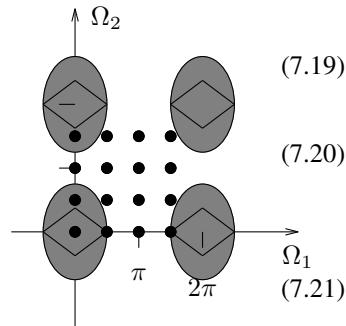
Relationship between DFT and DSFT (important!)

For a **finite-support** signal $x[m, n]$ with domain \mathbb{Z}^2 supported on $\mathcal{D}_{M,N}$, we can relate the DFT to the DSFT as follows:

$$X[k, l] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m, n] e^{-i2\pi(\frac{k}{M}m + \frac{l}{N}n)} \quad (7.19)$$

$$= \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x[m, n] e^{-i2\pi(\frac{k}{M}m + \frac{l}{N}n)} \quad (\text{because of finite support}) \quad (7.20)$$

$$= X(\Omega_1, \Omega_2) \Big|_{\Omega_1=2\pi k/M, \Omega_2=2\pi l/N} = X\left(\frac{2\pi}{M}k, \frac{2\pi}{N}l\right), \quad \begin{array}{l} k = 0, \dots, M-1 \\ l = 0, \dots, N-1. \end{array} \quad (7.21)$$



When $x[m, n]$ has finite support $\mathcal{D}_{M,N}$, we can also write the DFS coefficients of $\tilde{x}[m, n]$ in terms of samples of the DSFT:

$$\tilde{X}[k, l] = X(\Omega_1, \Omega_2) \Big|_{\Omega_1=2\pi k/M, \Omega_2=2\pi l/N} = X\left(\frac{2\pi}{M}k, \frac{2\pi}{N}l\right). \quad (7.22)$$

Relating DFT and DSFT and FT?

Caution: recall from (5.17) that if $g_d[m, n] = g_a(m\Delta_x, n\Delta_y)$, then

$$G_d(\Omega_1, \Omega_2) = \frac{1}{\Delta_x \Delta_y} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} G_a\left(\frac{\Omega_1/2\pi - k}{\Delta_x}, \frac{\Omega_2/2\pi - l}{\Delta_y}\right).$$

This relationship holds for any sampling rate regardless of whether $g_a(x, y)$ is band-limited. But if $g_a(x, y)$ were band-limited, then the central replicate of $G_d(\Omega_1, \Omega_2)$ would tell us everything about the original analog signal spectrum. And if (7.21) also held, we could relate the analog spectrum to the DFT coefficients. But (7.21) requires a finite-support signal, which is (in general) incompatible with the analog signal being band-limited.

What simple band-limited analog signal, when critically sampled, is a finite-support DS signal? ??

(do in class)

What happens though if you shift that analog signal slightly? ??

(do in class)

Properties of the DFT

The following properties are all written so that only the values of $x[m, n]$ for $m = 0, \dots, M-1$ and $n = 0, \dots, N-1$ are involved, as well as only the values of $X[k, l]$ for $k = 0, \dots, M-1$, $l = 0, \dots, N-1$.

Missing properties (relative to CS FS): scaling, because scaling changes the extent. (Presumably there is such a property, but it probably differs significantly from the usual scaling properties.)

Because the (M, N) -point DFT is defined in terms of the (M, N) -point DFS, it inherits many of its properties. However, there are some differences too, due to the finite-extent nature of both $x[m, n]$ and $X[k, l]$.

Properties that are identical to those of the DFS

- **linearity** if $x[m, n]$ and $y[m, n]$ have *domain* containing $\mathcal{D}_{M,N}$, then:

$$\alpha x[m, n] + \beta y[m, n] \xleftrightarrow{\text{DFT}} \alpha X[k, l] + \beta Y[k, l]$$

- **separability** (new in 2D!)

(prove in class)

$$x[m, n] = x_1[m] x_2[n] \xleftrightarrow{\text{DFT}} X_1[k] X_2[l],$$

where $X_1[k]$ denotes the usual M -point 1D DFT of $x_1[m]$ and $X_2[l]$ denotes the N -point 1D DFT of $x_2[n]$.

- **average value** (Caution: $X[0, 0]$ is often called the DC value, but it is the sum, not the average value!)

$$\frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m, n] = \frac{1}{MN} X[0, 0]$$

- **Parseval's theorem / Rayleigh's theorem:**

$$\sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m, n] y^*[m, n] = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} X[k, l] Y^*[k, l]$$

$$\sum_{m=0}^{M-1} \sum_{n=0}^{N-1} |x[m, n]|^2 = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} |X[k, l]|^2$$

Properties that differ somewhat from those of the DFS

Many of these properties depend on $x[m \bmod M, n \bmod N]$, the (M, N) -point circular extension of a signal, as defined in (7.13), or the (M, N) -point circular extension of the DFT coefficients $X[k, l]$.

- **complex conjugate**

$$x^*[m, n] \xleftrightarrow{\text{DFT}} X^*[-k \bmod M, -l \bmod N]$$

- **duality**

$$x[m, n] \xleftrightarrow{\text{DFT}} X[k, l] \implies \begin{array}{l} X[m, n] \xleftrightarrow{\text{DFT}} MN x[-k \bmod M, -l \bmod N] \\ X^*[m, n] \xleftrightarrow{\text{DFT}} MN x^*[k, l] \end{array}$$

- **circular shift / complex modulation**

A circular shift of a 1D signal by n_0 changes $(x[0], \dots, x[N-1])$ into $(x[n_0], \dots, x[N-1], x[0], \dots, x[n_0-1])$. We write this circular shift as $x[(n-n_0) \bmod N]$.

For a 2D signal, the circular shift by $(m_0, n_0) \in \mathbb{Z}^2$ and its (M, N) -point DFT are related by

$$x[(m-m_0) \bmod M, (n-n_0) \bmod N] \xleftrightarrow{\text{DFT}} e^{-i2\pi(km_0/M+ln_0/N)} X[k, l].$$

- **circular frequency shift** (modulation)

$$e^{j2\pi(k_0 m/M + l_0 n/N)} x[m, n] \xleftrightarrow{\text{DFT}} X[(k - k_0) \bmod M, (l - l_0) \bmod N], \quad k_0, l_0 \in \mathbb{Z}$$

- **Circular space reversal**

For the DSFT, we know that if $x[m, n] \xleftrightarrow{\text{DSFT}} X(\Omega_1, \Omega_2)$, then $x[-m, -n] \xleftrightarrow{\text{DSFT}} X(-\Omega_1, -\Omega_2)$. Likewise, the corresponding DFS property (7.10) is fairly straightforward.

What is the corresponding formula for the DFT?

First define: the **circular space reversal** of a 2D signal $x[m, n]$ is given by $x[-m \bmod M, -n \bmod N]$.

Example. If $x[m, n] = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} \begin{matrix} n \\ \uparrow \\ m \end{matrix}$, then $y[m, n] = x[-m \bmod 4, -n \bmod 3] = \begin{bmatrix} 5 & 8 & 7 & 6 \\ 1 & 4 & 3 & 2 \\ 9 & 12 & 11 & 10 \end{bmatrix}$.

For example, $y[3, 1] = x[-3 \bmod 4, -1 \bmod 3] = x[1, 2] = 2$.

The corresponding DFT property is:

$$x[-m \bmod M, -n \bmod N] \xleftrightarrow{\text{DFT}} X[-k \bmod M, -l \bmod N]$$

- **Symmetry properties**

$$x^*[m, n] \xleftrightarrow{\text{DFT}} X^*[-k \bmod M, -l \bmod N]$$

If $x[m, n]$ is **circularly even**, i.e., if $x[-m \bmod M, -n \bmod N] = x[m, n]$, then $X[k, l]$ is also **circularly even**.

If $x[m, n]$ is real, then $X[k, l]$ is Hermitian **circularly symmetric**, i.e.,

$$X[-k \bmod M, -l \bmod N] = X^*[k, l].$$

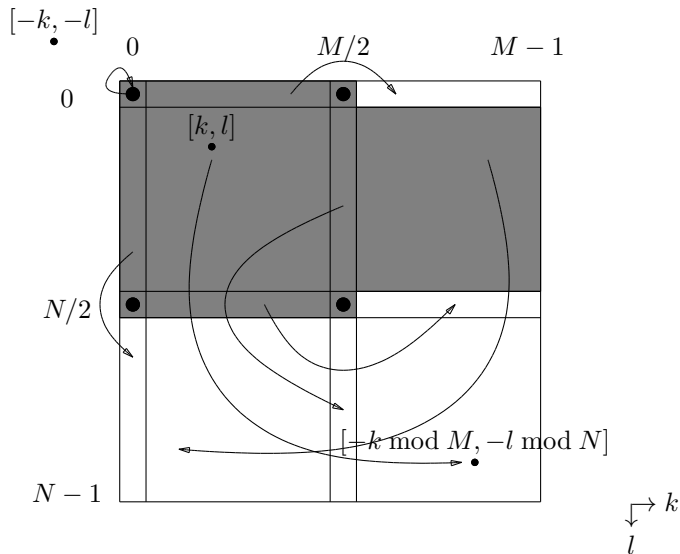
Likewise, $X[k, l]$ is real if and only if $x[m, n]$ has **circular Hermitian symmetry**:

$$x[-m \bmod M, -n \bmod N] = x^*[m, n], \quad m = 0, \dots, M-1, \quad n = 0, \dots, N-1.$$

Hermitian circular symmetry illustrated

If $x[m, n]$ is real, the Hermitian circular symmetry formula $X[-k \bmod M, -l \bmod N] = X^*[k, l]$ may not convey visually the symmetry properties of the DFT coefficients. If we arrange the DFT coefficients as a $M \times N$ array, the following diagram illustrates which coefficients are complex conjugates of which other coefficients. The shaded area represents a set of “sufficient” coefficients; when storage is at a premium, for real signals it suffices to store the DFT coefficients in the shaded region.

The points indicated by the dots are values that are their own complex conjugates, and hence must be real. (This figure assumes that M and N are *even* natural numbers.)



Proof of complex conjugate property _____ **(skim)**

Here is a proof of the complex conjugate property for 1D signals, to provide an example of how one performs such proofs.

By the synthesis property:

$$x[n] = \frac{1}{N} \sum_{k=0}^{M-1} X[k] e^{i2\pi nk/N},$$

so taking the complex conjugate of both sides yields

$$\begin{aligned} x^*[n] &= \frac{1}{N} \sum_{k=0}^{M-1} X^*[k] e^{-i2\pi nk/N} \\ &= \frac{1}{N} X^*[0] + \frac{1}{N} \sum_{k=1}^{N-1} X^*[k] e^{-i2\pi nk/N} && \text{(treating } k=0 \text{ separately because } N-0=N) \\ &= \frac{1}{N} X^*[0] + \frac{1}{N} \sum_{l=1}^{N-1} X^*[N-l] e^{-i2\pi n(N-l)/N} && \text{(for } l=N-k) \\ &= \frac{1}{N} X^*[0 \bmod N] + \frac{1}{N} \sum_{l=1}^{N-1} X^*[N-l] e^{i2\pi nl/N} && \text{(because } e^{-i2\pi n} = 1) \\ &= \frac{1}{N} X^*[0 \bmod N] + \frac{1}{N} \sum_{l=1}^{N-1} X^*[-l \bmod N] e^{i2\pi nl/N} \\ &= \frac{1}{N} \sum_{l=0}^{N-1} X^*[-l \bmod N] e^{i2\pi nl/N}. \end{aligned}$$

Because this is the DFT synthesis expression for $x^*[n]$, we have shown the following property:

$$x^*[n] \stackrel{\text{DFT}}{\longleftrightarrow} X^*[-k \bmod N].$$

Convolution property

One of the main uses of the DFT is to implement fast convolution via the FFT. So the convolution property is particularly important. In fact, this property is probably the primary reason why the DFT is used so much more frequently than the many alternative orthogonal transforms.

What happens if we take two sets of (M, N) -point DFT coefficients, multiply them, and take the inverse DFT? Clearly the answer cannot in general match **linear convolution**, because linear convolution of two signals results in a signal with larger support.

Derivation:

Suppose $h[m, n]$ and $x[m, n]$ are both signals with domain containing $\mathcal{D}_{M,N}$, with corresponding (M, N) -point DFT coefficients $H[k, l]$ and $X[k, l]$.

Define $Y[k, l] = H[k, l] X[k, l]$, and let $y[m, n]$ be the inverse (M, N) -point DFT of $Y[k, l]$. Considering the DFT's construction:

$$y[m, n] = \begin{cases} \tilde{y}[m, n], & [m, n] \in \mathcal{D}_{M,N} \\ \text{undefined}, & \text{otherwise,} \end{cases}$$

where

$$\tilde{y}[m, n] \stackrel{\text{DFS}}{\longleftrightarrow} \tilde{Y}[k, l] = H[k, l] \widetilde{X[k, l]} = \tilde{H}[k, l] \tilde{X}[k, l],$$

where $\tilde{H}[k, l]$ and $\tilde{X}[k, l]$ are the periodic extensions of $H[k, l]$ and $X[k, l]$ respectively.

It follows that $\tilde{y}[m, n] = \tilde{h}[m, n] \widetilde{\otimes}_{M,N} \tilde{x}[m, n]$, and so

$$\begin{aligned} y[m, n] &= \begin{cases} \tilde{h}[m, n] \widetilde{\otimes}_{M,N} \tilde{x}[m, n], & [m, n] \in \mathcal{D}_{M,N} \\ \text{undefined}, & \text{otherwise} \end{cases} \\ &= \begin{cases} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} \tilde{h}[k, l] \tilde{x}[m-k, n-l], & [m, n] \in \mathcal{D}_{M,N} \\ \text{undefined}, & \text{otherwise.} \end{cases} \end{aligned}$$

This is a correct but incomplete answer because we want a direct relationship between $y[m, n]$ and the original signals $h[m, n]$ and $x[m, n]$. Considering that the summation limits in (7.11) extend only from 0 to $N - 1$ and 0 to $M - 1$, we can write

$$y[m, n] = \begin{cases} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} h[k, l] x[(m - k) \bmod M, (n - l) \bmod N], & [m, n] \in \mathcal{D}_{M,N} \\ \text{undefined}, & \text{otherwise} \end{cases}$$

$$\triangleq x[m, n] \otimes_{M,N} h[m, n]. \quad (7.23)$$

Note that the above definition of **circular convolution** of two finite-support signals is *slightly* different than the definition of **periodic convolution** of two periodic signals. In words, if we start with two finite-support signals, multiplying their DFT coefficients is equivalent to: first forming their periodic extension, then performing periodic convolution, then truncating the result.

In summary, one usually expresses the **convolution property** of the DFT as follows:

$$h[m, n] \otimes_{M,N} x[m, n] \xleftrightarrow{\text{DFT}} H[k, l] X[k, l],$$

where one must remember the truncation and the slightly modified definition in (7.23) of **circular convolution**, denoted $\otimes_{M,N}$ or often simply \otimes .

Multiplication property

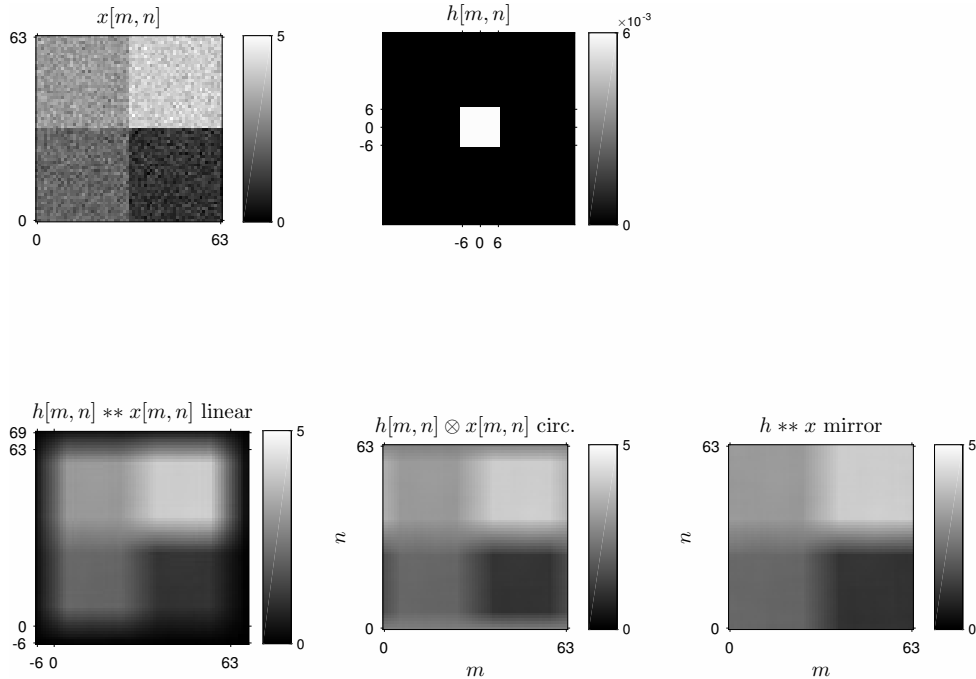
Using duality, one can show that if $X[k, l]$ is the (M, N) -point DFT of $x[m, n]$ and $Y[k, l]$ is the (M, N) -point DFT of $y[m, n]$, then the (M, N) -point DFT of their product is given by:

$$x[m, n] y[m, n] \xleftrightarrow{\text{DFT}} \frac{1}{MN} X[k, l] \otimes_{M,N} Y[k, l].$$

The convolution and multiplication properties of the DFT hold *regardless* of whether $x[m, n]$ and $y[m, n]$ are finite support signals! But of course only the values of $x[m, n]$ and $y[m, n]$ over the set $\mathcal{D}_{M,N}$ affect the circular convolution results.

Illustration of wrap around effect

Periodic convolution can cause signals from the borders of an image to propagate influence to the “opposite” borders as illustrated here. Generally this is an *undesirable* property and we will want to overcome it as described next.



Emulating linear convolution

The effect of circular convolution is a **wrap around effect**, where signal values from one border of an image can “wrap around” to affect values on the other side after filtering via a DFT. This effect is generally undesirable.

How can we eliminate this wrap around effect of the DFT? **Using zero padding.**

Convolving an $N_1 \times N_2$ image $x[m, n]$ with a $M_1 \times M_2$ filter $h[m, n]$ yields a $L_1 \times L_2$ result $y[m, n]$, where $L_k = N_k + M_k - 1$. Thus, if we **zero pad** both $x[m, n]$ and $h[m, n]$ to a size $L_1 \times L_2$ before taking the DFT, then the final result of

$$\text{iDFT}(\text{DFT}(x) \cdot \star \text{DFT}(h))$$

will be exactly equivalent to the result of **linear convolution** of $x[m, n]$ with $h[m, n]$. (Of course this only works for finite-support images, but that is always adequate in practice for finite-support filters!)

Where do we put the zeros when we zero pad? It can depend on the situation, and on how one chooses the coordinate system.

A common situation is when we have an even-sized image $x[m, n]$ defined on $\mathcal{D}_{M,N}$ that we wish to convolve with a odd-sized FIR filter $h[m, n]$, such as a moving-average filter. The following function uses zero padding and the 2D FFT to emulate linear convolution for such cases.

The simple MATLAB function `psf2otf` or the equivalent function in JULIA in the `ImageProcessing` package is convenient here.

```

function y = fft2_pad(x, h)
% convolve filter h and image x using zero-padded fft
[N1, N2] = size(x); % assume x is even sized but too lazy to check
[M1, M2] = size(h); % assume h is odd sized but too lazy to check
L1 = N1 + M1 - 1;
L2 = N2 + M2 - 1;
if 0 % hard way
    hpad = zeros(L1,L2);
    hpad(L1/2+1+[-(M1-1)/2:(M1-1)/2], L2/2+1+[-(M2-1)/2:(M2-1)/2]) = h;
    hpad = ifftshift(hpad); % needed put "0" of filter h in proper spot
    y = ifft2(fft2(x, L1, L2) .* fft2(hpad));
else % easy way
    y = ifft2(fft2(x, L1, L2) .* psf2otf(h, [L1 L2]));
end
y = y(1:N1,1:N2); % extract the part we wanted in the first place

```

Exercise. Generalize this function to handle input images $x[m, n]$ that have either even or odd dimensions.

Practical implementation

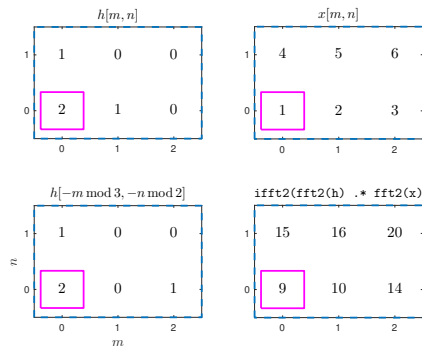
There is a practical inconvenience with zero padding however. Suppose we wish to convolve a 256×256 image with a 17×17 filter. The result will be 272×272 . The prime factors of 272 are 2 and 17. The FFT works fastest for small prime factors (especially 2), so a prime factor of 17 is undesirable computationally. One could pad to a 512×512 image, but then only 28% of the final image would be the part we care about (the rest would be zero in exact arithmetic). There are a few practical choices.

- Abandon the zero padding, and tolerate or disregard the errors due to wrap around near the image edges.
 - Zero pad to a value smaller than 512; *e.g.*, 288 has factors 2 and 3 and is only slightly bigger than 272. MATLAB's `fft2` routine, built on the FFTW library, is clever enough to exploit these small factors. Consider the following results of `tic, fft2(rand(N)); toc` on a Pentium II (200MHz).
- | | N | time [s] |
|---|-----|----------|
| | 256 | 0.32 |
| | 272 | 0.45 |
| | 288 | 0.37 |
| This approach <i>still uses circular convolution</i> , but the zero padding has the effect of yielding the same <i>results</i> as linear convolution, <i>for finite-support signals</i> . | 512 | 1.25 |

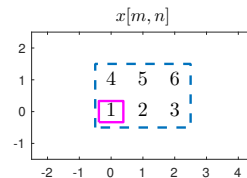
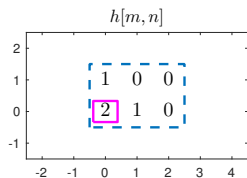
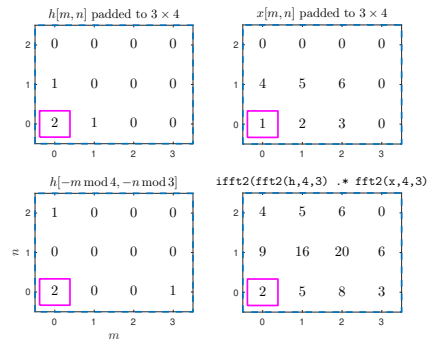
Clearly intelligent zero padding can reduce computation considerably over the simple “next higher power of 2” approach.

- Use the **overlap-add method** or **overlap-save method** described below.

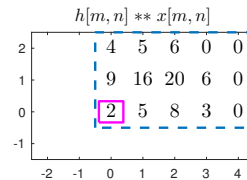
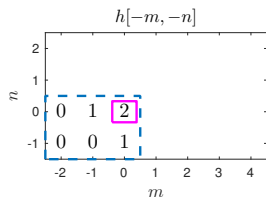
Example. Circular convolution using FFT:



Circular convolution with zero padding:



Linear convolution:



The overlap-add method**(skip)**

Because convolution is linear, we can decompose convolution of a large image with a small filter into the sum of the convolution of the filter with each of several modest-sized blocks of the image. This is called the **overlap-add method**. Let

$$x[m, n] = \sum_k x_k[m, n], \quad \text{Example: } \begin{array}{|c|c|c|} \hline x_1[m, n] & x_2[m, n] & x_3[m, n] \\ \hline x_4[m, n] & x_5[m, n] & x_6[m, n] \\ \hline \end{array}$$

then by linearity of convolution:

$$\begin{aligned} y[m, n] &= h[m, n] ** x[m, n] = h[m, n] ** \sum_k x_k[m, n] \\ &= \sum_k h[m, n] ** x_k[m, n] \triangleq \sum_k y_k[m, n], \quad \text{where } y_k[m, n] \triangleq h[m, n] ** x_k[m, n]. \end{aligned}$$

If block is $M_1 \times M_2$ and the filter is $L_1 \times L_2$, then the result of each block convolution will be $(M_1 + L_1 - 1) \times (M_2 + L_2 - 1)$. So the output blocks $y_k[m, n]$ will overlap. Nevertheless, superposition still applies, so we will get the correct final result provided:

- we implement linear convolution for each block (using zero padding if needed),
- and we properly *add* the resulting overlapping blocks.

The MATLAB command `fftfilt` does this in 1D.

Is there a 2D overlap-add method in MATLAB? (The `conv2` routine is a MEX file.)

There is a related **overlap-save method** [1].

To implement linear convolution for each block, we can either use space-domain convolution (which would be reasonable for small filters, especially if separable), or use zero-padded convolution via the FFT. One should choose the block size such that the zero-padded FFT size factors into small primes.

When $h[m, n]$ has much smaller support than $x[m, n]$, this approach avoids having to take a large FFT of a version of $h[m, n]$ with a lot of zero padding.

Sampling the DSFT and “aliasing”**(skim)**

We have seen that if $x[m, n]$ has domain \mathbb{Z}^2 but has finite-support $\mathcal{D}_{M,N}$, then its DFT consists of samples of its DSFT:

$$X[k, l] = X(\Omega_1, \Omega_2) \Big|_{\Omega_1 = \frac{2\pi}{M}k, \Omega_2 = \frac{2\pi}{N}l}, \quad \text{if } x[m, n] \text{ has support } \mathcal{D}_{M,N}. \quad (7.24)$$

Often, however, we “take samples” of an analytical expression for the DSFT of some unknown signal, and then compute the iDFT to examine the signal. What happens if the underlying signal does not have finite support?

Suppose $x[m, n] \xleftrightarrow{\text{DSFT}} X(\Omega_1, \Omega_2)$ for some DS signal $x[m, n]$, and we choose some integers (M, N) and define

$$Y[k, l] \triangleq X(\Omega_1, \Omega_2) \Big|_{\Omega_1 = \frac{2\pi}{M}k, \Omega_2 = \frac{2\pi}{N}l}, \quad k = 0, \dots, M-1, l = 0, \dots, N-1.$$

If we compute the (M, N) -point iDFT of $Y[k, l]$, how does the resulting signal $y[m, n]$ relate to the original $x[m, n]$? Graphically:

$$x[m, n] \rightarrow \underbrace{\boxed{\text{DSFT}} \rightarrow X(\Omega_1, \Omega_2)}_{\text{typically on paper}} \rightarrow \underbrace{\boxed{\text{sample/truncate}} \rightarrow Y[k, l] \rightarrow \boxed{\text{iDFT}}}_{\text{typically in computer}} \rightarrow y[m, n]$$

By construction, and by decomposing sums over \mathbb{Z}^2 into blocks of size $M \times N$, we have:

$$\begin{aligned}
 Y[k, l] &= X\left(\frac{2\pi}{M}k, \frac{2\pi}{N}l\right) \\
 &= \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x[m, n] e^{-i2\pi\left(\frac{k}{M}m + \frac{l}{N}n\right)} \\
 &= \sum_{k'=-\infty}^{\infty} \sum_{l'=-\infty}^{\infty} \sum_{m'=0}^{M-1} \sum_{n'=0}^{N-1} x[m' + k'M, n' + l'N] e^{-i2\pi(k[m'+k'M]/M + l[n'+l'N]/N)} \quad \left(\begin{array}{l} m = m' + k'M \\ n = n' + l'N \end{array} \right) \\
 &= \sum_{m'=0}^{M-1} \sum_{n'=0}^{N-1} \left[\sum_{k'=-\infty}^{\infty} \sum_{l'=-\infty}^{\infty} x[m' + k'M, n' + l'N] \right] e^{-i2\pi(km'/M + ln'/N)}.
 \end{aligned}$$

This is simply a (M, N) -point DFT of the inner bracketed expression. Thus, the iDFT of $Y[k, l]$ yields a signal $y[m, n]$ given by that inner expression (with $k = -k'$ and $l = -l'$):

$$y[m, n] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} x[m - kM, n - lN] = x[m, n] ** \delta_2[[m, n]]_{(M, N)} = x_{\text{ps}}[m, n],$$

for $[m, n] \in \mathcal{D}_{M, N}$, where

$$\delta_2[[m, n]]_{(M, N)} = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \delta_2[m - kM, n - lN] = \delta_2[m \bmod M, n \bmod N].$$

In other words, sampling in the Fourier domain causes superimposed replicas in the space domain, which means spatial **aliasing** if $x[m, n]$ is not a finite-extent signal.

(skim)

Example. Consider the 1D DS signal having spectrum $X(\Omega) = \frac{1}{1 - a e^{-i\Omega}}$ for $|a| < 1$.

Taking N samples of this yields $Y[k] = X\left(\frac{2\pi}{N}k\right) = \frac{1}{1 - a e^{-i2\pi k/N}}$.

Taking the inverse N -point DFT yields, for $n = 0, \dots, N-1$:

$$\begin{aligned} y[n] &= \frac{1}{N} \sum_{k=0}^{M-1} Y[k] e^{i2\pi kn/N} = \frac{1}{N} \sum_{k=0}^{M-1} \frac{1}{1 - a e^{-i2\pi k/N}} e^{i2\pi kn/N} = \frac{1}{N} \sum_{k=0}^{M-1} \sum_{m=0}^{\infty} \left(a e^{-i2\pi k/N}\right)^m e^{i2\pi kn/N} \\ &= \sum_{m=0}^{\infty} a^m \left(\frac{1}{N} \sum_{k=0}^{M-1} e^{-i2\pi mk/N} e^{i2\pi kn/N} \right) = \sum_{m=0}^{\infty} a^m \delta[(n - m) \bmod N] = \sum_{l=0}^{\infty} a^{n+lN} = a^n \frac{1}{1 - a^N}. \end{aligned}$$

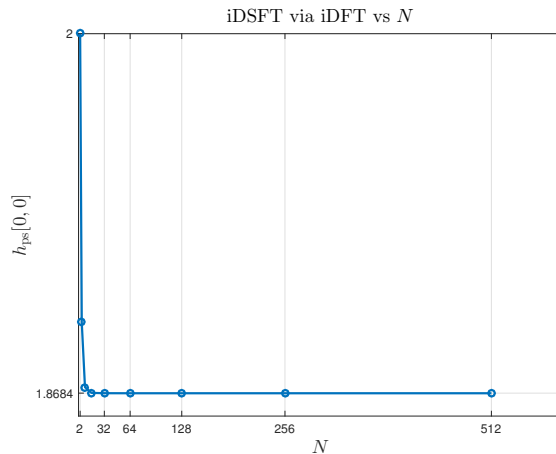
The inverse DSFT is $x[n] = a^n u[n]$, which is not space limited, and the extra term $\left(\frac{1}{1 - a^N}\right)$ is an error due to spatial aliasing. As we increase the number of samples N , the error decreases.

This is a rare example where we can solve analytically. Usually we use the iDFT numerically, as in the next example.

Example. For filter design 3 at the end of Ch. 5, we wanted to find the impulse response of the filter with frequency response

$$H_3(\Omega_1, \Omega_2) = 4(1 + \cos \Omega_o) \operatorname{rect}\left(\frac{\Omega_o}{2\pi}\right), \quad \Omega_o \triangleq \sqrt{(\operatorname{mod}(\Omega_1 + \pi, 2\pi) - \pi)^2 + (\operatorname{mod}(\Omega_2 + \pi, 2\pi) - \pi)^2}.$$

I did this numerically by sampling the DSFT, and then computing the iDFT. Because $h_3[m, n]$ in this example is IIR, sampling the DSFT and using an iDFT results in a spatially aliased reconstruction of $h_3[m, n]$, denoted $h_{\text{ps}}[m, n]$, defined as in (7.14). That example used $M = N = 128$. Let us see if that was a reasonable choice by computing iDFT for various values of N and seeing how $h_{\text{ps}}[0, 0]$ varies with N .

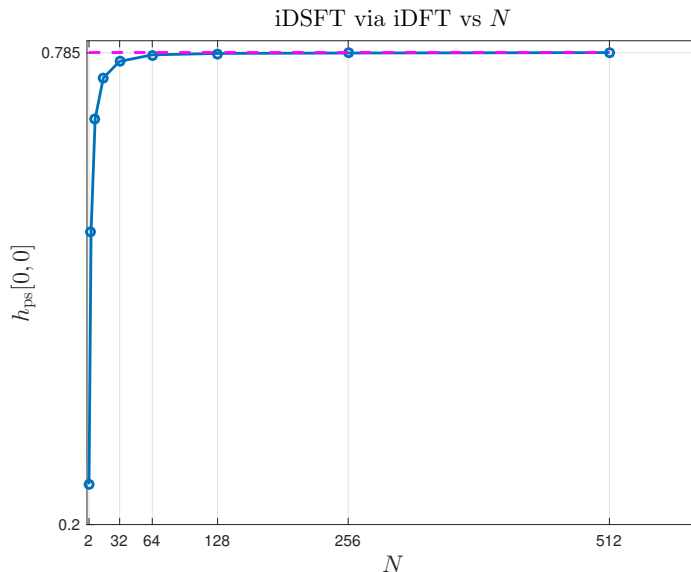


In this case, $h_{\text{ps}}[m, n]$ converges rapidly, so even $N = 32$ appears to suffice for $h_3[0, 0]$. But we would need to check other values of $h_3[m, n]$ to be more confident that $N = 32$ is adequate.

Example. As a more challenging example, consider the ideal lowpass filter spectrum:

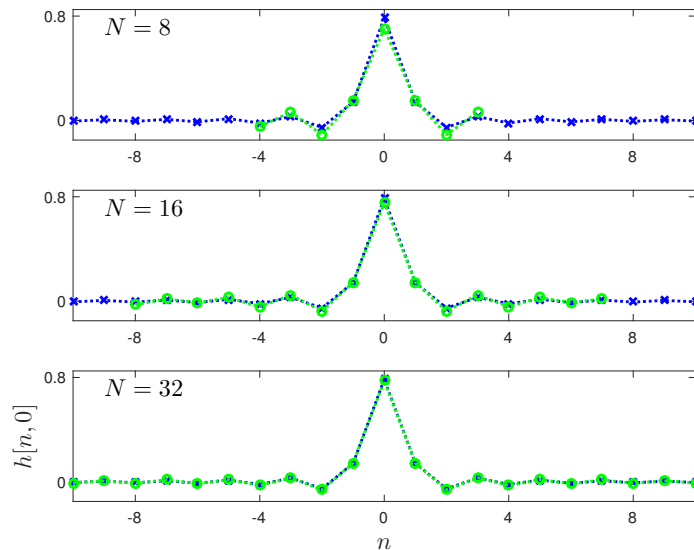
$$H(\Omega_1, \Omega_2) = \text{rect}\left(\frac{\Omega_o}{2\pi}\right), \quad \Omega_o \triangleq \sqrt{(\text{mod}(\Omega_1 + \pi, 2\pi) - \pi)^2 + (\text{mod}(\Omega_2 + \pi, 2\pi) - \pi)^2}.$$

Here is $h_{\text{ps}}[0, 0]$ versus N . Ideally $h[0, 0] = \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} H(\Omega_1, \Omega_2) d\Omega_1 d\Omega_2 = \frac{1}{(2\pi)^2} \pi \pi^2 = \pi/4 \approx 0.785$.



In this case $h[m, n]$ is a jinc function, which decays more slowly (has larger sidelobes), so larger N values are needed to achieve negligible effects of the spatial aliasing.

The following figure shows profiles through the exact (IIR) impulse response $h[m, n]$ (blue), determined analytically, and the approximate impulse response $h_{\text{ps}}[m, n]$ (green), determined by sampling the DSFT and using an inverse DFT, for various values of N . As N increases the approximation improves.



To sanity check the above figure, determine analytically $h[m, n]$ and find the numerical value for $h[1, 0]$.

Hint. Refer to Ch. 6. ??

(do in class)

Matrix representations of DFT in 1D and 2D

Matrix representation of 1D DFT

The DFT is a linear transformation of the sequence $x[0], \dots, x[N-1]$ to the coefficients $X[0], \dots, X[N-1]$. We can always represent any such linear transformation in matrix-vector form.

Define

$$\mathbf{x} = \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix} \quad \text{and} \quad \mathbf{X} = \begin{bmatrix} X[0] \\ X[1] \\ \vdots \\ X[N-1] \end{bmatrix}.$$

Then we can write the 1D DFT expression (cf. (7.17)):

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-i2\pi nk/N}$$

in matrix-vector form as follows [6, Section 5.1.3]:

$$\mathbf{X} = \mathbf{W}\mathbf{x}, \tag{7.25}$$

where \mathbf{W} is a $N \times N$ matrix with elements

$$W_{kn} = e^{-i2\pi kn/N}, \quad n = 0, \dots, N-1, \quad k = 0, \dots, N-1.$$

The MATLAB command `W = dftmtx(N)` generates this matrix, or just use `W = fft(eye(N))`.

Because harmonic complex exponential signals are orthogonal (see (7.4)), the columns (and rows) of the matrix \mathbf{W} are orthogonal. Hence $\mathbf{W}'\mathbf{W}$ is a diagonal matrix, with diagonal elements corresponding to the energy of each of the columns of \mathbf{W} . Specifically,

$$\mathbf{W}'\mathbf{W} = N\mathbf{I} = \mathbf{W}\mathbf{W}' \quad (7.26)$$

where \mathbf{I} is the $N \times N$ identity matrix. Unfortunately, we cannot say the matrix \mathbf{W} is an **orthogonal matrix**, because (due to slightly inconsistent terminology in the field) an orthogonal matrix has *orthonormal* columns. Furthermore, because \mathbf{W} is complex, even if it had orthonormal columns, the correct term would be **unitary**, not orthogonal.

It follows from (7.26) and definition of matrix inverse that:

$$\mathbf{W}^{-1} = \frac{1}{N}\mathbf{W}'.$$

Thus, from (7.25), the matrix-vector form of the synthesis equation (7.18) is:

$$\mathbf{x} = \mathbf{W}^{-1}\mathbf{X} = \frac{1}{N}\mathbf{W}'\mathbf{X}.$$

We can also view **Parseval's relation** for the DFT in matrix vector form:

$$\mathbf{x}'\mathbf{y} = \left(\frac{1}{N}\mathbf{W}'\mathbf{X}\right)' \left(\frac{1}{N}\mathbf{W}'\mathbf{Y}\right) = \frac{1}{N^2}\mathbf{X}'\mathbf{W}\mathbf{W}'\mathbf{Y} = \frac{1}{N^2}\mathbf{X}'(N\mathbf{I})\mathbf{Y} = \frac{1}{N}\mathbf{X}'\mathbf{Y},$$

and as a special case:

$$\|\mathbf{x}\|^2 = \mathbf{x}'\mathbf{x} = \frac{1}{N}\mathbf{X}'\mathbf{X} = \frac{1}{N}\|\mathbf{X}\|^2.$$

Unitary DFT

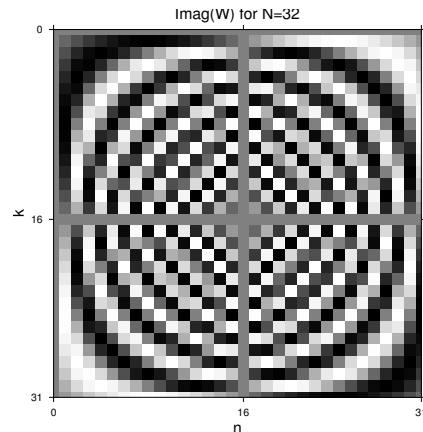
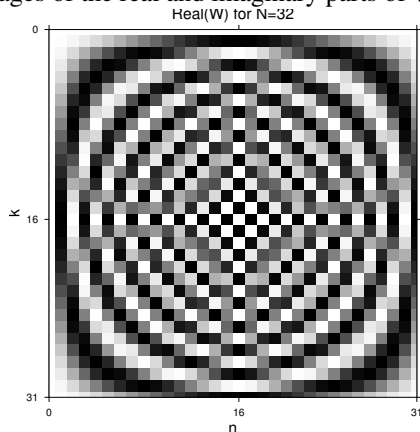
Sometimes to avoid the $1/N$ factor, we define a **unitary DFT** matrix

$$Q = \frac{1}{\sqrt{N}} W,$$

for which Parseval's theorem becomes $\|Qx\|^2 = \|x\|^2$. In this case Q is a **unitary matrix**, but sometimes people say say "orthonormal DFT."

The matrix formulation is very useful for analysis, but it is rarely used for implementation because matrix multiplication Wx is $O(N^2)$ operations whereas the FFT is $O(N \log N)$.

Here are images of the real and imaginary parts of W .



Matrix representation of 2D DFT _____ (See [2, Section 5.2].)

The 2D DFT is also a linear transformation from the MN signal values

$$x[m, n], \quad m = 0, \dots, M - 1, \quad n = 0, \dots, N - 1$$

to the MN DFT coefficients

$$X[k, l], \quad k = 0, \dots, M - 1, \quad l = 0, \dots, N - 1.$$

We can also represent this linear transformation by a $MN \times MN$ matrix. Yet apparently MATLAB does not have a single built-in command for constructing this matrix. How can we construct it?

The first thing we must do is arrange the 2D set of image values into a 1D vector. The standard method for this is called **lexicographic ordering**, defined as follows:

$$\mathbf{x} = [x[0, 0] \ x[1, 0] \ \dots \ x[M - 1, 0] \ x[0, 1] \ \dots \ x[M - 1, 1] \ \dots \ x[0, N - 1] \ \dots \ x[M - 1, N - 1]]^T$$

In MATLAB, if the image array is represented by the $M \times N$ matrix:

$$\text{xarray} = \begin{bmatrix} x[0, 0] & \dots & x[0, N - 1] \\ \vdots & \ddots & \vdots \\ x[M - 1, 0] & \dots & x[M - 1, N - 1] \end{bmatrix},$$

then forming the vector \mathbf{x} from the array `xarray` is as simple as typing:

$$\mathbf{x} = \text{xarray}(:,),$$

which is sometimes written mathematically as

$$\mathbf{x} = \text{vec}(\text{xarray}).$$

We can similarly arrange the DFT coefficients $X[k, l]$ in lexicographic ordering as a vector \mathbf{X} .

With the above definitions, the relationship between \mathbf{X} and \mathbf{x} is given as follows:

$$\mathbf{X} = (\mathbf{W}_N \otimes \mathbf{W}_M) \mathbf{x}, \quad (7.27)$$

where $\mathbf{A} \otimes \mathbf{B}$ denotes the **Kronecker product** of two matrices, defined by

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11} \mathbf{B} & \dots & a_{1n} \mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1} \mathbf{B} & \dots & a_{mn} \mathbf{B} \end{bmatrix}.$$

MATLAB's `kron` command computes the Kronecker product. So `kron(dftmtx(N), dftmtx(M))` constructs the (M, N) -point 2D DFT matrix. For a 128×128 image, this matrix would be $128^2 \times 128^2$, which would require 4Gbyte to store in the usual double precision format. And even if we could store it, the expression (7.27) is very inefficient computationally compared to the 2D FFT. However, for *analysis* purposes, the representation (7.27) can be quite useful.

Example. If $\mathbf{Y} = \mathbf{A}\mathbf{X}$, where \mathbf{X} is a random vector with covariance matrix $\text{Cov}\{\mathbf{X}\}$, then the covariance matrix of \mathbf{Y} is given by

$$\text{Cov}\{\mathbf{Y}\} = \text{Cov}\{\mathbf{A}\mathbf{X}\} = \mathbf{A} \text{Cov}\{\mathbf{X}\} \mathbf{A}'.$$

Suppose $x[n]$ includes both a deterministic signal component and an additive random noise component, *i.e.*,

$$x[n] = \mu[n] + \varepsilon[n],$$

where $\mu[n]$ is deterministic and $\varepsilon[n]$ is an uncorrelated sequence of random variables all having the same variance σ^2 . Then of course $\text{Cov}\{\mathbf{x}\} = \sigma^2 \mathbf{I}$. Suppose we take the N -point DFT of $x[n]$; what is the covariance of the DFT coefficients?

$$\text{Cov}\{\mathbf{X}\} = \text{Cov}\{\mathbf{W}\mathbf{x}\} = \mathbf{W} \text{Cov}\{\mathbf{x}\} \mathbf{W}' = \mathbf{W} \sigma^2 \mathbf{I} \mathbf{W}' = \sigma^2 \mathbf{W} \mathbf{W}' = \sigma^2 \mathbf{N} \mathbf{I}.$$

So if the signal values are uncorrelated (and have the same variance) then the DFT coefficients are also uncorrelated (and have the same variance).

This general conclusion applies to any orthogonal transformation, and is the foundation for modern image processing methods like “denoising using wavelets” [7].

Separable 2D linear operations in matrix-vector form, and Kronecker products (skip)

For a 1D linear operation of the form

$$y[k] = \sum_{n=0}^{N-1} a_{kn} x[n], \quad k = 0, \dots, K-1,$$

it is trivial to represent it in matrix form:

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

where \mathbf{A} is a $K \times N$ matrix with elements a_{kn} , \mathbf{y} is the length- K column vector with elements $\mathbf{y} = (y[0], \dots, y[K-1])$ and similarly $\mathbf{x} = (x[0], \dots, x[N-1])$. The only subtle point here is that the indices start from 0 rather than 1.

Now consider a 2D linear operation of the following form

$$y[k, l] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} c[k, l; m, n] x[m, n], \quad k = 0, \dots, K-1, \quad l = 0, \dots, L-1. \quad (7.28)$$

To represent this in matrix form, we form the length- MN column vector \mathbf{x} using **lexicographic ordering**:

$$\mathbf{x} = \left(\underbrace{x[0, 0], \dots, x[M-1, 0]}_{n=0}, \underbrace{x[0, 1], \dots, x[M-1, 1]}_{n=1}, \dots, \underbrace{x[0, N-1], \dots, x[M-1, N-1]}_{n=N-1} \right).$$

Similarly we define the length- KL column vector \mathbf{y} . Then we can write (7.28) in matrix-vector form as $\mathbf{y} = \mathbf{C}\mathbf{x}$, where \mathbf{C} is the $KL \times MN$ matrix having elements

$$C_{lK+k, nM+m} = c[k, l; m, n], \quad \text{or equivalently: } C_{ij} = c[i \bmod K, \lfloor i/K \rfloor; j \bmod M, \lfloor j/M \rfloor], \quad \begin{array}{l} i = 0, \dots, KL-1 \\ j = 0, \dots, MN-1, \end{array} \quad (7.29)$$

where $\lfloor \cdot \rfloor$ denotes the **floor** function.

The expression for the matrix C simplifies in the important special case where the factors $c[k, l; m, n]$ are **separable**, i.e., when $c[k, l; m, n] = a[k, m]b[l, n]$. In this case, (7.28) becomes the separable 2D operation:

$$y[k, l] = \sum_{n=0}^{N-1} b[l, n] \left(\sum_{m=0}^{M-1} a[k, m] x[m, n] \right). \quad (7.30)$$

The **Kronecker product** of a $L \times N$ matrix B with a $K \times M$ matrix A is the $KL \times MN$ matrix defined by:

$$B \otimes A = \begin{bmatrix} b_{11}A & \dots & b_{1N}A \\ \vdots & \vdots & \vdots \\ b_{L1}A & \dots & b_{LN}A \end{bmatrix}. \quad (7.31)$$

Properties of the Kronecker product include

- $(A \otimes B)(C \otimes D) = (AC \otimes BD)$, provided the sizes of the matrices are compatible
- $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$, provided A and B are invertible
- $(A \otimes B)' = A' \otimes B'$
- $[B \otimes A]_{lK+k, nM+m} = a_{k,m}b_{l,n}$.

Comparing this last property with (7.29), we see that the required matrix C is simply

$$C = B \otimes A.$$

Example. For the 2D DFT we have $K = N$ and $L = M$ and $c[k, l; m, n] = e^{-i\frac{2\pi}{M}km} e^{-i\frac{2\pi}{N}ln}$. In this case \mathbf{A} is the M -point DFT matrix and \mathbf{B} is the N -point DFT matrix.

Example. For 2D **linear convolution** of a $M \times N$ image with a separable $J_1 \times J_2$ impulse response $h[m, n] = h_1[m] h_2[n]$ we have $\bar{K} = M + J_1 - 1$ and $L = N + J_2 - 1$ and $c[k, l; m, n] = h_1[k - m] h_2[l - n]$. In this case, \mathbf{A} and \mathbf{B} are both (rectangular) **Toeplitz** matrices, because they are constant valued along the “diagonals.” We call $\mathbf{B} \otimes \mathbf{A}$ a **block Toeplitz - Toeplitz block (BTTB)** matrix.

Example. For 2D **circular convolution** of a $M \times N$ image $x[m, n]$ with a separable impulse response $h[m, n] = h_1[m] h_2[n]$ we have $\bar{K} = M$ and $L = N$ and $c[k, l; m, n] = h_1[(k - m) \bmod M] h_2[(l - n) \bmod N]$. In this case, \mathbf{A} and \mathbf{B} are both (square) **circulant** matrices. We call $\mathbf{B} \otimes \mathbf{A}$ a **block circulant - circulant block (BCCB)** matrix.

7.4 Fast Fourier transforms (FFT) in 2D

Brute-force evaluation of the 2D DFT (7.17) or its inverse (7.18) would require $O((MN)^2)$ flops. By recognizing that the 2D DFT is a **separable** operation, we can reduce greatly the computation.

Ignoring the braces, we can rewrite the DFT expression as follows:

$$X[k, l] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m, n] e^{-i2\pi\left(\frac{k}{M}m + \frac{l}{N}n\right)} = \sum_{n=0}^{N-1} e^{-i\frac{2\pi}{N}ln} \left[\sum_{m=0}^{M-1} x[m, n] e^{-i\frac{2\pi}{M}km} \right].$$

This is called the **row-column decomposition**. Apply the 1D DFT to each row of the image, and then apply the 1D DFT to each column of the result. Naturally we will want to use the **fast Fourier transform (FFT)** for these 1D DFTs, which thus reduces the computation to $O(N \cdot M \log M)$ for the inner set of 1D FFTs, and then $O(M \cdot N \log N)$ for the outer set of 1D FFTs, for a total of $O(MN \log MN)$ flops. For a 512^2 image, the savings in using the row-column with 1D FFTs is about a factor of 15000 relative to the brute-force 2D DFT!

See [\[wiki\]](#) for the 1D FFT method.

In MATLAB, the `fft` routine applies the 1D FFT to each column of the supplied matrix. So the most basic version of the `fft2` routine could be written in one line as follows:

```
fft(fft(x) .') .' or fft(fft(x, [], 1), [], 2)
```

Why the `.'` above? [We need the regular transpose, whereas `'` is a conjugate transpose in MATLAB.](#)

There are also “vector FFT” approaches that can improve over this row-column approach [8].

Nonuniform FFTs

When would one not use the FFT? In some applications, such as MRI and certain versions of tomography, one needs frequency samples that are nonuniformly spaced. Several papers have addressed fast algorithms for this problem beginning with [9] and including [10–17]. Such methods are often called the nonuniform FFT, or NUFFT. Many of these algorithms have been presented only for the case of 1D signals. We have addressed the multidimensional case in a min-max optimal way in [18]. For least-squares optimal NUFFT see [19, 20].

Tricks

There are many useful FFT tricks to further accelerate DFT calculations.

Example. One can compute the DFT of a $2N$ -point *real* signal by just one N -point FFT call [6, p. 476].

Example. One can compute the DFT of two *real* signals by just one FFT call [6, p. 475].

(HW!)

This trick, when extended to 2D, is quite useful for convolving two real images.

To perform 2D convolution of an $M \times N$ image $f[m, n]$ with a (typically odd-sized) filter $h[m, n]$, the simplest way in MATLAB is

```
g = ifft2(psf2otf(h) .* fft2(f));
```

Notice that no `fftshift` operations are needed here because some appropriate shift is built into the `psf2otf` command.

7.5 Relations between various Fourier transforms

We have now covered all of the flavors of Fourier transforms used in this course.

This section elaborates on how they are related.

Which transform?

- In DS, the DFS is the most appropriate Fourier tool for 2D periodic sequences defined on \mathbb{Z}^2 .
- The DSFT is the most appropriate tool for finite-energy 2D images defined on \mathbb{Z}^2 , particularly for the frequency response $H(\Omega_1, \Omega_2)$ of LSI systems.
- The DFT is the most natural tool for **finite-domain** / **finite-extent** / **finite-support** signals.

In a sense, the DFS and the DFT represent opposite “extremes” in terms of the signal domain. The DSFT is somewhat in between.

2D Fourier transform summary

CS FT:

- Analysis: $G_a(\nu_x, \nu_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g_a(x, y) e^{-i2\pi(\nu_x x + \nu_y y)} dx dy$
- Synthesis: $g_a(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} G_a(\nu_x, \nu_y) e^{i2\pi(\nu_x x + \nu_y y)} d\nu_x d\nu_y$

DS FT:

- Analysis: $G_d(\Omega_1, \Omega_2) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g_d[m, n] e^{-i(\Omega_1 m + \Omega_2 n)}$
- Synthesis: $g_d[m, n] = \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} G_d(\Omega_1, \Omega_2) e^{i(\Omega_1 m + \Omega_2 n)} d\Omega_1 d\Omega_2$

2D DFT:

- Analysis: $X[k, l] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m, n] e^{-i2\pi(\frac{k}{M}m + \frac{l}{N}n)}$, $k = 0, \dots, M-1$, $l = 0, \dots, N-1$
- Synthesis: $x[m, n] = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} X[k, l] e^{i2\pi(\frac{k}{M}m + \frac{l}{N}n)}$, $m = 0, \dots, M-1$, $n = 0, \dots, N-1$

Discrete-space relationships

If $g_d[m, n]$ has domain \mathbb{Z}^2 and is **space limited** with support on $\mathcal{D}_{M,N}$, then its (M, N) -point 2D DFT and DSFT are related as follows:

$$\underbrace{G[k, l] = G_d\left(\frac{2\pi}{M}k, \frac{2\pi}{N}l\right)}_{\text{spectral sampling}}, \quad \underbrace{G_d(\Omega_1, \Omega_2) = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} G[k, l] p_M\left(\Omega_1 - \frac{2\pi}{M}k\right) p_N\left(\Omega_2 - \frac{2\pi}{N}l\right)}_{\text{spectral interpolation}}, \quad (7.32)$$

where p_N is a kind of **periodic sinc** interpolator defined by $p_N(\Omega) \triangleq \frac{1}{N} \sum_{n=0}^{N-1} e^{-i\Omega n}$, related to the **Dirichlet kernel**.

Exercise. Verify the spectral interpolation formula (7.32) using the DS FT analysis formula and the 2D DFT synthesis formula.

Sampling relationships

If $g_d[m, n] = g_a(m\Delta_x, n\Delta_y)$ for any 2D image, then as discussed in (5.17):

$$G_d(\Omega_1, \Omega_2) = \frac{1}{\Delta_x \Delta_y} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} G_a\left(\frac{\Omega_1/2\pi - k}{\Delta_x}, \frac{\Omega_2/2\pi - l}{\Delta_y}\right).$$

Furthermore, if $g_a(x, y)$ is **band-limited** with $\nu_x^{\max} < \frac{1}{2\Delta_x}$ and $\nu_y^{\max} < \frac{1}{2\Delta_y}$ then the DS FT and the CS FT are related by

$$G_d(\Omega_1, \Omega_2) = \frac{1}{\Delta_x \Delta_y} G_a\left(\frac{\text{mod}(\Omega_1 + \pi, 2\pi) - \pi}{2\pi \Delta_x}, \frac{\text{mod}(\Omega_2 + \pi, 2\pi) - \pi}{2\pi \Delta_y}\right) \quad (7.33)$$

$$G_a(\nu_x, \nu_y) = \Delta_x \Delta_y G_d(2\pi \Delta_x \nu_x, 2\pi \Delta_y \nu_y) \text{rect}_2(\nu_x \Delta_x, \nu_y \Delta_y). \quad (7.34)$$

The modulo operation ensures that as Ω_1 ranges over $[-\pi, \pi]$, the first argument of $G_a(\nu_x, \nu_y)$ ranges over $\left[-\frac{1}{2\Delta_x}, \frac{1}{2\Delta_x}\right]$.

The relationship (7.34) offers the enticing possibility of determining the spectrum of an analog signal from (a finite set of) its samples using (7.32). However, usually this can at best be an approximation as explained below.

Here is one more relationship worth noting.

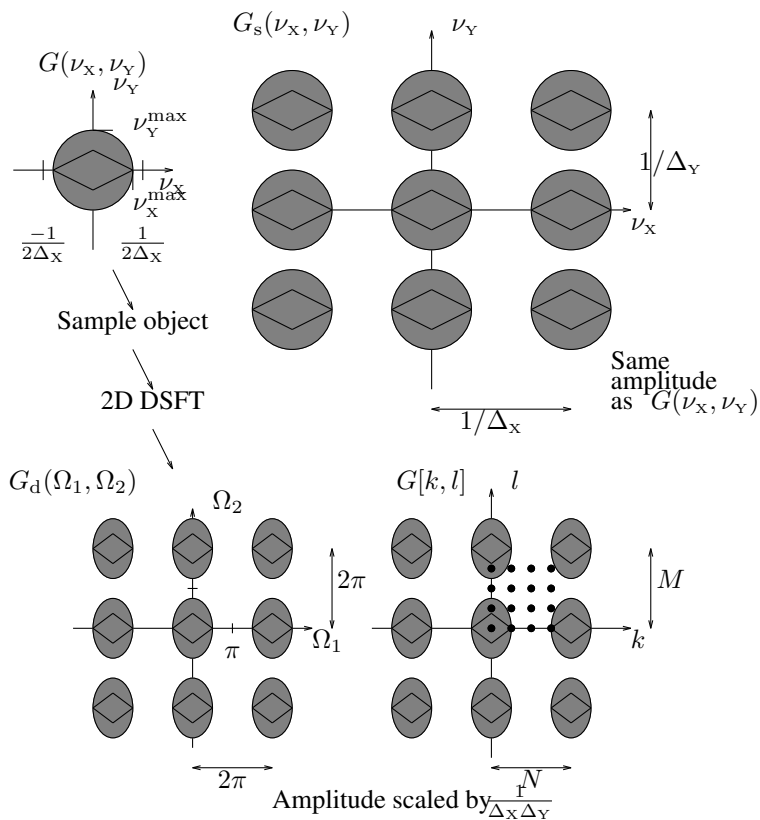
Recall from (4.3) that

$$g_s(x, y) \triangleq \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \Delta_x \Delta_y g_d[m, n] \delta_2(x - m\Delta_x, y - n\Delta_y).$$

Taking the 2D FT of both sides yields the following relationship between the spectrum $G_s(\nu_x, \nu_y)$ and the DSFT:

$$\begin{aligned} G_s(\nu_x, \nu_y) &= \Delta_x \Delta_y \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g_d[m, n] e^{-i2\pi(m\Delta_x \nu_x + n\Delta_y \nu_y)} \\ &= \Delta_x \Delta_y G_d(\Omega_1, \Omega_2) \Big|_{\Omega_1=2\pi\nu_x \Delta_x, \Omega_2=2\pi\nu_y \Delta_y} = \Delta_x \Delta_y G_d(2\pi\nu_x \Delta_x, 2\pi\nu_y \Delta_y). \end{aligned} \quad (7.35)$$

This figure illustrates the relationships between the various spectra. (Circles can become ellipses when $\Delta_x \neq \Delta_y$.)



Using FFT to approximate analog spectra $G_a(\nu_x, \nu_y)$

(important!)

So far we have not made any approximations. However, we have also not yet directly related $G[k, l]$ to $G_a(\nu_x, \nu_y)$. We would like to have such a relationship, because often we are interested in the spectrum $G_a(\nu_x, \nu_y)$ of an image $g_a(x, y)$, yet all we have available is a finite set of samples $\{g_d[m, n]\}_{m=0}^{M-1} \{n=0}^{N-1}$ of that image. All digital oscilloscopes rely on such a relationship!

Recall from (7.33) that if $g_a(x, y)$ is suitably **band-limited** then

$$G_d(\Omega_1, \Omega_2) = \frac{1}{\Delta_x \Delta_y} G_a \left(\frac{\text{mod}(\Omega_1 + \pi, 2\pi) - \pi}{2\pi \Delta_x}, \frac{\text{mod}(\Omega_2 + \pi, 2\pi) - \pi}{2\pi \Delta_y} \right).$$

Also recall from (7.32) that if the samples $g_d[m, n]$ are suitably **space-limited** then

$$G[k, l] = G_d \left(\frac{2\pi}{M} k, \frac{2\pi}{N} l \right).$$

Combining these two expressions and simplifying yields the following very useful relationship:

$$G[k, l] = \frac{1}{\Delta_x \Delta_y} G_a \left(\frac{\text{mod}(k + \frac{M}{2}, M) - \frac{M}{2}}{M \Delta_x}, \frac{\text{mod}(l + \frac{N}{2}, N) - \frac{N}{2}}{N \Delta_y} \right), \quad \begin{array}{l} k = 0, \dots, M-1 \\ l = 0, \dots, N-1. \end{array} \quad (7.36)$$

The modulo operations in (7.36) represent the first-quadrant sampling shown in the previous figure. Rarely is the relationship written this precisely, because it is hard to remember this way. More often it is written as follows:

$$\boxed{\tilde{G}[k, l] = \frac{1}{\Delta_x \Delta_y} G_a \left(\frac{k}{M \Delta_x}, \frac{l}{N \Delta_y} \right)}, \quad \begin{array}{l} k = -M/2, \dots, M/2 - 1 \\ l = -N/2, \dots, N/2 - 1, \end{array} \quad (7.37)$$

where one must remember to apply a `fftshift` to $G[k, l]$ to create $\tilde{G}[k, l]$.

Note the important frequency sampling relationships:

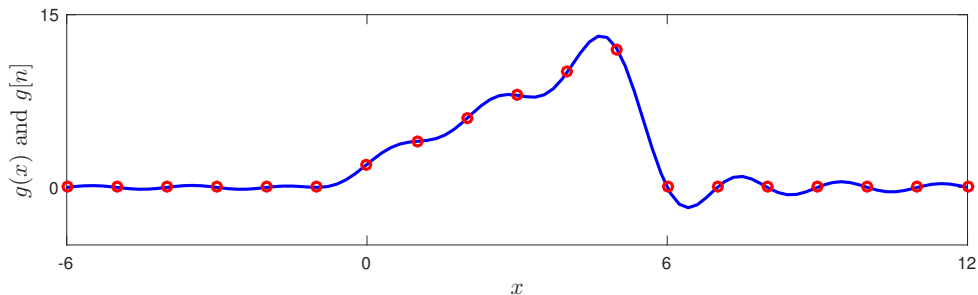
$$\nu_x = \frac{k}{M\Delta_x}, \quad \nu_y = \frac{l}{N\Delta_y}, \quad k = -M/2, \dots, M/2 - 1, \quad l = -N/2, \dots, N/2 - 1. \quad (7.38)$$

The expressions (7.36) and (7.37) hold for any signal that is both suitably **band-limited** and whose samples are suitably **space-limited**. *This is a very restricted family of signals!* Every signal in this family can be written as the following *finite* sum:

$$g_a(x, y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} g_d[m, n] \operatorname{sinc}_2\left(\frac{x}{\Delta_x} - m, \frac{y}{\Delta_y} - n\right).$$

Rarely do real-world signals belong to this family!

Example. Here is a 1D signal that is both band-limited (with $\nu_x^{\max} = 1/2$ cycles/sample) and whose samples are space-limited.



Does this example contradict the time-bandwidth product constraints that say a signal cannot be both band-limited and space-limited? ??

Fortunately, many practical real-world images of interest are *approximately* **band-limited**, even if they are **space limited**. Thus we can treat (7.36) and (7.37) as useful **approximations**, provided the sampling intervals are “reasonably small.”

Exercise. Manipulate (7.37) to find an expression for the spectrum $G_a(\nu_x, \nu_y)$ in terms of the DFT coefficients $G[k, l]$.

Example. The brain spectrum figure on page 2.26 used (7.37) and (7.38) to label the spectra appropriately!

FFT and fftshift

The 2D DFT can be computed rapidly using the **2D Fast Fourier Transform (FFT)**, e.g., the `fft2` routine in MATLAB. The Numerical Recipes book [21] gives a nice discussion, particularly about `fftshift`!

However, there is a subtle point that is often a source of confusion when working with the FFT. This point can be described using the 1D FFT as follows. Most 1D FFT routines return coefficients as a vector ordered as follows:

$$[G_0 \ G_1 \ \dots \ G_{N/2-1} \ | \ G_{N/2} \ G_{N/2+1} \ \dots \ G_{N-1}],$$

i.e., for $k = 0, \dots, N-1$, where we have inserted the bar “|” to show the point that divides the vector into two pieces each with $N/2$ elements. This is not the same ordering as one might expect from (7.37). If you apply the `fftshift` command to the FFT output, then the vector is re-ordered as follows:

$$[G_{N/2} \ G_{N/2+1} \ \dots \ G_{N-1} \ | \ G_0 \ G_1 \ \dots \ G_{N/2-1}].$$

Because the DFT can be interpreted as being periodic with period N , this ordering is equivalent to:

$$[G_{-N/2} \ G_{-N/2+1} \ \dots \ G_{-1} \ | \ G_0 \ G_1 \ \dots \ G_{N/2-1}].$$

This corresponds to (7.37). Note that the point $k = N/2$ is not included. It is not needed because $G_{N/2} = G_{-N/2}$ due to the periodicity of the DSFT. Note that the above ordering may look “asymmetric,” but it is exactly the right sampling for the FFT. Note that the DC value (G_0) is just to the *right* of center. It is not quite in the “middle” of the vector.

Mathematically, if $H = \text{fftshift}(G)$ when N is even, then $H[k] = G[(k - N/2) \bmod N]$ for $k = 0, \dots, N-1$.

Similar considerations apply in 2D. (The four quadrants are swapped around by `fftshift` and `ifftshift`.)

Caution: never use `(-n/2) : (n/2)` or `linspace(-pi, pi, n)` anywhere near a `fft`!!!

Here is how to remember whether to use `fftshift` or `ifftshift`.

- **fftshift**: input array has 0 location at the array start, output array has 0 location in the middle.

“Shift zero-frequency component to center”

Useful for more intuitive display of the output of `fft` and sometimes of `ifft`

Example. `fftshift([0 10 20 10])` is `[20 10 0 10]`

- **ifftshift**: input array has 0 location in the middle, output array has 0 location at the start.

Useful if the data coordinates have 0 in the middle and we want to apply `fft` or `ifft` to it.

Example. `ifftshift([20 10 0 10 20])` is `[0 10 20 20 10]`

This is an inverse of `fftshift`, not a “shift for `ifft`”

Summary of key relationships

Space domain:

$$g_d[m, n] = g_a(m\Delta_x, n\Delta_y)$$

$$g_s(x, y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g_d[m, n] \Delta_x \Delta_y \delta_2(x - m\Delta_x, y - n\Delta_y) = g_a(x, y) \left[\text{comb}_2\left(\frac{x}{\Delta_x}, \frac{y}{\Delta_y}\right) \right]$$

$$g_a(x, y) = g_s(x, y) ** \left[\frac{1}{\Delta_x \Delta_y} \text{sinc}_2\left(\frac{x}{\Delta_x}, \frac{y}{\Delta_y}\right) \right] \quad (\text{if suitably bandlimited})$$

$$g_a(x, y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g_d[m, n] \text{sinc}_2\left(\frac{x - m\Delta_x}{\Delta_x}, \frac{y - n\Delta_y}{\Delta_y}\right) \quad (\text{if suitably bandlimited})$$

Frequency domain:

$$G_s(\nu_x, \nu_y) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} G_a(\nu_x - k/\Delta_x, \nu_y - l/\Delta_y)$$

$$G_a(\nu_x, \nu_y) = G_s(\nu_x, \nu_y) \operatorname{rect}_2(\nu_x \Delta_x, \nu_y \Delta_y) \quad (\text{if suitably bandlimited})$$

$$G_s(\nu_x, \nu_y) = \Delta_x \Delta_y G_d(2\pi \Delta_x \nu_x, 2\pi \Delta_y \nu_y)$$

$$G_a(\nu_x, \nu_y) = \Delta_x \Delta_y G_d(2\pi \Delta_x \nu_x, 2\pi \Delta_y \nu_y) \operatorname{rect}_2(\nu_x \Delta_x, \nu_y \Delta_y) \quad (\text{if suitably bandlimited})$$

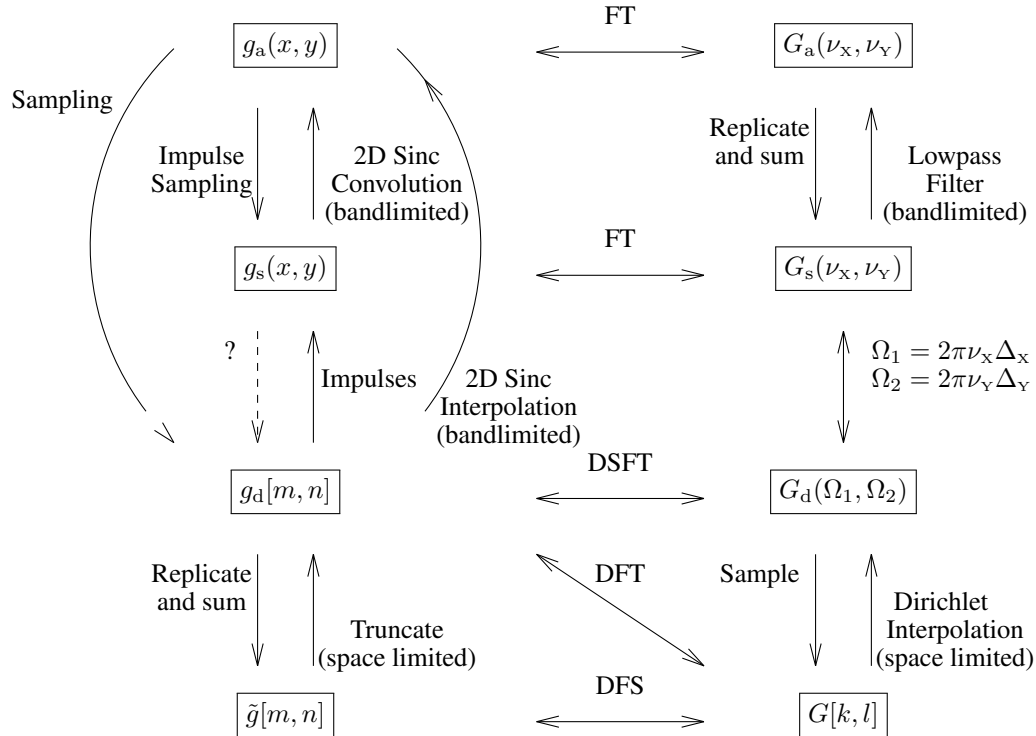
$$G[k, l] = G_d\left(\frac{2\pi}{M}k, \frac{2\pi}{N}l\right) \quad (\text{if suitably spacelimited})$$

$$G[k, l] = \frac{1}{\Delta_x \Delta_y} G_s\left(\frac{k}{M\Delta_x}, \frac{l}{N\Delta_y}\right) \quad (\text{if suitably spacelimited})$$

$$\tilde{G}[k, l] \approx \frac{1}{\Delta_x \Delta_y} G_a\left(\frac{k}{M\Delta_x}, \frac{l}{N\Delta_y}\right), \quad -M/2 \leq k \leq M/2 - 1, \quad -N/2 \leq l \leq N/2 - 1$$

$$G_d(\Omega_1, \Omega_2) = \frac{1}{\Delta_x \Delta_y} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} G_a\left(\frac{\Omega_1/2\pi - k}{\Delta_x}, \frac{\Omega_2/2\pi - l}{\Delta_y}\right)$$

The following chart summarizes some of the above relationships.



Exercise. In the diagram, one arrow is labeled with a question mark.
Give a mathematical formula that expresses $g_d[m, n]$ in terms of $g_s(x, y)$.

??

Is the answer unique? ??

Does $g_d[m, n] = \left(g_s(x, y) ** \left(\text{tri}\left(\frac{x}{\Delta_x}\right) \text{tri}\left(\frac{y}{\Delta_y}\right) \right) \right) \Big|_{x=m\Delta_x, y=n\Delta_y}$? ??

[RQ]

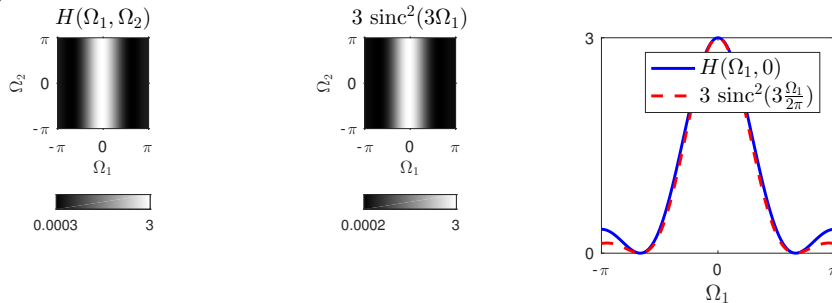
7.6 Numerical evaluation of Fourier transforms via the FFT

Numerical evaluation of DSFT via the FFT

Per (7.21), if $x[m, n]$ is supported on $\mathcal{D}_{M,N}$ then the DFT $X[k, l]$ of $x[m, n]$ corresponds to samples of the DSFT $X(\Omega_1, \Omega_2)$. Often we have a signal $h[m, n]$ that is supported on, say, $\{-M \leq m \leq M\} \times \{-N \leq n \leq N\}$ and we wish to evaluate its DSFT $H(\Omega_1, \Omega_2)$ quickly using a FFT. The easiest way to do this is to use MATLAB's `psf2otf` command.

Example. Consider the signal $h[m, n] = \text{tri}(m/3) \delta[n]$ defined on \mathbb{Z}^2 . One way to determine its DSFT is to note that $h[m, n] = [1/3 \ 2/3 \ 1/3] \delta[n]$ so $H(\Omega_1, \Omega_2) = 1 + \frac{4}{3} \cos(\Omega_1) + \frac{2}{3} \cos(2\Omega_1)$. Alternatively, we can view $\text{tri}(m/3)$ as samples of $\text{tri}(x/3)$ with spacing $\Delta_x = 1$, so using the sampling relationship (5.17), we have $H(\Omega_1, \Omega_2) = \sum_{k=-\infty}^{\infty} 3 \text{sinc}^2(3(\frac{\Omega_1}{2\pi} - k))$. Interestingly, these two expressions are equal and 2π -periodic, but neither of them is the same as the (not-periodic) popular but incorrect expression $3 \text{sinc}^2(3\Omega_1)$. Because $h[m, n]$ is real and symmetric here, so is $H(\Omega_1, \Omega_2)$.

In this case we have analytical expression for $H(\Omega_1, \Omega_2)$, but often we do not, so to evaluate $H(\Omega_1, \Omega_2)$ numerically for an FIR filter $h[m, n]$ we use `psf2otf` as follows. Note that we want fine spacing in Ω_1, Ω_2 to visualize $H(\Omega_1, \Omega_2)$ well, so we choose large (M, N) , regardless of the support size of $h[m, n]$. Although the grayscale image of $H(\Omega_1, \Omega_2)$ looks similar to $3 \text{sinc}^2(3\Omega_1)$, the profiles show that they differ.



Numerical evaluation of CSFT via the FFT

In imaging work, we often need to evaluate the spectrum $G(\nu_x, \nu_y)$ of an image $g(x, y)$ for which the analytical expression for $G(\nu_x, \nu_y)$ is unknown. In such cases we often compute the FT $G(\nu_x, \nu_y)$ numerically using the **fast Fourier transform (FFT)**.

We derive the method in 1D but the principles generalize to higher dimensions.

The expression for the 1D FT $G(\nu)$ of a signal $g(x)$ is:

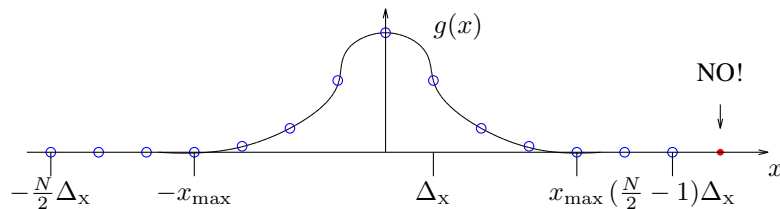
$$g(x) \xrightarrow{\mathcal{F}} G(\nu) = \int_{-\infty}^{\infty} g(x) e^{-i2\pi\nu x} dx. \quad (7.39)$$

Assume that we have an analytical expression for $g(x)$, but are unable (or unwilling) to find $G(\nu)$ analytically. To evaluate the 1D FT numerically for a signal $g(x)$ whose energy is concentrated near the origin $x = 0$, we can approximate the integral by a Riemann summation:

$$G(\nu) \approx \tilde{G}(\nu) \triangleq \Delta_x \sum_{n=-N/2}^{N/2-1} \underbrace{g(n\Delta_x)}_{\text{“sampling”}} e^{-i2\pi\nu n\Delta_x}. \quad (7.40)$$

The sum is *finite* to enable computation, and the sample spacing Δ_x takes the role of dx . We must use a finite value for N *even if the signal $g(x)$ has infinite support*. So the summation (7.40) is useful only if $g(x)$ has at least *approximately* finite support, and a minimum requirement is that $g(x) \rightarrow 0$ as $|x| \rightarrow \infty$. Often we first plot $g(x)$ and choose some x_{\max} beyond which $g(x)$ is “essentially” zero (*i.e.*, $g(x) \approx 0$ for $|x| \geq x_{\max}$) and then we must choose N and Δ_x such that

$$\frac{N}{2} \Delta_x \geq x_{\max}. \quad (7.41)$$



Of course we cannot evaluate the summation (7.40) for every conceivable value of ν . Usually we are content with computing a finite set of equally spaced ν values, say $\nu \in \{k\Delta_\nu : k = -K/2, \dots, K/2 - 1\}$, for some frequency domain sample spacing Δ_ν that we must choose. Evaluating $G(\nu)$ at those sample locations we have

$$\tilde{G}(k\Delta_\nu) = \Delta_x \sum_{n=-N/2}^{N/2-1} g(n\Delta_x) e^{-i2\pi kn\Delta_x\Delta_\nu}, \quad k = -K/2, \dots, K/2 - 1. \quad (7.42)$$

This is a sum of N terms to be evaluated for K arguments, so one could implement it directly at a cost of $O(KN)$ flops. This would be expensive for large K and N .

To reduce computation, we would like to evaluate (7.42) using an FFT. This will be possible *if* we choose Δ_x and Δ_ν such that $\Delta_x\Delta_\nu = 1/L$ for some integer L . In that case, we can evaluate (7.42) using a L -point DFT (using zero-padding if $L > N$). MATLAB's `fft` command has an optional second argument for specifying L if $L > N$.

Usually (but not always) we choose $L = K = N$, which requires that we choose Δ_x and Δ_ν such that

$$\Delta_x\Delta_\nu = 1/N. \quad (7.43)$$

In this case, (7.42) simplifies to

$$\tilde{G}(k\Delta_\nu) = \Delta_x \sum_{n=-N/2}^{N/2-1} g(n\Delta_x) e^{-i2\pi kn/N}, \quad k = -N/2, \dots, N/2 - 1, \quad (7.44)$$

which is *almost* the usual N -point DFT formula but with an extra factor of Δ_x to account for the dx in the original integral (7.39).

The approximation (7.40) involves sampling $g(x)$, which is reasonable if $g(x)$ is at least approximately band-limited. Typically $g(x)$ is not exactly band-limited, but there will be some frequency ν_{\max} beyond which $G(\nu)$ is approximately zero, in which case we say $g(x)$ is “essentially band-limited.” To avoid aliasing, we will want to satisfy the Nyquist criterion, so we want

$$\Delta_x < \frac{1}{2\nu_{\max}}.$$

We often do not know ν_{\max} in advance, so we must make a guess, compute $\tilde{G}(k\Delta_\nu)$, plot it, and then look for signs of aliasing (spectral overlap). If such overlap is present, then we can decrease the sampling interval Δ_x . Then we must also increase N accordingly to satisfy (7.41). To get a “pretty plot” of $G(\nu)$, we may need to decrease Δ_ν , and then again increase N accordingly, which will also have the effect of increasing $x_{\max} = \frac{N}{2}\Delta_x$.

The `fftshift` and `ifftshift` commands in MATLAB

We say (7.44) is “almost” the usual DFT because the usual DFT has a summation from $n = 0$ to $N - 1$, not from $-N/2$ to $N/2 - 1$. Likewise, the usual DFT involves arguments k from 0 to $N - 1$, not from $-N/2$ to $N/2 - 1$. For N even, if we define

$$\bar{g}[n] \triangleq \begin{cases} g(n\Delta_x), & n = 0, \dots, N/2 - 1 \\ g((n - N)\Delta_x), & n = N/2, \dots, N - 1, \end{cases}$$

then we can rewrite (7.44) as follows:

$$\tilde{G}(k\Delta_\nu) = \Delta_x \sum_{n=0}^{N-1} \bar{g}[n] e^{-i2\pi kn/N}, \quad k = -N/2, \dots, N/2 - 1. \quad (7.45)$$

In MATLAB, we use the command `ifftshift` to form $\bar{g}[n]$ from $g(n\Delta_x)$.

Furthermore, the complex exponential in (7.45) has the periodicity property $e^{-i2\pi kn/N} = e^{-i2\pi(k \bmod N)n/N}$. Thus, we first compute

$$\bar{G}[k] \triangleq \sum_{n=0}^{N-1} \bar{g}[n] e^{-i2\pi kn/N}, \quad k = 0, \dots, N - 1,$$

and then we form $\tilde{G}(k\Delta_\nu)$ by

$$\tilde{G}(k\Delta_\nu) = \begin{cases} \Delta_x \bar{G}[k], & k = 0, \dots, N/2 - 1 \\ \Delta_x \bar{G}[k + N], & k = -N/2, \dots, -1. \end{cases}$$

In MATLAB, we use the command `fftshift` to get $\tilde{G}(k\Delta_\nu)$ from $\bar{G}[k]$.

Note the `fftshift` and `ifftshift` commands are identical in the usual case when N is even, as has been assumed in the above derivations. When N is odd they differ, and the difference becomes very important.

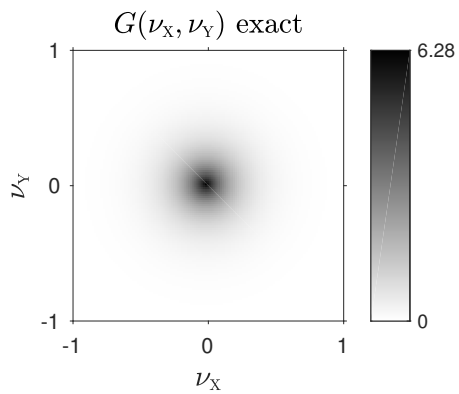
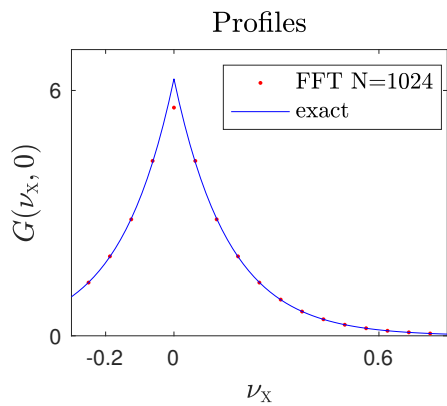
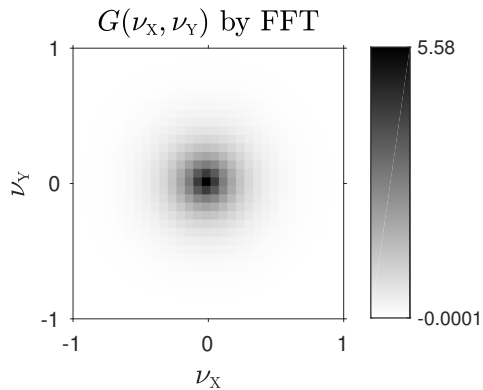
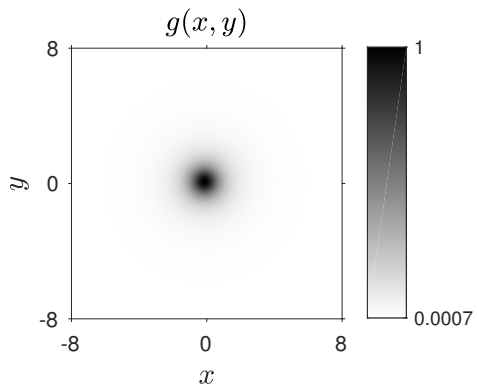
The 2D case

The 2D FT can be computed rapidly using the **2D Fast Fourier Transform (FFT)**, e.g., the `fft2` routine in Matlab. Now there is a double integral so both Δ_x and Δ_y are needed. Again, `fftshift` is often necessary. In 2D, `fftshift` swaps the four quadrants appropriately.

Example. Consider the signal $g(x, y) = 1/(1 + x^2 + y^2)^{3/2}$. What does its 2D FT look like?

See figure on next page. Here I used $N = 2^{10}$ and $x_{\max} = 8$.

Why is the DC value approximation poor here? ??



Here is the mfile that produced that figure. Notice the use of `ndgrid`.

```
% ft_example2.m Illustrate approximate 2D FT via FFT
% hankel pair: 1 / (a^2 + r^2)^3/2 <-> (2 pi / a) * exp(-2 pi a * q)

ir_fontsize reset
ir_fontsize label 15
ir_fontsize title 15

nx = 2^10; ny = nx-2;
xmax = 2^3;
dx = 2*xmax / nx;
dy = 2*xmax / ny;

x = [-nx/2:nx/2-1] * dx;
y = [-ny/2:ny/2-1] * dy;
[xx yy] = ndgrid(x, y); % make 2D array from 1D sampling coordinates

gxy = 1 ./ (1 + xx.^2 + yy.^2).^1.5; % g(x,y) sampled

Guv = dx * dy * fftshift(fft2(iffshift(gxy))); % dx dy to be quantitative
%Guv = reale(Guv);
frac = max(abs(imag(Guv(:)))) / max(abs(Guv(:)))
if frac > 1e-13, warning 'non-negligible imaginary part', end
Guv = real(Guv); % discard (almost) negligible imaginary part

nu = nx;
nv = ny;
du = 1/(nx*dx); dv = 1/(ny*dy);
u = [-nu/2:nu/2-1] * du;
v = [-nv/2:nv/2-1] * dv;

nu2 = 4*nx;
nv2 = 4*ny;
du2 = 1/(nu2*dx);
dv2 = 1/(nv2*dy);
u2 = [-nu2/2:nu2/2-1] * du2;
v2 = [-nv2/2:nv2/2-1] * dv2;
[uu vv] = ndgrid(u2,v2); % grid of frequency domain sample locations
Guv_true = 2*pi * exp(-2*pi*sqrt(uu.^2+vv.^2));

im plc 2 2
im(1, x, y, gxy), axis equal, cbar
axis([-1 1 -1 1]*8)
xtick(-8:8:8), ytick(-8:8:8)
xlabel('$x$')
ylabel('$y$')
title('$g(x,y)$')
colorbar(gca, 1-gray(256))
```

```

ax = [-1 1 -1 1]*1;
im(2, u, v, Guv), cbar
axis(ax)
xtick(-1:1:1), ytick(-1:1:1)
xlabel('\kx')
ylabel('\ky')
titlef('%G(\kx,\ky)$ by FFT')
colormap(gca, 1-gray(256))

im(4, u2, v2, Guv_true, [0 max(Guv_true(:))]), axis image, cbar
axis(ax)
xtick(-1:1:1), ytick(-1:1:1)
xlabel('\kx')
ylabel('\ky')
titlef('%G(\kx,\ky)$ exact')
colormap(gca, 1-gray(256))

if 0
im(3, u, v, abs(Guv-Guv_true)), axis image, cbar
axis(ax)
xlabel('\kx')
ylabel('\ky')
titlef('%G(\kx,\ky)$ error')
end

im subplot 3
iv = nv/2+1;
iv2 = imin(abs(v2) - v(iv));
plot(u, Guv(:,iv), 'r.', u2, Guv_true(:,iv2), 'b-')
xtick([-0.2 0 0.6])
ytick([0 6])
axis([-0.3 0.8 0 7])
xlabel('\kx')
ylabel('%G(\kx,0)$')
titlef('Profiles')
tmp = sprintf('FFT N=%d', nx);
ir_legend([tmp, 'exact'])

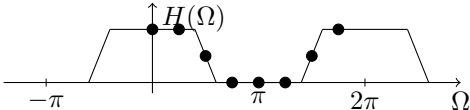
%ir_savefig cw ft_example2

```

7.7 Design of FIR filters by frequency sampling

Design of FIR filters by frequency sampling (1D)

For the **frequency sampling** method of FIR filter design, to design a M -point FIR filter we specify the desired frequency response at a set of equally-spaced frequency locations over $[0, 2\pi)$:

$$H(\Omega) \Big|_{\Omega=\frac{2\pi}{M}k}, \quad k = 0, \dots, M-1.$$


Recall from the DTFT formula that if $h_d[n]$ is nonzero only for $n = 0, \dots, M-1$, then $H(\Omega) = \sum_{n=0}^{M-1} h_d[n] e^{-i\Omega n}$. Thus, at the given frequency locations, we have

$$H\left(\frac{2\pi}{M}k\right) = \sum_{n=0}^{M-1} h_d[n] e^{-i\frac{2\pi}{M}kn}, \quad k = 0, \dots, M-1.$$

This is the formula for the M -point DFT. We can think of it as a system of M equations in M unknowns.

We compute $h_d[n]$ from the frequency response samples $\{H(\frac{2\pi}{M}k)\}_{k=0}^{M-1}$ by using the **inverse DFT** ($h = \text{ifft}(H)$):

$$h_d[n] = \frac{1}{M} \sum_{k=0}^{M-1} H\left(\frac{2\pi}{M}k\right) e^{i\frac{2\pi}{M}kn}, \quad n = 0, \dots, M-1.$$

This will be the impulse response of the M -point FIR filter as designed by the frequency sampling method.

- If we want $h_d[n]$ to be real, then $H(\Omega)$ must be Hermitian symmetric, *i.e.*, $H^*(\Omega) = H(-\Omega) = H(2\pi - \Omega)$.
So after we specify the value $H(\frac{2\pi}{M}k)$, we also have specified $H^*(\frac{2\pi}{M}k) = H(2\pi - \frac{2\pi}{M}k) = H(\frac{2\pi}{M}(M - k))$.
- Thus, software such as MATLAB's `fir2` command requires $H(\Omega)$ only on the interval $[0, \pi]$.
- If $h_d[n]$ is to be real, then it also follows that $H(\pi) = H(\frac{2\pi}{M}\frac{M}{2})$ must be real-valued when M is even.
- Often we choose $H(\Omega)$ to be linear phase, *e.g.*, $H(\Omega) = H_R(\Omega) e^{-i\Omega(M-1)/2}$, where $H_R(\Omega)$ is real-valued.

If we design a lowpass filter with cutoff $\Omega_c = \pi/2$ using just $M = 2$ frequency samples, what is the impulse response $h_d[n]$? Give the numerical values of $h_d[n]$, not a formula. [RQ]

?? What is the corresponding frequency response? Is it lowpass? ??

(do $M = 3$ in class)

Example. Suppose we want to “differentiate” an image along the horizontal direction. (We will see later in Ch. 9 that this is useful for detecting vertical edges.) For band-limited images, the “ideal” frequency response for this purpose is (see Ch. 9):

$$H_d(\Omega_1, \Omega_2) = \imath\Omega_1, \quad (\Omega_1, \Omega_2) \in [-\pi, \pi) \times [-\pi, \pi). \quad (7.46)$$

We could apply an FFT filtering approach but that would incur wrap-around effects. So we design a small $M \times N$ FIR filter using the **frequency sampling** method:

$$H_d\left(\frac{2\pi}{M}k, \frac{2\pi}{N}l\right) = \imath\frac{2\pi}{M}k.$$

We would like to use this expression for $k = 0, \dots, M-1$, $l = 0, \dots, N-1$, but we must be careful because (7.46) is valid only for $(\Omega_1, \Omega_2) \in [-\pi, \pi) \times [-\pi, \pi)$. Instead we use $k = -M/2, \dots, M/2-1$, $l = -N/2, \dots, N/2-1$, and apply `ifftshift` before taking the inverse FFT, as illustrated in the code below, leading to the figure shown after the code.

```
% fig_sample_diff1
% differentiator design by frequency sampling

M = 7;
N = 1;
k = [-(M-1)/2:(M-1)/2]; % M odd
l = [-(N-1)/2:(N-1)/2]; % N odd
[kk ll] = ndgrid(k,l);
Hd = 2i * pi / M * kk;
hmn = fftshift(ifft2(ifftshift(Hd)));

if 1 % show response
    clf, subplot(221)
    stem(k, hmn, 'filled')
    axis([-4 4 -1.5 1.5])
    xtick([-3 0 3])
    ytick([-1 0 1])
    xlabel('$m$')
    ylabel('$h[m,0]$')
```

```

om = linspace(0, pi, 101); % for plotting
Mm = (M-1)/2;
Hi = 0; % compute imaginary part of frequency response
for mm=1:Mm
    Hi = Hi - 2 * hmn(Mm+mm+1) * sin(om * mm);
end
subplot(223)
plot(om, om, 'b-', om, Hi, 'g--', ...
     2 * pi / M * k, 2 * pi / M * k, 'o')
xlabel '\omx'
ylabel 'Imag H(\omx,0)'
ir_legend({'Ideal', 'Actual'}, 'location', 'northwest')
axis_pi, yaxis_pi
axis([0 pi 0 1.2*pi])
axis equal

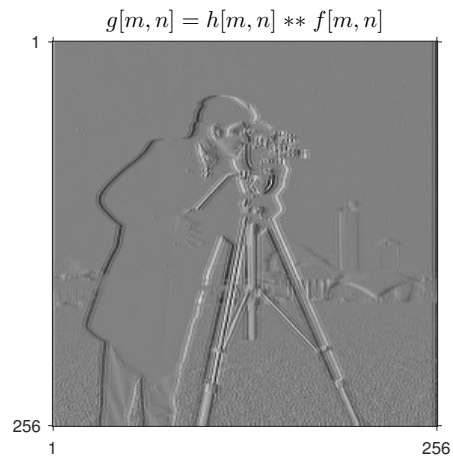
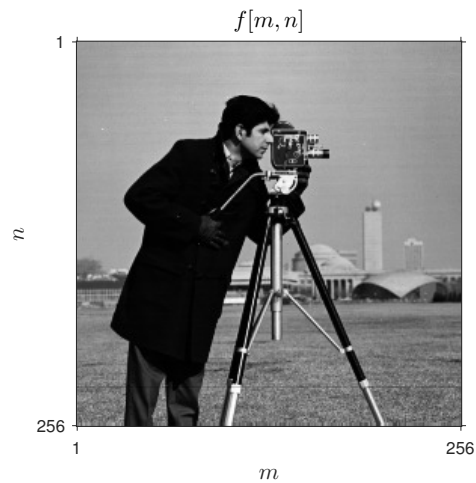
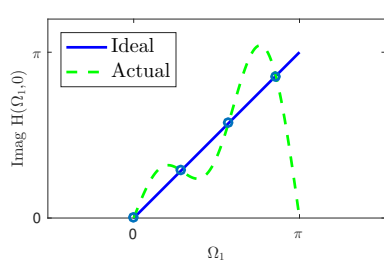
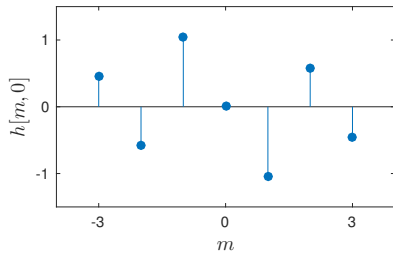
% ir_savefig cw fig_sample_diff1
%return
end

if 1 % apply to image
    f = single(imread('cameraman.tif'));

    im_plc 1 2
    im(1, f)
    titlef '\fmm'
    xlabel '$m$'
    ylabel '$n$'

    g1 = conv2(f, hmn, 'same');
    im(2, g1)
    titlef '$\gmm = \hmn \cconv \fmm$'
% ir_savefig cw fig_sample_diff1b
end

```

Design of FIR filters by frequency sampling: 2D example

Example. Return to the circularly symmetric desired frequency response specification in (5.19), namely

$$H_d(\Omega_1, \Omega_2) = \frac{1}{2} \left((1 + \cos \Omega_o) \operatorname{rect} \left(\frac{\Omega_o}{2\pi} \right) \right)_{(2\pi, 2\pi)}, \quad \Omega_o \triangleq \sqrt{\Omega_1^2 + \Omega_2^2}. \quad (7.47)$$

To design a $M \times N$ 2D FIR filter by the frequency sampling method that approximates this frequency response, we use the 2D inverse DFT of samples of $H_d(\Omega_1, \Omega_2)$ as follows:

$$h_d[m, n] = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} H_d \left(\frac{2\pi}{M} k, \frac{2\pi}{N} l \right) e^{j2\pi \left(\frac{k}{M} m + \frac{l}{N} n \right)}.$$

Careful use of `fftshift` simplifies treatment of the $(2\pi, 2\pi)$. See MATLAB's `fsamp2`.

The resulting 5×5 2D FIR filter, when quantized to two base-10 digits, is identical to that shown after (5.19). Here we used only a (5×5) -point inverse FFT whereas we used a much larger inverse FFT in the code shown after (5.19).

```
% fig_sample_cos2
% 2D FIR low-pass filter design by frequency sampling

M = 5; N = 5;
k = [-(M-1)/2:(M-1)/2]; % M odd
l = [-(N-1)/2:(N-1)/2]; % N odd
[kk ll] = ndgrid(k,l);

wmod = @(w) mod(w + pi, 2*pi) - pi; % in [-pi, pi)
%om = linspace(-3*pi, 3*pi, 201); plot(om, wmod(om))

Hd_om = @(om) 0.5 * (1 + cos(om)) .* (om < pi);
Hd_middle = @(wx,wy) Hd_om(sqrt(wx.^2 + wy.^2));
Hd_fun = @(wx,wy) Hd_middle(wmod(wx), wmod(wy));
```

```
Hdk1 = Hd_fun(2*pi*kk/M, 2*pi*ll/N);  
hmn = fftshift(iff2(iff2shift(Hdk1)));  
hmn = round(hmn * 100) / 100 % quantize to 2 decimal places
```

What part of this chapter (up to this point) did you find most confusing? ??

[RQ]

7.8 Discrete cosine transform (DCT)

The DFT/FFT are excellent for convolution, and useful for frequency-domain analysis of sampled analog signals. So why did someone invent a new transform, the DCT?

For good image compression, we would like **energy compaction**; a good transform will result in “Fourier coefficients” that are mostly near zero; we can discard or coarsely quantize the small coefficients, and use most of the bits to represent the larger coefficients. Note that for any orthogonal transform $\{\phi_{k,l}\}$ total energy is always preserved by Parseval’s theorem:

$$\sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} |x[m, n]|^2 = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \|\phi_{k,l}\|^2 |c_{k,l}|^2.$$

For image compression, what matters is how the energy is distributed among the various components.

Image compression is often done in blocks. Suppose we select a small block from some natural image. The DFT of the block gives us the values of the discrete Fourier series of the periodic extension of that signal. Suppose the periodic extension has a discontinuity at the block boundaries. Then the DFT coefficients will decay slowly, just like the FT of a square wave (discontinuous) decay as $1/k$, whereas those of a triangle wave decay as $1/k^2$. So *any* discontinuities in an image, including at the boundary of a block, lead to poor energy compaction of the DFT coefficients.

As an additional drawback of the DFT, if the image is real, then its coefficients are complex. All other things being equal, when developing image compression methods one would usually prefer real valued quantities over complex values if the original image is real.

To overcome these drawbacks of the DFT, **discrete cosine transform (DCT)** uses the trick of taking the image (block) and forming a symmetrized version of it before computing a DFT. This symmetrization is equivalent to using **mirror boundary conditions** and has the effect of

- eliminating discontinuities at the block edges, and
- yielding real coefficients for real input images.

Interestingly though, the DCT is derived via the DFT. So Fourier analysis is still the fundamental tool, even for this new transform.

1D DCT

Consider the signal of extent $N = 4$: $x[n] = (\underline{2}, 4, 6, 8)$. Its 4-point circular extension is $\tilde{x}[n] = (\dots, 2, 4, 6, 8, \underline{2}, 4, 6, 8, 2, \dots)$. Note the “discontinuity.”

Now consider instead the new signal of length $2N$:

$$y[n] = \begin{cases} x[n], & 0 \leq n \leq N - 1 \\ x[2N - 1 - n], & N \leq n \leq 2N - 1 \\ \text{undefined,} & \text{otherwise,} \end{cases}$$

which is $y[n] = (2, 4, 6, 8, 8, 6, 4, 2)$ in the example. This signal has no jumps at the boundaries.

We now derive the DCT via the DFT. For $0 \leq k \leq 2N - 1$, the $2N$ -point DFT of $y[n]$ is:

$$\begin{aligned} Y[k] &= \sum_{n=0}^{2N-1} y[n] e^{-i\frac{2\pi}{2N}kn} = \sum_{n=0}^{2N-1} y[n] W_{2N}^{kn} = \sum_{n=0}^{N-1} x[n] W_{2N}^{kn} + \sum_{n=N}^{2N-1} x[2N-1-n] W_{2N}^{kn} \\ &= \sum_{n=0}^{N-1} x[n] W_{2N}^{kn} + \sum_{n=0}^{N-1} x[n] W_{2N}^{k(2N-1-n)} = W_{2N}^{-k/2} \sum_{n=0}^{N-1} x[n] \left(W_{2N}^{k(n+1/2)} + W_{2N}^{-k(n+1/2)} \right) \\ &= W_{2N}^{-k/2} \sum_{n=0}^{N-1} x[n] 2 \cos\left(\frac{2\pi k(2n+1)}{4N}\right), \end{aligned}$$

where $W_N \triangleq e^{-i\frac{2\pi}{N}}$.

Suppose we started with a length N signal $x[n]$. Now we have $2N$ complex DFT coefficients $Y[0], \dots, Y[2N-1]$. This hardly seems like progress towards compaction! But obviously there must be some redundancies among the $Y[k]$ values.

- Note that $Y[N] = 0$, which is a consequence of the symmetry in the construction of $y[n]$.
- If $x[n]$ is real, then $W_{2N}^{k/2} Y[k]$ is *real* because it is a sum of $x[n]$ value times a cosine.
(Nevertheless the DCT is defined even for complex signals.)
- One can verify that we really need only save *half* of the $Y[k]$ values due to the following form of odd symmetry:

$$-W_{2N}^{(2N-k)/2} Y[2N-k] = W_{2N}^{k/2} Y[k], \quad k = 1, \dots, 2N-1.$$

In light of these properties, the 1D DCT is defined as follows:

$$C_x[k] \triangleq \begin{cases} W_{2N}^{k/2} Y[k], & 0 \leq k \leq N-1 \\ \text{undefined,} & \text{otherwise.} \end{cases}$$

$$C_x[k] = \sum_{n=0}^{N-1} 2x[n] \cos\left(\frac{\pi k(2n+1)}{2N}\right), \quad k = 0, \dots, N-1 \quad (\text{analysis}). \quad (7.48)$$

A few properties of the DCT:

- Maps an N -point sequence to another N -point sequence.
- If $x[n]$ is real, then so is its DCT.
- $C_x[0] = Y[0] = 2X[0]$, so the 0th component of the DCT is twice that of the DFT.
- We can express it using basis functions: $C_x[k] = \langle x, \phi_k \rangle$, where $\phi_k[n] \triangleq 2 \cos\left(\frac{\pi k(2n+1)}{2N}\right)$.
- Is the set of signals $\{\phi_k\}$ an orthogonal set over $n = 0, \dots, N-1$? **Yes**
 Why does orthogonality matter? **Simplicity in reconstruction:** $x[n] = \sum_{k=0}^{N-1} \frac{\langle x, \phi_k \rangle}{\|\phi_k\|^2} \phi_k[n]$.

For the inverse DCT, one can recover the $2N$ -point DFT coefficients $Y[k]$ from the DCT coefficients $C_x[\cdot]$ as follows:

$$Y[k] = \begin{cases} W_{2N}^{-k/2} C_x[k], & k = 0, \dots, N-1 \\ 0, & k = N \\ -W_{2N}^{-k/2} C_x[2N-k], & k = N+1, \dots, 2N-1. \end{cases}$$

Substituting into the iDFT formula and simplifying (or applying the orthogonality of the DCT basis) yields the **synthesis** formula:

$$x[n] = y[n] = \frac{1}{N} \sum_{k=0}^{N-1} w[k] C_x[k] \cos\left(\frac{\pi k(2n+1)}{2N}\right), \quad n = 0, \dots, N-1 \quad (\text{synthesis}), \quad (7.49)$$

where, because $Y[0] = 2 \sum_{n=0}^{N-1} x[n]$, we have

$$w[k] \triangleq \begin{cases} 1/2, & k = 0 \\ 1, & \text{otherwise.} \end{cases} \quad (7.50)$$

Basic algorithm for 1D DCT

Rarely does one *implement* the DCT using the two boxed formulas above, because that would require $O(N^2)$ operations. Instead one uses an FFT-based algorithm as follows.

- Extend $x[n]$ to form $y[n]$. (Use MATLAB's `fliplr` or `flipud` command.)
- Compute $2N$ -point DFT $Y[k]$ from $y[n]$. (Use MATLAB's `fft` command.)
- $C_x[k] = W_{2N}^{k/2} Y[k]$, $k = 0, \dots, N-1$ (Use MATLAB's `real` command after scaling by $W_{2N}^{k/2}$ because there will be some residual imaginary part due to finite precision.)

Similar for inverse DCT. In fact it can be done using N -point DFTs too [1, Pr. (3.21)].

Caution! MATLAB's `dct` command uses a slightly different definition of the DCT that is normalized so that it is an **orthonormal** transformation, following [2, p. 150-3].

Example: $x[n] = (2, 4, 6, 8)$ has DFT $(20, -4 + i4, -4, -4 - i4)$. The DCT is $(40, -12.6, 0, -0.897)$, which has nominally better compaction because one of the entries is zero.

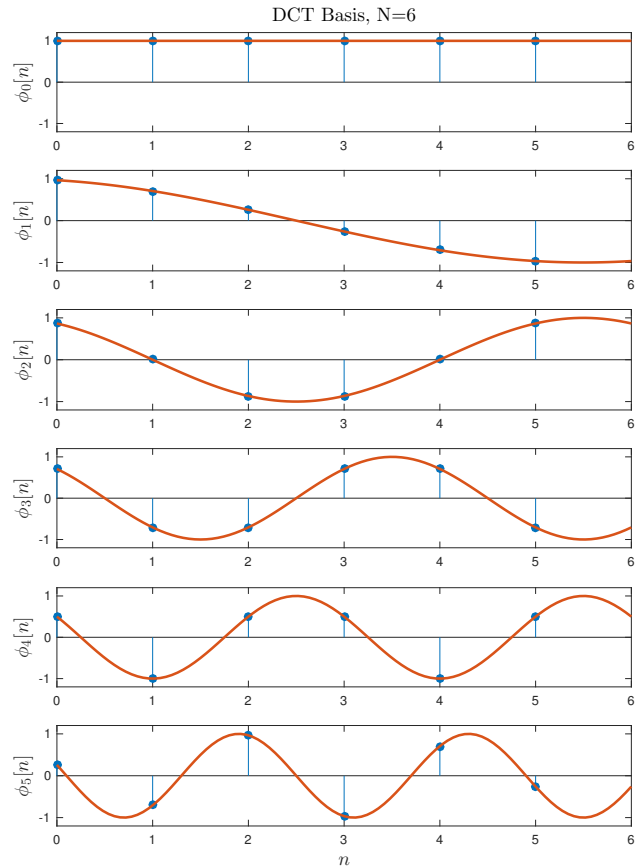
Because the DCT input sequence $y[n]$ has no extraneous sharp discontinuities, it will lead to better energy compaction in the frequency domain than the DFT input sequence $x[n]$, *i.e.*, more energy is concentrated in low frequency components.

What is the catch? What signals are better compacted by the DFT than by the DCT?

??

[RQ]

Example. Illustration of 1D DCT basis for $N = 6$.



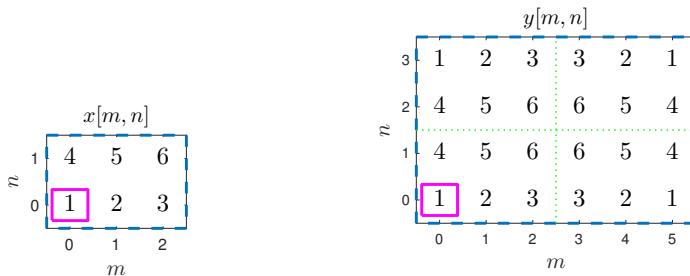
2D DCT

The 2D DCT is defined similarly to the 1D DCT, except that the symmetrizing extension (mirror boundary conditions) is a 2D operation:

$$y[m, n] = \begin{cases} x[m, n], & m = 0, \dots, M-1, n = 0, \dots, N-1 \\ x[2M-1-m, n], & m = M, \dots, 2M-1, n = 0, \dots, N-1 \\ x[m, 2N-1-n], & m = 0, \dots, M-1, n = N, \dots, 2N-1 \\ x[2M-1-m, 2N-1-n], & m = M, \dots, 2M-1, n = N, \dots, 2N-1 \\ \text{undefined,} & \text{otherwise,} \end{cases}$$

assuming $x[m, n]$ is a finite-extent signal that is nonzero only over $m = 0, \dots, M-1$, $n = 0, \dots, N-1$.

Example. For the case $M = 3$ and $N = 2$:



By a similar derivation as the 1D case [1]:

$$C_x[k, l] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} 4 x[m, n] \cos\left(\frac{\pi k(2m+1)}{2M}\right) \cos\left(\frac{\pi l(2n+1)}{2N}\right), \quad (\text{analysis}) \quad (7.51)$$

for $(k, l) \in \mathcal{D}_{M,N}$, and, where $w[k]$ was defined in (7.50):

$$x[m, n] = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} w[k] w[l] C_x[k, l] \cos\left(\frac{\pi k(2m+1)}{2M}\right) \cos\left(\frac{\pi l(2n+1)}{2N}\right), \quad (\text{synthesis}). \quad (7.52)$$

Is the 2D DCT based on a **separable** basis? **??**

[RQ]

Again, rather than using the boxed equations above, one typically uses an FFT-based algorithm.

Basic algorithm for 2D DCT

- Extend $x[m, n]$ to form $y[m, n]$ (using mirror boundary conditions).
Use MATLAB's `fliplr`, `flipud`, and `rot90` routines.
- Compute $(2M, 2N)$ -point DFT $Y[k, l]$ from $y[m, n]$. Use MATLAB's `fft2` routine.
- $C_x[k, l] = W_{2M}^{k/2} W_{2N}^{l/2} Y[k, l]$, $k = 0, \dots, M-1$, $l = 0, \dots, N-1$.
Use MATLAB's `ndgrid` or `meshgrid` routine to create the weights, and then take the `real` part to eliminate residual complex component caused by finite numerical precision.

Similar algorithm flow for inverse DCT.

There is ongoing research on even faster algorithms for the 2D DCT, e.g., [22].

Discrete sine transform

There are also multiple types of the **discrete sine transform** (DST). One such version, called DST-IV [\[wiki\]](#), is used in recent image compression methods that work block-wise from upper left to bottom right. This version is (nearly) zero along the left and upper boundaries (except for the DC component) in 2D, which is useful for predictive coding methods.

Properties of the 2D DCT

linearity, separability

symmetry: $x^*[m, n] \xleftrightarrow{\text{DCT}} C_x^*[k, l]$

If $x[m, n]$ is real, then $C_x(k, l)$ is real.

Parseval: $\sum_{m=0}^{M-1} \sum_{n=0}^{N-1} |x[m, n]|^2 = \frac{1}{4MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} w[k] w[l] |C_x[k, l]|^2$

Most of these theoretical properties are of relatively little practical importance, except for the realness property.

The important property in practice is the empirical observation that the DCT tends to yield better energy compaction than the DFT for typical signals.

The DCT is used in the JPEG image compression standard. [\[wiki\]](#)

7.9 Summary

- The major topics in this chapter were the 2D DFS, the 2D DFT, the DCT and 2D DCT, and the 2D FFT.
- The 2D DFT corresponds to samples of the DSFT which can lead to spatial aliasing except for finite support signals.
- The DFT and 2D DFT are finite linear transforms that therefore have an orthogonal matrix representation.
- We can relate the DFT values $G[k, l]$ to the spectrum $G_a(\nu_x, \nu_y)$ of a sampled analog signal (usually approximately and rarely exactly).
- Sampling theory is also a useful guide for using FFTs to approximate FT integrals.
- Except for separability and notation, the 1D and 2D cases have all the same concepts.

Note: FFT can be used for a form of up-sampling as described in Ch. 8.

What part of this chapter did you find most confusing? ??

[RQ]

List the name(s) of your other project team members. ??

[RQ]

Bibliography

- [1] J. S. Lim. *Two-dimensional signal and image processing*. New York: Prentice-Hall, 1990. ISBN: 978-0139353222.
- [2] A. K. Jain. *Fundamentals of digital image processing*. New Jersey: Prentice-Hall, 1989. ISBN: 978-0133361650.
- [3] R. Robucci, J. D. Gray, L. K. Chiu, J. Romberg, and P. Hasler. “Compressive sensing on a CMOS separable-transform image sensor”. In: *Proc. IEEE* 98.6 (June 2010), 1089–1101.
- [4] S. P. Fard and Z. Zainuddin. “Toroidal approximate identity neural networks are universal approximators”. In: *Neural Information Processing*. LNCS 8834. 2014, 135–142.
- [5] O. Tüzünalp. “Generalized time space approach and convolution arrays”. In: *Underwater Acoustics and Signal Processing, Volume 66 of the series NATO Advanced Study Institutes Series*. 1981, 441–54.
- [6] J. Proakis and D. Manolakis. *Digital signal processing: Principles, algorithms and applications*. New York: Prentice-Hall, 1996.
- [7] D. L. Donoho. “De-noising by soft-thresholding”. In: *IEEE Trans. Info. Theory* 41.3 (May 1995), 613–27.
- [8] H. R. Wu and J. Paoloni. “The structure of vector radix fast Fourier transforms”. In: *IEEE Trans. Acoust. Sp. Sig. Proc.* 37.9 (Sept. 1989), 1415–24.
- [9] A. Dutt and V. Rokhlin. “Fast Fourier transforms for nonequispaced data”. In: *SIAM J. Sci. Comp.* 14.6 (Nov. 1993), 1368–93.
- [10] C. Anderson and M. D. Dahleh. “Rapid computation of the discrete Fourier transform”. In: *SIAM J. Sci. Comp.* 17.4 (July 1996), 913–9.
- [11] G. Beylkin. “On the fast Fourier transform of functions with singularities”. In: *Applied and Computational Harmonic Analysis* 2.4 (Oct. 1995), 363–81.
- [12] A. Dutt and V. Rokhlin. “Fast Fourier transforms for nonequispaced data, II”. In: *Applied and Computational Harmonic Analysis* 2.1 (Jan. 1995), 85–100.
- [13] Q. H. Liu and N. Nguyen. “An accurate algorithm for nonuniform fast Fourier transforms (NUFFT’s)”. In: *IEEE Microwave and Guided Wave Letters* 8.1 (Jan. 1998), 18–20.
- [14] Q. H. Liu and X. Y. Tang. “Iterative algorithm for nonuniform inverse fast Fourier transform”. In: *Electronics Letters* 34.20 (Oct. 1998), 1913–4.

- [15] N. Nguyen and Q. H. Liu. “The regular Fourier matrices and nonuniform fast Fourier transforms”. In: *SIAM J. Sci. Comp.* 21.1 (1999), 283–93.
- [16] G. Steidl. “A note on the fast Fourier transforms for nonequispaced grids”. In: *Adv. in Comp. Math.* 9.3 (Nov. 1998), 337–52.
- [17] A. F. Ware. “Fast approximate Fourier transforms for irregularly spaced data”. In: *SIAM Review* 40.4 (Dec. 1998), 838–56.
- [18] J. A. Fessler and B. P. Sutton. “Nonuniform fast Fourier transforms using min-max interpolation”. In: *IEEE Trans. Sig. Proc.* 51.2 (Feb. 2003), 560–74.
- [19] M. Jacob. “Optimized least-square nonuniform fast Fourier transform”. In: *IEEE Trans. Sig. Proc.* 57.6 (June 2009), 2165–77.
- [20] Z. Yang and M. Jacob. “Mean square optimal NUFFT approximation for efficient non-Cartesian MRI reconstruction”. In: *J. Mag. Res.* 242 (May 2014), 126–35.
- [21] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical recipes in C*. New York: Cambridge Univ. Press, 1988.
- [22] G. Bi, G. Li, K. K. Ma, and T. C. Tan. “On the computation of the two-dimensional DCT”. In: *IEEE Trans. Sig. Proc.* 48.4 (Apr. 2000), p. 1171.

Chapter 8

2D Interpolation

Contents (class version)

| | |
|---|-------------|
| 8.0 Introduction | 8.3 |
| Problem statement and background | 8.4 |
| Ideal sinc interpolation | 8.5 |
| 8.1 Polynomial interpolation | 8.7 |
| Zero-order or nearest neighbor (NN) interpolation | 8.7 |
| 1D linear interpolation | 8.11 |
| 2D linear or bilinear interpolation | 8.12 |
| Interpolator properties | 8.20 |
| Lagrange interpolation | 8.24 |
| Quadratic interpolation | 8.27 |
| Cubic interpolation | 8.29 |
| Non-polynomial interpolation: Lanczos kernel | 8.33 |
| 8.2 Interpolation in shift-invariant subspaces | 8.34 |
| Basis kernels supported on $(-2,2)$ | 8.40 |
| B-spline interpolation | 8.48 |
| 2D interpolation in shift-invariant spaces | 8.55 |
| 8.3 Applications | 8.58 |
| Image registration | 8.58 |

| | |
|---|-------------|
| Image zooming (up-sampling) using the FFT | 8.66 |
| 8.4 Motion estimation for video temporal interpolation | 8.71 |
| Translation motion estimation | 8.74 |
| Region-matching methods | 8.76 |
| Space-time constraint methods | 8.80 |
| Region selection | 8.83 |
| Advanced topics | 8.84 |
| Motion compensated temporal interpolation | 8.85 |
| Application of motion estimation methods to spatial interpolation | 8.87 |
| 8.5 Summary | 8.87 |
| Cubic Hermite interpolation | 8.88 |

8.0 Introduction

This chapter focuses on practical **2D interpolation** methods. Image interpolation is very important and used widely. The ideas generalize readily to 3D interpolation problems.

We also consider motion estimation and video interpolation. In the context of video, *i.e.*, temporal image sequences, temporal interpolation can also be important. Even though video is “2D+time” which is a kind of “3D problem,” we use techniques that differ from 3D spatial interpolation problems. We focus initially on spatial interpolation.

We focus on interpolation given equally spaced samples, because those arise most often in DSP and image processing problems. But there are also many applications that require interpolation from unequally spaced data and there are methods for those problems too [1, 2]. The MATLAB command `griddata` is one of those methods. In 1D, one classic method is **spline interpolation**.

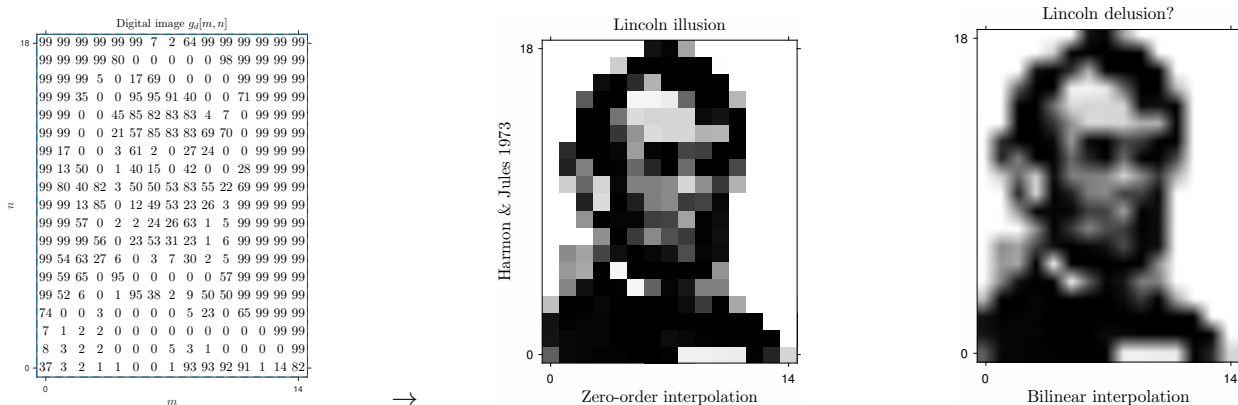
Nonlinear **interpolation** methods, *e.g.*, “morphological” methods have been proposed [3], but we focus on linear methods.

For an excellent historical overview of **interpolation**, see [4]. For a recent fun use of interpolation (and other image processing steps) see [this article about time-lapse flickr mining](#).

Interpolation for images (2D) seems much more prevalent than for typical 1D signals like audio. Why? Perhaps because there is often a mismatch between the array size of imaging sensors and image displays. Modern mobile phone cameras have far more pixels than mobile phone displays.

Problem statement and background

The image **interpolation** problem is the following. We are given a discrete-space image $g_d[m, n]$ that we assume corresponds to samples of some continuous space image $g_a(x, y)$. Typically we assume the samples are ideal samples taken on a rectilinear grid, i.e., $g_d[m, n] = g_a(m\Delta_x, n\Delta_y)$. The usual goal is to compute values of the CS image $g_a(x, y)$ at arbitrary (x, y) locations.



There are two distinct situations where this type of operation is important.

- **D/A conversion**, such as a video monitor. Here we really want to generate a CS image $g_a(x, y)$.
- **Resampling** situations where we want to evaluate $g_a(x, y)$ at a discrete set of (x, y) coordinates that are *different* from the original sampling coordinates $(m\Delta_x, n\Delta_y)$.

Why? For image zooming, display, rotating, warping, coding, motion estimation, ...

JAVA demonstration of the importance of good interpolation for image rotation:

<http://bigwww.epfl.ch/demo/jrotation/start.php>

Generally speaking, interpolation is an example of an **ill-posed** inverse problem. We are given only a finite or countable collection of values $\{g_d[m, n]\}$, yet we hope to recover $g_a(x, y)$, a continuous-space function of uncountable arguments $(x, y) \in \mathbb{R}^2$. There is an uncountably infinite collection of functions $g_a(x, y)$ that agree with $g_d[m, n]$ at the sample points. (Connecting the dots is not unique!) The only way for us to choose among the many possibilities is to make *assumptions* about the original CS signal $g_a(x, y)$. Different interpolation methods are based on different assumptions or **models**.

Ideal sinc interpolation

If $g_d[m, n] = g_a(m\Delta_x, n\Delta_y)$, and if we assume that $g_a(x, y)$ is **band-limited** and the sampling rates are appropriate, and the sampling has infinite extent, then in principle we can recover $g_a(x, y)$ from $g_d[m, n]$ exactly by the **sinc interpolation** formula:

$$g_a(x, y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g_d[m, n] h(x/\Delta_x - m, y/\Delta_y - n), \quad (8.1)$$

cf. (4.9), where

$$h(x, y) = \text{sinc}_2(x, y) = \text{sinc}(x) \text{sinc}(y).$$

This theoretical result is impractical because the **sinc function** (aka the **sine cardinal** function) has unbounded support and the summations require infinitely many samples. Furthermore, real world images need not be exactly band-limited.

Practical interpolation

In practice we always have a finite amount of data, so we replace (8.1) by a practical method such as a general **linear interpolator**:

$$\hat{g}_a(x, y) \triangleq \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} g_a[m, n] h(x/\Delta_x - m, y/\Delta_y - n). \quad (8.2)$$

Rarely will $\hat{g}_a(x, y)$ exactly match the true $g_a(x, y)$ except *perhaps* at the sample locations (in the absence of noise etc.).

We simply *hope* that $\hat{g}_a(x, y) \approx g_a(x, y)$, at least for $0 \leq x \leq (M-1)\Delta_x$ and $0 \leq y \leq (N-1)\Delta_y$.

Different linear interpolation methods correspond to different choices for the **interpolation kernel** $h(x, y)$.

Note that any interpolator of the form (8.1) or (8.2) is **linear** in the sense that it is a linear function of the *samples* $\{g_a[m, n]\}$.

Usually we choose interpolation kernels that are **separable**, with the same 1D kernel shape in both x and y , *i.e.*,

$$h(x, y) = h(x) h(y).$$

So we focus hereafter primarily on choices for the 1D interpolation kernel $h(x)$, often with unit spacing $\Delta = 1$ for simplicity.

Caution. If the units of the argument x in the image $g_a(x, y)$ are meters (m), what are the units of the argument of the interpolation kernel $h(\cdot, \cdot)$? ?? [RQ]

8.1 Polynomial interpolation

We first consider **polynomial interpolators**, *i.e.*, those that have kernels that are **piecewise polynomials**.

MATLAB's `interp1` and `interp2` commands support several such options.

Zero-order or nearest neighbor (NN) interpolation

The simplest form of interpolation is called **nearest neighbor (NN)** or **zero-order** interpolation, and has the following recipe: *use the value at the nearest sample location*.

Those words give a *procedural* description of the interpolator, but we also want to have a *mathematical* description (and efficient software). A mathematical expression for this approach in 1D is:

$$\hat{g}_a(x) = g_d[\text{round}(x/\Delta_x)] = g_d[n] \Big|_{n=\text{round}(x/\Delta_x)}. \quad (8.3)$$

One can verify that the interpolator (8.3) is **integer shift invariant** in the following sense. If $f[n] = g_d[n - n_0]$, for an integer shift $n_0 \in \mathbb{Z}$, and we interpolate $f[n]$ using the same nearest neighbor rule (8.3) to obtain $\hat{f}_a(x)$, then $\hat{f}_a(x) = \hat{g}_a(x - n_0\Delta_x)$. So shifting $g_d[n]$ by n_0 samples causes the resulting interpolated signal to be shifted by $n_0\Delta_x$. Note that the two shifts have different units so this type of “shift invariance” has a different meaning than in earlier chapters!

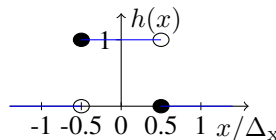
Because this 1D interpolator is integer shift invariant, we can also write it in a form like (8.1):

$$\hat{g}_a(x) = \sum_{n=-\infty}^{\infty} g_d[n] h(x/\Delta_x - n) \quad (8.4)$$

for some 1D interpolation kernel $h(x)$. When we are given a “recipe” like “use the nearest neighbor,” how can we determine that kernel? From (8.4) we see that, to answer this question, it suffices to use a Kronecker impulse $g_d[n] = \delta[n]$ as the input, and

use $\Delta_x = 1$, because $h(x) = \sum_{n=-\infty}^{\infty} \delta[n] h(x - n)$. So interpolating a **Kronecker impulse** yields the **interpolation kernel** for **integer shift invariant** interpolation methods. For 1D NN interpolation:

$$h(x) = \delta[\text{round}(x)] = \begin{cases} 1, & -1/2 \leq x < 1/2 \\ 0, & \text{otherwise.} \end{cases} \quad (8.5)$$



This formula assumes 0.5 is rounded up to 1, which is how the MATLAB and C `round` function works. Unfortunately, MATLAB and C `round` function rounds *away* from 0 which is not quite integer shift invariant for $x \leq 1/2$. To avoid this problem, replace `round(x)` with `floor(x+0.5)`.

These technicalities about what to do when $x/\Delta_x = n + 0.5$ for $n \in \mathbb{Z}$ stem from the ambiguity of the original recipe. There is not a unique “nearest neighbor” at these midway points! A more precise recipe is: “use the nearest neighbor, except at points that are midway between two samples use the sample to the right.” This more precisely stated interpolator has the mathematical form (8.4) with interpolation kernel (8.5). We often write the interpolation kernel for NN interpolation as a rect function:

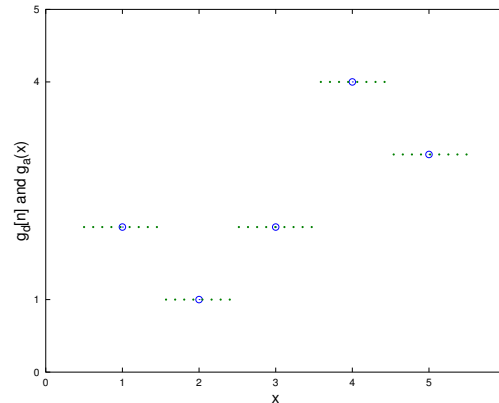
$$h(x) = \text{rect}(x) = \begin{cases} 1, & -1/2 \leq x < 1/2 \\ 0, & \text{otherwise.} \end{cases} \quad (8.6)$$

In previous chapters, the definition of the rect function at $x = \pm 1/2$ was unimportant. But for interpolation it is important and the appropriate definition is given in (8.5) for the “rounding up” convention.

Exercise. Determine $h(x)$ for the modified NN interpolator where at the midway points one uses the *average* of the two nearest neighbors. (HW)

Although the general form (8.4) and interpolation kernel (8.5) are convenient for mathematical analysis, the software implementation of 1D NN interpolation looks far more like the “recipe” version (8.3).

```
gd = [2 1 2 4 3]; % signal sample values
N = length(gd);
n = 1:N;
x = linspace(0.5,N+0.49,7*(N+1)+1);
dx = 1;
index = round(x / dx); % 1 .. N hopefully
gi = gd(index);
plot(1:N, gd, 'o', x, gi, '.')
```



Exercise. How to modify code to handle values of x outside the range of the sample locations, *i.e.*, per (8.2)? (do in class / pair)

??

2D NN interpolation

In 2D, using the NN interpolator (with the same “rounding up” convention as in 1D) can be written as

$$g_a(x, y) = g_d[\text{round}(x/\Delta_x), \text{round}(y/\Delta_y)].$$

Within the region that is sampled, this type of interpolation is equivalent to using (8.2) with the following kernel:

[RQ]

$$h(x) = \text{rect}(x) \implies h(x, y) = \boxed{??}$$

Again, in a practical implementation with a finite number of samples, one must handle (x, y) locations that are outside of the sample locations appropriately. The “nearest neighbor” location can be much less “near” for such samples!

End conditions

For a separable interpolation kernel $h(x, y) = h(x)h(y)$ where the 1D kernel $h(x)$ is symmetric with half-width equal to w , the expression (8.2) evaluates to zero for (x, y) locations not within the support rectangle

$$[-w\Delta_x, (M - 1 + w)\Delta_x, -w\Delta_y, (N - 1 + w)\Delta_y].$$

Often we want to **extrapolate** the border values instead of using zero.

MATLAB's `interp1` and `interp2` commands offer options to control the extrapolation method.

Exercise. Compare the MATLAB output for the following `interp1` options.

```
interp1([0 1], [10 20], [-1 0 1 2])
interp1([0 1], [10 20], [-1 0 1 2], 'nearest')
interp1([0 1], [10 20], [-1 0 1 2], 'nearest', 0)
interp1([0 1], [10 20], [-1 0 1 2], 'nearest', 'extrap')
```

How would we write the last option mathematically?

??

(do in class)

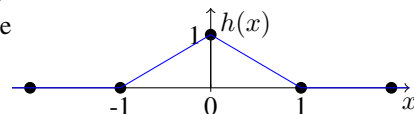
1D linear interpolation

The **linear interpolation** method is the second simplest method. As a recipe, in 1D we interpolate by connecting the two nearest points by a line. This interpolator literally “connects the dots” with line segments. It assumes (implicitly) that $g_a(x)$ is **piecewise linear**. A mathematical formula for this recipe is

$$\hat{g}_a(x) = (n_x + 1 - x/\Delta_x) g_d[n_x] + (x/\Delta_x - n_x) g_d[n_x + 1], \quad n_x \triangleq \lfloor x/\Delta_x \rfloor. \quad (8.7)$$

This interpolation procedure is also **integer shift invariant** so we can determine the corresponding **interpolation kernel** by interpolating the Kronecker impulse $g_d[n] = \delta[n]$. The resulting 1D interpolation kernel is a triangular function:

$$h(x) = \text{tri}(x).$$

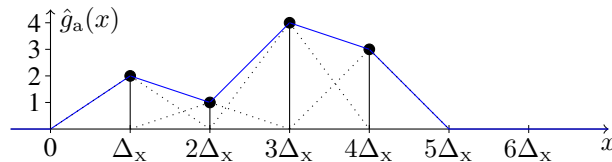
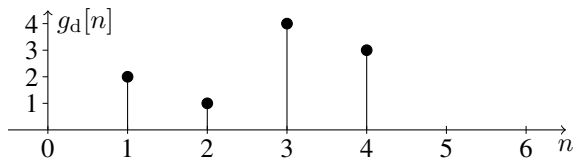


Thus a mathematical expression for linear interpolation is

$$\hat{g}_a(x) = \sum_n g_d[n] \text{tri}\left(\frac{x}{\Delta_x} - n\right). \quad (8.8)$$

This expression is convenient for mathematical analysis, but the software implementation usually looks more like (8.7) than (8.8).

Example.

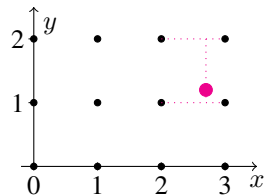


Here there is an additional meaning of the word **linear** because the tri function is a piecewise linear function of its (spatial) argument. The more general meaning of **linear** is that the interpolated signal $\hat{g}_a(x)$ is a linear function of the samples $g_d[n]$.

2D linear or bilinear interpolation

For **bilinear interpolation** of 2D samples, roughly speaking one first does **linear interpolation** along x , and then linear interpolation along y , (or vice versa.)

More precisely, the procedure is to find the four nearest sample locations, interpolate the top and bottom pair along x , and then interpolate the resulting two points along y . This figure illustrates the process graphically for the case $\Delta_x = \Delta_y = 1$.



Clearly this process is (integer) shift invariant, so we can determine its interpolation kernel by interpolating the 2D Kronecker impulse $\delta_2[m, n]$. For a point (x, y) that is within the unit square $[0, 1] \times [0, 1]$, one can verify that the interpolation kernel is $(1-x)(1-y)$. Considering the symmetries of the problem, for locations (x, y) within the square $[-1, 1] \times [-1, 1]$ the interpolation kernel is $(1-|x|)(1-|y|)$. Otherwise the kernel is zero, *i.e.*,

$$h(x, y) = \begin{cases} (1-|x|)(1-|y|), & |x| < 1, |y| < 1 \\ 0, & \text{otherwise.} \end{cases}$$

In other words, bilinear interpolation is equivalent to using (8.1) or (8.2) with the following interpolation kernel:

[RQ]

$$h(x, y) = \boxed{??}$$

An alternative way to view bilinear interpolation is as follows. For any point (x, y) , we find the four nearest sample points, and fit to those four points a 2D polynomial of the form

$$\alpha_0 + \alpha_1 x + \alpha_2 y + \alpha_3 xy. \quad (8.9)$$

Then we evaluate that polynomial at the desired (x, y) location.

Bilinear interpolation is so important that modern GPUs can perform it in hardware, meaning that the time for a bilinear interpolation is comparable to the time required for loading a value from memory, using **texture memory** [5].

Example. Consider a point (x, y) that is within the unit square $[0, 1] \times [0, 1]$. Then the four nearest samples are $g_d[0, 0]$, $g_d[1, 0]$, $g_d[0, 1]$, $g_d[1, 1]$. To fit the polynomial we set up 4 equations in 4 unknowns:

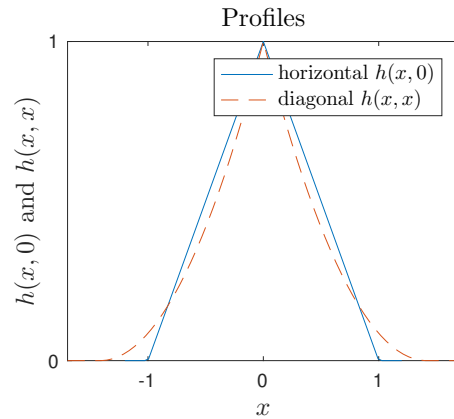
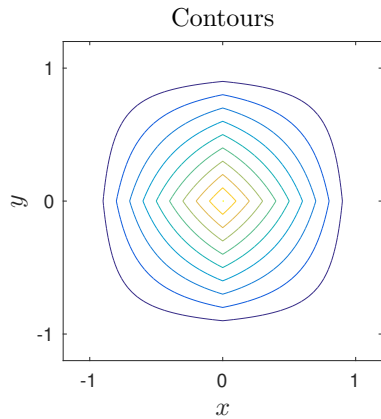
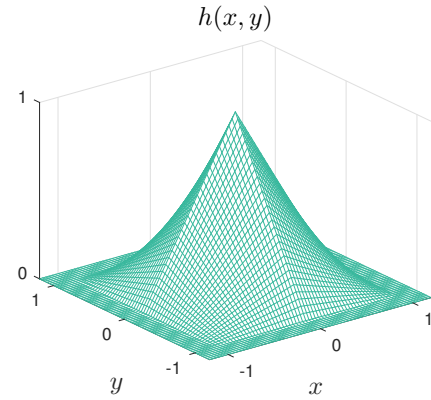
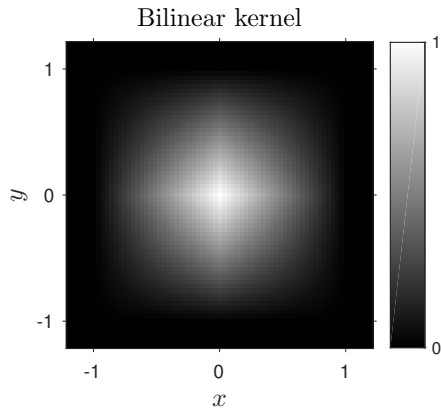
$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{\begin{bmatrix} 1 & x & y & xy \end{bmatrix}} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} g_d[0, 0] \\ g_d[1, 0] \\ g_d[0, 1] \\ g_d[1, 1] \end{bmatrix}.$$

Solving for the coefficients and substituting into the polynomial and simplifying yields

$$g_d[0, 0](1-x)(1-y) + g_d[1, 0]x(1-y) + g_d[0, 1](1-x)y + g_d[1, 1]xy.$$

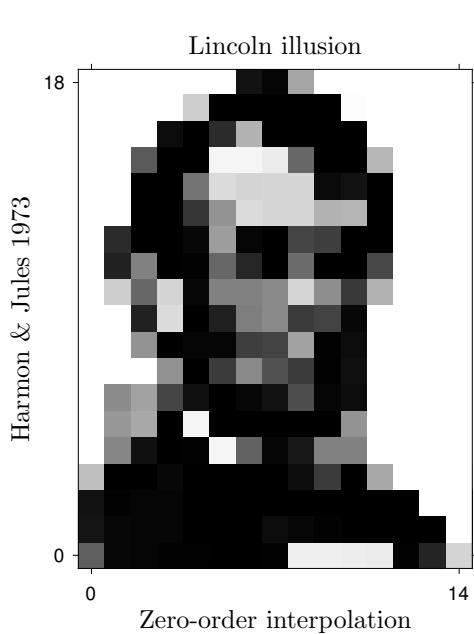
Considering the symmetries of the problem, for locations (x, y) within the square $[-1, 1] \times [-1, 1]$ the $g_d[0, 0]$ value will in general be multiplied by $(1-|x|)(1-|y|) \text{rect}(x/2) \text{rect}(y/2) = \text{tri}(x) \text{tri}(y)$. (Because the method is invariant to integer shifts, it suffices to find the expression for the $g_d[0, 0]$ term.)

The following figure shows the (separable) bilinear interpolation kernel and its contours and profiles. Note that the profile along the diagonal of the kernel is *not* a piecewise linear function!



Example. Here is an example of **nearest neighbor** and **bilinear** interpolation of a famous image.

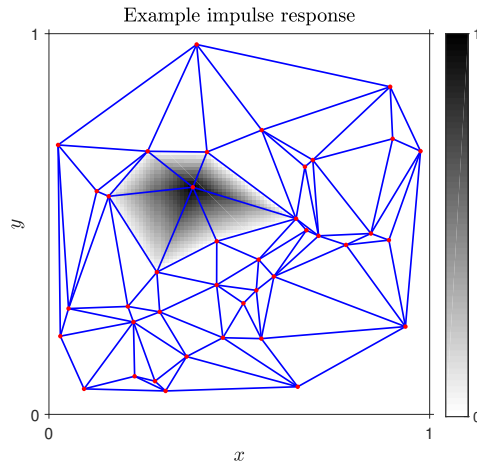
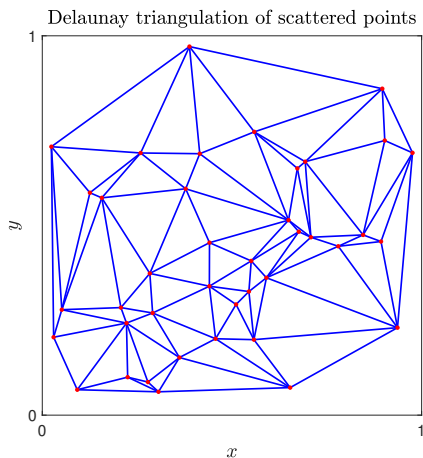
The “pyramid” shape of the interpolation kernel is evident in the right image, and perhaps less desirable than the smoother interpolators discussed later.



Delaunay triangulation and scattered data interpolation

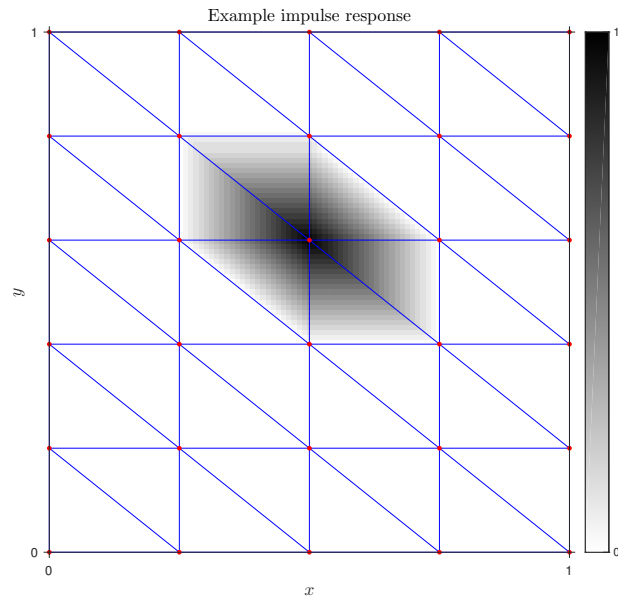
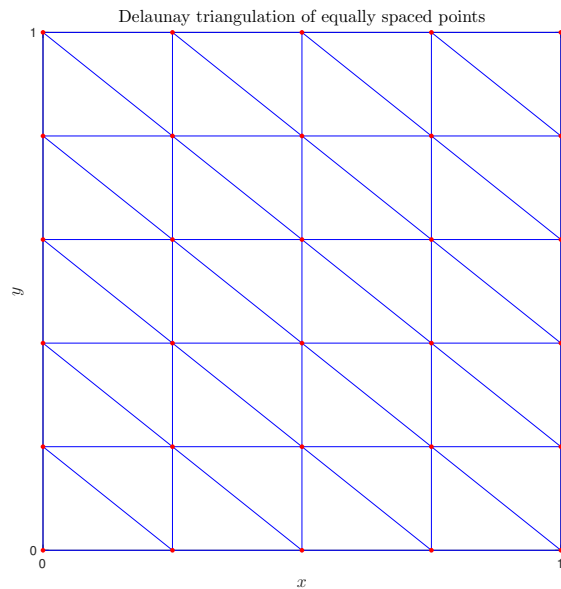
An alternative interpolation approach that is strictly **piecewise-linear** is to first use **Delaunay triangulation** of the sample locations (this is applicable even to non-rectilinear samples) and then use a linear function (a plane) within each triangle (because three points determine a plane). This is how MATLAB's `griddata` works with the (default) `'linear'` option.

The left figure illustrates scattered (x_i, y_i) locations that have been **triangulated** using MATLAB's `delaunay` command. Given data values $\{f(x_i, y_i)\}$ at each point location, the `griddata` command interpolates those. The right figure illustrates the “impulse response” of the `griddata` interpolator, where all data values are zero except one. The interpolated function is piecewise linear.

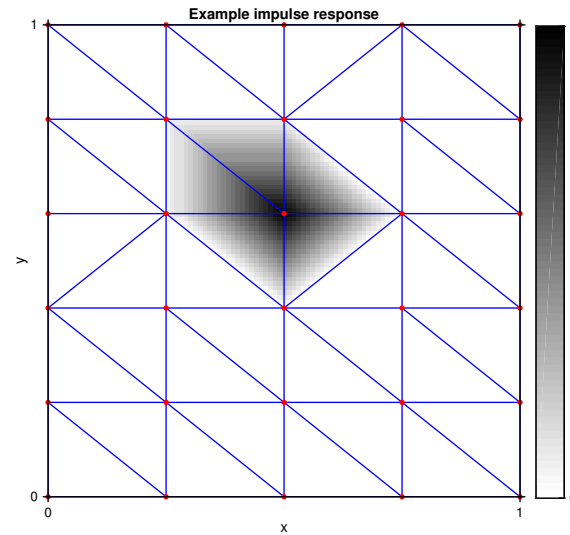
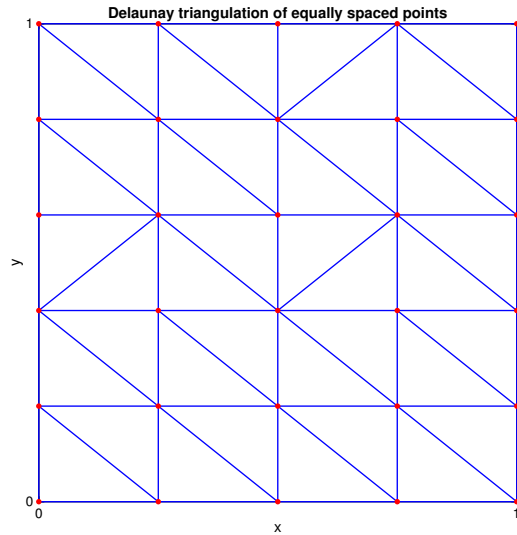


Scattered data interpolation arises in many applications, but is less relevant to image processing because imaging sensors usually provide equally spaced data.

For equally spaced data, Delaunay triangulation and `griddata` is inappropriate. The left figure shows the **Delaunay triangulation** of an equally spaced grid of data points. Triangulation is not unique for such points. The right figure shows an example impulse response, for input data $\delta_2[m-2, n-3]$. The lack of even symmetry is undesirable, and is a reason *not* to use `griddata` for Cartesian sampled data.



Here are the results from an older version of MATLAB's `griddata` that had a clear lack of shift invariance that was undesirable.

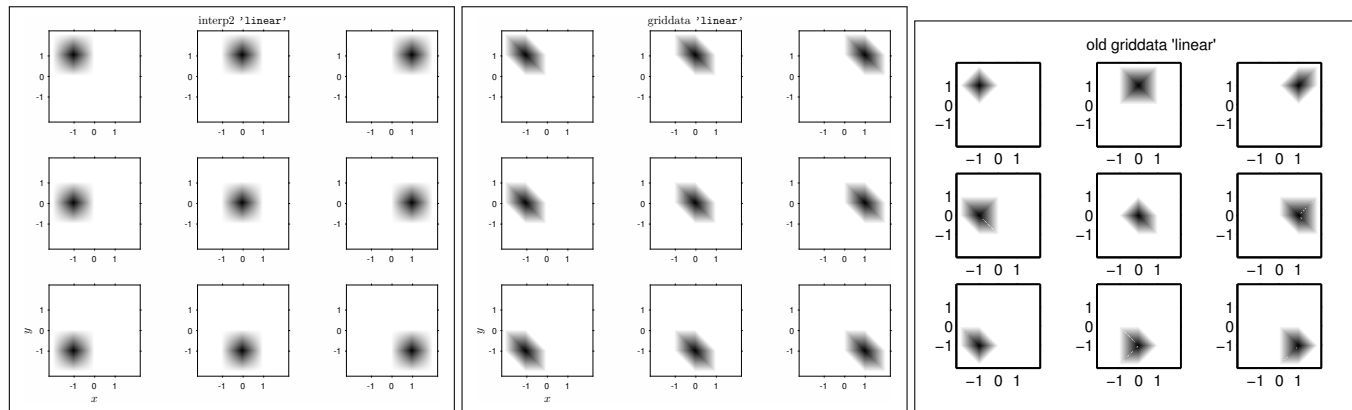


MATLAB's `griddata` also offers a **natural neighbor** option. (I have not studied whether it is shift invariant and symmetric.)

The following figures show the result of interpolating $\delta_2[m - m_0, n - n_0]$ for 9 choices of (m_0, n_0) locations.

The `interp2` bilinear method is invariant to integer shifts, which is called **shift invariant** in the context of interpolation methods, and the interpolation kernel has even symmetry. In contrast, the `griddata` approach lacks even symmetry due to the triangulation.

An older version of `griddata` (right group of figures) even lacked shift invariance.



The main benefit of `griddata` is that it can handle nonuniformly sampled data. As a secondary point, `griddata` results are **piecewise linear** functions, whereas these `interp2` results are piecewise polynomials of the form (8.9), but this does not seem like a compelling benefit.

The bottom line is that `griddata` is not a good method for interpolating equally spaced data! (It is fine for scattered data.)

Interpolator properties

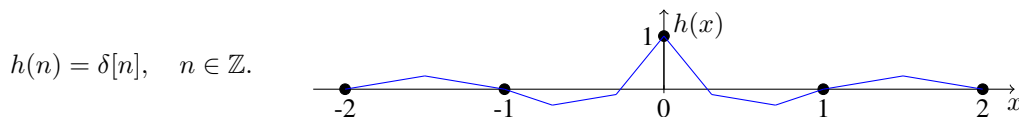
The preceding interpolators are used quite frequently.

Why should we need anything else? One reason is that neither the zero-order or bilinear interpolator are differentiable, a property that is desirable for edge detection (Ch. 9) and image registration.

Before proceeding, it seems reasonable to assess desirable properties of interpolators.

In general, desirable properties of interpolation kernels include the following. (See [6] for an excellent modern treatment.)

- $h(0) = 1$ and $h(n) = 0$ for $n = \pm 1, \pm 2, \dots$, *i.e.*,



This property ensures the **interpolation property** holds, *i.e.*, that the following “self consistency” holds:

$$\hat{g}_a(x) \Big|_{x=n\Delta} = \sum_{m=-\infty}^{\infty} g_a(m\Delta) h(x/\Delta - m) \Big|_{x=n\Delta} = \sum_{m=-\infty}^{\infty} g_a(m\Delta) \delta[n - m] = g_a(n\Delta) = g_d[n]. \quad (8.10)$$

- Continuous and **smooth** (differentiable up to some order), particularly when derivatives are of interest:

$$\frac{d}{dx} \hat{g}_a(x) = \sum_{n=-\infty}^{\infty} g_d[n] \frac{d}{dx} (h(x/\Delta - n)).$$

- Short spatial extent (**support**) to minimize computation. If $\mathcal{S} \subset \mathbb{R}$ denotes the support of $h(x)$, then (**Picture**)

$$\hat{g}_a(x) = \sum_{n=-\infty}^{\infty} g_d[n] h(x/\Delta - n) = \sum_{\{n \in \mathbb{Z} : x - n\Delta \in \mathcal{S}\}} g_d[n] h(x/\Delta - n).$$

- **Frequency response** approximately $H(\nu) \approx \text{rect}(\nu)$.
- **Symmetric**. (To ensure invariance to mirror images.)
- Small **side lobes** to avoid ringing “artifacts”
- $\sum_n h(x - n) = 1, \forall x \in \mathbb{R}$. (Perfect DC interpolation, also called **partition of unity**.)
- More generally, we might want to be able to interpolate (sampled) monomials x^m perfectly, *i.e.*, if we sample a monomial and then interpolate the samples, we hope to get back the original monomial perfectly. Note that monomials are *not* band-limited so perfect monomial interpolation requires a non-sinc interpolation kernel.

If $h(x)$ perfectly interpolates monomials up to x^{L-1} , then L is called the **approximation order** of the interpolator [6]. The reason for calling it “ L ” instead of “ $L - 1$ ” is that for an interpolator having approximation order L , the approximation error decays as Δ^L :

$$\|\hat{g} - g\|_{\mathcal{L}_2} = C_h \Delta^L \left\| g^{(L)} \right\|_{\mathcal{L}_2} \text{ as } \Delta \rightarrow 0$$

for some constant C_h that depends on $h(x)$, where $\|g\|_{\mathcal{L}_2}^2 = \int |g(x)|^2 dx$ and $g^{(L)}$ denotes the L th derivative of $g(x)$.

If an interpolator with kernel $h(x)$ has approximation order L , then sampling x^{L-1} and interpolating it returns the original function, *i.e.*,

$$x^{L-1} = \sum_{n=-\infty}^{\infty} n^{L-1} h(x - n), \quad \forall x \in \mathbb{R}. \quad (8.11)$$

If $h(x)$ satisfies the **partition of unity**, then what can we say about its approximation order? ?? [RQ]

Example. The linear interpolation kernel $\text{tri}(x)$ can recover affine functions $g_a(x) = a + bx$ from samples perfectly, so its approximation order is $L = 2$.

How would you generalize (8.11) to 2D?

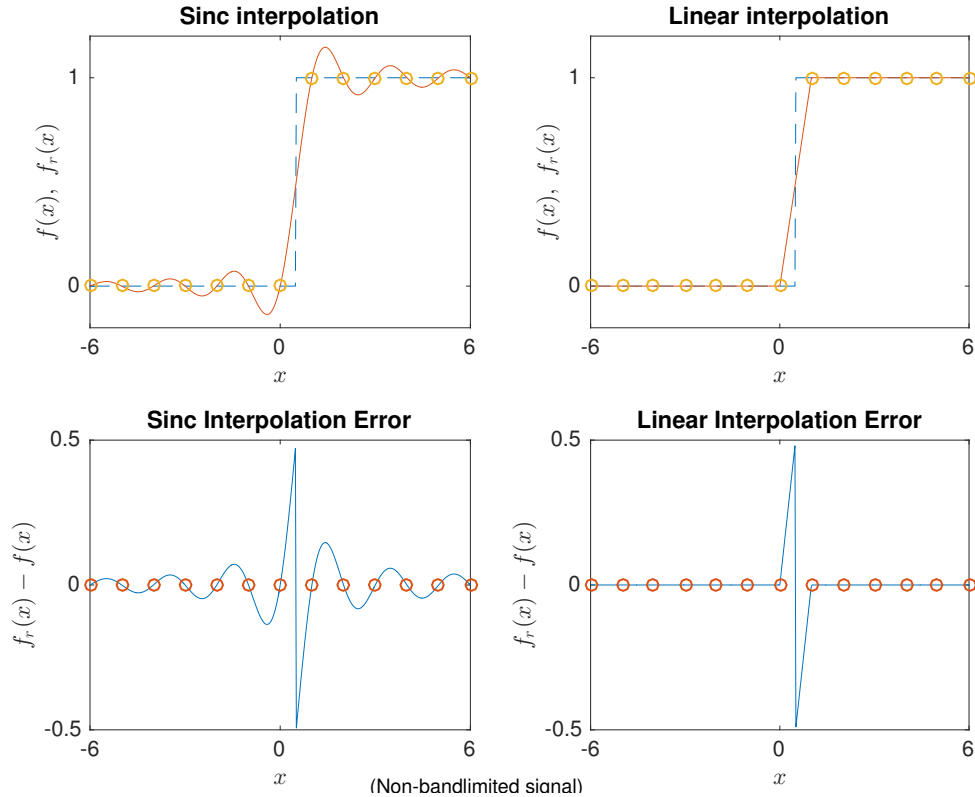
class/team

??

One must compromise between the above desirable properties; they cannot all be achieved simultaneously.

Why is **circular symmetry** missing from this list? The only nontrivial separable function that is circularly symmetric is the 2D gaussian, which does not satisfy the interpolation property.

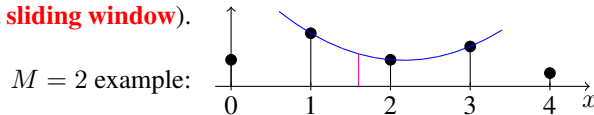
Example. The following figure illustrates why we may prefer interpolators with small **side lobes** in image processing.



Lagrange interpolation

The classical **Lagrange interpolation** method works as follows.

- For any given value of x , choose the $M + 1$ nearest sample locations (a **sliding window**).
- Fit a M th-degree polynomial to those nearest $M + 1$ points.
- Evaluate fitted polynomial at desired location(s).



Remarks.

- See <http://mathworld.wolfram.com/LagrangeInterpolatingPolynomial.html>
- This method is used fairly frequently.
- Equivalent to linear interpolation in 1D for $M = 1$.
- One can smooth noise by using more points than the polynomial degree and performing a least-squares fit.

Does this method have the form (8.1)? In other words, in 1D can we write: $\hat{g}_a(x) = \sum_n g_d[n] h(x/\Delta_x - n)$.

Yes: the method is a **linear** function of the data values and is (integer) **shift invariant**.

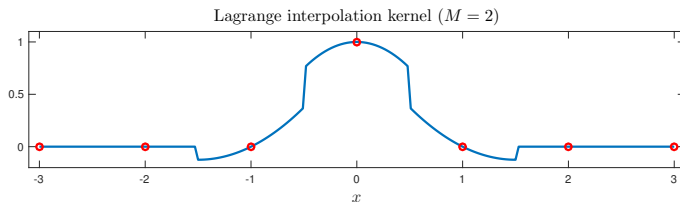
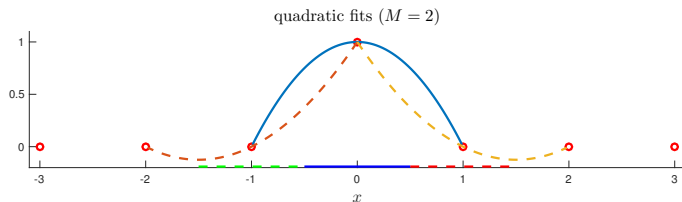
Knowing that the method is linear and (integer) shift invariant, we can determine the corresponding kernel $h(x)$ by “interpolating” a Kronecker impulse signal: $g_d[n] = \delta[n]$ for the case $\Delta_x = 1$ because in this situation $\hat{g}_a(x) = h(x)$.

Lagrange interpolation for $M = 2$

Example. Consider the case $M = 2$. We will determine $h(x)$ by interpolating $\delta[n]$.

- If $|x| \leq 1/2$ then the $M + 1 = 3$ nearest sample points are $\{-1, 0, 1\}$, and the corresponding 2nd-degree polynomial is $\frac{(x-1)(x+1)}{(0-1)(0+1)} = 1 - x^2$.
- If $1/2 \leq x \leq 3/2$ then the $M + 1 = 3$ nearest sample points are $\{0, 1, 2\}$, and for a Kronecker impulse input the corresponding 2nd-degree polynomial (which has roots at 1 and 2) can be found to be $\frac{(x-1)(x-2)}{(0-1)(0-2)} = (x-1)(x-2)/2$.
- If $3/2 \leq x$ then the $M + 1 = 3$ nearest sample points all have value zero.
- Combining using symmetry, we conclude that the Lagrange interpolation kernel for $M = 2$ is

$$h(x) = \begin{cases} 1 - x^2, & |x| \leq 1/2 \\ (|x-1|)(|x-2|)/2, & 1/2 \leq |x| \leq 3/2 \\ 0, & \text{otherwise.} \end{cases}$$



The discontinuity at $x = \pm 1/2$ seems like a rather undesirable property.

Exercise. Repeat for $M = 3$ and see if the kernel is continuous. Find expression and plot.

(do in class: team)

??

Any undesirable properties? ??

Desirable properties? ??

What is the approximation order of a Lagrange interpolator with some given M ? ??

[RQ]

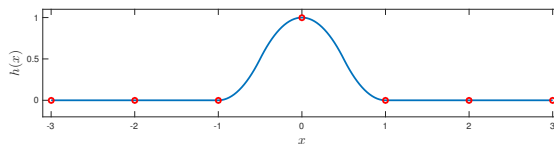
Quadratic interpolation

The Lagrange interpolator for $M = 2$ is **piecewise quadratic**, and it has approximation order 3, meaning that it can interpolate (samples of) polynomials of degree 2 perfectly, but its discontinuity seems quite undesirable.

Other **quadratic interpolation** kernels have been proposed. These seem to be used less than the cubic interpolators described next. Apparently this is in part due to a misperception that quadratic interpolators are nonzero phase [7].

A quadratic kernel on $[-1, 1]$ that is continuous, differentiable, nonnegative, and has a continuous first derivative is:

$$h(x) = \begin{cases} 1 - 2x^2, & |x| \leq 1/2 \\ 2(|x| - 1)^2, & 1/2 < |x| \leq 1 \\ 0, & \text{otherwise.} \end{cases}$$



Putting the “break point” at $x = 1/2$ is a design choice. One could put it anywhere between 0 and 1.

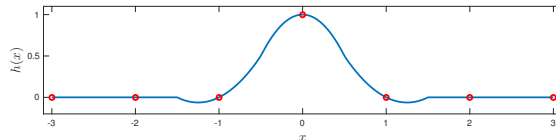
Exercise. Can you think of any advantage to putting the break at $x = 1/2$? ??

Exercise. What is the approximation order of this kernel? (Hint. See figures next page.) ??

[RQ]

Dodgson [7] discusses a different quadratic kernel:

$$h(x) = \begin{cases} 1 - 2x^2, & |x| \leq 1/2 \\ |x|^2 - 5/2|x| + 3/2, & 1/2 < |x| < 3/2 \\ 0, & \text{otherwise.} \end{cases}$$



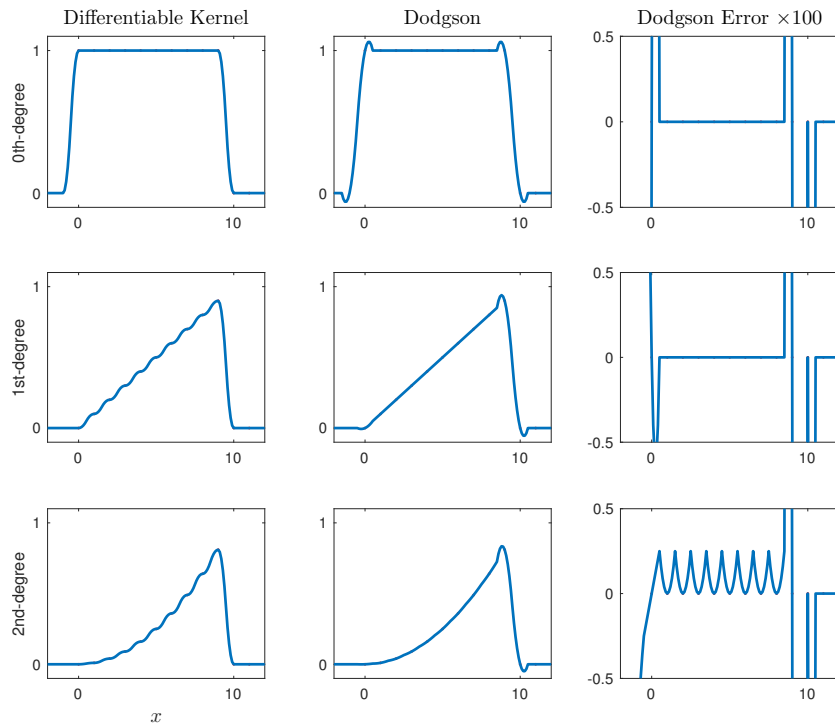
This kernel has support $[-3/2, 3/2]$, which is consistent with the rect, tri, quad, cubic series, but it is not differentiable at $\pm 3/2$.

Exercise. What is the approximation order of this kernel? ??

[RQ]

Example. The following figure illustrates empirically the approximation orders of the two preceding interpolation kernels. The “error” is measured against the ideal monomial, *i.e.*, x^0 , x^1 , x^2 for each respective row.

What tradeoff is illustrated here? [Computation \(support of interpolation kernel\) vs approximation order.](#)



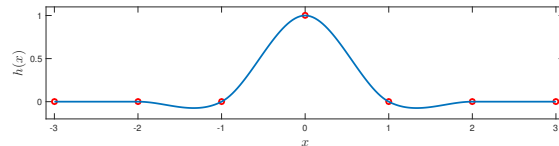
Cubic interpolation

There are several **cubic interpolation** methods in the literature. Some, but not all, have the form (8.2) for some (piecewise) cubic kernel. The typical choices of these kernels are differentiable, which is convenient for taking derivatives.

Care must be taken when using cubic interpolation; there are incorrect formulations in the literature that have led to false conclusions. See [8] [9] for a nice analysis based on **B-splines**. There is a *family* of cubic interpolators described in [10].

MATLAB's `interp2` function has a `'cubic'` option that provides a so-called “cubic convolution” method that is actually cubic interpolation based on a paper by R. Keys [11] that gives the following interpolation kernel:

$$h(x) = \begin{cases} 1 - \frac{5}{2}|x|^2 + \frac{3}{2}|x|^3, & |x| \leq 1 \\ 2 - 4|x| + \frac{5}{2}|x|^2 - \frac{1}{2}|x|^3, & 1 < |x| < 2 \\ 0, & \text{otherwise.} \end{cases} \quad (8.12)$$



The term “cubic convolution” is inappropriate because linear interpolation of the form (8.2) is interpolation, *not* convolution!

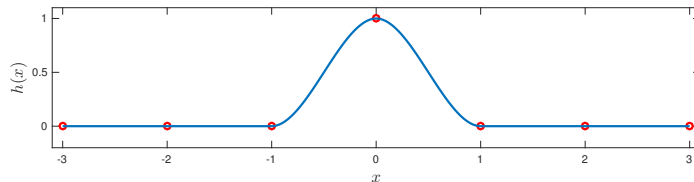
Oddly, to use this kernel with MATLAB's `interp1` function for 1D signals, one must select the `'v5cubic'` option. MATLAB certainly has added to confusion about cubic interpolation by this inconsistency between `interp1` and `interp2` options. The `interp1` documentation in 2017b warns that future versions of `interp1` will use “cubic convolution” for its `'cubic'` option.

To further add confusion, in recent versions of MATLAB, including 2017b, selecting the `'cubic'` option to `interp1` is equivalent to choosing the `'pchip'` option that is named for **piecewise cubic Hermite interpolation**. However, the `'pchip'` option to `interp1` actually corresponds to **monotone cubic Hermite interpolation** [12]. The `'pchip'` method is very different from all the other kernel-based interpolators that we have discussed here. It is in fact a nonlinear method.

As of version R2017b, MATLAB's `imresize` command with the default `'bicubic'` option continues to use the Key's cubic kernel (8.12) [11].

An alternative cubic interpolation kernel having smaller support is:

$$h(x) = \left(1 - 3x^2 + 2|x|^3\right) \operatorname{rect}\left(\frac{x}{2}\right).$$



This is the unique symmetric function that is cubic on $[0,1]$ and satisfies $h(0) = 1$, $\dot{h}(0) = 0$, $h(x) + h(x-1) = 1$ for $x \in [0,1]$.

It also happens to be the output of MATLAB's `interp1` with the `'pchip'` option (currently the same as the default `'cubic'` option) for a Kronecker impulse input.

Its spectrum is

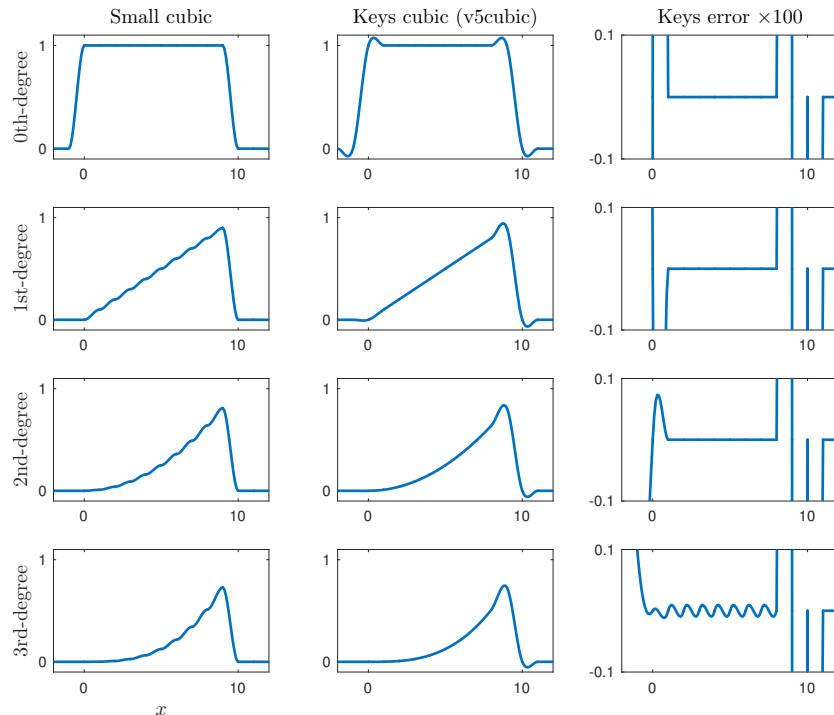
$$\begin{aligned} H(\nu) &= \int_{-1}^1 \left(1 - 3x^2 + 2|x|^3\right) e^{-i2\pi\nu x} dx = 2 \int_0^1 (1 - 3x^2 + 2x^3) \cos(2\pi\nu x) dx \\ &= \begin{cases} \frac{3 - 3\pi\nu \sin(2\pi\nu) - 3 \cos(2\pi\nu)}{2\pi^4\nu^4}, & \nu \neq 0 \\ 1, & \nu = 0. \end{cases} \end{aligned}$$

(The symbolic toolbox in MATLAB is convenient for such integrals.)

Example. This figure shows that the “small cubic” interpolation kernel has approximation order $L = 1$, whereas Keys has approximation order

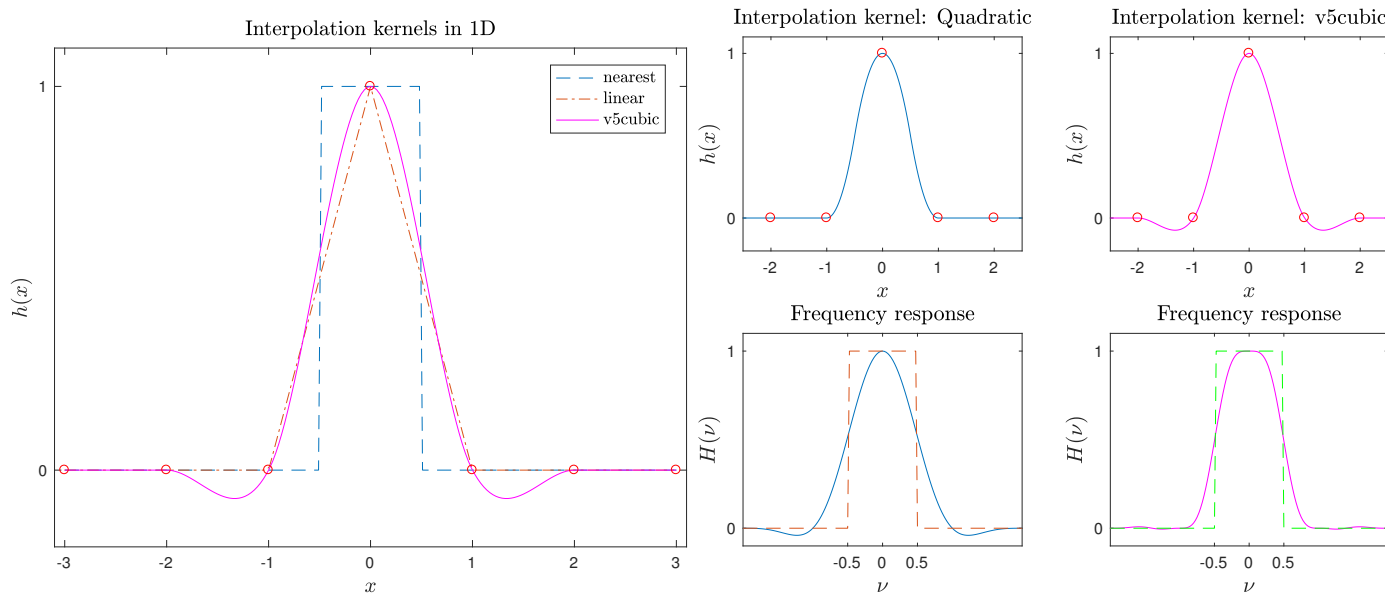
$L = \boxed{??}$

[RQ]



The following figures illustrate that for interpolators based on piecewise polynomials, generally as the polynomial degree increases:

- the interpolator support widens typically, increasing computation,
- the frequency response can better approximate the “ideal” $\text{rect}(\nu)$.



To perform cubic interpolation with GPU texture memory, one can combine two linear interpolation operations [5, 13].

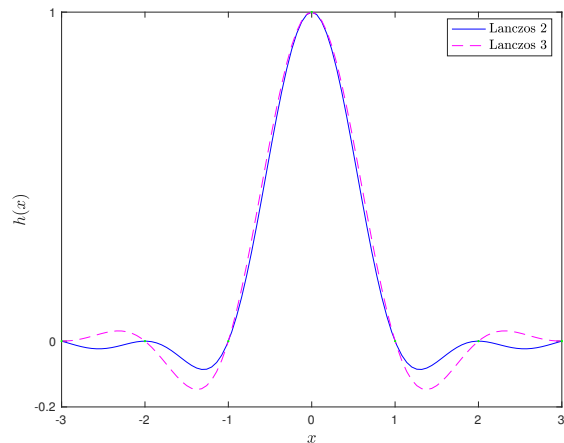
Non-polynomial interpolation: Lanczos kernel

MATLAB's `imresize` function includes two **Lanczos** interpolation kernels [14, p. 156-8] defined as follows.

$$h_2(x) \triangleq \text{sinc}(x) \text{sinc}(x/2) \text{rect}\left(\frac{x}{4}\right), \quad h_3(x) \triangleq \text{sinc}(x) \text{sinc}(x/3) \text{rect}\left(\frac{x}{6}\right).$$

These are both special cases of the general **Lanczos** interpolation kernel of the following form:

$$h_a(x) \triangleq \underbrace{\text{sinc}(x)}_{\text{ideal kernel}} \underbrace{\text{sinc}\left(\frac{x}{a}\right) \text{rect}\left(\frac{x}{2a}\right)}_{\text{central lobe stretched over } [-a, a]}.$$



This kernel is twice differentiable for $a \in \mathbb{N}$. Some investigators have described it as a favorable compromise in terms of reduction of aliasing, sharpness, and minimal ringing. Of course it is somewhat more expensive to compute than a polynomial.

What part of this reading assignment did you find least clear? What part was the most interesting? ??

[RQ]

8.2 Interpolation in shift-invariant subspaces

The “consistency” requirement (8.10) makes sense if we use the signal samples $g_a(m\Delta_x)$ themselves as the coefficients in (8.10), but one can generalize this type of expression by allowing other values to serve as the coefficients. The B-spline interpolators, discussed next, use this remarkable generalization. We focus initially on the 1D case for simplicity.

Thus far we have used the following type of linear interpolation:

$$\hat{g}_a(x) = \sum_n \underbrace{g_d[n]}_{\substack{\text{signal} \\ \text{samples}}} \underbrace{h(x/\Delta_x - n)}_{\substack{\text{interpolation} \\ \text{kernel}}} . \quad (8.13)$$

It is quite feasible to generalize this approach to consider **arbitrary** interpolation coefficients:

$$\hat{g}_a(x) = \sum_n c[n] b(x/\Delta_x - n) = \sum_k c[k] b(x/\Delta_x - k) . \quad (8.14)$$

Clearly we will have to determine the coefficients $c[n]$ from the given samples $g_d[n]$ somehow.

The set of signals $\{b(\cdot - k) : k \in \mathbb{Z}\}$ is not orthogonal in general, but this set will be linearly independent if we choose $b(x)$ appropriately. (Note: the argument x of $b(x)$ is unitless in (8.14) and throughout these notes.)

We now use the notation $b(x)$ to emphasize that $b(x)$ is part of a **basis** and no longer is necessarily the **impulse response**.

The representation (8.14) is a subspace of all possible signals, and such subspaces are called **shift invariant** subspaces. (One must remember that they are invariant only to shifts by *integer* multiples of Δ_x in general.)

Model: $\mathcal{S} = \{g : g(x) = \sum_k c[k] b(x/\Delta_x - k)\}$ has property: $f \in \mathcal{S} \implies f(\cdot - \Delta_x) \in \mathcal{S}$.

With this more general representation, it is not necessary for the **basis kernel** $b(x)$ to satisfy the “sampling conditions” $b(0) = 1$ and $b(n) = 0$ for $n \in \mathbb{Z} - \{0\}$. This opens the door to using many possible kernels.

From samples to basis coefficients

Suppose we have chosen a basis kernel $b(x)$. How shall we determine the coefficients $c[n]$?

A natural approach is to require that (8.14) exactly interpolate the given data samples, *i.e.*, we want

$$\hat{g}_a(n\Delta_x) = g_d[n] \quad (8.15)$$

for all $n \in \mathcal{N}$, where \mathcal{N} is the set of sample indexes. Typically $\mathcal{N} = \{0, \dots, N-1\}$ or $\mathcal{N} = \mathbb{Z}$.

Consider first the case $\mathcal{N} = \mathbb{Z}$ because it is easier to analyze. The interpolation condition (8.15) above is equivalent to requiring

$$g_d[n] = \sum_{k=-\infty}^{\infty} c[k] b(n-k) = c[n] * b_1[n],$$

where we define $b_1[n]$ to denote the unit-spaced samples of the kernel $b(x)$ as follows:

$$b_1[n] \triangleq b(n) = b(x)|_{x=n}.$$

Using the convolution property of the **DTFT**, the above expression becomes

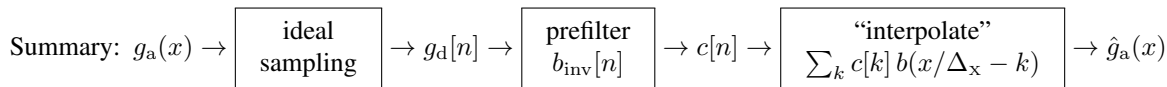
$$G(\Omega) = C(\Omega) B_1(\Omega) \implies C(\Omega) = G(\Omega) / B_1(\Omega).$$

Define the **inverse filter** $b_{\text{inv}}[n] \xleftrightarrow{\text{DSFT}} B_{\text{inv}}(\Omega) = \frac{1}{B_1(\Omega)}$ so that $b_1[n] * b_{\text{inv}}[n] = \delta[n]$. Then it follows that we can determine the interpolation coefficients by **prefiltering** the signal samples:

$$c[n] = b_{\text{inv}}[n] * g_d[n]. \quad (8.16)$$

Here is a summary of this approach to interpolation.

- Start with samples $g_d[n]$, $n \in \mathbb{Z}$.
- Choose a basis kernel $b(x)$.
- Determine the samples $b_1[n] = b(n)$ of that kernel.
- Determine the inverse filter $b_{\text{inv}}[n]$ for that sampled kernel such that $b_1[n] * b_{\text{inv}}[n] = \delta[n]$ and equivalently $B_{\text{inv}}(\Omega) = 1/B_1(\Omega)$. Often this step is done with Z-transforms.
- Convolve (“prefilter”) the samples $g_d[n]$ with the inverse filter $b_{\text{inv}}[n]$ to get the interpolation coefficients: $c[n] = b_{\text{inv}}[n] * g_d[n]$.
- Use those coefficients in the linear interpolation summation (8.14) for any and all values of x of interest.



Example. Suppose we choose a triangle kernel: $b(x) = \text{tri}(x)$. Then $b_1[n] = [0 \quad \underline{1} \quad 0] = \delta[n]$, so $b_{\text{inv}}[n] = \delta[n]$ and thus $\overline{c[n]} = \overline{g_d[n]}$ and the “generalized” method reverts to conventional linear interpolation with impulse response $h(x) = \text{tri}(x)$.

Example. Suppose we choose a triangle kernel: $b(x) = \text{tri}\left(\frac{x}{w}\right)$ where $1 < w < 2$.

Then $b_1[n] = [a \quad \underline{1} \quad a] = a\delta[n+1] + \delta[n] + a\delta[n-1]$, where $a = 1 - 1/w \in (0, 1/2)$.

To determine $b_{\text{inv}}[n]$, note that $B_1(z) = az + 1 + az^{-1}$ so $B_{\text{inv}}(z) = \frac{1}{B_1(z)} = \frac{1}{az + 1 + az^{-1}}$, where $z \triangleq e^{j\Omega}$.

We will see shortly that this is a stable filter that can be implemented as two passes of an IIR filter: one pass from left to right (causal), and one pass from right to left (anti-causal). Applying these IIR filters to $g_d[n]$ yields the coefficients $c[n]$.

The equivalent impulse response

Combining (8.14) and (8.16) we see that this kind of interpolator has the form

$$\begin{aligned}
 \hat{g}_a(x) &= \sum_{n=-\infty}^{\infty} c[n] b(x/\Delta_x - n) = \sum_{n=-\infty}^{\infty} (b_{\text{inv}}[n] * g_d[n]) b(x/\Delta_x - n) = \sum_{n=-\infty}^{\infty} \left(\sum_{k=-\infty}^{\infty} b_{\text{inv}}[n-k] g_d[k] \right) b(x/\Delta_x - n) \\
 &= \sum_{k=-\infty}^{\infty} g_d[k] \left(\sum_{n=-\infty}^{\infty} b_{\text{inv}}[n-k] b(x/\Delta_x - n) \right) = \sum_{k=-\infty}^{\infty} g_d[k] \left(\sum_{m=-\infty}^{\infty} b_{\text{inv}}[m] b(x/\Delta_x - k - m) \right) \\
 &= \sum_{k=-\infty}^{\infty} g_d[k] h(x/\Delta_x - k),
 \end{aligned}$$

where we let $m = n - k$ and the **equivalent impulse response** for this interpolation method is

$$\boxed{h(x) = \sum_{m=-\infty}^{\infty} b_{\text{inv}}[m] b(x - m).} \tag{8.17}$$

In other words, (8.14) is equivalent mathematically to linear interpolation of the form (8.13) with the interpolation kernel (8.17)! Although this shows there is “nothing new” mathematically, from a practical perspective there is an important difference, because usually (8.17) has infinite support, so it is impractical, whereas (8.14) is practical if we choose $b(x)$ to have finite support.

The equivalent frequency response

Taking the (1D CS) Fourier transform (using the shift property) of both sides of (8.17) yields the spectrum of the interpolation kernel:

$$H(\nu) = \sum_{m=-\infty}^{\infty} b_{\text{inv}}[m] B(\nu) e^{-i2\pi\nu m} = B(\nu) \sum_{m=-\infty}^{\infty} b_{\text{inv}}[m] e^{-i2\pi\nu m} = B(\nu) B_{\text{inv}}(\Omega) \Big|_{\Omega=2\pi\nu} = \boxed{\frac{B(\nu)}{B_1(2\pi\nu)}} \quad (8.18)$$

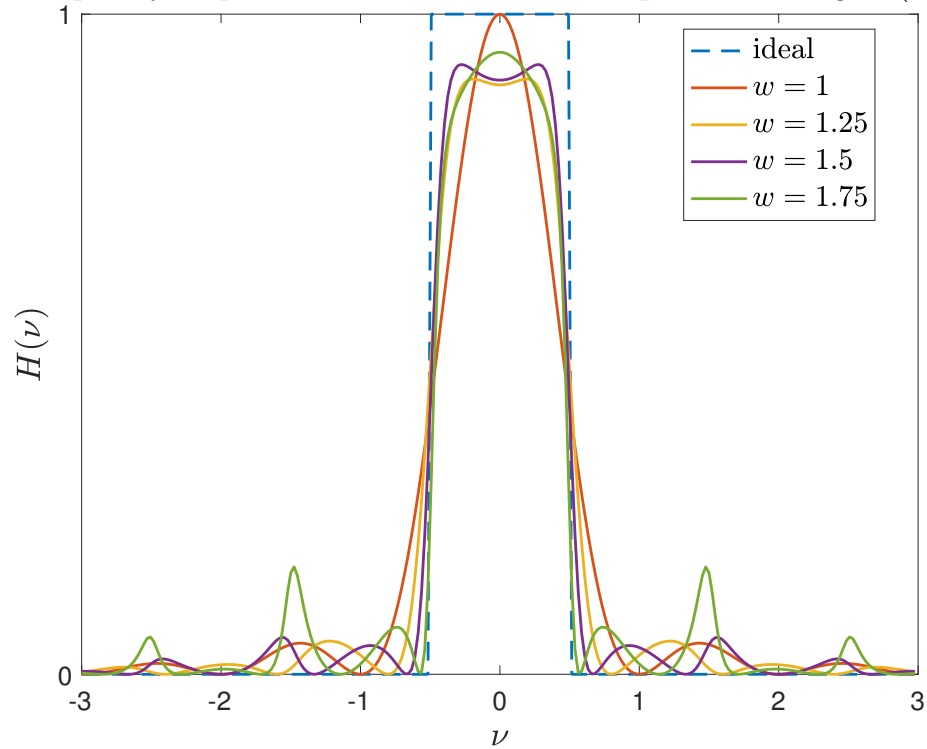
where $b(x) \xleftrightarrow{\mathcal{F}} B(\nu)$ and $b_{\text{inv}}[n] \xleftrightarrow{\text{DTFT}} B_{\text{inv}}(\Omega)$ and $b_1[n] \xleftrightarrow{\text{DTFT}} B_1(\Omega)$. What are ν 's units? **??** [RQ]
 Using basis functions $b(x)$ having small support limits how well the numerator $B(\nu)$ in (8.18) can approximate the ideal $\text{rect}(\nu)$. The presence of the denominator offers the opportunity to shape the frequency response towards that ideal.

Example. For the triangle basis kernel $b(x) = \text{tri}(x/w)$ discussed above, the equivalent interpolation kernel has spectrum

$$H(\nu) = \frac{w \text{sinc}^2(w\nu)}{a e^{i2\pi\nu} + 1 + a e^{-i2\pi\nu}} = \frac{w \text{sinc}^2(w\nu)}{1 + 2\frac{w-1}{w} \cos(2\pi\nu)}.$$

The following figure shows the frequency response $H(\nu)$ of the equivalent interpolation kernel $h(x)$ for a few values of the width parameter w .

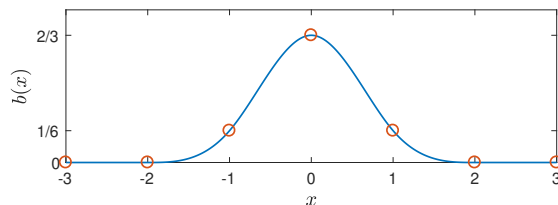
It is debatable whether any of the values of $w > 1$ are better than the usual linear interpolation kernel where $w = 1$. Apparently $\text{tri}(x/w)$ is a poor choice for the basis kernel, so we move on to find better options for $b(x)$.

Frequency response of shift-invariant interpolation using $\text{tri}(x/w)$ 

Basis kernels supported on $(-2,2)$

For any kernel supported on $(-1,1)$ for which $b(0) = 1$, what is $B_1(\Omega)$? ?? [RQ]

In other words, we need kernels with support wider than $(-1,1)$ to provide the opportunity to shape the frequency response $H(\nu)$. Therefore we now focus on basis kernels supported on $(-2,2)$, such as the Keys interpolator described previously. This family includes those supported on $(-3/2, 3/2)$; these also can be useful. Here is an example of a basis function having support $(-2,2)$.



We want such kernels to be symmetric and continuous, so we must have $b(\pm 2) = 0$. Therefore, the general form of the unit-spaced samples of such kernels is

$$b_1[n] = a \delta[n + 1] + d \delta[n] + a \delta[n - 1]$$

where $a = b(1)$, $d = b(0)$. We assume $d > 0$ and $a > 0$ hereafter. (The case $a = 0$ is trivial.) The frequency response of $b_1[n]$ is

$$B_1(\Omega) = a e^{i\Omega} + d + a e^{-i\Omega} = d + 2a \cos(\Omega).$$

If $d - 2a > 0$, then $B_1(\Omega) > 0 \forall \Omega$ and the filter may be invertible. Thus hereafter we assume $d > 2a > 0$, i.e., $b(0) > 2b(1) > 0$. This condition excludes $b(x) = \text{tri}(x/2)$ from consideration, for example.

The prefilter

Recall that the **Z-transform** of a signal is related to the DTFT of that signal by $z = e^{i\Omega}$.

To determine the inverse filter $b_{\text{inv}}[n]$, note that $b_1[n] \xleftrightarrow{Z} B_1(z) = az + d + az^{-1}$, so defining $r = \frac{d}{2a} > 1$ we have

$$b_{\text{inv}}[n] \xleftrightarrow{Z} B_{\text{inv}}(z) = \frac{1}{B_1(z)} = \frac{1}{az + d + az^{-1}} = \frac{z/a}{z^2 + 2rz + 1} = \frac{z/a}{(z-p)(z-p^{-1})} = \underbrace{\frac{1/a}{1-pz^{-1}}}_{\text{causal}} \underbrace{\frac{z^{-1}}{1-p^{-1}z^{-1}}}_{\text{anti-causal}}, \quad (8.19)$$

where $2r = -p - p^{-1}$ so $p = -r \pm \sqrt{r^2 - 1}$.

This form of $B_{\text{inv}}(z)$ is the product of two IIR filters, one that is causal and one that is anti-causal. This factorization will be useful only if those filters are **stable**. For the causal filter that means we need $|p| < 1$ (pole within the unit circle), whereas for the anti-causal filter we need the pole at p^{-1} to be *outside* the unit circle, which again means we need $|p| < 1$.

Note from our earlier condition on a and d that $r > 1$. Therefore we choose the following root:

$$p = -r + \sqrt{r^2 - 1}.$$

One can verify that this root satisfies $-1 < p < 0$ so p is within the unit circle. In terms of the basis kernel values:

$$p = -\frac{b(0)}{2b(1)} + \sqrt{\left(\frac{b(0)}{2b(1)}\right)^2 - 1}.$$

By **partial fraction expansion (PFE)**, or by using a table of Z-transforms, one can verify the following Z-transform pair:

$$p^{|n|} \xleftrightarrow{Z} \frac{(p - p^{-1})z^{-1}}{1 - (p + p^{-1})z^{-1} + z^{-2}} = \frac{(p - p^{-1})z^{-1}}{(1 - pz^{-1})(1 - p^{-1}z^{-1})}, \quad |p| < |z| < \frac{1}{|p|}.$$

Thus by (8.19) the impulse response of the inverse filter is

$$b_{\text{inv}}[n] = \frac{1}{a(p - p^{-1})} p^{|n|}. \quad (8.20)$$

This is a relatively rare situation where an inverse filter (a deconvolution filter) has a simple explicit form for its impulse response.

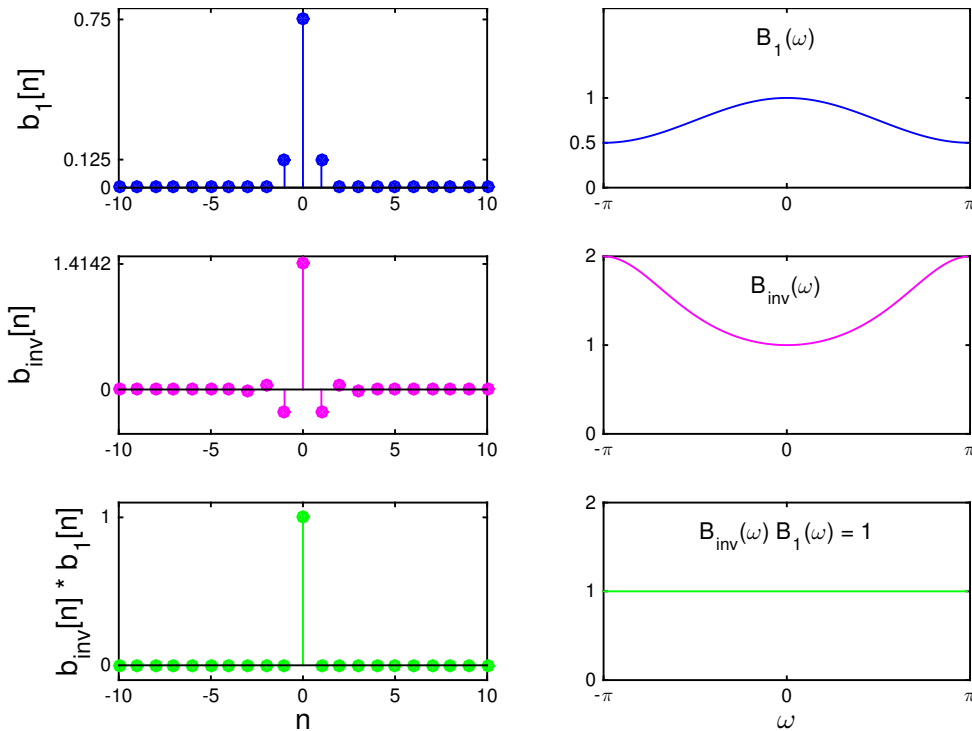
Using (8.19), the corresponding frequency response of this inverse filter is

$$b_{\text{inv}}[n] \xleftrightarrow{\text{DTFT}} B_{\text{inv}}(\Omega) = \frac{1}{d + 2a \cos(\Omega)}. \quad (8.21)$$

This will be a “high boost” filter.

Note how we have used CS FT and DS FT (actually DTFT) and Z-transform to analyze fully this type of interpolation!

Example. The following figure illustrates the case where $k b x[0] = 6/8$, $b(1) = 1/8$. This corresponds to the quadratic B-spline discussed later, for example.



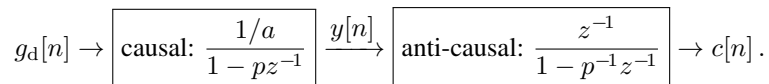
Efficient computation of interpolation coefficients

To compute the interpolation coefficients $c[n]$ we must prefilter the signal samples $g_d[n]$ per (8.16), i.e., $c[n] = b_{\text{inv}}[n] * g_d[n]$.

However, convolving $g_d[n]$ with the impulse response (8.20) directly would be quite inefficient, because $b_{\text{inv}}[n]$ has infinite support. In fact, performing this convolution directly would require effort comparable to having used the infinitely long ideal sinc interpolator in the first place.

Based on the previous example, it appears that one could truncate the inverse filter $b_{\text{inv}}[n]$, but the following recursive approach is more efficient.

The practical approach is to recognize that $B_{\text{inv}}(z)$ (8.19) represents the product of two IIR filters. So we can compute $c[n]$ from $g_d[n]$ using two IIR filter passes: one pass from left to right (causal), and one pass from right to left (anti-causal). A block diagram for this operation uses the following: IIR filter cascade:



This IIR filters have the following recursive forms (in principle for $n \in \mathbb{Z}$) that are very practical for implementation:

$$\begin{aligned} y[n] &= \frac{1}{a} g_d[n] + p y[n-1] && \text{(causal)} \\ c[n] &= p (c[n+1] - y[n]) && \text{(anti-causal).} \end{aligned} \tag{8.22}$$

These two filters require only 3 multiplications and 2 additions per sample.

In contrast, if we truncated the inverse filter to $|n| \leq 3$, how many multiplications would we need per sample? ??

How many additions? ??

So the IIR approach requires less computation.

[RQ]

As a side note, the recursions (8.22) are akin to the $O(N)$ algorithm for inverting a $N \times N$ **tridiagonal matrix**.

End conditions aka boundary conditions _____ **(skip)** ♦♦

In practical situations we have a finite number of samples, $n = 0, \dots, N - 1$, so we must initialize the IIR filters (8.22) properly.

The output of the ideal causal filter is:

$$y[n] = \frac{1}{a} \sum_{k=0}^{\infty} p^k g_d[n - k] \implies y[0] = \frac{1}{a} \sum_{k=0}^{\infty} p^k g_d[-k] \approx \frac{1}{a} \sum_{k=0}^K p^k g_d[-k],$$

where we choose K such that $p^K \approx 0$.

For **periodic end conditions**, we assume $g_d[-n] = g_d[N - n]$, and we have $y[0] \approx \frac{1}{a} g_d[0] + \frac{1}{a} \sum_{k=1}^K p^k g_d[N - k]$.

For **mirror end conditions**, we assume $g_d[-n] \approx g_d[n]$, and we have $y[0] \approx \frac{1}{a} \sum_{k=0}^K p^k g_d[k]$.

Having chosen one of these initial conditions for $y[0]$, we can then use the recursion in (8.22) for $y[n]$ for $n = 1, \dots, N - 1$.

The output of the ideal anti-causal filter is:

$$c[n] = - \sum_{k=0}^{\infty} p^{k+1} y[n + k] \implies c[N - 1] = - \sum_{k=0}^{\infty} p^{k+1} y[N - 1 + k] \approx - \sum_{k=0}^K p^{k+1} y[N - 1 + k],$$

where we choose K such that $p^K \approx 0$.

For **periodic end conditions**, we assume $y[N - 1 + k] = y[k - 1]$, for $k = 1, 2, \dots$, so $c[N - 1] \approx -p y[N - 1] - \sum_{k=1}^K p^{k+1} y[k - 1]$.

For **mirror end conditions**, we assume $y[N - 1 + k] \approx y[N - 1 - k]$, so $c[N - 1] \approx - \sum_{k=0}^K p^{k+1} y[N - 1 - k]$.

Unser [8, Box 2] recommends the following “in place” initialization for mirror end conditions:

$$c[N-1] = -\frac{p}{1-p^2} (y[N-1] + p y[N-2]).$$

Having chosen one of these boundary conditions for $c[N-1]$ we can then use the recursion in (8.22) for $c[n]$ for $n = N-2, \dots, 0$.

Equivalent impulse response

Combining (8.20) with (8.17) we see that the **equivalent impulse response** for an interpolation method with a basis kernel supported on $(-2, 2)$ is

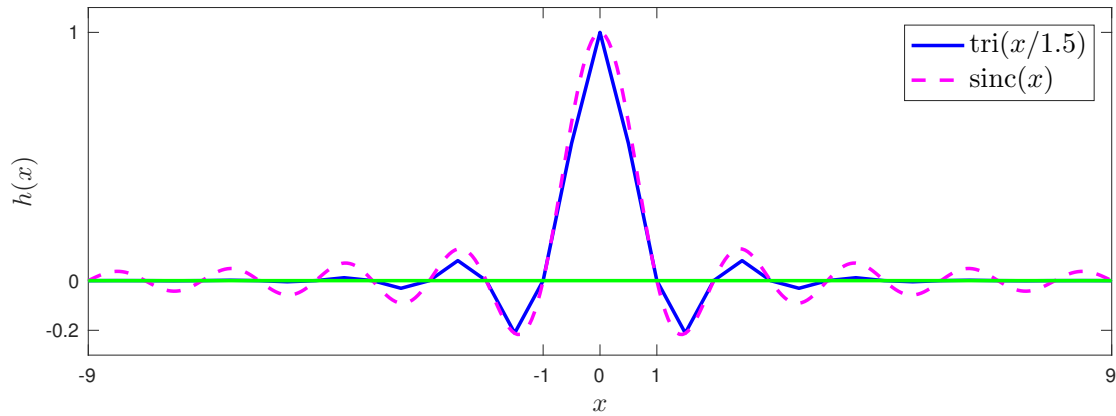
$$h(x) = \sum_{m=-\infty}^{\infty} b_{\text{inv}}[m] b(x-m) = \frac{1}{a(p-p^{-1})} \sum_{m=-\infty}^{\infty} p^{|m|} b(x-m). \quad (8.23)$$

Likewise, the equivalent frequency response from (8.18) is

$$H(\nu) = \frac{B(\nu)}{B_1(2\pi\nu)} = \frac{B(\nu)}{b(0) + 2b(1)\cos(2\pi\nu)}. \quad (8.24)$$

The key point here is that the equivalent impulse response $h(x)$ has infinite support, but our interpolation process requires just two simple first-order recursive filters and a finite-length basis kernel $b(x)$.

Example. We return to the triangular basis kernel $b(x) = \text{tri}(x/w)$ for $w = 1.5$. Here $a = 1 - 1/w = 1/3$ and $r = 1/(2a) = 3/2$ and $p = -r + \sqrt{r^2 - 1} = \frac{\sqrt{5}-3}{2} \approx -0.382$. Using (8.23) yields the following equivalent impulse response $h(x)$.



B-spline interpolation

See [8].

A B-spline model for $g_a(x)$ uses the shift-invariant space model (8.14) with basis kernel $b(x) = \beta^{(m)}(x)$, where $\beta^{(m)}(\cdot)$ is a **B-spline** of degree m . For equally spaced sample points (called **knots** in the spline literature), one can define B-splines as follows:

$$\beta^{(m)}(x) = \underbrace{\text{rect}(x) * \cdots * \text{rect}(x)}_{m+1 \text{ times}} \xleftrightarrow{\mathcal{F}} B^{(m)}(\nu) = \text{sinc}^{m+1}(\nu).$$

To develop a more explicit expression for $\beta^{(m)}(x)$, one can use the following trick for $\nu \neq 0$:

$$\begin{aligned} B^{(m)}(\nu) &= \frac{\sin^{m+1}(\pi\nu)}{(\pi\nu)^{m+1}} = \sin^{m+1}(\pi\nu)(2i)^{m+1} \left[\frac{1}{i2\pi\nu} + \frac{1}{2} \delta(\nu) \right]^{m+1} \\ &= [e^{i\pi\nu} - e^{-i\pi\nu}]^{m+1} \text{Step}^{m+1}(\nu) \\ &= \text{Step}^{m+1}(\nu) \sum_{k=0}^{m+1} \binom{m+1}{k} e^{-i\pi\nu k} (-1)^k e^{i\pi\nu(m+1-k)} \\ &= \text{Step}^{m+1}(\nu) \sum_{k=0}^{m+1} \binom{m+1}{k} (-1)^k e^{-i2\pi\nu[k - \frac{m+1}{2}]}, \end{aligned}$$

where

$$\text{step}(x) \xleftrightarrow{\mathcal{F}} \text{Step}(\nu) = \frac{1}{i2\pi\nu} + \frac{1}{2} \delta(\nu), \quad \frac{1}{m!} x^m \text{step}(x) \xleftrightarrow{\mathcal{F}} \text{Step}^{m+1}(\nu), \quad m \in \{0, 1, 2, \dots\}.$$

Letting $(t)_+ \triangleq t \text{step}(t)$ and taking the inverse FT yields the following “explicit” expression for a B-spline of degree m :

$$\beta^{(m)}(x) = \frac{1}{m!} \sum_{k=0}^{m+1} \binom{m+1}{k} (-1)^k \left(x - k + \frac{m+1}{2} \right)_+^m. \quad (8.25)$$

As an alternative to the above Fourier derivation, one can prove (8.25) by induction by using the following easily verified equality:

$$(t)_+^n * \text{step}(t) = \frac{1}{n+1} (t)_+^{n+1}.$$

The following recursive expressions can also be useful:

$$\beta^{(m)}(x) = \text{rect}(x) * \beta^{(m-1)}(x) = \frac{1}{m} \left(x + \frac{m+1}{2} \right) \beta^{(m-1)} \left(x + \frac{1}{2} \right) + \frac{1}{m} \left(\frac{m+1}{2} - x \right) \beta^{(m-1)} \left(x - \frac{1}{2} \right).$$

The derivatives also have a useful recursion:

$$\begin{aligned} \frac{d}{dx} \beta^{(m)}(x) &= \frac{d}{dx} \left(\text{rect}(x) * \beta^{(m-1)}(x) \right) = \left(\frac{d}{dx} \text{rect}(x) \right) * \beta^{(m-1)}(x) = (\delta(x+1/2) - \delta(x-1/2)) * \beta^{(m-1)}(x) \\ &= \beta^{(m-1)}(x+1/2) - \beta^{(m-1)}(x-1/2). \end{aligned}$$

The support of $\beta^{(m)}(x)$ is $[-(m+1)/2, (m+1)/2]$ so $m+1$ points are needed for interpolation.

Fact. As m increases, $\beta^{(m)}(x)$ appears more like a “bell curve.” Why? [central limit theorem \(CLT\)](#).

Fact. As m increases, the equivalent impulse response $h^{(m)}(x)$ for $\beta^{(m)}(x)$ approaches $\text{sinc}(x)$ [15].

Example. For $m=0$ we have: $\beta^{(0)}(x) = (x+1/2)_+^0 - (x-1+1/2)_+^0 = \text{step}(x+1/2) - \text{step}(x-1/2) = \boxed{??}$ [RQ]

Note that this difference of step functions matches (8.6).

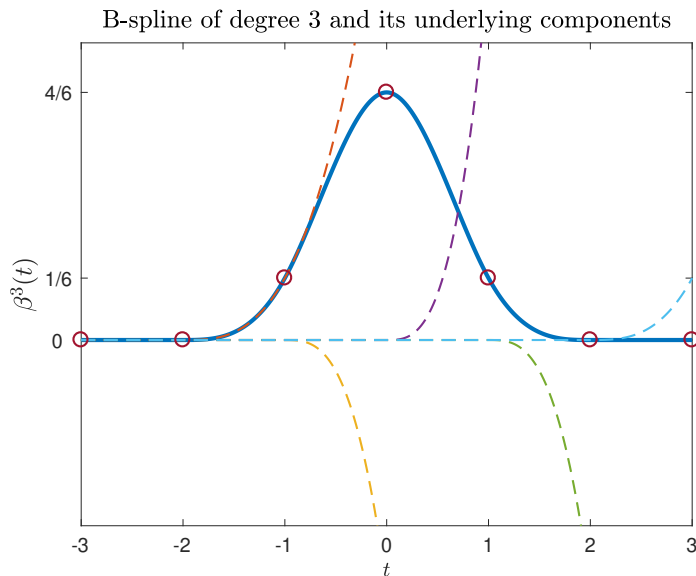
So **nearest-neighbor interpolation** and B-spline interpolation of degree $m=0$ are equivalent.

As a side note, the family of B-splines has been generalized to **fractional B-splines** [16] having spectra $B^{(m)}(\nu) = \text{sinc}^{m+1}(\nu)$ for non-integer values of m .

Example. The most important B-spline is the cubic B-spline ($m = 3$); it can be written

$$\beta^{(3)}(x) = \frac{1}{6} [(x+2)_+^3 - 4(x+1)_+^3 + 6(x)_+^3 - 4(x-1)_+^3 + (x-2)_+^3] = \begin{cases} 2/3 - x^2 + \frac{1}{2}|x|^3, & |x| \leq 1 \\ \frac{1}{6}(2 - |x|)^3, & 1 < |x| < 2 \\ 0, & \text{otherwise,} \end{cases} \quad (8.26)$$

and is shown below. This function does not satisfy the interpolation property! But it still can be used as a **basis kernel** in (8.14).



Exercise. Examine derivatives of $\beta^{(3)}(x)$ near $x = 2$.

Example. Determine the basis kernel and equivalent frequency response for the quadratic B-spline case ($m = 2$):

$$\beta^{(2)}(x) = \frac{1}{2} [(x - 3/2)_+^2 - 3(x - 1/2)_+^2 + 3(x + 1/2)_+^2 - (x + 3/2)_+^2] = \begin{cases} 3/4 - x^2, & |x| \leq 1/2 \\ \frac{1}{2}(|x| - 3/2)^2, & 1/2 < |x| < 3/2 \\ 0, & \text{otherwise.} \end{cases}$$

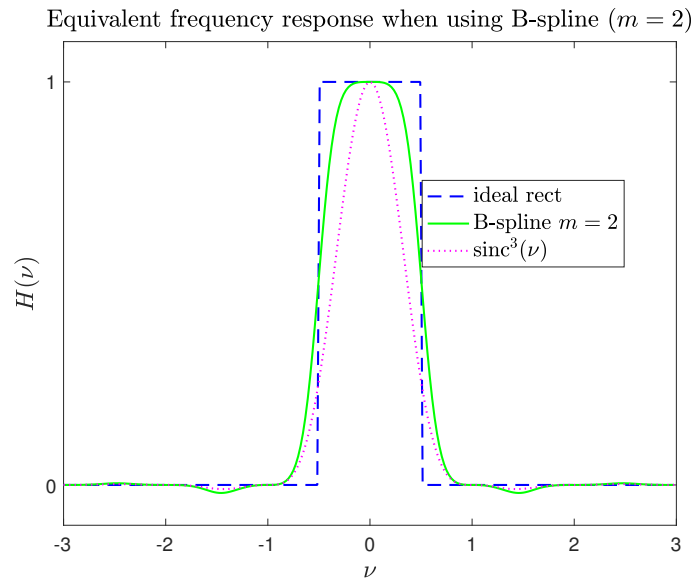
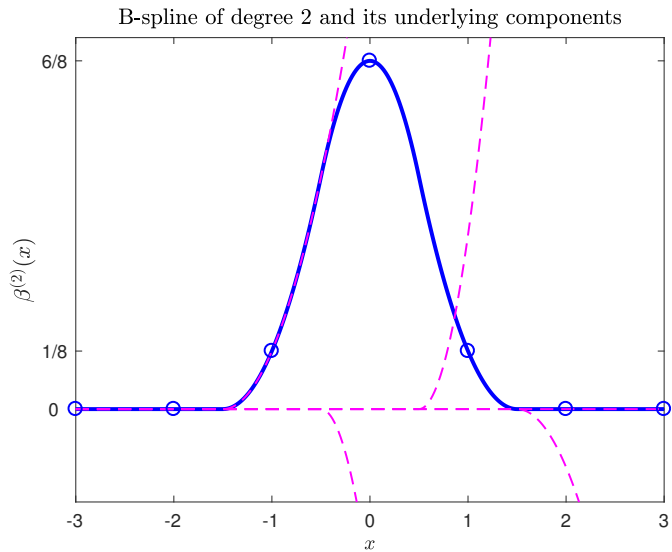
So the sampled quadratic B-spline is

[RQ]

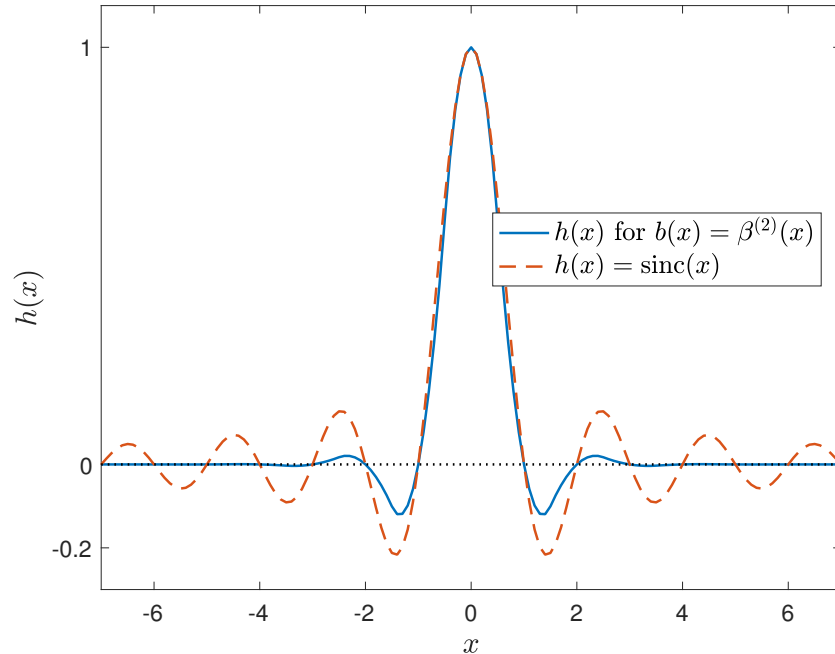
$$b_1^{(2)}[n] = \beta^{(2)}(n) = \boxed{??}$$

By (8.24), the equivalent frequency response is thus

$$H^{(2)}(\nu) = \frac{B^{(2)}(\nu)}{B_1^{(2)}(2\pi\nu)} = \frac{\text{sinc}^3(\nu)}{6/8 + 2/8 \cos(2\pi\nu)}.$$



The equivalent impulse response $h(x)$ has the form (8.23) with $r = \frac{\beta^{(2)}(0)}{2\beta^{(2)}(1)} = 3$ and $p = \sqrt{8} - 3 \approx -0.172$.

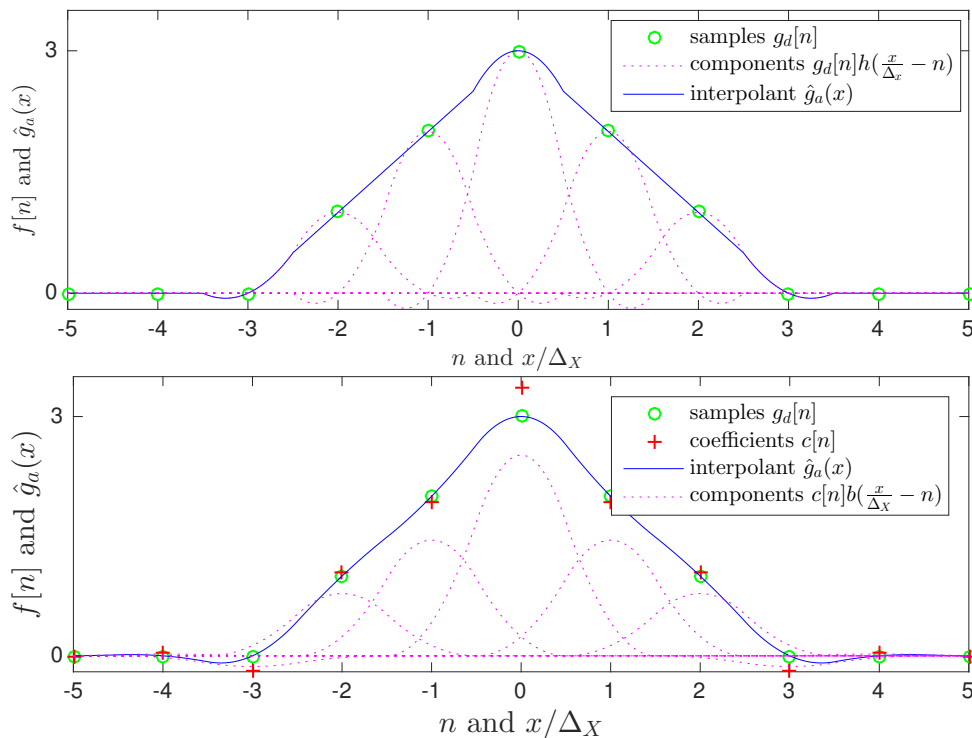


We can see that the equivalent $h(x)$ is smooth and has a bit of the ripple of a sinc interpolator but with much lower side-lobes.

Exercise. Consider the cubic case ($m = 3$) and find $H(\nu)$.

Example.

Comparing Dodgson interpolator (top) and quadratic B-spline interpolator (bottom).



2D interpolation in shift-invariant spaces

In 2D, using a separable basis kernel, the interpolator has the following form:

$$\hat{g}_a(x, y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} c[m, n] b\left(\frac{x - m\Delta_x}{\Delta_x}\right) b\left(\frac{y - n\Delta_y}{\Delta_y}\right). \quad (8.27)$$

We compute the coefficients $c[m, n]$ from the image sample values $g_d[m, n]$ by pre-filtering with a separable discrete-space filter:

$$c[m, n] = g_d[m, n] ** (b_{\text{inv}}[n] b_{\text{inv}}[m]).$$

The practical implementation applies the appropriate IIR filters left to right and then back right to left, and then top to bottom and back bottom to top. We filter each row independently, so this step can be parallelized. Likewise for filtering the columns.

A practical issue on modern computer systems is the memory access order; it may be beneficial to transpose the image between the row and column filtering operations so that the image pixel values being filtered are ordered sequentially in memory, thereby improving cache utilization.

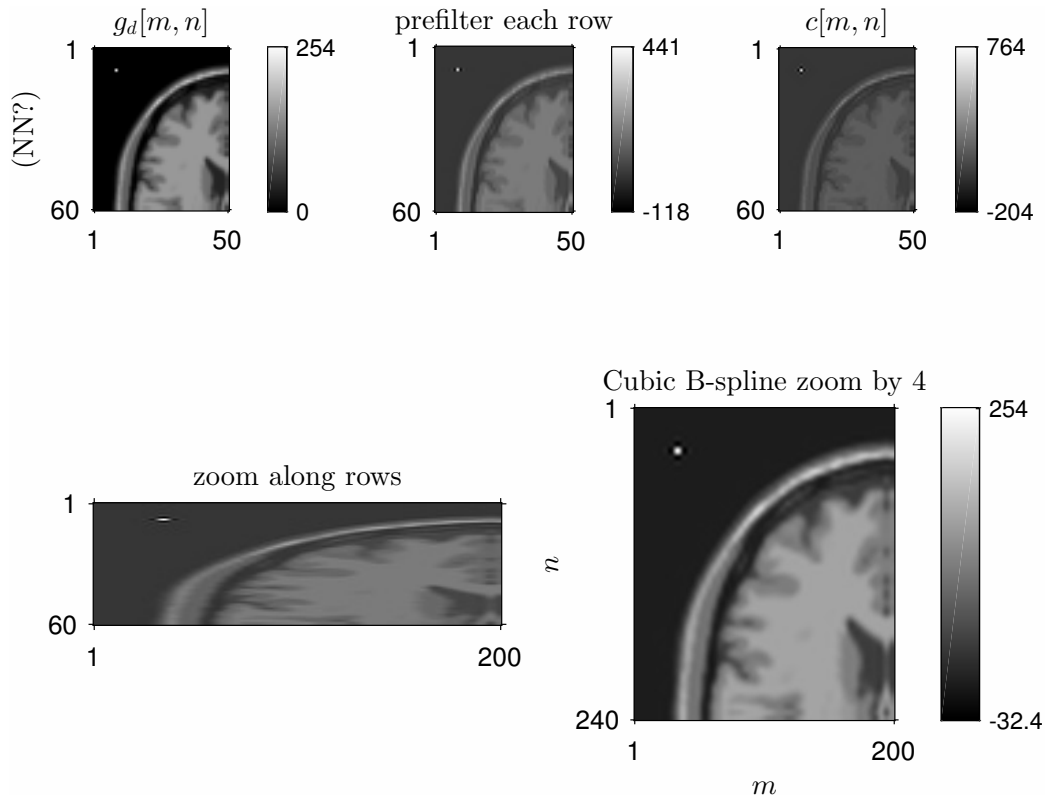
After computing the coefficients $c[m, n]$, for any location (x, y) of interest we evaluate (8.27) efficiently considering the support $(-T, T)$ of $b(x)$ as follows:

$$\hat{g}_a(x, y) = \sum_{n: -T < y/\Delta_y - n < T} \left(\sum_{m: -T < x/\Delta_x - m < T} c[m, n] b\left(\frac{x - m\Delta_x}{\Delta_x}\right) \right) b\left(\frac{y - n\Delta_y}{\Delta_y}\right).$$

The inner summation requires $2T$ operations for each the $2T$ values of m in the outer sum and then the outer sum needs another $2T$ operations so in total $4T^2 + 2T$ operations are needed, where each “operation” is an evaluation of $b(x)$, a multiplication, and an addition. The $O(T^2)$ load is a significant motivator for using small values of T , like $T = 2$ for a cubic spline.

- In 3D the computations are $O(T^3)$ and most medical imaging problems are 3D so interpolation (*e.g.*, for image registration) can require quite significant computation.
- There are open-source software packages like **ITK** that implement 3D B-spline interpolation.
- Last I checked, **MATLAB** does not provide B-spline based image interpolation, even though it is considered an excellent method. There are user-contributed tools available.

Example. The following figure shows (from left to right): an sampled image $g_a[m, n]$, the intermediate image after prefiltering each row, and then the final interpolation coefficients $c[m, n]$ after prefiltering each column. Here we used a cubic B-spline kernel $b(\cdot)$ in (8.27). We can use the coefficients $c[m, n]$ in (8.27) to do any interpolation operation, such as zooming.



8.3 Applications

We now describe a couple of applications of image interpolation.

Image registration**(skim)**

There are many applications where we would like to transform the spatial coordinates of an image $f(x, y)$, called the **moving image**, so that (after being transformed), it better “matches” another image $g(x, y)$, called the **fixed image**. Specifically, we want to find a spatial coordinate transformation (T_x, T_y) such that

$$g(x, y) \approx f(T_x(x, y), T_y(x, y)).$$

Letting $\vec{x} = (x, y)$ in 2D or $\vec{x} = (x, y, z)$ in 3D, a more concise way of writing this is

$$g(\vec{x}) \approx f(T(\vec{x})),$$

where $\vec{x} \in \mathbb{R}^d$ and $T : \mathbb{R}^d \rightarrow \mathbb{R}^d$ denotes the spatial transformation. Here $f(T(\vec{x}))$ is a transformed version of the original $f(\vec{x})$ hence f is the “moving image.”

When T is a transformation that does not preserve distances between points, then it is called a **nonrigid** transformation and the process is also called **morphing** or **warping** an image.

Example: warping many brain images into a standard coordinate system to facilitate automatic image analysis *e.g.*, [17, 18].

For surveys on the topic of **image registration**, see [19–23] or refer to books on the topic [24] [25].

The main research topics in the field of image registration are:

- choice of spatial **transformation**
- choice of **similarity metric**
 - same modality
 - different modality
- optimization method
- constraints, such as invertibility [26, 27]
- handling multiple (more than two) images

Spatial transformations (aka warping) _____ (skim)

Depending on the application, one might choose from among the many possible families of **spatial transformations**:

- Shift/translate only:

$$g(x, y) = f(x - x_0, y - y_0)$$

- Rotate/translate (**rigid**):

$$g(x, y) = f_{\theta}(x - x_0, y - y_0)$$

- Affine

$$g(x, y) = f(a_{11}x + a_{12}y - x_0, a_{21}x + a_{22}y - y_0)$$

- **Tensor-product** B-splines [28–30]:

$$T_x(x, y) = x + \sum_{k=0}^{K-1} \sum_{l=0}^{L-1} \alpha_x[k, l] b(x/w_x - k) b(y/w_y - l),$$

where the **knot spacing** w_x is a tunable parameter, and $\alpha_x[k, l]$ are called **deformation parameters**. The y component T_y is defined similarly.

What are the units of $\alpha_x[k, l]$? (cf. units of $c[m, n]$ in (8.27)) ?? [RQ]

See <http://www.mathworks.com/help/curvefit/multivariate-tensor-product-splines.html>

- **thin-plate splines (TPS)** (describes smooth warpings using control points) [31] which in 2D has the form:

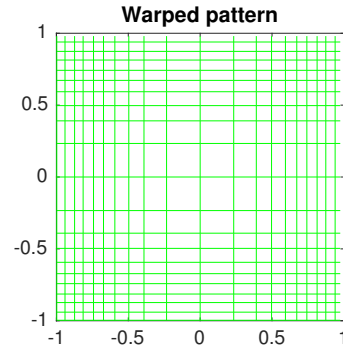
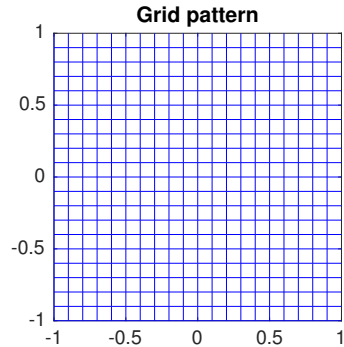
$$T_x(x, y) = \underbrace{x_0 + a_{11}x + a_{12}y}_{\text{affine portion}} + \sum_{k=1}^K \alpha_k b\left(\sqrt{(x - x_k)^2 + (y - y_k)^2}\right), \quad b(r) = r^2 \log r,$$

where the coefficients α_k are chosen such that $T_x(x_k, y_k) = x_0 + a_{11}x_k + a_{12}y_k$ by solving a system of equations.

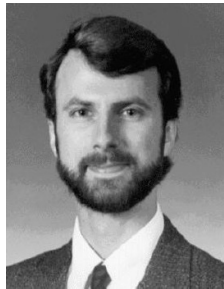
- **radial basis functions** [32] [33] [34] have the same form as TPS with other choices for $b(r)$.
- Other general nonrigid transformations; see [18] for a survey.

See MATLAB's `cp2tform` function for examples of defining spatial transforms from **control points**.

Example. The following figure shows an example of a nonrigid transformation, where, for illustration, $T_X(x, y) = x - 0.2 \sin(\pi x)$ and $T_Y(x, y) = y - 0.2 \sin(\pi y)$.



A professor



A warped professor



Interpolation**(skim)**

The descriptions above are in terms of hypothetical continuous-space images $g(x, y)$ and $f(x, y)$. In practice we have two digital images $g[m, n]$ and $f[m, n]$ that we want to align. To apply any of the preceding spatial transformations, we first interpolate one of the images, say $f[m, n]$, to define a continuous-space image, e.g., using B-spline based interpolation:

$$f(x, y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} c[m, n] b(x/\Delta_x - m) b(y/\Delta_y - n). \quad (8.28)$$

We then define a warped version of this image according to some candidate registration parameters α :

$$\tilde{f}_\alpha(x, y) = f(T_x(x, y; \alpha), T_y(x, y; \alpha)).$$

Then we sample this defined warped image to compute a warped digital image:

$$\tilde{f}_\alpha[m, n] = \tilde{f}_\alpha(m\Delta_x, n\Delta_y).$$

Similarity metrics**(skim)**

To perform image registration, we must measure the **similarity** (or dissimilarity) of $g[m, n]$ and $\tilde{f}_\alpha[m, n]$ to find the “best” transformation. There are many similarity metrics in the literature and this remains an active research area.

- sum of squared differences (for images of same modality / same scale):

$$\sum_{m=0}^{M-1} \sum_{n=0}^{N-1} |g[m, n] - \tilde{f}_\alpha[m, n]|^2 \quad (8.29)$$

- normalized correlation coefficient (for images that differ by an amplitude scale factor):

$$\frac{\sum_{m=0}^{M-1} \sum_{n=0}^{N-1} g[m, n] \tilde{f}_\alpha[m, n]}{\sum_{m=0}^{M-1} \sum_{n=0}^{N-1} |g[m, n]|^2}$$

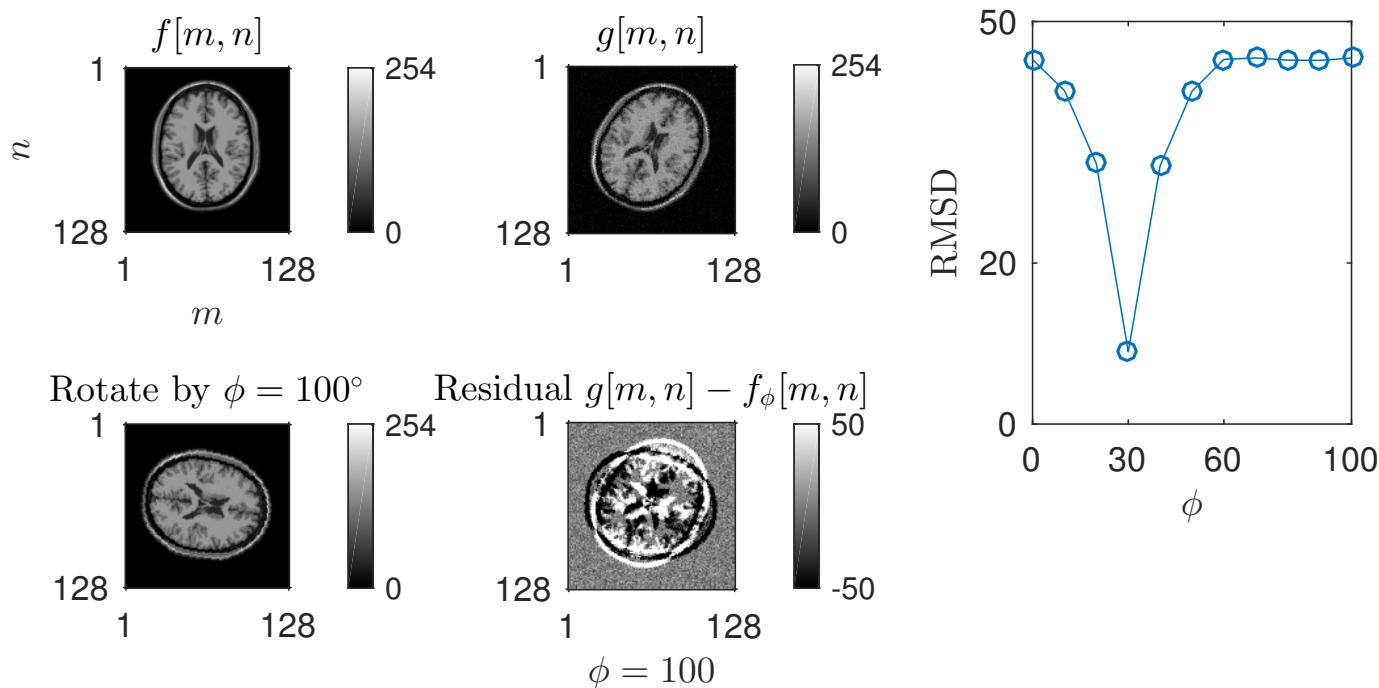
- Mutual information and its variants (for images from different modalities like CT and MRI)
- ...

Optimization algorithms _____ (skim)

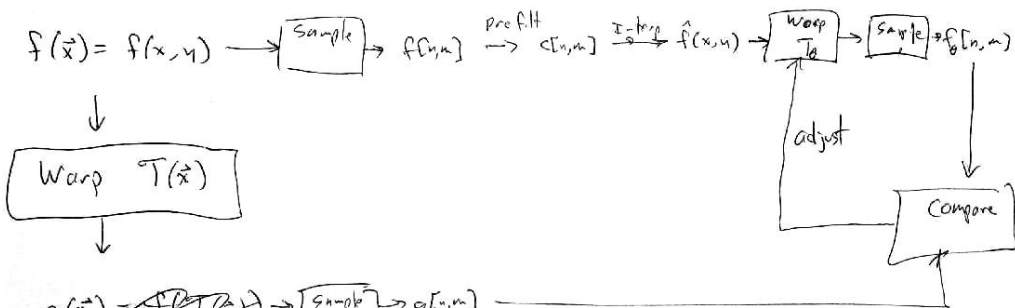
Finally we need an algorithm that optimizes the free parameters α of the allowable transformations to find the best match (highest similarity). This too is an active research area. It is a challenging problem because (dis)similarity metrics are non-convex functions of α and algorithms often converge to local optima.

Recently **deep learning** has been applied to image registration (and to every other image processing topic) [35].

Example. `demo_image_register1.m` The following figure illustrates a case where $g[m, n]$ is a rotated version of $f[m, n]$, with (in practice) unknown rotation. We try several different rotation angles (by “brute force search here”) and see which leads to the lowest RMSD, equivalent to sum of squared differences (8.29). Rotation of course requires interpolation.



Overview of image registration



$g(\vec{x}) = f(T(\vec{x})) \rightarrow \text{Sample} \rightarrow g[n, m]$
 " "
 $f(T(\vec{x}))$ model
 $T = T_\theta$ parameterize $\theta \in \Theta$

e.g. $t = (x_0, y_0) \cong \vec{x}_0$
 $g(x, y) = f(x - x_0, y - y_0)$
 $f_b(x, y) =$
 $g(\vec{x}) = f(T_b(\vec{x})) = f(\vec{x} - \vec{x}_0)$

Image zooming (up-sampling) using the FFT**(skim)**

Given a $M \times N$ digital image $f[m, n]$ for $m = 0, \dots, M-1$, $n = 0, \dots, N-1$ often we need to “up-sample” the image to say a larger size $K \times L$ image $g[m, n]$ for display. In computer graphics, this is called **image scaling**.

One way to do this is to use the interpolation methods described previously, *i.e.*, represent (using $\Delta_x = \Delta_y = 1$ for simplicity):

$$\hat{f}_a(x, y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f[m, n] h(x - m, y - n)$$

and then evaluate $\hat{f}_a(x, y)$ using a fine grid, *i.e.*,

$$g[m, n] \triangleq \hat{f}_a\left(m\frac{M}{K}, n\frac{N}{L}\right), \quad m = 0, \dots, K-1, \quad n = 0, \dots, L-1.$$

An alternative approach is to use the FFT to compute the (M, N) -point DFT of the image $f[m, n]$, then appropriately **zero pad** the DFT coefficients into a $K \times L$ array, and then compute the inverse (K, L) -point DFT (using an inverse FFT) to get an up-sampled image $g[m, n]$ for $m = 0, \dots, K-1$, $n = 0, \dots, L-1$. *One must “zero pad” quite carefully for this to work properly.*

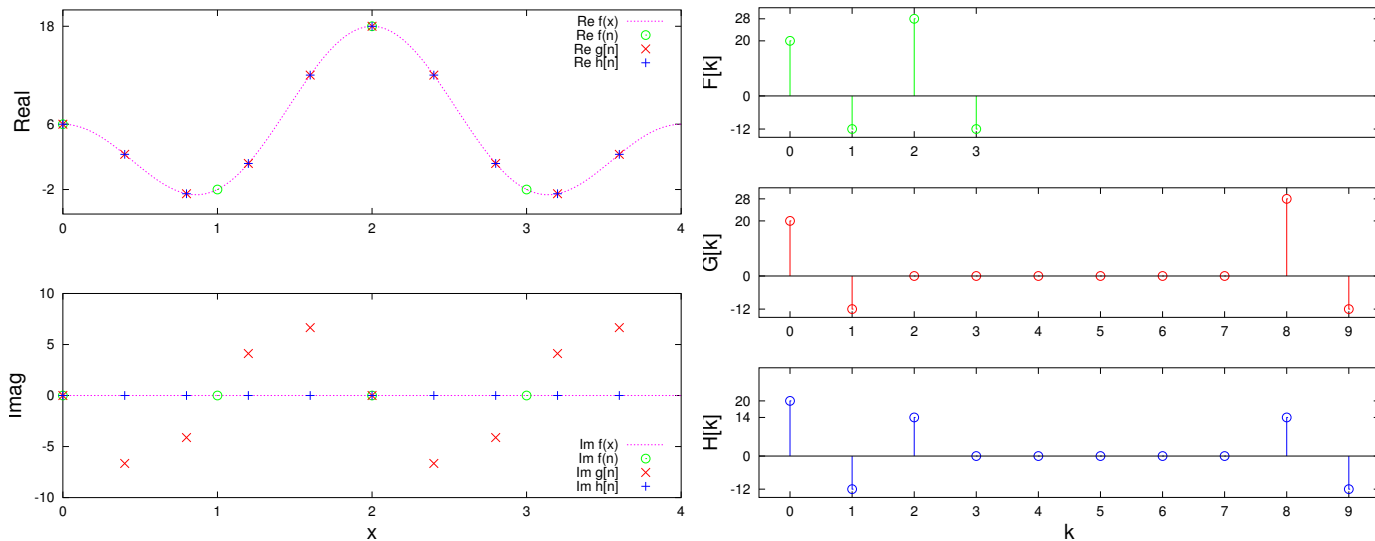
MATLAB’s `interpft` routine does this for 1D vectors or for each column of a 2D array. Unfortunately, `interpft` (in MATLAB and octave as of 2013-02-17) contains a subtle error that yields a nonzero imaginary part when interpolating a real signal.

For a 1D signal $f[n]$, the `interpft` routine first uses the FFT to compute $F[k]$, for $k = 0, \dots, N-1$, and then defines a new vector $G[k]$ of length $K \geq N$ by inserting $K - N$ zeros (*i.e.*, **zero padding**) in the “high frequency” indexes as follows:

$$G[k] \triangleq \begin{cases} F[k], & k = 0, \dots, N/2 - 1 \\ 0, & k = N/2, \dots, K - N/2 - 1 \\ F[k - K + N], & k = K - N/2, \dots, K - 1, \end{cases} \quad (8.30)$$

where we assume N is even for simplicity of exposition. Then `interpft` uses the inverse FFT of $G[k]$ and multiplies by K/N to compute $g[n]$ corresponding to $f_a(nN/K)$ for $n = 0, \dots, K-1$.

Unfortunately, the definition (8.30) does not satisfy the DFT **Hermitian symmetry** property corresponding to real signals, leading to a nonzero imaginary part as illustrated in the figures.



To use FFTs properly for DFT-based up-sampling, we instead define the following zero-padded version (again assuming N is even for simplicity):

$$H[k] = \begin{cases} F[k], & k = 0, \dots, N/2 - 1 \\ \frac{1}{2} F[N/2], & k = N/2 \\ 0, & k = N/2 + 1, \dots, K - N/2 - 1 \\ \frac{1}{2} F[N/2], & k = K - N/2 \\ F[k - K + N], & k = K - N/2 + 1, \dots, K - 1, \end{cases} \quad (8.31)$$

Then we use the inverse FFT and multiply by K/N to compute $h[n]$ corresponding to $f_a(nN/K)$ for $n = 0, \dots, K - 1$.

Exercise. Analyze the relationship between $h[n]$ and $f[n]$ mathematically.

Note. Apparently MATLAB R2015b addressed this issue, and also added the following “failsafe” to the end of `interpft`:

```
if isreal(x), y = real(y); end
```

Why? (class/pair)

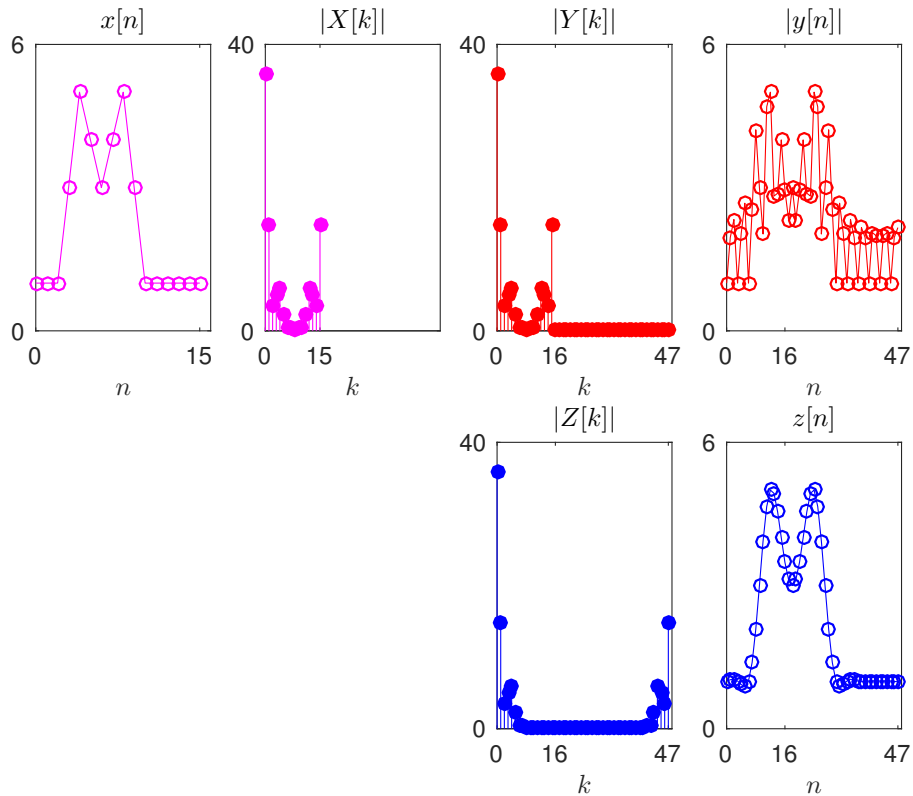
How would one do such FFT-based interpolation using zero padding for complex signals? ??

(class/pair)

It is essential to insert the zeros in the “middle” of the DFT coefficients (*i.e.*, around π in the DSFT) where the high spatial frequencies are located. The following example illustrates the importance of being careful about where zeros are inserted.

Example. In the following figure, we formed $Y[k]$ from $X[k]$ by adding zeros at the end of the $N = 16$ DFT samples and we formed $Z[k]$ from $X[k]$ by inserting zeros in the middle of the DFT samples per (8.31). The latter works correctly.

Clearly `g = ifft(fft(f), K);` does not work!

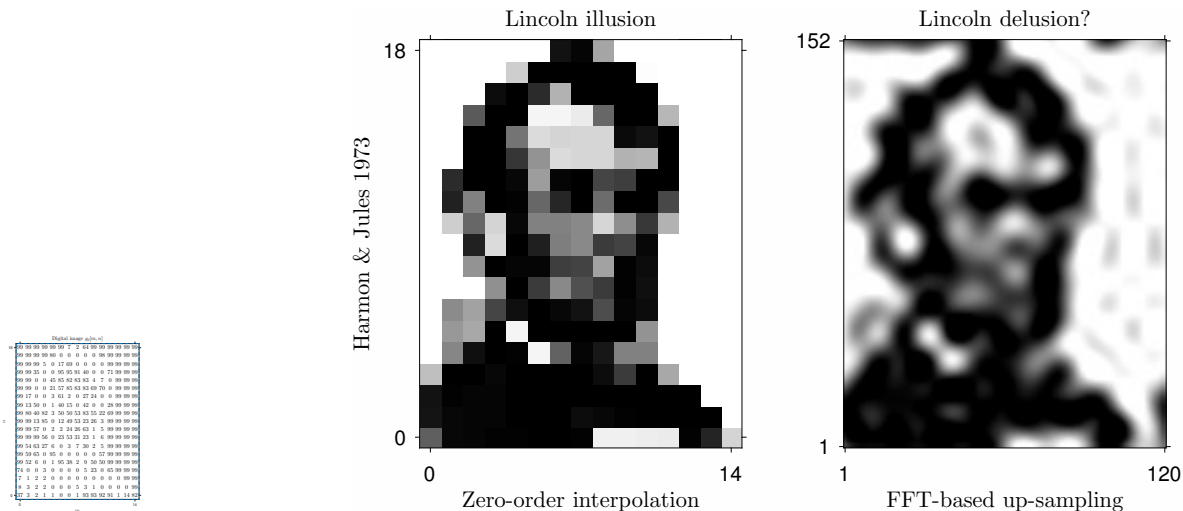


Using the **separability** of 2D Fourier transforms, we can achieve 2D up-sampling via the FFT in MATLAB using

```
g = ir_interpft( ir_interpft(f, K) .', L ) .';
```

where `ir_interpft` is an improved version of `interpft` based on (8.31).

Example. Here is 2D image zooming (by a factor of 8) using 2D FFT.



What causes the dark smudge along the lower right edge? **Wrap-around effects, *i.e.*, the periodic sinc or Dirichlet.**
 What causes the ripples or ringing? **Side-lobes of the periodic sinc or Dirichlet.**

Many variations of this method exist, including using the **DCT** [36].

8.4 Motion estimation for video temporal interpolation

(Just skim this section.)

The preceding ideas generalize readily to 3D interpolation problems such as image registration of 3D medical images $g(x, y, z)$. Because that generalization is straightforward, we instead move beyond 2D by turning to motion estimation and video interpolation. See [37]. A video sequence

$$f(x, y, t)$$

is a function of 3 arguments, so in principle “2D+time” could be viewed as a kind of “3D problem.” However, the time dimension often has different practical considerations than the spatial dimensions. Different applications need different models and methods, but many applications involving motion are **real time**, *e.g.*, in video processing, so the temporal part of the processing should be **causal** or nearly so. Thus in practice we often use techniques for video that differ from 3D spatial interpolation problems.

A typical problem is the following. Given two image frames $f(x, y, t_-)$ and $f(x, y, t_0)$, find (or approximate) $f(x, y, t)$ for $t_- < t < t_0$. (Of course in practice we will only have a finite number of spatial samples, but because time sampling is the larger issue here we focus on that aspect.)

Example. Converting 24 frames/sec movie images to 60 frame/sec NTSC TV images. Conventional 3:2 approach shows 1 cine frame for 3 TV frames and the next for 2 TV frames, etc. This is a form of **nearest neighbor interpolation** in time. This simple interpolation method can cause unnatural motion (temporal “blocking”).

Example. Smooth cross-fade between two images [38].

Example. “SteadyCam” features of camcorders. For a fun application, see [this article](#).

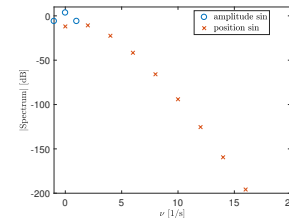
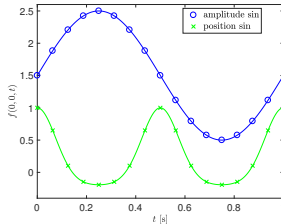
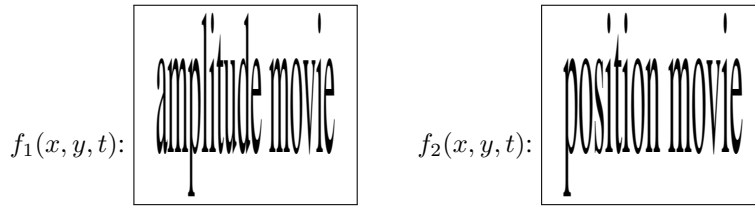
In principle, we could develop **space-time interpolation** theory for moving objects $f(x, y, t)$ that are **bandlimited** both in terms of spatial frequencies **and** in terms of time variations, *e.g.*, [39, 40]. Such a theory may have specialized applications such as cardiac imaging, but in general such “bandlimited” space-time theory may have limited practical use for video processing, because

- moving objects may not be bandlimited
- even if the objects are bandlimited, the sampling rates may be inadequate (video sampling rates are governed more by human perception than by *object* properties, as evidenced by backwards turning wagon wheels in western movies!)
- even if the sampling rates were adequate, the theoretical interpolation methods are both IIR *and* noncausal, hence are impractical.

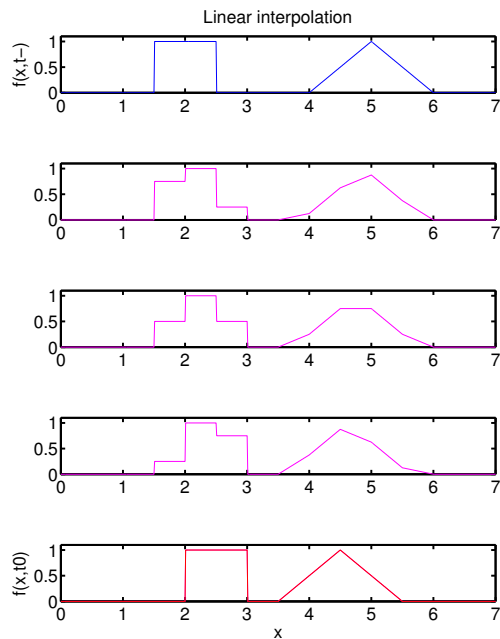
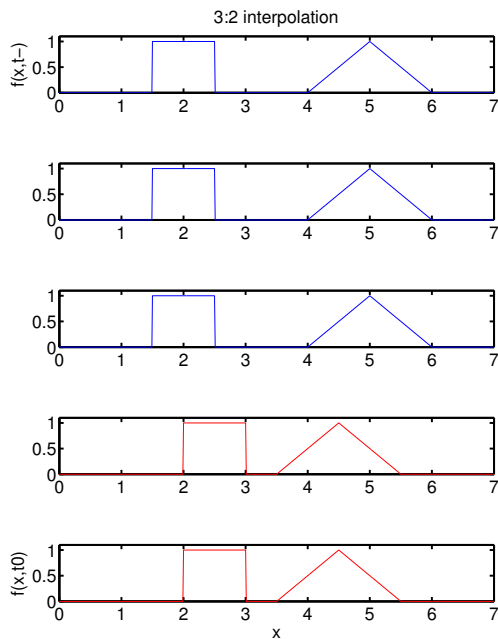
Example.

Time-varying amplitude: $f_1(x, y, t) = (1.5 + \sin(2\pi t)) \text{sinc}_2\left(\frac{x}{10}, \frac{y}{10}\right)$. Is this signal **space-time band-limited?** ??

Time-varying position: $f_2(x, y, t) = \text{sinc}_2\left(\frac{x - \sin(2\pi t)}{10}, \frac{y}{10}\right)$. Is this signal **space-time band-limited?** ??



Example. Using **linear interpolation** (in time) between frames can cause artifacts too for moving objects, as shown below.



We need more sophisticated but still simple (and probably suboptimal) interpolation methods that take into account interframe **motion** of objects within the scene.

Interestingly, names of **videos** and **films** include **movies** and **motion pictures**, reflecting the importance of *motion*.

Translation motion estimation

Having a model for an object's temporal variations helps in the design of a motion-compensated temporal interpolation method.

The simplest model is

$$f(x, y, t_0) = f(x - d_x, y - d_y, t_-), \quad (8.32)$$

where

- $f(x, y, t_0)$ denotes the current frame
- $f(x, y, t_-)$ denotes the past frame,
- (d_x, d_y) denotes the object shift that occurs between the two frames.

This is a global displacement model, but in practice it is applied only locally to small regions (e.g., 8 by 8 pixels) in the image.

What about $t \in (t_-, t_0)$? Because displacement is the time integral of velocity, we can write

$$d_x = x(t_0) - x(t_-) = \int_{t_-}^{t_0} v_x(t) dt.$$

To simplify further, we assume **uniform velocity** between t_- and t_0 , so

$$d_x = v_x \cdot (t_0 - t_-).$$

Thus, under the above two assumptions we have the following *model* that forms the basis of an interpolation method:

$$\hat{f}(x, y, t) = f(x - v_x \cdot (t - t_-), y - v_y \cdot (t - t_-), t_-) = s(\alpha(x, y, t), \beta(x, y, t)), \quad t_- \leq t \leq t_0, \quad (8.33)$$

where

- $s(x, y) = f(x, y, t_-)$ is the “starting image”
- $\alpha(x, y, t) = x - v_x \cdot (t - t_-)$ is the x -coordinate transformation

- $\beta(x, y, t) = y - v_Y \cdot (t - t_-)$ is the y -coordinate transformation.

By construction, we have $\hat{f}(x, y, t_-) = f(x, y, t_-)$. But if the *actual* motion departs from the above simple model, then in general $\hat{f}(x, y, t_0) \neq f(x, y, t_0)$ for some values of x, y , regardless of how we choose v_X and v_Y .

Our next goal thus becomes choosing v_X and v_Y , or equivalently d_X and d_Y , so that

$$\hat{f}(x, y, t_0) \approx f(x, y, t_0).$$

Region-matching methods (based on (8.32))

We want to determine v_x and v_y (or equivalently d_x and d_y) to minimize some measure of “error” between the current frame $f(x, y, t_0)$ and its prediction

$$\hat{f}(x, y, t_0) = f(x - d_x, y - d_y, t_-).$$

A natural approach is to select a region R and find the (d_x, d_y) pair that minimizes the integrated squared difference between the two images over that region:

$$\min_{d_x, d_y} \iint_{(x, y) \in R} |f(x, y, t_0) - f(x - d_x, y - d_y, t_-)|^2 dx dy. \quad (8.34)$$

Metrics other than squared error are also useful.

Unfortunately, finding the best d_x, d_y pair is a *very* nonlinear minimization problem, fraught with difficulties such as local minima. There is no closed-form analytical solution, so one must apply **iterative methods**.

One approach to help reduce computation and avoid local minima is to use a **multiresolution** search strategy; one begins searching on a *coarse* grid of d_x and d_y values, and then refines the search over successively finer grids. With such strategies (and appropriate interpolation), one can achieve **subpixel** accuracy.

To achieve subpixel matching (which is often desirable because objects do not usually move by integer amounts between frames!) one must apply some type of spatial interpolation to evaluate the above integral (which will of course be replaced by a summation in practice). A standard choice is **bilinear interpolation**.

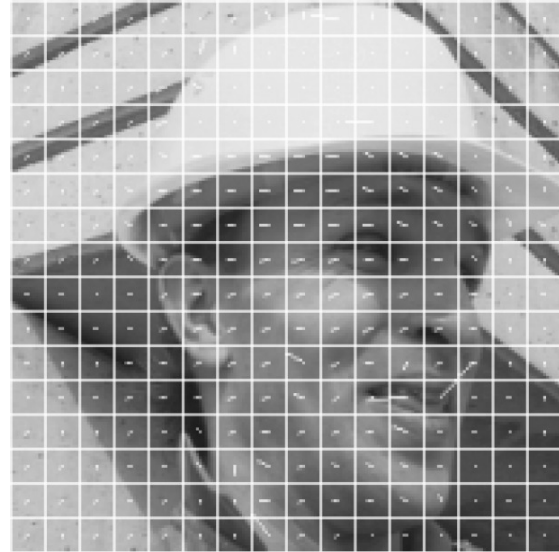
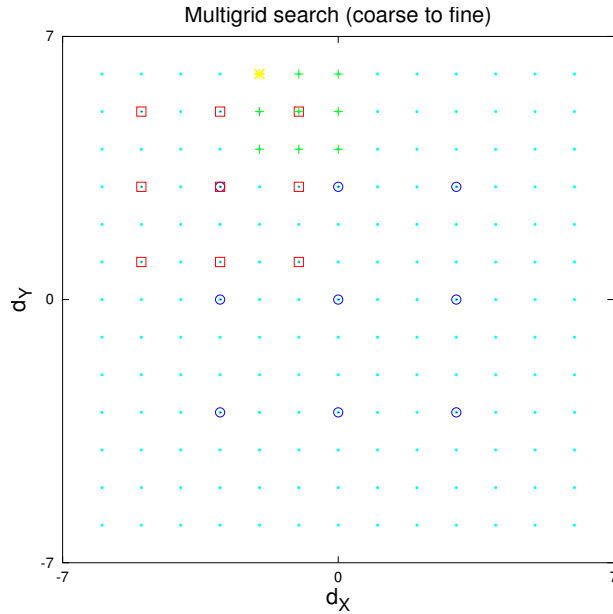
One can also apply a coordinate descent algorithm where one alternates between minimizing over d_x with d_y fixed to its most recent estimate and then updating d_y with d_x fixed to its most recent estimate.

Region-matching or **block-matching** methods are used routinely for **video coding**.

Example.

The left figure illustrates a coarse-to-fine or multigrid search for the best displacement parameters d_x, d_y .

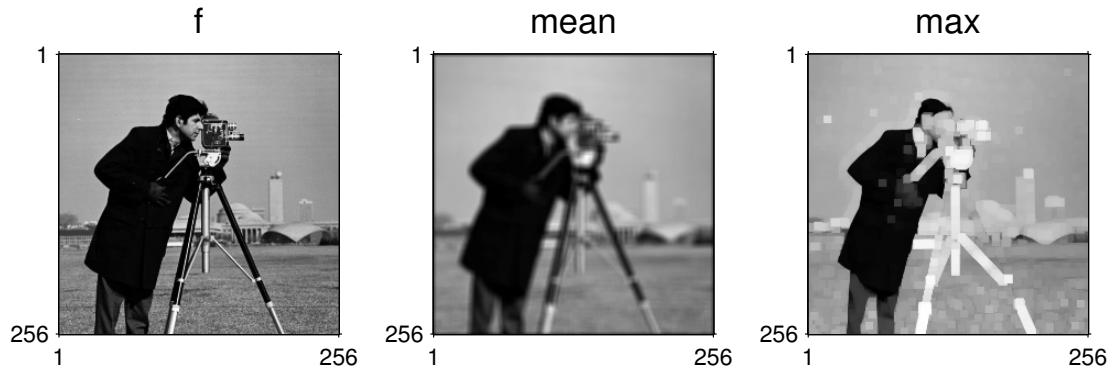
The right figure illustrates the kind of estimated displacement vectors that result from such an approach.



<http://www.youtube.com/watch?v=9-v801bv6A0>

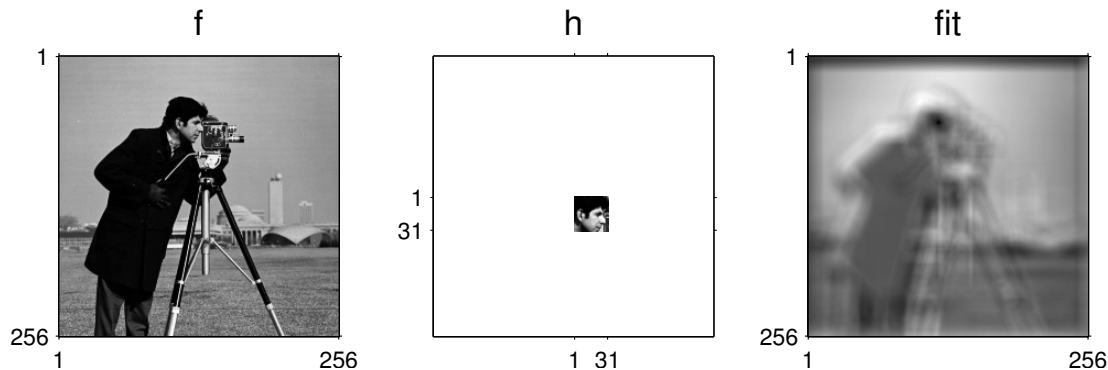
Example. To illustrate how one can perform block matching (without interpolation) with minimal MATLAB coding, it is helpful to first illustrate use of the `colfilt` function.

```
% demo_colfilt.m illustrate colfilt()  
f = single(imread('cameraman.tif'));  
fun1 = @(x) mean(x); fun2 = @(x) max(x);  
g1 = colfilt(f, [7 7], 'sliding', fun1);  
g2 = colfilt(f, [7 7], 'sliding', fun2);  
im plc 1 3, im(1, f, 'f'), im(2, g1, 'mean'), im(3, g2, 'max')
```



Here is an illustration of how to use `colfilt` to find the “best match” for a given region (a face in this case).

```
% fig_region_match1.m
f = single(imread('cameraman.tif'));
h = f(120+[-15:15],61+[-15:15]);
fun = @(x) sum(abs(x - repmat(h(:), 1, ncol(x))).^2);
fit = colfilt(f, size(h), 'sliding', fun);
im plc 1 3, im(1, f, 'f'), im(2, h, 'h'), axis([0 1 0 1]*256-128), im(3, fit, 'fit')
```



Space-time constraint methods (based on (8.35))

Using the relationship (8.33), *i.e.*, $\hat{f}(x, y, t) = s(\alpha(x, y, t), \beta(x, y, t))$, by the chain rule the partial derivatives are

$$\begin{aligned}\frac{\partial}{\partial x} \hat{f}(x, y, t) &= \frac{\partial s}{\partial \alpha} \frac{\partial \alpha}{\partial x} + \frac{\partial s}{\partial \beta} \frac{\partial \beta}{\partial x} = \frac{\partial s}{\partial \alpha} \\ \frac{\partial}{\partial y} \hat{f}(x, y, t) &= \frac{\partial s}{\partial \alpha} \frac{\partial \alpha}{\partial y} + \frac{\partial s}{\partial \beta} \frac{\partial \beta}{\partial y} = \frac{\partial s}{\partial \beta} \\ \frac{\partial}{\partial t} \hat{f}(x, y, t) &= \frac{\partial s}{\partial \alpha} \frac{\partial \alpha}{\partial t} + \frac{\partial s}{\partial \beta} \frac{\partial \beta}{\partial t} = -v_x \frac{\partial s}{\partial \alpha} - v_y \frac{\partial s}{\partial \beta}.\end{aligned}$$

When combined, these equalities lead to the following **space-time constraint equation**:

$$\boxed{v_x \frac{\partial f(x, y, t)}{\partial x} + v_y \frac{\partial f(x, y, t)}{\partial y} + \frac{\partial f(x, y, t)}{\partial t} = 0.} \quad (8.35)$$

This equation disregards other object motion such as rotations, zooms, or multiple objects with different velocities, although it can be generalized.

Instead of using (8.32) and (8.34), we can also estimate the motion using (8.35).

From (8.35) we can find v_x and v_y (from which we can determine d_x and d_y), as follows.

How many equations and how many unknowns per pixel? ??

Physically, we cannot determine the component of motion perpendicular to the gradient (*i.e.*, parallel to the edge) without additional assumptions.

This problem is related to the **barber pole illusion**.

One solution is to use a **spatial coherence constraint** introduced by Lucas and Kanade [41].

- Choose a space-time region over which (v_x, v_y) is *assumed* to be constant.
- Let $\{(x_i, y_i, t_i)\}_{i=1}^N$ denote a set of samples within that region.
- Use these samples to form estimates of the three types of partial derivatives of $f(x, y, t)$, using any of the **derivative calculation** methods described previously.

Denote these estimates of the partial derivatives as follows:

$$g_i^x \triangleq \left. \frac{\partial f(x, y, t)}{\partial x} \right|_{(x_i, y_i, t_i)}, \quad g_i^y \triangleq \left. \frac{\partial f(x, y, t)}{\partial y} \right|_{(x_i, y_i, t_i)}, \quad g_i^t \triangleq \left. \frac{\partial f(x, y, t)}{\partial t} \right|_{(x_i, y_i, t_i)}.$$

Then from (8.35):

$$v_x g_i^x + v_y g_i^y + g_i^t \approx 0.$$

We would like to find the velocity estimates \hat{v}_x and \hat{v}_y that “best agree” with the constraint equation.

A natural approach is to use a least-squares criterion:

$$(\hat{v}_x, \hat{v}_y) = \arg \min_{v_x, v_y} \sum_{i=1}^N (v_x g_i^x + v_y g_i^y + g_i^t)^2.$$

This is a linear least-squares problem which can also be written

$$\min_{\boldsymbol{\theta}} \|\mathbf{A}\boldsymbol{\theta} - \mathbf{b}\|^2,$$

where $\boldsymbol{\theta} = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$, \mathbf{A} is a $N \times 2$ matrix and \mathbf{b} is a vector of length N with entries

$$\mathbf{A} = \begin{bmatrix} g_1^x & g_1^y \\ \vdots & \vdots \\ g_N^x & g_N^y \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -g_1^t \\ \vdots \\ -g_N^t \end{bmatrix}.$$

The solution to such problems is given by the **normal equations**:

$$(\mathbf{A}'\mathbf{A})\boldsymbol{\theta} = \mathbf{A}'\mathbf{b}.$$

If \mathbf{A} has full column rank, *i.e.*, if the two columns are linearly independent, then the LS solution is:

$$\boldsymbol{\theta} = (\mathbf{A}'\mathbf{A})^{-1}\mathbf{A}'\mathbf{b}.$$

However, it is possible for the columns of \mathbf{A} to be linearly *dependent*. When?

(Over a spatially uniform region, all spatial derivatives are zero!)

In this case one usually adopts a minimum-norm LS solution, or used the following strategy based on the eigenvalues of $\mathbf{A}'\mathbf{A}$:

- If λ_1 and λ_2 fall below a threshold, use $\boldsymbol{\theta} = 0$. (No motion, or uniform intensity region.)
- If $\lambda_1 \gg \lambda_2$, then the motion appears to be primarily along one direction (1D).

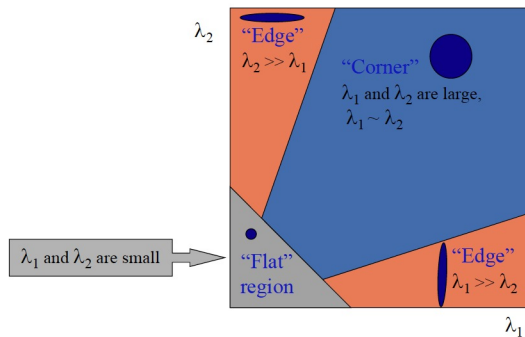
Let the **Gram matrix** $\mathbf{G} = \mathbf{A}'\mathbf{A}$ have the eigenvector decomposition:

$$\mathbf{G} = [\mathbf{u}_1 \ \mathbf{u}_2] \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} [\mathbf{u}_1 \ \mathbf{u}_2]',$$

where \mathbf{u}_1 and \mathbf{u}_2 denote orthonormal eigenvectors (because \mathbf{G} is symmetric nonnegative definite). Then the LS solution gives:

$$\hat{\boldsymbol{\theta}} = \mathbf{G}^{-1}\mathbf{A}'\mathbf{b} = [\mathbf{u}_1 \ \mathbf{u}_2] \begin{bmatrix} \frac{1}{\lambda_1} & 0 \\ 0 & \frac{1}{\lambda_2} \end{bmatrix} [\mathbf{u}_1 \ \mathbf{u}_2]' \mathbf{A}'\mathbf{b}.$$

The following figure illustrates the behavior of the eigenvalues for different types of image patches. The Gram matrix can be interpreted as the empirical covariance of the gradient vectors (g_i^x, g_i^y) . In an edge region, most of the gradient vectors will point along a single direction.



(Figure by Prof. Savarese)

The reciprocal of λ_2 will be very unstable in the presence of noise if $\lambda_1 \gg \lambda_2$. A solution is to zero out the component along \mathbf{u}_2 , which is a kind of **pseudo inverse**:

$$\hat{\boldsymbol{\theta}} = [\mathbf{u}_1 \ \mathbf{u}_2] \begin{bmatrix} \frac{1}{\lambda_1} & 0 \\ 0 & 0 \end{bmatrix} [\mathbf{u}_1 \ \mathbf{u}_2]' \mathbf{A}' \mathbf{b} = \frac{\mathbf{u}_1' \mathbf{A}' \mathbf{b}}{\lambda_1} \mathbf{u}_1.$$

Of course to implement such methods one must estimate the three partial derivatives. Local polynomial fitting (in 3D) is a reasonable choice.

To reduce computation, one can use variable sized grids and select the region accordingly.

Region selection

In both the region matching method and the space-time constraint method, reducing region size will reduce computation.

However, if the region is too small, then the object may move out of the region on the next frame. One method used to minimize the size of regions while reducing the influence of large displacements is to prediction. In this approach, the region is not constant between frames. Based on estimates of the displacement in the previous frame, the center of the region is translated. This

procedure allows tracking of displacements comparable to or greater than the extent of the region. Although reducing computations, this method is susceptible to “run away” if the original estimates of the displacement are inaccurate.

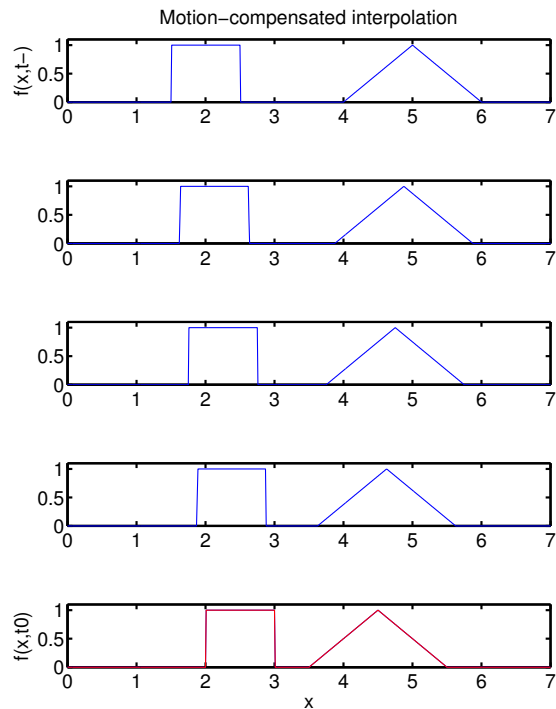
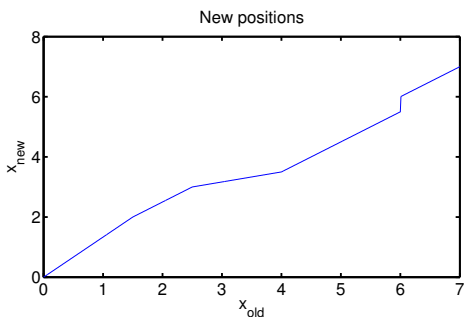
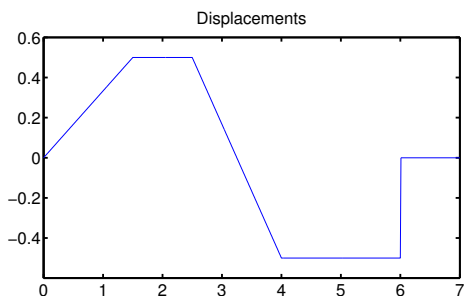
Advanced topics

More general approaches to finding **optical flow** are available [42], including from photon-limited measurements [43]. See [44] for a nice unified treatment. Complications that arise in practice are discontinuities [45–48] and occlusions [49, 50]. See also [51] [52] [53].

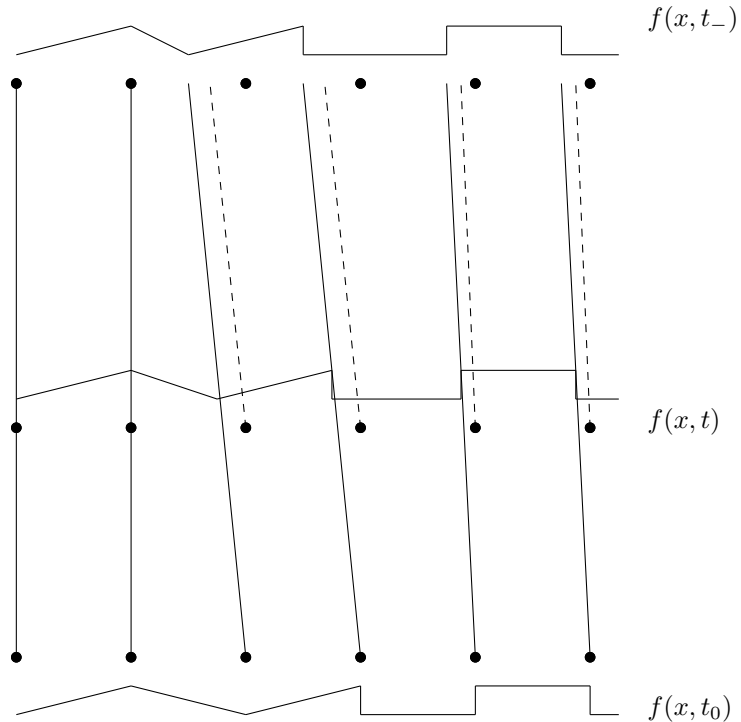
Can combine these ideas with **super resolution** methods [54].

Motion compensated temporal interpolation

By matching the rectangle and triangle in the previous frame to the current frame, we can estimate their displacements. Then we interpolate intermediate frames using the model (8.33). This *nonlinear* interpolation method is quite effective.



Example. Here is another example illustrating that if we can find corresponding **features** in two frames, then we can estimate the displacements and interpolate between those frames using the estimated motion.



Application of motion estimation methods to spatial interpolation

Consider the following *spatial* interpolation problem.

Given $f(x, y_0)$ and $f(x, y_1)$, find $f(x, y)$ for $y_0 < y < y_1$.

This problem is essentially identical to the problem considered by (8.32), so similar approaches apply.

In fact, in the 1D interpolation figures shown previously, the running dimension could be *either* time or y , and the results would be identical.

It is sometimes claimed that such an approach can outperform sinc interpolation.

How is this possible? Because this type of approach is *nonlinear* and *shift varying*.

We can use 1D region matching or a 1D version of the space-time constraint equation to estimate the correspondences between $f(x, y_0)$ and $f(x, y_1)$.

This approach may be explored in a HW problem with comparison to conventional linear interpolation methods.

8.5 Summary

- There are numerous applications of image interpolation and numerous available methods.
- Classical methods use interpolation kernels. Modern methods use shift invariant spaces.
- Methods based on B-splines offer a particularly good tradeoff between interpolation quality and computation.
- FFT can be used for zooming if one is careful with the zero padding.
- Temporal interpolation of video sequences benefits from motion estimation. For more image processing in cinema / movies see [55].

What part of this reading assignment did you find least clear? What part was the most interesting? ??

[RQ]

Cubic Hermite interpolation

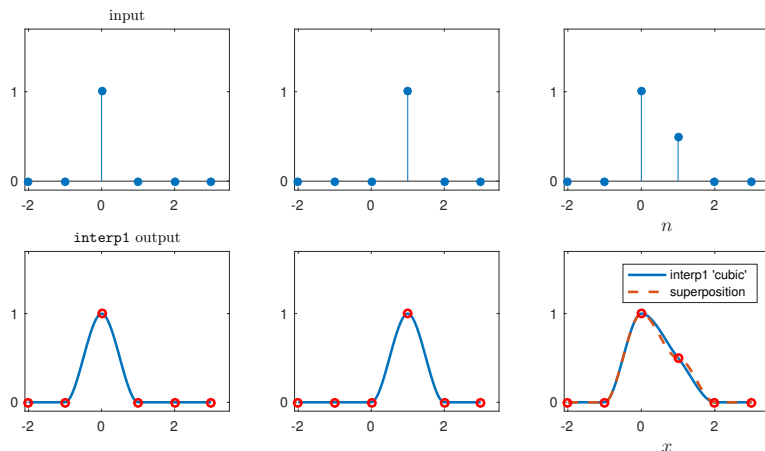
The 'pchip' option of MATLAB's `interp1` command is named for **piecewise cubic Hermite interpolation**. It actually corresponds to **monotone cubic Hermite interpolation** [12]. The 'pchip' method is very different from all the other kernel-based interpolators that we have discussed here, including the 'v5cubic' option of `interp1`; it is in fact a nonlinear method.

It defines a piecewise cubic function specified by the following conditions:

- given value at each sample point,
- continuous derivative at each sample point,
- continuous second derivative at each sample point,
- specified derivative (and second derivative) at end points.

To find such a cubic function in 1D from N points, one must solve an equation with $4N$ unknowns. These properties imply that: interpolation at a point depends on the values of *all* points, not just the neighboring points, the method is nonlinear, and computation is much more expensive than using kernels.

This figure illustrates the nonlinearity of the 'cubic' = 'pchip' option of `interp1`.



Bibliography

- [1] D. Shepard. “A two-dimensional interpolation function for irregularly-spaced data”. In: *ACM Conf.* 1968, 517–24.
- [2] O. V. Morozov, M. Unser, and P. Hunziker. “Reconstruction of large, irregularly sampled multidimensional images. A tensor-based approach”. In: *IEEE Trans. Med. Imag.* 30.2 (Feb. 2011), 366–74.
- [3] V. Chatzis and I. Pitas. “Interpolation of 3-D binary images based on morphological skeletonization”. In: *IEEE Trans. Med. Imag.* 19.7 (July 2000), 699–710.
- [4] E. Meijering. “A chronology of interpolation: from ancient astronomy to modern signal and image processing”. In: *Proc. IEEE* 90.3 (Mar. 2002), 319–42.
- [5] C. Sigg and M. Hadwiger. “Fast third-order texture filtering”. In: *GPU Gems 2*. Ed. by M. Pharr. Reading: Addison-Wesley, 2005, pp. 307–29.
- [6] P. Thevenaz, T. Blu, and M. Unser. “Interpolation revisited”. In: *IEEE Trans. Med. Imag.* 19.7 (July 2000), 739–58.
- [7] N. A. Dodgson. “Quadratic interpolation for image resampling”. In: *IEEE Trans. Im. Proc.* 6.9 (Sept. 1997), 1322–6.
- [8] M. Unser. “Splines: A perfect fit for signal and image processing”. In: *IEEE Sig. Proc. Mag.* 16.6 (Nov. 1999), 22–38.
- [9] M. Unser, A. Aldroubi, and M. Eden. “B-spline signal processing: Part I—theory”. In: *IEEE Trans. Sig. Proc.* 41.2 (Feb. 1993), 821–33.
- [10] E. Maeland. “On the comparison of interpolation methods”. In: *IEEE Trans. Med. Imag.* 7.3 (Sept. 1988), 213–7.
- [11] R. G. Keys. “Cubic convolution interpolation for digital image processing”. In: *IEEE Trans. Acoust. Sp. Sig. Proc.* 29.6 (Dec. 1981), 1153–60.
- [12] F. N. Fritsch and R. E. Carlson. “Monotone piecewise cubic interpolation”. In: *SIAM J. Numer. Anal.* 17.2 (Apr. 1980), 238–46.
- [13] F. Andersson, M. Carlsson, and V. V. Nikitin. “Fast algorithms and efficient GPU implementations for the Radon transform and the back-projection operator represented as convolution operators”. In: *SIAM J. Imaging Sci.* 9.2 (2016), 637–64.
- [14] A. S. Glasser. *Graphics gems*. Morgan Kaufman, 1990.
- [15] M. Unser. “Sampling-50 years after Shannon”. In: *Proc. IEEE* 88.4 (Apr. 2000), 569–87.

- [16] M. Unser and T. Blu. “Fractional splines and wavelets”. In: *SIAM Review* 42.1 (Mar. 2000), 43–67.
- [17] A. C. Evans, C. Beil, S. Marrett, C. J. Thompson, and A. Hakim. “Anatomical-functional correlation using an adjustable MRI-based region of interest atlas with positron emission tomography”. In: *J. Cerebral Blood Flow and Metabolism* 8.4 (Aug. 1988), 513–30.
- [18] M. Holden. “A review of geometric transformations for nonrigid body registration”. In: *IEEE Trans. Med. Imag.* 27.1 (Jan. 2008), 111–28.
- [19] J. B. A. Maintz and M. A. Viergever. “A survey of medical image registration”. In: *Med. Im. Anal.* 2.1 (Mar. 1998), 1–36.
- [20] J. M. Fitzpatrick, D. L. G. Hill, and C. R. Maurer. “Image registration”. In: *Handbook of Medical Imaging, Volume 2. Medical Image Processing and Analysis*. Ed. by M. Sonka and J. Michael Fitzpatrick. Bellingham: SPIE, 2000, pp. 477–513.
- [21] D. L. G. Hill, P. G. Batchelor, M. Holden, and D. J. Hawkes. “Medical image registration”. In: *Phys. Med. Biol.* 46.3 (Mar. 2001), R1–47.
- [22] J. P. W. Pluim, J. B. A. Maintz, and M. A. Viergever. “Mutual-information-based registration of medical images: a survey”. In: *IEEE Trans. Med. Imag.* 22.8 (Aug. 2003), 986–1004.
- [23] B. Zitová and J. Flusser. “Image registration methods: a survey”. In: *Im. and Vision Computing* 21.11 (Oct. 2003), 977–1000.
- [24] J. Modersitzki. *Numerical methods for image registration*. Oxford, 2003.
- [25] J. Modersitzki. *FAIR: Flexible algorithms for image registration*. SIAM, 2009.
- [26] S. Y. Chun and J. A. Fessler. “A simple regularizer for B-spline nonrigid image registration that encourages local invertibility”. In: *IEEE J. Sel. Top. Sig. Proc.* 3.1 (Feb. 2009). Special Issue on Digital Image Processing Techniques for Oncology., 159–69.
- [27] M. Sdika. “A sharp sufficient condition for B-spline vector field invertibility. Application to diffeomorphic registration and interslice interpolation”. In: *SIAM J. Imaging Sci.* 6.4 (2013), 2236–57.
- [28] W. M. Hsu, J. F. Hughes, and H. Kaufman. “Direct manipulation of free-form deformations”. In: *Proc. 19th Annual Conf. on Computer Graphics and Interactive Techniques*. 1992, 177–84.
- [29] S. Lee, G. Wolberg, K-Y. Chwa, and S. Y. Shin. “Image metamorphosis with scattered feature constraints”. In: *IEEE Trans. Vis. Comp. Graphics* 2.4 (Dec. 1996), 337–54.
- [30] D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. Leach, and D. J. Hawkes. “Nonrigid registration using free-form deformations: application to breast MR images”. In: *IEEE Trans. Med. Imag.* 18.8 (Aug. 1999), 712–21.

- [31] F. L. Bookstein. “Principal warps: thin-plate splines and the decomposition of deformations”. In: *IEEE Trans. Patt. Anal. Mach. Int.* 11.6 (June 1989), 567–87.
- [32] Z. Wu. “Compactly supported positive definite radial functions”. In: *Adv. in Comp. Math.* 4 (1995), 283–92.
- [33] H. Wendland. “Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree”. In: *Adv. in Comp. Math.* 4.1 (1995), 389–96.
- [34] Z. Wu. “Generalized Bochner’s theorem for radial function”. In: *Approximation Theory and its Applications* 13.3 (1997), 47–57.
- [35] X. Yang, R. Kwitt, and M. Niethammer. *Quicksilver: Fast predictive image registration - a deep learning approach*. arxiv 1703.10908. 2017.
- [36] M-K. Cho and B-U. Lee. “Discrete cosine transform domain image resizing using correlation of discrete cosine transform coefficients”. In: *J. Electronic Imaging* 15.3 (July 2006), p. 033009.
- [37] M. A. Tekalp. *Digital video processing*. New York: Prentice-Hall, 1996.
- [38] I. Kemelmacher-Shlizerman, E. Shechtman, R. Garg, and S. M. Seitz. “Exploring photobios”. In: *ACM Trans. on Graphics* 30.4 (July 2011). Proc. of ACM SIGGRAPH., p. 61.
- [39] N. P. Willis and Y. Bresler. “Optimal scan for time-varying tomography I: Theoretical analysis and fundamental limitations”. In: *IEEE Trans. Im. Proc.* 4.5 (May 1995), 642–53.
- [40] N. P. Willis and Y. Bresler. “Lattice-theoretic analysis of time-sequential sampling of spatiotemporal signals. II. Large space-bandwidth product asymptotics”. In: *IEEE Trans. Info. Theory* 43.1 (Jan. 1997), 208–20.
- [41] B. Lucas and T. Kanade. “An iterative image registration technique with an application to stereo vision”. In: *Proc. Seventh International Joint Conference on Artificial Intelligence*. Vol. 2. 1981, 674–9.
- [42] B. Horn and B. G. Schunck. “Determining optical flow”. In: *Artif. Intell.* 18.1-3 (Aug. 1981), 185–203.
- [43] C. L. Chan and A. K. Katsaggelos. “Iterative maximum likelihood displacement field estimation in quantum-limited image sequences”. In: *IEEE Trans. Im. Proc.* 4.6 (June 1995), 743–51.
- [44] A. Bruhn, J. Weickert, and C. Schnörr. “Lucas/Kanade meets Horn/Schunck: combining local and global optic flow methods”. In: *Intl. J. Comp. Vision* 61.3 (Feb. 2005), 211–31.

- [45] L. Blanc-Féraud, M. Barlaud, and T. Gaidon. “Motion estimation involving discontinuities in a multiresolution scheme”. In: *Optical Engineering* 32.7 (July 1993), 1475–82.
- [46] F. Heitz and P. Bouthemy. “Multimodal estimation of discontinuous optical flow using Markov random fields”. In: *IEEE Trans. Patt. Anal. Mach. Int.* 15.12 (Dec. 1993), 1217–32.
- [47] P. Nesi. “Variational approach to optical flow estimation managing discontinuities”. In: *Im. and Vision Computing* 11.7 (Sept. 1993), 419–39.
- [48] R. Deriche, P. Kornprobst, and G. Aubert. “Optical-flow estimation while preserving its discontinuities: A variational approach”. In: *Proc. 2nd Asian Conference on Computer Vision*. Vol. 2. 1995, 71–80.
- [49] D. Sun, S. Roth, and M. J. Black. “A quantitative analysis of current practices in optical flow estimation and the principles behind them”. In: *Intl. J. Comp. Vision* 106.2 (Jan. 2014), 115–37.
- [50] D. Sun, C. Liu, and H. Pfister. “Local layering for joint motion estimation and occlusion detection”. In: *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*. 2014.
- [51] M. Dawood, F. Buther, X. Jiang, and K. P. Schafers. “Respiratory motion correction in 3-D PET data with advanced optical flow algorithms”. In: *IEEE Trans. Med. Imag.* 27.8 (Aug. 2008), 1164–75.
- [52] D. Cremers and S. Soatto. “Variational space-time motion segmentation”. In: *Proc. Intl. Conf. Comp. Vision*. Vol. 2. 2003, 886–93.
- [53] B. Goldluecke, E. Strekalovskiy, and D. Cremers. “Tight convex relaxations for vector-valued labeling”. In: *SIAM J. Imaging Sci.* 6.3 (2013), 1626–64.
- [54] S. Høgild Keller, François Lauze, and M. Nielsen. “Motion compensated video super resolution”. In: *Proc. 1st intl. conf. on scale space and variational methods in computer vision*. 2007, 801–812.
- [55] M. Bertalmio. *Image processing for cinema*. Chapman, 2014.

Chapter 9

Image analysis basics

Contents (class version)

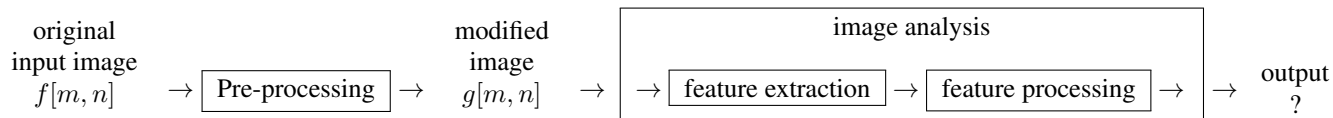
| | |
|---|-------------|
| 9.0 Introduction to image analysis | 9.2 |
| 9.1 Edge detection basics | 9.4 |
| Overview | 9.5 |
| Gradient-based methods | 9.6 |
| Canny edge detector | 9.18 |
| 9.2 Edge detection - 2nd derivatives and other methods | 9.21 |
| Laplacian-based methods | 9.21 |
| Marr and Hildreth methods | 9.26 |
| Parametric methods (signal modeling) | 9.28 |
| Texture segmentation | 9.29 |
| 9.3 Corner detection | 9.30 |
| 9.4 Summary | 9.33 |

9.0 Introduction to image analysis

Having established many of the fundamental tools needed for image processing, we can now turn applying these tools to the processing of images. This chapter focuses on **image analysis** methods. Most image analysis techniques extract features from images that are then used for subsequent “higher level” computer processing.

Entire books are devoted to the topic of **image analysis** [1–5]. We highlight only a few aspects here.

A coarse block diagram for image analysis is the following.



The final output is application dependent, so it can be difficult to form general theoretical frameworks here. Nevertheless, many image analysis methods are very useful in practical applications. For example, the features of interest might be the location of the edges of a tumor in a medical image such as an X-ray CT scan, and the final processed output value might be the estimated volume of that tumor. This type of quantification is important when monitoring cancer progression or treatment. In satellite-based remote sensing, the goal might be to monitor deforestation by assessing the fraction of a region that is forested. Extracted features are used extensively in image analysis and computer vision, *e.g.*, for various object recognition tasks, as well as in medical imaging. (There is some overlap here with EECS 442/504/542.)

Here are some of the MATLAB commands for image analysis under `help images`. For demonstrations, see `iptdemos`.

```
...
Image analysis.
  bwboundaries - Trace region boundaries in binary image.
  corner       - Find corners in intensity image.
  edge         - Find edges in intensity image.
  hough        - Hough transform.
  imgradient   - Find the gradient magnitude and direction of an image.
...
```

This chapter focuses on “non-random” aspects of image analysis (e.g., **edge detection** and **corner detection**). These are non-random in the sense that they typically are not based on statistical models, though the results of edge and corner detection certainly are influenced by (random) noise in the image and we will mention heuristic methods for reducing the influence of such noise.

Many image analysis methods are based explicitly on statistical tools, including **machine learning**. We postpone most statistical matters like texture segmentation until after introducing random processes in Ch. 11.

9.1 Edge detection basics

Motivation

Often much of the “information” in an image is in the edges of the various structures within the image. (B/W cartoon drawings are quite interpretable despite lack of internal intensity gradations!)

Edge detection is usually for the purpose of subsequent computer processing (e.g., house detection), rather than for human visual interpretation. Many useful algorithms start with an edge map and go from there.



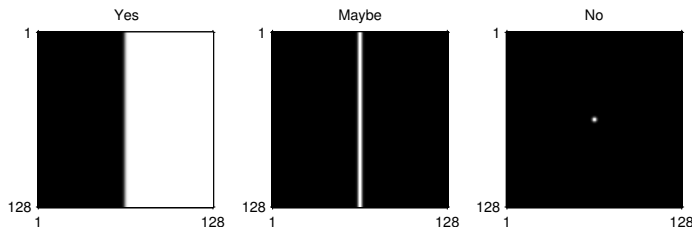
Example. **Image segmentation**, **object recognition**, **object detection**, object localization.

Example. For an illustration of how to *avoid* face recognition, see [this interesting article](#).

The definition of **edge** depends on the application. Roughly an edge corresponds to a “significant” change in image intensity.

Example. Let $g(x, y)$ denote a smooth 2D bump function like a gaussian.

- $g(x, y) ** \text{step}(x)$ has an obvious edge.
- $g(x, y) ** \delta(x)$ is like a “line,” which might be called an edge depending on the application.
- $g(x, y) ** \delta_2(x, y)$ would usually not be said to have edges. (Otherwise every noisy pixel in the image might be called an edge.)



Overview of edge detection

Interestingly, much of the edge detection literature uses derivations based on continuous-space images, with discretization left as a detail for implementation. A typical block diagram of an edge detection method is:

$$\underbrace{f(x, y)}_{\text{image}} \rightarrow \boxed{\text{edge detection system}} \rightarrow \underbrace{e(x, y)}_{\text{edge map}} = \begin{cases} 1, & (x, y) \in \text{edge} \\ 0, & \text{otherwise.} \end{cases}$$

Edge detection often is implemented as two or more steps:

$$\underbrace{f(x, y)}_{\text{image}} \rightarrow \underbrace{\boxed{\text{noise smoothing}} \rightarrow \boxed{\text{edge enhancer}} \rightarrow \boxed{\text{edge localizer}} \rightarrow \boxed{\text{thinning}}}_{\text{edge detection}} \rightarrow \underbrace{e(x, y)}_{\text{edge map}} .$$

Basic steps

- Noise smoothing (hopefully reducing noise without blurring edges)
- Edge enhancement (use a filter whose output local extrema are large near edges; many examples will follow)
- Edge localization (determine which of the local extrema are edges and which are due to noise)
- **Edge thinning** (reduce thick “edges” to be just 1-pixel wide)

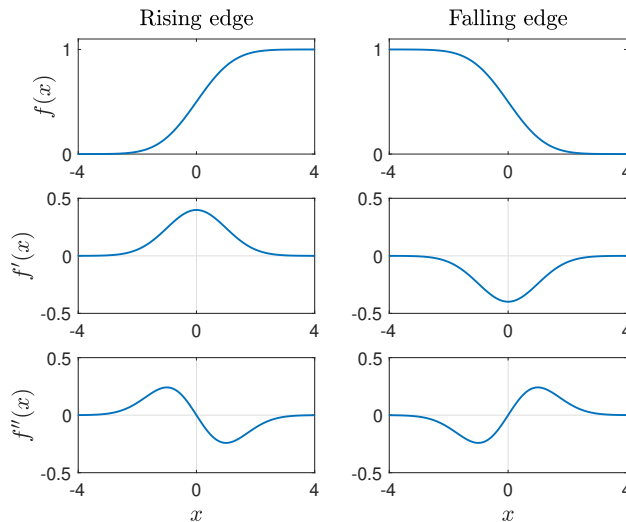
Often the noise smoothing the edge enhancement operations are LSI so they can be combined into a single filter.

Types of errors: missed edge points, spurious edge points, thick edges, displaced edges.

How do we quantify performance of an edge detection method? It is difficult to define general metrics. But if detected edges are used for a task like object classification, then we can use classification error rate as an overall metric.

Gradient-based methods

We initially focus on edge detection with 1D CS signals $f_a(x)$, and deal with 2D and DS images later. Sharp edges are easy to find, so we illustrate a blurry edge—the more challenging and interesting case. Because edges correspond to (possibly rapid) *changes* in image intensity, it is natural to examine the *derivatives* of $f_a(x)$.



This figure suggests a natural method for detecting “edges” in a 1D signal:

- Find locations where $|f'_a(x)|$ is large (exceeds some threshold) and where $f''_a(x) \approx 0$ (local extrema of derivative).

What is the “location” of the falling edge above? **??**

[RQ]

Practical derivatives in 1D

In practice we do not have a CS function $f_a(x)$, but rather its samples $f[n] = f_a(n\Delta_x)$. (We assume the ideal sampling model for now.) Recalling the definition of a derivative, one simple way to approximate the derivative is by a **finite difference**:

$$f'_a(x) \Big|_{x=n\Delta_x} \approx \begin{aligned} & \frac{f[n] - f[n-1]}{\Delta_x} = f[n] * \frac{1}{\Delta_x} (\delta[n] - \delta[n-1]) && \text{left difference} \\ & \frac{f[n+1] - f[n]}{\Delta_x} && \text{right difference} \\ & \frac{f[n+1] - f[n-1]}{2\Delta_x} && \text{central difference (average)}. \end{aligned}$$

The latter is equivalent to a filter with the impulse response $h_3[n] = [\frac{1}{2} \ 0 \ \frac{-1}{2}]$ if $\Delta_x = 1$.

What kind of filter is the central difference?

For $\Delta_x = 1$, it has frequency response (see figure below):

$$H_3(\Omega) = \frac{1}{2} e^{i\Omega} - \frac{1}{2} e^{-i\Omega} = i \sin(\Omega).$$

What is the frequency response of an “ideal” derivative operator? Recall FT derivative property:

$$f'_a(x) \xleftrightarrow{\mathcal{F}} i2\pi\nu F_a(\nu).$$

So if $g_a(x) = f'_a(x)$ and $g[n] = g_a(n\Delta_x)$ then for $\Delta_x = 1$, assuming $f_a(x)$ is suitably bandlimited:

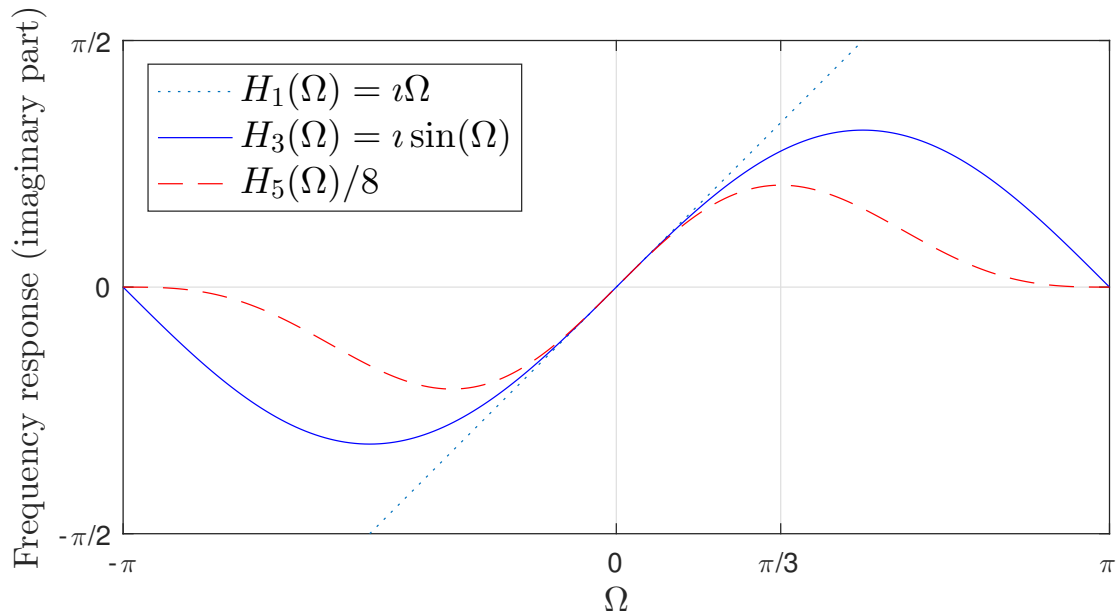
$$g[n] \xleftrightarrow{\text{DSFT}} G(\Omega) = G_a(\Omega/2\pi)_{(2\pi)} = (i\Omega F_a(\Omega/2\pi))_{(2\pi)} = (i\Omega)_{(2\pi)} F(\Omega),$$

where $F(\Omega)$ is the DTFT of the samples $f[n]$. In other words, the frequency response of an ideal 1D discrete-space differentiator

$h_1[n]$ is $\boxed{H_1(\Omega) = (i\Omega)_{(2\pi)}}$. This is a kind of high-boost filter (it boosts the amplitudes of higher frequency components, since $|H_1(\Omega)| = |\Omega|$), so noise would be amplified if this filter were used directly.

Example. MATLAB's `imgradientxy` function uses a filter having impulse response $h_5[n] = [1 \ 2 \ 0 \ -2 \ -1]$. What is the corresponding frequency response? $H_5(\Omega) = \imath(4 \sin(\Omega) + 2 \sin(2\Omega))$

This figure compares the ideal filter frequency response $H_1(\Omega) = \imath\Omega$ to that of the central difference filter $h_3[n]$ above and to that of $h_5[n]$ used in MATLAB. They all match near DC. Note that by Taylor expansion of \sin around $\Omega = 0$, we see that $H_5(\Omega) \approx \imath(4\Omega + 2(2\Omega)) = \imath 8\Omega$. So I divide by 8 in the plot because $H_5(\Omega)/8 \approx \imath\Omega$ for $\Omega \approx 0$.



Derivatives of band-limited signals**(skim)**

Now we elaborate further on what happens in 1D if $f_a(x)$ is suitably bandlimited, with samples $f[n] = f_a(n\Delta_x)$. Then

$$f_a(x) = \sum_{k=-\infty}^{\infty} f[k] \operatorname{sinc}\left(\frac{x - k\Delta_x}{\Delta_x}\right) \quad (9.1)$$

so its derivatives are

$$f'_a(x) = \sum_{k=-\infty}^{\infty} f[k] \frac{d}{dx} \operatorname{sinc}\left(\frac{x - k\Delta_x}{\Delta_x}\right) = \sum_{k=-\infty}^{\infty} f[k] \frac{1}{\Delta_x} d_1\left(\frac{x - k\Delta_x}{\Delta_x}\right)$$

$$f''_a(x) = \sum_{k=-\infty}^{\infty} f[k] \frac{d^2}{dx^2} \operatorname{sinc}\left(\frac{x - k\Delta_x}{\Delta_x}\right) = \sum_{k=-\infty}^{\infty} f[k] \frac{1}{\Delta_x^2} d_2\left(\frac{x - k\Delta_x}{\Delta_x}\right),$$

where, letting $y = \pi x$, so $dy = \pi dx$, we have

$$d_1(x) \triangleq \frac{d}{dx} \operatorname{sinc}(x) = \begin{cases} \pi \frac{y \cos y - \sin y}{y^2}, & x \neq 0 \\ 0, & x = 0 \end{cases} \quad h_1[n] = d_1(x) \Big|_{x=n} = \begin{cases} (-1)^n/n, & n \neq 0 \\ 0, & n = 0, \end{cases}$$

$$d_2(x) \triangleq \frac{d^2}{dx^2} \operatorname{sinc}(x) = \begin{cases} \pi^2 \frac{y(2-y^2) \sin y - 2y^2 \cos y}{y^4}, & x \neq 0 \\ -\pi^2/3, & x = 0. \end{cases} \quad h_2[n] = d_2(x) \Big|_{x=n} = \begin{cases} -2(-1)^n/n^2, & n \neq 0 \\ -\pi^2/3, & n = 0, \end{cases}$$

where $h_1[n] = [\dots -1/2 \ 1 \ 0 \ -1 \ 1/2 \ \dots]$ and $h_2[n] = [\dots \ 2 \ \underline{-\pi^2/3} \ 2 \ \dots]$.

So for band-limited signals the ideal *discrete-space* filters for estimating $f'_a(x)$ and $f''_a(x)$ at $x = n\Delta_x$ would have impulse responses $h_1[n]/\Delta_x$ and $h_2[n]/\Delta_x^2$ respectively, because using (9.1):

$$f'_a(x) \Big|_{x=n\Delta_x} = \sum_{k=-\infty}^{\infty} f[k] \frac{1}{\Delta_x} d_1 \left(\frac{n\Delta_x - k\Delta_x}{\Delta_x} \right) = \frac{1}{\Delta_x} \sum_{k=-\infty}^{\infty} f[k] h_1[n-k] = \frac{1}{\Delta_x} f[n] * h_1[n].$$

Because $h_1[n]$ and $h_2[n]$ are not FIR filters, they are used rarely. They are illustrated on the next page.

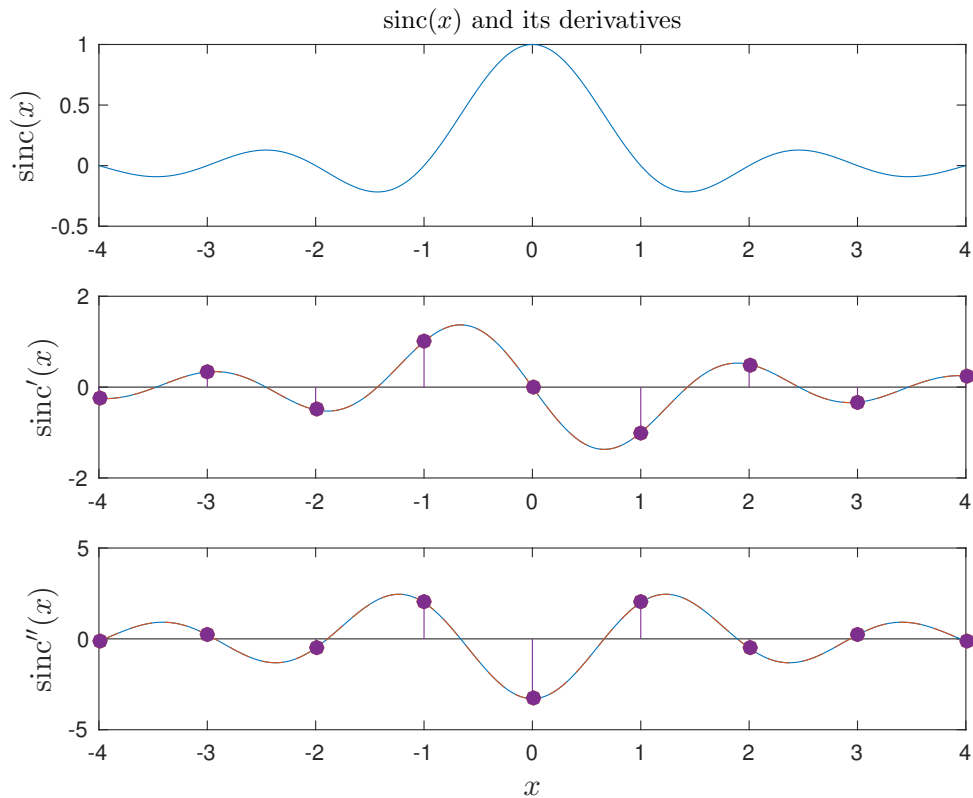
What is the frequency response of the filter $h_1[n]$?

$h_1[n]$ is samples of $\frac{d}{dx} \text{sinc}(x) \xleftrightarrow{\mathcal{F}} i2\pi\nu_x \text{rect}(\nu_x)$, so $H_1(\Omega) = (i\Omega)_{(2\pi)}$ See earlier picture.

These are noise amplifying filters, which also limits their utility.

What is the frequency response of the filter $h_2[n]$? **??**

[RQ]



Practical derivatives for polynomial models _____ (skim) ◆◆

The left difference $f'_a(x)|_{x=n\Delta} \approx \frac{f(n\Delta) - f((n-1)\Delta)}{\Delta}$ is in fact exactly the derivative if $f_a(x)$ happens to be linear over the interval $[(n-1)\Delta, n\Delta]$, because two points determine a line.

What if we would like a second-order polynomial approximation?

Because three points determine a parabola, we can make a locally *quadratic* fit to $f(x)$ using the pairs $(n-1, f[n-1])$, $(n, f[n])$, and $(n+1, f[n+1])$, where $f[n] = f(n\Delta)$, and then compute the derivative of this parabola at $x = n\Delta$.

This process is shift-invariant, so we can just consider the case $n = 0$. Simple calculus shows that

$$f(x) = f_0 + a \frac{x}{\Delta} + b \left(\frac{x}{\Delta} \right)^2$$

where $f_1 = f_0 + a + b$ and $f_{-1} = f_0 - a + b$ so

$$b = \frac{f_1 + f_{-1}}{2} - f_0, \quad \& \quad a = \frac{f_1 - f_{-1}}{2}.$$

The quadratic fit to the three points is thus

$$f(x) = f_0 + \frac{f_1 - f_{-1}}{2} \frac{x}{\Delta} + \left[\frac{f_1 + f_{-1}}{2} - f_0 \right] \left(\frac{x}{\Delta} \right)^2,$$

the derivatives of which are

$$f'(x) = \frac{1}{\Delta} \frac{f_1 - f_{-1}}{2} + [f_1 + f_{-1} - 2f_0] \frac{x}{\Delta^2}, \quad \& \quad f'(0) = \frac{1}{\Delta} \frac{f_1 - f_{-1}}{2}, \quad \& \quad f''(x) = \frac{1}{\Delta^2} [f_1 + f_{-1} - 2f_0].$$

Thus this strategy is equivalent to convolving with filters having the following impulse responses:

$$\boxed{h_1[n] = \frac{1}{2\Delta} [1 \quad \underline{0} \quad -1]} \quad \& \quad \boxed{h_2[n] = \frac{1}{\Delta^2} [1 \quad \underline{-2} \quad 1]}$$

for the 1st and 2nd derivatives respectively. This analysis provides another interpretation of the central difference approximation.

2D case

The natural generalization of derivative-based edge detection methods to 2D uses the **gradient** of the image, defined by

$$\nabla f(x, y) = \left[\frac{\partial}{\partial x} f(x, y) \quad \frac{\partial}{\partial y} f(x, y) \right],$$

where $\frac{\partial}{\partial x} f(x, y)$ is the “slope in the x direction.”

More generally, $\left[\frac{\partial}{\partial x} f(x, y) \quad \frac{\partial}{\partial y} f(x, y) \right] \begin{bmatrix} \cos \theta \\ \sin \theta \end{bmatrix}$ gives the “slope” in direction θ .

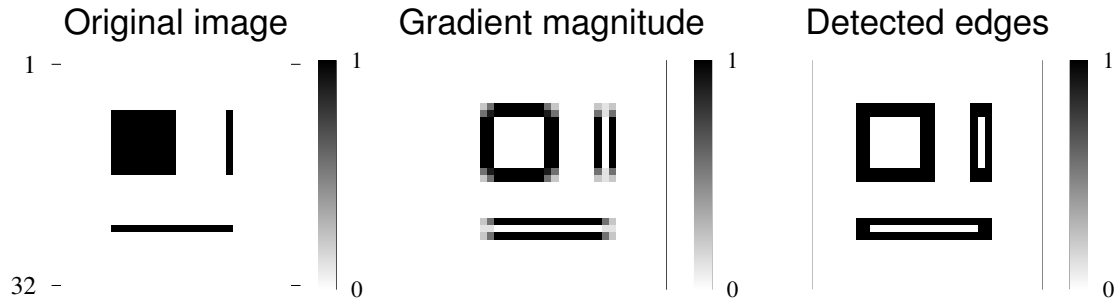
By the Cauchy-Schwarz inequality, the slope in the direction of maximum change is given by the **gradient magnitude**:

$$|\nabla f(x, y)| = \sqrt{\left(\frac{\partial}{\partial x} f(x, y) \right)^2 + \left(\frac{\partial}{\partial y} f(x, y) \right)^2}. \quad (9.2)$$

This suggests a simple gradient-based **edge-detection** method using a threshold:

$$e(x, y) = \begin{cases} 1, & |\nabla f(x, y)| > c \\ 0, & \text{otherwise,} \end{cases} \quad (9.3)$$

for some threshold c . Unfortunately, this approach yields edges as “strips” rather than “lines,” so some method of **edge thinning** is required (*cf.* 1D case), as illustrated in this example figure.



One method for edge thinning checks to see if $|\nabla f(x, y)|$ has a **local maximum** along at least one of the x or y directions [6, Prob. 8.17].

Exercise. Apply 1D central difference to $\text{step}(n)$ to see why thinning is needed.

class/team

??

Practical 2D derivatives

In 2D, given samples $f[m, n] = f(m\Delta_x, n\Delta_y)$, we can again use the **central difference**, for example, to approximate each of the derivatives, *e.g.*:

$$\left. \frac{\partial}{\partial x} f(x, y) \right|_{x=m\Delta_x, y=n\Delta_y} \approx \frac{f[m+1, n] - f[m-1, n]}{2\Delta_x},$$

and similarly for $\frac{\partial}{\partial y}$. For $\Delta_x = 1$, this is equivalent to 1D convolution (along n) with the impulse response $h_x[n] = [1/2, 0, -1/2]$. Again, this is a kind of “high-boost” filter, so noise will be amplified. In 2D, one way to reduce noise is to combine this filter with a low-pass filter in the perpendicular direction, forming a **separable** filter, such as the **Prewitt** filter (an option to `imggradientxy`):

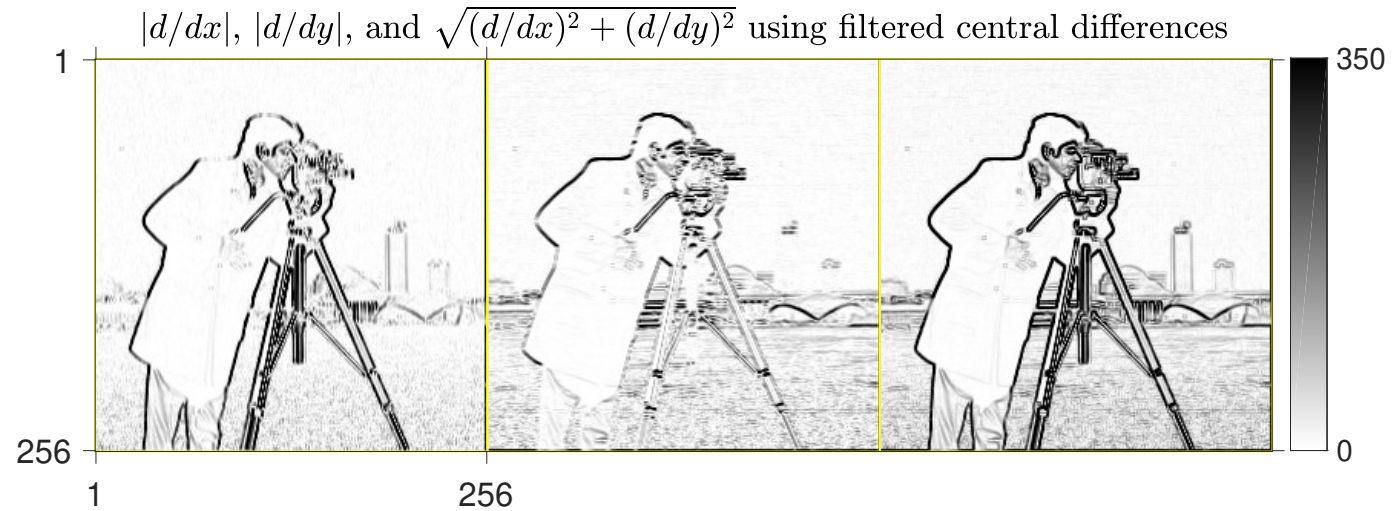
$$h[m, n] = h_x[m] h_y[n], \text{ where } h_y[n] = [2 \ 2 \ 2], \text{ so } h[m, n] = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}. \quad (9.4)$$

The frequency response of this separable filter is

$$H(\Omega_1, \Omega_2) = \frac{e^{i\Omega_1} - e^{-i\Omega_1}}{2} (1 + 2 \cos \Omega_2) = \underbrace{(i \sin \Omega_1)}_{\text{“derivative”}} \underbrace{(1 + 2 \cos \Omega_2)}_{\text{low-pass}}.$$

Why is the low-pass filter part not normalized to be unity at DC? **Unimportant because threshold in (9.3) is arbitrary.**

The following figure illustrates this type of edge detection.



There are many such approximations to the derivative, each with an associated name. MATLAB's `edge` command has at least six different edge-finding methods!

For example, in **Sobel's edge detection method** one compares the gradient magnitude

$$|\nabla f(x, y)| \Big|_{x=m\Delta_X, y=n\Delta_Y} \approx \sqrt{(f_x[m, n])^2 + (f_y[m, n])^2} \quad (9.5)$$

with a threshold, where the **Sobel operators** are

$$f_x[m, n] \triangleq f[m, n] ** h_x[m, n] \quad \text{and} \quad f_y[m, n] \triangleq f[m, n] ** h_y[m, n]$$

$$h_x[m, n] \triangleq \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad h_y[m, n] \triangleq h_x[n, m] = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}. \quad (9.6)$$

What is the frequency response of $h_x[m, n]$?

class/team

What might be preferable about (9.6) over (9.4)?

class/team

MATLAB's edge also has the **Prewitt** method and the **Roberts** method.

All are **high-boost** filters, with zero DC response.

Do you see any benefits to working in 2D instead of 1D?

In 1D, if we want to reduce noise (in a LSI way) we will have to apply some low-pass filtering along the same (1D) direction where we are looking for an edge, and that blurring degrades the edge at the same time it reduces noise.

In 2D, we can smooth *along* the edge, *i.e.*, in the opposite direction of the change we seek, thereby reducing noise with less blurring of the edges of interest, at least for edges that are perfectly horizontal or vertical.

A HW problem shows that (in continuous space) the gradient magnitude (9.2) is rotation invariant.

Challenge. How rotation invariant is (9.5) with (9.6)?

Canny edge detector*(skim)* ♦♦

In 1986, John Canny proposed the following formalism for edge detection [7], now called the **Canny edge detector**.

- Formulate a mathematical model for the edges of interest and the noise.
This approach is logical because presumably the edge detection method should depend on what type of “edges” are of interest: step edges, erf edges, ridge edges, roof edges...
- Formulate performance criteria that quantify desirable properties of the edge detector.
This formulation allows objective comparisons of different methods.
- Use optimization methods to determine the best filter (over some class, such as linear methods followed by a threshold) according to the criteria.

This approach sounds promising, but some ad hoc steps crept in along the way, so we skip the derivation. Nevertheless, the method is popular.

A long derivation led to the following filter called the **derivative of a Gaussian (DOG)**

$$h(x) = \frac{-x}{\chi^2} \exp(-x^2/2\sigma^2), \quad (9.7)$$

which was within 20% of the optimal filter performance.

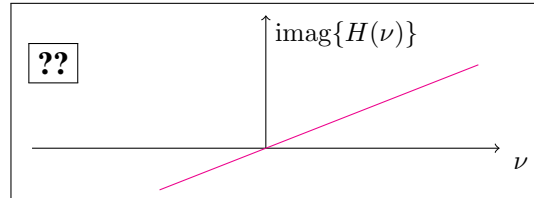
Caution: the DOG acronym also can mean **difference of Gaussians**.

What is the frequency response of this filter? Derivative in space domain corresponds to multiplication by $i2\pi\nu$ in the frequency

domain, and the FT of a Gaussian is a Gaussian. So for some nonnegative constants α and β :

(draw in class)

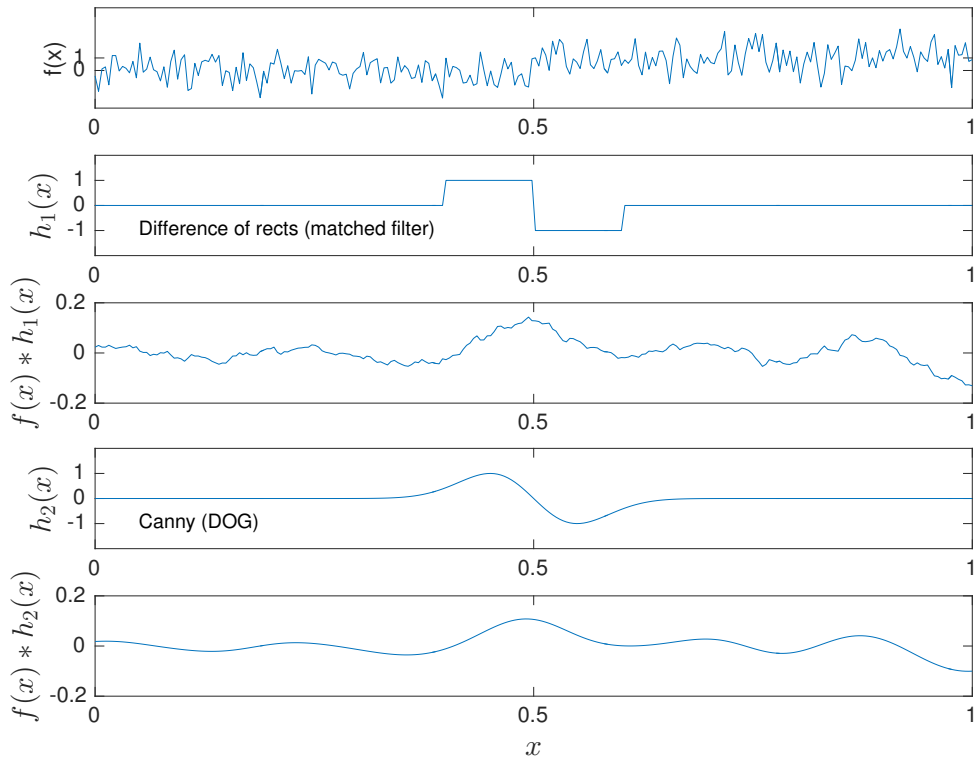
$$H(\nu) = i\alpha\nu e^{-\beta\nu^2}$$



There is a localization-detection trade-off. Filter spatial scale must compromise between localization and detection criteria.

Discuss differentiation property of convolution and convolving Gaussian filters repeatedly for scale.

Example. This is a very noisy version of a step function $\text{step}(x - 0.5)$. Note that the DOG filter output has a peak that is less noisy than the output using $h_1(x)$, = a difference of rect functions, but the DOG output is also less sharply peaked.



Demo: <http://imagedatabase.cs.washington.edu/demo/>

9.2 Edge detection - 2nd derivatives and other methods

Laplacian-based methods

(Also available in MATLAB's edge command.)

From 1D figure above, locations where the second derivative has a zero crossing should correspond to edges. (This approach ignores the first derivative.) The generalization of the second derivative to 2D functions is the **Laplacian** given by

$$\nabla^2 f(x, y) = \frac{\partial^2}{\partial x^2} f(x, y) + \frac{\partial^2}{\partial y^2} f(x, y) \xrightarrow{\mathcal{F}_2} (i2\pi\nu_x)^2 F(\nu_x, \nu_y) + (i2\pi\nu_y)^2 F(\nu_x, \nu_y) = -(2\pi\rho)^2 F(\nu_x, \nu_y).$$

The Laplacian or **Laplace operator** is sometimes denoted $\nabla \cdot \nabla f$ or Δf . The notation ∇^2 is also used for the **Hessian matrix** in optimization problems so one must determine which is meant from context.

How can we compute (approximately) $\frac{\partial^2}{\partial x^2} f(x, y)$ for discrete-space images?

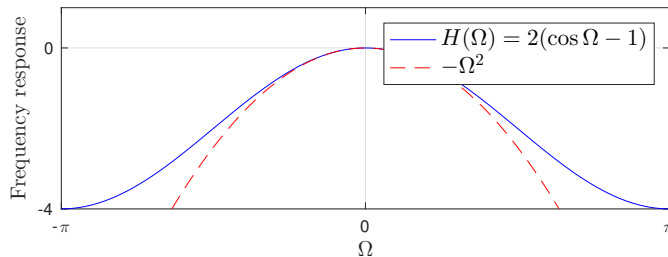
Because convolving with $[1 \ -1]$ approximates a first derivative, convolving again with $[1 \ -1]$ should approximate a second derivative:

$$h[n] = [1 \ -1] * [1 \ -1] = [1 \ -2 \ 1],$$

(cf. samples of ideal sinc-based 2nd derivative!), which has frequency response

$$H(\Omega) = 2(\cos \Omega - 1) \approx -\Omega^2 \text{ for } \Omega \approx 0.$$

(Recall $\cos \Omega \approx 1 - \Omega^2/2$ for $\Omega \approx 0$ by Taylor series.)



A 2D version of this filter (because the Laplacian is the sum of the derivatives with respect to x and y) corresponds to the following **discrete Laplacian operator**:

$$h_2[m, n] = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

which has frequency response $H_2(\Omega_1, \Omega_2) = 2(\cos \Omega_1 + \cos \Omega_2 - 2) \approx -(\Omega_1^2 + \Omega_2^2)$ for $\Omega_1^2 + \Omega_2^2 \approx 0$.

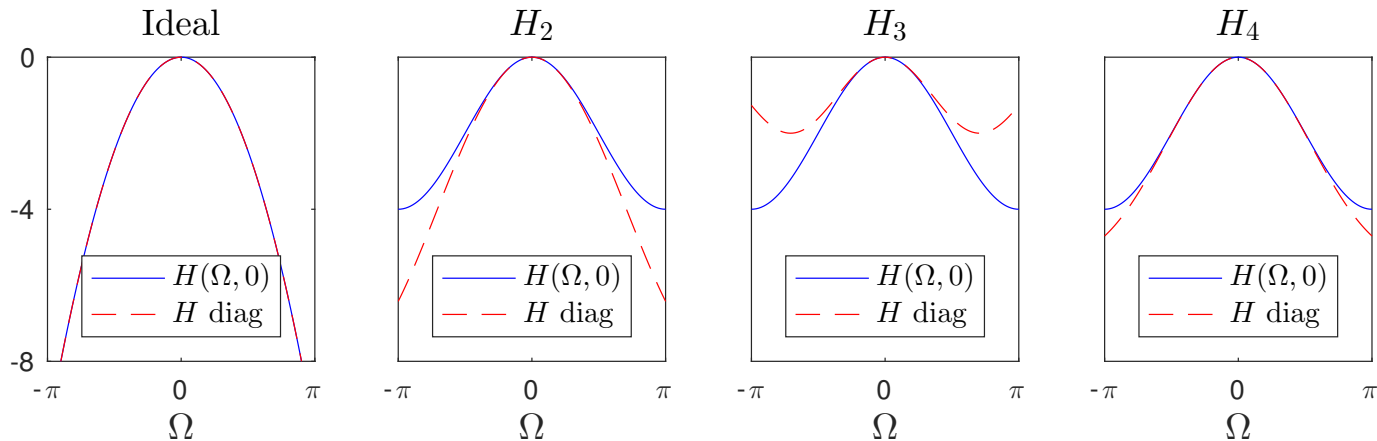
Many other variations of **discrete-space Laplacian filters** are possible, e.g., [9] [10, p. 131, p. 202]:

$$h_3[m, n] = \frac{1}{2} \begin{bmatrix} 1 & 0 & 1 \\ 0 & -4 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \quad h_4[m, n] = \frac{1}{6} (4h_2[m, n] + 2h_3[m, n]) = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{bmatrix}.$$

Note that $h_4[m, n] = \frac{1}{6} (h[n] h[m] - 36 \delta_2[m, n])$ where $h[n] = [1 \ 4 \ 1]$. This form facilitates practical implementation.

Exercise. Determine $H_3(\Omega_1, \Omega_2)$ and $H_4(\Omega_1, \Omega_2)$. Show that $H_3(\Omega_1, \Omega_2) \approx H_4(\Omega_1, \Omega_2) \approx -(\Omega_1^2 + \Omega_2^2)$. (do in class)

The following figure shows horizontal $H(\Omega, 0)$ and diagonal $H(\Omega/\sqrt{2}, \Omega/\sqrt{2})$ profiles through the frequency responses of the ideal Laplacian filter and the above practical 3×3 filters. Because of transpose symmetry of these filters, $H(\Omega, 0) = H(0, \Omega)$.



Which filter (H_2 or H_3 or H_4) seems the best and why? **??**

Any trade-off? **??**

[RQ]

(class / pair)

Advantages of Laplacian-based methods:

- Zero crossing points tend to be “continuous.”
- Edge thinning is not needed.

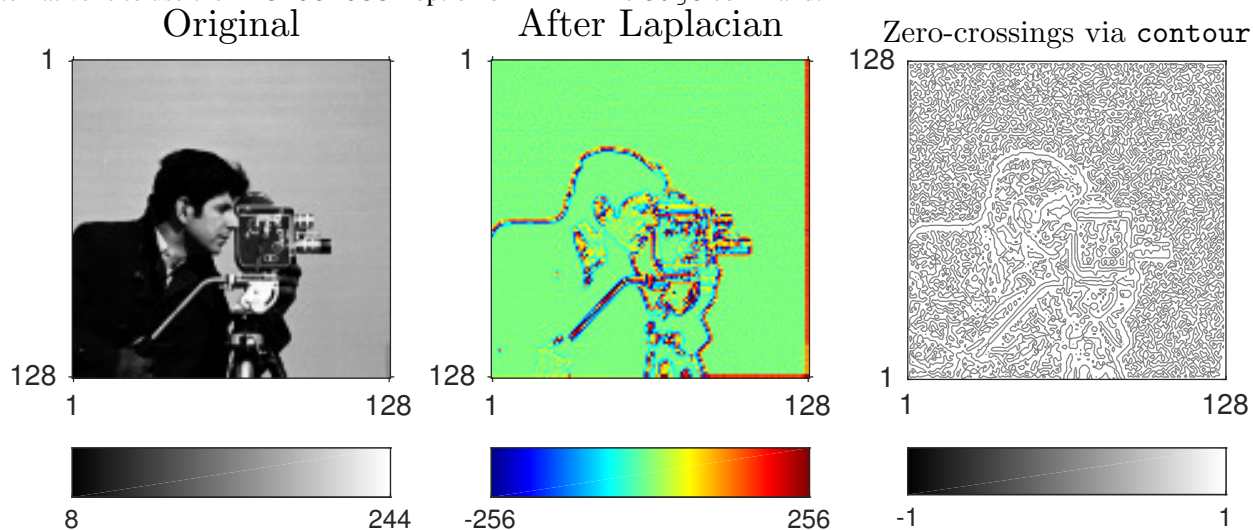
Disadvantages:

- Sensitive to noise. Why? ??
May require preprocessing to reduce noise.
- Numerous false edges, such as in nearly uniform regions.

(class / pair)

Example. Apply Laplacian filter and then use `contour` command to find zero crossings (cf. [6, Fig. 8.33]).

An alternative is to use the `'zerocross'` option of MATLAB's `edge` command.



Notice the many spurious zero crossings in nearly uniform image regions. One solution to this problem is to use only zero crossings where **local variance** of $f[m, n]$ is sufficiently large, *i.e.*, not in nearly uniform regions [6, Fig. 8.35].

This approach is an example of **two channel** methodology: one channel related to local variance, the other related to Laplacian. Such methods will be discussed further in Ch. 10.

Marr and Hildreth methods


Marr and Hildreth, computer vision experts, argued that we should separate edges at different scales to improve subsequent image analysis. (This can help address the spurious edge problem.)

By filtering the image with various low-pass filters prior to applying an edge detection method, we can remove “small” details and just leave the “large” scale variations for the edge detector to find, where “large” and “small” depend on the amount of filtering.

A typical choice (due to its optimality in terms of space-bandwidth product, as discussed in Ch. 2) is a Gaussian filter:

$$h_l(x, y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)} \xleftrightarrow{\mathcal{F}_2} H_l(\nu_x, \nu_y) = e^{-\pi(\rho\sigma\sqrt{2\pi})^2},$$

where σ is related to the width of the PSF.

If the subsequent edge detection method is Laplacian based, then we can combine the filtering with the Laplacian:

$$\nabla^2 [f(x, y) ** h_l(x, y)] = f(x, y) ** \nabla^2 h_l(x, y) = f(x, y) ** h(x, y),$$

due to the linearity of convolution, where one can show with some calculus that

$$h(x, y) = \nabla^2 h_l(x, y) = \boxed{\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} h_l(x, y)}. \quad (9.8)$$

This is called the **Laplacian of Gaussian (LOG)** filter. Note that it is not **separable**, but it is **radially symmetric**.

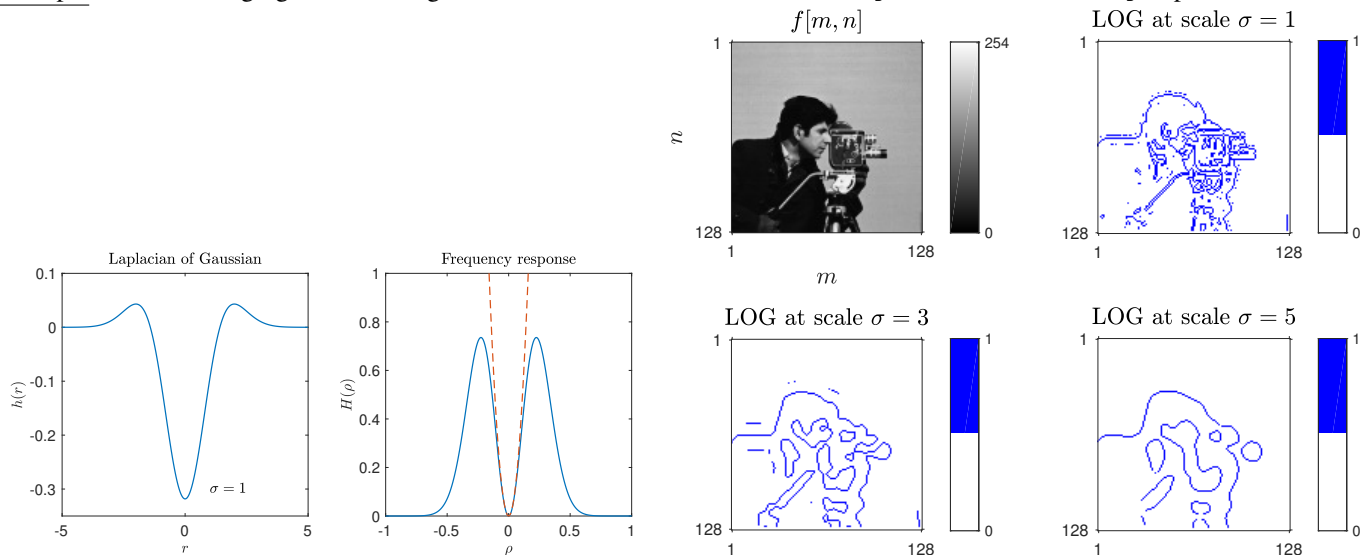
Frequency response? By the differentiation properties of the CSFT:

$$H(\nu_x, \nu_y) = [(i2\pi\nu_x)^2 + (i2\pi\nu_y)^2] H_l(\nu_x, \nu_y) = -(2\pi\rho)^2 e^{-\pi(\rho\sigma\sqrt{2\pi})^2}, \text{ (see next figure)}$$

which is a bandpass filter with maximum magnitude at $\rho_{\max} = \frac{1}{2\pi\sigma\sqrt{2}}$.

Note that for $\Delta_x = 1$, so that $\Omega = \sqrt{\Omega_1^2 + \Omega_2^2} = 2\pi\rho$, the corresponding 2D digital filter has circularly symmetric frequency response $H(\Omega) = -\Omega^2 H_l(\frac{\Omega}{2\pi})$, for $|\Omega| \leq \pi$, which is essentially $-\Omega^2$ but with attenuated high spatial frequencies.

Example. The following figure shows edges at various scales, from MATLAB's edge command with 'log' option.



The left figure shows the impulse response $h(r)$ of the LoG filter, and its magnitude frequency response $|H(\rho)|$ which approximates the "ideal" parabola ρ^2 shown as a dashed line in the plot.

Parametric methods (signal modeling)**(skim)** ♦♦

Traditional edge detection methods like those described above find the location of an edge only to within the nearest pixel, due to the pixelized nature of digital images. In some applications we would like to locate an edge more accurately, *i.e.*, to **subpixel** accuracy. Parametric methods can achieve this goal. See [6, Sect. 8.3.4] for an nice angiography example.

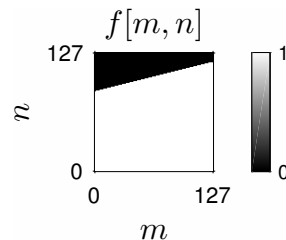
Suppose we observe a noisy image $g[m, n]$ that is a latent image $f[m, n]$ with additive noise $\varepsilon[m, n]$:

$$g[m, n] = f[m, n] + \varepsilon[m, n],$$

where $f[m, n] = f[m, n; a, b]$ is samples of an ideal analog image of an object having a straight edge:

$$f(x, y) = \begin{cases} 1, & 0 \leq y \leq ax + b, 0 \leq x \leq W \\ 0, & \text{otherwise,} \end{cases}$$

where a and b are unknown parameters describing the edge.



Example: measuring orientation of an object for subsequent robotic arm movement.

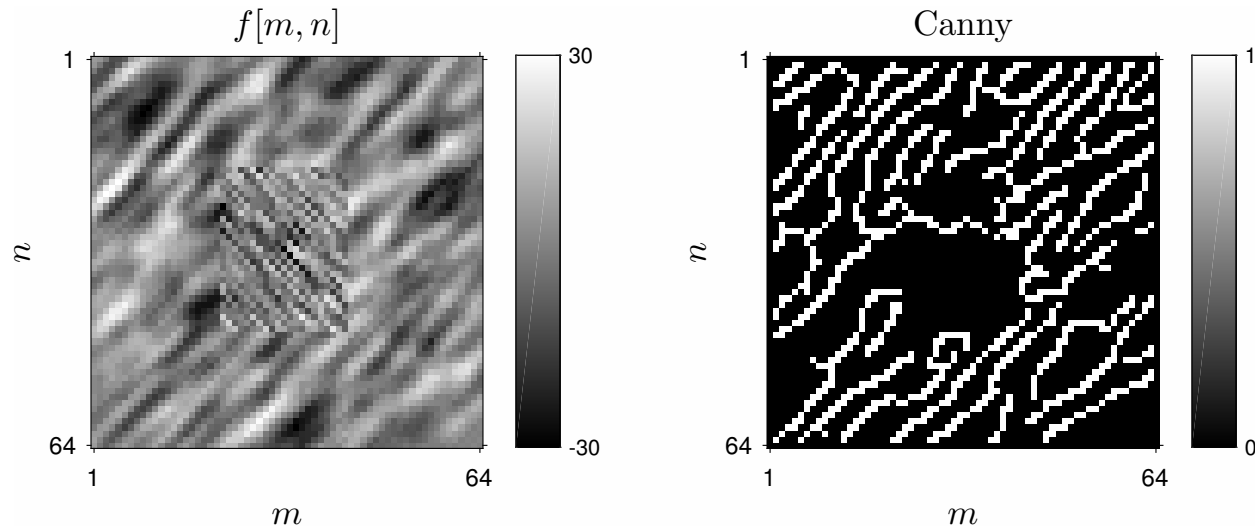
We can **estimate** the edge parameters a and b using the method of **least-squares**:

$$\min_{a, b} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} (g[m, n] - f[m, n; a, b])^2.$$

The resulting edge can certainly have subpixel accuracy. **Why? Because we are averaging over multiple (noisy) pixels.** In contrast, a conventional edge detection method can at best find the nearest pixel.

Texture segmentation**(skim)** ♦♦

So far we have considered edges to be changes in image *intensity*. Other types of “edges” are also possible, as illustrated by the following figure. Your eyes can see an “edge” here, but none of the preceding edge detection methods can!



In this case the two different regions both have an average of zero intensity, so intensity-based edge detection fails completely. Clearly other approaches are required to find edges that separate regions of different texture.

We may consider statistical methods for image segmentation later in the course if time permits [11].

9.3 Corner detection

Although edges capture very important information, experiments have shown that human recognition is better with just corner and junction information than with just edge information [12].

This property has inspired much research on **corner detectors** [13].

A famous example is the **Harris corner detector**. This detector uses the same types of derivatives we used for edge detection, but combines them differently. Let $f[m, n]$ be the original image. Use the derivative-of-a-Gaussian (DoG) filter $h_d[n]$, having width σ_d called the **differentiation scale**, along each direction:

$$\begin{aligned} f_x[m, n] &\triangleq h_d[n] * f[m, n] \quad (\text{derivative along } n) \\ f_y[m, n] &\triangleq h_d[m] * f[m, n] \quad (\text{derivative along } m). \end{aligned}$$

For each pixel location m, n , form the following 2×2 matrix:

$$\mathbf{C}_{m,n} = \begin{bmatrix} c_x[m, n] & c_{xy}[m, n] \\ c_{xy}[m, n] & c_y[m, n] \end{bmatrix} = \begin{bmatrix} c_x[m, n] & \triangleq b[m, n] ** f_x^2[m, n] \\ c_y[m, n] & \triangleq b[m, n] ** f_y^2[m, n] \\ c_{xy}[m, n] & \triangleq b[m, n] ** f_x[m, n] f_y[m, n] \end{bmatrix},$$

where $b[m, n]$ denotes a 2D Gaussian low-pass filter with width σ_I called the **integration scale**. Finally, compute the (two) eigenvalues of $\mathbf{C}_{m,n}$ for each pixel location. Locations where both eigenvalues are large correspond to corners.

Can we use Fourier analysis to better understand this operation? **Not easily because of the nonlinear squaring operation.**

To avoid computing eigenvalues, Harris proposed to use the determinant and the trace:

$$\text{cornerness}[m, n] \triangleq \det\{\mathbf{C}_{m,n}\} - 0.04 \text{trace}^2\{\mathbf{C}_{m,n}\}.$$

The MATLAB function `cornermetric` (called by `corner`) performs this computation.

Is the above corner detector invariant to image intensity DC shift, *i.e.*, $g[m, n] = c + f[m, n]$.

[RQ]

??

Is it shift invariant? ??

[RQ]

Is it invariant to intensity scaling, *i.e.*, $g[m, n] = \alpha f[m, n]$? ??

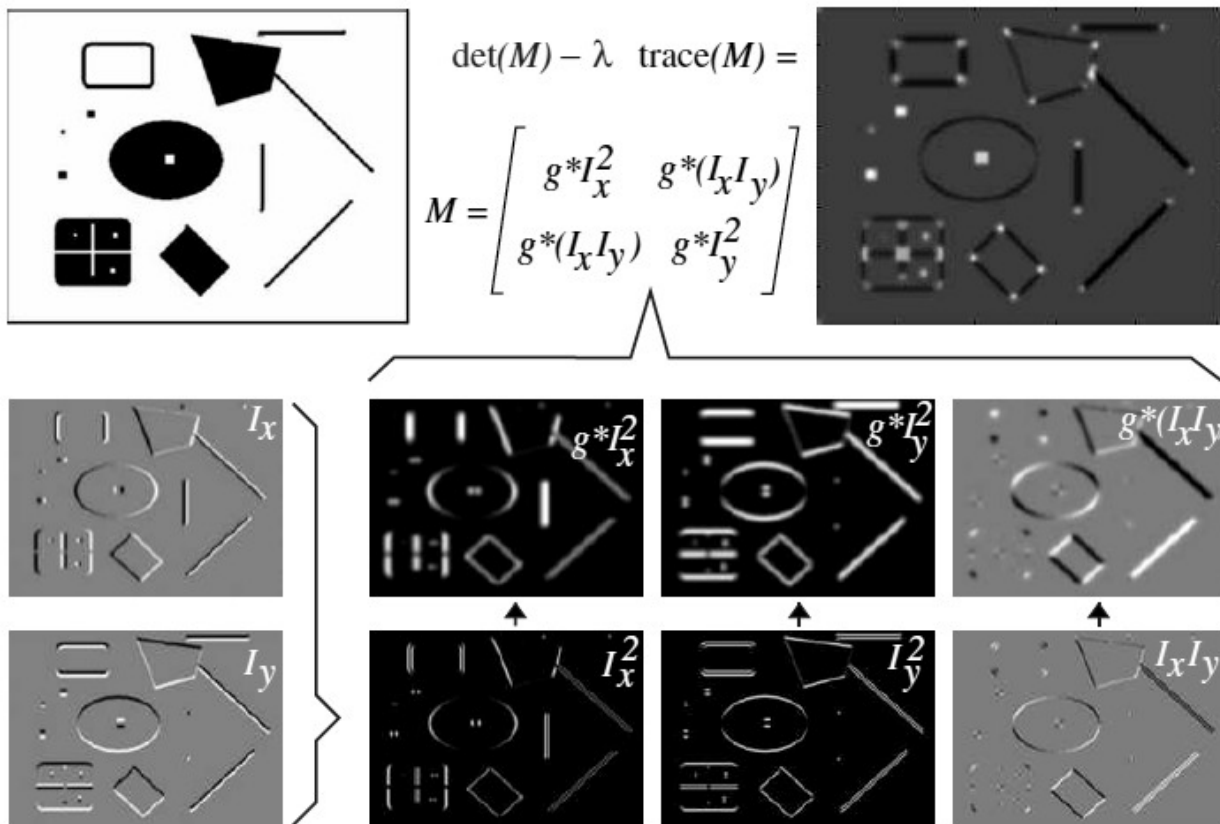
(do in class / pair)

[wiki] has a nice motivation for this method, that is most easily described in continuous space. Take an image patch and do a (Gaussian-weighted) sum-of-squared differences between the patch and a neighboring patch spaced (d_x, d_y) away:

$$s(d_x, d_y) \triangleq \iint w(x, y) (f(x, y) - f(x - d_x, y - d_y))^2 dx dy.$$

Applying a Taylor series of $f(x, y)$ shows that the dependence of this metric involves the 2×2 matrix above. The behavior of $s(d_x, d_y)$ differs for patches in uniform regions, along edges, or near corners.

Example. Taken from [12, Fig. 3.1]. The “M” in the following figure corresponds to $C_{m,n}$ and g corresponds to the blur $b[m, n]$. The formula [12, eqn. (3.4)] and the figure from that paper seem not to include the square of trace. I am unsure if that is a typo.



9.4 Summary

This chapter discussed the basic ideas of some image analysis methods. We were able to use Fourier analysis to gain insight into some of the methods, including edge detection. However, there are also nonlinearities (*e.g.*, thresholds) involved in many of these methods. These make a complete analysis quite difficult, so one usually resorts to empirical evaluation.

Often one chooses features that are invariant to various image properties such as pixel size and illumination [12].

The **scale-invariant feature transform (SIFT)** is a famous example [14].

Edge detection is a particularly important form of image analysis. Most methods use combinations of approximations to first and second derivatives. See also [15].

Bibliography

- [1] M. Sonka, V. Hlavac, and R. Boyle. *Image processing, analysis, and machine vision*. 2nd ed. Pacific Grove, CA: PWS Pub., 1999.
- [2] G. Sapiro. *Geometric partial differential equations and image analysis*. Cambridge: Cambridge University Press, 2001.
- [3] T. F. Chan and J. J. Shen. *Image processing and analysis: Variational, PDE, wavelet, and stochastic methods*. Soc. Indust. Appl. Math., 2005.
- [4] T. Svoboda, J. Kybic, and Václav Hlavác. *Image processing, analysis, and machine vision: A MATLAB companion*. ISBN: 0495295957. Thomson Learning, 2007.
- [5] A. A. Farag. *Biomedical image analysis*. Cambridge, 2014.
- [6] J. S. Lim. *Two-dimensional signal and image processing*. New York: Prentice-Hall, 1990. ISBN: 978-0139353222.
- [7] J. Canny. “A computational approach to edge detection”. In: *IEEE Trans. Patt. Anal. Mach. Int.* 8.6 (Nov. 1986), 679–98.
- [8] S. O. Rice. “Mathematical analysis of random noise”. In: *Bell Syst. Tech. J.* 23.3 (July 1944), 282–332.
- [9] T. Lindeberg. “Scale-space for discrete signals”. In: *IEEE Trans. Patt. Anal. Mach. Int.* 12.3 (Mar. 1990), 234–54.
- [10] W. E. Milne. *Numerical solution of differential equations*. 2nd ed. New York: Dover, 1970.
- [11] M. L. Comer and E. J. Delp. “The EM/MPM algorithm for segmentation of textured images: analysis and further experimental results”. In: *IEEE Trans. Im. Proc.* 9.10 (Oct. 2000), 1731–44.
- [12] T. Tuytelaars and K. Mikolajczyk. “Local invariant feature detectors: A survey”. In: *Found. & Trends in Comp. Graph. and Vision* 3.1 (2008), 1–110.
- [13] C. Harris and M. Stephens. “A combined corner and edge detector”. In: *Proc. of the 4th Alvey Vision Conference*. 1988, 147–51.
- [14] D. G. Lowe. “Object recognition from local scale-invariant features”. In: *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*. Vol. 2. 1999, 1150–7.
- [15] V. Torre and T. A. Poggio. “On edge detection”. In: *IEEE Trans. Patt. Anal. Mach. Int.* 8.2 (Mar. 1986), 147–63.

Chapter 10

Image enhancement

Contents (class version)

| | |
|--|--------------|
| 10.0 Introduction | 10.2 |
| 10.1 Contrast and dynamic range modification | 10.4 |
| Piecewise linear amplitude adjustment | 10.5 |
| Nonlinear monotone adjustment | 10.6 |
| Histogram-based methods for contrast enhancement | 10.7 |
| False color and pseudocolor | 10.16 |
| 10.2 Image sharpening | 10.17 |
| Highpass filtering and unsharp masking | 10.17 |
| Homomorphic processing | 10.20 |
| Adaptive modification of local contrast and local luminance mean | 10.21 |
| 10.3 Image denoising | 10.22 |
| Lowpass filtering | 10.25 |
| Median filtering | 10.26 |
| Out-range pixel smoothing | 10.29 |
| Adaptive image smoothing | 10.30 |
| Bilateral filter | 10.32 |
| 10.4 Summary | 10.35 |

10.0 Introduction

Another application of the fundamental tools developed thus far is **image enhancement** or **image editing**.

What does it mean to “enhance” an image? The purpose of **image enhancement** [1, Ch. 8] is to “improve” images in terms of

- the visual appearance for human interpretation [1, Ch. 7], or
- the suitability for subsequent computer processing.

$$\text{original input image} \rightarrow \boxed{\text{image enhancement}} \rightarrow \text{output (improved) image} \rightarrow \begin{cases} \text{eye} \\ \text{algorithm} \end{cases}$$

The meaning of “improved” is very application dependent, so it can be difficult to form general theoretical frameworks here. Furthermore, many of the methods are nonlinear, thereby precluding use of the analytical tools developed in previous chapters, although most basic enhancement methods are shift invariant.

So most image enhancement work is *qualitative*, one exception being **image denoising**. Indeed, Lim [1, p. 452] describes image enhancement as “an algorithm that is simple and ad hoc...”

What are typical ways in which one “enhances” an image?

Adjusting brightness and contrast, sharpening, noise smoothing, red eye removal, etc.

Somewhat related to image enhancement is the problem of **image restoration**, where we want to estimate an original image $f[m, n]$ given a related degraded image $g[m, n]$, as follows:

$$f[m, n] \rightarrow \boxed{\text{degradation}} \rightarrow g[m, n] \rightarrow \boxed{\text{restoration}} \rightarrow \hat{f}[m, n].$$

In this problem, our goal is to make $\hat{f}[m, n]$ as “close” to the original $f[m, n]$ as possible. We can quantify “close” and develop “optimal” methods. By comparison, in image enhancement we are often *not* trying to recover an original; instead we are deliberately modifying the image values to “improve” some characteristics, such as reducing degradations or accentuating certain features.

Overview

- Contrast / dynamic range adjustment
- Sharpening
- Noise smoothing (*ad hoc* methods; we will discuss model-based image de-noising methods later)

Here are some of the MATLAB commands for image enhancement under `help images`. For demos, see `iptdemos`.

...

Image enhancement.

```
histeq      - Enhance contrast using histogram equalization.
imadjust    - Adjust image intensity values or colormap.
medfilt2    - Perform 2-D median filtering.
ordfilt2    - Perform 2-D order-statistic filtering.
wiener2     - Perform 2-D adaptive noise-removal filtering.
```

...

Colormap manipulation.

...

Image enhancement operations include both

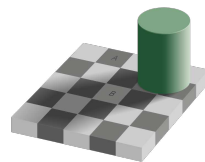
- point processing: operations on a single pixel at a time (memoryless), and
- neighborhood processing: operations groups of pixels.

10.1 Contrast and dynamic range modification

Most grayscale displays accept 8-bit inputs in the range 0-255. Often we cannot see image “details” because the displayed grayscale values of an image are “too similar.” In particular, the human visual system is nonlinear [1, Ch. 7], and responds differently to intensity contrast depending on the local background intensity.

Example. Here is an illustration of how challenging it can be to try to optimize images for human perception.

This **checker shadow illusion** was published by Edward H. Adelson of MIT in 1995.



Which patch has higher image intensity values: A or B? ??

[RQ]

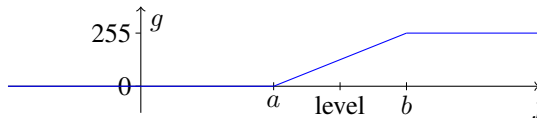
Often human perception of an image can be enhanced by a simple **memoryless** or **point-wise** intensity transformation $T : \mathbb{R} \rightarrow \mathbb{R}$

$$g[m, n] = T(f[m, n]).$$

Example. Brightness and contrast adjustments in digital photo software.

Piecewise linear amplitude adjustment

$$g[m, n] = \begin{cases} 0, & f[m, n] < a \\ \frac{255}{b-a}(f[m, n] - a), & a \leq f[m, n] \leq b \\ 255, & b < f[m, n] \end{cases}$$

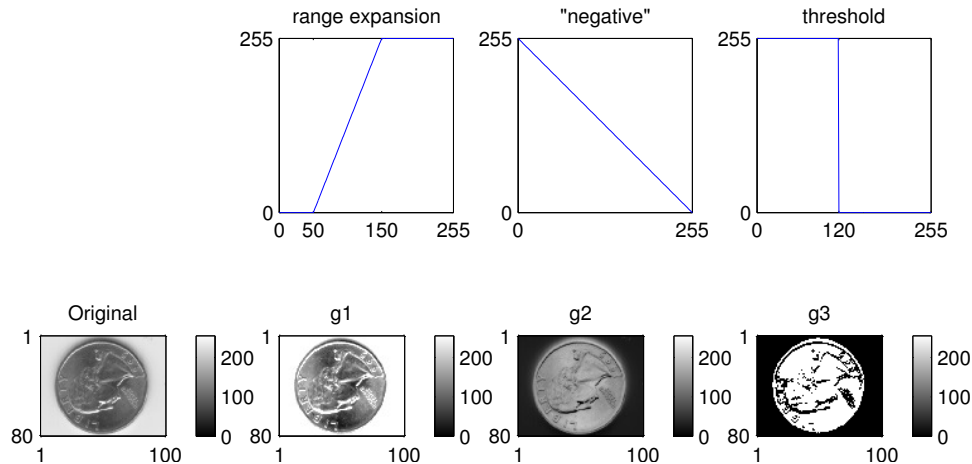


This process is often used to map an image to the range (0, 255) for display. Performed with MATLAB's `imagesc` function.

The `clim` option of `imagesc` adjusts a and b .

Alternate terminology: **window** = $b - a$ and **level** = $\frac{b+a}{2}$ used routinely in medical imaging.

Example.



Nonlinear monotone adjustment

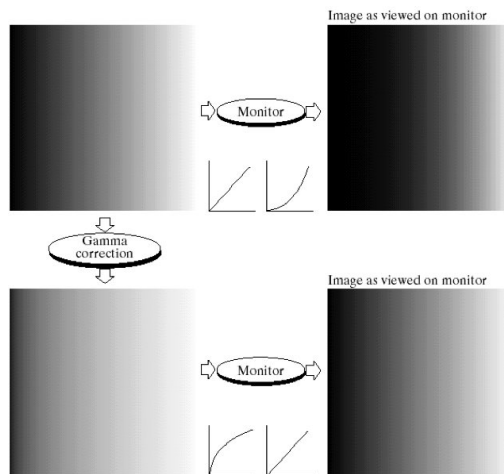
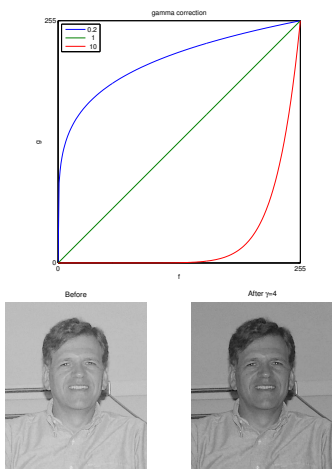
Such piecewise-linear adjustments are insufficient in many cases, and nonlinear operations can be helpful.

Example. To compensate for nonlinearities in I/O devices like photographic film, scanners, printers, monitors, and the eye, one often applies **gamma correction** to an image prior to display:

$$g[m, n] = (f[m, n])^\gamma,$$

where $\gamma \in \mathbb{R}$ is a tuning parameter. MATLAB's `cmggamma` function is related to this operation.

Example. $g[m, n] = 255 (f[m, n] / 255)^\gamma$



Histogram-based methods for contrast enhancement

One way to enhance the contrast of an image is to adjust its **histogram**.

Mathematical definition

Mathematically, given observations X_1, \dots, X_L , a **histogram** is a function that counts the number of observations that fall into each of K disjoint sets \mathcal{B}_k known as **bins**:

$$h_k = \underbrace{\sum_{l=1}^L \mathbb{I}_{\{X_l \in \mathcal{B}_k\}}}_{\text{\# of } X_i \text{ values in } k\text{th bin}}, \quad k = 1, \dots, K, \quad \sum_{k=1}^K h_k = L.$$

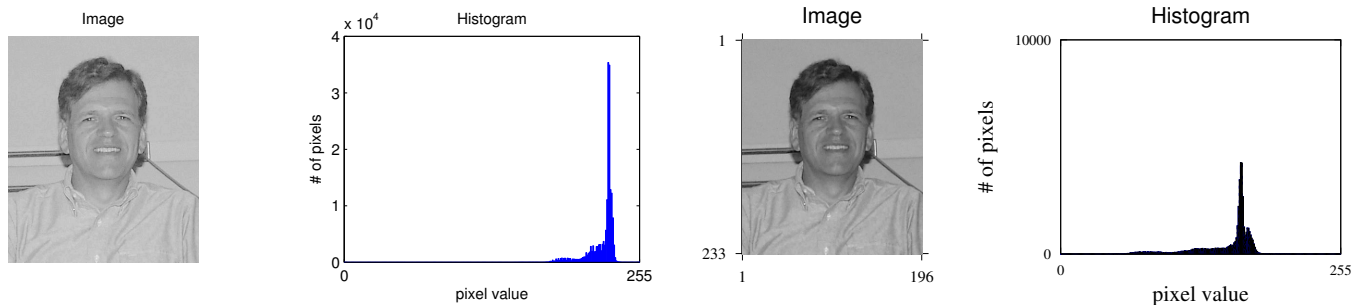
In 1D, usually the bins are disjoint intervals, e.g., $\mathcal{B}_k = (t_{k-1}, t_k]$.

This is what MATLAB's `hist` command does, choosing $K = 10$ bins by default.

MATLAB's `histc` command allows you to specify the bin “edges” as $[t_{k-1}, t_k)$

The graph of a histogram is one way to visualize a histogram.

Example. For an 8-bit gray-scale image $f[m, n]$ of size $M \times N$, typically $L = MN$ (total number of pixels), $K = 256$ and $\mathcal{B}_k = \{k - 1\}$ for $k = 1, \dots, 256$, so the image histogram represents how many pixels have values $0, 1, \dots, 255$. The following image has fairly poor contrast and its histogram is a clump of high values. Modifying the histogram can improve image appearance.



Statistical definition

In statistics, often observations $\{X_1, \dots, X_L\}$ are treated as **independent and identically distributed (IID) random variables** having **probability density function (pdf)** $f_X(x)$. Normalizing the histogram by L provides an estimate of the following probability:

$$\hat{p}_k = \frac{h_k}{L} \approx p_k \triangleq \mathbb{P}\{X_l \in \mathcal{B}_k\} = \int_{\mathcal{B}_k} f_X(x) dx.$$

This is an unbiased estimate: $\mathbb{E}[\hat{p}_k] = p_k$. If we have discrete-valued random variables taking values x_1, \dots, x_K and we let $\mathcal{B}_k = \{x_k\}$, then p_k is simply the **probability mass function (PMF)** of the random variable and \hat{p}_k is an estimate of that PMF.

A histogram is the simplest form of **density estimation**, *i.e.*, estimating $f_X(x)$ from data.

In most real-world images, the pixel values are *not* independent random variables because neighboring pixel values tend to be

correlated. (We will consider such correlations in image processing methods described later.)

However, for the purposes of contrast enhancement, we can imagine a probability experiment in which we toss all the pixel values into a hat and then select values from the hat at random. The result of this experiment will be a discrete random variable, whose probability mass function is the relative frequency of each image value, *i.e.*, $p_k = h_k/L$.

Example. The histogram showed above, if normalized by $L = MN$, is such a PMF.

Histogram equalization

- If the histogram is “too clustered” then often the image will be less appealing than if we “spread out” the image values to “more evenly” cover the range [0,255].
- Supposedly it is often visually pleasing to have a displayed image’s histogram peak around 128 and fall off more or less Gaussian-like towards 0 and 255.
- Histogram-based methods for contrast enhancement usually involve a nonlinear transformation to achieve this more desirable “distribution” of pixel values.
- MATLAB’s `histeq` command defaults to a uniform target distribution, but allows a user-specified histogram.
- Typically this transformation is “more nonlinear” than simple gamma correction, and is adapted to the specific distribution of the input image.

To perform **histogram equalization** we must design a transformation $g = T(f)$ that will have the effect of producing an output image $g[m, n]$ having (approximately) some desired histogram such as a (discrete) uniform histogram from an input image $f[m, n]$.

The problem of designing the transformation T is closely related to the “transformation of random variables” problem.

(Given a pseudo-random number generator in a computer, typically producing uniform[0,1] variates, how do we generate random variables with some other distribution?)

We first consider continuous random variables.

Definitions:

cumulative distribution function (cdf) $F_X(x) = \mathbb{P}\{X \leq x\}$

probability density function (pdf) $f_X(x) = \frac{d}{dx}F_X(x)$

(roughly a picture of “relative frequency” (density))

For a continuous random variable, a **histogram** ideally would be the probabilities that a random variable falls into each of a set of intervals t_0, \dots, t_K :

$$p_k = \mathbb{P}\{t_{k-1} < X \leq t_k\} = \int_{t_{k-1}}^{t_k} f_X(x) dx = F_X(t_k) - F_X(t_{k-1}).$$

A bar plot of p_k as a function of either k or t_k is an ideal histogram.

Suppose we have random variable X with pdf f_X and we want a transformation $Z = T(X)$ such that Z has a desired pdf is $f_d(z)$, with corresponding cdf $F_d(z)$. Here is how we design T .

Fact. For a continuous random variable X , the following new random variable

$$Y = F_X(X)$$

is distributed uniformly between 0 and 1. Furthermore the following new random variable will have pdf $f_d(z)$:

$$\boxed{Z = F_d^{-1}(Y) = F_d^{-1}(F_X(X)) \triangleq T(X).} \quad (10.1)$$

Proof:

$$\mathbb{P}\{Z \leq z\} = \mathbb{P}\{F_d^{-1}(F_X(X)) \leq z\} = \mathbb{P}\{F_X(X) \leq F_d(z)\} = \mathbb{P}\{X \leq F_X^{-1}(F_d(z))\} = F_X(F_X^{-1}(F_d(z))) = F_d(z).$$

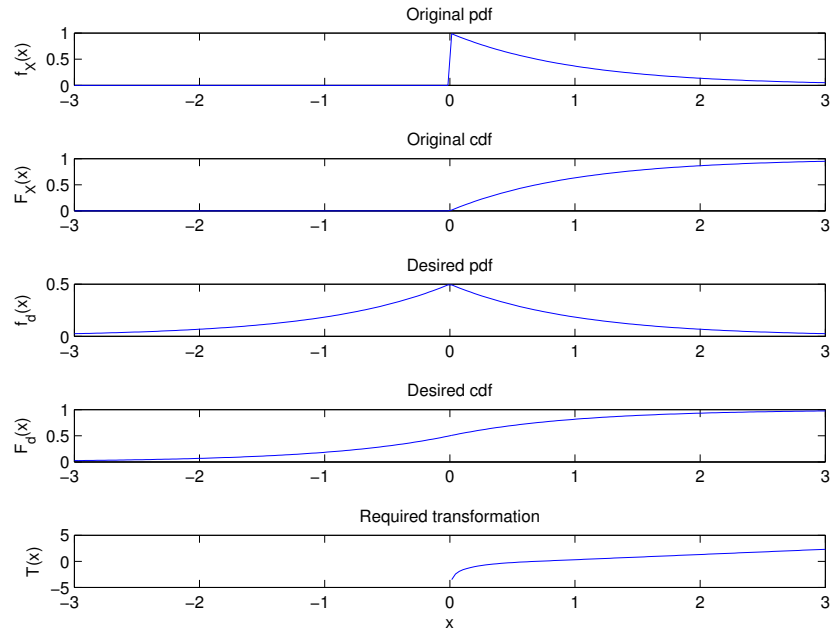
This proof assumes there are no “flat regions” in F_d or F_X , *i.e.*, strictly continuous random variables.

Example. Suppose r.v. X has pdf $f_X(x) = e^{-x} \text{step}(x)$ (exponential), but we desire $f_d(z) = \frac{1}{2} e^{-|z|}$ (Laplacian).

$$F_X(x) = (1 - e^{-x}) \text{step}(x) \text{ and } F_d(z) = \begin{cases} \frac{1}{2} e^z, & z \leq 0 \\ 1 - \frac{1}{2} e^{-z}, & z > 0, \end{cases} \text{ so } F_d^{-1}(y) = \begin{cases} \log(2y), & 0 < y \leq 1/2 \\ \log \frac{1}{2(1-y)}, & 1/2 < y < 1. \end{cases}$$

Thus the required transformation is

$$T(x) = F_d^{-1}(F_X(x)) = \begin{cases} \log(2(1 - e^{-x})), & 0 < 1 - e^{-x} \leq 1/2 \\ \log \frac{1}{2(1 - (1 - e^{-x}))}, & 1/2 < 1 - e^{-x} < 1 \end{cases} = \begin{cases} \log(2(1 - e^{-x})), & 0 < x \leq \log 2 \\ x - \log 2, & x > \log 2. \end{cases}$$



Note that the transformation function T is **monotone increasing**. This will always be the case for continuous random variables, because cdfs are strictly monotone over the range of the random variable.

Discrete random variables

Digital images are always discrete valued, and usually represented by integers. For a discrete random variable, similar *general* concepts apply, but there are some differences in the details because the cdf of a discrete r.v. is not strictly monotone.

For simplicity assume $k = 0, \dots, K - 1$.

So if $p(k)$ denotes the **probability mass function (PMF)** of the input image $f[m, n]$, then its CDF is

$$F(x) = \sum_{k=0}^{K-1} p(k) \text{step}(x - k),$$

where $\text{step}(t) = \begin{cases} 1, & t \geq 0 \\ 0, & \text{otherwise.} \end{cases}$ Here we must define the unit step function carefully!

Suppose $p_d(k)$ denotes the desired PMF, with corresponding CDF

$$F_d(x) = \sum_{k=0}^{K-1} p_d(k) \text{step}(x - k).$$

Is this CDF invertible? **No!** We rewrite the transformation relationship presented in (10.1) above as follows:

$$T(x) = F_d^{-1}(F_X(x)) = \max \{y : F_d(y) \leq F_X(x)\},$$

i.e., “for each x choose the largest y such that $F_d(y) \leq F_X(x)$.”

This generalization makes T well-defined even for non-continuous RVs where $F_d(\cdot)$ is not invertible.

Similarly, in the discrete case, for each (input) value m , we should map it to an output value n for which

$$F_d(n) \approx F(m), \quad \text{i.e.,} \quad \sum_{k=0}^n p_d(k) \approx \sum_{k=0}^m p(k).$$

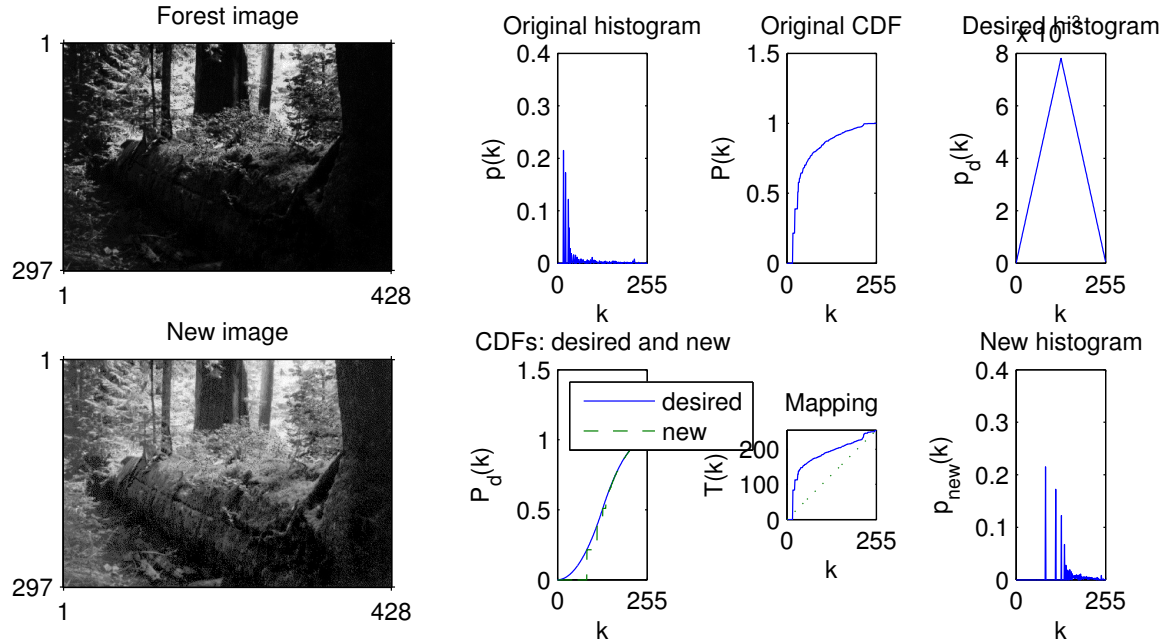
A reasonable criterion is

$$T(m) = \max \left\{ n : \sum_{k=0}^n p_d(k) \leq \sum_{k=0}^m p(k) \right\}. \quad (10.2)$$

One can show easily that this mapping is **monotone**, because as m increases the set expands. But $T(m)$ is rarely strictly monotone. For PMF/CDF arrays \mathbb{P} and \mathbb{P}_d of length K , this mapping is easily implemented in MATLAB using the command

$$n = \text{sum}(\mathbb{P}_d \leq \mathbb{P}(m)) \quad . \quad (10.3)$$

Example. An image with poor contrast is “enhanced” by histogram equalization using (10.3).



See [2] for a sophisticated modern version.

Is the histogram equalization method (10.2) a **memoryless** operation?

??

[RQ]

False color and pseudocolor

(skim)

Usually grayscale images are shown using a grayscale color map, but sometimes it is helpful to show scalar-valued images using color. MATLAB's `colormap` command provides many options.

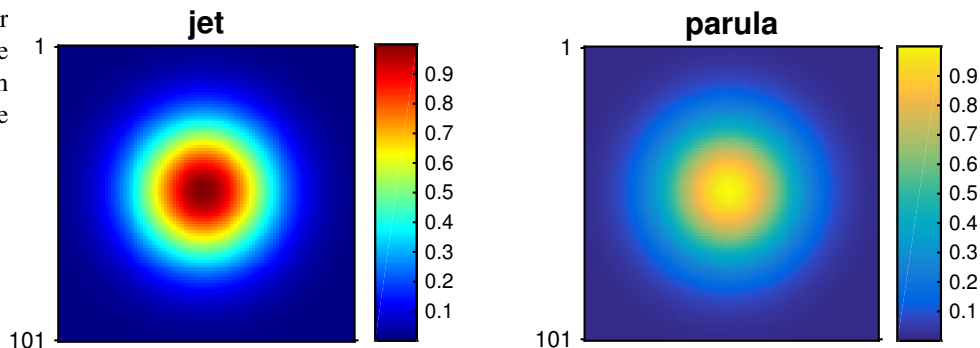
Prior to MATLAB 2014b, the default color map was a rainbow named `jet`. A disadvantage of that color map is that it can cause false impressions of contours.

A new color map called **parula** became the default in MATLAB 2014b. Do not use the `jet`/rainbow colormap! [3]

For explanation of the rationale behind it, based on human perception, see the URL below.

<http://blogs.mathworks.com/steve/2014/10/20/a-new-colormap-for-matlab-part-2-troubles-with-rainbows/>

This example compares the two color maps for a Gaussian image. Notice that the center of the Gaussian, which is the highest value, looks dark in the `jet` color map.



For more about color and human perception, see:

<http://www.wired.com/2015/02/science-one-agrees-color-dress/>

10.2 Image sharpening

Highpass filtering and unsharp masking

Because fine details like edges are often the principal source of high spatial frequency components, one can “**sharpen**” an image by “**highpass filtering**.” For historical reasons related to photographic film development, this is also called **unsharp masking**. Originally this method was performed by analog methods using film and negatives!

The natural analogy of the photographic processing approach is to scale up the original image $f[m, n]$ and then subtract a lowpass filtered version of image from itself:

$$g[m, n] = (1 + \alpha) f[m, n] - \alpha h_l[m, n] ** f[m, n],$$

where $0 < \alpha$ and $h_l[m, n]$ is a lowpass filter. Equivalently,

$$g[m, n] = f[m, n] ** h[m, n], \text{ where } h[m, n] = (1 + \alpha) \delta_2[m, n] - \alpha h_l[m, n],$$

which has a frequency response of the form

$$H(\Omega_1, \Omega_2) = 1 + \alpha - \alpha H_l(\Omega_1, \Omega_2).$$

Because $1 - H_l(\Omega_1, \Omega_2)$ is a highpass filter, another interpretation that might be more intuitive is to see this as a “high boost” filter:

$$H(\Omega_1, \Omega_2) = 1 + \alpha(1 - H_l(\Omega_1, \Omega_2)).$$

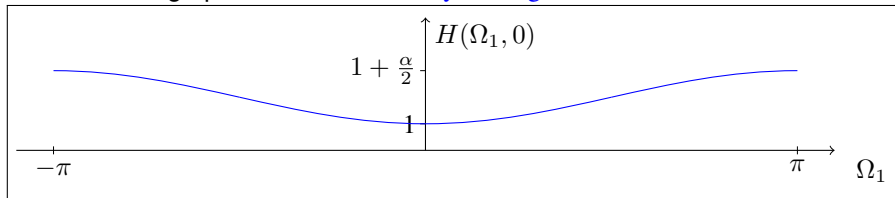
Example. The following is a typical sharpening filter. Note t which has DC response unity (so uniform regions are unchanged).

$$h[m, n] = \begin{bmatrix} 0 & -\alpha/8 & 0 \\ -\alpha/8 & 1 + \alpha/2 & -\alpha/8 \\ 0 & -\alpha/8 & 0 \end{bmatrix} = (1 + \alpha) \delta_2[m, n] - \alpha \begin{bmatrix} 0 & 1/8 & 0 \\ 1/8 & 1/2 & 1/8 \\ 0 & 1/8 & 0 \end{bmatrix}.$$

What is this filter's frequency response at DC? **??**

[RQ]

Frequency response profile. Is it a "high pass" filter? **Not really. A "high boost" filter is a better name.**



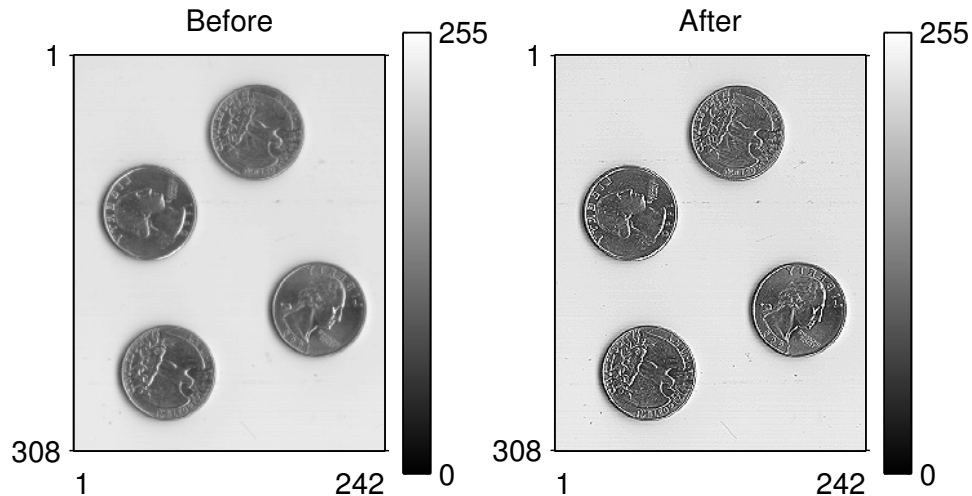
This example used the above filter with $\alpha = 8$, for which

$$h[m, n] = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

is particularly simple.

The details in the letters on the coins become more visible after sharpening. It is not a more truthful representation of the reflectance image of these objects, but it may facilitate post-processing such as reading the letters for example. Note that this filter is *very different* from the “ideal filters” discussed earlier, but it is useful nevertheless.

Limitation: amplification of high spatial frequencies will tend to increase noise.



Homomorphic processing**(skip)**

In optical reflectance imaging, *i.e.*, photography, the recorded signal is the product of the illumination times the object reflectance.

Sometimes the **illumination** of a scene causes large dynamic range changes in comparison to the variations of the object's **reflectance**, *e.g.*, due to shadows of occluding structures. If the object properties are of greater interest than the illumination properties, then it can be desirable to reduce the contribution of the illumination variations in the image. Fortunately, this is often possible because often the illumination is spatially smoothly varying (low spatial frequencies).

Because the image is the product

$$f[m, n] = i[m, n]r[m, n],$$

it is natural to apply a logarithm to “separate” the two parts:

$$\log f[m, n] = \log i[m, n] + \log r[m, n].$$

Applying a lowpass filter to $\log f[m, n]$ should primarily pass the illumination component, where as a highpass filter should primarily pass the reflectance component.

$$f[m, n] \rightarrow \boxed{\log} \rightarrow \boxed{\text{highpass from } \alpha \text{ at DC to } \beta \text{ at } \pi} \rightarrow \boxed{\exp} \rightarrow g[m, n].$$

By choosing $\alpha < \beta$, hopefully the low frequency illumination variations are reduced, highlighting the object.

See [1, Fig. 8.11] (a machine in shadow).

Note the use of a nonlinear transformation so that we could then apply LSI systems concepts!

Adaptive modification of local contrast and local luminance mean

(skip)

The preceding method can be viewed as a **two-channel processing** method. We attempt to separate the image into two components (reflectance and luminance previously), apply separate processing to each component (lowpass and highpass with different gains previously), and then add the two processed components back together again.

Many algorithms have been based on variations of this theme.

10.3 Image denoising

Up until this point, we have focused primarily on **deterministic** aspects of image processing. We now turn to methods that consider **noise** in images, which is an inherently statistical topic.

All real-world recorded images are degraded by **noise**, and often it is desirable to process an image to reduce noise. Such methods are called **image denoising**.

Here we first consider simple methods for image denoising; later we will develop more sophisticated methods based on statistical models (random processes) for images and noise.

Example. Here is a real-world example of an image denoising algorithm in a product for X-ray radiography.

<http://www.samsung.com/global/business/healthcare/healthcare/digital-radiography/DGR-C2BB2J/US>

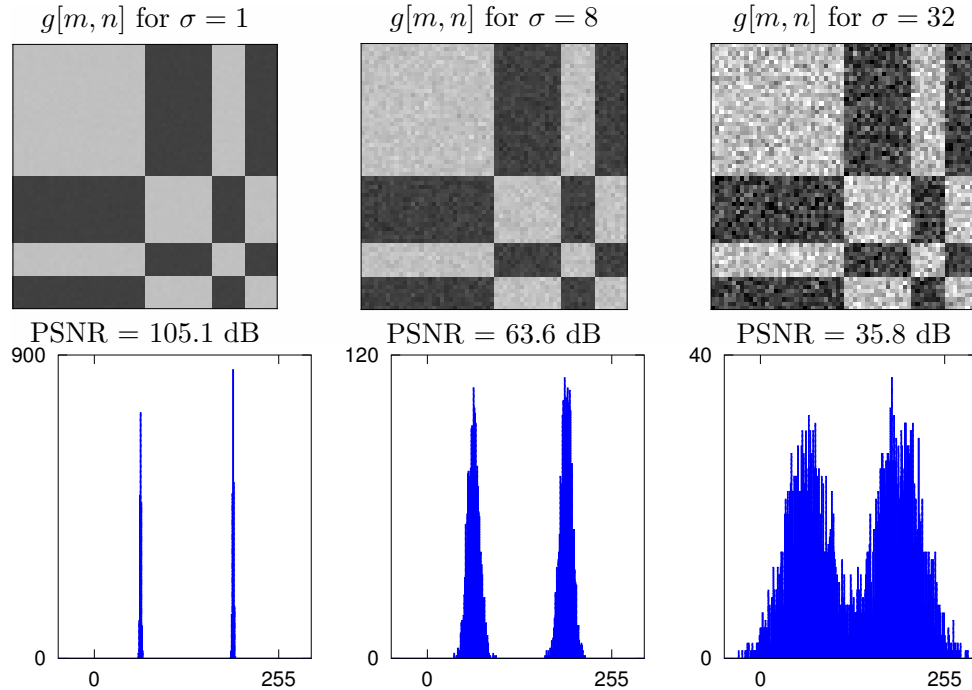
The nature of noise in images depends on the image acquisition method, but often we assume a simple additive noise model

$$\underbrace{g[m, n]}_{\substack{\text{observed} \\ \text{image}}} = \underbrace{f[m, n]}_{\text{ideal image}} + \underbrace{e[m, n]}_{\text{noise}}. \quad (10.4)$$

Often we assume the noise $e[m, n]$ is zero-mean, additive **white** Gaussian noise (**AWGN**), which we will define precisely later. We write this as $e[m, n] \sim \mathcal{N}(0, \sigma^2)$. The variance σ^2 of the noise affects the image quality, as illustrated below.

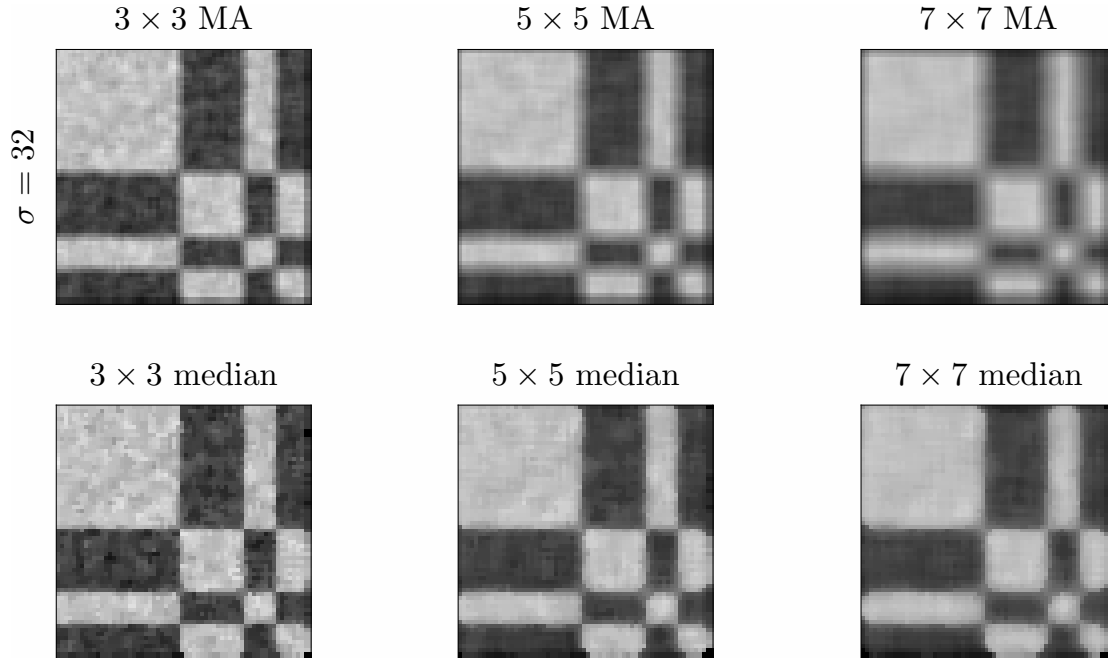
The goal of image denoising is to recover a good estimate $\hat{f}[m, n]$ of $f[m, n]$ from the noisy observed image $g[m, n]$.

Example. The true image $f[m, n]$ here has just two values: 64 and 196. Shown below are $g[m, n]$ images for 3 noise levels. The histograms show how the noise variance affects the distributions of the image values.



A popular measure of image quality is the **peak signal to noise ratio (PSNR)**, defined by $\text{PSNR} = 20 \log_{10} \left(\frac{\max_{m,n} |f[m, n]|}{\sigma} \right)$.

Example. The following figure shows the highest noise case ($\sigma = 32$) above smoothed by different sizes of moving average (MA) filters and median filters (discussed shortly).



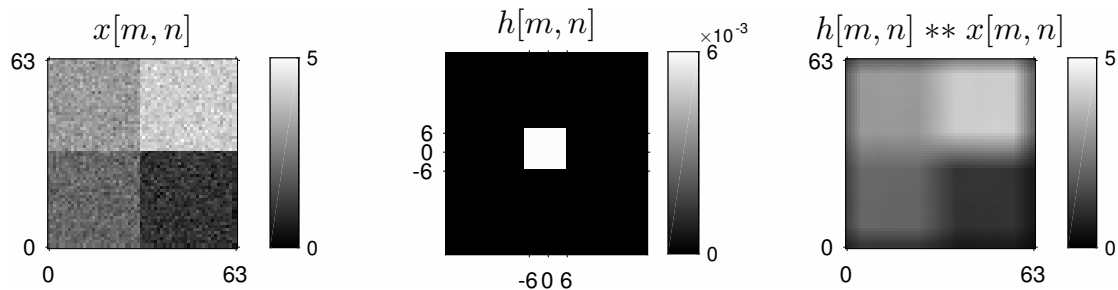
Which filter best preserves edges? [Median filter.](#)

Which filter best reduces noise? [Moving average; notice the better uniformity in the large upper left region.](#)

Lowpass filtering

The (magnitude) spectra of typical images are decreasing functions of spatial frequency, meaning that there is less power at high spatial frequencies. In contrast, noise power spectra (defined precisely in Ch. 11) are often flat (e.g., white noise) or approximately flat. So the SNR is poor for high spatial frequencies. So it is natural to use a lowpass filter to remove those poor SNR components and preserve the good SNR components at low spatial frequencies.

Example. This illustrates reduced noise but degraded spatial resolution (blur).



A significant portion of the image processing literature is devoted to nonlinear methods that attempt to reduce noise while preserving important image features like edges without blurring them out. We will consider such methods soon.

The canonical example of a lowpass filter is the 3×3 moving average, but it has a poor frequency response as we have seen.

Ideally we might want such lowpass filtering to be rotationally invariant, but for practical use one more often settles for filters like

$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ that have 8-fold symmetry. This example filter is also separable, and easily implemented because dividing by 16 is a trivial bit-shift operation in base-2 arithmetic.

Median filtering

The classical example of an edge-preserving, nonlinear method for reducing noise, especially impulsive noise is **median filtering**, and its generalizations: **stack filtering** and **order-statistic filtering**.

To see what happens to an edge in a image when we apply the 3by3 moving average, we consider the 1D case:

$$[\dots 000 \underline{1}111 \dots] * \begin{bmatrix} 1 & 1 & 1 \\ 3 & 3 & 3 \end{bmatrix} = \left[\dots 00 \frac{1}{3} \frac{2}{3} 111 \dots \right],$$

so the edge is blurred out.

The 1D median filter of length 3 takes the median of each set of three adjacent values (the middle value when rank ordered). Applying MATLAB's `medfilt1` command to the above step function will not perturb the edge.

Note: $\text{median}(1,1,1,2,2) = 1$ but in MATLAB, $\text{median}(1,1,1,2,3,5) = 1.5$. Usually we use odd window sizes anyway.

What happens to impulsive noise (in an otherwise uniform region) when using a lowpass filter (like the moving average) and when using a median filter? **Lowpass filtering smears it out whereas median filtering can remove the impulse completely.**

Using larger filter sizes means more reduction of impulsive noise values, but potentially more loss of fine signal values.

Example. Consider the 5-point and 3-point median filters applied to the signal $[0.2 \ 0.1 \ 1 \ 1 \ 0.1 \ -0.1]$. Are those two “impulses” in the middle signal or noise? It might depend on the application.

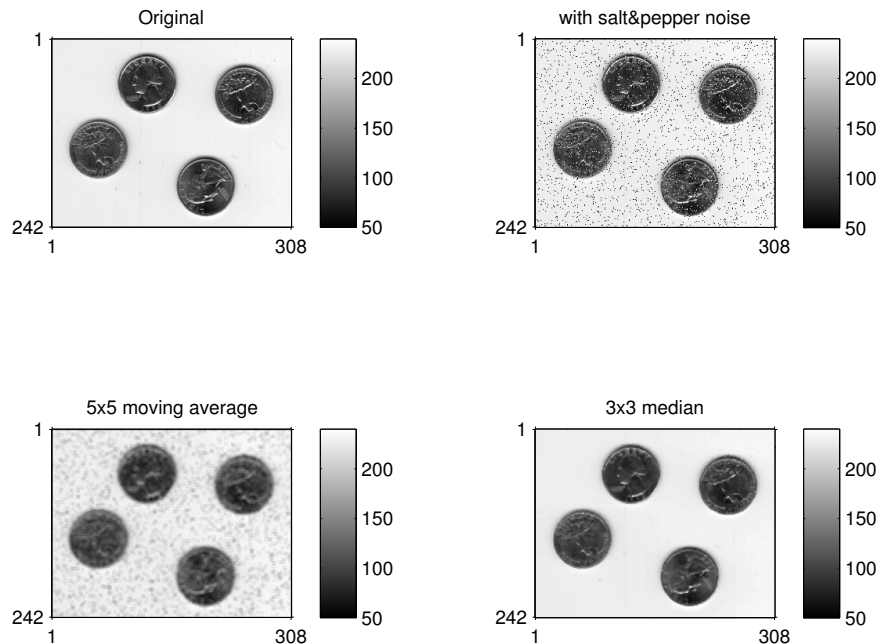
Drawbacks of median filters:

- nonlinear - complicates analysis
- Complete loss of fine signal values (whereas linear smoothing would just blur them out).
- Computationally demanding due to sorting, but some tricks are available for sorting for moving windows.
- Does not do any *averaging*. (See below.)
- An M -point median filter can introduce artificial small plateaus of length $(M + 1)/2$.

- Sensitive to window size: the 1D signal $\delta[n] + \delta[n - 1]$ is unaffected by a 3-pt median, but removed by a 5-pt median.

Median filtering can be related to a maximum-likelihood estimation method for **Laplacian** noise distribution: $\frac{1}{2} e^{-|x|}$. This distribution has “heavy tails” compared to Gaussian noise (e^{-x} vs e^{-x^2}), so impulsive “outliers” are much more likely with Laplacian noise.

Example. This example uses MATLAB’s `medfilt2` command to remove impulsive “salt and pepper” noise more effectively than a MA filter.



Noise-reducing properties of median filtering

Here is a statistical analysis of how much a median filter reduces noise compared to a moving average filter.

Let X_1, \dots, X_n be IID random variables with pdf $f_X(x)$, mean μ , and variance σ^2 .

It is well known that the sample mean estimate of μ (which is analogous to a moving average filter) has a variance that diminishes with n :

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i, \quad \text{Var}\{\bar{X}\} = \frac{\sigma^2}{n}.$$

In fact, by the **central limit theorem (CLT)**, \bar{X} is asymptotically normal in distribution:

$$\sqrt{n}(\bar{X} - \mu) \rightarrow \text{N}(0, \sigma^2).$$

Now consider the **sample median** estimator $\hat{X} = \text{median}(X_1, \dots, X_n)$. Under the additional assumption that $f_X(x)$ is symmetric about μ , one can show [4, p. 401] that this estimator is also asymptotically normal:

$$\sqrt{n}(\hat{X} - \mu) \rightarrow \text{N}\left(0, \frac{1}{4f_X^2(\mu)}\right).$$

In other words, for large n

$$\text{Var}\{\hat{X}\} \approx \frac{1}{n4f_X^2(\mu)}.$$

Example. For $X_i \sim \text{Uniform}(a, b)$, where $f_X(x) = \frac{1}{b-a}\mathbb{I}_{\{a < x < b\}}$ we have $\text{Var}\{\bar{X}\} = \frac{(b-a)^2}{12n}$ and $\text{Var}\{\hat{X}\} \approx \frac{(b-a)^2}{4n}$.

The median has a larger variance than the mean for a uniform distribution, so it reduces noise less in the smooth regions of an image. But this penalty often may be acceptable when preserving edges is important.

For an image having pixel values uniformly distributed between 0 and 255, what is the (approximate) standard deviation of the output of a 5×5 median filter? ?? [RQ]

Rank order filtering (skip)

The median filter is a special case of **rank order filtering** [5].

Out-range pixel smoothing (skip)

Why another method? The median filter selects one of the input image's pixel values as the output value. So in uniform image regions, one does not benefit from the noise-reducing characteristics of **averaging**. On the other hand, averaging (*e.g.*, the moving average) works great in uniform regions, but blurs edges. There are *many* nonlinear compromise methods that essentially attempt to determine whether each pixel belongs to a uniform region (in which case an average is used) or an edge region (in which case something like the median filter is used). The **out-range pixel smoothing** method is one such method. Take the average of each pixel's neighbors. If the pixel value is close to that average, replace it with the average. Otherwise no change. Here, "close to" is quantified by some **threshold**, and a limitation of many image enhancement methods is that they depend on various **tuning parameters** aka **fiddle factors**. It is difficult to specify procedures for selecting such factors in general, although one can use empirical experience to find reasonable values for each specific class of images of interest (for example MRI brain scans).

Here is a mathematical representation of the input-output relationship:

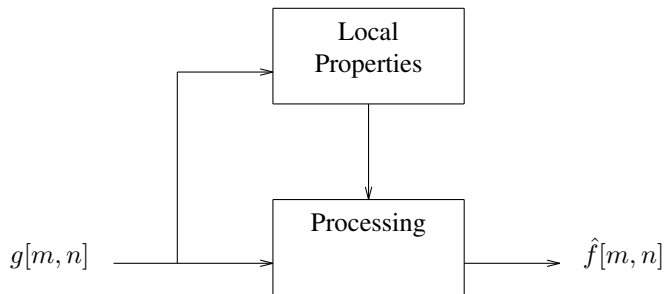
$$g[m, n] = \begin{cases} f[m, n], & |f[m, n] - (f ** h)[m, n]| > \gamma \\ (f ** h)[m, n], & \text{otherwise.} \end{cases}$$

Adaptive image smoothing

An “ideal” noise reduction method would smooth a lot where the true signal is (nearly) uniform (low frequency), and smooth less where the true signal varies rapidly (near edges in particular). Obviously such considerations are application dependent.

In **adaptive** methods, one deliberately changes the behavior of the “filter” in different image regions according to some criteria, *e.g.*, [6].

A typical adaptive method has the following form.



One can apply adaptive methods either **pixel by pixel** or **block by block**. The pixel by pixel methods generally require more computation. However, the block by block methods can cause abrupt changes in image intensity between blocks, called the **blocking effect**. A common and effective method for reducing the blocking effect is to use **overlapping blocks**. This increases computation somewhat, but not as much as full pixel by pixel methods.

Example. An adaptive moving-average filter. At each pixel, compute the local variance over some $M \times M$ local neighborhood:

$$\hat{\sigma}_g^2[m, n] = \frac{1}{(2M+1)^2 - 1} \sum_{k=-M}^M \sum_{l=-M}^M (g[m-k, n-l] - \bar{g}[m, n])^2,$$

where the local mean is given by the usual moving average:

$$\bar{g}[m, n] \triangleq \frac{1}{(2M+1)^2} \sum_{k=-M}^M \sum_{l=-M}^M g[m-k, n-l].$$

Then choose window sizes based on the local variance, *e.g.*, $N(\sigma^2)$ is a decreasing function of σ^2 .

Then form the output image value by a local average with an adaptive neighborhood size:

$$\hat{f}[m, n] = \frac{1}{(2N+1)^2} \sum_{k=-N}^N \sum_{l=-N}^N g[m-k, n-l] \Bigg|_{N=N(\hat{\sigma}_g^2[m, n])}.$$

The “art” in a method like this is choosing M appropriately and $N(\sigma^2)$.

If the variance is high near a given location $[m, n]$, then there is likely an edge near there and we should decrease the amount of averaging, perhaps even letting $N = 0$ so that $\hat{f}[m, n] = g[m, n]$ at such locations.

There are numerous such methods in the literature, often called **denoising** algorithms. For a recent review and methods, see [7–14]. Because of the nonlinearities, performance analysis is difficult.

Bilateral filter**(skim)**

Another popular noise smoothing method that is an adaptive method is the **bilateral filter** [15–18].

This filter is available in image processing tools like Photoshop and GIMP. [\[wiki\]](#)

Given a noisy input image $g[m, n]$ with the additive noise model (10.4), the output of bilateral filtering is

$$\hat{f}[m, n] = \frac{1}{s[m, n]} \sum_{(k,l) \in \mathcal{N}_{[m,n]}} w[m, n; k, l] g[k, l],$$

where $\mathcal{N}_{[m,n]}$ denotes a set of pixel locations near $[m, n]$, e.g., for a 5×5 neighborhood: $\mathcal{N}_{[m,n]} \triangleq \cup_{k=-2}^2 \cup_{l=-2}^2 \{[m+k, n+l]\}$.

The adaptive weight $w[\cdot]$ is defined by

$$w[m, n; k, l] \triangleq \psi_r(|g[m, n] - g[k, l]|) \psi_s(\|(m, n) - (k, l)\|)$$

and the following normalization term preserves the DC value:

$$s[m, n] \triangleq \sum_{(k,l) \in \mathcal{N}_{[m,n]}} w[m, n; k, l].$$

Often Gaussian functions are used for the range kernel ψ_r and the spatial kernel ψ_s , in which case the adaptive weights are

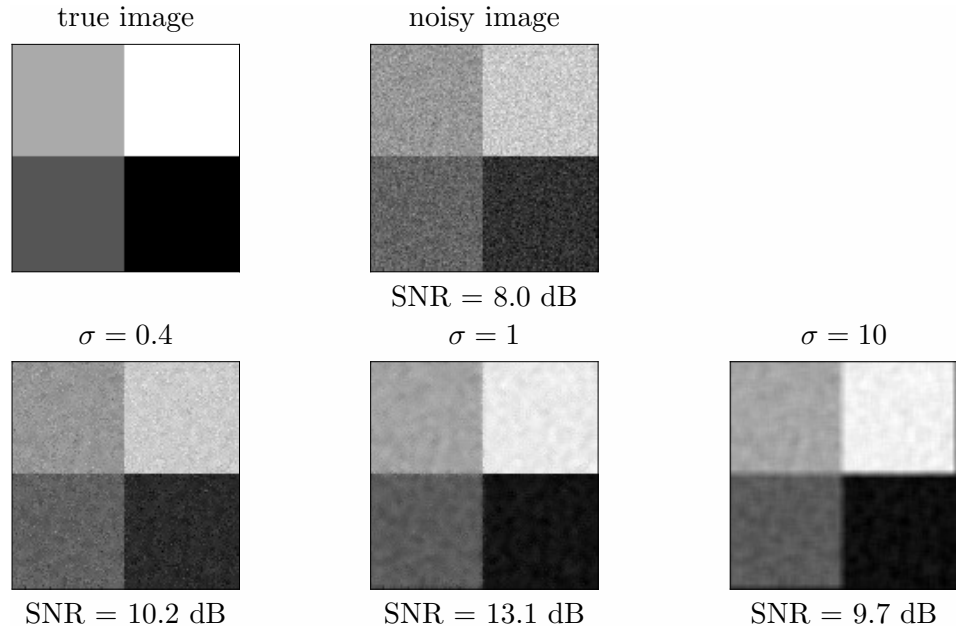
$$w[m, n; k, l] = \exp\left(-\frac{|g[m, n] - g[k, l]|^2}{\sigma_r^2} - \frac{(m-k)^2 + (n-l)^2}{\sigma_s^2}\right),$$

where σ_r and σ_s are tuning parameters.

What happens when $\sigma_r \rightarrow \infty$? ??

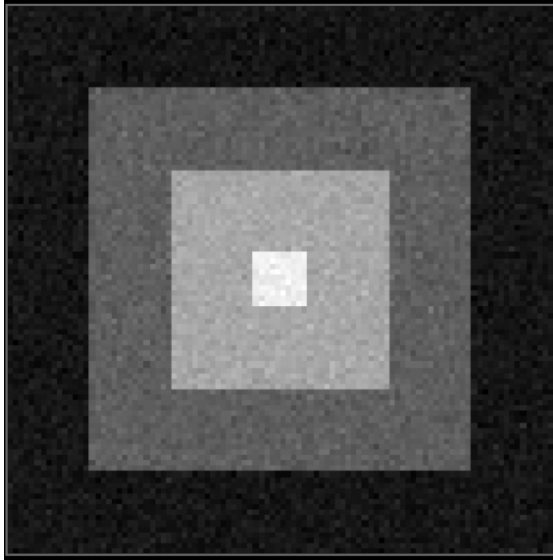
[RQ]

Example. This figure illustrates applying the bilateral filter to a noisy image $g[m, n]$ using a 5×5 neighborhood \mathcal{N} with spatial kernel $\psi_s = 1$ and the Gaussian range kernel with three values of σ_r .

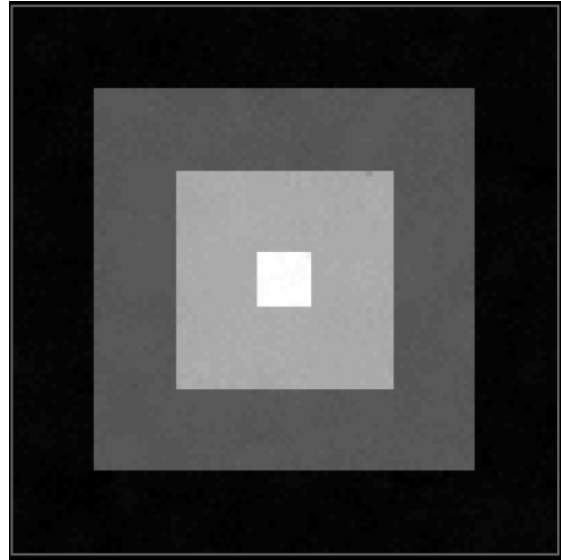


Example. Images from [17].

Left: original noisy image $g[m, n]$



Right: bilateral filtered image $\hat{f}[m, n]$



For this piecewise constant image, the bilateral filter does a reasonable job of reducing noise while preserving edges.

10.4 Summary

We have discussed the basic ideas of some image enhancement methods. We were able to use Fourier analysis was useful only for unsharp masking; the other methods were too nonlinear to apply linear systems methods for analysis. Thus one usually resorts to empirical evaluation.

Bibliography

- [1] J. S. Lim. *Two-dimensional signal and image processing*. New York: Prentice-Hall, 1990. ISBN: 978-0139353222.
- [2] W. Wang and M. Ng. “A variational histogram equalization method for image contrast enhancement”. In: *SIAM J. Imaging Sci.* 6.3 (2013), 1823–49.
- [3] D. Borland and R. M. Taylor. “Rainbow color map (still) considered harmful”. In: *IEEE Computer Graphics and Appl.* 27.2 (Mar. 2007), 14–7.
- [4] P. J. Bickel and K. A. Doksum. *Mathematical statistics*. Oakland, CA: Holden-Day, 1977.
- [5] H. D. Tagare. “Order filters”. In: *Proc. IEEE* 73.1 (Jan. 1985), 163–5.
- [6] H. Knutsson, R. Wilson, and G. H. Granlund. “Anisotropic non-stationary image estimation and its applications-Part I: Restoration of noisy images”. In: *IEEE Trans. Comm.* 31.3 (Mar. 1983), 388–97.
- [7] A. Buades, B. Coll, and J. M. Morel. “A review of image denoising methods, with a new one”. In: *SIAM Multiscale Modeling and Simulation* 4.2 (2005), 490–530.
- [8] M. Mahmoudi and G. Sapiro. “Fast image and video denoising via nonlocal means of similar neighborhoods”. In: *IEEE Signal Proc. Letters* 12.12 (Dec. 2005), 839–42.
- [9] A. Buades, B. Coll, and J-M. Morel. “The staircasing effect in neighborhood filters and its solution”. In: *IEEE Trans. Im. Proc.* 15.6 (June 2006), 1499–505.
- [10] C. Kervrann and J. Boulanger. “Optimal spatial adaptation for patch-based image denoising”. In: *IEEE Trans. Im. Proc.* 15.10 (Oct. 2006), 2866–78.

- [11] M. Elad and M. Aharon. “Image denoising via sparse and redundant representations over learned dictionaries”. In: *IEEE Trans. Im. Proc.* 15.12 (Dec. 2006), 3736–45.
- [12] H. Takeda, S. Farsiu, and P. Milanfar. “Kernel regression for image processing and reconstruction”. In: *IEEE Trans. Im. Proc.* 16.2 (Feb. 2007), 349–66.
- [13] P. Milanfar and P. Chatterjee. “A generalization of non-local means via kernel regression”. In: *Proc. SPIE 6814 Computational Imaging VI*. 2008, 68140P.
- [14] P. Chatterjee and P. Milanfar. “Is denoising dead?” In: *IEEE Trans. Im. Proc.* 19.4 (Apr. 2010), 985–911.
- [15] C. Tomasi and R. Manduchi. “Bilateral filtering for gray and color images”. In: *Proc. Intl. Conf. Comp. Vision*. 1998, 839–46.
- [16] N. Sochen, R. Kimmel, and A. M. Bruckstein. “Diffusions and confusions in signal and image processing”. In: *J. Math. Im. Vision* 14.3 (May 2001), 195–209.
- [17] M. Elad. “On the origin of the bilateral filter and ways to improve it”. In: *IEEE Trans. Im. Proc.* 11.10 (Oct. 2002), 1141–51.
- [18] K. N. Chaudhury, D. Sage, and M. Unser. “Fast $O(1)$ bilateral filtering using trigonometric range kernels”. In: *IEEE Trans. Im. Proc.* 20.12 (Dec. 2011), 3376–82.

Chapter 11

Random processes and Wiener filters

Contents (class version)

| | |
|---|--------------|
| 11.0 Introduction | 11.2 |
| 11.1 Random processes | 11.4 |
| Random variables [review] | 11.4 |
| Random vectors [review] | 11.8 |
| Random processes | 11.11 |
| Nonnegativity and positive-semidefinite functions [skip] | 11.18 |
| Pairs of random processes | 11.22 |
| Random processes through LSI systems | 11.25 |
| Synthesizing random vectors and processes numerically [skip] | 11.29 |
| 2nd-order statistics - an incomplete story | 11.32 |
| 11.2 The noncausal Wiener filter for “optimal” image denoising | 11.35 |
| Stronger optimality conditions | 11.41 |
| Minimization of complex quadratic forms [skip] | 11.43 |
| 11.3 Wiener filter for image deblurring | 11.45 |
| Performance measures (MSE, NRMSE, SNR, etc.) | 11.48 |
| 11.4 Spectral estimation [skip] | 11.51 |
| Periodogram | 11.51 |
| 11.5 Patch-based denoising methods | 11.61 |

| | |
|--|--------------|
| Non-local means | 11.61 |
| 11.6 Markov random field models [skip] | 11.64 |
| 11.7 Image segmentation / classification [skip] | 11.73 |
| Maximum likelihood image segmentation | 11.74 |
| Using MRF models for image segmentation | 11.76 |
| 11.8 Summary | 11.82 |

11.0 Introduction

All of the major topics up until this point were essentially deterministic:

- imaging systems, Fourier analysis, sampling, filtering,
- interpolation, edge detection, image enhancement

For the remaining topics, instead of *ad hoc* methods, it is more appealing to consider methods based on statistical image models.

The main applications forthcoming are:

- image denoising
- image restoration / reconstruction
- image segmentation
- image compression

This chapter presents material related to these motivating applications and describes various image statistical models that one can apply in a variety of contexts.

Filters revisited

Although we may have downplayed the importance of ideal low-pass filters (and their relatives) earlier in Ch. 6, filtering is nevertheless still quite important in image processing. But some of the filters of most interest have more interesting forms and motivations than simple ideal low-pass filters. This chapter motivates one classic filtering method: the **Wiener filter**.

The Wiener filter (and its relatives) are formulated based on random process principles. So we will begin by reviewing 2nd-order properties of random processes [1, Ch. 6].

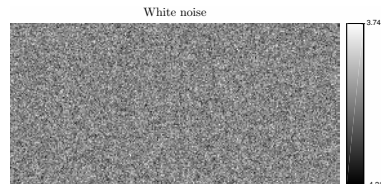
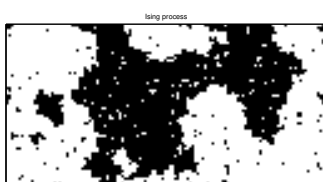
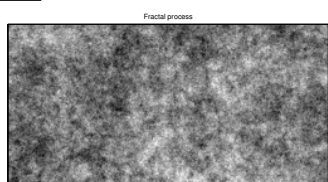
Outline

- random variables, random vectors, random processes
- 2nd-order properties, through LSI systems
- Wiener filter (for denoising and deblurring), spectral estimation
- Markov random field (**MRF**) models
- Applications in this chapter: **image denoising, image segmentation**

11.1 Random processes

We focus on discrete-space random processes, *i.e.*, **random images**, or **random fields**, that consist of a lattice of **random variables**.

Example. Here are examples of a gray-scale random field, a binary random field, and white noise. We will discuss all these models.



So first we need to review random variables, assuming the reader is familiar with probability.

Random variables [review]

We first review **random variables**, for completeness.

The defining characteristic of a real-valued random variable X is its **cumulative distribution function (CDF)**, given by

$$F_X(x) \triangleq \mathbb{P}\{X \leq x\}, \forall x \in \mathbb{R}.$$

Note that a CDF has the following **end conditions**: $\lim_{t \rightarrow \infty} F_X(t) = 1$ and $\lim_{t \rightarrow -\infty} F_X(t) = 0$.

Caution: earlier in the course x denoted a spatial coordinate, but now it denotes a random variable!

For **continuous random variables**, the CDF is differentiable, and we define a corresponding **probability density function (pdf)**:

$$p_X(x) \triangleq \frac{d}{dx} F_X(x).$$



By the fundamental theorem of calculus, it follows from the end conditions above that

$$P\{X \in \mathcal{B}\} = \int_{\mathcal{B}} p_X(x) dx.$$

This holds for all the subsets \mathcal{B} of \mathbb{R} that one would ever need in practice (and lots more). In particular

$$F_X(x) \triangleq P\{X \leq x\} = \int_{-\infty}^x p_X(x') dx'.$$

Some important discrete random variables:

- **Bernoulli**
- **binomial**
- **Poisson**

Example. The **probability mass function (PMF)** of a Poisson random variable with mean $\lambda > 0$:

$$P\{X = k\} = e^{-\lambda} \lambda^k / k!, \quad k = 0, 1, \dots$$

Some important continuous random variables:

- **uniform**
- Gaussian or **Normal**
- **Laplace distribution** or Laplacian (double-sided exponential, for impulsive noise)

Example. The **probability density function (pdf)** of a Gaussian random variable with mean μ and variance σ^2 :

$$p_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(x-\mu)^2/\sigma^2}.$$

Properties of random variables

The following definitions apply to both real- and complex-valued random variables.

- **mean** or **expected value** : $\mu_X \triangleq \mathbb{E}[X] = \int x p_X(x) dx$
- **expectation of a function** of a RV : $\mathbb{E}[g(X)] = \int g(x) p_X(x) dx$
- **variance** : $\text{Var}\{X\} \triangleq \mathbb{E}[|X - \mathbb{E}[X]|^2] = \mathbb{E}[|X|^2] - |\mathbb{E}[X]|^2$
- **correlation**¹ : $\mathbb{E}[XY^*] = \iint x y^* p_{XY}(x, y) dx dy$
- **covariance** : $\text{Cov}\{X, Y\} \triangleq \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])^*] = \mathbb{E}[XY^*] - \mathbb{E}[X]\mathbb{E}[Y^*]$

We call variance, correlation, and covariance **2nd-order properties** because they depend on the square or products of two terms.

We say X and Y are **independent random variables** iff their **joint distribution** factors into the product of **marginal distributions**:

$$p_{XY}(x, y) = p_X(x) p_Y(y).$$

The expected value $\mathbb{E}[X]$ of a random variable is also called its **ensemble average** because it is the “average” (weighted by the pdf $p_X(x)$) of all possible values of the random variable. In contrast the **sample mean** of a collection of numbers x_1, \dots, x_n is given by the **average**: $\frac{1}{n} \sum_{i=1}^n x_i$. One must be careful to distinguish between the mean of a random variable and the sample mean of a collection of numbers. In MATLAB, the `mean` function computes the sample mean, not the ensemble mean.

¹ If $X = X_r + iX_i$ and $Y = Y_r + iY_i$ are complex, then the integral for correlation is four-dimensional:
 $\mathbb{E}[XY^*] = \iiint \int (a + ib)(c - id) p_{X_r, X_i, Y_r, Y_i}(a, b, c, d) da db dc dd.$

Properties of correlation / covariance

- $\text{Cov}\{X, Y\} = E[XY^*] - \mu_X \mu_Y^*$
- $\text{Cov}\{X, Y\} = \text{Cov}^*\{Y, X\}$
- $\text{Cov}\{X, X\} = \text{Var}\{X\}$
- $\text{Cov}\{aX + b, cY + d\} = ac^* \text{Cov}\{X, Y\}$
- **Cauchy-Schwarz inequality:** $|\text{Cov}\{X, Y\}| \leq \sigma_X \sigma_Y, \quad |E[XY^*]| \leq \sqrt{E[|X|^2] E[|Y|^2]}$
- If X and Y are **independent random variables** then $E[XY] = \mu_X \mu_Y$ so $\text{Cov}\{X, Y\} = 0$.
- The reverse is *not true* in general (uncorrelated does not ensure independence); an exception is Gaussian distributed RVs.
- covariance is a **bilinear operation:** $\text{Cov}\left\{\sum_i X_i, \sum_j Y_j\right\} = \sum_i \sum_j \text{Cov}\{X_i, Y_j\}$

Random vectors [review]

A finite collection of random variables (over a common probability space) is called a **random vector**, often denoted $\mathbf{X} = (X_1, \dots, X_n) \in \mathcal{X}$, where each X_i is a random variable and typically $\mathcal{X} = \mathbb{R}^n$ or $\mathcal{X} = \mathbb{C}^n$ for real or complex random vectors respectively.

The defining characteristic of a random vector is its **cumulative distribution function (CDF)**, given by

$$F_{\mathbf{X}}(x_1, \dots, x_n) = \mathbb{P}\{X_1 \leq x_1, \dots, X_n \leq x_n\}.$$

Note that a CDF has the following **end conditions**: $\lim_{t \rightarrow \infty} F_{\mathbf{X}}(t, \dots, t) = 1$ and $\lim_{t_k \rightarrow -\infty} F_{\mathbf{X}}(t_1, \dots, t_n) = 0$ for $k = 1, \dots, n$ and for any $\{t_i\} \in \mathbb{R}$.

For **continuous random vectors**, the CDF is differentiable, and we define a corresponding **probability density function (pdf)** by

$$p_{\mathbf{X}}(\mathbf{x}) \triangleq \frac{\partial^n}{\partial x_1 \dots \partial x_n} F_{\mathbf{X}}(x_1, \dots, x_n).$$

By the fundamental theorem of calculus, it follows from the end conditions above that

$$\mathbb{P}\{\mathbf{X} \in \mathcal{B}\} = \int_{\mathcal{B}} p_{\mathbf{X}}(\mathbf{x}) \, d\mathbf{x}.$$

This holds for all the subsets \mathcal{B} of \mathcal{X} that one would ever need in practice (and lots more).

We say X_1, \dots, X_n are **independent random variables** iff

$$\mathbb{P}\{X_1 \leq x_1, \dots, X_n \leq x_n\} = \prod_{i=1}^n \mathbb{P}\{X_i \leq x_i\}, \quad \forall x_i \in \mathbb{R}.$$

In particular, the elements of a continuous random vector are **independent** if and only if their pdf is **separable**:

$$p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{X}}(x_1, \dots, x_n) = \prod_{i=1}^n p_{X_i}(x_i), \quad \forall \{x_i\} \in \mathbb{R}.$$

Properties of random vectors

The **mean** or **expected value** of a random vector is defined by:

$$\mathbb{E}[\mathbf{X}] = \int_{\mathcal{X}} \mathbf{x} p_{\mathbf{X}}(\mathbf{x}) d\mathbf{x}.$$

The expectation of a function of a random vector is:

$$\mathbb{E}[g(\mathbf{X})] = \int_{\mathcal{X}} g(\mathbf{x}) p_{\mathbf{X}}(\mathbf{x}) d\mathbf{x}.$$

The $n \times n$ **correlation matrix** of a length- n random vector (column vector) is

$$\mathbf{R}_{\mathbf{X}} = \mathbb{E}[\mathbf{X}\mathbf{X}'], \text{ i.e., } [\mathbf{R}_{\mathbf{X}}]_{ij} = \mathbb{E}[x_i x_j^*] = \int_{\mathcal{X}} x_i x_j^* p_{\mathbf{X}}(x_1, \dots, x_n) d\mathbf{x},$$

where “ $'$ ” denotes Hermitian transpose (for complex-valued RVs).

The $n \times n$ **covariance matrix** of a length- n random vector is

$$\mathbf{K}_{\mathbf{X}} = \text{Cov}\{\mathbf{X}\} = \mathbb{E}[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])'] = \mathbf{R}_{\mathbf{X}} - \mathbb{E}[\mathbf{X}]\mathbb{E}[\mathbf{X}'], \text{ i.e., } [\mathbf{K}_{\mathbf{X}}]_{ij} = \text{Cov}\{X_i, X_j\}.$$

If the elements of a random vector are **independent**, then its covariance matrix is **diagonal**: $\mathbf{K}_{\mathbf{X}} = \text{diag}\{\sigma_1^2, \dots, \sigma_n^2\}$.

Covariance matrices are **positive semidefinite**, which we denote by writing $\text{Cov}\{\mathbf{X}\} \succeq \mathbf{0}$.

We call mean, correlation, and covariance **2nd-order properties of random vectors**.

Example. A particularly important type of continuous random vector \mathbf{X} has a multivariate **gaussian** or **normal distribution**, denoted

$$\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{K}),$$

where $\boldsymbol{\mu} = E[\mathbf{X}]$ denotes the **mean** of \mathbf{X} and $\mathbf{K} = \text{Cov}\{\mathbf{X}\}$ denotes the **covariance** of \mathbf{X} . For the usual case where \mathbf{X} is real, its **probability density function (pdf)** is given by

$$p_{\mathbf{X}}(\mathbf{x}) = \frac{1}{\sqrt{\det\{2\pi\mathbf{K}\}}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})' \mathbf{K}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right).$$

Note that if \mathbf{K} is a $n \times n$ matrix, then $\det\{\alpha\mathbf{K}\} = \alpha^n \det\{\mathbf{K}\}$.

Random processes

An infinite collection of random variables is called a **random process**.

In the context of imaging problems, random processes are often called **random fields**.

If $x[\cdot, \cdot]$ denotes a (2D) random process, then each $x[m, n]$ is a random variable, and all those random variables are defined over a common probability space.

Note: because we use capital letters for spectra in this course, we usually use small letters for random processes.

In principle, the behavior of a random process is specified completely by all of its joint distribution functions. In practice we often do not have all the information necessary to specify such joint distributions, so we “make do” by working with the moments of the random process. Gaussian random processes are specified completely by the first two moment functions (mean function and autocorrelation function defined below), so working with just the first two moments is adequate anyway in the important Gaussian case. The moment functions we need are defined as follows.

- The **mean function**: $\mu_x[m, n] \triangleq E[x[m, n]]$
- The **autocorrelation function**: $R_x[m, n; k, l] \triangleq E[x[m, n] x^*[k, l]]$.
We want to be able to consider **complex**-valued random processes because of the importance of complex exponentials in Fourier analysis and for applications like MRI that involve complex-valued images.
- The **autocovariance function**: $K_x[m, n; k, l] \triangleq \text{Cov}\{x[m, n], x[k, l]\} = E[(x[m, n] - \mu_x[m, n])(x[k, l] - \mu_x[k, l])^*]$
 $= R_x[m, n; k, l] - \mu_x[m, n] \mu_x^*[k, l]$. (11.1)

The autocorrelation and autocovariance functions satisfy the following Hermitian symmetry conditions based on their definitions:

$$\begin{aligned} K_x[k, l; m, n] &= K_x^*[m, n; k, l] \\ R_x[k, l; m, n] &= R_x^*[m, n; k, l]. \end{aligned} \tag{11.2}$$

Correlation is a kind of **inner product**, and thus the autocorrelation and autocovariance functions satisfy the following

Cauchy-Schwarz inequalities:

$$|K_x[m, n; k, l]| \leq \sqrt{K_x[m, n; m, n] K_x[k, l; k, l]}$$

$$|R_x[m, n; k, l]| \leq \sqrt{R_x[m, n; m, n] R_x[k, l; k, l]}.$$

The square roots are well defined because $K_x[m, n; m, n] \geq 0$ and $R_x[m, n; m, n] \geq 0$.

2D WSS random processes

A random process $x[m, n]$ for $m, n \in \mathbb{Z}$ is called **wide-sense stationary (WSS)** if

- the mean function is a constant

$$\mu_x[m, n] = \mu, \quad \forall m, n \in \mathbb{Z}$$

- the autocorrelation function depends only on the difference in spatial coordinates:

$$R_x[m, n; k, l] = R_x[m - k, n - l; 0, 0], \quad \forall m, n, k, l \in \mathbb{Z}. \quad (11.3)$$

For WSS processes we define a concise version of the autocorrelation function:

$$R_x[m, n] \triangleq R_x[m, n; 0, 0]$$

and it follows that

$$R_x[m, n; k, l] = R_x[m - k, n - l], \quad \forall m, n, k, l \in \mathbb{Z}.$$

Likewise the autocovariance function of a WSS random process is

$$K_x[m, n] \triangleq K_x[m, n; 0, 0] = K_x[m + k, n + l; k, l]$$

and by (11.1) it satisfies

$$K_x[m, n] = R_x[m, n] - |\mu|^2, \quad \forall m, n \in \mathbb{Z}. \quad (11.4)$$

Properties of WSS random processes

- shift invariance: $R_x[m, n] = E[x[m, n] x^*[0, 0]] = E[x[m+k, n+l] x^*[k, l]], \quad \forall k, l \in \mathbb{Z}$.

In particular a shifted version $y[m, n] = x[m - m_0, n - n_0]$ for any $m_0, n_0 \in \mathbb{Z}$ of a WSS random process $x[m, n]$ has the same second-order properties as the original: $\mu_y = \mu_x$ and $R_y[m, n] = R_x[m, n]$.

- variance: $K_x[0, 0] = \text{Var}\{x[m, n]\}, \quad \forall m, n \in \mathbb{Z}$

and $R_x[0, 0] = \text{Var}\{x[m, n]\} + |\mu|^2 = K_x[0, 0] + |\mu|^2$.

- **Hermitian symmetry:** $R_x[-m, -n] = R_x^*[m, n]$.

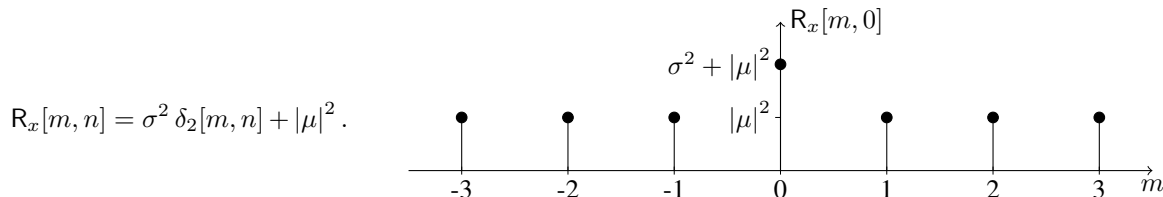
Proof: $R_x[-m, -n] = R_x[0 - m, 0 - n; 0, 0] = R_x[0, 0; m, n] = R_x^*[m, n; 0, 0] = R_x^*[m, n]$

- **Cauchy-Schwarz:** $|R_x[m, n]| \leq R_x[0, 0]$.

The behavior of the autocorrelation function greatly affects the characteristics of a random process, as seen in Fig. 11.1.

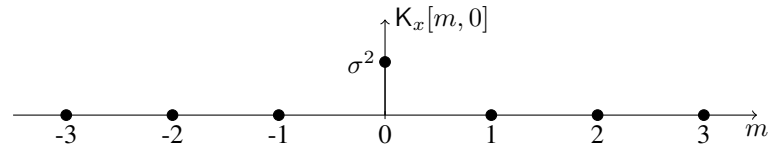
Note that mathematically a WSS random process is defined for all $m, n \in \mathbb{Z}$, in part so that (11.3) is well defined. In practice we work with a finite-sized image and it would be imprecise to say that finite-sized image is a WSS random process; instead it would be more precise to say that such a finite-sized image is a *portion* of an infinite extent WSS random process.

Example. A WSS **white random process** with variance σ^2 has the following autocorrelation function (see Fig. 11.1):



Its autocovariance function is

$$K_x[m, n] = \sigma^2 \delta_2[m, n].$$



In the often-assumed case of zero-mean noise, these two functions become identical.

If a zero-mean 2D WSS random process $x[m, n]$ with variance $\text{Var}\{x[m, n]\} = 3$ is up-sampled (by zero insertion), is the resulting process $y[m, n] = x_{\uparrow 2}[m, n]$ WSS?

??

[RQ]

Power spectral density of WSS processes

It is also important to consider the behavior of WSS random processes in the frequency domain. A naive attempt at frequency-domain analysis would be to try to analyze the DSFT of $x[m, n]$. However, WSS processes have infinite extent, and for almost all realizations the DSFT of $x[m, n]$ will not exist! (Of course the DSFT of any finite rectangular portion of $x[m, n]$ will always exist, but that DSFT will not generally have the properties of primary interest.)

So we will never refer to the “DSFT $X(\Omega_1, \Omega_2)$ ” of a WSS process!

Instead we consider the DSFT of the **autocorrelation function** of a WSS random process, which is called its **power spectrum** or **power spectral density**:

$$R_x[m, n] \xleftrightarrow{\text{DSFT}} P_x(\Omega_1, \Omega_2) \triangleq \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} R_x[m, n] e^{-i(\Omega_1 m + \Omega_2 n)}.$$

This relationship is made rigorous by the **Wiener-Khinchin theorem** [2] sketched in (11.28) on p. 11.54.

- The wider the power spectrum, the more high frequency fluctuations the random process has on average, which typically means that neighboring values are less correlated.
- Because the autocorrelation function of a WSS random process is **Hermitian symmetric**, the power spectrum is *real*.
- Power spectra are always **nonnegative**. (For a proof, see (11.5) below.)

Example. A zero-mean WSS **white random process** $x[m, n]$ with variance σ^2 has a constant (flat) power spectrum:

$$R_x[m, n] = K_x[m, n] = \sigma^2 \delta_2[m, n] \xleftrightarrow{\text{DSFT}} P_x(\Omega_1, \Omega_2) = \sigma^2.$$

If $x[m, n]$ has nonzero mean μ , then

$$K_x[m, n] = \sigma^2 \delta_2[m, n] \xleftrightarrow{\text{DSFT}} \sigma^2, \quad (\text{We do not have a name for the DSFT of } K_x[m, n].)$$

$$R_x[m, n] = \sigma^2 \delta_2[m, n] + |\mu|^2 \xleftrightarrow{\text{DSFT}} P_x(\Omega_1, \Omega_2) = \sigma^2 + |\mu|^2 (2\pi)^2 \delta_2((\Omega_1, \Omega_2))_{(2\pi, 2\pi)}.$$

Note: the term “white” implies that the process has a constant power spectrum (except possibly for a Dirac impulse at 0), and saying that it has a power spectrum means that it is WSS. So we usually say “white noise” rather than “WSS white noise” because WSS is implied.

Is the term “white” more reasonable for images or for audio? **Neither!** White light contains all (visible) *wavelengths* but here we are considering the *spatial frequencies*, not wavelengths.

Example. Fig. 11.1 shows realizations of three different WSS random processes and estimates of their autocorrelation functions. Also shown are scatter plots of the pairs $(x[m, n], x[m - 1, n - 1])$ and $(x[m, n], x[m - 1, n])$ to illustrate how neighboring pixel values are correlated. Why is the bottom right scatter plot isotropic?

??

[RQ]

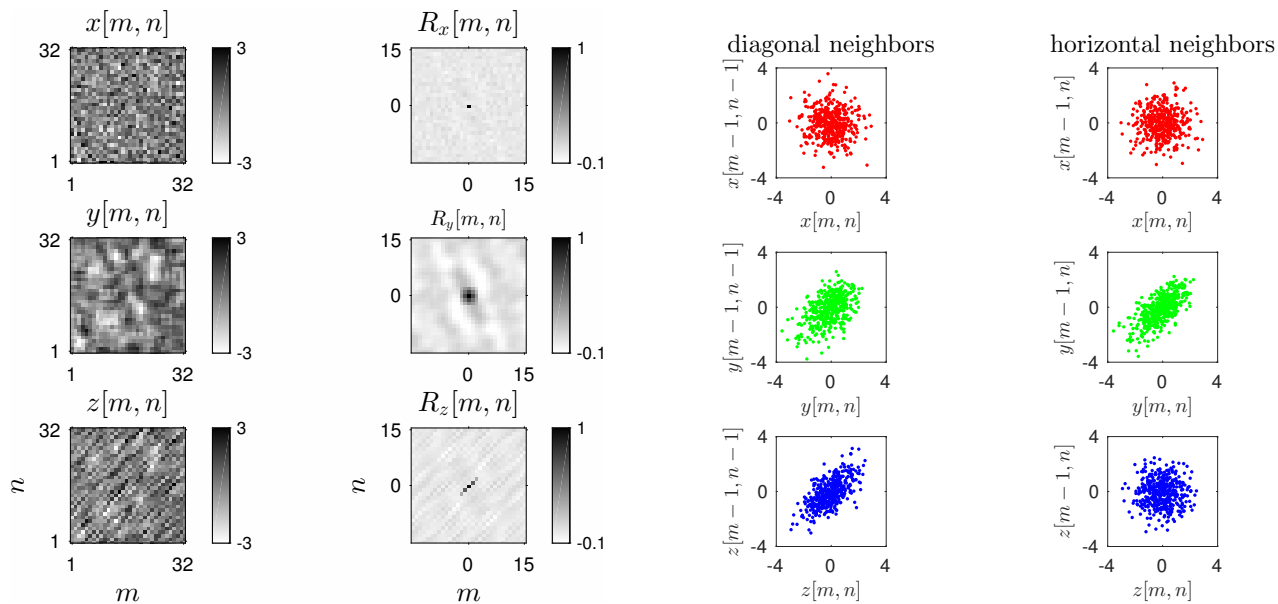


Figure 11.1: Examples of WSS random processes, their empirically estimated autocorrelation functions, *e.g.*, $\hat{R}_x[m, n]$, etc., and scatter plots for neighboring pixel pairs.

Nonnegativity and positive-semidefinite functions [skip]
**random variables**

It is well known that variances are nonnegative, because variance is defined as an integral of nonnegative quantities:

$$\sigma^2 = \mathbb{E}\left[|X - \mu|^2\right] = \int |x - \mu|^2 p_X(x) dx.$$

We write $\sigma^2 \geq 0$. We also need the generalizations of this inequality to random vectors and random processes.

random vectors

A square $N \times N$ matrix \mathbf{A} is called **nonnegative definite** or **positive semidefinite** iff $\mathbf{v}'\mathbf{A}\mathbf{v} \geq 0$ for all $\mathbf{v} \in \mathbb{C}^N$. We write $\mathbf{A} \succeq \mathbf{0}$.

Claim. Any **covariance matrix** is a positive-semidefinite matrix.

Proof. A covariance matrix has the form $\mathbf{K} = \mathbb{E}[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})']$ so

$$\mathbf{v}'\mathbf{K}\mathbf{v} = \mathbf{v}'\mathbb{E}[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})']\mathbf{v} = \mathbb{E}\left[\left(\mathbf{v}'(\mathbf{x} - \boldsymbol{\mu})\right)\left((\mathbf{x} - \boldsymbol{\mu})'\mathbf{v}\right)\right] = \mathbb{E}\left[|\mathbf{v}'(\mathbf{x} - \boldsymbol{\mu})|^2\right] \geq 0.$$

Corollary. Any **correlation matrix** $\mathbf{R} = \mathbb{E}[\mathbf{x}\mathbf{x}']$ is a positive-semidefinite matrix.

To transition from random vectors to random processes, note that $\mathbf{v}'\mathbf{K}\mathbf{v} = \sum_{i=1}^N \sum_{j=1}^N v_i^* k_{ij} v_j$.

1D random processes

A function $a : \mathbb{Z} \times \mathbb{Z} \mapsto \mathbb{C}$, written as $a[n; m]$, is called a **positive-semidefinite** function iff

$$\langle \mathbf{A}\mathbf{v}, \mathbf{v} \rangle = \sum_{n=-\infty}^{\infty} v^*[n] \left[\sum_{m=-\infty}^{\infty} a[n; m] v[m] \right] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} v^*[n] a[n; m] v[m] \geq 0$$

for all functions $v : \mathbb{Z} \mapsto \mathbb{C}$ for which the above summation converges.

Claim. The autocorrelation function $R_x[n; m] = E[x[n] x^*[m]]$ of any 1D random process $x[n]$ is a positive-semidefinite function.

Proof:

$$\begin{aligned} \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} v^*[n] R_x[n; m] v[m] &= \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} v^*[n] E[x[n] x^*[m]] v[m] \\ &= E \left[\left(\sum_{n=-\infty}^{\infty} v^*[n] x[n] \right) \left(\sum_{m=-\infty}^{\infty} v[m] x^*[m] \right) \right] = E \left[\left| \sum_{n=-\infty}^{\infty} v^*[n] x[n] \right|^2 \right] \geq 0. \end{aligned}$$

Corollary. The autocovariance function $K_x[n; m] = E[(x[n] - \mu_x[n])(x[m] - \mu_x[m])^*]$ of a 1D random process $x[n]$ is a positive-semidefinite function. (Note that we have *not* assumed WSS here.)

2D random processes

A function $a : (\mathbb{Z} \times \mathbb{Z}) \times (\mathbb{Z} \times \mathbb{Z}) \mapsto \mathbb{C}$ is called a **positive-semidefinite** function iff

$$\begin{aligned} \langle Av, v \rangle &= \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} v^*[m, n] \left(\sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} a[m, n; k, l] v[k, l] \right) \\ &= \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} v^*[m, n] a[m, n; k, l] v[k, l] \geq 0, \end{aligned}$$

for all functions $v : \mathbb{Z} \times \mathbb{Z} \mapsto \mathbb{C}$ for which the above summation converges.

Claim. The autocorrelation function $R_x[m, n; k, l] = E[x[m, n] x^*[k, l]]$ of a 2D random process $x[m, n]$ is a positive-semidefinite function.

Proof: (following analogous intermediate steps as in 1D)

$$\sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} v^*[m, n] R_x[m, n; k, l] v[k, l] = \mathbb{E} \left[\left| \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} v^*[m, n] x[m, n] \right|^2 \right] \geq 0.$$

Corollary. The autocovariance function $K_x[m, n; k, l] = \mathbb{E}[(x[m, n] - \mu_x[m, n])(x[k, l] - \mu_x[k, l])^*]$ of a 2D random process $x[m, n]$ is a positive-semidefinite function. (Again we have not assumed WSS here.)

2D WSS random processes

If $x[m, n]$ is a WSS random process, then for any $v[m, n]$ for which the following sums converge:

$$\begin{aligned} 0 &\leq \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} v^*[m, n] R_x[m, n; k, l] v[k, l] \\ &= \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} v^*[m, n] R_x[m - k, n - l] v[k, l] \\ &= \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} v^*[m, n] (v[\cdot, \cdot] ** R_x[\cdot, \cdot])[m, n] \quad \text{Why the next equality? Parseval} \\ &= \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} V^*(\Omega_1, \Omega_2) V(\Omega_1, \Omega_2) P_x(\Omega_1, \Omega_2) d\Omega_1 d\Omega_2 = \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} |V(\Omega_1, \Omega_2)|^2 P_x(\Omega_1, \Omega_2) d\Omega_1 d\Omega_2, \end{aligned}$$

where $v[m, n] \xleftrightarrow{\text{DSFT}} V(\Omega_1, \Omega_2)$. Thus, $R_x[m, n]$ is positive-semidefinite if and only if $P_x(\Omega_1, \Omega_2)$ is nonnegative.

We have shown already that all autocorrelation functions are positive semidefinite, so we have shown the following. ★★

The power spectral density function of any WSS random process is nonnegative. (11.5)

One can show easily that $P_x(\Omega_1, \Omega_2) \geq 0 \implies |R_x[m, n]| \leq R_x[0, 0]$, so the Cauchy-Schwarz inequality is a weaker requirement than nonnegativity of the power spectrum.

Example. Does there exist a 1D random process $x[n]$ with autocorrelation function $R_x[n] = [2/3 \ 1 \ 2/3]$?

??

(class/team)

Example. What about $R_y[n] = (2/3)^{|n|}$?

Yes, because $p^{|n|} \xleftrightarrow{\text{DTFT}} \frac{1/p - p}{p + 1/p - 2 \cos \Omega} > 0$ if $|p| < 1$.

Example. What can we say about the mean μ_x of the WSS random process $x[m, n]$ having autocorrelation function ◆◆

$R_x[m, n] = \cos^2(\Omega_0 n)$ where $\Omega_0 \neq 0$? Note that $R_x[m, n] = 1/2 + 1/2 \cos(2\Omega_0 n)$ has a nonnegative DSFT. Hint: use (11.4).

But $K_x[m, n] = R_x[m, n] - |\mu_x|^2 = (1/2 - |\mu_x|^2) + 1/2 \cos(2\Omega_0 n)$ has a nonnegative DSFT iff $|\mu_x|^2 < 1/2$.

Pairs of random processes

★★

Because we will consider both input and output images for LSI systems, as well as additive noise, we need to have tools to analyze *multiple* random processes simultaneously.

If $x[m, n]$ and $y[m, n]$ are two jointly distributed random processes, we define their **cross-correlation function** as follows²:

$$R_{xy}[m, n; k, l] = E[x[m, n] y^*[k, l]]. \quad (11.6)$$

Properties:

- **Hermitian symmetry** (Caution!):

$$R_{yx}^*[m, n; k, l] = R_{xy}[k, l; m, n].$$

- Linearity (or conjugate linearity):

$$\begin{aligned} R_{(u+v),x}[m, n; k, l] &= E[(u[m, n] + v[m, n]) x^*[k, l]] \\ &= E[u[m, n] x^*[k, l]] + E[v[m, n] x^*[k, l]] \\ &= R_{ux}[m, n; k, l] + R_{vx}[m, n; k, l]. \end{aligned}$$

More generally cross-correlation is **bilinear** or **sesquilinear**:

$$R_{\sum_i x_i, \sum_j y_j}[m, n; k, l] = E \left[\left(\sum_i x_i[m, n] \right) \left(\sum_j y_j^*[k, l] \right) \right] = \sum_i \sum_j E[x_i[m, n] y_j^*[k, l]] = \sum_i \sum_j R_{x_i, y_j}[m, n; k, l]$$

- Auto-correlation:

$$R_x[m, n; k, l] \triangleq R_{xx}[m, n; k, l]$$

² For deterministic complex signals, the usual definition of **cross-correlation** puts the complex conjugate on the *first* signal, as defined in Ch. 2. For random processes, many books use the convention adopted in (11.6) where the conjugate is on the **second** signal. These definitions are not universal, so watch out when reading papers.

- Cauchy-Schwarz inequality:

$$|\mathbf{R}_{xy}[m, n; k, l]| \leq \sqrt{\mathbf{R}_x[m, n; m, n] \mathbf{R}_y[k, l; k, l]}$$

Jointly WSS random processes

★★

We say $x[m, n]$ and $y[m, n]$ are **jointly WSS**, iff $x[m, n]$ and $y[m, n]$ are individually WSS and

$$\mathbf{R}_{xy}[m, n; k, l] = \mathbf{R}_{xy}[m - k, n - l; 0, 0] = \mathbf{R}_{xy}[m - k, n - l], \quad \forall m, n, k, l \in \mathbb{Z},$$

where in this case we define

$$\mathbf{R}_{xy}[m, n] \triangleq \mathbf{R}_{xy}[m, n; 0, 0] = \mathbf{E}[x[m, n] y^*[0, 0]].$$

For jointly WSS processes we have the following form of Hermitian symmetry (Caution!):

$$\mathbf{R}_{yx}^*[-m, -n] = \mathbf{R}_{xy}[m, n], \quad \forall m, n \in \mathbb{Z}.$$

Example. If $x[m, n]$ is WSS and we let $y[m, n] = x[-m, n]$ then $y[m, n]$ is also WSS.

Are $x[m, n]$ and $y[m, n]$ **jointly WSS**? ??

(class/team)

For jointly WSS random processes, we define the **cross power spectrum** to be the DSFT of the cross-correlation function:

$$\mathbf{R}_{xy}[m, n] \xleftrightarrow{\text{DSFT}} P_{xy}(\Omega_1, \Omega_2).$$

Properties of cross power spectrum (inherited from those of the cross-correlation function)

- Linearity
- Caution: in general $\mathbf{R}_{xy}^*[-n, -m] \neq \mathbf{R}_{xy}[m, n]$, so we cannot conclude that $P_{xy}(\Omega_1, \Omega_2)$ is real, despite the unfortunate word “power” in its name. Nor can we conclude it is nonnegative. All we can conclude is the following Hermitian symmetry property:

$$P_{xy}(\Omega_1, \Omega_2) = P_{yx}^*(\Omega_1, \Omega_2).$$



Uncorrelated random processes

We say $x[m, n]$ and $y[m, n]$ are **uncorrelated random processes** iff

$$R_{xy}[m, n; k, l] = E[x[m, n]] E[y^*[k, l]], \quad \forall m, n, k, l \in \mathbb{Z}.$$

If $x[m, n]$ and $y[m, n]$ are WSS and uncorrelated, then it follows that

$$\begin{aligned} R_{x+y}[m, n] &= R_x[m, n] + R_y[m, n] + \mu_x \mu_y^* + \mu_x^* \mu_y \\ K_{x+y}[m, n] &= K_x[m, n] + K_y[m, n]. \end{aligned}$$

If in addition at least one of the (uncorrelated) processes is zero mean, then it follows that their autocorrelation functions and power spectra add:

$$\boxed{R_{x+y}[m, n] = R_x[m, n] + R_y[m, n]} \quad \& \quad \boxed{P_{x+y}(\Omega_1, \Omega_2) = P_x(\Omega_1, \Omega_2) + P_y(\Omega_1, \Omega_2)}. \quad (11.7) \quad \star\star\star$$

Signal and noise random processes are usually modeled as uncorrelated so these addition properties are particularly important.

Random processes through LSI systems

For denoising applications, we will consider filtering images. What happens to jointly distributed random processes $w[m, n]$ and $x[m, n]$ after being passed through filters?

$$\begin{aligned} x[m, n] &\rightarrow \boxed{h[m, n]} \rightarrow y[m, n] = h[m, n] ** x[m, n] \\ w[m, n] &\rightarrow \boxed{g[m, n]} \rightarrow z[m, n] = g[m, n] ** w[m, n]. \end{aligned}$$

Firstly, by linearity,

$$\mu_y[m, n] = h[m, n] ** \mu_x[m, n].$$

It is more interesting to examine the cross-correlation:

$$\begin{aligned} R_{yz}[m, n; k, l] &= \mathbb{E}[y[m, n] z^*[k, l]] \\ &= \mathbb{E}[(h[m, n] ** x[m, n]) (g[k, l] ** w[k, l])^*] \\ &= \mathbb{E} \left[\left(\sum_{m'=-\infty}^{\infty} \sum_{n'=-\infty}^{\infty} h[m', n'] x[m - m', n - n'] \right) \left(\sum_{m''=-\infty}^{\infty} \sum_{n''=-\infty}^{\infty} g^*[m'', n''] w^*[k - m'', l - n''] \right) \right] \\ &= \sum_{m'=-\infty}^{\infty} \sum_{n'=-\infty}^{\infty} h[m', n'] \sum_{m''=-\infty}^{\infty} \sum_{n''=-\infty}^{\infty} g^*[m'', n''] \mathbb{E}[x[m - m', n - n'] w^*[k - m'', l - n'']] \\ &= \sum_{m'=-\infty}^{\infty} \sum_{n'=-\infty}^{\infty} h[m', n'] \sum_{m''=-\infty}^{\infty} \sum_{n''=-\infty}^{\infty} g^*[m'', n''] R_{xw}[m - m', n - n'; k - m'', l - n'']. \end{aligned}$$

There is little more we can say about this in general, so we focus on jointly WSS processes next.

If the inputs $x[m, n]$ and $w[m, n]$ are **jointly WSS**, then the above result simplifies to:

$$\begin{aligned} R_{yz}[m, n; k, l] &= \mathbb{E}[y[m, n] z^*[k, l]] \\ &= \sum_{m'=-\infty}^{\infty} \sum_{n'=-\infty}^{\infty} h[m', n'] \sum_{m''=-\infty}^{\infty} \sum_{n''=-\infty}^{\infty} g^*[m'', n''] R_{xw}[m - m' - (k - m''), n - n' - (l - n'')]. \end{aligned}$$

- Because this expression depends only on the differences $m - k$ and $n - l$,

the output random processes are jointly WSS.

This is a generalization of the “WSS in \implies WSS out” property of LSI systems.

- Equivalently:

$$R_{yz}[m, n] = h[m, n] ** g^*[-m, -n] ** R_{xw}[m, n], \quad (11.8)$$

- or in the spectral domain:

$$P_{yz}(\Omega_1, \Omega_2) = H(\Omega_1, \Omega_2) G^*(\Omega_1, \Omega_2) P_{xw}(\Omega_1, \Omega_2), \quad (11.9)$$

recalling the following **symmetry properties** of the DSFT from Ch. 5:

$$g[m, n] \stackrel{\text{DSFT}}{\longleftrightarrow} G(\Omega_1, \Omega_2) \implies \begin{aligned} &g[-m, -n] \stackrel{\text{DSFT}}{\longleftrightarrow} G(-\Omega_1, -\Omega_2) \\ &g^*[m, n] \stackrel{\text{DSFT}}{\longleftrightarrow} G^*(-\Omega_1, -\Omega_2) \\ &g^*[-m, -n] \stackrel{\text{DSFT}}{\longleftrightarrow} G^*(\Omega_1, \Omega_2). \end{aligned}$$

Special case 1

$$x[m, n] \rightarrow \boxed{\text{LSI } h[m, n]} \rightarrow y[m, n] = h[m, n] ** x[m, n].$$

Relate behavior of output to behavior of input.

Again, if WSS input, then WSS output.

Mean function:

$$\mathbb{E}[y[m, n]] = \mathbb{E}[h[m, n] ** x[m, n]] = h[m, n] ** \mathbb{E}[x[m, n]] = h[m, n] ** \mu_x[m, n].$$

If zero-mean input, then zero-mean output.

Auto-correlation function:

$$\begin{aligned} R_y[m, n; k, l] &= \mathbb{E}[y[m, n] y^*[k, l]] \\ &= \mathbb{E}[(h[m, n] ** x[m, n])(h[k, l] ** x[k, l])^*] \\ &= \dots \end{aligned}$$

By taking $w[m, n] = x[m, n]$ and $g[m, n] = h[m, n]$ (so $z[m, n] = y[m, n]$) in (11.8) and (11.9), we get

$$\boxed{R_y[m, n] = R_{h**x}[m, n] = h[m, n] ** h^*[-m, -n] ** R_x[m, n],} \quad (11.10)$$

$$\boxed{P_y(\Omega_1, \Omega_2) = P_{h**x}(\Omega_1, \Omega_2) = H(\Omega_1, \Omega_2) H^*(\Omega_1, \Omega_2) P_x(\Omega_1, \Omega_2) = |H(\Omega_1, \Omega_2)|^2 P_x(\Omega_1, \Omega_2).} \quad (11.11)$$

This latter formula is perhaps the best justification of the term “power spectrum.”

What is our terminology for the expression $h[m, n] ** h^*[-m, -n]$? ??

[RQ]

Special case 2

By taking $g[m, n] = \delta_2[m, n]$ (so $z[m, n] = w[m, n]$) in (11.8) and (11.9), we get

★★

$$\boxed{R_{yw}[m, n] = R_{h^{**x}, w}[m, n] = h[m, n] ** R_{xw}[m, n]} \quad (11.12)$$

$$\boxed{P_{yw}(\Omega_1, \Omega_2) = P_{h^{**x}, w}(\Omega_1, \Omega_2) = H(\Omega_1, \Omega_2) P_{xw}(\Omega_1, \Omega_2)}. \quad (11.13)$$

Synthesizing random vectors and processes numerically [skip]



Synthesizing random vectors

```

int getRandomNumber()
{
    return 4; // chosen by fair dice roll.
             // guaranteed to be random.
}

```

<http://xkcd.com/221>

This section describes methods for computer synthesis of random vectors and random processes. How can we simulate in MATLAB a WSS random process with a given desired autocorrelation function, so that we can “see what it looks like” and study it empirically?

First consider the simpler question of how to generate Gaussian random variables with arbitrary variance σ^2 . If z is a standard Gaussian random variable, *i.e.*, $z \sim \mathcal{N}(0, 1)$ then if we define $y = sz$ for some constant s , then $\text{Var}\{y\} = s^2 \text{Var}\{z\} = s^2$. In MATLAB we can generate Gaussian random variables with unit variance using the `randn` function. So

$$y = \text{sig} * \text{randn}(1),$$

where $\text{sig} \equiv \sigma$, will generate a random variable y with variance σ^2 . Note that in the code we use the value $\sigma = \sqrt{\sigma^2}$, the *square root* of the variance, which is well defined because variances are nonnegative.

Now we generalize this result by considering how to generate a random vector \mathbf{y} with covariance \mathbf{K} . Any (Hermitian symmetric) positive-semidefinite matrix \mathbf{K} has an orthonormal eigenvector decomposition $\mathbf{K} = \mathbf{V} \text{diag}\{\lambda_i\} \mathbf{V}'$ with nonnegative eigenvalues λ_i , and a corresponding a matrix **square root** $\mathbf{S} \triangleq \mathbf{K}^{1/2} = \mathbf{V} \text{diag}\{\sqrt{\lambda_i}\} \mathbf{V}'$ satisfying $\mathbf{S}' = \mathbf{S}$, such that

$$\mathbf{K} = \mathbf{S}\mathbf{S}.$$

So if \mathbf{z} is a random vector whose covariance matrix is the identity matrix, *i.e.*, $\text{Cov}\{\mathbf{z}\} = \mathbf{I}$, then $\mathbf{y} = \mathbf{S}\mathbf{z}$ has covariance

$$\text{Cov}\{\mathbf{y}\} = \text{Cov}\{\mathbf{S}\mathbf{z}\} = \mathbf{S} \text{Cov}\{\mathbf{z}\} \mathbf{S}' = \mathbf{S}\mathbf{I}\mathbf{S}' = \mathbf{S}\mathbf{S} = \mathbf{K}.$$

So we can generate random vectors with any such covariance matrix easily, because typical pseudo random number generators make independent realizations. In MATLAB:

$$\mathbf{S} = \text{sqrtm}(\mathbf{K}); \mathbf{y} = \mathbf{S} * \text{randn}(\text{size}(\mathbf{K},1),1);$$

It is crucial to use `sqrtm` here *not* `sqrt` to get the correct results.

(One could use the statistics toolbox command `mvnrnd` instead.)

Synthesizing random processes

Similarly, a power spectrum also has a square root, so one can generate WSS random processes with any desired (legitimate) autocorrelation function by filtering white noise.

In MATLAB we can generate a white Gaussian random process using the `randn` function. (It is an approximation due to the use of pseudo-random number generators, but that is a small detail.) The output of $\mathbf{z} = \text{randn}(\dots)$ essentially is a (portion of a) random process $z[m, n]$ with autocorrelation function

$$R_z[m, n] = \delta_2[m, n].$$

Now consider (11.10). This equation says that if we convolve $z[m, n]$ with a filter with impulse response $h[m, n]$ to form a new random process $y[m, n]$, then the autocorrelation function of the output signal will be

$$R_y[m, n] = h[m, n] ** h^*[-m, -n]. \quad (11.14)$$

So to synthesize a random process $y[m, n]$ that has a given autocorrelation function $R_y[m, n]$, we just need to find a filter $h[m, n]$ that satisfies (11.14). Taking the FT of both sides:

$$P_y(\Omega_1, \Omega_2) = |H(\Omega_1, \Omega_2)|^2,$$

so *any* filter whose magnitude response is the square root of the desired power spectrum will suffice:

$$|H(\Omega_1, \Omega_2)| = \sqrt{P_y(\Omega_1, \Omega_2)}. \quad (11.15)$$

The corresponding impulse response is

$$h[m, n] = \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \sqrt{P_y(\Omega_1, \Omega_2)} e^{j\theta(\Omega_1, \Omega_2)} e^{j(\Omega_1 m + \Omega_2 n)} d\Omega_1 d\Omega_2,$$

where $\theta(\Omega_1, \Omega_2)$ is any phase function we wish to choose! Typically we prefer zero-phase filters, so we usually choose $\theta(\Omega_1, \Omega_2)$ to equal some integer multiple of π for every (Ω_1, Ω_2) . Often we would like to choose $\theta(\Omega_1, \Omega_2)$ in such a way that makes $h[m, n]$ FIR or approximately FIR.

Of course it is also possible to (approximately) generate a random process with any desired power spectrum by working in the frequency domain using FFT operations.

Example. Fig. 11.1 shows random processes synthesized by convolving white noise with various filters $h[m, n]$.

2nd-order statistics - an incomplete story

Fig. 11.1 may leave the impression that one can learn a lot about a random process from its autocorrelation function, and vice versa. This is generally true, but the autocorrelation function does not *uniquely* determine the properties of a random process.

Fig. 11.2 shows two very different random processes with *identical* autocorrelation functions. The first random process $y[m, n]$ was generated by filtering white Gaussian noise with the separable filter $h[m, n] = h_1[m] h_1[n]$, where $h_1[n] = [1 \ 1 \ \underline{1} \ 1 \ 1]$. By (11.10), the resulting autocorrelation function is

$$R_y[m, n] = h[m, n] \star h[m, n] = (h_1[m] h_1[n]) \star (h_1[m] h_1[n]) = (h_1[m] \star h_1[m]) (h_1[n] \star h_1[n]) = \Lambda_5(m) \Lambda_5(n),$$

where $\Lambda_5(n) \triangleq [1 \ 2 \ 3 \ 4 \ \underline{5} \ 4 \ 3 \ 2 \ 1]$. So $R_y[m, n]$ looks like a pyramid with a peak of 25 at the origin.

The second random process $z[m, n]$ was generated by the following method:

$$z[m, n] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \left(\frac{x[k, l] - p}{\sqrt{p(1-p)}} \right) h[m - k, n - l], \quad (11.16)$$

where $x[m, n]$ is an **independent and identically distributed (IID)** Bernoulli random process:

$$x[m, n] = \begin{cases} 1, & \text{w.p. } p \\ 0, & \text{w.p. } 1 - p, \end{cases}$$

where **w.p.** means *with probability*. Any value of $p \in (0, 1)$ leads to the same autocorrelation function $R_z[m, n] = R_y[m, n]$.

Exercise. Show analytically that this random process $z[m, n]$ has the same autocorrelation function as $y[m, n]$. (HW)

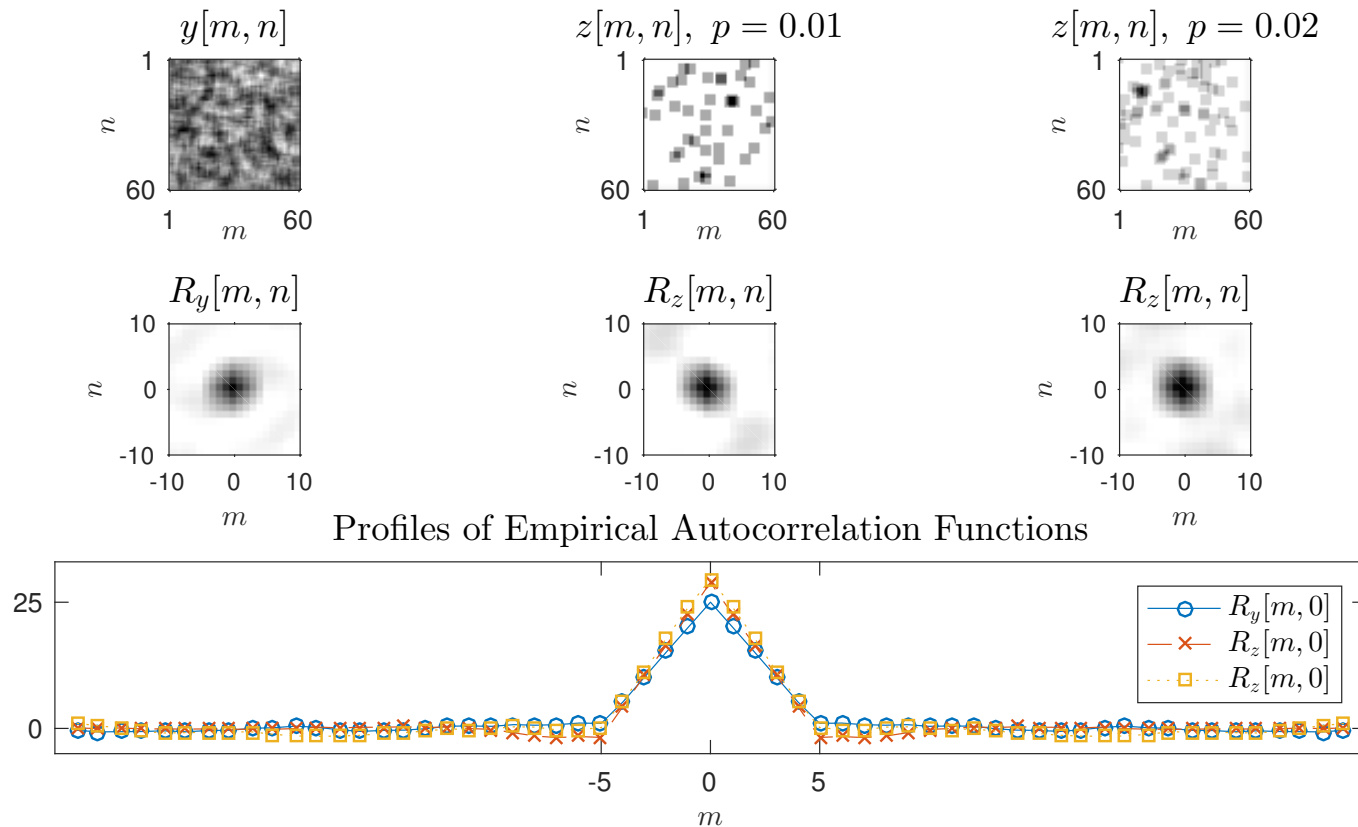


Figure 11.2: Drastically different random processes with identical autocorrelation functions.

Here is the MATLAB code that made those images.

```
% fig_auto3.m
% show different random fields with identical auto-correlation functions

xcorr2 = @(x,y) conv2(x, rot90(conj(y), 2));
rng(0)
nx = 60; ny = nx;

x0 = randn(nx,ny); % white gaussian random process
h1 = ones(5);
x1 = conv2(x0, h1, 'same'); % filtered

p2 = 0.01;
h2 = h1 / sqrt(p2*(1-p2));
x2 = (rand(nx,ny) < p2) - p2; % bernoulli random process
x2 = conv2(x2, h2, 'same'); % filtered
p3 = 0.02;
h3 = h1 / sqrt(p3*(1-p3));
x3 = (rand(nx,ny) < p3) - p3; % same with a different value of p
x3 = conv2(x3, h3, 'same');

r1 = xcorr2(x1, x1) / (nx*ny); % empirical autocorrelation estimate
r1 = r1(1+nx/2:end-nx/2,1+ny/2:end-ny/2);

r2 = xcorr2(x2, x2) / (nx*ny);
r2 = r2(1+nx/2:end-nx/2,1+ny/2:end-ny/2);

r3 = xcorr2(x3, x3) / (nx*ny);
r3 = r3(1+nx/2:end-nx/2,1+ny/2:end-ny/2);

fig_auto3_plot
```

11.2 The noncausal Wiener filter for “optimal” image denoising

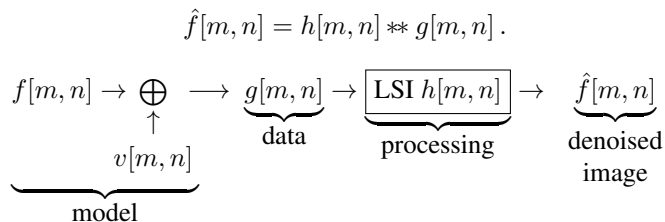
The **Wiener filter** is a classic method for attempting to remove noise from images. It was developed (for 1D applications) by Norbert Wiener in the 1930’s and 1940’s. It is now a *de facto* “straw man” method against which researchers compare new more sophisticated methods (often nonlinear) that they develop for denoising. The derivation assumes an additive noise **model**:

$$g[m, n] = f[m, n] + v[m, n],$$

where $g[m, n]$ is the observed (noisy) image, $v[m, n]$ is additive random noise, and $f[m, n]$ is the true but unknown underlying image. Note: in this model there is no **blur**, only additive noise. Generalizing to include blur is an important exercise.

Goal: we would like to recover (estimate) $f[m, n]$ from $g[m, n]$.

There are many approaches to “solving” this problem. An approach that is convenient both for analysis and for implementation is to use linear shift-invariant filtering with some filter $h[m, n]$ that we must design:



How should we choose this filter $h[m, n]$? Of course we would like to find $h[m, n]$ such that $h[m, n] ** g[m, n] = f[m, n]$, because such a filter, if it existed, would exactly recover the true signal. Due to the additive noise $v[m, n]$, no such filter exists in general—there will always be error between $\hat{f}[m, n]$ and $f[m, n]$.

A criterion that is convenient both for analysis and for implementation is to choose the filter $h[m, n]$ that provides the **minimum mean squared error (MMSE)** between the estimated signal and the true signal:

$$\text{MSE}_h = \mathbb{E} \left[\left| \hat{f}[m, n] - f[m, n] \right|^2 \right].$$

This linear MMSE approach requires that we

- assume that the measurements $g[m, n]$ and signal $f[m, n]$ are jointly WSS,
- know the autocorrelation function $R_f[m, n]$ of the true signal $f[m, n]$,
- know the autocorrelation function $R_g[m, n]$ of the measurements $g[m, n]$, and
- know the cross-correlation function $R_{fg}[m, n]$ between the signal and measurement processes.
- We also assume that the signals are **zero mean** for simplicity, although generalizations are possible.

We will return later to examine how these correlation functions can be determined.

One can apply the **orthogonality principle** to derive the “optimal” filter $h_{\text{opt}}[m, n]$ that minimizes MSE_h . An alternative “elementary” derivation is as follows:

$$\begin{aligned} \text{MSE}_h &= \mathbb{E} \left[\left| \hat{f}[m, n] - f[m, n] \right|^2 \right] \\ &= \mathbb{E} \left[\left(\hat{f}[m, n] - f[m, n] \right) \left(\hat{f}[m, n] - f[m, n] \right)^* \right] \\ &= \mathbb{E} \left[\left| \hat{f}[m, n] \right|^2 \right] - \mathbb{E} \left[\hat{f}[m, n] f^*[m, n] \right] - \mathbb{E} \left[f[m, n] \hat{f}^*[m, n] \right] + \mathbb{E} \left[|f[m, n]|^2 \right] \\ &= R_{\hat{f}}[0, 0] - R_{\hat{f}, f}[0, 0] - R_{f, \hat{f}}[0, 0] + R_f[0, 0] \\ &= \left(\frac{1}{2\pi} \right)^2 \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} P_{\hat{f}}(\Omega_1, \Omega_2) - P_{\hat{f}, f}(\Omega_1, \Omega_2) - P_{f, \hat{f}}(\Omega_1, \Omega_2) + P_f(\Omega_1, \Omega_2) \, d\Omega_1 \, d\Omega_2, \end{aligned}$$

by “time 0” property of the DSFT. Recall that we are trying to derive the optimal filter here, which means we want to find the filter $h[m, n]$ (or equivalently the frequency response $H(\Omega_1, \Omega_2)$) that minimizes MSE_h . The above MSE expression depends on $H(\Omega_1, \Omega_2)$, but in a way that is hidden from view. We need to bring $H(\Omega_1, \Omega_2)$ to view.

By “special case 1” (11.11):

$$P_{\hat{f}}(\Omega_1, \Omega_2) = |H(\Omega_1, \Omega_2)|^2 P_g(\Omega_1, \Omega_2),$$

and by “special case 2” (11.13):

$$P_{\hat{f}, f}(\Omega_1, \Omega_2) = H(\Omega_1, \Omega_2) P_{g, f}(\Omega_1, \Omega_2) = H(\Omega_1, \Omega_2) P_{fg}^*(\Omega_1, \Omega_2).$$

Substituting in and **completing the square**³ yields:

$$\text{MSE}_H = \left(\frac{1}{2\pi}\right)^2 \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} |H|^2 P_g - H P_{fg}^* - H^* P_{fg} + P_f \, d\Omega_1 \, d\Omega_2 \quad (11.17)$$

$$= \text{MSE}_{\text{opt}} + \left(\frac{1}{2\pi}\right)^2 \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} P_g \left| H - \frac{P_{fg}}{P_g} \right|^2 \, d\Omega_1 \, d\Omega_2, \quad (11.18)$$

where the following constant is independent of $H(\Omega_1, \Omega_2)$:

$$\text{MSE}_{\text{opt}} = \left(\frac{1}{2\pi}\right)^2 \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} P_f - \frac{|P_{fg}|^2}{P_g} \, d\Omega_1 \, d\Omega_2. \quad (11.19)$$

³ This is where the zero-mean assumption is invoked, because otherwise P_g would have a Dirac impulse at the origin, prohibiting us from dividing by P_g .

Because every term in the MSE integral (11.18) is nonnegative, the MSE is minimized if we can equate every term to zero. This leads to the solution for the optimal filter frequency response, called the noncausal **Wiener filter**: ★★

$$H(\Omega_1, \Omega_2) \triangleq \arg \min_H \text{MSE}_H, \quad H_{\text{opt}}(\Omega_1, \Omega_2) = \begin{cases} \frac{P_{fg}(\Omega_1, \Omega_2)}{P_g(\Omega_1, \Omega_2)}, & P_g(\Omega_1, \Omega_2) \neq 0 \\ 0, & P_g(\Omega_1, \Omega_2) = 0. \end{cases} \quad (11.20)$$

When using this optimal filter, the resulting minimum MSE is MSE_{opt} .

It is called **noncausal** because in general the corresponding impulse response $h[m, n]$ will be noncausal. This was quite important to Wiener, who was considering 1D prediction problems with defense applications (target tracking etc.), where causality would be quite important. (See EECS 564.)

Could the Wiener filter $H_{\text{opt}}(\Omega_1, \Omega_2)$ be complex valued in general? Why? **Yes, because $P_{fg}(\Omega_1, \Omega_2)$ can be complex.**

The form of the Wiener filter given in (11.20) above can be inconvenient because it depends on the power spectrum of $g[m, n]$ and the cross-power spectrum of $g[m, n]$ and $f[m, n]$, whereas in practice it would usually be the spectra of $f[m, n]$ and $v[m, n]$ that we “know” (or make assumptions about).

If $g[m, n] = f[m, n] + v[m, n]$ and the signal $f[m, n]$ and the noise $v[m, n]$ are **zero-mean** and **uncorrelated**, which are standard assumptions, then

$$P_g(\Omega_1, \Omega_2) = P_f(\Omega_1, \Omega_2) + P_v(\Omega_1, \Omega_2)$$

and

$$P_{fg}(\Omega_1, \Omega_2) = P_{f, f+v}(\Omega_1, \Omega_2) = P_f(\Omega_1, \Omega_2) + P_{fv}(\Omega_1, \Omega_2) = P_f(\Omega_1, \Omega_2),$$

and the Wiener filter (11.20) simplifies to ★★★

$$H(\Omega_1, \Omega_2) = \frac{P_f(\Omega_1, \Omega_2)}{P_f(\Omega_1, \Omega_2) + P_v(\Omega_1, \Omega_2)}, \quad (11.21)$$

with resulting minimum MSE

$$\text{MSE}_{\text{opt}} = \left(\frac{1}{2\pi}\right)^2 \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \frac{P_f P_v}{P_f + P_v} d\Omega_1 d\Omega_2.$$

Could the Wiener filter $H(\Omega_1, \Omega_2)$ in (11.21) be complex valued?

??

[RQ]

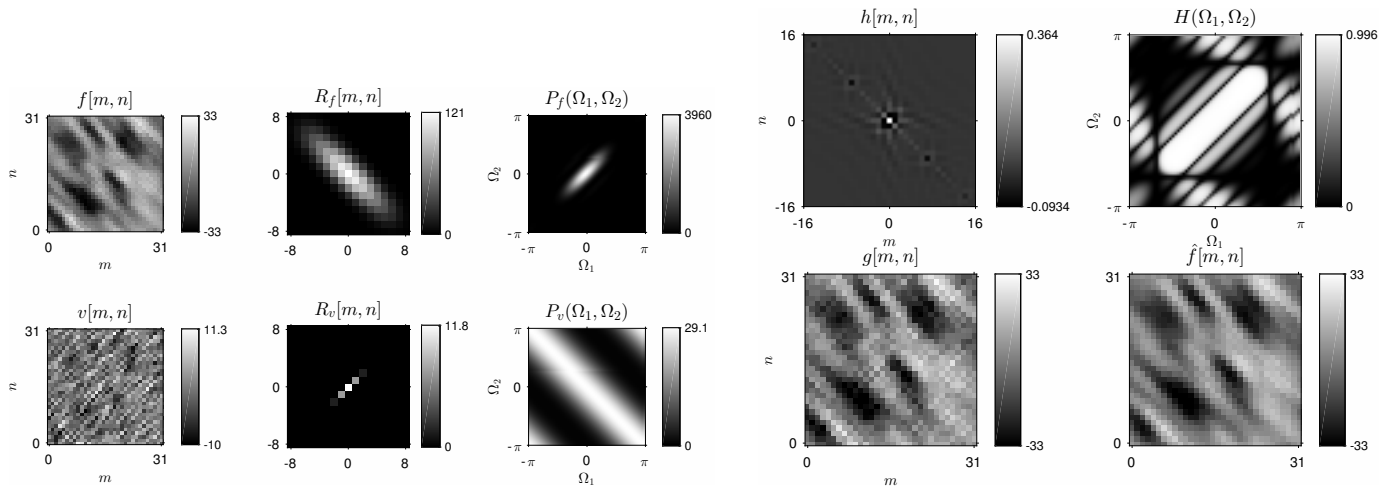
Interpretation:

- If $P_v(\Omega_1, \Omega_2) \ll P_f(\Omega_1, \Omega_2)$, then $H(\Omega_1, \Omega_2) \approx 1$. Why? If no/little noise power at that frequency, then no need to filter it out.
- If $P_v(\Omega_1, \Omega_2) \gg P_f(\Omega_1, \Omega_2)$, then $H(\Omega_1, \Omega_2) \approx 0$. Why? ?? (pair in class)

Note. If \mathbf{x} and \mathbf{y} are jointly Gaussian random vectors, then $E[\mathbf{x} | \mathbf{y}] = \boldsymbol{\mu}_x + \mathbf{K}_{x,y} \mathbf{K}_y^{-1} (\mathbf{y} - \boldsymbol{\mu}_y)$. The “ $\mathbf{K}_{x,y} \mathbf{K}_y^{-1}$ ” part of this expression is closely related to the P_{fg} / P_g ratio in (11.20).

Example. The following figures represent an artificial case where everything fits the Wiener theory. The spectral properties of the signal and the noise differ greatly, so the Wiener filter works better than it usually does in practice.

Explain why is $H(\Omega_1, \Omega_2)$ is nearly 1 for some frequencies and nearly zero for others. ??



The MATLAB command `fhats = deconvwnr(g, 1, vcorr, fcorr);` provides an FFT-based implementation of the Wiener filter (using periodic boundary conditions). The [documentation](#) claims “The algorithm is optimal in a sense of least mean square error between the estimated and the true images.” Is it? ?? (in class)

If the Wiener filter is “optimal” why are there so many modern alternatives published that improve upon it? (in class)

-
-
-

Stronger optimality conditions

We have seen that the Wiener filter is the minimum MSE estimator in the class of LSI estimators. Can we say anything stronger? What about nonlinear estimators?

If the signal and noise are jointly WSS *and* jointly Gaussian, then the MMSE estimator over *all* estimators is the **conditional mean** estimator:

$$\hat{f}[m, n] = \mathbf{E}[f[m, n] | g[\cdot]].$$

For Gaussian random processes this conditional expectation is linear (and also shift invariant for jointly WSS signal and noise) and this leads to exactly the same Wiener filter solution.

So for jointly WSS, jointly Gaussian signal and noise, the Wiener filter is the minimum MSE estimator over *all* estimators!

Unfortunately, the jointly WSS, jointly Gaussian, known autocorrelation functions, etc. rarely hold in practice.

Variations of Wiener filtering [skip]



One of many *ad hoc* variations is **power spectrum filtering**, where the following frequency response is used:

$$H(\Omega_1, \Omega_2) = \sqrt{\frac{P_f(\Omega_1, \Omega_2)}{P_f(\Omega_1, \Omega_2) + P_v(\Omega_1, \Omega_2)}}. \quad (11.22)$$

Why is this reasonable? Because when the signal $f[m, n]$ and noise $v[m, n]$ are uncorrelated:

$$P_{\hat{f}}(\Omega_1, \Omega_2) = |H(\Omega_1, \Omega_2)|^2 P_g(\Omega_1, \Omega_2) = \frac{P_f(\Omega_1, \Omega_2)}{P_f(\Omega_1, \Omega_2) + P_v(\Omega_1, \Omega_2)} [P_f(\Omega_1, \Omega_2) + P_v(\Omega_1, \Omega_2)] = P_f(\Omega_1, \Omega_2).$$

So this choice has the interesting property that the power spectrum of the estimated image matches that of the true image.

Other powers between 0 and 1 all give additional *ad hoc* variations on the Wiener filter.

- The filter (11.22) over-emphasizes low SNR regions (due to shape of sqrt function) so it is not a MMSE estimator.
- If the power spectrum is of great interest, then perhaps the problem should be formulated directly as a spectrum estimation problem!

Minimization of complex quadratic forms [skip] (a cautionary tale)
 ◆◆

In deriving the Wiener filter (11.20) and (11.21), we completed the square in (11.17). An alternative is to try to minimize w.r.t. the filter H by taking derivatives, but this is subtle because H might be complex.

Consider the following problem where r is real and c and d are complex:

$$\hat{z} = \arg \min_{z \in \mathbb{C}} f(z), \text{ where } f(z) = |c - dz|^2 + r|z|^2.$$

Usually for minimization problems we differentiate, set to zero, and solve. That fails in this problem because the above f is not an **analytic** function. It is not differentiable in the sense needed for complex numbers [3, p. 24].

Thus, we solve the minimization problem by returning to first principles.

Let $z = x + iy$, where x and y are real and $i = \sqrt{-1}$. Then

$$\begin{aligned} f(x, y) &= |c - d(x + iy)|^2 + r|x + iy|^2 \\ &= |c|^2 - c^*d(x + iy) - cd^*(x - iy) + |d|^2(x^2 + y^2) + r(x^2 + y^2) \\ \frac{\partial}{\partial x} f(x, y) &= -(c^*d + cd^*) + 2(|d|^2 + r)x \\ \frac{\partial}{\partial y} f(x, y) &= -i(c^*d - cd^*) + 2(|d|^2 + r)y. \end{aligned}$$

Equating to zero and solving yields

$$\begin{aligned} \hat{x} &= \frac{1}{|d|^2 + r} \frac{d^*c + dc^*}{2} = \frac{\text{real}\{d^*c\}}{|d|^2 + r} \\ \hat{y} &= \frac{1}{|d|^2 + r} \frac{d^*c - dc^*}{2i} = \frac{\text{imag}\{d^*c\}}{|d|^2 + r}, \end{aligned}$$

so the minimizing z is

$$\hat{z} = \hat{x} + \imath \hat{y} = \frac{d^* c}{|d|^2 + r}.$$

A naive attempt would be to ignore the fact that z is complex, obtaining $0 = \frac{d}{dz} [(c - dz)^2 + rz^2] = -2d(c - dz) + r2z$ for which $z = dc/(d^2 + r)$, which is wrong in the complex case!

We can also derive the Wiener filter (11.20) using the correct approach above.

Differentiating the integrand of (11.17):

$$|H|^2 P_g - H P_{fg}^* - H^* P_{fg} = |H|^2 P_g - 2 \operatorname{real}\{H P_{fg}^*\} = H_r^2 P_g + H_i^2 P_g - 2H_r \operatorname{real}\{P_{fg}\} + 2H_i \operatorname{imag}\{P_{fg}\}$$

with respect to H_r and H_i yields

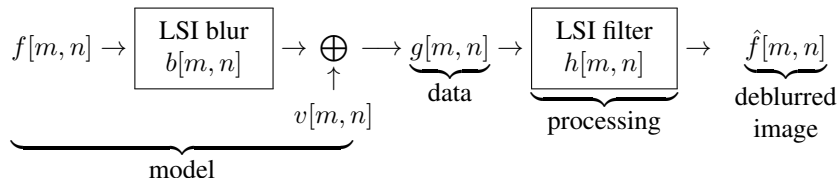
$$\begin{aligned} 0 &= 2H_r P_g - 2 \operatorname{real}\{P_{fg}\} \\ 0 &= 2H_i P_g - 2 \operatorname{imag}\{P_{fg}\}, \end{aligned}$$

so the MMSE filter matches (11.20):

$$H = H_r + \imath H_i = \frac{\operatorname{real}\{P_{fg}\}}{P_g} + \imath \frac{\operatorname{imag}\{P_{fg}\}}{P_g} = \frac{P_{fg}}{P_g}.$$

11.3 Wiener filter for image deblurring

Now we generalize to the case where the image is corrupted by both blur and noise.



The measurement model here is

$$g[m, n] = b[m, n] ** f[m, n] + v[m, n].$$

We want to design a filter $h[m, n]$ to apply to the noisy, blurry image $g[m, n]$ to form an estimate $\hat{f}[m, n]$ of the unknown image $f[m, n]$ that is optimal in the MMSE sense.

Under the same assumptions used for the denoising case, including that the image $f[m, n]$ and the noise $v[m, n]$ are uncorrelated, one can show (HW) that the ideal (noncausal) filter, called the **Wiener filter** has frequency response

$$H(\Omega_1, \Omega_2) = \frac{B^*(\Omega_1, \Omega_2) P_f(\Omega_1, \Omega_2)}{|B(\Omega_1, \Omega_2)|^2 P_f(\Omega_1, \Omega_2) + P_v(\Omega_1, \Omega_2)} = \frac{B^*(\Omega_1, \Omega_2)}{|B(\Omega_1, \Omega_2)|^2 + R(\Omega_1, \Omega_2)}, \quad R(\Omega_1, \Omega_2) \triangleq \frac{P_v(\Omega_1, \Omega_2)}{P_f(\Omega_1, \Omega_2)}. \quad (11.23)$$

The `deconvwnr` command implements this filtering using inputs $g[m, n]$, $b[m, n]$, $R_v[m, n]$, and $R_f[m, n]$.

In practice the power spectra of the noise and signal are usually not known, but often it is reasonable to assume that $P_v(\Omega_1, \Omega_2)$ is a constant (e.g., white noise) and that $P_f(\Omega_1, \Omega_2)$ is a decreasing function of frequency (like the fractal models shown later). Thus the “noise to signal power” ratio $R(\Omega_1, \Omega_2)$ typically is an increasing function of frequency. It is sometimes called a **regularizer** because if $R(\Omega_1, \Omega_2) = 0$ then $H(\Omega_1, \Omega_2) = 1/B(\Omega_1, \Omega_2)$ which is an inverse filter that is unstable (blows up) where $B(\Omega_1, \Omega_2)$ approaches zero. Using a (nonzero) regularizer $R(\Omega_1, \Omega_2)$ avoids this instability. Since $P_f(\Omega_1, \Omega_2)$ is usually unknown, in practice often we just pick $R(\Omega_1, \Omega_2)$ to be some increasing function of frequency and “see if it works,” as illustrated in the following example.

Example. The following figure shows a case where the blur $b[m, n]$ is an 11×11 Gaussian PSF and we chose the regularizer to be

$$R(\Omega_1, \Omega_2) = \beta |C(\Omega_1, \Omega_2)|^2, \quad c[m, n] = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \xleftrightarrow{\text{DSFT}} C(\Omega_1, \Omega_2),$$

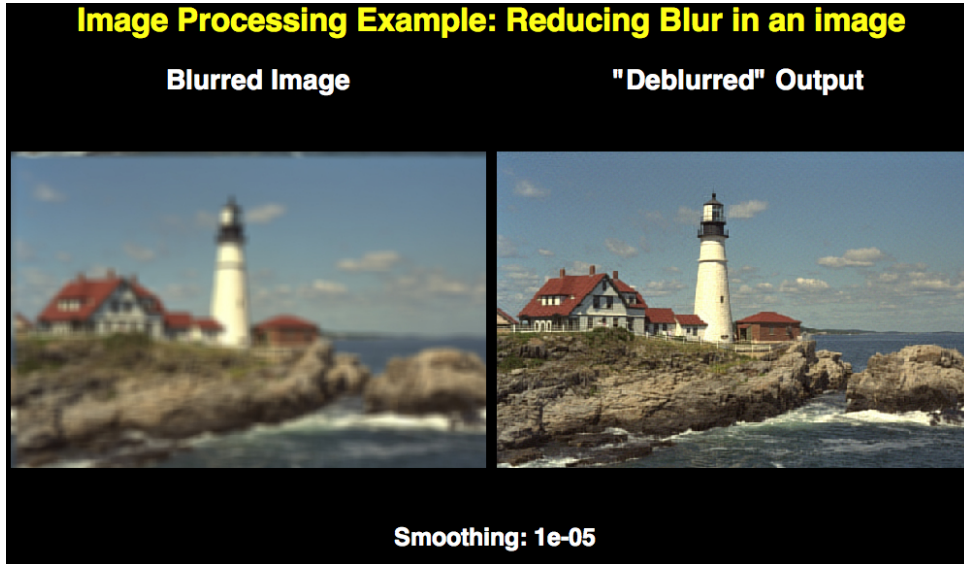
where $R(\Omega_1, \Omega_2) \approx \beta 9(\Omega_1^2 + \Omega_2^2)^2$ (suggesting the noise to signal power increases as the 4th power of the frequency) because

$$C(\Omega_1, \Omega_2) = 8 - 2 \cos(\Omega_1) - 2 \cos(\Omega_2) - 2 \cos(\Omega_1 + \Omega_2) - 2 \cos(\Omega_1 - \Omega_2) \approx 3\Omega_1^2 + 3\Omega_2^2.$$

Applying the Wiener (deblurring) filter (11.23) to each color channel yields the image on the right.

One must choose the **regularization parameter** β appropriately [4].

Related to deconvreg.



Demo created by Sathish Ramani. `src/matlab/contrib/ramani-demo-tech-day/deconvExample.m`

What boundary conditions were used to simulate this blurry image?

(in class)

Performance measures (MSE, NRMSE, SNR, etc.)

★★

To quantify both the degree of degradation as well as the improvements due to restoration algorithms, various performance measures are used in the literature. Here are some examples.

- The **mean square error (MSE)** between a restoration $\hat{f}[m, n]$ and an original $M \times N$ image $f[m, n]$ is often defined to be the per-pixel average squared error:

$$\text{MSE} = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \left| \hat{f}[m, n] - f[m, n] \right|^2. \quad (11.24)$$

Note that this is a *spatial* average, not an ensemble average (expectation).

- If we have a statistical model for the images, *i.e.*, we treat $f[m, n]$ as random process, then often we define **MSE** in terms ensemble averages:

$$\text{MSE} = \text{E} \left[\left| \hat{f}[m, n] - f[m, n] \right|^2 \right]. \quad (11.25)$$

Note that this MSE definition could be a function of pixel location m, n , but typically we use this MSE in the context of WSS random process models, in which case the MSE is not a function of m, n .

- A slight inconvenience of MSE is that its units are the square of the original image units. In photographic imaging, where the intensities typically are arbitrary units, this is a minor issue. But in applications like X-ray CT, where the pixel values have physically meaningful units, it is far preferable to use error measures that are expressed in those very units. One simple solution is to take the square root of MSE, known as the **root mean-squared error (RMSE)**:

$$\text{RMSE} = \sqrt{\text{MSE}}.$$

- In cases where the image units are arbitrary, it is preferable to report error measures that are invariant to the choice of units by normalizing. There are a multiple definitions of **normalized root mean square error (NRMSE)** in the literature, so when

reporting results it is wise to specify which version was used. A typical definition for a single realization is:

$$\text{NRMSE} = \frac{\text{RMSE}}{\sqrt{\frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} |f[m, n]|^2}},$$

and for cases with random process models:

$$\text{NRMSE} = \frac{\text{RMSE}}{\sqrt{\mathbb{E}[|f[m, n]|^2]}}.$$

- No matter which NRMSE definition is used, usually we define the **signal-to-noise ratio (SNR)** as its reciprocal, often in dB:

$$\text{SNR} = 20 \log_{10} \frac{1}{\text{NRMSE}}.$$

- The **peak SNR** or **PSNR** is also popular, usually in dB:

$$\text{PSNR} \triangleq 20 \log_{10} \frac{\max_{(m,n)} |f[m, n]|}{\text{RMSE}}. \quad (11.26)$$

- The **blur SNR** or **BSNR** is defined in terms of the noisy, blurry image $g[m, n]$ as follows:

$$\text{BSNR} \triangleq 10 \log_{10} \frac{\frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} |g[m, n] - \mu_g|^2}{\sigma^2},$$

where the mean value of the (noiseless) blurred image is $\mu_g = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \mathbb{E}[g[m, n]]$ and σ^2 denotes the noise variance. Adding a constant value to an image does not change the BSNR.

- All of the above are “squared error” performance measures, which are convenient for analysis. Worst-case errors and ℓ_1 errors can also be of interest:

$$\left\| \hat{f} - f \right\|_{\infty} = \max_{m,n} \left| \hat{f}[m,n] - f[m,n] \right|, \quad \frac{1}{MN} \left\| \hat{f} - f \right\|_1 = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \left| \hat{f}[m,n] - f[m,n] \right|.$$

Clearly there are a variety of possible performance measures.

Because there are so many variations on the definition of SNR and NRMSE, one should always define which one is used.

Although “squared error” and SNR-type performance measures are used frequently in image processing, increasing SNR does not necessarily mean that the image is “better” for a given purpose. Other approaches try to use perceptual metrics, *e.g.*, [5–8], or metrics that relate to how well a human observer can perform a task (such as detecting a tumor) from the image [9–11]. See also [12, 13].

A particular popular perceptual error metric is the **structural similarity image metric (SSIM)** [14, 15]. See also [16].

11.4 Spectral estimation [skip]



We have seen that the Wiener filter requires knowledge of the signal power spectrum $P_f(\Omega_1, \Omega_2)$ and the noise power spectrum $P_v(\Omega_1, \Omega_2)$, or equivalently the autocorrelation functions of the signal and of the noise.

Although sometimes we might be able to model the spectra (particularly the noise spectra) based on knowledge of how an imaging system works, often we must **estimate** these quantities from noisy measurements.

This raises the general question: given a realization (or several realizations) of a random process $x[m, n]$, how can we estimate its autocorrelation function $R_x[m, n]$ and/or its power spectrum $P_x(\Omega_1, \Omega_2)$?

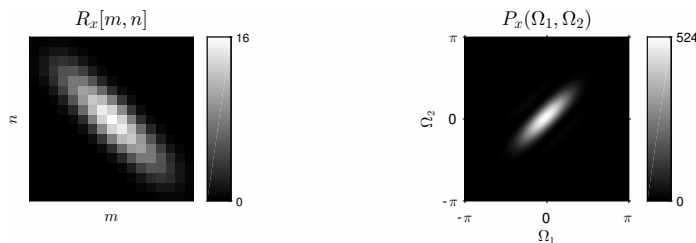
Periodogram

Given a (finite-extent portion of a) realization of a random process $x[m, n]$, the **periodogram** is defined as the normalized squared magnitude of its Fourier transform:

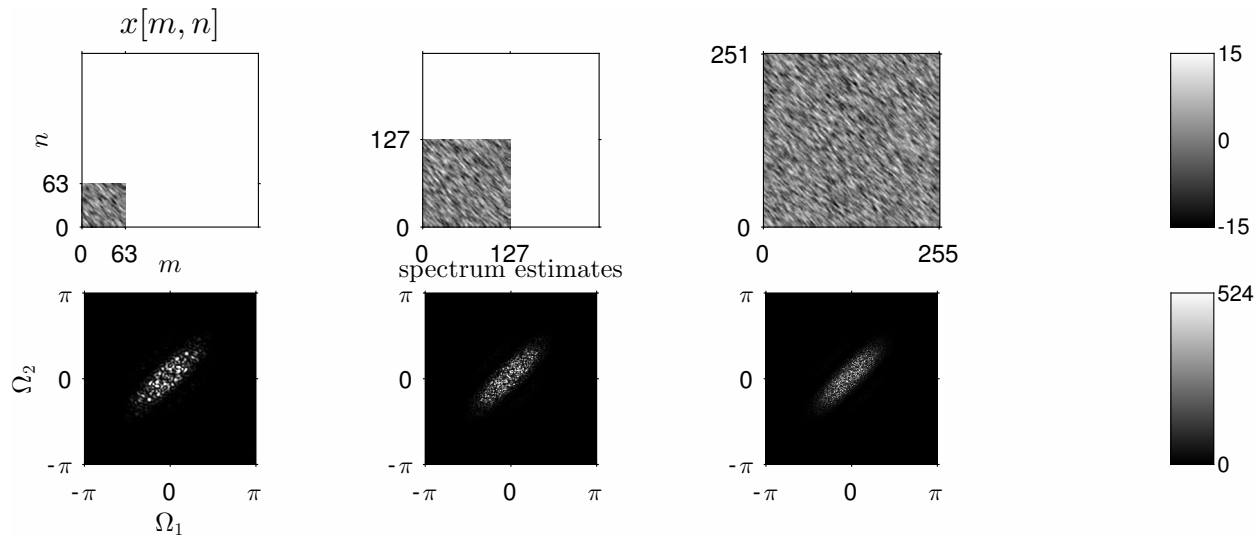
$$\hat{P}_x(\Omega_1, \Omega_2) \triangleq \frac{1}{MN} |X(\Omega_1, \Omega_2)|^2 = \frac{1}{MN} \left| \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m, n] e^{-i(\Omega_1 m + \Omega_2 n)} \right|^2. \quad (11.27)$$

This is fairly useless as an estimate of the power spectrum $P_x(\Omega_1, \Omega_2)$ in practice, because it is quite “noisy” (high variance).

Example. Consider the WSS random process $x[m, n]$ having the following autocorrelation function $R_x[m, n]$ and power spectral density $P_x(\Omega_1, \Omega_2)$.



The following figure shows periodograms for various finite-sized images. Even as the image size increases, the estimate does not improve, *i.e.*, the variance does not decrease, unlike a sample mean.



One might wonder why the strong law of large numbers principle does not apply here. The reason is that (11.27) is *not* in the form of a sample mean, due to the squaring, unlike (11.29) below.

Asymptotic unbiasedness of periodogram **(skim)**

Although the periodogram is excessively noisy, it does have one somewhat favorable property: it is an **asymptotically unbiased** estimate of the power spectral density [17, p. 411].

Given a WSS random process $x[m, n]$ on \mathbb{Z}^2 , we can always compute the DSFT of a $N \times N$ *truncated* portion of $x[m, n]$:

$$X_N(\Omega_1, \Omega_2) \triangleq \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} x[m, n] e^{-i(\Omega_1 m + \Omega_2 n)}.$$

The **periodogram** estimate of $P_x(\Omega_1, \Omega_2)$ is simply the (normalized) squared magnitude of this spectrum:

$$\hat{P}_N(\Omega_1, \Omega_2) \triangleq \frac{1}{N^2} |X_N(\Omega_1, \Omega_2)|^2.$$

The expected value is:

$$\begin{aligned} \mathbb{E} \left[\hat{P}_N(\Omega_1, \Omega_2) \right] &= \frac{1}{N^2} \mathbb{E} \left[|X_N(\Omega_1, \Omega_2)|^2 \right] = \frac{1}{N^2} \mathbb{E} \left[\sum_{m=0}^{N-1} \sum_{n=0}^{N-1} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} x[m, n] x^*[k, l] e^{-i[\Omega_1(m-k) + \Omega_2(n-l)]} \right] \\ &= \frac{1}{N^2} \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} \mathbb{R}_x[m-k, n-l] e^{-i[\Omega_1(m-k) + \Omega_2(n-l)]} \\ &= \frac{1}{N^2} \sum_{m=-(N-1)}^{N-1} \sum_{n=-(N-1)}^{N-1} (N-|m|)(N-|n|) \mathbb{R}_x[m, n] e^{-i(\Omega_1 m + \Omega_2 n)} \\ &= \sum_{m=-(N-1)}^{N-1} \sum_{n=-(N-1)}^{N-1} \left(1 - \frac{|m|}{N}\right) \left(1 - \frac{|n|}{N}\right) \mathbb{R}_x[m, n] e^{-i(\Omega_1 m + \Omega_2 n)}. \end{aligned}$$

This is not $P_x(\Omega_1, \Omega_2)$, so $\hat{P}_N(\Omega_1, \Omega_2)$ is **biased** for finite N . However, as $N \rightarrow \infty$,

$$\mathbb{E} \left[\hat{P}_N(\Omega_1, \Omega_2) \right] \rightarrow \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} R_x[m, n] e^{-i(\Omega_1 m + \Omega_2 n)} = P_x(\Omega_1, \Omega_2), \quad (11.28)$$

(for sufficiently well-behaved $R_x[m, n]$) so the periodogram is **asymptotically unbiased**.

Correlation-based methods

Let us consider working in the spatial domain instead, *i.e.*, with the autocorrelation function.

Recall that for a WSS random process, the autocorrelation function is the correlation between any pixel and another pixel that is $[m, n]$ away:

$$R_x[m, n] = \mathbb{E}[x[m, n] x^*[0, 0]] = \mathbb{E}[x[m + k, n + l] x^*[k, l]], \quad \forall k, l \in \mathbb{Z}.$$

Now the question becomes how do we compute this expectation given only a single realization of $x[m, n]$?

Recall the **strong law of large numbers**. For independent and identically distributed random variables Z_n with mean $\mathbb{E}[Z]$:

$$\lim_{N \rightarrow \infty} \frac{1}{N} \sum_{n=1}^N Z_n = \mathbb{E}[Z] \text{ w.p. } 1.$$

This fundamental result from probability gives a theoretical foundation for estimating unknown quantities by **averaging**. Because Z can be any random variable, we may also conclude

$$\lim_{N \rightarrow \infty} \frac{1}{N} \sum_{n=1}^N X_n Y_n^* = \mathbb{E}[XY^*] \text{ w.p. } 1,$$

if (X_n, Y_n) are independent and identically distributed pairs of random variables with correlation $\mathbb{E}[XY^*]$.

Given a $M \times N$ noisy image that (supposedly) is a realization of a WSS random process, it is natural to estimate the autocorrelation function for a given separation $[m, n]$ by averaging the correlations of *all pairs of pixels having that separation*:

$$\hat{R}_x[m, n] = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} x[m + k, n + l] x^*[k, l]. \quad (11.29)$$

What operation on the image $x[m, n]$ is this? [The \(deterministic\) autocorrelation function.](#)

What MATLAB command was used to make the earlier examples, e.g., Fig. 11.1 and Fig. 11.2? ?? (pair/class)

Performing this average for all possible $[m, n]$ separations yields autocorrelations estimates like those in Fig. 11.1 and Fig. 11.2.

Taking the FT or DFT of such “raw” autocorrelation estimates gives uselessly noisy results due to contributions of noise in tails. In fact, by the Wiener-Khinchin theorem, the DSFT of the empirical autocorrelation estimate is simply the periodogram again:

$$\hat{R}_x[m, n] \xleftrightarrow{\text{DSFT}} \hat{P}_x(\Omega_1, \Omega_2).$$

However, if we “know” that the true $R_x[m, n]$ has a small central core surrounded by zeros, we can safely remove the “noise” in the tails by multiplying by a **window** function $w[m, n]$:

$$\hat{R}_x[m, n] \triangleq \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} x[k+n, l+m] x^*[k, l] w[m, n] \xleftrightarrow{\text{DSFT}} \hat{P}_x(\Omega_1, \Omega_2) \triangleq \frac{1}{(2\pi)^2} \hat{P}_x(\Omega_1, \Omega_2) \boxed{*}_{2\pi} W(\Omega_1, \Omega_2).$$

Windowing the autocorrelation function in the spatial domain corresponds to [smoothing](#) in the spatial frequency domain.

An efficient implementation for a finite-sized image uses FFT operations:

$$x[m, n] \rightarrow \boxed{\text{FFT}} \xrightarrow{X[k, l]} \boxed{|\cdot|^2} \rightarrow \begin{matrix} \otimes \\ \uparrow \\ 1/MN \end{matrix} \xrightarrow{\hat{P}_x(\Omega_1, \Omega_2)} \boxed{\text{iFFT}} \xrightarrow{\hat{R}_x[m, n]} \begin{matrix} \otimes \\ \uparrow \\ w[m, n] \end{matrix} \xrightarrow{\hat{R}_x[m, n]} \boxed{\text{FFT}} \rightarrow \hat{P}_x(\Omega_1, \Omega_2)$$

Using spatial averaging to estimate the autocorrelation function $R_x[m, n]$ relies on an assumption that the random process $x[m, n]$ is **ergodic**, meaning that spatial averages converge to ensemble averages. (This is a subtle technical point.)

After such windowing/smoothing, we have (sampled) estimates of the power spectrum. Applying this technique to samples of both the noise and signal images (in the Wiener filter context) we can estimate the power spectra of each, and then form the Wiener filter from those estimates.

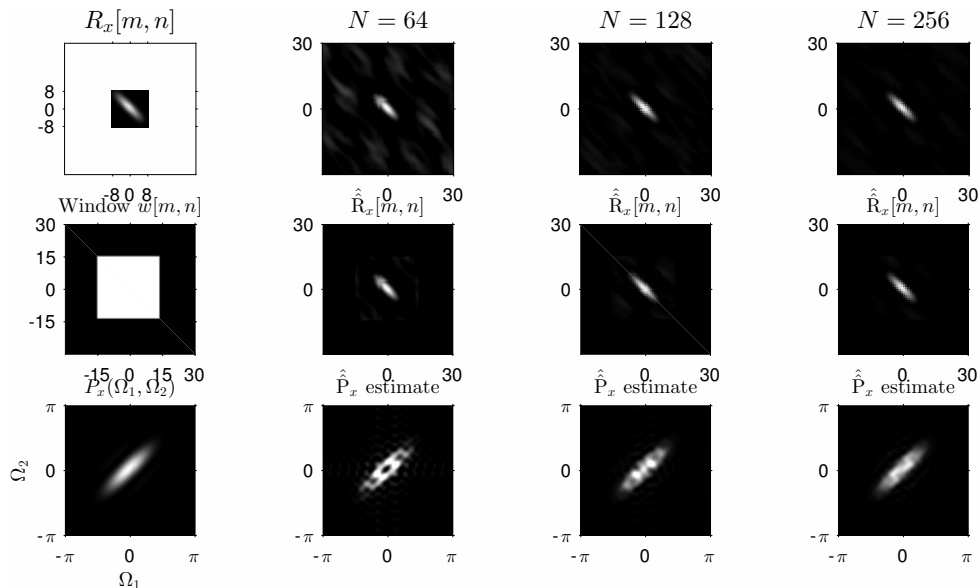
Unfortunately, this means we need to either know the signal power spectrum from some modeling principles, or we must have at least one signal-only image from which we can examine an autocorrelation function. But usually the Wiener filter is of interest in problems where we only have noisy data so there are no signal-only images available!

Example. Continuing the previous example, consider this figure.

The 1st row shows the true $R_x[m, n]$ and finite-sample estimates $\hat{R}_x[m, n]$ for three values of N .

The 2nd row shows the (29×29) rectangular window $w[m, n]$ and the windowed autocorrelation estimates $\hat{\tilde{R}}_x[m, n]$.

The 3rd row shows the true power spectrum $P_x(\Omega_1, \Omega_2)$ and the corresponding (windowed) power spectrum estimates $\hat{\tilde{P}}_x(\Omega_1, \Omega_2)$. Due to the window, the final power spectrum estimates are much less noisy than those shown previously.



Parametric spectral estimation [skip] ◆◆

Another common strategy for spectral estimation is to assume that the autocorrelation function has some **parametric** form. For example one might assume that it decays geometrically:

$$R_f[m, n] = \sigma_f^2 r^{\sqrt{m^2+n^2}} = \sigma_f^2 \exp\left((\log r)\sqrt{m^2+n^2}\right),$$

where $r \in (0, 1)$ characterizes the correlation properties.

Exercise. Determine whether the above expression is a legitimate autocorrelation function by determining the corresponding power spectrum $P_f(\Omega_1, \Omega_2)$ analytically.

One benefit of parametric methods is that one can estimate the parameters *even in the presence of noise*. Consider the usual additive white noise model:

$$g[m, n] = f[m, n] + v[m, n]$$

and assume that the signal $f[m, n]$ and noise $v[m, n]$ are uncorrelated. Then the autocorrelation function of the noisy measured image $g[m, n]$ is:

$$R_g[m, n] = \underbrace{R_f[m, n] + R_v[m, n]}_{\text{uncorrelated}} = \underbrace{\sigma_f^2 \rho^{\sqrt{n^2+m^2}}}_{\text{signal part}} + \underbrace{\sigma_v^2 \delta_2[m, n]}_{\text{noise part}}.$$

One can first estimate the autocorrelation function $R_g[m, n]$ empirically using the windowing technique described above. Then one can estimate the parameters of $R_f[m, n]$ by fitting the three parameters $(\sigma_v^2, \sigma_f^2, \rho)$ to it. This can be done by nonlinear least squares for example.

Fractal processes

★★

Another kind of parametric random process model that has been used for images is **fractals** or **self similar** random processes. Such distributions arise frequently [18]. The basic idea is most easily described for a 1D continuous-space random process.

We say a WSS random process $g(x)$ is **fractal** or **self similar** if its power spectral density satisfies (for some power p):

$$P_g(\nu) = c \frac{1}{|\nu|^p}. \quad (11.30)$$

This is called a **power law** spectrum because it decreases as frequency raised to some power.

To understand why such processes are called **self similar**, suppose we define a new random process

$$w(x) \triangleq g(\alpha x)$$

for some spatial scaling factor α . Then the autocorrelation function of $w(x)$ is

$$R_w(x_1, x_2) = \mathbb{E}[w(x_1)w(x_2)] = \mathbb{E}[g(\alpha x_1)g(\alpha x_2)] = R_g(\alpha(x_1 - x_2)).$$

Is the new, scaled random process WSS? ??

[RQ]

What is the power spectral density of $w(x)$?

(class)

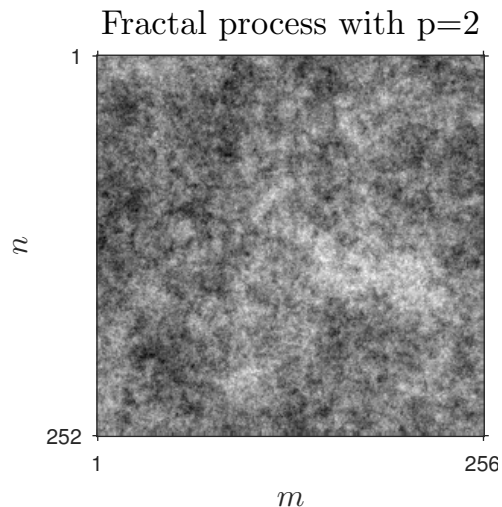
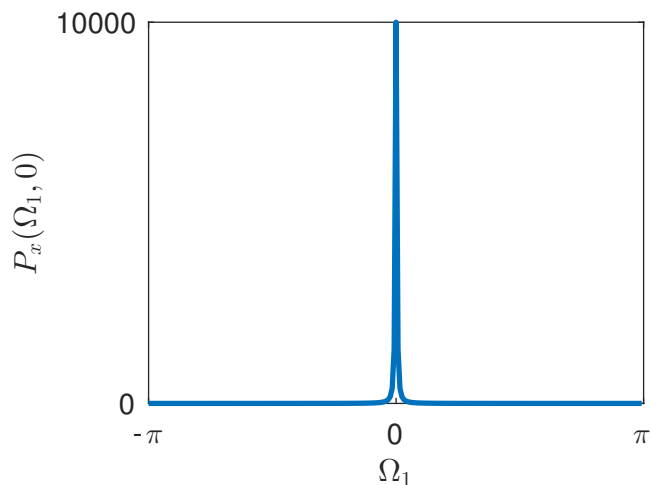
$$R_w(x) \xleftrightarrow{\mathcal{F}} P_w(\nu) = \text{??}$$

Because this is a constant times $1/|\nu|^p$, we see that $w(x)$ has the same (second order) statistical properties as $g(x)$. In other words, magnifying or minifying $g(x)$ by any factor α leads to a random process with the *same* power spectral density (to within a constant). This self similarity property is unique to random processes having power spectra of the form (11.30).

The power spectral density (11.30) is not band limited, so there is no exact analogy of this property in discrete space. But some kinds of digital images have a similar behavior, *e.g.*,

$$P_x(\Omega_1, \Omega_2) = \left(c_0 \frac{1}{c_1 + \left(\sqrt{\Omega_1^2 + \Omega_2^2} \right)^p} \right)_{(2\pi, 2\pi)} .$$

Example. Here is the case $p = 2$ and $c_1 = 0.0001$. Note that this image has somewhat more “natural” looking randomness than pure white noise, *i.e.*, it corresponds somewhat better to textured parts of scenes you might see in real life. What do you see?



11.5 Patch-based denoising methods

★★

WSS random process models for *noise* are often reasonable. WSS models for the signals of interest are now considered “classical” (*i.e.*, old-fashioned). They have the benefit of often leading to simple linear estimation methods, but usually are suboptimal because they describe only correlations and not higher-order relationships between pixels in real-world images.

Many modern alternative methods have been developed. Some of these try to “learn” the image properties from training data or from the noisy image itself. Often these methods use small (*e.g.*, 8×8) **patches** of the image for statistical analysis.

Non-local means

One particularly famous patch-based method is called **non-local means (NLM)** [19].

Given a $M \times N$ noisy image $g[m, n] = f[m, n] + v[m, n]$, where $v[m, n]$ denotes additive noise (not necessarily Gaussian noise), the NLM method estimates $f[m, n]$ by a weighted combination of image values:

$$\hat{f}[m, n] = \sum_{(k,l) \in \Omega_{[m,n]}} w[m, n; k, l; g] g[k, l],$$

where $\Omega_{[m,n]} \subseteq \mathcal{D}_{M,N}$ denotes a **search region** centered at $[m, n]$, possibly the entire image.

A typical search region is a $S \times S$ square centered at $[m, n]$, namely: $\Omega_{[m,n]} = \{[k, l] \in \mathcal{D}_{M,N} : |k - m| \leq S, |l - n| \leq S\}$.

The weights $w[m, n; k, l; g] \in [0, 1]$ determine how much input pixel value $g[k, l]$ contributes to estimated pixel value $\hat{f}[m, n]$. We design these weights with two criteria in mind:

- If the $P \times P$ patch centered at $[k, l]$ is *similar* to the $P \times P$ patch centered at $[m, n]$, then use a larger weight value.
- The weights should sum to unity: $\sum_{k=0}^{M-1} \sum_{l=0}^{N-1} w[m, n; k, l; g] = 1$.

Why? ??

[RQ]

A typical choice is

$$w[m, n; k, l; g] \triangleq \frac{h(\|\mathbf{R}_{m,n}g - \mathbf{R}_{k,l}g\|)}{\sum_{[k',l'] \in \Omega_{[m,n]}} h(\|\mathbf{R}_{m,n}g - \mathbf{R}_{k',l'}g\|)}, \quad h(t) = \exp\left(-\frac{t^2}{2P^2\sigma_h^2}\right),$$

where $\mathbf{R}_{m,n}$ is the $P^2 \times MN$ matrix that extracts the $P \times P$ patch of $g[m, n]$ centered at $[m, n]$, and σ_h is a tuning parameter.

The classical moving average takes the mean of $g[m, n]$ over a $P \times P$ neighborhood, implicitly assuming that neighboring values are similar (except for noise). This is a poor assumption near edges in an image. NLM finds similar patches anywhere in the image (or search region) and averages them. If patch $\mathbf{R}_{m,n}g$ and patch $\mathbf{R}_{k,l}g$ are similar, then their RMS difference $\frac{1}{P} \|\mathbf{R}_{m,n}g - \mathbf{R}_{k,l}g\|$ will be small, hence $h(\cdot)$ will be close to 1 so the corresponding weight w will be large.

Example. Figures from [19]. For each of the 6 pairs, the left is a small image $g[m, n]$ and the right shows the weights $w[0, 0; k, l; g]$ for the center pixel (shown in white).

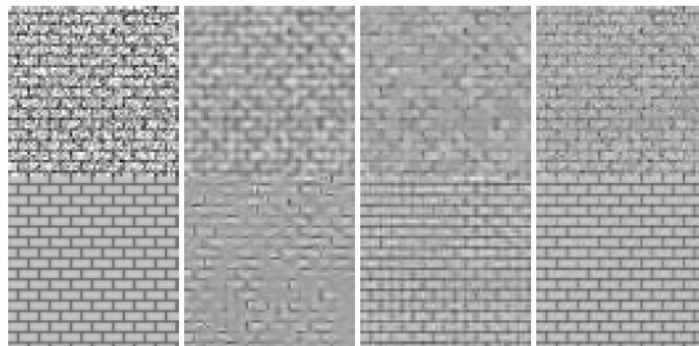
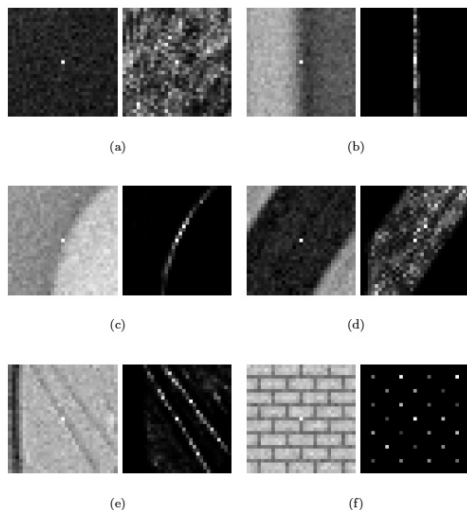


FIG. 18. Denoising experience on a periodic image. From left to right and from top to bottom: noisy image (standard deviation 35), Gauss filtering, total variation, neighborhood filter, Wiener filter (ideal filter), hard TIWT, DCT empirical Wiener filtering, and the NL-means algorithm.

- Benefit: better resolution-noise trade-off than local filters.
- Benefit: residual $g[m, n] - \hat{f}[m, n]$ looks more like white noise
- Drawback: more computation than local filters. Computing all of w would require $O(M^2N^2)$ operations. In practice, often only a local search in a $L \times L$ neighborhood is used, which is $O(MNL^2)$, where $L \ll \min(M, N)$.

Another important patch-based method that uses sparse representations is the **K-SVD** approach [20]. Whereas K-SVD requires optimization algorithms, the higher-order SVD (HOSVD) approach [21] is non-iterative and works quite well for denoising. For a modern survey of image denoising methods, see [22]. See also [23] [24–26].

11.6 Markov random field models [skip]

◆◆

If one wants to move beyond second-order statistical models for images, there are innumerable possible models. Some type of simplification is needed to get anywhere. One popular family of models for images is **Markov random field (MRF)** models. These models are multidimensional generalizations of **Markov chains**. They are popular for **image segmentation** applications [27–30]. MRF models have also been applied for many image restoration and image reconstruction problems [31–33].

Markov chains

We begin by considering 1D random process $\mathbf{x} = (x_1, \dots, x_N)$. The probability distribution for this process can always be written:

$$p(\mathbf{x}) = p(x_1, \dots, x_N) = p(x_N | x_{N-1}, \dots, x_1) p(x_{N-1} | x_{N-2}, \dots, x_1) \cdots p(x_2 | x_1) p(x_1). \quad (11.31)$$

This factorization is called the **chain rule for probability**.

This form is often too general to be practical, so usually we make simplifying assumptions. For example, for **Markov chain** models we assume $p(x_n | x_{n-1}, \dots, x_1)$ depends on only a few of the previous values. For a **first-order Markov chain** we assume

$$p(x_n | x_{n-1}, \dots, x_1) = p(x_n | x_{n-1}). \quad (11.32)$$

Under this assumption, the overall distribution (11.31) simplifies to

$$p(\mathbf{x}) = p(x_N | x_{N-1}) p(x_{N-1} | x_{N-2}) \cdots p(x_2 | x_1) p(x_1). \quad (11.33)$$

This model is much more manageable because we need only to model the **transition distributions** $p(x_n | x_{n-1})$.

Example. We could choose a conditionally Gaussian model where $p(x_1) = \mathbf{N}(0, \sigma_0^2)$ and $p(x_n | x_{n-1}) = \mathbf{N}(x_{n-1}, \sigma_1^2)$. This is called a **random walk** model because $x_n = x_{n-1} + z_j$ where $\{z_j\}$ is an IID Gaussian random process with $z_j \sim \mathbf{N}(0, \sigma_1^2)$. To write down $p(\mathbf{x})$ in (11.33), we need the formulas for all those conditional distributions. Writing “ $p(x_n | x_{n-1}) = \mathbf{N}(x_{n-1}, \sigma_1^2)$ ” is equivalent to

$$p(x_n | x_{n-1}) = \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{1}{2\sigma_1^2}(x_n - x_{n-1})^2\right), \text{ i.e., } -\log p(x_n | x_{n-1}) = \frac{1}{2} \log(2\pi\sigma_1^2) + \frac{1}{2\sigma_1^2}(x_n - x_{n-1})^2. \quad (11.34)$$

Substituting into (11.33), for this model, the overall (negative log) distribution is

$$-\log p(\mathbf{x}) = c_0 + \frac{x_1^2}{2\sigma_0^2} + \sum_{n=2}^N \frac{(x_n - x_{n-1})^2}{2\sigma_1^2}, \quad c_0 = \frac{1}{2} \log((2\pi\sigma_0^2)(2\pi\sigma_1^2)^{N-1}). \quad (11.35)$$

Sequences $\{x_n\}$ for which $-\log p(\mathbf{x})$ is small, or equivalently for which $p(\mathbf{x})$ is large, are the “most likely” according to this first-order Gaussian random walk model.

What signal \mathbf{x} is the most likely in this example? ??

(pair/class)

What signals \mathbf{x} are “very likely” if σ_0^2 is large? ??

Properties of first-order Markov chain **(skim)**

Under the general first-order Markov chain model (11.32), one can show that

$$\mathbf{p}(x_n | x_1, \dots, x_{n-1}, x_{n+1}, \dots, x_N) = \frac{\mathbf{p}(x_{n+1} | x_n) \mathbf{p}(x_n | x_{n-1})}{\int \mathbf{p}(x_{n+1} | x'_n) \mathbf{p}(x'_n | x_{n-1}) dx'_n}. \quad (11.36)$$

In other words, the distribution of x_n given *all* other values in the process depends *only* on the two neighboring values x_{n+1} and x_{n-1} . This is the key concept that is generalized by Markov random field models.

Derivation:

$$\begin{aligned} \mathbf{p}(x_n | x_1, \dots, x_{n-1}, x_{n+1}, \dots, x_N) &= \frac{\mathbf{p}(x_1, \dots, x_{n-1}, x_n, x_{n+1}, \dots, x_N)}{\mathbf{p}(x_1, \dots, x_{n-1}, x_{n+1}, \dots, x_N)} = \frac{\mathbf{p}(x_1, \dots, x_{n-1}, x_n, x_{n+1}, \dots, x_N)}{\int \mathbf{p}(x_1, \dots, x_{n-1}, x'_n, x_{n+1}, \dots, x_N) dx'_n} \\ &= \frac{\mathbf{p}(x_N | x_{N-1}) \cdots \mathbf{p}(x_{n+1} | x_n) \mathbf{p}(x_n | x_{n-1}) \cdots \mathbf{p}(x_2 | x_1) \mathbf{p}(x_1)}{\int \mathbf{p}(x_N | x_{N-1}) \cdots \mathbf{p}(x_{n+1} | x'_n) \mathbf{p}(x'_n | x_{n-1}) \cdots \mathbf{p}(x_2 | x_1) \mathbf{p}(x_1) dx'_n} = \cdots \end{aligned}$$

Example. Continuing the previous Gaussian random walk example, after considerable simplification one can show

$$\mathbf{p}(x_n | x_1, \dots, x_{n-1}, x_{n+1}, \dots, x_N) = \frac{\mathbf{p}(x_{n+1} | x_n) \mathbf{p}(x_n | x_{n-1})}{\int \mathbf{p}(x_{n+1} | x'_n) \mathbf{p}(x'_n | x_{n-1}) dx'_n} = \frac{\frac{1}{2\pi\sigma_1^2} e^{-(x_n - \bar{x}_n)^2 / \sigma_1^2}}{\frac{1}{2\pi\sigma_1^2} \int e^{-(x_n - \bar{x}_n)^2 / \sigma_1^2} dx'_n} = \mathbf{N}\left(\bar{x}_n, \frac{\sigma_1^2}{2}\right)$$

where $\bar{x}_n \triangleq (x_{n+1} + x_{n-1})/2$ denotes the average of the left and right neighbors of x_n .

Markov random fields

Typically **Markov chain** models are described using causal transition probabilities like (11.32), which is often natural for 1D temporal random processes. But for random processes in 2D (and higher), we often prefer models that are isotropic and hence non-causal. **Markov random fields** are suitable generalizations.

Let $\mathbf{x} = \{x_s : s \in \mathcal{S}\}$ denote a **random field**, for some countable index set \mathcal{S} called **sites**. For simplicity one can think of $\mathcal{S} = \mathbb{N}$ in 1D, so $\mathbf{x} = (x_1, x_2, \dots)$, but the idea of a more general index set is to include multi-dimensional cases, e.g., for 2D images $\mathcal{S} = \mathbb{Z}^2$ or $\mathcal{S} = \{1, \dots, M\} \times \{1, \dots, N\}$.

In a MRF, the sites in \mathcal{S} are related to one another statistically via a (possibly multi-dimensional) **neighborhood system**:

$$\mathcal{N} = \{\mathcal{N}_s : s \in \mathcal{S}\},$$

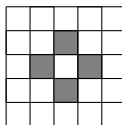
where \mathcal{N}_s is the set of sites neighboring site s , and these neighborhoods must satisfy


- $s \notin \mathcal{N}_s, \forall s \in \mathcal{S}$ (a site is not in its own neighborhood) and
- $r \in \mathcal{N}_s \implies s \in \mathcal{N}_r \quad \forall r, s \in \mathcal{S}$ (the notion of a neighbor is symmetric).

Example. In 1D, where $\mathcal{S} = \{1, \dots, N\}$ we could choose the neighborhood of each pixel to be its left and right neighbors: $\mathcal{N}_s = \{s-1, s+1\}$ for $s = 2, \dots, N-1$. For the end conditions we could choose $\mathcal{N}_1 = \{2\}$ and $\mathcal{N}_N = \{N-1\}$, or we could use periodic boundary conditions: $\mathcal{N}_1 = \{N, 2\}$ and $\mathcal{N}_N = \{N-1, 1\}$.

Example. In 2D with $\mathcal{S} = \mathbb{Z}^2$, a site is indexed by two integers, i.e., $s = (m, n)$ for $m, n \in \mathbb{Z}$. A **first-order 2D neighborhood** for $s = (m, n)$ is $\mathcal{N}_s = \{(m-1, n), (m+1, n), (m, n-1), (m, n+1)\}$.

1st-order (4-point) 2D neighborhood



 neighbors of center pixel

Notation:

- x_s denotes the random field value at a single site s ,
- $\mathbf{x} = \mathbf{x}_{\mathcal{S}}$ denotes the entire random field, and
- $\mathbf{x}_{\mathcal{N}_s}$ denotes the vector of length $|\mathcal{N}_s|$ consisting of the elements $\{x_t : t \in \mathcal{N}_s\}$.

Definition. We say $\mathbf{x} \in \mathcal{X}$ is a **MRF** on \mathcal{S} with respect to a neighborhood system \mathcal{N} if and only if

$$\boxed{\mathbf{p}(x_s | \mathbf{x}_{\mathcal{S}-\{s\}}) = \mathbf{p}(x_s | \mathbf{x}_{\mathcal{N}_s}), \quad \forall \mathbf{x} \in \mathcal{X}, \quad \forall s \in \mathcal{S}.}$$
 (11.37)

In other words, the statistical properties of a given site (*e.g.*, pixel) given all the other sites (the rest of the image) depend only on the values of the neighboring sites (pixels). This definition generalizes (11.32). This is the key “simplification” in a MRF model! (Throughout, $\mathbf{p}(\cdot)$ is either a pdf or a PMF depending on whether \mathbf{x} is continuous-valued or discrete-valued.)

According to the **Hammersley-Clifford theorem** [34], a MRF for which $\mathbf{p}(\mathbf{x}) > 0, \forall \mathbf{x} \in \mathcal{X}$, can equivalently be characterized by a **Gibbs distribution**, *i.e.*,

$$\mathbf{p}(\mathbf{x}) = \frac{1}{Z} e^{-U(\mathbf{x})}, \quad \text{where } Z \triangleq \sum_{\mathbf{x} \in \mathcal{X}} e^{-U(\mathbf{x})}$$
 (11.38)

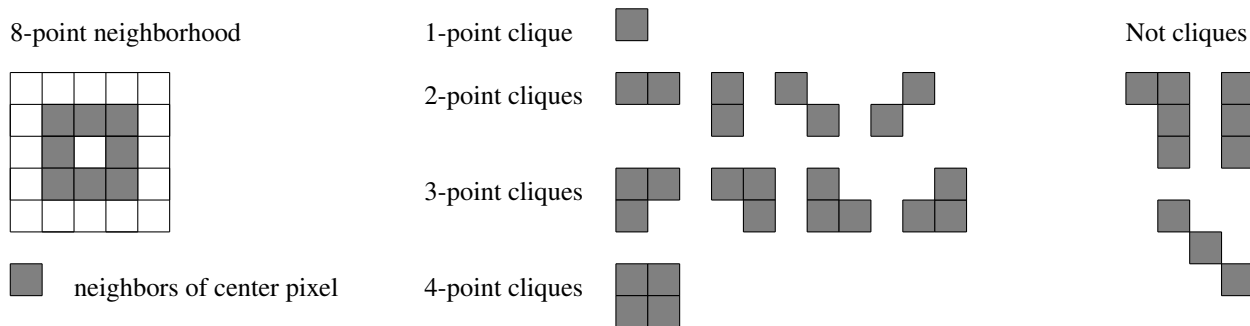
is a normalizing constant called the **partition function**, and $U(\mathbf{x})$ is an **energy function** of the form

$$U(\mathbf{x}) = \sum_{c \in \mathcal{C}} V_c(\mathbf{x}),$$

which is a sum of **clique potentials** $V_c(\mathbf{x})$ over all possible cliques. A **clique** c is defined as follows. It is either a single site $s \in \mathcal{S}$, or it is a subset of sites in \mathcal{S} in which every pair of distinct sites are neighbors:

$$s, r \in c \ \& \ r \neq s \implies r \in \mathcal{N}_s.$$

The value of $V_c(\mathbf{x})$ depends on the local configuration on clique c . For more details on MRF and Gibbs distributions, see [27].



The practical importance of the form (11.38) is that the negative log of the distribution of a MRF has the following form

$$-\log p(\mathbf{x}) = c_0 + \sum_{c \in \mathcal{C}} V_c(\mathbf{x}),$$

where $c_0 = \log Z$ is a constant independent of \mathbf{x} . So to define a MRF, we just need to select a neighborhood system and corresponding cliques and then choose the clique potential functions V_c .

Example. (Illustrates that 1st-order Markov chains are special cases of MRFs.)

Consider $\mathcal{S} = \mathbb{Z}$, $\mathcal{N}_j = \{j-1, j+1\}$, $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$, $\mathcal{C}_1 = \{j : j \in \mathbb{Z}\}$, $\mathcal{C}_2 = \{(j, j-1) : j \in \mathbb{Z}\}$ and the choice

$$V_c(\mathbf{x}) = \begin{cases} \beta_1 \frac{1}{2} |x_j|^2, & c = \{j\} \in \mathcal{C}_1 \\ \beta_2 \frac{1}{2} |x_j - x_{j-1}|^2, & c = \{(j, j-1)\} \in \mathcal{C}_2, \end{cases}$$

for which (cf. the random walk case (11.35))

$$-\log p(\mathbf{x}) = \log Z + \sum_{j=-\infty}^{\infty} \beta_1 \frac{1}{2} |x_j|^2 + \sum_{j=-\infty}^{\infty} \beta_2 \frac{1}{2} |x_j - x_{j-1}|^2.$$

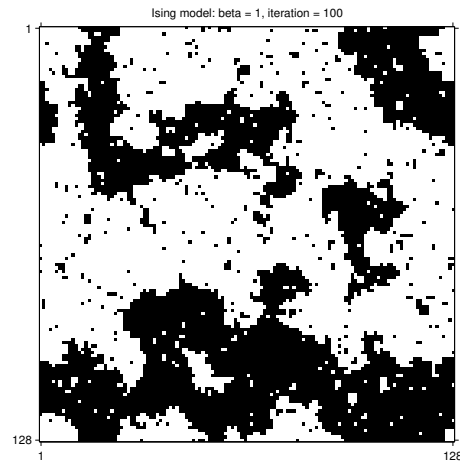
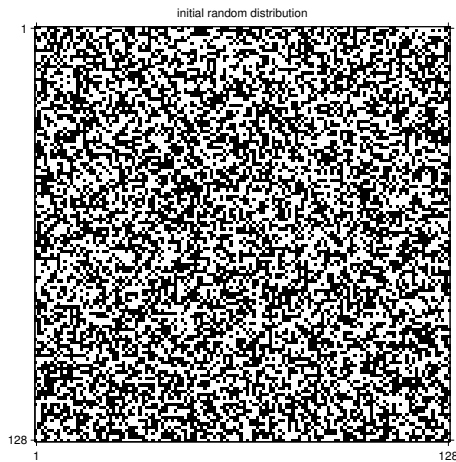
One can verify that this joint distribution is invariant to time shifts, *i.e.*, if $z_j = x_{j+M}$ for any $M \in \mathbb{Z}$, then $p(\mathbf{z}) = p(\mathbf{x})$. Thus this particular random process is strict sense stationary (SSS), and hence also WSS. Thus some MRF models (defined on \mathbb{Z}^d) are WSS random processes. Most of the MRF models used in image processing are shift invariant (*i.e.*, SSS) up to boundary conditions.

Generating sample realizations from a Gibbs distribution is challenging in general due to the non-causal structure. Statistical methods for doing this include the **Metropolis sampler** and **Gibbs sampler** [27].

Example. Consider the **Ising model**, which has horizontal cliques and vertical cliques and counts number of differing neighbors:

$$U(\mathbf{x}) = \beta \sum_n \sum_m \mathbb{I}_{\{x[m,n] \neq x[m-1,n]\}} + \mathbb{I}_{\{x[m,n] \neq x[m,n-1]\}} = \beta \cdot \text{boundary_length}. \quad (11.39)$$

Applying the **Metropolis sampler** to the MRF model $p(\mathbf{x}) = \frac{1}{Z} e^{-U(\mathbf{x})}$ yields realizations such as the one on the right below.



Demo with `ising1.m` (The left image is an IID Bernoulli image; right image used periodic boundary conditions.)

What (binary) image(s) are the most likely under the Ising model? ??

[RQ]

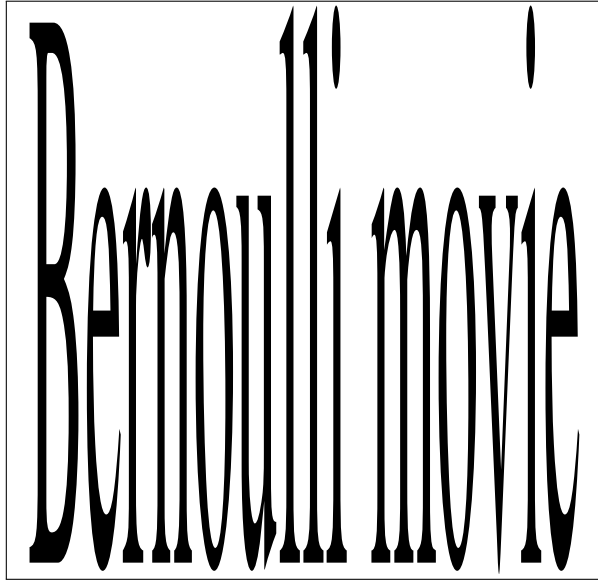
What are the least likely? ??

pair/class

When we draw samples from a distribution, we usually get “representative” or “reasonably likely” outcomes, not the “most likely” outcome; *cf.* a bell curve.

When designing a statistical model $p(\mathbf{x})$ for images in some application, we prefer models where the “reasonably likely” images have characteristics similar to the actual images in that application. In most image segmentation applications, the right image above, where $p(\mathbf{x})$ is the Ising model, is more representative than the random Bernoulli model seen the left image where $U(\mathbf{x}) = 1$.

Example.



Bernoulli movie



MRE movie

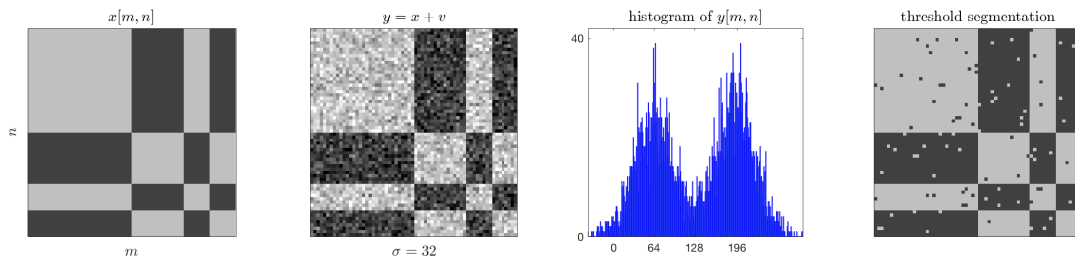
11.7 Image segmentation / classification [skip]



Now we use MRF models for the application of **image segmentation**.

Consider a simple signal plus noise model $\mathbf{y} = \mathbf{x} + \mathbf{v}$ where \mathbf{v} denotes additive zero-mean white Gaussian noise.

Example.



Suppose we know that each pixel in $x[m, n]$ belongs to one of $K \in \mathbb{N}$ classes. In other words:

$$x[m, n] = \mu_{c[m, n]} \in \mathcal{U} \triangleq \{\mu_1, \dots, \mu_K\}$$

where μ_1, \dots, μ_K denote the **class means** and $c[m, n] \in \{1, \dots, K\}$ denotes the **class index** or **class label** of pixel $[m, n]$. The image $c[m, n]$ is also called the **label image**.

Goal. The **image segmentation** or **classification** problem is to determine the class label $c[m, n]$ for each pixel, or equivalently to estimate $\mathbf{x} \in \mathcal{X} \triangleq \mathcal{U}^{MN}$ from the noisy image \mathbf{y} . In words, \mathcal{X} denotes the set of $M \times N$ images where each pixel is one of the K class means. In some cases we know the class means $\{\mu_k\}$ in advance; in other cases we must also determine them. Here we assume they are known.

Example. For a binary image, $K = 2$ so $c[m, n] \in \{1, 2\}$ and each $x[m, n]$ is either μ_1 or μ_2 . (See figure above.)

Maximum likelihood image segmentation

The **maximum likelihood (ML)** method finds the label image for which the observed image \mathbf{y} has the highest log-likelihood:

$$\hat{\mathbf{c}} = \arg \max_{\mathbf{c} \in \{1, \dots, K\}^{MN}} \log p(\mathbf{y} | \mathbf{c}), \quad \text{or equivalently:} \quad \hat{\mathbf{x}} = \arg \max_{\mathbf{x} \in \mathcal{X}} \log p(\mathbf{y} | \mathbf{x}).$$

When the noise is independent from pixel to pixel (e.g., for white Gaussian noise) the log-likelihood is a summation:

$$\log p(\mathbf{y} | \mathbf{x}) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \log p(y[m, n] | x[m, n]).$$

So the ML approach simplifies to finding the best class for each pixel independently:

$$\hat{x}[m, n] = \arg \max_{\mu \in \mathcal{U}} p(y[m, n] | x[m, n] = \mu). \quad (11.40)$$

For Gaussian noise with variance σ^2 , the likelihood is $p(y[m, n] | x[m, n] = \mu) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y[m, n] - \mu)^2}$, so the maximization (11.40) reduces to simply choosing the nearest class mean to each data pixel:

$$\hat{x}[m, n] = \arg \min_{\mu \in \mathcal{U}} |y[m, n] - \mu|^2.$$

And if there are just two classes ($K = 2$), with $\mu_1 < \mu_2$, then a simple **threshold** midway between the two class means gives the ML segmentation:

$$\hat{x}[m, n] = \begin{cases} \mu_1, & y[m, n] < \frac{\mu_1 + \mu_2}{2} \\ \mu_2, & y[m, n] \geq \frac{\mu_1 + \mu_2}{2}. \end{cases}$$

Example. The preceding figure illustrates that a simple threshold makes many classification errors, even for a very simple image. The ML method ignores our prior knowledge (assumption?) that in most applications, neighboring pixels often have the same class.

One could do some post-processing using **median filters** or binary **morphological** image processing to try to “clean up” classification errors. These operations use groups of pixels, and it would be more elegant to formulate the image segmentation problem using such groups in the first place, rather than as a way to “fix up” errors.

Using MRF models for image segmentation

MRF models are a way to capture prior statistical assumptions about neighboring pixels in applications like image segmentation. Given a prior model $p(\mathbf{x})$, there are several possible ways to estimate \mathbf{x} (or equivalently the label image \mathbf{c}) from a noisy image \mathbf{y} . The method that is optimal from a mean-squared error perspective is the **conditional mean** estimator:

$$\hat{\mathbf{x}} = E[\mathbf{x} | \mathbf{y}] = \int \mathbf{x} p(\mathbf{x} | \mathbf{y}) d\mathbf{x} .$$

However, mean-squared error is probably a suboptimal performance metric for classification problems. Indeed it is unlikely that this $\hat{\mathbf{x}} \in \mathcal{X}$. Furthermore, it is often extremely difficult to compute for MRF models anyway!

Therefore, more often one uses **maximum a posteriori (MAP)** methods that seek the **mode** of the posterior distribution:

$$\hat{\mathbf{x}} = \arg \max_{\mathbf{x} \in \mathcal{X}} \underbrace{p(\mathbf{x} | \mathbf{y})}_{\text{posterior}} = \arg \max_{\mathbf{x} \in \mathcal{X}} \frac{p(\mathbf{y} | \mathbf{x}) p(\mathbf{x})}{\underbrace{p(\mathbf{y})}_{\text{Bayes rule}}} = \arg \max_{\mathbf{x} \in \mathcal{X}} \left[\underbrace{\log p(\mathbf{y} | \mathbf{x})}_{\text{likelihood}} + \underbrace{\log p(\mathbf{x})}_{\text{prior}} \right].$$

Example. The simplest image prior would be to assume each pixel is **independent and identically distributed (IID)** and distributed uniformly over all K classes, *i.e.*, $p(x[m, n] = \mu_k) = 1/K$ and $p(\mathbf{x}) = \prod_{m,n} p(x[m, n]) = (1/K)^{MN}$.

What do sample images \mathbf{x} drawn from this distribution look like for $K = 2$? ??

[RQ]

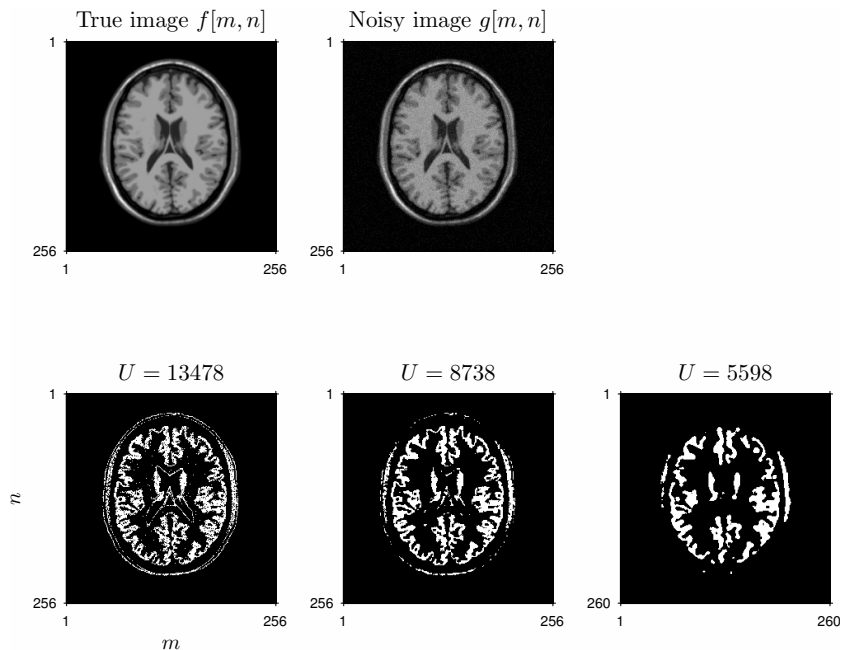
In this case, $\log p(\mathbf{x})$ does not depend on \mathbf{x} (for any \mathbf{x} in the sample space \mathcal{X}) so the MAP method reduces to the ML method.

Example. For an image segmentation problem, we might choose a first-order neighborhood the cliques corresponding to each horizontal and vertical pair of pixels, using periodic boundary conditions:

$$-\log \mathbf{p}(\mathbf{x}) = c_0 + U(\mathbf{x}), \quad U(\mathbf{x}) = \beta \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \mathbb{I}_{\{x[m,n] \neq x[m-1 \bmod M, n]\}} + \mathbb{I}_{\{x[m,n] \neq x[m, n-1 \bmod N]\}}. \quad (11.41)$$

In words, this log-prior counts how many neighboring pixels have different values. The more such discrepancies, the lower the prior probability. What images \mathbf{x} have the highest prior probability under this model? ?? [RQ]

Example. This figure illustrates how the Ising model might be used to help segment gray matter from a noisy MR image by defining a prior probability for different class label configurations.



Computing MAP estimates

Unfortunately, finding the MAP estimate also can be computationally very challenging, requiring stochastic methods to ensure a global maximizer is reached [27] [28].

One reasonably practical algorithm for this problem is called **iterated conditional modes (ICM)** [28]. The ICM method is an algorithm that iteratively maximizes the probability of each class label conditioned on all the others. We start with an initial guess $\mathbf{x}_s^{(0)}$, and then update as follows:

$$x_s^{(i+1)} = \arg \max_{x_s \in \mathcal{U}} p(x_s | \mathbf{y}, \mathbf{x}_{\mathcal{S}-\{s\}}^{(i)}), \quad i = 0, 1, 2, \dots,$$

where $\mathbf{x}_{\mathcal{S}-\{s\}}^{(i)}$ denotes the values of all other pixels in the image from the previous iteration. Using Bayes rule and the MRF properties one can show [28] that

$$x_s^{(i+1)} = \arg \max_{x_s \in \mathcal{U}} p(y_s | x_s) p(x_s | \mathbf{x}_{\mathcal{N}_s}^{(i)}) = \arg \max_{x_s \in \mathcal{U}} \left[\log(p(y_s | x_s)) + \log(p(x_s | \mathbf{x}_{\mathcal{N}_s}^{(i)})) \right], \quad (11.42)$$

where $\mathbf{x}_{\mathcal{N}_s}^{(i)}$ denotes the values of the *neighbors* from the preceding iteration. The ICM iteration (11.42) is quite simple and usually converges quickly, presumably to a local maximizer of the posterior $p(\mathbf{x} | \mathbf{y})$.

Implementation

- If we update one pixel at a time then $p(\mathbf{x}^{(i)} | \mathbf{y})$ is non-decreasing in i (and bounded above by 1) and thus the sequence of posterior probabilities $\{p(\mathbf{x}^{(i)} | \mathbf{y})\}$ is guaranteed to converge. That property alone does not ensure that the image sequence $\{\mathbf{x}^{(i)}\}$ converges but in practice it seems to work.
- In MATLAB it is more convenient to update all pixels simultaneously, as shown in the code example on p. 11.81; using (11.42) with simultaneous updates loses the guarantee that $\{p(\mathbf{x}^{(i)} | \mathbf{y})\}$ converges but also seems to work in practice.
- Alternatives based on updating (conditionally) independent groups of pixels are possible [28]; this is called **grouped coordinate ascent** [35].

Example. For additive white Gaussian noise with variance σ^2 , and the simple MRF model (11.41), the iteration (11.42) simplifies to be

$$x^{(i+1)}[m, n] = \arg \min_{\chi \in \mathcal{U}} \left(\frac{1}{2\sigma^2} |y[m, n] - \chi|^2 + 2\beta \left(\mathbb{I}_{\{\chi \neq x^{(i)}[m-1, n]\}} + \mathbb{I}_{\{\chi \neq x^{(i)}[m+1, n]\}} + \mathbb{I}_{\{\chi \neq x^{(i)}[m, n-1]\}} + \mathbb{I}_{\{\chi \neq x^{(i)}[m, n+1]\}} \right) \right).$$

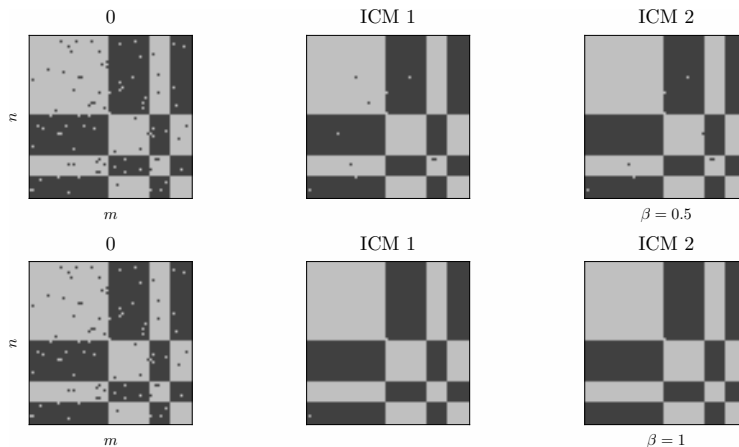
For each pixel we simply try each class value $\mu_k \in \mathcal{U}$ and see which one best agrees with both the data $y[m, n]$ and the neighbors. If $\beta = 0$ then agreement with the neighbors becomes irrelevant and the method reduces to the ML approach.

What happens as $\beta \rightarrow \infty$?

??

pair/class

The preceding case is pretty easy and ICM converges in 2 iterations, starting from the usual ML segmentation, as shown here for two values of β .



The ICM algorithm is very simple to implement in MATLAB. The following code uses periodic boundary conditions because it uses the `circshift` command.

```
% icm iteration for 2D image with 2 classes
% input is noisy image ynm and noise standard deviation sigma

beta = 2^(-1); % regularization parameter
values = [64 192]; % class means
xhat = values(1) + (values(2)-values(1)) * (ynm > 128); % ML classifier
for iter=1:9
    for kk=1:numel(values)
        val = values(kk);
        penalty = ...
            (circshift(xhat, [1 0]) ~= val) ...
            + (circshift(xhat, [-1 0]) ~= val) ...
            + (circshift(xhat, [0 1]) ~= val) ...
            + (circshift(xhat, [0 -1]) ~= val);
        score{kk} = (ynm - val).^2/2/sigma^2 + 2 * beta * penalty;
    end
    xhat = repmat(values(1), size(f));
    xhat(score{2} < score{1}) = values(2);
end
```

Color image segmentation

How would you generalize the MRF-ICM approach to segmenting color images? Hint: combining colors to make a grayscale “intensity” (luminance) image would be suboptimal, and segmenting each color component separately would also be suboptimal. ?? class/team

To quantify segmentation / classification accuracy, popular measures include **Jaccard index**, **Sorensen-Dice index**, and the **Rand index**.

Many other optimization methods have been developed including graph cuts, convex relaxation [38], and ADMM [39].

11.8 Summary

One way to reduce noise in an image is to apply a low-pass filter. To reduce blur, more sophisticated methods are needed such as the bilinear filter and the NLM method.

If the 2nd-order properties of the signal and noise are known, then the MMSE LSI method is the Wiener filter, given by (11.20) and (11.21). If the autocorrelation functions (or equivalently the power spectra) of the signal and/or noise are unknown, then they can, in principle, be estimated from measurements. Spectral estimation is a classic topic in statistical signal processing and entire books discuss it [40]; it was an active research area before sparsity models became more popular.

Likewise, there are numerous methods for image segmentation, some based on statistical models, others based on different formulations. In any case, groups of pixels must be considered jointly; methods that try to classify based on one pixel at a time will not perform well for noisy images. Some type of neighborhood information is essential. Markov random fields are one approach to modeling the statistical relationships between neighboring pixels.

Image denoising and image segmentation both are active research areas.

Bibliography

- [1] J. S. Lim. *Two-dimensional signal and image processing*. New York: Prentice-Hall, 1990. ISBN: 978-0139353222.
- [2] L. Cohen. “Generalization of the Wiener-Khinchin theorem”. In: *IEEE Signal Proc. Letters* 5.11 (Nov. 1998), 292–294.
- [3] L. V. Ahlfors. *Complex analysis*. 3rd ed. New York: McGraw-Hill, 1979.
- [4] S. Ramani, Z. Liu, J. Rosen, J-F. Nielsen, and J. A. Fessler. “Regularization parameter selection for nonlinear iterative image restoration and MRI reconstruction using GCV and SURE-based methods”. In: *IEEE Trans. Im. Proc.* 21.8 (Aug. 2012), 3659–72.
- [5] S. Daly. *The visual differences predictor: An algorithm for the assessment of image fidelity*. A.B. Watson (Ed.), Digital images and human vision, MIT Press, Cambridge (MA), pp. 179-206. 1993.
- [6] S. J. Reeves and A. C. Higdon. “Perceptual evaluation of the mean-square error choice of regularization parameter”. In: *IEEE Trans. Im. Proc.* 4.1 (Jan. 1995), 107–10.
- [7] Y. H. Jiang, D. L. Huo, and D. L. Wilson. “Methods for quantitative image quality evaluation of MRI parallel reconstructions: detection and perceptual difference model”. In: *Mag. Res. Im.* 25.5 (June 2007), 712–21.
- [8] P. Marziliano, F. Dufaux, S. Winkler, and T. Ebrahimi. “Perceptual blur and ringing metrics: application to JPEG2000”. In: *Signal Processing: Image Communication* 19.2 (Feb. 2004), 163–72.
- [9] H. H. Barrett. “Objective assessment of image quality: effects of quantum noise and object variability”. In: *J. Opt. Soc. Am. A* 7.7 (July 1990), 1266–1278.
- [10] H. H. Barrett, J. L. Denny, R. F. Wagner, and K. J. Myers. “Objective assessment of image quality. II. Fisher information, Fourier crosstalk, and figures of merit for task performance”. In: *J. Opt. Soc. Am. A* 12.5 (May 1995), 834–52.
- [11] H. H. Barrett, C. K. Abbey, and E. Clarkson. “Objective assessment of image quality III: ROC metrics, ideal observers and likelihood-generating functions”. In: *J. Opt. Soc. Am. A* 15.6 (June 1998), 1520–35.
- [12] H. R. Sheikh, M. F. Sabir, and A. C. Bovik. “A statistical evaluation of recent full reference image quality assessment algorithms”. In: *IEEE Trans. Im. Proc.* 15.11 (Nov. 2006), 3440–51.
- [13] Z. Wang and A. C. Bovik. “Mean squared error: Love it or leave it? A new look at signal fidelity measures”. In: *IEEE Sig. Proc. Mag.* 26.1 (Jan. 2009), 98–117.

- [14] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. “Image quality assessment: from error visibility to structural similarity”. In: *IEEE Trans. Im. Proc.* 13.4 (Apr. 2004), 600–12.
- [15] M. P. Sampat, Z. Wang, S. Gupta, A. C. Bovik, and M. K. Markey. “Complex wavelet structural similarity: A new image similarity index”. In: *IEEE Trans. Im. Proc.* 18.11 (Nov. 2009), 2385–401.
- [16] V. Laparra, A. Berardino, J. Ballé, and E. P. Simoncelli. *Perceptually optimized image rendering*. arxiv 1701.06641. 2017.
- [17] A. Leon-Garcia. *Probability and random processes for electrical engineering*. 2nd ed. New York: Addison-Wesley, 1994.
- [18] A. Clauset, C. Shalizi, and M. Newman. “Power-law distributions in empirical data”. In: *SIAM Review* 51.4 (Dec. 2009), 661–703.
- [19] A. Buades, B. Coll, and J. M. Morel. “A review of image denoising methods, with a new one”. In: *SIAM Multiscale Modeling and Simulation* 4.2 (2005), 490–530.
- [20] M. Elad and M. Aharon. “Image denoising via sparse and redundant representations over learned dictionaries”. In: *IEEE Trans. Im. Proc.* 15.12 (Dec. 2006), 3736–45.
- [21] A. Rajwade, A. Rangarajan, and A. Banerjee. “Image denoising using the higher order singular value decomposition”. In: *IEEE Trans. Patt. Anal. Mach. Int.* 35.4 (Apr. 2013), 849–62.
- [22] P. Milanfar. “A tour of modern image filtering: New insights and methods, both practical and theoretical”. In: *IEEE Sig. Proc. Mag.* 30.1 (Jan. 2013), 106–28.
- [23] G. Vaksman, M. Zibulevsky, and M. Elad. “Patch ordering as a regularization for inverse problems in image processing”. In: *SIAM J. Imaging Sci.* 9.1 (2016), 287–319.
- [24] T. Weissman, E. Ordentlich, G. Seroussi, S. Verdu, and M. J. Weinberger. “Universal discrete denoising: known channel”. In: *IEEE Trans. Info. Theory* 51.1 (Jan. 2005), 5–28.
- [25] K. Sivaramakrishnan and T. Weissman. “A context quantization approach to universal denoising”. In: *IEEE Trans. Sig. Proc.* 57.6 (June 2009), 2110–2129.
- [26] Y. Ma, J. Zhu, and D. Baron. “Approximate message passing algorithm with universal denoising and Gaussian mixture learning”. In: *IEEE Trans. Sig. Proc.* 64.21 (Nov. 2016), 5611–22.

- [27] S. Geman and D. Geman. “Stochastic relaxation, Gibbs distributions, and Bayesian restoration of images”. In: *IEEE Trans. Patt. Anal. Mach. Int.* 6.6 (Nov. 1984), 721–41.
- [28] J. Besag. “On the statistical analysis of dirty pictures”. In: *J. Royal Stat. Soc. Ser. B* 48.3 (1986), 259–302.
- [29] Z. Kato and J. Zerubia. “Markov random fields in image segmentation”. In: *Found. & Trends in Sig. Pro.* 5.1-2 (2012), 1–155.
- [30] C. A. Bouman. *Model based image processing [A guide to the tools of]*. ., 2013.
- [31] R. Dubes and A. Jain. “Random field models in image analysis”. In: *J. Appl. Stat.* 16.2 (1989), 131–64.
- [32] D. Geman and G. Reynolds. “Constrained restoration and the recovery of discontinuities”. In: *IEEE Trans. Patt. Anal. Mach. Int.* 14.3 (Mar. 1992), 367–83.
- [33] A. S. Willsky. “Multiresolution Markov models for signal and image processing”. In: *Proc. IEEE* 90.8 (Aug. 2002), 1396–1458.
- [34] J. Besag. “Spatial interaction and the statistical analysis of lattice systems”. In: *J. Royal Stat. Soc. Ser. B* 36.2 (1974), 192–236.
- [35] S. T. Jensen, S. Johansen, and S. L. Lauritzen. “Globally convergent algorithms for maximizing a likelihood function”. In: *Biometrika* 78.4 (Dec. 1991), 867–77.
- [36] M. A. T. Figueiredo. “Bayesian image segmentation using wavelet-based priors”. In: *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*. 2005, 437–43.
- [37] M. A. T. Figueiredo, D. S. Cheng, and V. Murino. “Clustering under prior knowledge with application to image segmentation”. In: *NIPS*. 2006.
- [38] T. Pock, A. Chambolle, D. Cremers, and H. Bischof. “A convex relaxation approach for computing minimal partitions”. In: *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*. 2009, 810–7.
- [39] M. Storath and A. Weinmann. “Fast partitioning of vector-valued images”. In: *SIAM J. Imaging Sci.* 7.3 (2014), 1826–52.
- [40] S. M. Kay. *Modern spectral estimation*. New York: Prentice-Hall, 1988.

Chapter 13

Sparsity and wavelets

Contents (class version)

| | |
|--|--------------|
| 13.0 Introduction | 13.2 |
| 13.1 Sparsity overview | 13.2 |
| Sparsity in the standard basis | 13.3 |
| Sparsity in the standard basis for a typical image | 13.7 |
| Sparsity in DFT basis? | 13.9 |
| Sparsity in blockwise / patch DCT basis | 13.11 |
| Sparsity in DWT basis | 13.13 |
| 13.2 Discrete wavelet transforms | 13.15 |
| 1D discrete wavelet transforms and filter banks | 13.15 |
| Discrete orthonormal wavelet transform: Haar | 13.20 |
| 2D DWT | 13.21 |
| Denoising Example | 13.27 |
| Denoising by hard thresholding example | 13.28 |
| 13.3 Denoising with sparsity-based regularization | 13.29 |

13.0 Introduction

This chapter provides some (very incomplete) notes and examples to supplement the material in the Image Restoration chapter [1]. For more thorough overviews of sparsity in signal processing, see the June 2010 issue of the IEEE Proceedings [2–8]. For an interesting example of sparsity used in ultrafast photography see [9].

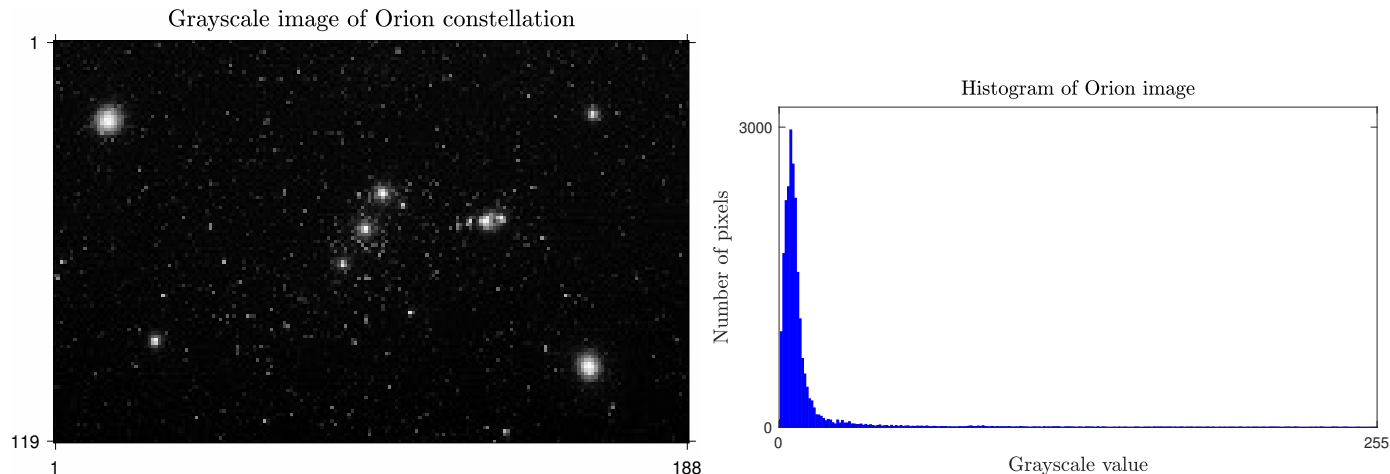
13.1 Sparsity overview

In the contemporary signal and image processing literature, it has become popular to use models based on some form of **sparsity** of signals as the foundation for developing new algorithms. This section describes some of the intuition underlying such models and shows how they are useful for image denoising and image restoration.

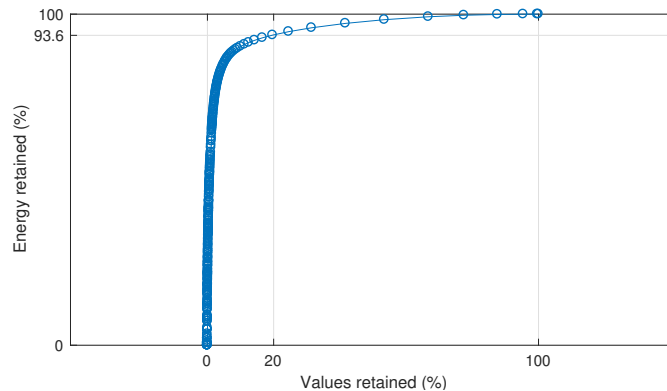
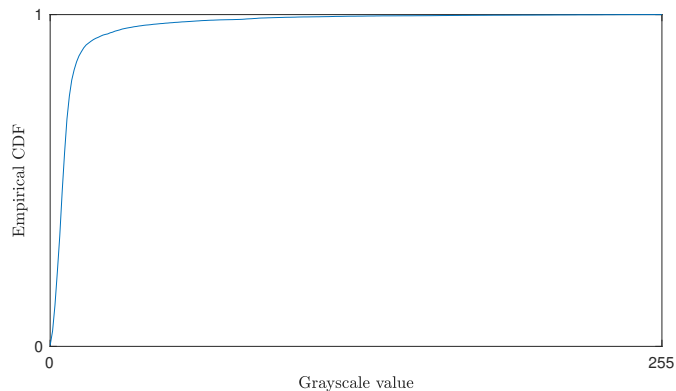
Sparsity in the standard basis

The following figure shows an image that looks to be **sparse**, meaning that many of the pixel values are zero. To be more precise we say that image is **compressible** (in the **standard basis** or **canonical basis**) because many of the pixel values are *nearly* zero rather than being exactly zero. But in casual speaking about sparsity people often do not distinguish carefully between sparsity and compressibility.

To examine the sparsity of this image quantitatively we can first look at its histogram shown on the right below; the histogram is concentrated near zero, as expected for a “starry night” image.



The empirical **cumulative distribution function (CDF)** is obtained by integrating the pdf (for continuous-valued random variables) or a running sum of the PMF (for discrete-valued random variables); here we normalize the histogram shown above by the number of pixels to make PMF, and then “integrate” using MATLAB’s `cumsum`. The resulting CDF, on the right below, rises sharply, also showing that many values are near zero.

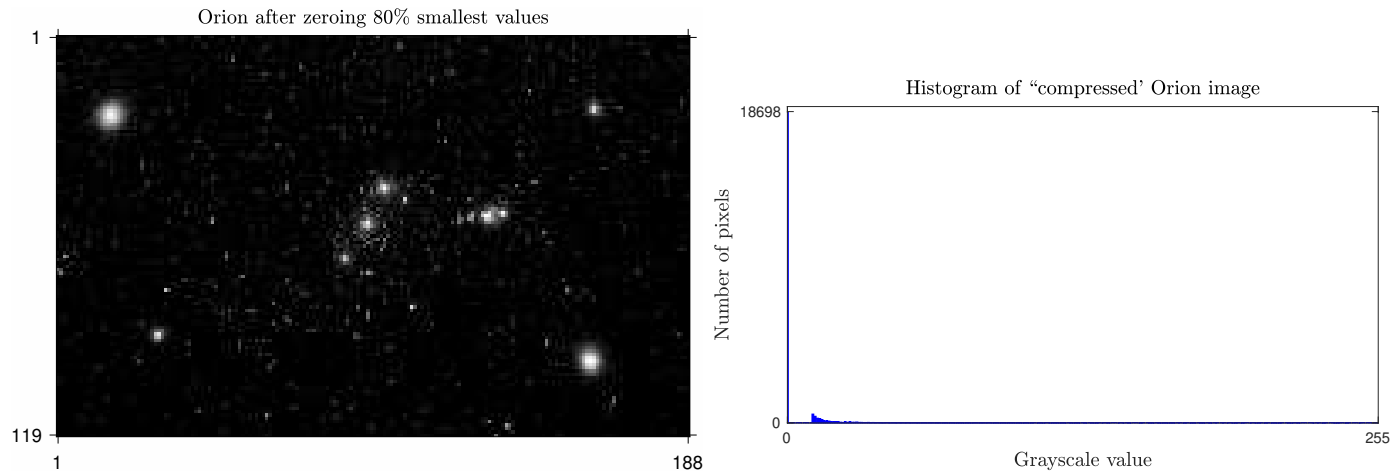


It might seem strange at first to discuss a statistical concept like the CDF for a single given image. As discussed in Ch. 10, the CDF shown above corresponds to a hypothetical probability experiment in which we select pixel values at random from the image.

Setting some small pixel values to zero only slightly changes the image energy.

The graph (above right) of energy retained as a function of percentage of image values not set to zero shows that we can “keep” (not zero out) only 20% of the values and still retain 93.5% of the image energy.

The figure on the left below shows that setting the smallest 80% of image values to zero leads to no visible change in the image, despite a fairly high NRMSE.



Exercise. What is the NRMSE of the thresholded image above?

$$\text{NRMSE} = \frac{\|\hat{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} = \boxed{??}$$

class/team

Mathematically one could say that here we representing an image $g[m, n]$ using a basis of Kronecker impulse functions (called the standard basis) as follows:

$$g[m, n] = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} \underbrace{\delta_2[m-k, n-l]}_{\text{basis functions}} \underbrace{g[k, l]}_{\text{coefficients}} ,$$

and we are considering whether the “coefficients” $\{g[k, l]\}$ are sparse.

In linear algebra notation we are using a model like:

$$\underbrace{\mathbf{x}}_{\text{signal}} = \underbrace{\mathbf{I}}_{\text{basis}} \underbrace{\mathbf{x}}_{\text{coefficients}} .$$

We will examine other bases shortly, because the pixel values in *most* images are *not* themselves sparse, *i.e.*, most images are not sparse with respect to the standard basis consisting of Kronecker impulses, as the following example shows.

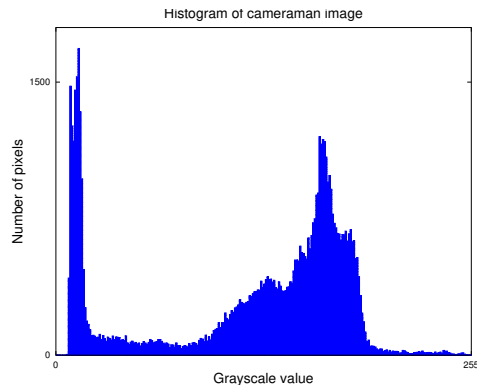
Sparsity in the standard basis for a typical image

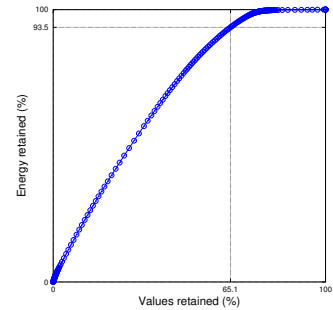
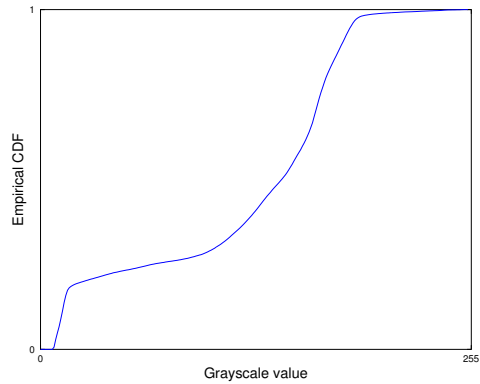
Example. Consider the cameraman image shown below. In this case the histogram is widely spread over 0 to 255, the empirical CDF rises slowly, and we need 65% of the largest values to retain 93% of the energy. Even with that large fraction of the values the image is visibly corrupted by zeroing the “small” values.

The NRMSE of the “sparsified” image shown on the next page is

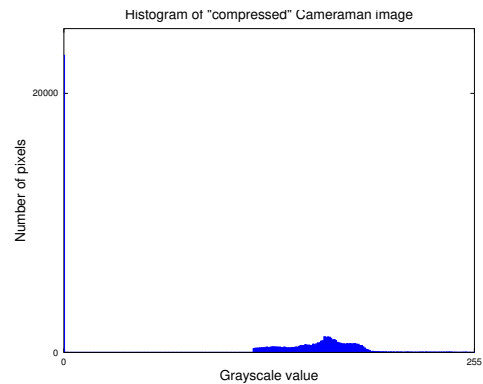
$$\text{NRMSE} = \frac{\|\hat{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} = \frac{\sqrt{\sum_j (\hat{x}_j - x_j)^2}}{\sqrt{\sum_j x_j^2}} 100\% = \sqrt{\frac{\sum_{j:\hat{x}_j=0} x_j^2}{\sum_j x_j^2}} 100\% = \sqrt{0.065} \times 100\% = 25\%.$$

This is the same NRMSE as the compressed version of the Orion image, but the errors are much more visible here.





After zeroing 35% smallest values

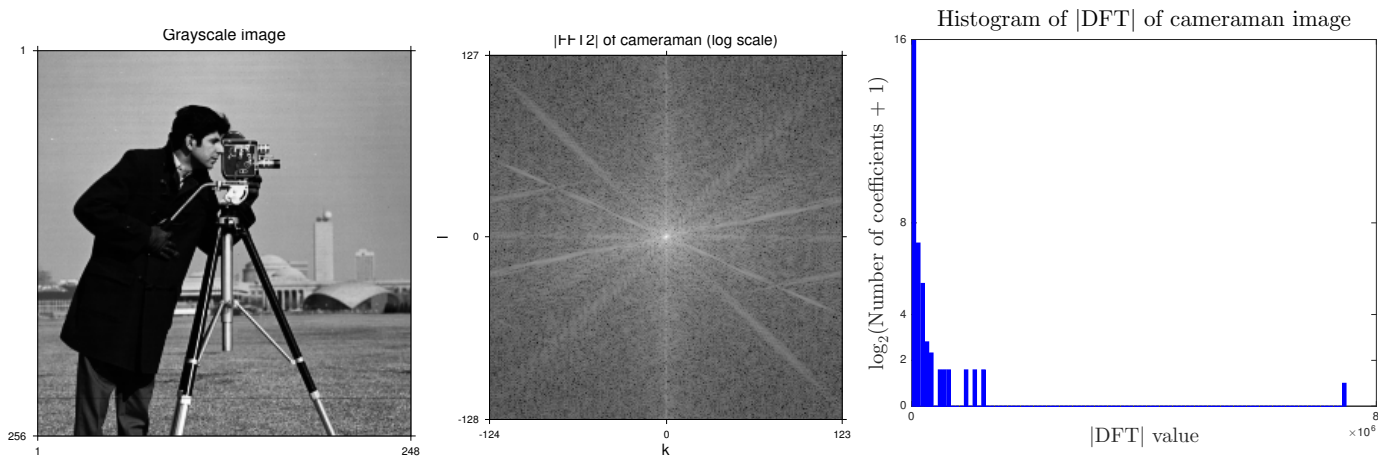


Sparsity in DFT basis?

If we take an appropriate *transform* (e.g., a wavelet transform) of a natural image, often the resulting coefficients are sparse. This is called **transform sparsity**. If the transform is invertible (e.g., unitary), then we say the image is sparse with respect to the basis.

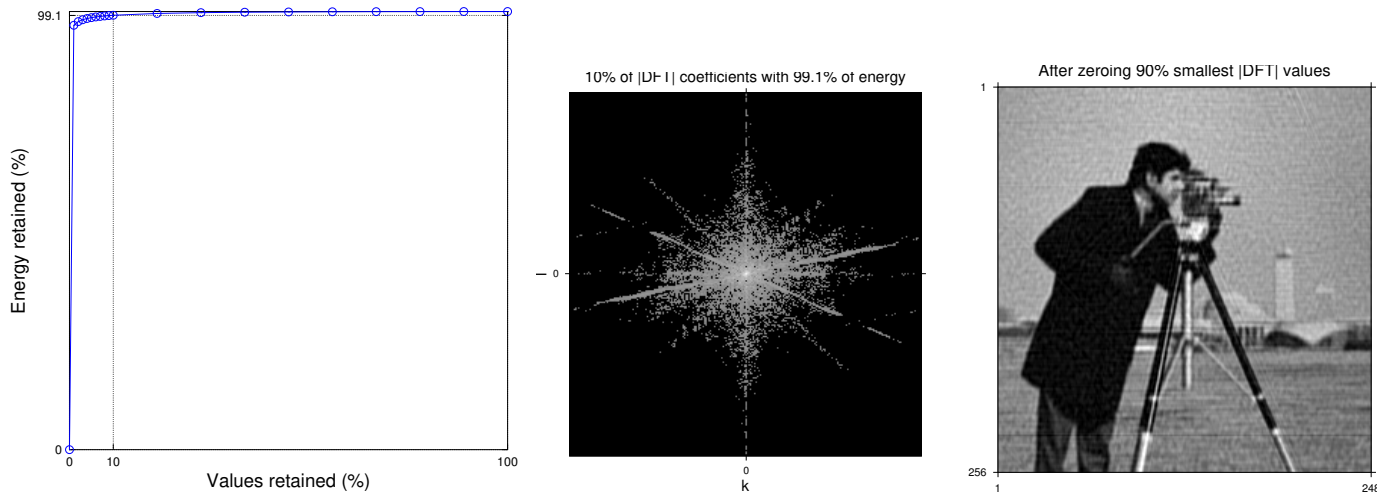
In other words, if \mathbf{W} is an invertible transform and $\mathbf{W}\mathbf{x}$ is (approximately) sparse, then we can write $\mathbf{x} \approx \mathbf{B}\mathbf{z}$ where $\mathbf{B} = \mathbf{W}^{-1}$ is the basis and \mathbf{z} is the vector of coefficients where many of the elements of \mathbf{z} are zero or “small.” (I like to use “ \mathbf{z} ” for sparse coefficients because the letter reminds us that many of the elements of \mathbf{z} are zero.)

We first examine the DFT (using a basis consisting of harmonic discrete-space complex exponential signals) of a natural image. We display the spectrum (FFT) on a log scale (otherwise it would look extremely sparse, but misleadingly so). The histogram of $|\text{DFT}|$ shows how sparse it appears to be, on a linear scale.



In this case, only 10% of the largest (magnitude!) DFT coefficients are needed to explain 99.1% of the energy of the image.

However, the image synthesized (by IFFT2) from those 10% of the coefficients having the largest magnitudes is severely corrupted to the eye; yet the NRMSE is 9.5%.



To elaborate mathematically on what is being done here, let B denote the (unitary) inverse DFT and define

$$\hat{\mathbf{x}} = B\mathbf{z}, \quad \mathbf{z} = \text{HardThreshold}(B'\mathbf{x}, \tau), \quad \text{i.e.,} \quad z_j = \begin{cases} [B'\mathbf{x}]_j, & |[B'\mathbf{x}]_j| > \tau \\ 0, & \text{otherwise,} \end{cases}$$

where the threshold τ was selected (by sorting the DFT coefficients) such that only 10% of the elements of \mathbf{z} are non-zero.

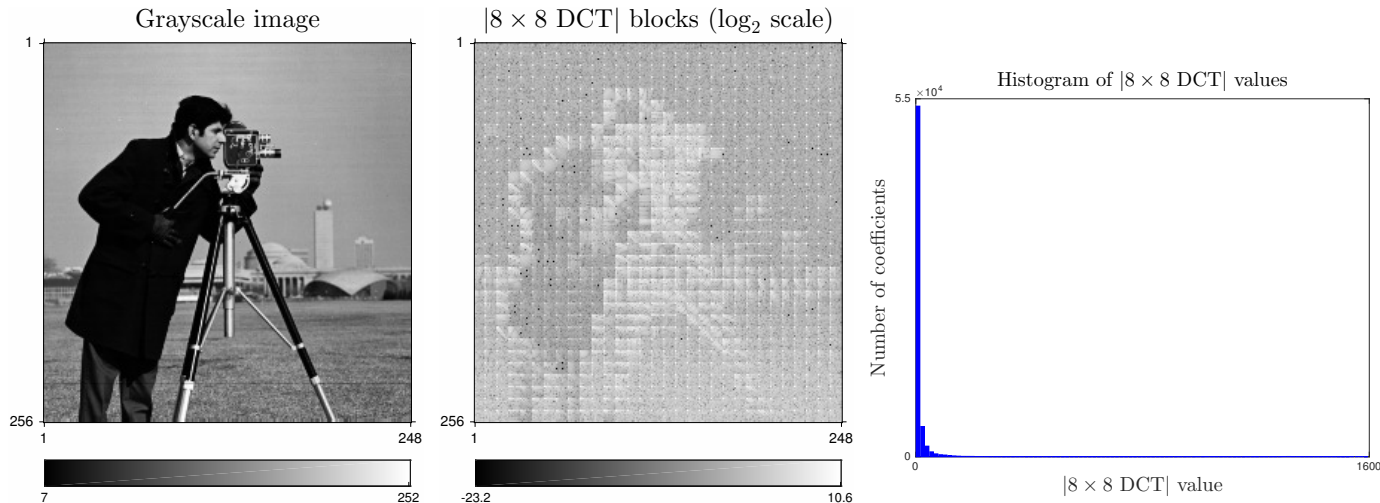
Sparsity in blockwise / patch DCT basis

A problem with the DFT is that its basis functions are all global, so changing (e.g., zeroing or quantizing) even just one DFT coefficient *affects the entire image*. Furthermore it is complex-valued which is inconvenient for implementation. The DCT is real valued, and thus more convenient, but the usual DCT basis also affects the entire image.

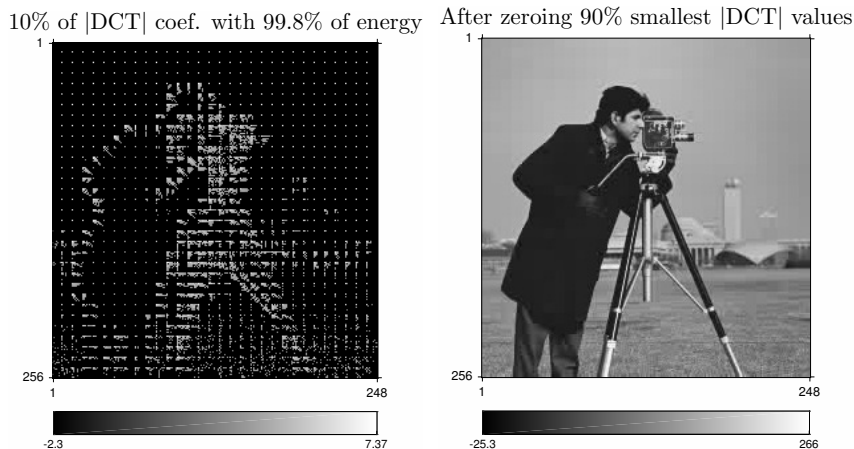
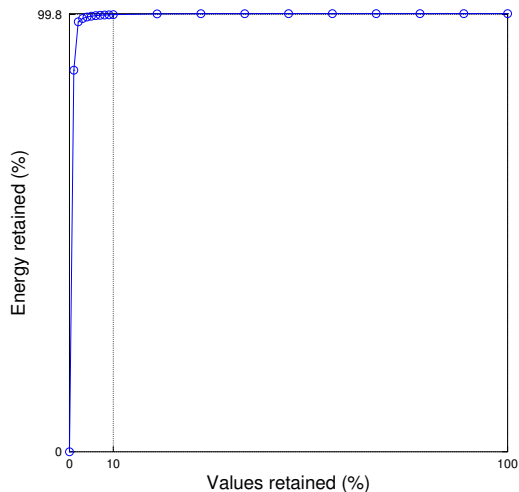
To get better spatial localization, we can apply the 2D DCT to each, say, 8×8 block (i.e., patch) of an image. The original **JPEG** image compression standard also uses such block-wise operations.

Is the block-wise DCT a unitary transform? **??**

class/team



This time, retaining only the 10% largest (absolute!) values maintains 99.8% of the energy with a NRMSE of 4.5%.



For most blocks of 8×8 pixels, which DCT coefficient is the largest? ??

[RQ]

Which blocks have additional especially large absolute DCT coefficient values? [blocks with high contrast edges](#).

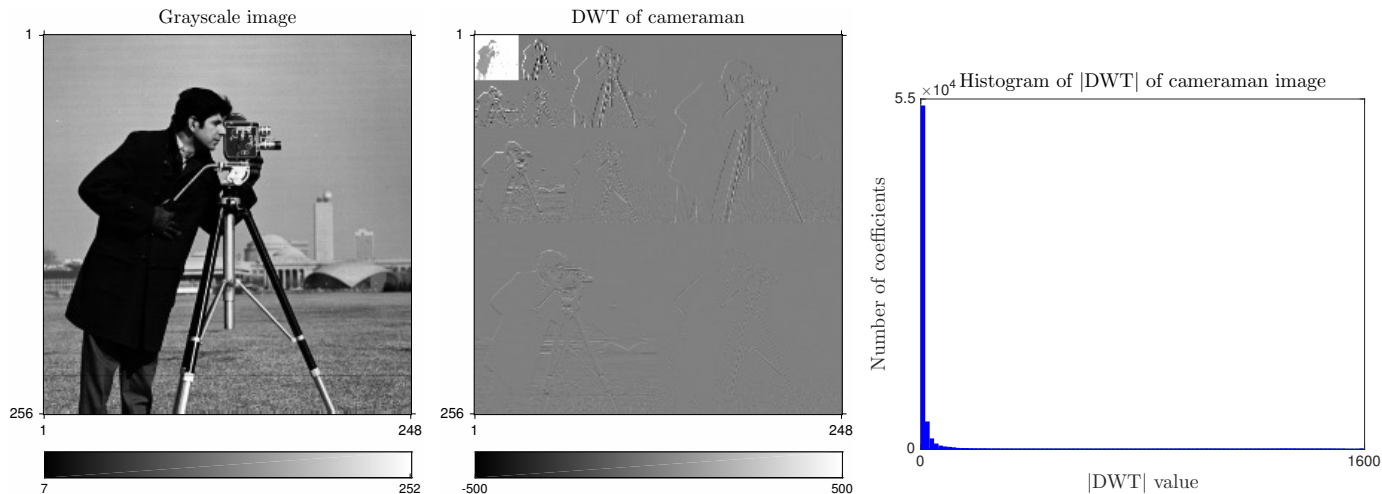
Exercise. If B_8 denotes the 8×8 (unitary) inverse 1D DCT matrix, then the basis for a single 8×8 patch stored lexicographically is $B_8 \otimes B_8$. Determine the form of the *image* basis matrix B here such that $x \approx Bz$, assuming x denotes a $M \times N$ image stored lexicographically and that M and N are multiples of 8. Hint: try 1D first. class/team

??

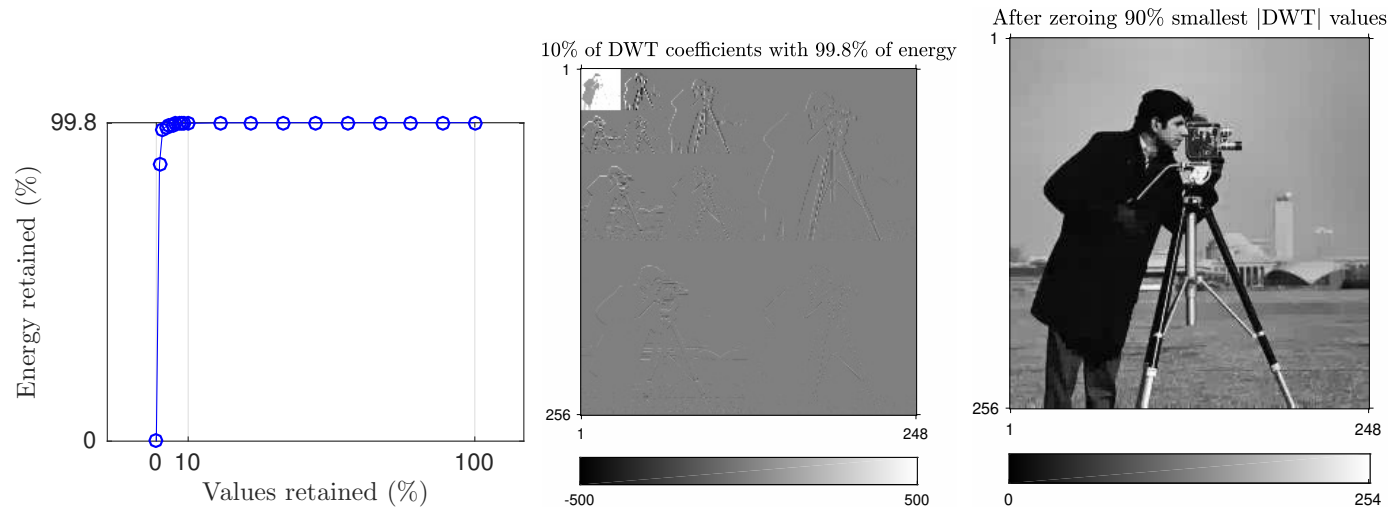
Sparsity in DWT basis

Next we examine sparsity in a **discrete wavelet transform (DWT)** basis. (This basis is discussed later in the chapter.)

It is clear in this middle figure that most of the wavelet coefficients are nearly 0 (gray is zero in the middle figure), *i.e.*, the image is sparse in the wavelet basis, as also seen in the histogram.



Interestingly, like the DCT, retaining only the 10% largest (absolute!) values of the DWT maintains 99.8% of the energy with a NRMSE of 4.5%. (The fact we get the same NRMSE values for DCT and DWT is just a coincidence.)



Wavelet transforms are particularly popular for sparsity based signal models.

For medical examples (in MRI) showing how setting small coefficients to zero affects image quality, see [10].

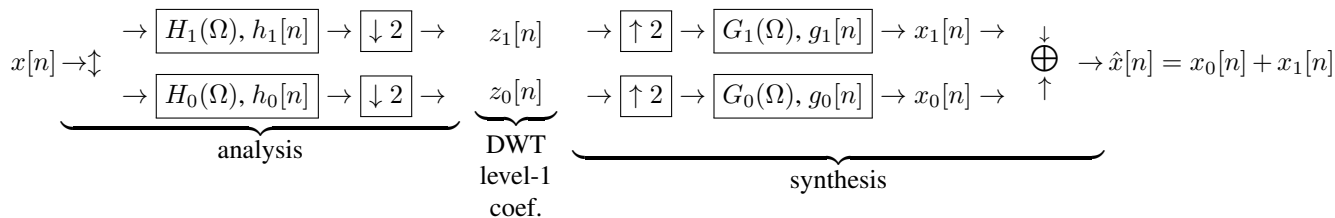
13.2 Discrete wavelet transforms

A family of orthonormal transforms that are often particularly effective at sparsifying natural images is **discrete wavelet transforms (DWT)**.

A basic DWT in 1D perhaps is easiest understood using a **filter bank** perspective.

1D discrete wavelet transforms and filter banks

The following diagram illustrates both the **analysis** and **synthesis** stages of a DWT implemented using filter banks and **decimation**.



- Usually we design the 4 filters so that $\hat{x}[n] = x[n]$ which is called **perfect reconstruction**.
- Usually $h_0[n]$ is a “low pass” filter and $h_1[n]$ is a “high pass” filter.
- Purpose: compress or regularize the details in $z_1[n]$, *i.e.*, we might set to zero many of the $z_1[n]$ coefficient values but still reconstruct a signal $\hat{x}[n]$ that is a close match to the original signal $x[n]$.
- We can apply the same type of decomposition to the down-sampled signal $z_0[n]$, recursively.

Example. The filters for the **Haar wavelet** are:

$$h_0[n] = \frac{1}{\sqrt{2}}[1 \ 1], \quad h_1[n] = \frac{1}{\sqrt{2}}[-1 \ 1], \quad g_0[n] = \frac{1}{\sqrt{2}}[1 \ 1], \quad g_1[n] = \frac{1}{\sqrt{2}}[1 \ -1]$$

Note that $h_0[n]$ and $g_0[n]$ are low-pass filters similar to a moving average, whereas $h_1[n]$ and $g_1[n]$ are finite-difference (thus high-pass) filters.

Analysis stage:

$$z_1[n] = (h_1 * x)_{\downarrow 2}[n] = (h_1 * x)[2n] = \frac{1}{\sqrt{2}}(x[2n] - x[2n + 1]) \quad \text{“details”}$$

$$z_0[n] = (h_0 * x)_{\downarrow 2}[n] = (h_0 * x)[2n] = \frac{1}{\sqrt{2}}(x[2n] + x[2n + 1]) \quad \text{“approximation”}$$

Synthesis:

$$x_1[n] = g_1[n] * z_{1\uparrow 2}[n] \implies \begin{cases} x_1[2n] &= \frac{1}{\sqrt{2}} z_1[n] &= \frac{1}{2} (x[2n] - x[2n + 1]) \\ x_1[2n + 1] &= -\frac{1}{\sqrt{2}} z_1[n] &= -\frac{1}{2} (x[2n] - x[2n + 1]) \end{cases}$$

$$x_0[n] = g_0[n] * z_{0\uparrow 2}[n] \implies \begin{cases} x_0[2n] &= \frac{1}{\sqrt{2}} z_0[n] &= \frac{1}{2} (x[2n] + x[2n + 1]) \\ x_0[2n + 1] &= \frac{1}{\sqrt{2}} z_0[n] &= \frac{1}{2} (x[2n] + x[2n + 1]) \end{cases}$$

$$\hat{x}[n] = x_0[n] + x_1[n] \implies \begin{cases} \hat{x}[2n] &= \frac{1}{\sqrt{2}} (z_0[n] + z_1[n]) &= x_0[2n] + x_1[2n] = x[2n] \\ \hat{x}[2n + 1] &= \frac{1}{\sqrt{2}} (z_0[n] - z_1[n]) &= x_0[2n + 1] + x_1[2n + 1] = x[2n + 1] \end{cases}$$

Do the Haar wavelet filters provide **perfect reconstruction**? Yes, because $\hat{x}[n] = x[n]$

Linear algebra perspective for $N = 4$:

$$\underbrace{\begin{bmatrix} z_0[0] \\ z_0[1] \\ z_1[0] \\ z_1[1] \end{bmatrix}}_{\text{coefficients}} = \frac{1}{\sqrt{2}} \underbrace{\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}}_{\text{analysis transform}} \underbrace{\begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix}}_{\text{signal}}, \quad \begin{bmatrix} \hat{x}[0] \\ \hat{x}[1] \\ \hat{x}[2] \\ \hat{x}[3] \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} z_0[0] \\ z_0[1] \\ z_1[0] \\ z_1[1] \end{bmatrix}$$

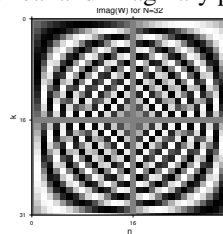
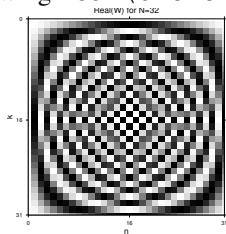
$$z = \mathbf{W}x \qquad x = \mathbf{B}z$$

This is an orthogonal wavelet transform because \mathbf{W} is unitary and $\mathbf{B} = \mathbf{W}^{-1} = \mathbf{W}'$.

How does this filter bank correspond to basis functions?

For a DFT, the basis functions are complex exponentials. We can determine the corresponding basis functions by setting all the transform (DFT) coefficients to zero except for one and seeing what is produced by the DFT synthesis formula. Let $X_l[k] = \delta[k - l]$ for $l = 0, \dots, N - 1$, then by the DFT synthesis formula: $x_l[n] = \frac{1}{N} e^{i2\pi nl/N}$.

Recall this figure showing `real(dftmtx(32))` and `imag(dftmtx(32))` for the real and imaginary parts of the DFT basis:



For the 1-level DWT, the transform coefficients are $\{z_0[n] : n = 0, \dots, N/2 - 1\}$ and $\{z_1[n] : n = 0, \dots, N/2 - 1\}$.

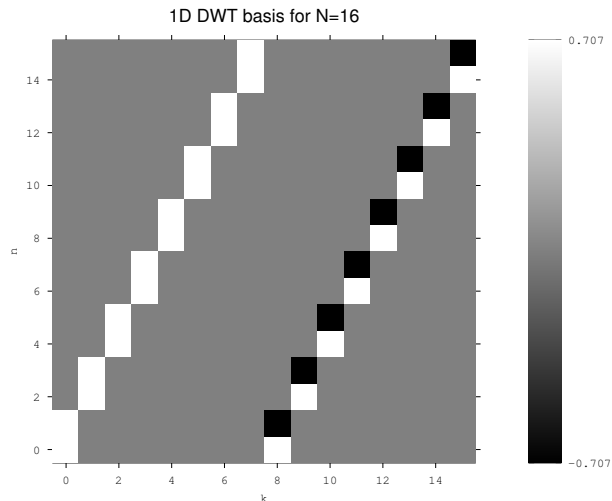
Again we can determine the corresponding basis functions by setting all the transform coefficients to zero except for one and seeing what is produced by the DWT synthesis process.

If we set $z_1[n] = 0$ and let $z_0[n] = \delta[n - k]$ for some $k \in \{0, \dots, N/2 - 1\}$, then we get $g_0[n - 2k]$.

If we set $z_0[n] = 0$ and let $z_1[n] = \delta[n - k]$ for some $k \in \{0, \dots, N/2 - 1\}$, then we get $g_1[n - 2k]$.

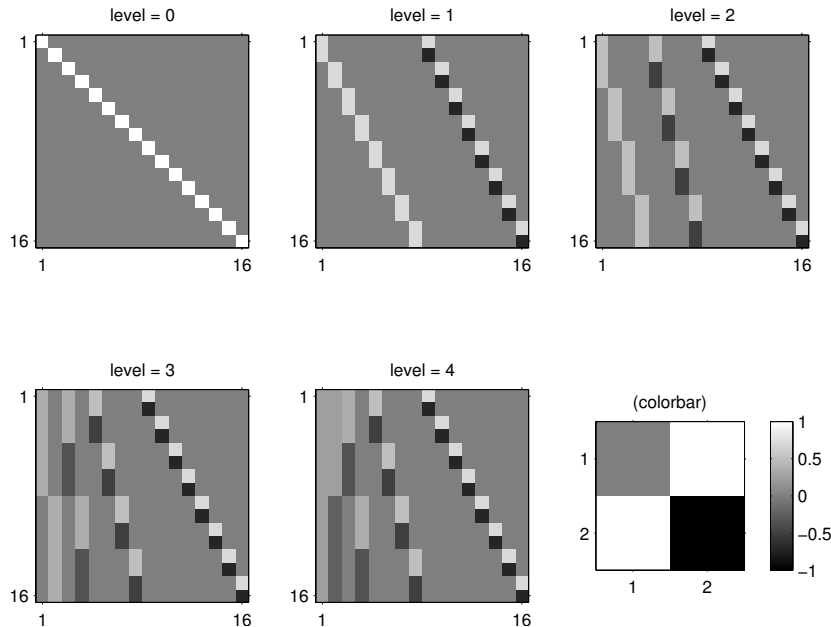
So the basis functions for a 1-level DWT are $\{g_0[n - 2k] : k = 0, \dots, N/2 - 1\} \cup \{g_1[n - 2k] : k = 0, \dots, N/2 - 1\}$.

This figure illustrates the 16 basis functions (each column) for $N = 16$ for the 1-level DWT for the Haar wavelet.



Discrete orthonormal wavelet transform: Haar

This figure shows the 1D Haar DWT for various levels of decomposition for a signal of length $N = 16$. These pictures should be viewed as N 1D signals of length N (each column of the pictures) rather than as 2D “images.”



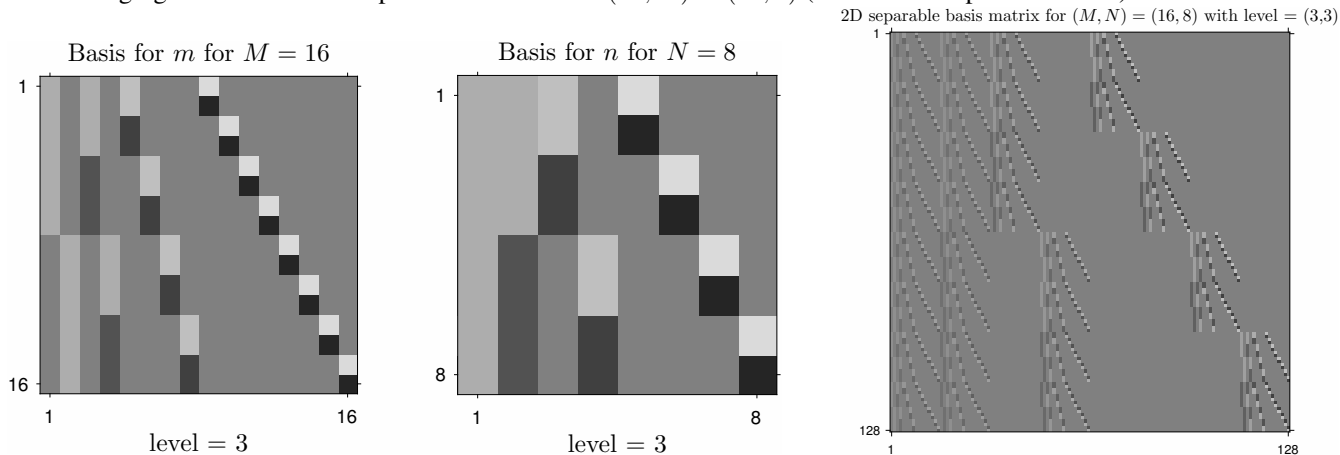
Most of these basis functions are localized spatially and localized in **scale**, unlike the DFT and DCT.

2D DWT

2D DWT using separability

When I first learned about wavelets, I assumed the “standard” 2D DWT would be separable, just like the 2D DFT and 2D DCT. It turns out that the standard 2D DWT is *not* separable, but I illustrate the separable version anyway for completeness.

The following figures illustrate the separable 2D DWT for $(M, N) = (16, 8)$ (for full decomposition levels)



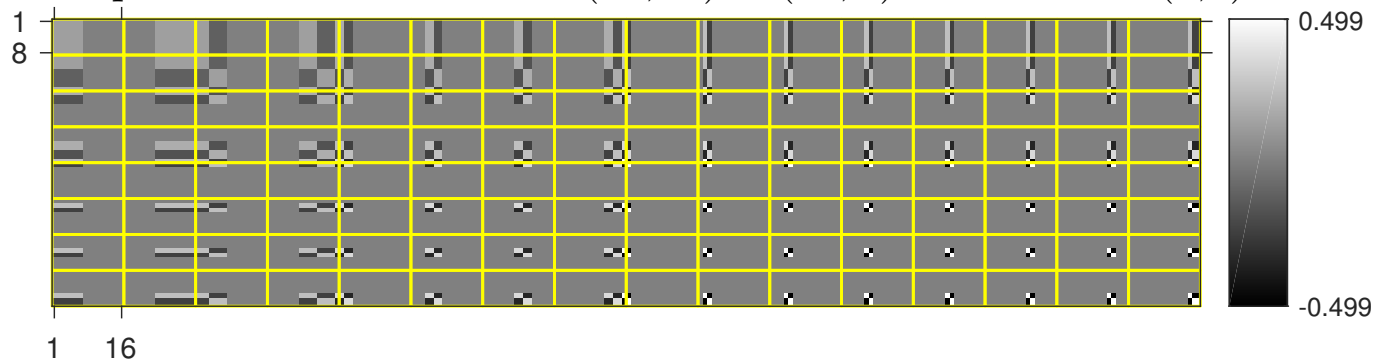
Mathematically, the orthogonal basis for the (non-standard) separable version of the 2D DWT has the form

$$B = B_N \otimes B_M = \underbrace{(B_N \otimes I_M)(I_N \otimes B_M)}_{\text{DWT along } m \text{ first, then along } n} = \underbrace{(I_N \otimes B_M)(B_N \otimes I_M)}_{\text{DWT along } n \text{ first, then along } m},$$

where \mathbf{B}_N denotes the N -point 1D DWT. This Kronecker product form is apparent in the right-hand figure above. However, viewing \mathbf{B} as a $MN \times MN$ matrix is not very intuitive because each column of \mathbf{B} corresponds to a 2D $M \times N$ basis function reshaped into a column. The next figure shows each of the MN columns of \mathbf{B} reshaped into a $M \times N$ array, and all of them arranged in a $M \times N$ mosaic. This is a more natural way to display the basis functions for this small 2D DWT.

Basis functions for separable 2D DWT for $(M, N) = (16, 8)$

2D separable basis mosaic for $(M, N) = (16, 8)$ with level = (3,3)

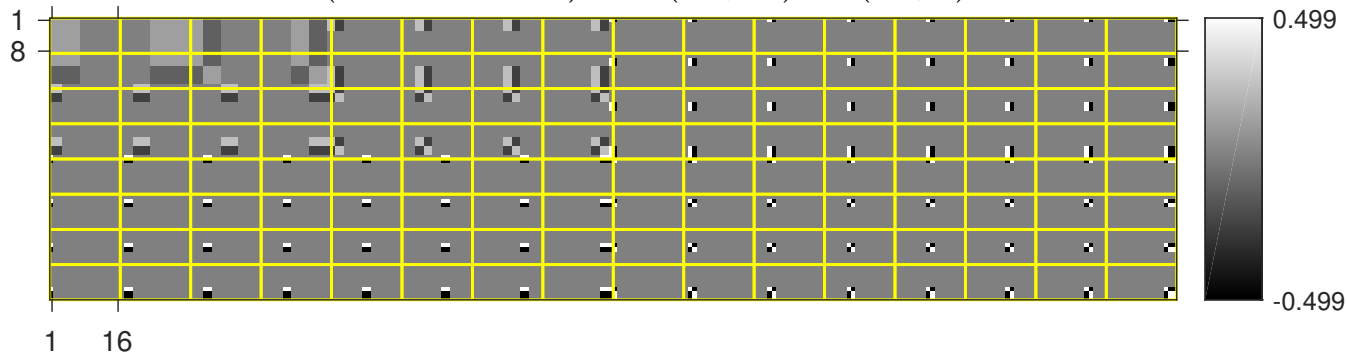


For comparison, here is the standard 2D orthonormal Haar wavelet basis in MATLAB (using `wavedec2`).

What is similar and what is different?

cf. 2D DFT basis

2D Haar wavelets (orthonormal) for $(M, N) = (16, 8)$ with level = 3

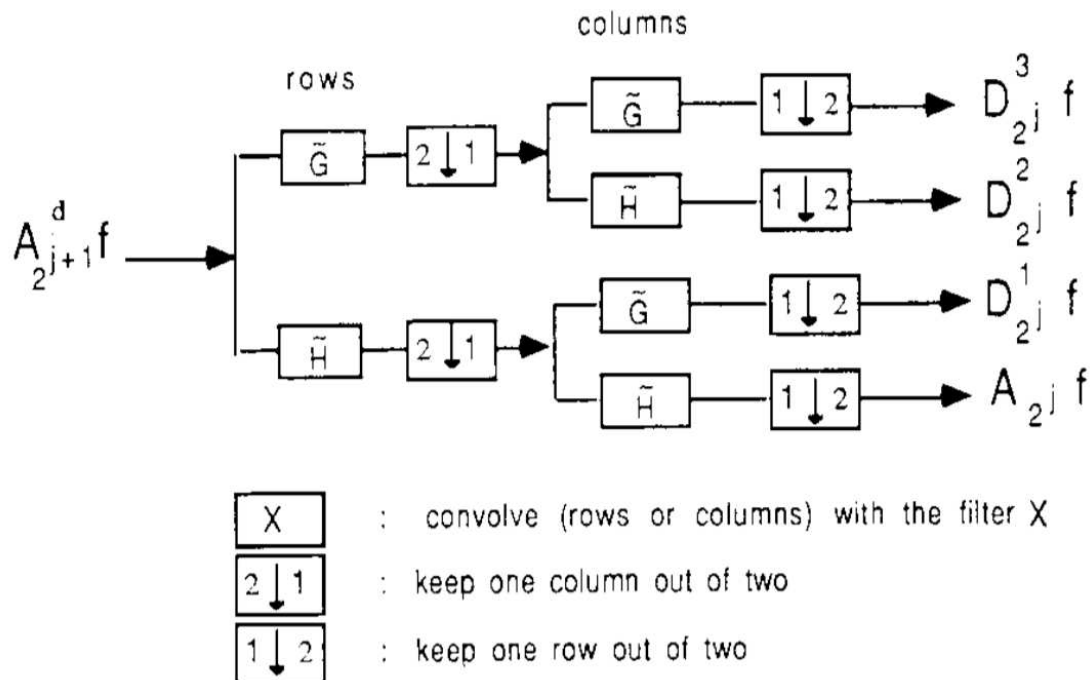


MATLAB's `dwt2` documentation says:

“For images, there exist an algorithm similar to the one-dimensional case for two-dimensional wavelets and scaling functions obtained from one-dimensional ones by **tensorial product**.

This kind of two-dimensional DWT leads to a decomposition of approximation coefficients at level j in four components: the approximation at level $j + 1$, and the details in three orientations (horizontal, vertical, and diagonal).”

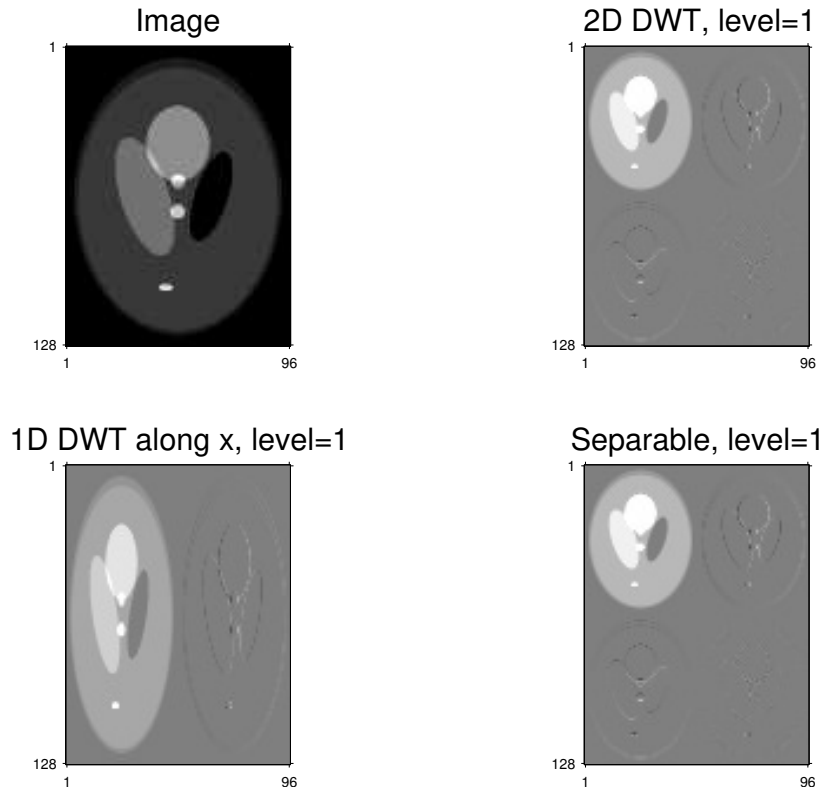
The following figure [11, Fig. 12] describes the basic decomposition steps for images.



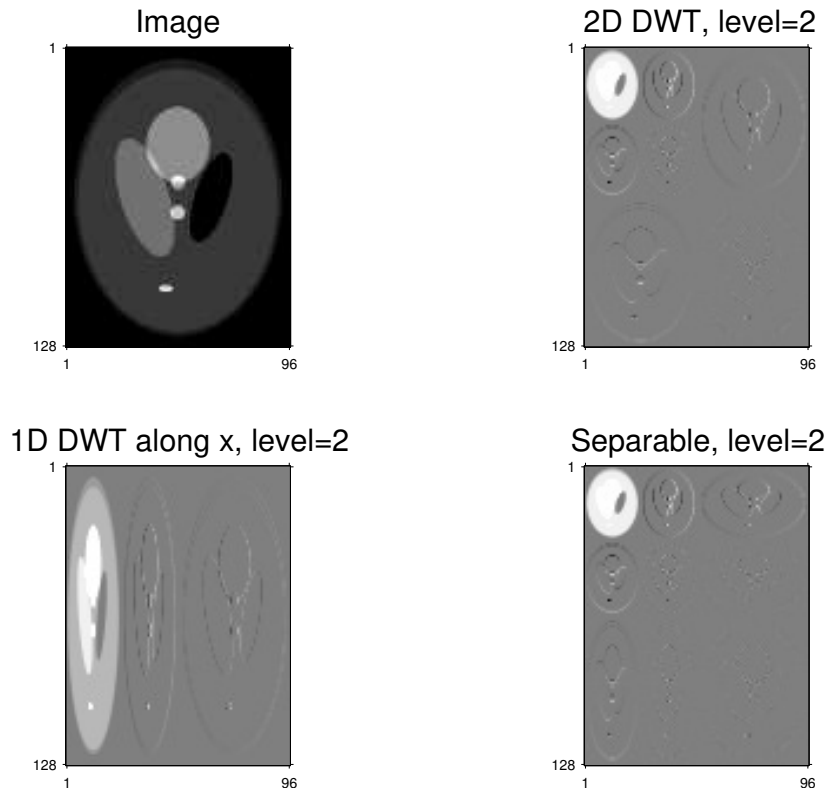
In this diagram, \tilde{H} denotes a low-pass filter and \tilde{G} denotes a high-pass filter.

Next we illustrate the similarities and differences between the separable 2D DWT approach and the standard 2D DWT approach used in image processing.

This figure shows the case of a single level. Here the separable version and the conventional version are the same. Although the grayscale pictures here are displayed like “images,” these pictures show the DWT **transform coefficients**.

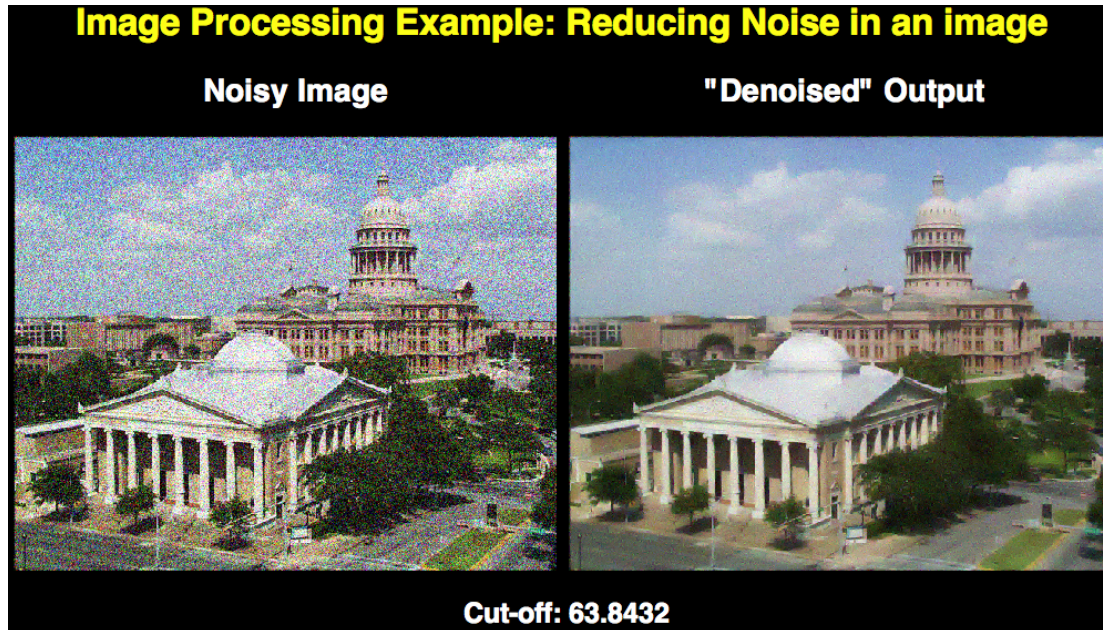


This figure shows the level = 2 case. The conventional version performs another decomposition of the approximation coefficients (upper left quadrant). The separable version decomposes all rows and columns, yielding different set of coefficients. Whereas DFT transform coefficients of a natural scene are somewhat difficult to interpret, DWT coefficients are perhaps somewhat easier for our eyes to understand. Nevertheless, both the DFT and the DWT are unitary transforms with fast methods.



Denoising Example

Now we apply the 2D DWT for image denoising, using methods explained mathematically after the figures.



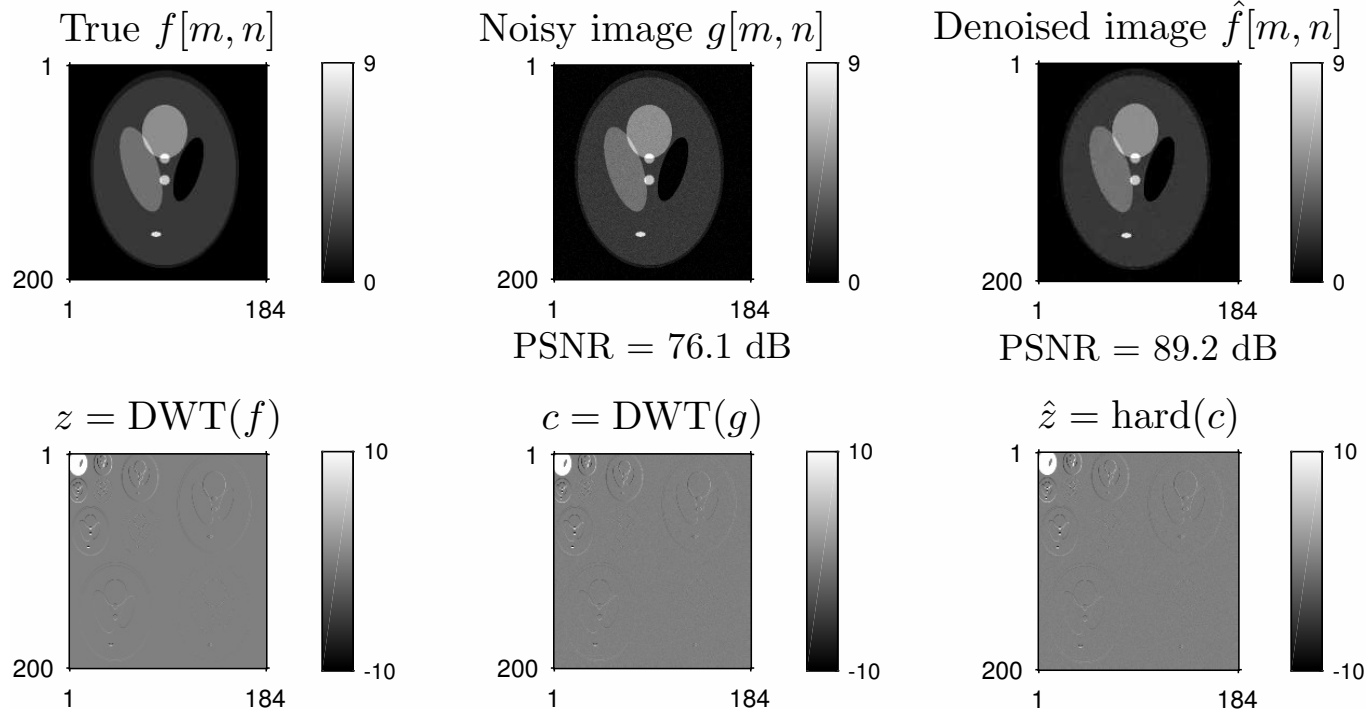
Using 2D discrete stationary wavelet transform (**SWT**) with soft thresholding, using MATLAB's `swt2` and `iswt2`.

Demo created by Sathish Ramani. `src/matlab/contrib/ramani-demo-tech-day/denoisExample.m`

Denoising by hard thresholding example

n-13-sparse/fig/hard_threshold1

The technique used for this example is explained in the next section.



13.3 Denoising with sparsity-based regularization

We often refer to two formulations for denoising an image with sparsity models.

We refer to the **analysis formulation** when we solve directly for the image pixel values \mathbf{x} assuming some transform of \mathbf{x} is sparse:

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x}} \frac{1}{2} \|\mathbf{y} - \mathbf{x}\|_2^2 + \beta \|\mathbf{C}\mathbf{x}\|_0, \quad (13.1)$$

where the following “pseudo-norm” counts the number of non-zero elements of a vector:

$$\|\mathbf{x}\|_0 \triangleq \sum_j \mathbb{I}_{\{x_j \neq 0\}}.$$

Optimization problem (13.1) is challenging because of the coupling caused by \mathbf{C} .

We refer to the **synthesis formulation** when we solve for the (sparse) coefficients \mathbf{z} :

$$\hat{\mathbf{z}} = \arg \min_{\mathbf{z}} \frac{1}{2} \|\mathbf{y} - \mathbf{B}\mathbf{z}\|_2^2 + \beta \|\mathbf{z}\|_0 \quad (13.2)$$

and then synthesize the denoised image using

$$\hat{\mathbf{x}} = \mathbf{B}\hat{\mathbf{z}}.$$

Optimization problem (13.2) is challenging (in general) because of the coupling caused by \mathbf{B} .

When \mathbf{B} and \mathbf{C} are invertible (e.g., unitary) and $\mathbf{B} = \mathbf{C}^{-1}$, then formulations (13.1) and (13.2) are equivalent.

The DFT, DCT, and DWT are **orthogonal transforms**, i.e., \mathbf{B} is a **unitary matrix**. The 2-norm is invariant to orthogonal transforms, i.e., when \mathbf{B} is unitary, $\|\mathbf{B}\mathbf{x}\|_2 = \sqrt{\mathbf{x}'\mathbf{B}'\mathbf{B}\mathbf{x}} = \sqrt{\mathbf{x}'\mathbf{x}} = \|\mathbf{x}\|_2$, so we can rewrite the synthesis problem in terms of the transform coefficients:

$$\hat{\mathbf{z}} = \arg \min_{\mathbf{z}} \frac{1}{2} \|\mathbf{B}'\mathbf{y} - \mathbf{z}\|_2^2 + \beta \|\mathbf{z}\|_0.$$

For convenience, let $\mathbf{c} \triangleq \mathbf{B}^{-1}\mathbf{y} = \mathbf{B}'\mathbf{y}$ denote the transform coefficients of the noisy image \mathbf{y} . We rewrite both norms as summations across the elements of \mathbf{c} and \mathbf{z} :

$$\hat{\mathbf{z}} = \arg \min_{\mathbf{z}} \sum_j \frac{1}{2} |c_j - z_j|^2 + \beta \sum_j \mathbb{I}_{\{z_j \neq 0\}}.$$

We call such optimization problems **separable**, because we can solve them one variable at a time.

Minimizing with respect to z_j requires simple (hard) thresholding:

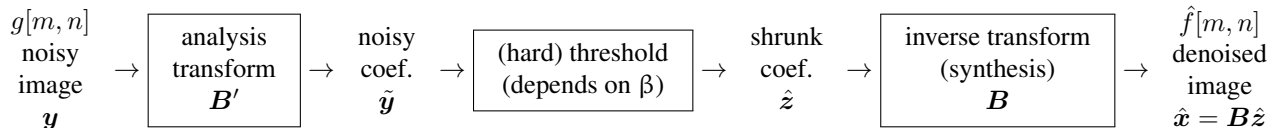
$$\hat{z}_j = \arg \min_{z_j} \frac{1}{2} |c_j - z_j|^2 + \beta \mathbb{I}_{\{z_j \neq 0\}} = \text{hard}(c_j, \beta),$$

where the **hard thresholding** function is defined by

$$\text{hard}(c_j, \beta) \triangleq \begin{cases} c_j & |c_j| \geq \sqrt{2\beta} \\ 0, & \text{otherwise.} \end{cases}$$

After thresholding / shrinking the transform coefficients, we synthesize the final denoised image using $\hat{\mathbf{x}} = \mathbf{B}\hat{\mathbf{z}}$.

In other words, wavelet-based **denoising** using sparsity of coefficients of an orthogonal transform (such as orthonormal wavelets) works as follows.



Soft thresholding with ℓ_1 norm

Using ℓ_1 norm instead of ℓ_0 in (13.1) leads to a **soft thresholding** approach [12] [13].

Find the solution to the following optimization problem:

$$\hat{z} = \arg \min_z \frac{1}{2} \|\tilde{y} - z\|_2^2 + \beta \|z\|_1.$$

$$\hat{z}_j = \arg \min_{z_j} \frac{1}{2} |c_j - z_j|^2 + \beta |z_j| = \text{sgn}(c_j) \max(|c_j| - \beta, 0).$$

A HW problem explores another regularizer based on $\|z\|_{3/2}$ that shrinks some coefficients but does not zero out any of them. Any $\|z\|_p$ for $0 \leq p \leq 1$ leads to some form of thresholding.

Summary

Sparse coding has been combined with MRF models [14].

Sparsity is a hot topic in signal/image processing and the models and methods are evolving continually.

Bibliography

- [1] J. A. Fessler. *Image reconstruction: Algorithms and analysis*. Book in preparation. ., 2006.
- [2] D. Donoho. “Scanning the technology”. In: *Proc. IEEE* 98.6 (June 2010), 910–912.
- [3] J. Wright, Y. Ma, J. Mairal, G. Sapiro, T. S. Huang, and S. Yan. “Sparse representation for computer vision and pattern recognition”. In: *Proc. IEEE* 98.6 (June 2010), 1031–1044.
- [4] R. Rubinstein, A. M. Bruckstein, and M. Elad. “Dictionaries for sparse representation modeling”. In: *Proc. IEEE* 98.6 (June 2010), 1045–57.

- [5] M. Elad, M. A. T. Figueiredo, and Y. Ma. “On the role of sparse and redundant representations in image processing”. In: *Proc. IEEE* 98.6 (June 2010), 972–982.
- [6] M. J. Fadili, J.-L. Starck, J. Bobin, and Y. Moudden. “Image decomposition and separation using sparse representations: an overview”. In: *Proc. IEEE* 98.6 (June 2010), 983–994.
- [7] R. G. Baraniuk, E. Candes, M. Elad, and Y. Ma. “Applications of sparse representation and compressive sensing [Scanning the issue]”. In: *Proc. IEEE* 98.6 (June 2010), 906–9.
- [8] J. A. Tropp and S. J. Wright. “Computational methods for sparse solution of linear inverse problems”. In: *Proc. IEEE* 98.6 (June 2010), 948–958.
- [9] L. Gao, J. Liang, C. Li, and L. V. Wang. “Single-shot compressed ultrafast photography at one hundred billion frames per second”. In: *Nature* 516.7529 (Dec. 2014), 74–7.
- [10] M. Lustig, D. Donoho, and J. M. Pauly. “Sparse MRI: The application of compressed sensing for rapid MR imaging”. In: *Mag. Res. Med.* 58.6 (Dec. 2007), 1182–95.
- [11] S. G. Mallat. “A theory for multiresolution signal decomposition: the wavelet representation”. In: *IEEE Trans. Patt. Anal. Mach. Int.* 11.7 (July 1989), 674–89.
- [12] D. L. Donoho. “De-noising by soft-thresholding”. In: *IEEE Trans. Info. Theory* 41.3 (May 1995), 613–27.
- [13] A. Antoniadis and J. Fan. “Regularization and wavelet approximations”. In: *J. Am. Stat. Assoc.* 96.455 (Sept. 2001), 939–55.
- [14] S. Roth and M. J. Black. “Fields of experts: a framework for learning image priors”. In: *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*. Vol. 2. 2005, 860–7.
- [15] Y. Huang, M. K. Ng, and Y.-W. Wen. “A fast total variation minimization method for image restoration”. In: *SIAM Multiscale Modeling and Simulation* 7.2 (2008), 774–95.
- [16] Y. Wang, J. Yang, W. Yin, and Y. Zhang. “A new alternating minimization algorithm for total variation image reconstruction”. In: *SIAM J. Imaging Sci.* 1.3 (2008), 248–72.
- [17] H. Bristow, A. Eriksson, and S. Lucey. “Fast convolutional sparse coding”. In: *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*. 2013, 391–8.

- [18] H. Bristow and S. Lucey. *Optimization methods for convolutional sparse coding*. arxiv 1406.2407. 2014.
- [19] J. Mairal, F. Bach, and J. Ponce. “Sparse modeling for image and vision processing”. In: *Found. & Trends in Comp. Graph. and Vision* 8.2-3 (2014), 85–283.

Chapter 14

Image Coding

Contents (class version)

| | |
|--|--------------|
| 14.0 Introduction | 14.2 |
| 14.1 Quantization | 14.4 |
| Scalar quantization | 14.5 |
| Quantization commands | 14.25 |
| Vector quantization | 14.27 |
| The K-means algorithm | 14.35 |
| Codeword assignment | 14.40 |
| 14.2 Waveform coding | 14.48 |
| Pulse code modulation | 14.48 |
| Pyramid coding | 14.55 |
| Analysis of the Laplacian pyramid | 14.62 |
| 14.3 Transform image coding | 14.68 |
| Karhunen-Loève Transform | 14.70 |
| Practical considerations in transform image coding | 14.82 |
| Reducing blocking effects | 14.83 |
| Adaptive coding schemes | 14.84 |
| JPEG coding | 14.86 |
| 14.4 Image model coding | 14.90 |

| | |
|---|--------------|
| 14.5 Interframe image coding | 14.91 |
| 14.6 Summary | 14.93 |

14.0 Introduction

This chapter introduces the basics of **image compression** or **image coding**.

Entire books are devoted to this problem, e.g., [1]. See also [2, Ch. 10] and [wallce:92:tjs].

Coding goal: represent an image with “as few bits as possible” (high compression ratio) while preserving the “quality” (meaning visual quality usually) required for the particular application, while keeping computation reasonable. There are trade-offs between:

- compression ratio (memory),
- image quality,
- complexity / computation.

Course goal: familiarity with the *elements* of image coding systems at a level suitable for reading the current literature.

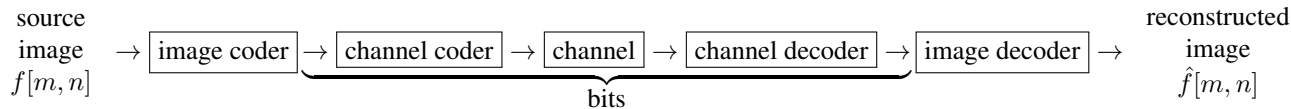
Relevant MATLAB commands: `dct2`, `idct2`, `dctdemo`, `blkproc`, and many more.

Overview

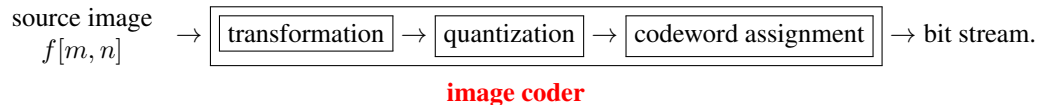
- Image coding methods fall into two categories
 - **lossless**: the recovered image $\hat{f}[m, n]$ exactly matches the original image $f[m, n]$
 - **lossy** aka “information preserving:” the recovered image $\hat{f}[m, n]$ does not exactly match the original image $f[m, n]$, but the “essential information” in $f[m, n]$ is retained.

Unfortunately, there is no widely accepted metric for “essential information.” Often researchers use simple measures of image fidelity, such as **peak signal-to-noise ratio (PSNR)**, as defined in (11.27) in Ch. 11.

- Typical image coding / decoding (“**codec**”) system:



- If the bits are stored on a reliable medium (like a hard disk?) then channel coding is unnecessary. (Actually modern disk drives have channel coding built into their firmware.)
- If the bits are transmitted over a noisy communication channel, *e.g.*, wirelessly, then channel coding may be essential.
- A good image coder will produce what appears to the channel coder to be a random bit sequence. In this type of scenario, the channel coding method can be designed independently of the source coding method. Channel coding is a separate topic considered in EECS 650. In this course we will focus on the image coding problem, which is a special case of the more general source coding problem covered in EECS 651.
- An image coder typically has the following form:



The transformation extracts image information prior to quantization. The transformation usually takes one of three forms.

- **Waveform Coder**: The intensity information itself is coded.
- **Transform Coder**: Some transform of the image, such as the DCT, is computed.
- **Model Coder**: A model of the image is formulated and parameters of the (typically nonlinear) model are estimated for a particular image. (This is analogous to LPC coding in speech processing.) The model parameters are passed to the quantizer.
- The steps of transformation, quantization, and codeword assignment are often closely linked.
- However, the general principles governing quantization and codeword assignment are separable from the choice of the image transformation. We first examine **quantization** and **codeword assignment** procedures independently before looking at different transformation procedures and complete image coding systems.

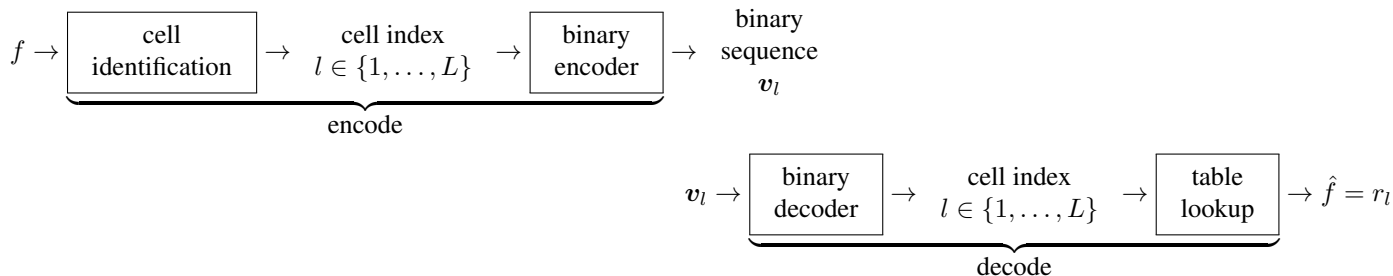
14.1 Quantization

The process of representing a real-valued image $f[m, n]$ (or some other set of real numbers such as transform coefficients or model parameters) by a finite set of bits is called **quantization**.

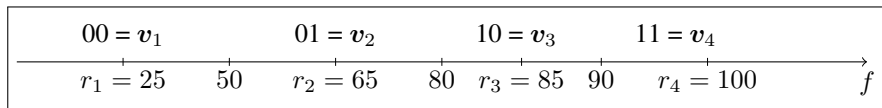
- If we quantize each value independently, then the procedure is called **scalar quantization**.
- If we group together multiple values and quantize them simultaneously, it is called **vector quantization** (as discussed later).

Memoryless scalar encoding / decoding

We wish to code a sequence of scalars $\{f_n\}$, e.g., image pixel values. For a **memoryless** scalar encoder, each scalar value is encoded (and decoded) individually.



Example. This case illustrates $L = 4$.



Choices:

- uniform vs nonuniform cells
- fixed-length vs variable-length binary sequences

Scalar quantization

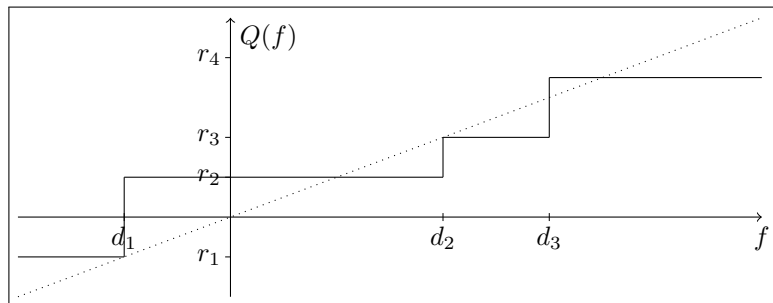
Let f denote a real-valued scalar quantity. We wish to represent f with a finite number of bits (m) resulting in one of $L = 2^m$ possible levels. Assigning a specific value of f to one of L levels is called **amplitude quantization**.

Let \hat{f} denote the quantized value of f . We focus on deterministic quantization strategies, where

$$\hat{f} = Q(f) \text{ and } Q : \mathbb{R} \rightarrow \{r_1, r_2, \dots, r_L\}.$$

We call Q the **quantization operator**. The $\{r_l\}$ values are called the **reconstruction levels**.

Example. The following figure shows a case where $L = 4$.



Preview of bit assignment

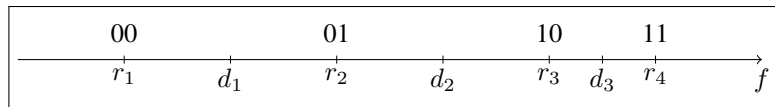
For $l = 1, \dots, L$, let \mathbf{v}_l denote the binary sequence (the **code**) assigned to reconstruction level r_l .
 Let the set of all such binary sequences be $\mathcal{V} = \{\mathbf{v}_1, \dots, \mathbf{v}_L\}$.

Mathematically:

- **encoder**: $\alpha : \mathbb{R} \rightarrow \mathcal{V}$, $\alpha(f) = \mathbf{v}_l$, $d_{l-1} < f \leq d_l$. The $\{d_l\}$ values are called **decision boundaries**.
- **decoder**: $\beta : \mathcal{V} \rightarrow \mathbb{R}$, $\beta(\mathbf{v}_l) = r_l$. (This is why the $\{r_l\}$ values are called **reconstruction levels**.)

Then **quantization** is the end-to-end process: $Q : \mathbb{R} \rightarrow \{r_1, \dots, r_L\}$, $Q(f) = \beta(\alpha(f))$.

The following diagram summarizes.



Design of a scalar quantizer

Any such quantization operation will always involve **quantization error** or **quantization noise**:

$$e_Q \triangleq \hat{f} - f = Q(f) - f \quad (14.1)$$

If we choose the $\{r_l\}$ values properly, then as the number of levels L increases, “on average” the quantization error should decrease.

Key questions:

- What is the “best” choice of the reconstruction levels $\{r_l\}$?
- How should we choose $Q(f)$, *i.e.*, how do we choose the regions or **cells** $\mathcal{C}_l = \{f : Q(f) = r_l\}$.
- How many levels L are required for a given “accuracy,” *i.e.*, how does error decrease versus L ?

In 1D one can argue easily that the \mathcal{C}_l regions should be contiguous intervals, so we generally form a quantizer Q as follows:

$$Q(f) = \begin{cases} r_1, & d_0 < f \leq d_1 \\ \vdots & \vdots \\ r_l, & d_{l-1} < f \leq d_l \\ \vdots & \vdots \\ r_L, & d_{L-1} < f \leq d_L, \end{cases}$$

where the $L + 1$ values $\{d_l\}$ are called **decision boundaries**. (It is possible to have $d_0 = -\infty$ and $d_L = \infty$.)

In other words, the **cells** are simply *intervals* in 1D (*i.e.*, scalar) quantization:

$$\mathcal{C}_l = (d_{l-1}, d_l].$$

Uniform scalar quantization

The simplest design of a scalar quantizer is to use *equally spaced* decision boundaries and to choose r_l to be the **midpoint** of each interval. This simple approach (which is rarely optimal in terms of quantization error) is called **uniform scalar quantization**:

$$d_l - d_{l-1} = \Delta, \quad r_l = \frac{d_l + d_{l-1}}{2}, \quad l = 1, \dots, L.$$

Although the form of (14.1) might suggest otherwise, in general the quantization error e_Q will be *signal dependent*. However, whether the quantization error is *correlated* or not with the signal depends on the specifics of the distribution of f and the quantization scheme.

Example. The following figures illustrate uniform scalar quantization for $L = 2^8, 2^7, \dots, 2^1$.

Original: using all 8 bits per pixel



255

0

PSNR: Inf dB

L=128 levels (7 bits)



254

0

PSNR: 51.12 dB

L=64 levels (6 bits)



PSNR: 46.40 dB

252

0

L=32 levels (5 bits)



PSNR: 40.74 dB

248

0

L=16 levels (4 bits)



240

PSNR: 34.79 dB

0

L=8 levels (3 bits)



224

PSNR: 29.38 dB

0

L=4 levels (2 bits)



192

0

PSNR: 23.40 dB

L=2 levels (1 bits)



128

0

PSNR: 16.12 dB

Design of reconstruction levels and decision boundaries

To design a quantizer that is better than uniform scalar quantization, we will need to design nonuniform reconstruction levels $\{r_l\}$ and decision boundaries $\{d_l\}$. As a general rule, to minimize quantization error, one should assign more reconstruction levels $\{r_l\}$ near the values of f that are *more likely*. To formulate objective criteria for choosing the “best” $\{r_l\}$ and $\{d_l\}$ values, we must choose a method to quantify the quantization error.

The first ingredient is a **distortion measure** $d(f, \hat{f})$. Typical choices are $|f - \hat{f}|$ or $|f - \hat{f}|^2$. To obtain a scalar cost function, we must somehow “average” over the possible values of f . To proceed, we *assume* that f is a random variable with known pdf $p(f)$. We then define the **average distortion** to be

$$D = \mathbb{E}[d(f, \hat{f})] = \int d(f, Q(f)) p(f) df.$$

If we choose the quadratic distortion measure $d(f, \hat{f}) = (f - \hat{f})^2$, then minimizing the average distortion is equivalent to a MMSE approach to quantizer design:

$$D = \mathbb{E}[|f - Q(f)|^2] = \int |f - Q(f)|^2 p(f) df = \sum_{l=1}^L \int_{d_{l-1}}^{d_l} (f - r_l)^2 p(f) df.$$

To minimize this average distortion, we can equate to zero the partial derivatives of D with respect to the design parameters (the $\{r_l\}$ and $\{d_l\}$ values). Specifically:

$$0 = \frac{\partial}{\partial r_l} D = \int_{d_{l-1}}^{d_l} 2(f - r_l) p(f) df = 2 \int_{d_{l-1}}^{d_l} f p(f) df - 2r_l \int_{d_{l-1}}^{d_l} p(f) df,$$

and by **Leibniz’s rule**:

$$0 = \frac{\partial}{\partial d_l} D = (d_l - r_l)^2 p(d_l) - (d_{l+1} - r_{l+1})^2 p(d_l), \quad l = 1, \dots, L - 1.$$

These equalities lead to the following set of coupled equations for MMSE scalar quantizer design:

$$\begin{aligned}
 r_l &= \frac{\int_{d_{l-1}}^{d_l} f p(f) df}{\int_{d_{l-1}}^{d_l} p(f) df} = \mathbb{E}[f | f \in (d_{l-1}, d_l]], \quad l = 1, \dots, L \text{ (weighted centroid)} \\
 d_l &= \frac{r_l + r_{l+1}}{2}, \quad l = 1, \dots, L - 1 \text{ (midpoint)} \\
 d_0 &= \inf \{f : p(f) > 0\} \\
 d_L &= \sup \{f : p(f) > 0\}
 \end{aligned} \tag{14.2}$$

Unfortunately, these equations are usually nonlinear and must be solved numerically. The **Lloyd-Max algorithm** updates $\{r_l\}$ and $\{d_l\}$ by alternating between the first two equations [3, 4].

Example. If f is uniformly distributed over $[a, b]$, then the integral for r_l in (14.2) becomes $r_l = \frac{1}{2} \frac{d_l^2 - d_{l-1}^2}{d_l - d_{l-1}} = \frac{1}{2}(d_l + d_{l-1})$, so the MMSE decision boundaries and reconstruction levels are given by $d_l = a + \frac{b-a}{L}l$ and $r_l = d_l - \frac{b-a}{2L} = a + \frac{b-a}{L}(l - \frac{1}{2})$ respectively, which corresponds to **uniform scalar quantization**.

Determine the RMSE in the preceding example.

(class/team)

??

Naturally, as L increases, the average distortion decreases.

★★★

Next we analyze this trade-off between the number of levels L and average distortion in more generality.

Basic “high-rate” rate-distortion analysis

If we use **fixed-length** binary sequences for a uniform scalar quantizer, then the **rate** (bits per encoded f value) is

$$R = \lceil \log_2 L \rceil .$$

If we encode blocks of N values simultaneously then we can approach the rate $R = \log_2 L$, because $\frac{\lceil \log_2 L^N \rceil}{N} \approx \log_2 L$. This requires the binary encoder / decoder to have memory, but still the quantizer is memoryless.

Suppose the distribution of f is supported on the interval $[a, b]$. If $\Delta = (b - a)/L$ is small, then for uniform scalar quantization we can use Riemann sums to approximate the average distortion:

$$\begin{aligned} D &= \int_a^b |f - Q(f)|^2 p(f) \, df = \sum_{l=1}^L \int_{d_{l-1}}^{d_l} (f - r_l)^2 p(f) \, df \approx \sum_{l=1}^L p(r_l) \int_{d_{l-1}}^{d_l} (f - r_l)^2 \, df = \sum_{l=1}^L p(r_l) \int_{-\Delta/2}^{\Delta/2} t^2 \, dt \\ &= \sum_{l=1}^L p(r_l) \frac{2}{3} \left(\frac{\Delta}{2} \right)^3 = \frac{\Delta^2}{12} \sum_{l=1}^L \Delta p(r_l) \approx \frac{\Delta^2}{12} \int p(f) \, df = \frac{\Delta^2}{12} = \frac{(b - a)^2}{12L^2} = \frac{(b - a)^2}{12(2^R)^2}. \end{aligned} \quad (14.3)$$

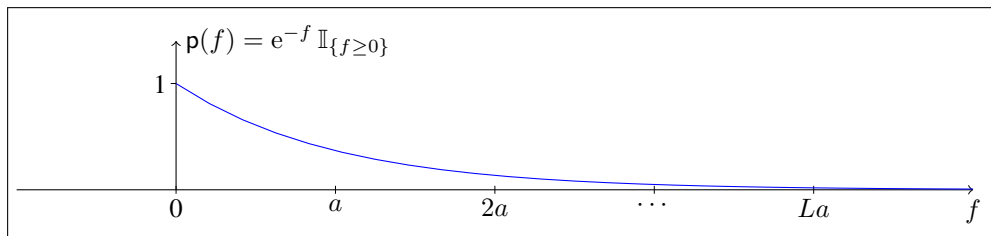
From this formulation for distortion, determine the approximate *PSNR gain per bit* where:

(class/pair)

$$\text{PSNR} = 10 \log_{10} \frac{f_{\max}^2}{D} \approx \boxed{??}$$

Example. The empirical PSNR values in the cameraman example shown previously agree remarkably well with this gain per bit!

Example. Suppose we apply uniform scalar quantization to variables having the exponential distribution $p(f) = e^{-f} \text{step}(f)$, using equally spaced levels $r_l = la$, $l = 1, \dots, L$. What is the MMSE spacing a ?



The average distortion is:

$$D = \int_0^{\frac{3}{2}a} (f-a)^2 e^{-f} df + \sum_{l=2}^{L-1} \int_{(l-1/2)a}^{(l+1/2)a} (f-la)^2 e^{-f} df + \int_{(L-1/2)a}^{\infty} (f-La)^2 e^{-f} df.$$

Using `int` and `diff` from MATLAB's symbolic toolbox yields:

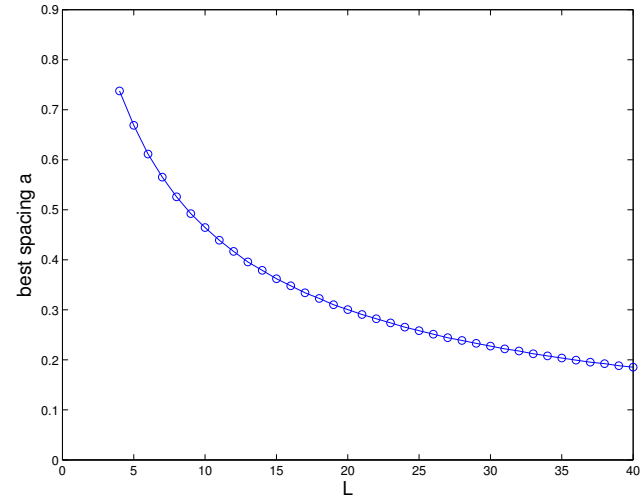
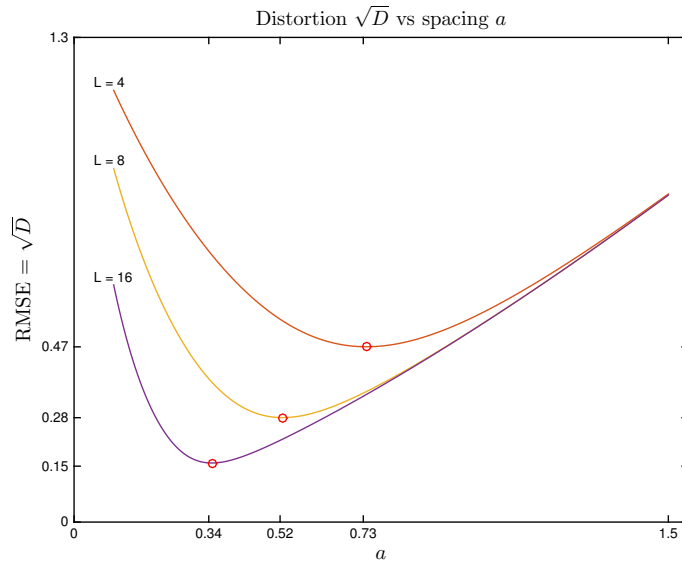
$$\begin{aligned} \frac{\partial}{\partial a} D = & e^{-a/2} \left(\frac{a^2}{8} - a + 2 \right) - 2 + 2a + \sum_{l=1}^{L-1} \left[\left(\frac{a^2}{4} + a + 2 \right) a e^{-(l+1/2)a} - \left(\frac{a^2}{4} - a + 2 \right) a e^{-(l-1/2)a} \right] \\ & + \left[\frac{a}{2} - 1 - \left(L - \frac{1}{2} \right) \left(\frac{a^2}{4} - a + 2 \right) \right] e^{-(L-1/2)a}. \end{aligned}$$

It is unlikely that we will find an analytical form for the minimizer. We can use `fmin` or `fsolve` to minimize D numerically.

The following figure shows the distortion D for this example as a function of the spacing a .

Why does D increase as $a \rightarrow 0$? Large quantization errors for large f values

Why does D increase as $a \rightarrow \infty$? Large quantization errors for the (most probable) small f values.



- Minimum distortion D decreases as L increases, as expected.
- Optimal spacing a does not halve as L doubles, unlike for uniform distribution. (Therefore redesign of the quantizer is needed for each L of interest.)

Example. As a consequence of the central limit theorem, Gaussian random variables arise in many problems (although not necessarily in image coding problems...). [2, Ch. 10] compares the MSE of uniform scalar quantization and the MSE of optimal (MMSE) scalar quantization for a Gaussian random variable. On average, the MMSE approach saves about 1/2 bits (per coded value) for 7 bit accuracy (128 levels). (The savings are more dramatic with vector quantization.)

See [5] for a thorough analysis of uniform scalar quantization.

Companding _____ (another scalar quantization method)

Because uniform scalar quantization is MSE-optimal for uniformly distributed random variables, a natural quantization approach is to first transform f to a uniformly distributed random variable, and then apply uniform scalar quantization to the result.

Illustration of companding approach:

$$f \rightarrow \boxed{T} \rightarrow g \rightarrow \boxed{\text{Uniform quantizer}} \rightarrow \hat{g} \rightarrow \boxed{T^{-1}} \rightarrow \hat{f}.$$

The two key questions are the following.

- What should the companding function $T(f)$ be?
- Is the resulting system optimal?

We have seen previously that if random variable X has cdf $F_X(x)$, then $Y = F_X(X)$ has a uniform distribution. Thus

$$g = \int_{-\infty}^f p(f') df' \triangleq T(f)$$

is uniformly distributed over $[0, 1]$. So the appropriate companding function is just the CDF of f (which we have assumed known so far but we might need to learn from training data).

Applying uniform scalar quantization to g will minimize

$$\mathbb{E}[(g - Q(g))^2] = \mathbb{E}[(T(f) - Q(T(f)))^2] \neq \mathbb{E}[(f - Q(f))^2],$$

so this **companding** approach is not MSE optimal in terms of f . However, the MMSE criterion is somewhat arbitrary anyway, and chosen primarily for its analytical convenience, so in practice minimizing $\mathbb{E}[(T(f) - Q(T(f)))^2]$ may produce reasonable coded images.

Often images are displayed using a certain type of transformation that we could interpret as a type of companding. What is it? [Gamma correction or histogram equalization.](#)

Example. Suppose f has the pdf $p(f) = 2/f^2$ for $f \in [1, 2]$, and we wish to assign L levels using **companding**. The required transformation for this random variable is

$$g = T(f) = \int_1^f 2/x^2 dx = 2 - 2/f, \quad f \in [1, 2].$$

The associated (equally spaced) reconstruction levels and decision boundaries (in terms of g) are

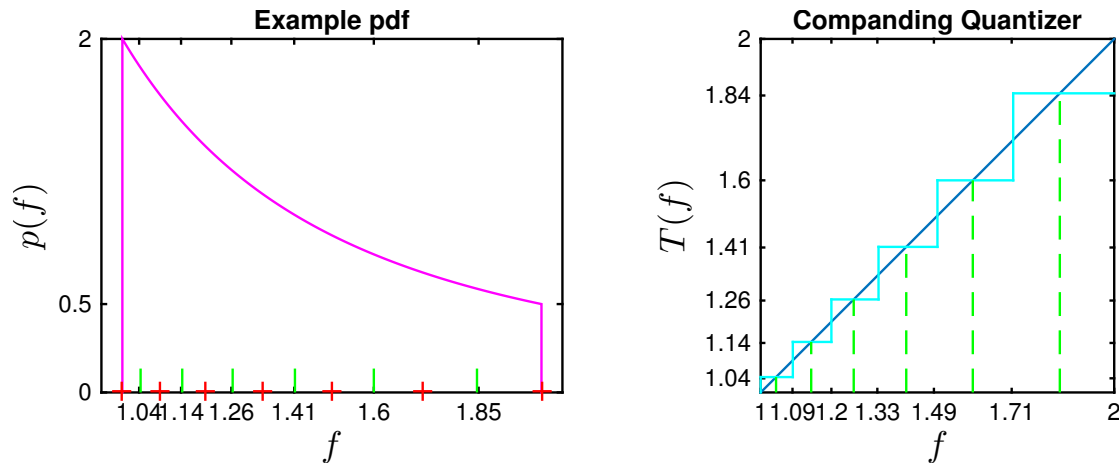
$$d_l^{(g)} = \frac{l}{L}, \quad l = 0, \dots, L, \quad r_l^{(g)} = \frac{l - 1/2}{L}, \quad l = 1, \dots, L. \quad (14.4)$$

Because $f = T^{-1}(g) = \frac{2}{2-g}$, the associated reconstruction levels and decision boundaries (in terms of f) are

$$d_l = \frac{2}{2 - l/L}, \quad l = 0, \dots, L, \quad r_l = T^{-1}(r_l^{(g)}) = T^{-1}\left(\frac{l - 1/2}{L}\right) = \frac{2}{2 - (l - 1/2)/L}, \quad l = 1, \dots, L.$$

The following figure shows the case $L = \boxed{??}$

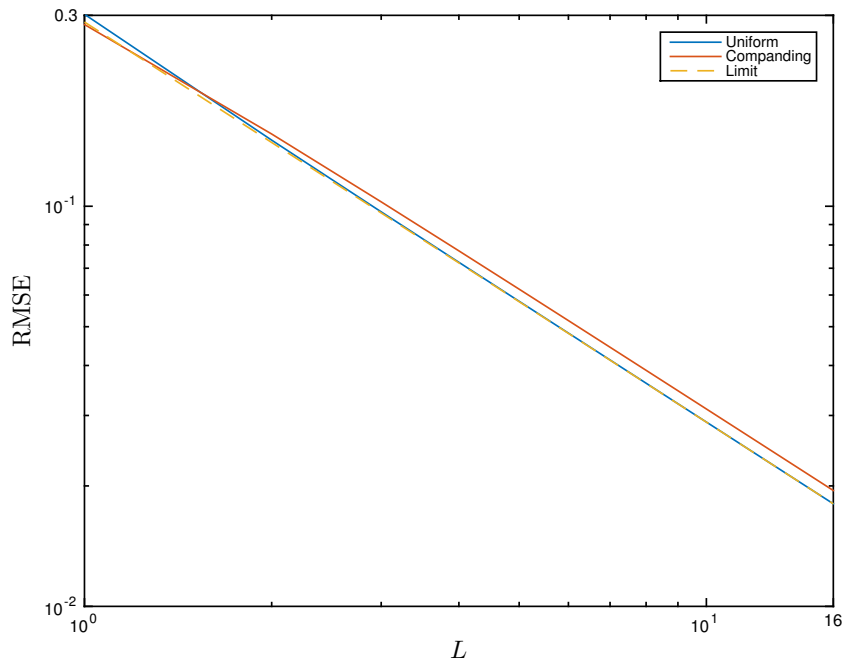
Why are the reconstruction levels farther apart for large f values? $\boxed{??}$



Continuing this example, the MSE is

$$\begin{aligned}
 \text{MSE} &= \sum_{l=1}^L \int_{d_{l-1}}^{d_l} (x - r_l)^2 \frac{2}{x^2} dx = \sum_{l=1}^L \int_{d_{l-1}}^{d_l} (x^2 - 2r_l x + r_l^2) \frac{2}{x^2} dx = 2 \sum_{l=1}^L \int_{d_{l-1}}^{d_l} \left(1 - 2\frac{r_l}{x} + \frac{r_l^2}{x^2} \right) dx \\
 &= 2 \sum_{l=1}^L \left((d_l - d_{l-1}) - 2r_l \log\left(\frac{d_l}{d_{l-1}}\right) - r_l^2 \left(\frac{1}{d_l} - \frac{1}{d_{l-1}}\right) \right) = 2 - 2 \sum_{l=1}^L \left(2r_l \log\left(\frac{d_l}{d_{l-1}}\right) + r_l^2 \left(\frac{1}{d_l} - \frac{1}{d_{l-1}}\right) \right).
 \end{aligned}$$

Evaluating this MSE expression for the case of uniform scalar quantization and for the compander design (14.4) leads to the following figure. Also shown is the “limit” $\frac{1}{12L^2}$ from (14.3). In this case companding is only slightly better than USQ for small values of L .



High-rate scalar quantization

As shown by Gersho [6], for k -dimensional quantization with distortion measured by $\mathbb{E} \left[\left\| \mathbf{f} - \hat{\mathbf{f}} \right\|_2^r \right]$, for high rate quantization, *i.e.*, for large values of L , the optimal **centroid density** is proportional to

$$p^{k/(k+r)}(\mathbf{f}).$$

In particular, in 1D ($k = 1$) using MSE ($r = 2$) distortion, the optimal centroid density is proportional to

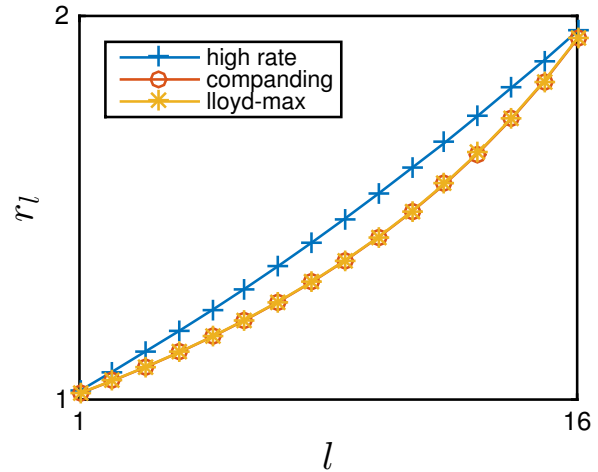
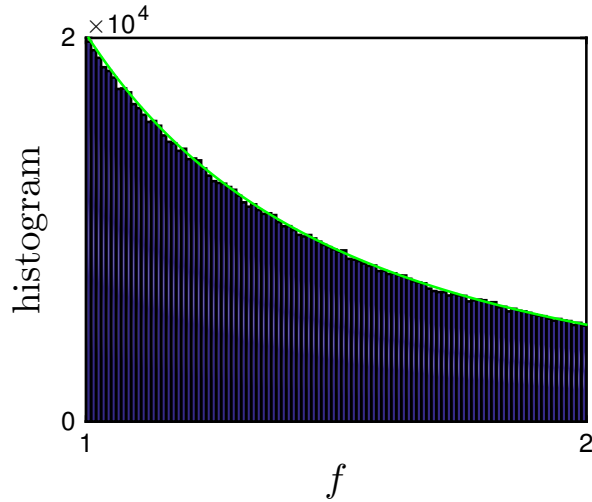
★★

$$p^{1/3}(\mathbf{f}).$$

Thus, for large L values, instead of solving for the $\{r_l\}$ values using the coupled system (14.2), if f is supported on $[a, b]$ then one can choose the centroids $\{r_l\}$ non-iteratively by using

$$r_l = G^{-1} \left(\frac{l - 1/2}{L} \right) \text{ where } G(t) \triangleq \frac{\int_a^t p^{1/3}(f) df}{\int_a^b p^{1/3}(f) df}.$$

Example. Continuing previous example, the following figures show that the high-rate $\{r_l\}$ values are “pretty close” to the Lloyd-Max optimized values.



Quantization commands

Some related (?) MATLAB commands.

| | |
|-----------------------|---|
| <code>quant</code> | Discretize values as multiples of a quantity. |
| <code>quantize</code> | An example M-File S-function vectorized quantizer |
| <code>lvq</code> | learning vector quantization. |
| <code>newlvq</code> | Create a learning vector quantization network. |
| <code>nnt2lvq</code> | Update NNT 2.0 learning vector quantization network to NNT 3.0. |
| <code>learnlvq</code> | Learning vector quantization rule. |
| <code>vmquant</code> | Variance Minimization Color Quantization. |
| <code>vmquantc</code> | Variance Minimization Color Quantization (MEX file). |

Bit allocation **(skim)**

If we choose $L_i = 2^{B_i}$ reconstruction levels for quantization the i th scalar f_i , then coding f_i alone would require B_i bits.

If we are coding N statistically identically distributed scalars f_1, \dots, f_N , then if we assign the B_i bits to variable f_i , then the total number of bits will be $B = \sum_{i=1}^N B_i$.

For waveform coding, using an equal number of bits for each pixel value may be reasonable. For transform coding, however, the different transform coefficients can have very different energies / variances (indeed the purpose of the transform is to compact most of the signal energy into a few coefficients). When the variances are nonuniform, we should also allocate the bits **nonuniformly** to the different coefficients.

This **bit allocation** problem is very important. If each f_i value has the same pdf *form* but with a different variance σ_i^2 , then an approximate solution to problem of allocating B total bits to N variables is given by

$$B_i = \frac{B}{N} + \frac{1}{2} \log_2 \frac{\sigma_i^2}{\left[\prod_{j=1}^N \sigma_j^2 \right]^{1/N}}.$$

- If the σ_i values are all identical, then $B_i = B/N$, which is uniform allocation.
- Otherwise, *the higher variance values are allocated more bits.*

Vector quantization

Rather than quantizing each scalar value individually, it is often advantageous to group scalars into “blocks” and jointly quantize the block. This type of approach is called **vector quantization (VQ)**.

Why consider VQ? **Because it can reduce average distortion for a given bit rate.**

Let $\mathbf{f} = (f_1, \dots, f_N)$ denote an N -dimensional vector consisting of N real-valued elements. In vector quantization, we map \mathbf{f} to one of L “**reconstruction levels**” \mathbf{r}_l , each of which is also an N -dimensional vector:

$$\hat{\mathbf{f}} = Q(\mathbf{f}) = \begin{cases} \mathbf{r}_1, & \mathbf{f} \in \mathcal{C}_1 \\ \vdots & \vdots \\ \mathbf{r}_l, & \mathbf{f} \in \mathcal{C}_l \\ \vdots & \vdots \\ \mathbf{r}_L, & \mathbf{f} \in \mathcal{C}_L, \end{cases} \quad \text{where } Q : \mathbb{R}^N \rightarrow \{\mathbf{r}_1, \dots, \mathbf{r}_L\},$$

where the set \mathcal{C}_l is called the l th **cell**, and we want the cells to partition \mathbb{R}^N , i.e., to be non-intersecting sets and $\bigcup_{l=1}^L \mathcal{C}_l = \mathbb{R}^N$.

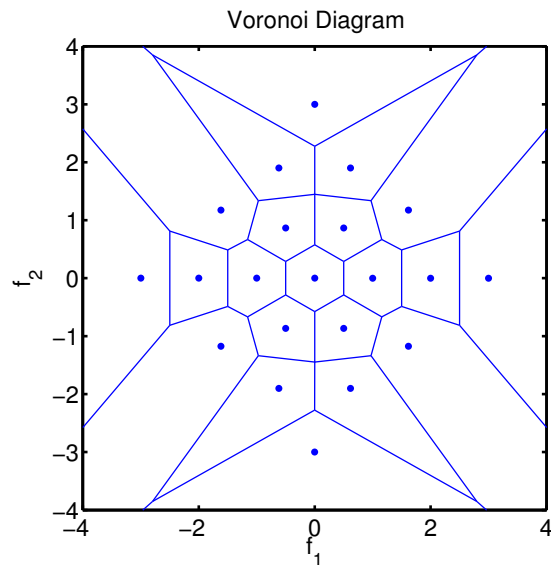
VQ Design

The same questions arise as in the scalar case, plus some additional ones.

- How many “levels” L are required to achieve a specified “accuracy.”
- What should we choose the reconstruction levels (aka centroids) $\{\mathbf{r}_l\}$?
- How should we choose the cells $\{\mathcal{C}_l\}$?
- What should the vector dimension N be?
- When does VQ outperform scalar quantization?

Graphical display of a quantizer Q is more complicated in VQ than in scalar quantization. The case $N = 2$ is about all that we can visualize easily. The MATLAB command `voronoi` computes the **Voronoi diagram** that, given the $\{r_l\}$ values (centroids), displays them as dots in the plane, and displays the boundaries of each C_l , assuming a **nearest neighbor** association rule. How is **nearest** defined in this diagram? [Euclidean distance](#).

Example. Here is a case with $L = 21$.
The dots correspond to r_1, \dots, r_L .



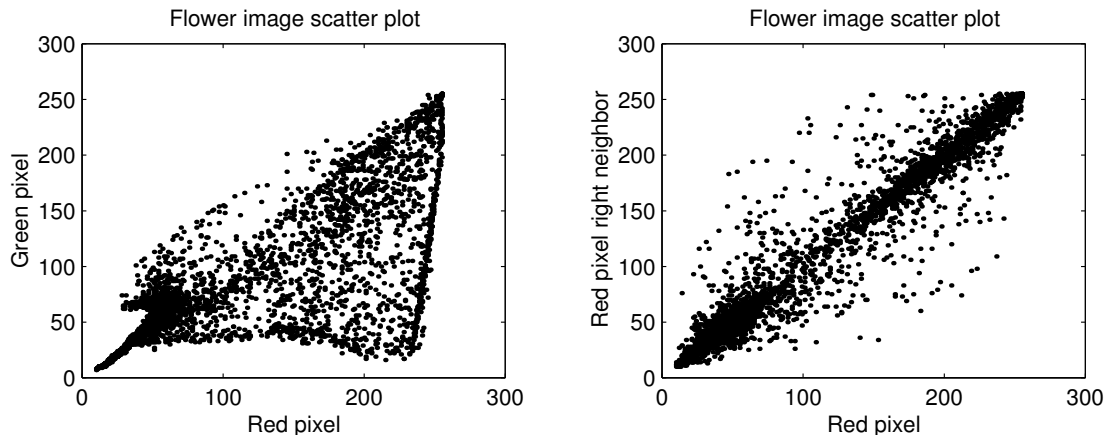
What does the corresponding picture for scalar quantization look like? ??

(class/pair)

The advantage of VQ is that it can exploit **statistical dependence** between elements of the vector \mathbf{f} to achieve either:

- lower average distortion than scalar quantization with the same number of levels, or
- fewer levels (better compression) for the same average distortion.

The following values were extracted from the `flower.tif` image in MATLAB's image processing toolbox.



Scalar quantization of each variable corresponds to a rectilinear or “plaid” decomposition of the 2D plane. Such a decomposition would needlessly allocate cells in the upper left hand corner. A VQ design can concentrate the cells where they are needed. The advantages of VQ can increase further as N increases beyond 2. On the other hand, quantizer complexity also increases somewhat as N increases, as does quantizer design.

To design a VQ, we again quantify the average distortion, which requires that we first define a distortion measure $d(\mathbf{f}, \hat{\mathbf{f}})$. Then, again assuming the \mathbf{f} is a **random vector** with known pdf $p(\mathbf{f})$, we can define the **average distortion** as follows:

$$D = E[d(\mathbf{f}, Q(\mathbf{f}))] = \int d(\mathbf{f}, Q(\mathbf{f})) p(\mathbf{f}) d\mathbf{f} = \sum_{i=1}^L \int_{\mathcal{C}_i} d(\mathbf{f}, \mathbf{r}_i) p(\mathbf{f}) d\mathbf{f}. \quad (14.5)$$

What is the dimension of the above distortion integrals for VQ? ??

[RQ]

For a given number of levels L , we would like to design the VQ by minimizing this average distortion with respect to the reconstruction levels $\{\mathbf{r}_l\}$ and the quantization cells $\{\mathcal{C}_l\}$.

Codebook design

The set $\{\mathbf{r}_1, \dots, \mathbf{r}_L\}$ is called a **codebook** because given the sequence of bits that describes an index $l \in \{1, \dots, L\}$, you “look up” the corresponding \mathbf{r}_l in the codebook to reconstruct the image (or its coefficients or model parameters).

Both the transmitter and the receiver must have a copy of the codebook through some “prior agreement” (e.g., it is built into the system design) or somehow transmitted separately.

The goal of VQ codebook design is to choose the \mathbf{r}_l values (and the \mathcal{C}_l sets) to minimize the average distortion D for a given class of images (i.e., for a given pdf $p(\mathbf{f})$).

Designing a VQ codebook is very challenging! (This problem alone has generated many dissertations.)

The following two conditions are *necessary* for a given design to be optimal.

Condition 1. For any input vector, the quantizer must choose the reconstruction level that minimizes the distortion:

$$Q(\mathbf{f}) = \mathbf{r}_l \text{ iff } d(\mathbf{f}, \mathbf{r}_l) \leq d(\mathbf{f}, \mathbf{r}_j) \text{ for } j = 1, \dots, L.$$

(The proof is easy; just consider the alternative.)

Therefore, cell \mathcal{C}_l consists of the values “closest” to \mathbf{r}_l :

How is “closest” quantified? **By distortion $d(\mathbf{f}, \mathbf{r}_l)$.**

$$\mathcal{C}_l = \{\mathbf{f} : d(\mathbf{f}, \mathbf{r}_l) \leq d(\mathbf{f}, \mathbf{r}_j) \text{ for } j = 1, \dots, L\}. \quad (14.6)$$

Condition 2. Each reconstruction level \mathbf{r}_l must minimize the average distortion in the corresponding cell \mathcal{C}_l :

$$\mathbf{r}_l = \arg \min_{\mathbf{r}} E[d(\mathbf{f}, \mathbf{r}) \mid \mathbf{f} \in \mathcal{C}_l]. \quad \text{“generalized centroid”}$$

Again, if this condition did not hold for a given supposedly optimal quantizer, we could easily construct a quantizer with lower average distortion, contradicting the purported optimality.

These two conditions suggest the initial design of an iterative procedure for determining the $\{r_l\}$ values and the $\{C_l\}$ sets given the pdf $p(\mathbf{f})$ and the desired number of levels L .

- If we somehow knew $\{r_l\}$, then in principle we could find each C_l using Condition 1 and (14.6).
- On the other hand, if we knew $\{C_l\}$, we could in principle find each r_l using Condition 2.

In particular, suppose the distortion measure is the quadratic function:

$$d(\mathbf{f}, \hat{\mathbf{f}}) = \|\mathbf{f} - \hat{\mathbf{f}}\|^2.$$

From estimation theory, the quantity that minimizes the expected squared error is the **conditional mean** (cf. (14.2)):

$$\arg \min_{\mathbf{r}} E[\|\mathbf{f} - \mathbf{r}\|^2 | \mathbf{f} \in C_l] = E[\mathbf{f} | \mathbf{f} \in C_l] = \frac{\int_{C_l} \mathbf{f} p(\mathbf{f}) d\mathbf{f}}{\int_{C_l} p(\mathbf{f}) d\mathbf{f}}. \quad (14.7)$$

In practice we know neither the $\{r_l\}$ values nor the $\{C_l\}$ sets, so we make an initial guess of the $\{r_l\}$ values and then iterate from there.

Example. We wish to find the MMSE quantization method (into L levels) for the 1D exponential distribution $p(f) = e^{-f} \text{step}(f)$. We can begin with an initial guess of $\{r_l\}$, say $r_l = l/2, l = 1, \dots, L$. The iterative algorithm then proceeds as follows.

- Given the current estimates of $\{r_l\}$, we evaluate (14.6) to update each \mathcal{C}_l . In this simple 1D case each \mathcal{C}_l is just an interval, and clearly the endpoints of those intervals are simply the midpoints between each of the r_l values as follows:

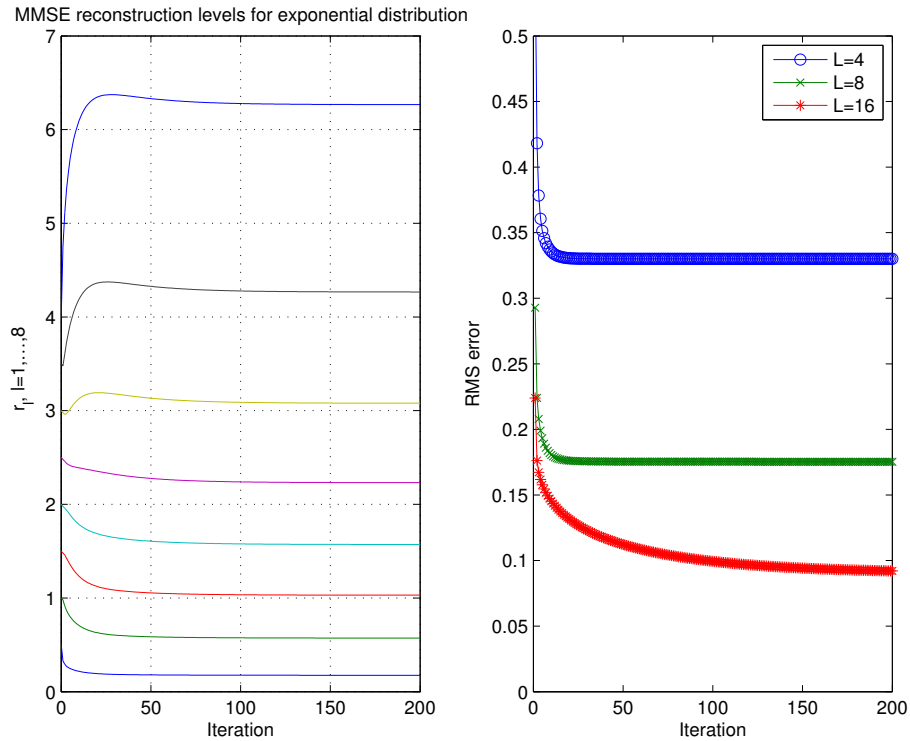
$$\mathcal{C}_l = \begin{cases} \left[0, \frac{r_1 + r_2}{2} \right], & l = 1 \\ \left[\frac{r_l + r_{l-1}}{2}, \frac{r_l + r_{l+1}}{2} \right], & l = 2, \dots, L-1 \\ \left[\frac{r_{L-1} + r_L}{2}, \infty \right], & l = L. \end{cases}$$

- Given these updated estimates of $\{\mathcal{C}_l\}$, we evaluate (14.7) to compute each new r_l . Each \mathcal{C}_l is an interval of the form $[a_l, b_l]$, where the a_l and b_l values depend on the previous r_l values. Thus

$$r_l^{\text{new}} = E[f | f \in \mathcal{C}_l] = \frac{\int_{\mathcal{C}_l} f p(f) df}{\int_{\mathcal{C}_l} p(f) df} = \frac{\int_{a_l}^{b_l} f e^{-f} df}{\int_{a_l}^{b_l} e^{-f} df} = \frac{(1 + a_l) e^{-a_l} - (1 + b_l) e^{-b_l}}{e^{-a_l} - e^{-b_l}}.$$

(This is a “textbook” type of example. Rarely will we get such a simple form.)

The following plot shows the r_l values as a function of iteration, starting from equally-spaced values. As the iterations proceed, the average distortion decreases towards a *local* minimum. There is no guarantee in general that the limiting r_l values produced by this method will be the *global* minimizers of the average distortion. To increase the chances of finding a global minimizer, one can initialize the algorithm multiple times with different initial guesses for the r_l values, and use the run that yields the lowest MSE.



Note that the algorithm puts a higher density of r_l values where the pdf $p(f)$ is larger.

The K-means algorithm

There is a serious practical problem with the above approach though. In practice, we rarely know $p(\mathbf{f})$; we just have **training data** consisting of images that are (hopefully) representative of the class of interest.

The **K-means algorithm** ($K = L$ here), discovered in the late 50's, is a practical solution to the problems mentioned above. It is also called the **LBG algorithm** after Linde, Buzo, and Gray who popularized it in the 1980's [8].

The K-means algorithm is `kmeans` in MATLAB. See also the related c-means method `fcm` and `fcmdemo`.

Why “means?” If $d(\cdot, \cdot)$ is the quadratic distortion measure, then the expectation in Condition 2 is the **conditional mean** (14.2).

If we have a sample $\{\mathbf{f}_j\}_{j=1}^M$ of training vectors (e.g., 2×2 blocks of pixels from an image), then the **maximum likelihood** estimate of the pdf $p(\mathbf{f})$ is given by:

$$\hat{p}(\mathbf{f}) = \frac{1}{M} \sum_{j=1}^M \delta_N(\mathbf{f} - \mathbf{f}_j),$$

where δ_N denotes the N -dimensional Dirac impulse. Substituting this estimator in for $p(\mathbf{f})$ into the condition mean expression (14.2) yields

$$\frac{\int_{C_l} \mathbf{f} \hat{p}(\mathbf{f}) d\mathbf{f}}{\int_{C_l} \hat{p}(\mathbf{f}) d\mathbf{f}} = \frac{\int_{C_l} \mathbf{f} \sum_{j=1}^M \delta_N(\mathbf{f} - \mathbf{f}_j) d\mathbf{f}}{\int_{C_l} \sum_{j=1}^M \delta_N(\mathbf{f} - \mathbf{f}_j) d\mathbf{f}} = \frac{\sum_{j: \mathbf{f}_j \in C_l} \mathbf{f}_j}{\sum_{j: \mathbf{f}_j \in C_l} 1} = \text{sample mean of the } \mathbf{f}_j \text{ values within } C_l.$$

This is an intuitive choice for \mathbf{r}_l : simply the empirical average of the \mathbf{f}_j vectors within cell C_l .

Replacing (14.2) with the above empirical average in the iterative algorithm described above yields the following practical algorithm, called the **K-means algorithm**, also called the **clustering algorithm** in the pattern recognition literature.

- Begin with an initial guess of the centroids $\{\mathbf{r}_l \in \mathbb{R}^N : l = 1, \dots, L\}$.
- Let \mathcal{C}_l be the subset of all vectors $\{\mathbf{f}_j\}_{j=1}^M$ that are closer to \mathbf{r}_l than to any of the other centroids, for $l = 1, \dots, L$.

This step is expensive because it requires many operations. How many? ?? (class/pair)

- Let $\mathbf{r}_l^{\text{new}}$ be the empirical average of the vectors \mathbf{f}_j in \mathcal{C}_l .

How much computation? ?? (class/pair)

- Repeat the previous two steps...

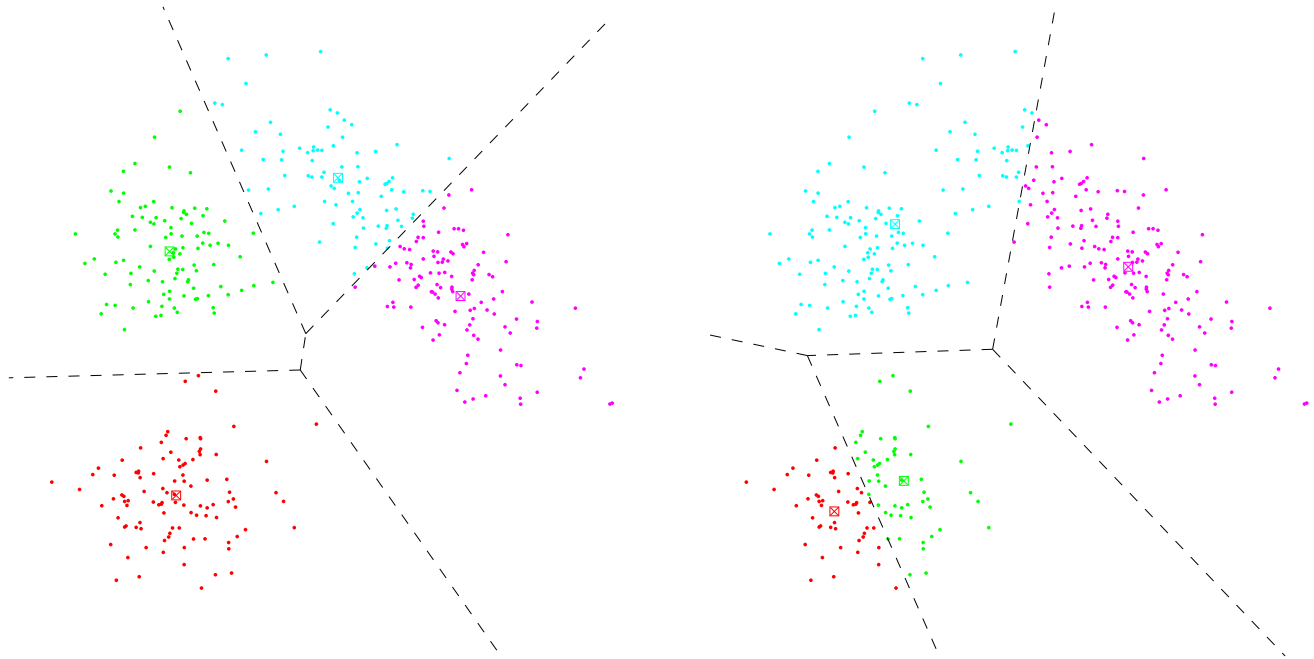
LBG showed that this iterative algorithm converges to a local minimum of the *empirical* average distortion D , defined by:

$$D = E_{\hat{\mathbf{p}}}[d(\mathbf{f}, Q(\mathbf{f}))] = \int d(\mathbf{f}, Q(\mathbf{f})) \hat{\mathbf{p}}(\mathbf{f}) d\mathbf{f} = \frac{1}{M} \sum_{j=1}^M \|\mathbf{f}_j - Q(\mathbf{f}_j)\|^2.$$

(See also [9].) Again, running with multiple initial guesses can reduce D at the price of increased computation. Careful initialization can help, using the **K-means++** algorithm [10, 11].

This cost function is non-convex, and convex relaxations have been proposed [12].

Example. demo_kmean.m



Use of VQ codebooks

- Transmission (coding): compare a given \mathbf{f} to all of the centroids $\{\mathbf{r}_l\}$, finding the \mathbf{r}_l that is the closest. Transmit bits that describe the index l .
The “compare” part of the coding operation is called a **full search** and is very computationally expensive! (Computations grow exponentially with the vector dimension N .)
- Decoding: look up in the codebook the \mathbf{r}_l associated with the received bits describing the index l .

To reduce coding computation, one can design the codebook $\{\mathbf{r}_l\}$ using certain constraints, such as requiring a **tree codebook**, which facilitates a **binary search** rather than a **full search** for finding the closest \mathbf{r}_l .

What will be the trade-off of constraining the structure of the codebook $\{\mathbf{r}_l\}$ in the interest of saving computation?
[Higher average distortion \$D\$ for a given bit rate.](#)

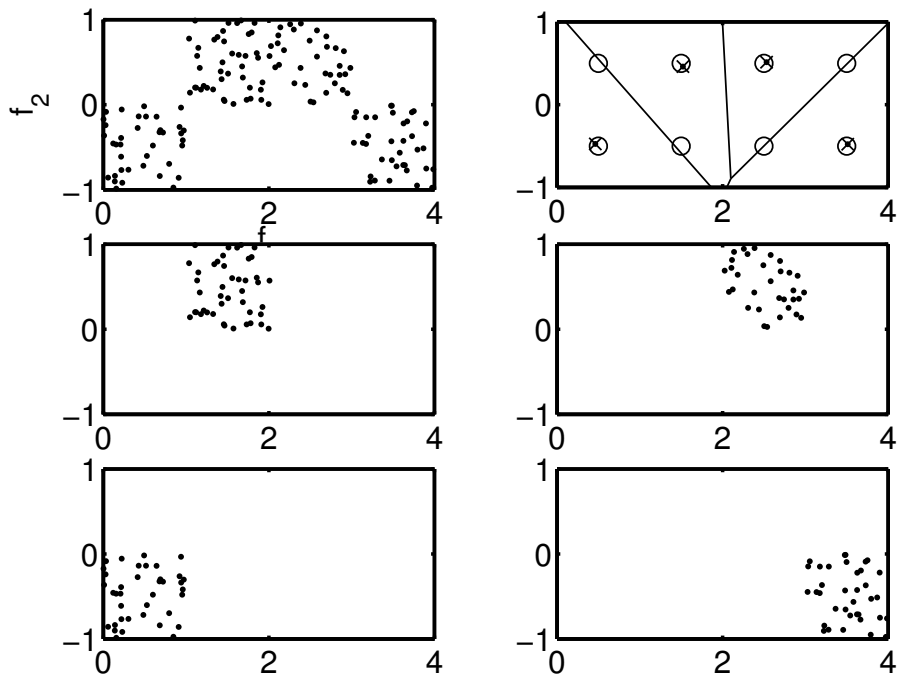
Summary of VQ

- Better performance (lower average distortion D) than scalar quantization for same bit rate.
- Greater complexity and computation.
- Appropriate for broadcast applications with one expensive central transmitter and many simpler receivers.

The principles here are useful not only for vector quantization, but also for **clustering** of data.

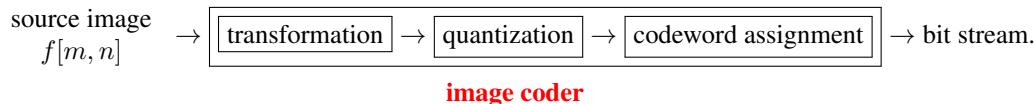
Instead of l_2 error, one can also use l_∞ error [13], among other options.

Example. Here is a (synthetic) distribution of vectors (with $N = 2$) for which vector quantization is significantly more efficient than uniform scalar quantization. The circles are the 8 centroids for scalar quantization whereas the x's are the 4 centroids needed for VQ, with exactly the same quantization error.



Codeword assignment**(skim)**

Recall:



After the quantizer has represented parameters of the image in terms of L possible levels, we must assign each level a specific **codeword** (*i.e.*, a sequence of 0s and 1s). The receiver in an image coding system inverts this relationship: it takes each received codeword and from it identifies the corresponding reconstruction level r_l .

For the receiver to be able to uniquely identify a reconstruction level, we must assign each level $\{1, \dots, L\}$ to a different codeword. The transmitter sends all of an image's codewords as a long sequence, so the codewords must be uniquely identifiable when received as a long sequence. Such a code is called **uniquely decodable**.

Some simple codes are not uniquely decodable.

Example. Consider the case where $L = 4$. The obvious natural codeword assignment is simply

$$\begin{bmatrix} r_1 \rightarrow 00 \\ r_2 \rightarrow 01 \\ r_3 \rightarrow 10 \\ r_4 \rightarrow 11 \end{bmatrix}$$

This obvious type of codeword assignment is called **uniform-length codeword assignment**. This choice guarantees that the sequence will be uniquely decodable.

When would you want to use anything other than uniform-length codeword assignment?

When the relative frequency of occurrence of the levels is nonuniform.

If r_1 and r_2 occur more frequently in the class of images of interest, then one might consider trying to assign fewer bits to the more likely centroids using the following codeword assignment:

$$\begin{bmatrix} r_1 \rightarrow 0 \\ r_2 \rightarrow 1 \\ r_3 \rightarrow 10 \\ r_4 \rightarrow 11 \end{bmatrix}.$$

Unfortunately, this approach is not **uniquely decodable**. If we receive the sequence 010, we cannot tell whether it corresponds to r_1, r_3 or r_1, r_2, r_1 . One way to fix this problem is to use a codeword assignment like the following:

$$\begin{bmatrix} r_1 \rightarrow 0 \\ r_2 \rightarrow 10 \\ r_3 \rightarrow 110 \\ r_4 \rightarrow 111 \end{bmatrix}.$$

This is called a **prefix code** because no codeword is a prefix of any other codeword.

The **average bit rate** of a coding scheme is given by

$$\bar{b} = \sum_{l=1}^L p_l b_l,$$

where p_l is the probability of occurrence of the of the l th reconstruction level (aka message), and b_l is the number of bits assigned to the l th message. For uniform codeword assignment, $b = \lceil \log_2 L \rceil$, so $\bar{b} = b$ because $\sum_{l=1}^L p_l = 1$.

Variable-length codeword assignment _____ **(skim)**

The latter example above uses **variable-length codeword assignment**, which can be based on the relative probabilities (the p_l values) of the L “messages.” If the p_l values are nonuniform, then we can usually reduce the average bit rate to be lower than $\lceil \log_2 L \rceil$ by using shorter codewords for the more probable messages.

What is the lowest possible average bit rate? Information theory shows that the average bit rate is never lower than the **entropy** of the messages, defined by:

$$H = - \sum_{l=1}^L p_l \log_2 p_l .$$

It can be shown that

$$0 \leq H \leq \log_2 L \text{ and } \bar{b} \geq H. \quad (14.8)$$

When is the entropy equal to $\log_2 L$? **When the messages are equally likely.**

When is the entropy equal to 0? **When one of the messages occurs with probability 1.**

From (14.8) we can conclude that **uniform-length codeword assignment** is optimal for $L = 2^{\bar{b}}$ equally probable messages.

Which scalar quantization method always yields equally probable reconstruction levels? **The companding approach.**

To assign codewords in a way that would achieve the lower limit on average bit rate, ideally we would use

$$b_l = -\log_2 p_l$$

bits for the l th message, so that the average bit rate would be

$$\bar{b} = \sum_l p_l b_l = \sum_l p_l (-\log_2 p_l) = H.$$

Unfortunately, however, $-\log_2 p_l$ is usually *not* an integer, so we must do some “rounding” which will result in a code that will not quite minimize the average bit rate. (This problem can be mitigated by grouping multiple messages together into a block and assigning codewords to each possible block.)

Huffman coding _____ (skim)

What is the lowest possible average bit rate if we constrain ourselves to a **uniquely decodable** code that (of course) must have an integer number of bits per message?

Although there may not be a simple analytical expression for this constrained minimizer, the **Huffman coding** procedure gives us an *algorithm* for designing the code (given the probabilities p_l) that is guaranteed to be optimal in that constrained sense. The

following figure (from [2]) illustrates the approach:

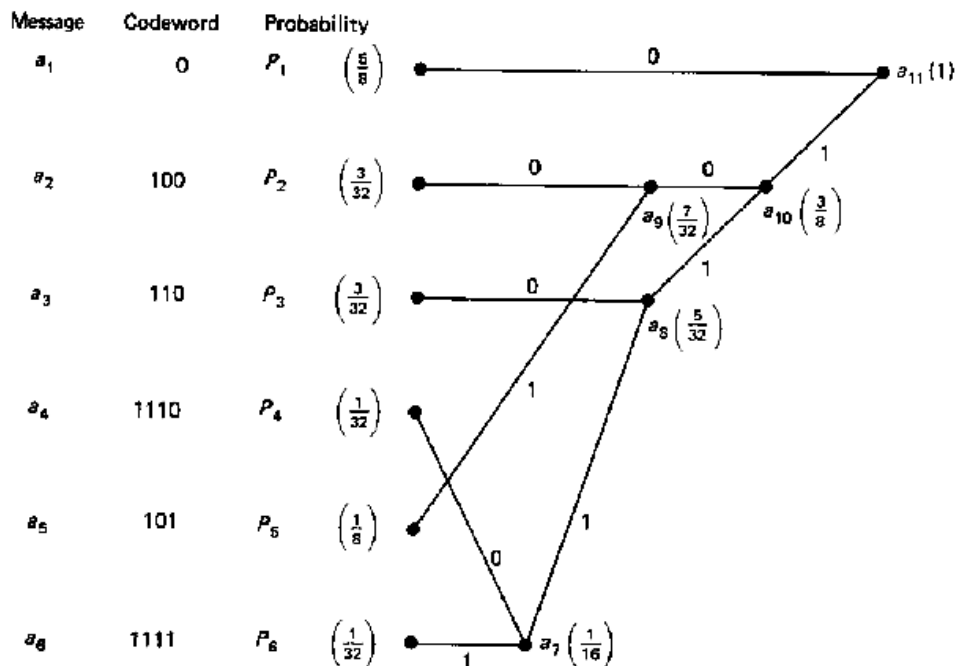


Figure 10.16 Illustration of codeword generation in Huffman coding. Message possibilities with higher probabilities are assigned with shorter codewords.

Algorithm:

- Start with the two messages with the lowest probability (a_4 and a_6).
- Assign a single bit to distinguish these messages from each other and combine the messages to form a new message a_7 .
- Based on the new “reduced” set of messages (a_1, a_2, a_3, a_5, a_7), assign a single bit to distinguish two messages with the lowest probability (a_3, a_7).
- New bits are added to the left of previously assigned bits.
- Continue this procedure until the most probable level is assigned.

The final code word assignment is shown above.

For this example, the entropy is

$$H = - \left[\frac{5}{8} \log_2 \frac{5}{8} + \frac{3}{32} \log_2 \frac{3}{32} + \cdots + \frac{1}{32} \log_2 \frac{1}{32} \right] \approx 1.752 \text{ bits / message,}$$

where as the average bit rate of the Huffman code is

$$\bar{b} = \frac{5}{8}1 + \frac{3}{32}3 + \cdots + \frac{1}{32}4 \approx 1.813 \text{ bits / message.}$$

What would the average bit rate be for uniform-length coding? 3

(But this example is a bit unfair because 6 is not a power of 2.)

Because $\log_2 6 \approx 2.585$, Huffman coding is still better than what would be achieved by uniform-length coding of long blocks.

A disadvantage of variable-length coding is that the **instantaneous bit rate** differs from the **average bit rate**, so buffering is required at the receiver, which adds complexity. This can be a concern in real-time image communication systems, but not a significant issue when coding for image archiving to storage.

By grouping multiple messages into a block, one can design variable-length codes whose average bit rate is as close as desired to the lower limit specified by the entropy. In practice the probabilities of the reconstruction levels are not exactly known anyway,

but must be approximated by a training image set, so designing super-complicated codes to approach a theoretical entropy limit is probably fruitless.

Joint optimization of quantization and codeword assignment _____ (skip)

So far we have considered *separately* the issues of quantizer design and codeword assignment. In reality, these issues are coupled.

Example. If one intends to use uniform-length codes to avoid buffering issues, then the only logical choice for L is $L = 2^{\bar{b}}$, where \bar{b} is the specified average bit rate. Using any L that is not a power of two would be pointless for uniform-length codes (unless block coding is applied of course).

- A uniform quantization method will usually yield *nonuniform* probabilities, so **variable-length codeword assignment** may significantly reduce the average bit rate.
- A nonuniform quantization method (companding in particular) may result in approximately equal probabilities, in which case **uniform-length codeword assignment** is likely to be adequate.

However, in our consideration of quantizer design, we tried to minimize the **average distortion** D subject to a given number of **reconstruction levels** L . In practice, it would often be preferable to minimize the average distortion *subject to a given constraint on the average bit rate* \bar{b} . (Depending on the probability distributions, different L values could yield the same \bar{b} with **variable-length codeword assignment**.)

This problem is called **joint optimization** of the **quantizer** and the **codeword assignment**. However, joint optimization is a challenging nonlinear problem. See [14] for joint consideration of bit allocation and coefficient subset selection.

Another interesting challenge is **entropy-constrained quantizer design** [15, 16].

Overall, the goal is to use the minimum number of bits to code an image with “sufficiently low” average distortion for a given application.

- In practical systems, most of the burden is usually put on either the quantization stage or the codeword assignment stage. For example, a nonuniform quantizer is used so that each reconstruction level has the same probability (*i.e.*, all messages are approximately equally probable). A simple uniform coder can be used for this system.
- In contrast, for the same level of overall distortion at the same bit rate, we could use a uniform quantizer with many more levels, and then use variable-length codewords to compensate for the fact that uniform quantization will yield unequally likely messages for any distribution of values other than uniform.
- Often practical considerations such as buffering size, computational requirements, etc., must be factored into any design and optimization procedures.

14.2 Waveform coding

Now that we have summarized quantization and codeword assignment, we return to the issue of what image variables are actually quantized and coded.

In this section we consider **waveform coding**, where the image pixel values themselves are coded. We consider **transform coding** and **model coding** in subsequent sections.

Pulse code modulation

The simplest waveform coding method is **pulse code modulation (PCM)**, which is just another name for **uniform scalar quantization** of the pixel values.

Typical image acquisition devices like CCD sensors are essentially PCM systems because the A/D converter maps continuous-valued electrical signals (corresponding to image intensities at each pixel) into a finite set of levels, *e.g.*, $L = 2^8$ for an 8-bit A/D converter.

8 bits/pixel is typical for most monochrome applications, although 12 bits/pixel is also common for medical image applications. The figures on page 14.11 illustrate the **contouring effects** of quantizing to small number of bits/pixels.

Using **nonuniform scalar quantization** can improve PCM system performance, in the spirit of **companding** as discussed earlier. Logarithmic transforms are one “common” choice.

Example. Given an 8-bit image, how does one apply uniform scalar quantization to yield an image with only $2^6 = 64$ levels? Do you just divide by 4 and round down?

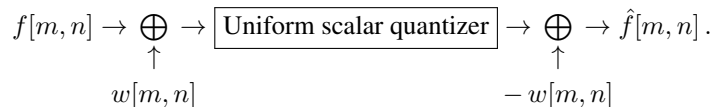
Not exactly, because then 0 through 3 would all get mapped to 0, which is a suboptimal choice for the reconstruction level. Instead we want the reconstruction level to be either 1 or 1.5 or 2, typically 2 if we assume the original 8-bit quantized image was defined by mapping $[0,1)$ to 0, $[1,2)$ to 1, etc.

So the decoder divides by 4 and rounds down to form the bits representing the reconstruction levels, but the decoder multiplies by 4 and adds 2, so the actual reconstruction levels are 2, 6, 10, ... For example, if the input value is $141 = 10001101_2$ then the 6-bit encoding is 100011 and the reconstructed value is $10001100_2 + 2 = 140 + 2 = 142$.

Robert's pseudonoise technique

Quantization noise is signal dependent, as discussed earlier.

To reduce the contouring effects of quantization noise, one can apply **Robert's pseudonoise technique** [17], called **dither**, making the noise more independent of the signal. The method has the following block diagram.



The white noise $w[m, n]$ should have a uniform distribution over $[-\Delta/2, \Delta/2]$, where Δ is the quantization interval.

Basically, we add noise before quantization and then attempt to subtract it back out after quantization. Because quantization is nonlinear, this subtraction does not work perfectly, and the resulting $\hat{f}[m, n]$ will certainly have errors. But the visual characteristics of those errors may be less objectionable than the contouring effects.

Both the encoder and the decoder “must” share identical copies of $w[m, n]$. (More practically, they must share a common pseudo-random method for generating $w[m, n]$.)

One can apply noise reduction methods to the output image to reduce the resulting additive noise because it is approximately signal independent.

(This idea does not easily apply to the CCD situation where the input image is actually analog.)

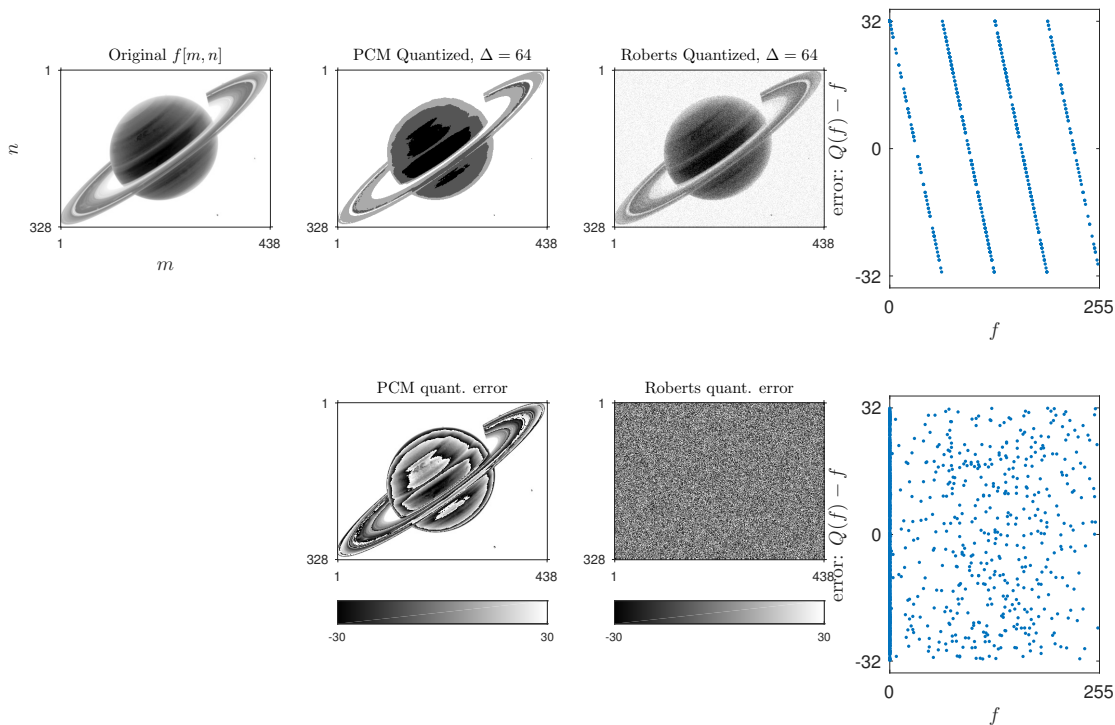
Example.

The plots on the following page illustrate applying Robert's pseudonoise technique to the Saturn image. The quantization error scatter plots ($\hat{f} - f$ vs f) suggest that the quantization noise is approximately independent of the signal f .

The quantized image is only 2 bits per pixel. With 4 bits per pixel (or more) it looks quite reasonable, because the eye is good at

averaging over the local signal-independent noise.

However, note the undesirable noise in the otherwise uniform background region. The marginal distribution of the error is essentially uniform $[-\Delta/2, \Delta/2]$ in both cases.



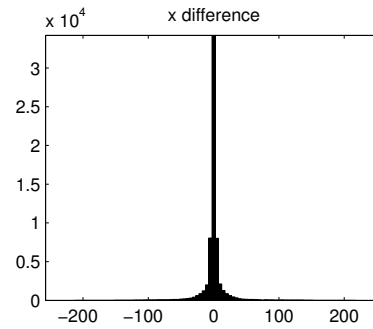
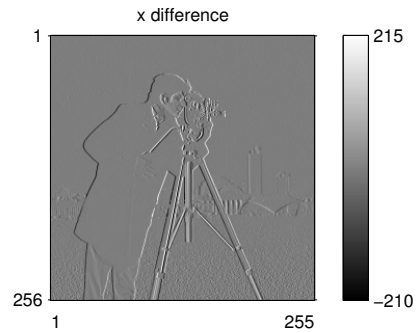
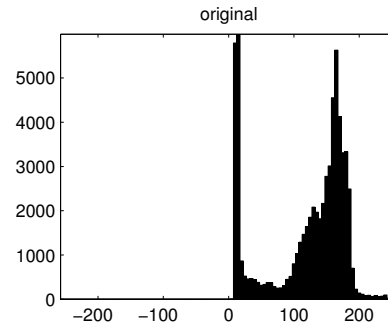
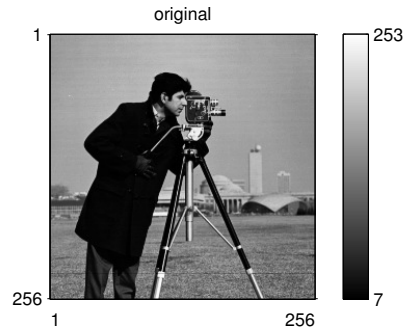
Delta modulation

The PCM quantization method completely ignores the presence of correlation (*cf.* flower scatter plot shown earlier) between neighboring pixel values, because each pixel is coded independently.

One way to exploit correlation, while still using scalar quantization, is to use **delta modulation (DM)**, which codes the *difference* between neighboring pixel values.

In fact, it is even possible (and often the case) that one can use a single bit for quantizing the difference. (A one bit quantizer is particularly easy to implement.)

Example. The following figure compares the histogram of an image and of the horizontal differences of that image.



Analysis of delta modulation

For Gaussian random variables, as well as many others, entropy is a monotonically increasing function of variance. So methods that reduce variance will reduce entropy and thus improve the rate-distortion trade-off.

Let $x[n]$ denote a WSS random process with autocorrelation function $R_x[n]$. Now define the process $y[n] = x[n] - x[n - 1]$ based on difference between neighboring sample values. When is the variance of $y[n]$ smaller than that of $x[n]$?

$$R_y[0] = E[|y[n]|^2] = E[(x[n] - x[n - 1])(x[n] - x[n - 1])^*] = 2R_x[0] - 2\text{real}\{R_x[1]\},$$

so

$$\text{Var}\{y[n]\} = \text{Var}\{x[n]\} 2(1 - \rho_1) \text{ where } \rho_1 = \frac{\text{real}\{R_x[1]\}}{\text{Var}\{x[n]\}}.$$

So $\text{Var}\{y[n]\} < \text{Var}\{x[n]\}$ if and only if $\rho_1 > 1/2$, *i.e.*, if there is sufficient positive correlation between neighboring signal values. This condition often holds.

Other variations (skip)

Another approach is **differential PCM** that is like DM but uses multiple bits to code the error, and uses an autoregressive prediction filter.

Another approach is **two-channel coders** that separately code the low-frequency and high-frequency components, using different bit-rates and tolerable distortions for each channel.

Pyramid coding**(skim)**

This is another method for exploiting correlation between pixels, both neighboring and farther away.

The basic idea is that if we appropriately down-sample an image and then up-sample, we should get the original image back except for some fine details. These ideas are historical precursors to **wavelets**.

In **pyramid coding** the image is viewed as having several different levels of resolution. Only the differences between the different resolution levels are transmitted. Also, at each level of resolution the image is subsampled producing an image with smaller dimensions. This type of processing is a classic example of **sub-band coding**.

Pyramid coding is based on successive subsampling. Subsampling is performed by first low pass filtering the image:

$$f_0^L[m, n] \triangleq h_L[m, n] ** f_0[m, n].$$

Following filtering, the image is decimated at a 2:1 ratio, *i.e.*,

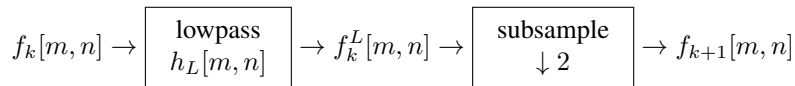
$$f_1[m, n] \triangleq f_{0\downarrow 2}^L[m, n] = \begin{cases} f_0^L[2m, 2n], & 0 \leq m \leq 2^{K-1}, 0 \leq n \leq 2^{K-1} \\ 0, & \text{otherwise.} \end{cases}$$

We assume $M = N = 2^K + 1$ and we use an odd number of points for each subimage so that the filter $h_L[m, n]$ is symmetric about the subsampled points, and $f_K[m, n]$ consists of a single point in the sub-band image.

The image $f_0[m, n]$ is called the **base of the pyramid**, and $f_1[m, n]$ is called the first-level image of the pyramid.

If $f_0[m, n]$ has $(2^K + 1) \times (2^K + 1)$ pixels, then $f_1[m, n]$ has $(2^{K-1} + 1) \times (2^{K-1} + 1)$ pixels.

We can repeat this process as illustrated below



The k th image in the pyramid has $(2^{K-k} + 1) \times (2^{K-k} + 1)$ pixels. If $N = M = 2^K + 1$, then the K th image in the pyramid is just one pixel.

There are a variety of possible choices for $h_L[m, n]$, each leading to a different “type” of pyramid.

The Gaussian and Laplacian pyramid representations

The so-called **Gaussian pyramid** is based on the zero-phase separable choice $h_L[m, n] = h[m] h[n]$, where

$$h[n] = \left[\left(\frac{1}{4} - \frac{a}{2} \right) \quad \frac{1}{4} \quad a \quad \frac{1}{4} \quad \left(\frac{1}{4} - \frac{a}{2} \right) \right],$$

and a is a free parameter. For $a = 0.4$, the filter looks like $[0.05 \ 0.25 \ \underline{0.4} \ 0.25 \ 0.05]$, roughly matching samples of a Gaussian function, hence the name.

Earlier in the course we saw that we can approximately recover a downsampled image by using interpolation-based up-sampling. Such interpolation will not *exactly* recover the original image, but the differences will be small and somewhat randomly distributed. The basic idea behind of pyramid coding (for a single level) is to code the low resolution version, and the **difference image** between the original image (at that level) and its prediction based on interpolation from the coarser level.

Let I denote the interpolation operator (linear, or not, as long as the coder and decoder agree!). The error between an image $f_k[m, n]$ and its prediction is given as follows:

$$e_k[m, n] \triangleq f_k[m, n] - I\{f_{k+1}[m, n]\}.$$

If there were no quantization applied, we could recover $f_k[m, n]$ exactly at any stage by

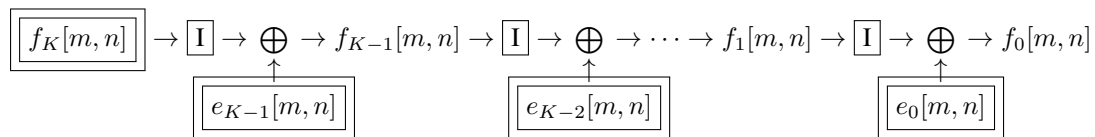
$$f_k[m, n] = \underbrace{I\{f_{k+1}[m, n]\} + e_k[m, n]}.$$

(Low resolution **prediction** plus additional **details**.)

Applying such ideas recursively, we can recover the original image $f_0[m, n]$ exactly using $f_K[m, n]$ and $e_k[m, n]$ for $k = 0, \dots, K - 1$.

The images $f_k[m, n]$ and $e_k[m, n]$ for $k = 0, \dots, K - 1$ form a **Laplacian pyramid** (so called for reasons to be described soon), where $f_K[m, n]$ is the top level of the pyramid (coarsest resolution), and $e_k[m, n]$ is the k th level of the pyramid.

The original base-level image $f_0[m, n]$ is reconstructed as follows:



For image compression purposes, the double-boxed items are quantized, coded, and transmitted or stored. Many pixels!

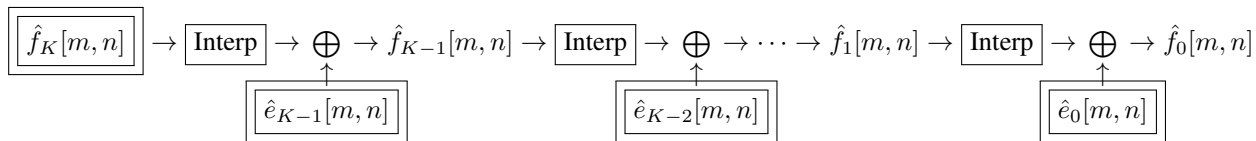
Coding issues

So far we have said little about coding.

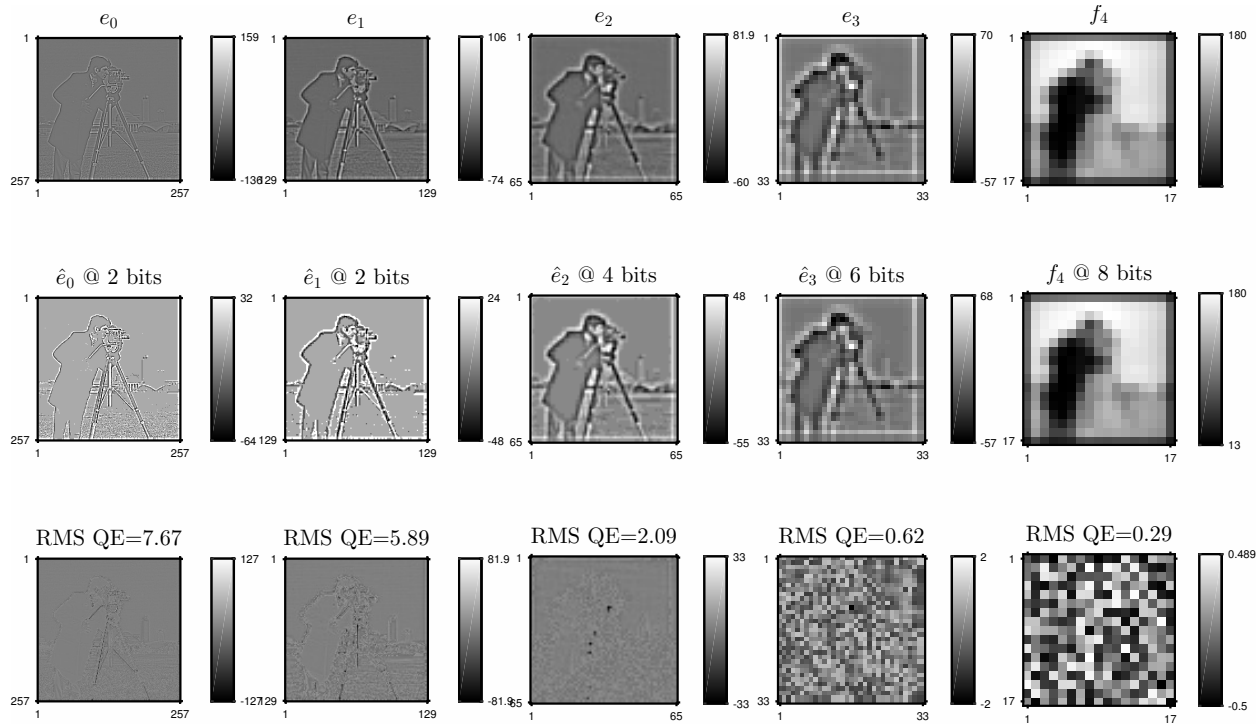
Indeed, so far we have actually *expanded* the number of pixels by about 33%, because e_0 is the same size as f_0 , and e_1 is about a quarter of that size, and e_2 is another quarter of that, so roughly $1 + 1/4 + 1/16 + \dots = \frac{1}{1-1/4} = 4/3$. At first glance this hardly looks like progress towards **image compression**!

- Because $f_K[m, n]$ and $e_k[m, n]$ have different characteristics (see analysis below) they are quantized differently.
- Higher level (coarser) images generally have higher variances than lower level images. This means that more bits per pixel must be assigned to these images. However, these images are smaller, resulting in a small number of total bits needed.
- Lower level (finer) images ($e_0[m, n]$ and $e_1[m, n]$) will have many zeros. These images, although large, require few bits for coding if some sophisticated entropy coding scheme is used.
- “Optimal” allocation of the bits between the different pyramid levels is an interesting and challenging problem.
- Quite faithful reproduction of images can be produced using pyramid coding at very low bit rates (e.g., 0.5 bit/pixel). Combinations of run-length encoding and other sophisticated entropy coding schemes are needed to reach the 0.5 bit/pixel level. Nevertheless, the somewhat complicated processing inherent in pyramid coding does yield high compression ratios.
- Pyramid coding is suitable for progressive data transmission. At any level of the reconstruction we can always produce a blurred representation of the image.

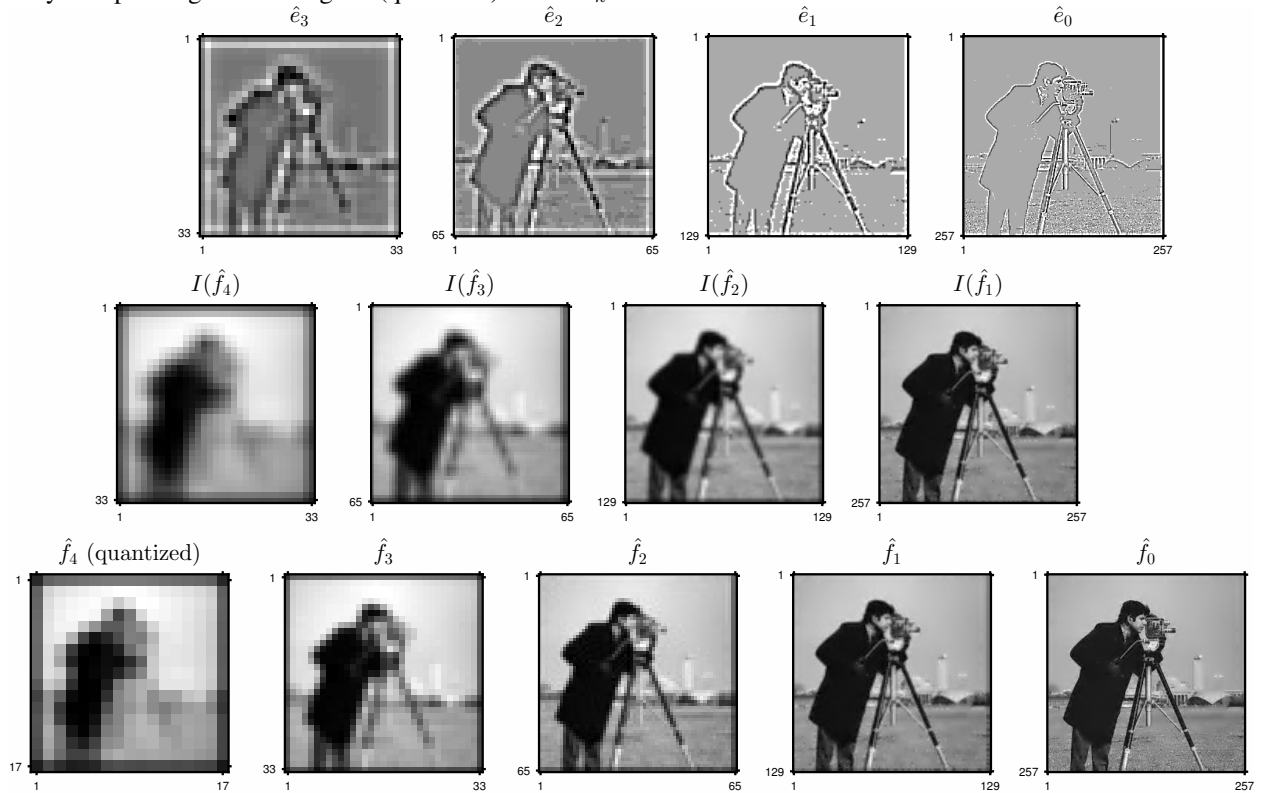
After quantizing, coding, and transmitting (or archiving) the bits corresponding to the base image and the error images, the decoder can (approximately) reconstruct the original base-level image $f_0[m, n]$ as follows:

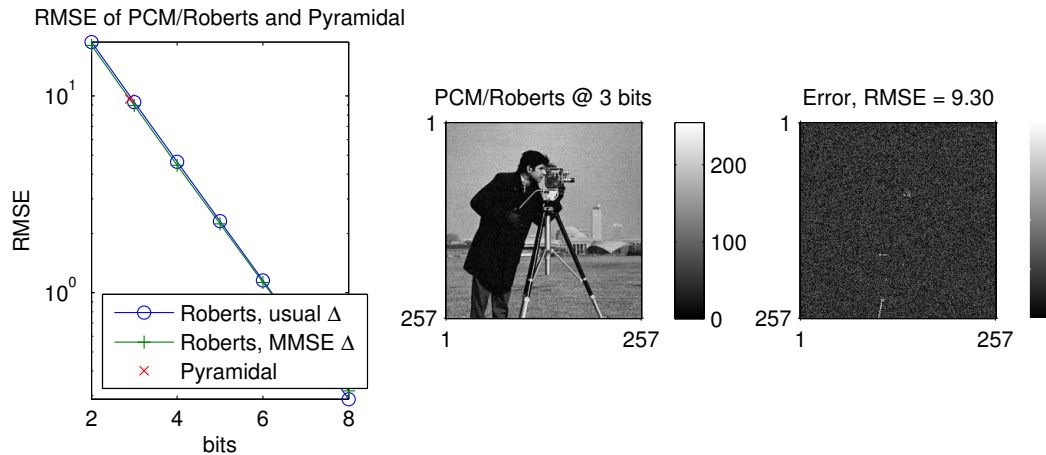
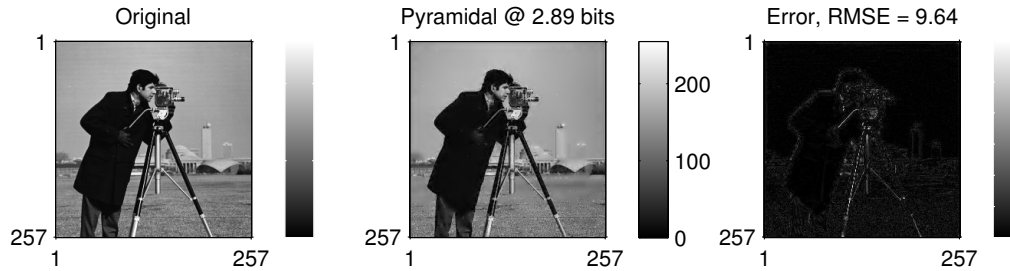


Example. The following figures show the case $K = 4$.



Repeatedly interpolating then adding the (quantized) details \hat{e}_k





Analysis of the Laplacian pyramid

(skim)

The Laplacian pyramid is just an invertible method for **image representation** (in fact it could be called an **image transformation**), and this representation is useful for other image processing tasks like **edge detection** and **object recognition**.

What are the properties of the $e_k[m, n]$ images?

(in 1D for simplicity)

The **down sampling** step:

$$f_k^L = h_L * f_k \xleftrightarrow{\text{DSFT}} F_k^L = H_L F_k$$

What should $H_L(\Omega)$ be ideally? $H_L(\Omega) = \text{rect}(\Omega/\pi)_{(2\pi)}$

$$\begin{aligned} f_{k+1} = \text{downsample}(f_k^L) \xleftrightarrow{\text{DSFT}} F_{k+1}(\Omega) &= \frac{1}{2} \left[F_k^L \left(\frac{\Omega}{2} \right) + F_k^L \left(\frac{\Omega}{2} \pm \pi \right) \right] \\ &= \frac{1}{2} \left[H_L \left(\frac{\Omega}{2} \right) F_k \left(\frac{\Omega}{2} \right) + H_L \left(\frac{\Omega}{2} \pm \pi \right) F_k \left(\frac{\Omega}{2} \pm \pi \right) \right]. \end{aligned}$$

In particular, if $H_L(\Omega) \approx \text{rect}(\frac{\Omega}{\pi})_{(2\pi)}$, then for $|\Omega| < \pi$ we have the following approximation:

$$\begin{aligned} F_k(\Omega) &\approx \frac{1}{2} H_L \left(\frac{\Omega}{2} \right) F_{k-1} \left(\frac{\Omega}{2} \right) \approx \frac{1}{2} H_L \left(\frac{\Omega}{2} \right) \left[\frac{1}{2} H_L \left(\frac{\Omega}{4} \right) F_{k-1} \left(\frac{\Omega}{4} \right) \right] \\ &= \frac{1}{4} H_L \left(\frac{\Omega}{2} \right) F_{k-1} \left(\frac{\Omega}{2} \right) \approx \dots \approx \frac{1}{2^k} H_L \left(\frac{\Omega}{2} \right) F_0 \left(\frac{\Omega}{2^k} \right). \end{aligned}$$

In particular, $F_k(\Omega)$ consists of those frequency components of $F_0(\Omega)$ up to $\pi/2^k$.

The **interpolation** step:

$$f_{k+1}^I \triangleq I\{f_{k+1}\} = h_I * f_{k+1}^U$$

where f_{k+1}^U is up-sampling by zero insertion, *i.e.*,

$$f_{k+1}^U[n] = \begin{cases} f_{k+1}[n/2], & n \text{ even} \\ 0, & \text{otherwise} \end{cases} \quad \xleftrightarrow{\text{DSFT}} F_{k+1}^U(\Omega) = F_{k+1}(2\Omega),$$

and h_I is the lowpass filter associated with the interpolation. (Typically $h_I = 2h_L$.) In the frequency domain:

$$\begin{aligned} F_{k+1}^I(\Omega) &= H_I(\Omega) F_{k+1}^U(\Omega) \\ &= H_I(\Omega) F_{k+1}(2\Omega) \\ &= H_I(\Omega) \frac{1}{2} [H_L(\Omega) F_k(\Omega) + H_L(\Omega \pm \pi) F_k(\Omega \pm \pi)] \\ &\approx H_L^2(\Omega) F_k(\Omega), \end{aligned}$$

assuming $h_I = 2h_L$ and making the very reasonable approximation (which is exact for ideal lowpass filters) that

$$H_I(\Omega) H_L(\Omega \pm \pi) \approx 0.$$

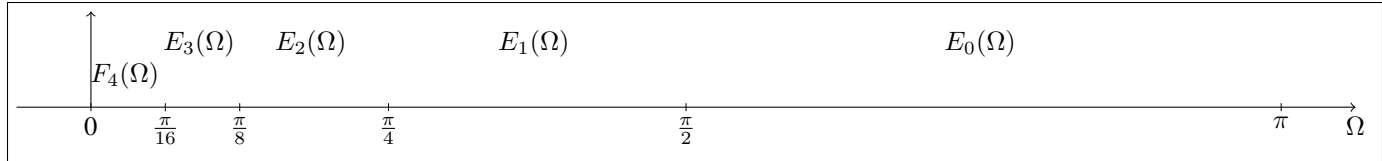
The **error** step:

$$\begin{aligned} e_k = f_k - I\{f_{k+1}\} &= f_k - f_{k+1}^I \xleftrightarrow{\text{DSFT}} E_k(\Omega) = F_k(\Omega) - F_{k+1}^I(\Omega) \approx [1 - H_L^2(\Omega)] F_k(\Omega) \\ &\approx [1 - H_L^2(\Omega)] \frac{1}{2^k} H_L\left(\frac{\Omega}{2}\right) F_0\left(\frac{\Omega}{2^k}\right), \end{aligned}$$

for $|\Omega| < \pi$. In particular, if $H_L(\Omega) = \text{rect}\left(\frac{\Omega}{\pi}\right)_{(2\pi)}$, then for $|\Omega| < \pi$:

$$E_k(\Omega) = \frac{1}{2^k} F_0\left(\frac{\Omega}{2^k}\right) \mathbb{I}_{\{\pi/2 < |\Omega| < \pi\}}.$$

Plugging in the limits of the indicator function into $F(\Omega/2^k)$ shows that $E_k(\Omega)$ contains information about the original image $F_0(\Omega)$ over the frequency band $\frac{\pi}{2^{k+1}} < \Omega < \frac{\pi}{2^k}$, which is closely related to **subband coding**.



More analysis of pyramid decomposition _____ (for ideal filters and interpolators) (**skip**)

First, lowpass: $f_k^L[m, n] = f_k[m, n] ** h[m, n]$ where $h[m, n] = \frac{1}{4} \text{sinc}_2\left(\frac{m}{2}, \frac{n}{2}\right)$. Hence

$$F_k^L(\Omega_1, \Omega_2) = F_k(\Omega_1, \Omega_2) H(\Omega_1, \Omega_2) = F_k(\Omega_1, \Omega_2) \text{rect}_2\left(\frac{\Omega_1}{\pi}, \frac{\Omega_2}{\pi}\right)_{(2\pi, 2\pi)}.$$

Second, downsample: $f_{k+1}[m, n] = f_k^L[2m, 2n]$. Hence from homework:

$$\begin{aligned} F_{k+1}(\Omega_1, \Omega_2) &= \frac{1}{4} \left[F_k^L\left(\frac{\Omega_1}{2}, \frac{\Omega_2}{2}\right) + F_k^L\left(\frac{\Omega_1}{2} - \pi, \frac{\Omega_2}{2}\right) + F_k^L\left(\frac{\Omega_1}{2}, \frac{\Omega_2}{2} - \pi\right) + F_k^L\left(\frac{\Omega_1}{2} - \pi, \frac{\Omega_2}{2} - \pi\right) \right] \\ &= \frac{1}{4} \left[F_k\left(\frac{\Omega_1}{2}, \frac{\Omega_2}{2}\right) \text{rect}_2\left(\frac{\Omega_1}{2\pi}, \frac{\Omega_2}{2\pi}\right)_{(2\pi, 2\pi)} + \dots \right]. \end{aligned}$$

To save writing, hereafter we only consider the range $-\pi \leq \Omega_1, \Omega_2 \leq \pi$ for the arguments on the left-hand side. Thus $F_{k+1}(\Omega_1, \Omega_2) = \frac{1}{4} F_k\left(\frac{\Omega_1}{2}, \frac{\Omega_2}{2}\right) \text{rect}_2\left(\frac{\Omega_1}{2\pi}, \frac{\Omega_2}{2\pi}\right)$, because the rect's eliminate the other terms over this range.

Applying this relationship recursively yields the following pyramid of lowpass filtered images:

$$F_k(\Omega_1, \Omega_2) = \left(\frac{1}{4}\right)^k F_0\left(\frac{\Omega_1}{2^k}, \frac{\Omega_2}{2^k}\right) \text{rect}_2\left(\frac{\Omega_1}{2\pi}, \frac{\Omega_2}{2\pi}\right), \quad k = 0, 1, 2, \dots$$

Third, interpolate: $f_{k+1}^I[m, n] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f_{k+1}[m, n] \text{sinc}_2(m/2 - k, n/2 - l)$, so from homework:

$$F_{k+1}^I(\Omega_1, \Omega_2) = 4 F_{k+1}(2\Omega_1, 2\Omega_2) \text{rect}_2\left(\frac{\Omega_1}{\pi}, \frac{\Omega_2}{\pi}\right) = F_k(\Omega_1, \Omega_2) \text{rect}_2\left(\frac{\Omega_1}{\pi}, \frac{\Omega_2}{\pi}\right).$$

Fourth, error image: $e_k[m, n] = f_k[m, n] - f_{k+1}^I[m, n]$, so

$$E_k(\Omega_1, \Omega_2) = F_k(\Omega_1, \Omega_2) - F_{k+1}^I(\Omega_1, \Omega_2) = F_k(\Omega_1, \Omega_2) \left[1 - \text{rect}_2\left(\frac{\Omega_1}{\pi}, \frac{\Omega_2}{\pi}\right) \right].$$

Thus, using $k = 0$ we have: $E_0(\Omega_1, \Omega_2) = F_0(\Omega_1, \Omega_2) \left[1 - \text{rect}_2\left(\frac{\Omega_1}{\pi}, \frac{\Omega_2}{\pi}\right) \right]$ (for $|\Omega_1, \Omega_2| \leq \pi$, and periodic otherwise).

Suppose we were to form an image $g_k[m, n] = I^k\{e_k[m, n]\}$ by taking the error image $e_k[m, n]$ at the k th level of the pyramid and performing k applications of the interpolation method given above to it (for $k \geq 1$).

Relate the spectrum of $g_k[m, n]$ to the spectrum of the base image $f_0[m, n]$. From above, for $k \geq 1$:

$$\begin{aligned} E_k(\Omega_1, \Omega_2) &= F_k(\Omega_1, \Omega_2) \left[1 - \text{rect}_2\left(\frac{\Omega_1}{\pi}, \frac{\Omega_2}{\pi}\right) \right] \\ &= \left(\frac{1}{4}\right)^k F_0\left(\frac{\Omega_1}{2^k}, \frac{\Omega_2}{2^k}\right) \left[\text{rect}_2\left(\frac{\Omega_1}{2\pi}, \frac{\Omega_2}{2\pi}\right) - \text{rect}_2\left(\frac{\Omega_1}{\pi}, \frac{\Omega_2}{\pi}\right) \right]. \end{aligned}$$

With $g_k[m, n] = I^k\{e_k[m, n]\}$ we have (for $k \geq 1$):

$$\begin{aligned} G_k(\Omega_1, \Omega_2) &= 4^k E_k[2^k \Omega_1, 2^k \Omega_2] \text{rect}_2\left(\frac{\Omega_1}{\pi/2^{k-1}}, \frac{\Omega_2}{\pi/2^{k-1}}\right) \\ &= F_0(\Omega_1, \Omega_2) \left[\text{rect}_2\left(\frac{2^k \Omega_1}{2\pi}, \frac{2^k \Omega_2}{2\pi}\right) - \text{rect}_2\left(\frac{2^k \Omega_1}{\pi}, \frac{2^k \Omega_2}{\pi}\right) \right] \text{rect}_2\left(\frac{\Omega_1}{\pi/2^{k-1}}, \frac{\Omega_2}{\pi/2^{k-1}}\right). \end{aligned}$$

Thus $G_k(\Omega_1, \Omega_2) = F_0(\Omega_1, \Omega_2) \left[\text{rect}_2\left(\frac{\Omega_1}{\pi/2^{k-1}}, \frac{\Omega_2}{\pi/2^{k-1}}\right) - \text{rect}_2\left(\frac{\Omega_1}{\pi/2^k}, \frac{\Omega_2}{\pi/2^k}\right) \right]$ (for $|\Omega_1, \Omega_2| \leq \pi$, and periodic otherwise), which is a set of concentric dyadic rectangular annular subbands.

Adaptive image coding and vector quantization _____ **(skim)**

Small sub-blocks of an image can be directly quantized using vector quantization.

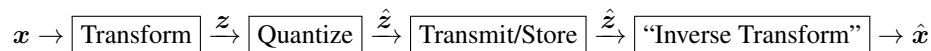
Typical block sizes range from 2×2 to 4×4 . Above 4×4 , an optimal vector quantizer gets very expensive.

Direct vector quantization is capable of modest reproduction but at very low bit rates (*e.g.*, less than 0.5 bits/pixel).

14.3 Transform image coding

- Transform coding techniques exploit the property that for typical images (and for appropriate transforms) most of the signal energy is concentrated in a small fraction of the transform coefficients. Because of this property, it is possible to code only a fraction of the transform coefficients without seriously affecting the image.
- The transform coefficients can be quantized and coded in any way. Traditionally, simple scalar quantization with uniform length coding has been used, where the number of reconstruction levels decreases away from the region of energy compaction.
- Vector quantization, especially of higher order coefficients, is starting to be used in real systems.
- To date, primarily linear transforms have been used for image coding.
- An ideal transform (like the KL transform) would make the coefficients uncorrelated (or even independent), in which case scalar quantization would be quite natural. (There still would be a benefit of going to VQ even in the case of independent coefficients because of the “sphere packing” problem.)

The basic block diagram of a transform codec is the following.



The vector $\mathbf{x} \in \mathbb{R}^N$ might represent the entire image, or, more typically, it represents some block of pixel values, *e.g.*, JPEG uses 8×8 blocks.

For simplicity of decoding, typically one uses a linear “inverse transform” in which the final image is represented as a linear combination of basis vectors:

$$\hat{\mathbf{x}} = \sum_{k=1}^K \mathbf{b}_k \hat{z}_k,$$

where the vectors $\{\mathbf{b}_1, \dots, \mathbf{b}_K\}$ are sometimes called the **dictionary** or **library**. The traditional choice uses a dictionary where $K = N$, but more recently **overcomplete** representations where $K > N$ have also been proposed.

For simplicity, we consider the case where $K = N$ hereafter, in which case if the \mathbf{b}_k vectors are a basis for \mathbb{R}^N , and hence linearly independent, then we can define a $N \times N$ transformation matrix

$$\mathbf{W} = [\mathbf{b}_1 \ \dots \ \mathbf{b}_N]^{-1},$$

so that the transform and inverse transform are given by:

$$\mathbf{z} = \mathbf{W}\mathbf{x}, \quad \hat{\mathbf{x}} = \mathbf{W}^{-1}\hat{\mathbf{z}}.$$

What are desirable properties of transformations?

- invertible
- easily computed
 - separable (for reduced computation via **row-column decomposition**)
 - real, fast, etc.
- correlation reduction (to facilitate scalar quantization)
- energy compaction
- energy preservation (like Parseval's theorem), for some constant c independent of x :

$$\|\mathbf{W}\mathbf{x}\|^2 = c \|\mathbf{x}\|^2.$$

This property holds with $c = 1$ if \mathbf{W} is a **unitary matrix**, *i.e.*, corresponding to a **orthogonal transformation**. In this case $\mathbf{W}^{-1} = \mathbf{W}'$, where \mathbf{W}' denotes the **Hermitian transpose** of \mathbf{W} , and

$$\|\mathbf{z}\|^2 = \|\mathbf{W}\mathbf{x}\|^2 = \mathbf{x}'\mathbf{W}'\mathbf{W}\mathbf{x} = \mathbf{x}'\mathbf{x} = \|\mathbf{x}\|^2.$$

This property is desirable because it assures us that small transform coefficients contribute only a small amount to the reconstructed signal's energy, so discarding a small transform coefficient (*i.e.*, quantizing it to zero) will not seriously degrade the image. (Of course, discarding too many "small" coefficients could have a synergistic effect that is noticeable.)

There is often a *trade-off* between the energy compaction property and computational considerations. For example, the 2D DFT is very easy to compute using the row-column 2D FFT. But image regions containing edges are poorly compacted by the DFT.

Karhunen-Loève Transform

Suppose that you model your image (or image block) \mathbf{x} as a realization of a random vector with known covariance function

$$\Sigma = \text{Cov}\{\mathbf{x}\} = \text{E}[(\mathbf{x} - \text{E}[\mathbf{x}])(\mathbf{x} - \text{E}[\mathbf{x}])'] .$$

Under this (often artificial) assumption, the *linear* transformation (actually, it is an **affine** transformation) that is optimal in terms of the energy compaction property, is the **Karhunen-Loève (KL)** transform (**KLT**).

The KLT solves the following problem. We wish to approximate $\mathbf{x} \in \mathbb{C}^N$ using the following affine expression:

$$\hat{\mathbf{x}} = \mathbf{B}\boldsymbol{\alpha} + \mathbf{b}_0,$$

where $\mathbf{B} = [\mathbf{b}_1 \dots \mathbf{b}_K]$ is a $N \times K$ matrix with $K \leq N$ orthonormal columns that does not depend on any given \mathbf{x} , but $\mathbf{b}_0 \in \mathbb{C}^N$ is also independent of \mathbf{x} , $\mathbf{B}'\mathbf{b}_0 = \mathbf{0}$, and each coefficient vector $\boldsymbol{\alpha} \in \mathbb{C}^K$ may depend (possibly even nonlinearly) on \mathbf{x} . We want to choose \mathbf{B} , $\boldsymbol{\alpha}$, and \mathbf{b}_0 to minimize the MSE $\text{E}[\|\hat{\mathbf{x}} - \mathbf{x}\|^2]$.

We derive shortly that the optimal choice for \mathbf{b}_0 is $\mathbf{b}_0 = (\mathbf{I} - \mathbf{B}\mathbf{B}')\boldsymbol{\mu}$, where $\boldsymbol{\mu} = \text{E}[\mathbf{x}]$, the optimal choice for $\boldsymbol{\alpha}$ is $\boldsymbol{\alpha} = \mathbf{B}'\mathbf{x}$, and most importantly, the optimal basis vectors \mathbf{b}_k are the K eigenvectors of Σ corresponding to the K largest eigenvalues.

In principle, using lexicographic ordering of many finite-sized images (of the same size) we could use MATLAB's `eig` function to find the required eigenvalues and eigenvectors. In practice this can be nontrivial for large image sizes.

Alternatively, one can find the KL representation of small (*e.g.*, 8×8) blocks of images from a representative training set (which are used to form an empirical estimate of the covariance matrix). This approach requires finding the eigenvectors of only a 64×64 covariance matrix, which is trivial. The eigenvectors found this way will be optimal for the training set; whether they will have good energy compaction for a subsequent image of interest will depend on how “similar” that image is to those in the training set.

If Σ is circulant, *i.e.*, if the image is **circularly WSS**, then what are the eigenvectors and eigenvalues?

The DFT vectors and the (circular) power spectrum of the image.

The sense in which the KL transform is optimal is that if we order the eigenvalues from largest to smallest, and then look at the *expected error* in a truncated expansion that only includes the first, say, K coefficients, then the KL transform will have the minimum such MSE over all linear transformations. In other words, the first K components describe as much of the variance of the signal (energy compaction) as is possible with a linear (actually affine) orthonormal transformation.

Furthermore, the KL coefficients are completely uncorrelated, as shown by the following argument for a finite-extent signal \mathbf{x} .

Denote the covariance matrix of the lexicographically ordered image vector \mathbf{x} by

$$\Sigma = \mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])'] .$$

Because Σ is symmetric positive semi-definite, it has an orthogonal eigen-decomposition:

$$\Sigma = \mathbf{V} \mathbf{\Lambda} \mathbf{V}' ,$$

where the columns of \mathbf{V} are its eigenvectors, and $\mathbf{\Lambda}$ is a diagonal matrix with its eigenvalues λ_k .

As proven shortly, the KL transform in matrix vector form is given by:

$$\mathbf{X} = \mathbf{V}' (\mathbf{x} - \mathbb{E}[\mathbf{x}]) \tag{14.9}$$

and the inverse transform is given by

$$\mathbf{x} = \mathbb{E}[\mathbf{x}] + \mathbf{V} \mathbf{X} .$$

(Note the small detail here that the transform is actually **affine** rather than **linear** if the mean is nonzero.)

To show that the KL coefficients are uncorrelated, we examine the covariance matrix of the coefficient vector \mathbf{X} :

$$\text{Cov}\{\mathbf{X}\} = \text{Cov}\{\mathbf{V}' \mathbf{x}\} = \mathbf{V}' \text{Cov}\{\mathbf{x}\} \mathbf{V} = \mathbf{V}' \Sigma \mathbf{V} = \mathbf{V}' (\mathbf{V} \mathbf{\Lambda} \mathbf{V}') \mathbf{V} = \mathbf{\Lambda} ,$$

which is diagonal. So the KL coefficients are perfectly **uncorrelated**. They also have the best **energy compaction** as shown below.

However, for the reasons described above, the KL transform (applied to whole images) is *impractical*, so is rarely used in practical image coding. It is useful for analysis and insight.

The KL transform is optimal for a *linear* (affine) approximation, but in contemporary image coding we often use **nonlinear approximation** in which one approximates the signal using a linear combination of K basis functions, but the set of basis functions is signal dependent [14].

The KL transform, particularly when evaluated from empirical data, is closely related to **principal components analysis** (PCA) [18]. See [19] for interesting generalizations. See [20] for ML formulation and sparse variations.

KLT derivation

Let \mathbf{x} be a random vector of length N with known $N \times N$ covariance matrix Σ and known mean μ .

Let $\mathbf{B} = [\mathbf{b}_1 \dots \mathbf{b}_N]$ denote an $N \times N$ **orthonormal** basis for \mathbb{R}^N or \mathbb{C}^N with basis vectors \mathbf{b}_i , $i = 1, \dots, N$, such that $\mathbf{B}'\mathbf{B} = \mathbf{B}\mathbf{B}' = \mathbf{I}$, i.e., $\|\mathbf{b}_i\| = 1$ and $\mathbf{b}_i'\mathbf{b}_j = 0$, $i \neq j$.

We want to form an approximation to \mathbf{x} using just $K < N$ basis components $\Phi \triangleq [\mathbf{b}_1 \dots \mathbf{b}_K]$:

$$\hat{\mathbf{x}}(\boldsymbol{\alpha}, \boldsymbol{\nu}) = \sum_{i=1}^K \mathbf{b}_i \alpha_i + \boldsymbol{\nu} = \Phi \boldsymbol{\alpha} + \boldsymbol{\nu},$$

where $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_K)$ can depend on \mathbf{x} , but $\boldsymbol{\nu}$ cannot (we must preselect $\boldsymbol{\nu}$), and where $\boldsymbol{\nu}'\mathbf{b}_i = 0$ for $i = 1, \dots, K$.

Using only K components is akin to an extreme form of quantization where we discard the other $N - K$ coefficients.

Goal: choose $\boldsymbol{\nu}$, $\boldsymbol{\alpha}$, and \mathbf{B} (the \mathbf{b}_i vectors) to minimize the MSE between $\hat{\mathbf{x}}$ and \mathbf{x} , subject to the constraint $\mathbf{B}'\mathbf{B} = \mathbf{B}\mathbf{B}' = \mathbf{I}$.

For a given basis \mathbf{B} , the approximation error is

$$\varepsilon(\boldsymbol{\alpha}, \boldsymbol{\nu}) = \|\hat{\mathbf{x}}(\boldsymbol{\alpha}, \boldsymbol{\nu}) - \mathbf{x}\|^2 = \left\| \sum_{i=1}^K \alpha_i \mathbf{b}_i + \boldsymbol{\nu} - \mathbf{x} \right\|^2 = \|\boldsymbol{\Phi} \boldsymbol{\alpha} + \boldsymbol{\nu} - \mathbf{x}\|^2$$

and we first minimize this error over $\boldsymbol{\alpha}$:

$$\min_{\boldsymbol{\alpha}} \varepsilon(\boldsymbol{\alpha}, \boldsymbol{\nu}) = \min_{\boldsymbol{\alpha}} \|\hat{\mathbf{x}}(\boldsymbol{\alpha}, \boldsymbol{\nu}) - \mathbf{x}\|^2 = \min_{\boldsymbol{\alpha}} \left\| \sum_{i=1}^K \alpha_i \mathbf{b}_i + \boldsymbol{\nu} - \mathbf{x} \right\|^2 = \min_{\boldsymbol{\alpha}} \|\boldsymbol{\Phi} \boldsymbol{\alpha} + \boldsymbol{\nu} - \mathbf{x}\|^2$$

$$\frac{\partial}{\partial \alpha_i} \varepsilon(\boldsymbol{\alpha}, \boldsymbol{\nu}) = \mathbf{b}_i' \left[\sum_{k=1}^K \alpha_k \mathbf{b}_k + \boldsymbol{\nu} - \mathbf{x} \right] = \alpha_i - \mathbf{b}_i' \mathbf{x},$$

because $\mathbf{B}'\mathbf{B} = \mathbf{B}\mathbf{B}' = \mathbf{I}$ and $\mathbf{b}_i' \boldsymbol{\nu} = 0$. Equating the partial derivatives to zero yields the usual LS solution:

$$\alpha_i = \mathbf{b}_i' \mathbf{x}, \quad i = 1, \dots, K, \quad \text{i.e.,} \quad \boldsymbol{\alpha} = [\boldsymbol{\Phi}' \boldsymbol{\Phi}]^{-1} \boldsymbol{\Phi}' \mathbf{x} = \boldsymbol{\Phi}' \mathbf{x}.$$

Thus we have

$$\hat{\mathbf{x}}(\boldsymbol{\nu}) = \sum_{i=1}^K \mathbf{b}_i \mathbf{b}_i' \mathbf{x} + \boldsymbol{\nu} = \boldsymbol{\Phi} \boldsymbol{\Phi}' \mathbf{x} + \boldsymbol{\nu},$$

with corresponding mean-squared error:

$$\text{MSE}\{\boldsymbol{\nu}\} \triangleq \mathbb{E} \left[\|\hat{\mathbf{x}}(\boldsymbol{\nu}) - \mathbf{x}\|^2 \right] = \mathbb{E} \left[\left\| \sum_{i=1}^K \mathbf{b}_i \mathbf{b}_i' \mathbf{x} + \boldsymbol{\nu} - \mathbf{x} \right\|^2 \right] = \mathbb{E} \left[\left\| \boldsymbol{\nu} - \sum_{i=K+1}^N \mathbf{b}_i \mathbf{b}_i' \mathbf{x} \right\|^2 \right] \equiv \|\boldsymbol{\nu}\|^2 - 2\boldsymbol{\nu}' \sum_{i=K+1}^N \mathbf{b}_i \mathbf{b}_i' \mathbf{x}.$$

Taking the gradient with respect to ν yields

$$\frac{1}{2} \nabla' \text{MSE}\{\nu\} = \nu - \sum_{i=K+1}^N \mathbf{b}_i \mathbf{b}_i' \mu.$$

Equating this gradient to zero yields:

$$\nu = \sum_{i=K+1}^N \mathbf{b}_i \mathbf{b}_i' \mu.$$

Fortunately, this solution for the best ν satisfies $\mathbf{b}_i' \nu = 0$, $i = 1, \dots, K$, otherwise we would have to minimize over ν using Lagrange multipliers.

We now have our MMSE (over α and ν) affine approximation:

$$\hat{\mathbf{x}} = \sum_{i=1}^K \mathbf{b}_i \mathbf{b}_i' \mathbf{x} + \sum_{i=K+1}^N \mathbf{b}_i \mathbf{b}_i' \mu = \sum_{i=1}^K \mathbf{b}_i \mathbf{b}_i' (\mathbf{x} - \mu) + \mu = \Phi \Phi' (\mathbf{x} - \mu) + \mu.$$

In words, we subtract the mean μ from \mathbf{x} , project onto the span of $\{\mathbf{b}_i, i = 1, \dots, K\}$ and then add back the mean.

We can now turn to the more interesting and challenging problem of designing the basis vectors $\{\mathbf{b}_i, i = 1, \dots, K\}$ that minimize the MSE.

The MSE is

$$\begin{aligned} \text{MSE} &= \mathbb{E} \left[\|\hat{\mathbf{x}} - \mathbf{x}\|^2 \right] = \mathbb{E} \left[\left\| \sum_{i=1}^K \mathbf{b}_i \mathbf{b}_i' (\mathbf{x} - \mu) + \mu - \mathbf{x} \right\|^2 \right] = \mathbb{E} \left[\left\| \sum_{i=K+1}^N \mathbf{b}_i \mathbf{b}_i' (\mathbf{x} - \mu) \right\|^2 \right] \\ &= \sum_{i=K+1}^N \mathbb{E} \left[(\mathbf{b}_i' (\mathbf{x} - \mu))^2 \right] = \sum_{i=K+1}^N \mathbf{b}_i' \mathbb{E} [(\mathbf{x} - \mu)(\mathbf{x} - \mu)'] \mathbf{b}_i = \sum_{i=K+1}^N \mathbf{b}_i' \Sigma \mathbf{b}_i. \end{aligned}$$

To minimize the MSE over the basis set $\{\mathbf{b}_i\}$, we must include the constraint $\|\mathbf{b}_i\|^2 = 1$ using a Lagrange multiplier:

$$\nabla' \text{MSE} - \lambda_i \nabla' (\|\mathbf{b}_i\|^2 - 1) = 2\Sigma\mathbf{b}_i - \lambda_i 2\mathbf{b}_i$$

Equating to zero yields:

$$\Sigma\mathbf{b}_i = \lambda_i\mathbf{b}_i,$$

which means that each \mathbf{b}_i must be an **eigenvector** of the covariance matrix Σ with corresponding **eigenvalue** λ_i .

Substituting this relationship back into the error expression yields:

$$\text{MSE} = \sum_{i=K+1}^N \lambda_i.$$

Clearly then, to minimize the MSE for any K we should arrange the eigenvalues $\{\lambda_i\}$ (and hence the corresponding eigenvectors $\{\mathbf{b}_i\}$) in *decreasing order*.

Thus, we have shown that, over all (affine) orthonormal transformations, truncated to $K < N$ terms, the KLT (14.9) has the MMSE, assuming that the covariance matrix Σ and mean $\boldsymbol{\mu}$ are known.

Example. Generally it is impractical to apply the KLT to an entire image. However, one can extract small blocks of pixels from an image (e.g., 8×8) and arrange those as 64-element vectors \mathbf{x}_l for $l = 1, \dots, L$ where L is the number of such blocks available in a training set. From the training set, we can *estimate* the covariance matrix Σ using the sample covariance:

$$\hat{\Sigma} = \frac{1}{L} \sum_{l=1}^L (\mathbf{x}_l - \bar{\mathbf{x}}) (\mathbf{x}_l - \bar{\mathbf{x}})' \text{ where the sample mean vector is: } \bar{\mathbf{x}} = \frac{1}{L} \sum_{l=1}^L \mathbf{x}_l.$$

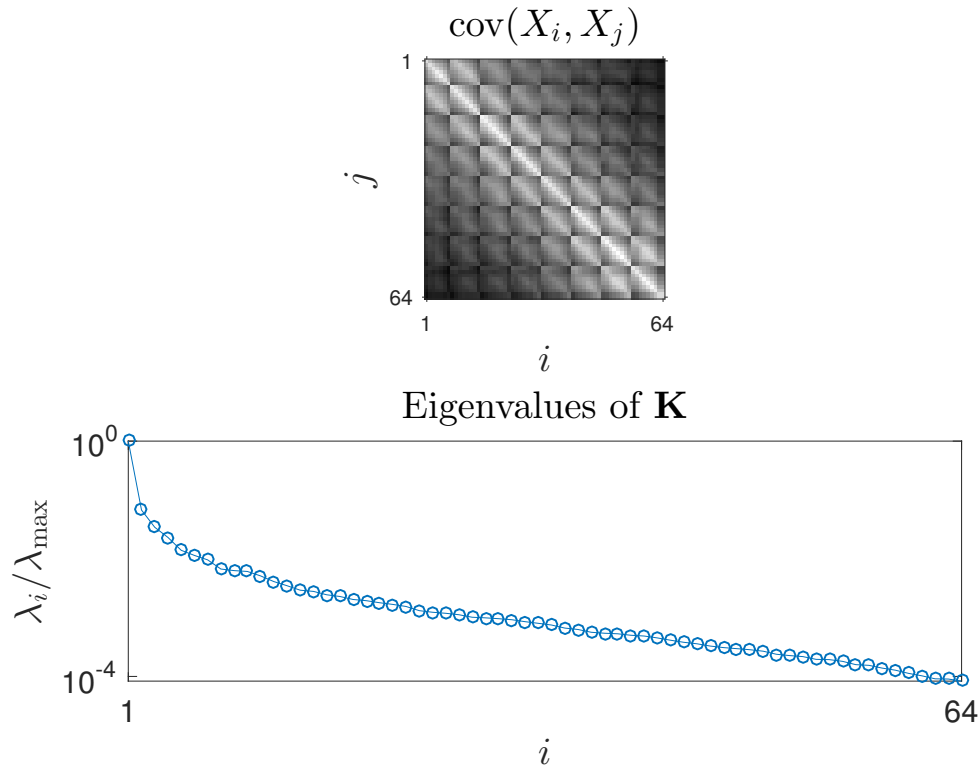
The empirical covariance matrix $\hat{\Sigma}$ is positive semi-definite, so we can use its (orthonormal) eigenvectors as an **empirical KL transform**.

A practical disadvantage of this approach is that the basis would be different for every image (or class of images), so in a heterogeneous environment like the internet, one would have to transmit the basis along with the image, which would add some overhead and complexity. Using a “generic” predetermined basis like the DCT avoids the need for communicating a different basis for each image.

Summary: the KLT gives the MMSE basis for a truncated representation of a random vector with known mean and covariance matrix. There was no consideration of quantization errors in this analysis. However, for “high resolution” (asymptotic) bit rates, one can generalize this conclusion to include the effects of quantization errors. (See EECS 651...)

Example.

I took the 256×256 `camerman.tif` image and partitioned it into $L = 32^2$ subimages of size 8×8 that I arranged as a collection of L vectors of length $N = 64$ and passed it to MATLAB's `COV` routine that computes the sample covariance $\hat{\Sigma}$ described above. The figure below shows that 64×64 covariance matrix.



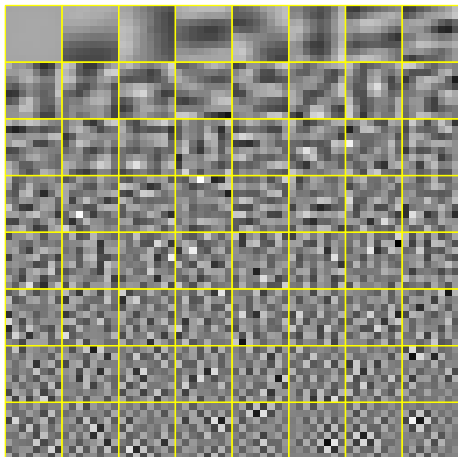
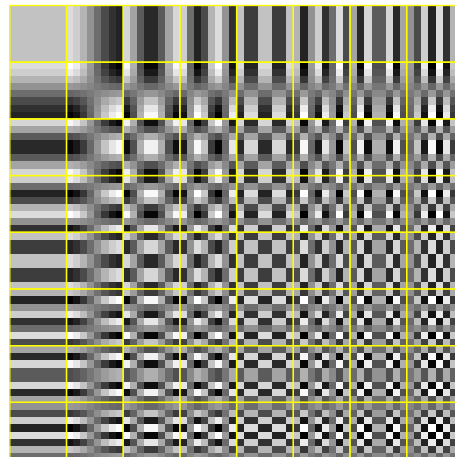
Why is the 8th diagonal bright? [High correlation between pixels and their neighbors above/below.](#)

I then passed the sample covariance matrix to MATLAB's `eig` function that computed its eigenvalues (shown below) and eigenvectors. I reshaped each 64×1 eigenvector into an 8×8 eigenimage; all 64 eigenimages are shown below.

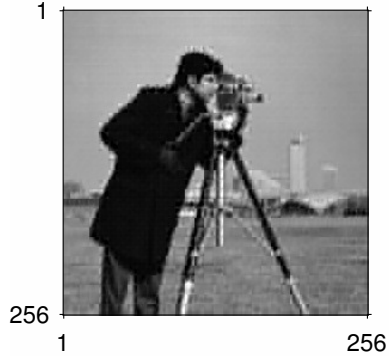
For comparison purposes, I also show the 64 basis images associated with the DCT command (using `kron` and `dctmtx`). There is a remarkable degree of similarity.

The following figure shows how using just $K = 6$ terms from the KLT basis yields (slightly) lower MSE than using $K = 6$ terms from the DCT.

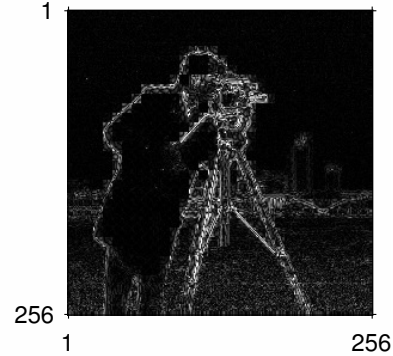
All 64 eigenimages

All 64 8×8 DCT basis images

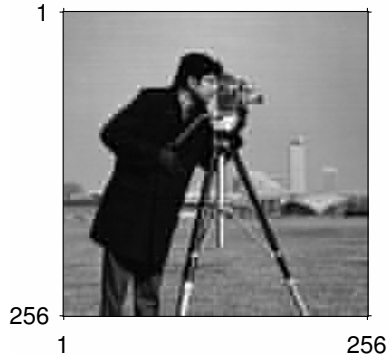
KL, $M=6$



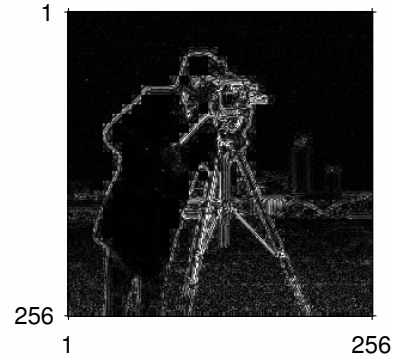
Error, RMSE=16.0



DCT, $M=6$ (zig-zag)



Error, RMSE=16.5



Other transforms **(skip)**

DCT, DFT, and Hadamard are the most common.

The **Hadamard transform** is computationally the simplest. Its $N \times N$, basis matrices are denoted \mathbf{H}_n where $N = 2^n$, $n = 1, 2, 3, \dots$. They can be generated easily by the core matrix

$$\mathbf{H}_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

and the recursion

$$\mathbf{H}_n = \mathbf{H}_{n-1} \otimes \mathbf{H}_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{H}_{n-1} & \mathbf{H}_{n-1} \\ \mathbf{H}_{n-1} & -\mathbf{H}_{n-1} \end{bmatrix}.$$

The Hadamard transform is **unitary**.

If we ignore the $\sqrt{2}$'s, computing the transform simply requires additions and subtractions!

However, the Hadamard transform usually has poorer energy compaction properties than either the DFT or DCT.

There usually is a trade-off between complexity and energy compaction.

We have previously discussed the fact that the DCT often has better energy compaction than the DFT because of the way edge conditions are handled.

Practical considerations in transform image coding**(skim)**

Because of computational constraints, most transform coding systems work on **subimages**.

However, the subimages must not be chosen too small or there will be little exploitation of correlations between image regions.

Typically 8 x 8 or 16 x 16 sub-images are used.

All coefficients of the transform are not coded with equal accuracy.

There are two basic approaches to transferring reduced information about transform coefficients.

- **Zonal and threshold coding**

- Only encode a limited set of transform coefficients.
- Each coefficient of the reduced set is coded with high accuracy.
- The set of coefficients to be coded can be determined as follows.
 - Use predetermined **zones**, as shown in [2, Fig. 10.43].
 - Select coefficients based on a threshold condition.

Obviously, the threshold approach produces a better representation, but must somehow also encode information about the positions of the selected frequencies.

Run-length coding and variations can be applied for this purpose.

- **Bit allocation methods**

Bits for coding subimages are allocated based on the expected variance of the transform coefficients. Coefficients with a large variance are assigned a large number of bits, and coefficients with small variances are assigned a small number of bits. An example of such an allocation for DCT coding at 0.5 bit/pixel is shown in [2, Fig. 10.44].

So that the same quantization step size can be used for all coefficients, typically the coefficients are first normalized by their “expected” standard deviation prior to scalar quantization.

Degradations

- Quantization noise can causes “graininess”
- Coefficients corresponding to high spatial frequencies are usually eliminated or assigned fewest bits, resulting in a loss of **spatial resolution**.
- Thus use of subimages causes **blocking effect** due to artificial discontinuities at block boundaries that are very noticeable to the human eye.

Summary of design parameters

- transform
- subimage size
- selection of coefficient set
- bit allocation
- quantization levels

One practical issue is whether a variable bit rate is tolerable in terms of decoder complexity. For example, using a threshold to select coefficients will yield different numbers of coded coefficients for different blocks.

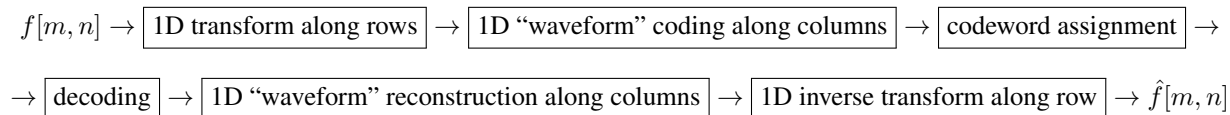
Reducing blocking effects

Blocking artifacts can be reduced by overlapping subimages. However, in image coding large overlaps are not acceptable because of the greatly increased number of coefficients. Consequently, only small overlap, or no overlap at all, is used in practice for transform image coding.

Blocking artifacts can also be reduced by low pass filtering in the neighborhood of subimage boundaries.

Hybrid transform coding

In a hybrid coder, one spatial dimension is waveform coded and the other is transform coded, as illustrated below



Although conceptually simple, hybrid systems have not been used very much in practice. This is because at very low bit rates, transform coders work best, but at very high bit rates where faithful reproductions are needed, a hybrid system does not perform any better than a simple waveform coder. Thus, hybrid coders have limited applicability in 2D.

Adaptive coding schemes

Bit allocations, zones, etc. can all be selected adaptively, either per image, or per block or group of blocks.

What is the trade-off of adaptive methods?

Extra information related to adaptation must be communicated to the receiver. This means extra complexity and reduced efficiency.

The overhead associated with adaptive methods usually is worth the overall savings!

A classic example of this is JPEG, which uses adaptive coding of DCT coefficients in 8×8 blocks.

Example. JPEG compressed image. Anything funny in the image here? **??**

(class/pair)



JPEG coding, a case-study in practical transform image coding (from [25])

Goals:

- Close to state-of-the art (circa 1990) in terms of compression rate
- Generally applicable (grayscale, color, multiple component)
- User selectable quality level
- Tractable computation
- Four modes (4 separate codecs because no single method adequately works for all situations):
 - Sequential - left-to-right, top-to-bottom scan
 - Progressive - blurry-to-clear recovery. First pass, low frequency DCT coefficients. Subsequent passes, additional frequency bands of DCT coefficients. Decoded images are all the same size at each pass.
 - Lossless - exact recovery (albeit with smaller compression ratios)
 - Hierarchical - access coarsely sampled version without decoding high resolution information (similar to **pyramid coding**). Useful for display on monitor before printing (thumbnail).

Design choices

1. **Transform**

Trade-off between complexity and energy compaction.

What are the two extremes?

- KL is best compaction, most complexity because it is image (covariance class) dependent.
- Waveform coding is lowest complexity but no energy compaction.

Practical compromises

- A low complexity choice is the **Hadamard transform**, discussed above, which requires only additions and subtractions. But its energy compaction underperforms the DFT and DCT.
- The **JPEG** committee chose the **DCT** based on a (blinded) competition using subjective image quality.

We would expect DCT to outperform DFT based on earlier discussion of edge conditions for DCT (mirroring) vs DFT (periodic).

- Accuracy specifications on DCT (due to cosines)
- 8 or 12 bit input images (easy to add front-end for floats)
- For 8-bit input images, the DCT coefficients are in the range $-2^{10}, 2^{10} - 1$.

2. Subimage size

Trade-off between computation and exploitation of correlation.

- Large subimages (or entire image) exploit more correlation, possibly yielding better compression ratios, although the “economies of scale” in things like shared quantization tables are reduced.
- Small subimages require less computation.
If an $N \times N$ image is broken into blocks of size $M \times M$, the computation required is $O(M^2 \log M)$ per block for $(N/M)^2$ blocks for a total of $O(N^2 \log M)$, so the smaller M is the less computation.
- The **JPEG** committee chose 8×8 , again based on blinded visual quality.

3. Quantization / coefficient subset

The 64 DCT coefficients for each block have different variances/energies, so one should quantize them differently.

Some subset of coefficients will even be discarded (quantized to zero).

See earlier discussion of predetermined **zones** (predetermined subset) vs **threshold coding** where the subset is chosen based on the coefficient magnitudes.

Trade-off: threshold coding preserves more signal energy, but requires extra bits to code indices of nonzero subset for every block.

The **JPEG** standard:

- 8 by 8 table describing (uniform scalar) quantizer step size for each of the 64 DCT coefficients, from 1 to 255.
In terms of the subset of nonzero coefficients, is this **zones** or **threshold based**? **zones**

- Adaptive: specified by “user” (coder) for each image.
(Default quantization tables are available based on psychovisual experiments.)

4. **Coding** (bit allocation)

- DC coefficient coded by difference from previous block (*cf.* Delta modulation ideas)
Why? Usually large (0-255 image values), and often varies slowly spatially.
- Remaining coefficients ordered in a zig-zag sequence from low to high spatial frequency.
- Entropy coding (Huffman or arithmetic) of (bits describing) quantized coefficients.
Requires separate “user supplied” Huffman code table be shared (*e.g.*, encoded) between the coder and decoder.
The need for this reflects the fact that DCT of 8×8 does not yield perfectly uncorrelated transform coefficients.

Other considerations

- Color images etc. (each plane compressed separately)
- Multiple quantization tables and coding tables supported, so they can differ for each component. (But only one table within a component.)

Performance (starting with 8-bit images)

- 0.25 - 0.5 bits/pixel: moderate to good quality, sufficient for some applications;
- 0.5 - 0.75 bits/pixel: good to very good quality, sufficient for many applications;
- 0.75 - 1/5 bits/pixel: excellent quality, sufficient for most applications;
- 1.5 - 2.0 bits/pixel: usually indistinguishable from the original, sufficient for the most demanding applications.
- Lossless mode (prediction from Left, Above, Above-Left neighbors) yields 2:1 compression of moderately complex color scenes.

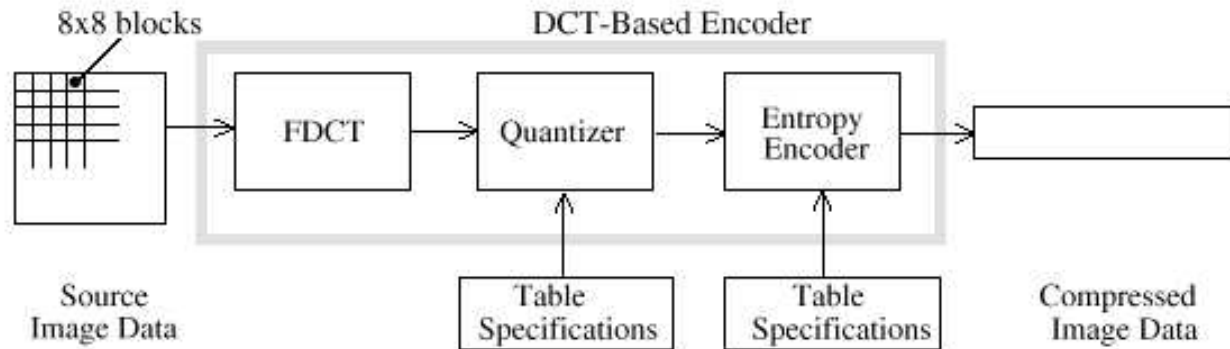


Figure 1. DCT-Based Encoder Processing Steps

Issue: “block artifacts” - many papers on correction methods.

14.4 Image model coding

The block diagram of a model coder is shown below.

Transmitter: $f[m, n] \rightarrow$ analysis \rightarrow model parameters \rightarrow encoder \rightarrow

Receiver: \rightarrow decoder \rightarrow quantized model parameters \rightarrow synthesis $\rightarrow \hat{f}[m, n]$

In principle, one can use any model of a class of images to generate a reduced set of model parameters capturing image content.

Example. Replace “grass” parts of an image with synthetically generate grass texture.

Example. [2, Fig. 10.53] A portion of the mandrill’s facial hair replaced with pixels synthesized by a simple first-order Markov model. Hundreds of pixels reduced to just a few parameters.

- Geometry models
 - Objects in the image are contained in a finite set of possible objects.
- Texture models.
 - Image texture often can be modeled as a random field with some statistical description. Psychophysical measurements have shown that random fields with the same 2nd order statistics (*i.e.*, covariance matrices) appear similar to a human observer.
- Combined geometry and texture.
 - Segment an image into uniform regions based on edge information and generate statistically based texture within identified regions.
- Fractals.
 - Use fractal measures to capture complex geometry using a small number of parameters. This method is similar to texture based methods, where a specific fractal model is used to capture statistics.

The **synthesis** part of such a system is quite challenging, and typically requires a decoder with sizable computational capability.

14.5 Interframe image coding

There are several ways that we can exploit interframe correlations to improve performance and/or reduce average bit rates.

In the simplest system, either direct waveform coding, such as DPCM, or transform coding, such as DCT, can be directly extended to three dimensions. Direct extension to 3D often is not optimal, especially because of the increased computation.

A hybrid system, unlike in 2D, can be very effective for interframe coding. A 3D hybrid system is shown below.

Transmit:

$$f(m, n, z) \rightarrow \boxed{\text{2D transform for each } z} \rightarrow T(k, l, z) \rightarrow \boxed{\text{“Waveform coding” along } z} \rightarrow \boxed{\text{codeword assignment}} \rightarrow$$

Receive:

$$\rightarrow \boxed{\text{decoding}} \rightarrow \boxed{\text{1D waveform reconstruction along } z} \rightarrow \boxed{\text{2D inverse transform for each } z} \rightarrow \hat{f}(m, n, z)$$

The waveform coding portion is usually DPCM so that only one or two frames must be stored at the decoder.

The hybrid system greatly reduces computations because only those DCT coefficients which are retained need be coded by the interframe DPCM system. That is, waveform coding is only applied to a fraction of the DCT coefficients.

There is no major delay in a hybrid system. For example, in a full 3D transform coder, one frame cannot be reconstructed until all transform coefficients within a 3D block are received. This means there is at least an N-frame delay in reconstructions for a 3D transform coder using N consecutive frames of information.

Some simple waveform coders based on DM or DPCM only transmit interframe differences that exceed a certain threshold. The instantaneous bit rate fluctuates, but the average rate can be very small. With proper buffering at the receiver, this simple type of system is capable of significantly reduced bit rates.

As discussed in the previous chapters, interframe motion can be estimated. Using estimates of the motion, we can transmit only the interframe difference between the expected, motion compensated image and the current image can be transmitted. Also, the motion parameters must be transmitted. The motion is estimated by the receiver using multiple frames, assuming the same motion parameters between frames. Thus, motion does not need to be transmitted given that the decoder has access to the previous estimated frames.

Color image coding

Many options: code each channel separately, treat as 3D with 3 frames, use VQ, ...

Channel error effects

Errors in communication have different effects on different image coding schemes. In PCM coding, a bit error only affects the particular pixel whose intensity is represented by the bit in error. This type of noise, at reasonable low bit rates, can be removed using median filtering, or other nonlinear filtering methods, or often just ignored.

In DPCM coding, bit reversals affect multiple pixel values. Because reconstructed intensities are used recursively, a bit reversal affects all subsequent pixel intensities from that point on. Because of the deleterious effect of a small number of bit reversals on DPCM reconstruction, these systems are often used with error correcting codes. Error correction, which is necessary to maintain fidelity, increases the bits/pixel needed to code the image.

A bit reversal in transform codings affects one particular coefficient. Each coefficient, however, affects all pixels within the subimage. This, the entire subimage may be corrupted solely by a single bit reversal. Again, as with DPCM, error correcting codes are often used with transform coders to eliminate the effect of channel noise. The addition of error correction once again increases the bit/pixel needed to code the image.

14.6 Summary

This chapter described methods for image coding. Image (and video) coding are very active research areas, due to the many commercial, medical, and scientific applications of digital image.

The JPEG2000 standard is based on wavelets, which offer many advantages including **progressive decoding** (*cf.* pyramid decoder). However, a drawback of wavelets (and JPEG 2000) is that block-based methods (*e.g.*, the original JPEG) are easier to realize in hardware.

Final comments on image coding

If a highly accurate reconstruction of an image is needed, then there is little difference between the bit rates of waveform and transform coders. Consequently, because of their simplicity and robustness to channel noise, waveform coders are usually selected for these applications.

Transform coders, such as JPEG, yield good performance at low bit rates (*i.e.*, in the 0.5-1 bit/pixel range). These coders are usually computationally complex, but can be justified in circumstances where the cost of the encoder and decoder are small compared to the cost of communication bandwidth (*e.g.*, the Web).

Model coders can yield modest performance but at greatly reduced bit rate (*i.e.*, < 0.1 bit/pixel). These coders are most commonly used in applications where intelligibility is the major concern and bandwidths are minimal.

Adaptive coding, although conceptually expensive, often can enhance the performance of almost any image coding system. For most applications, the increased complexity and slight loss of bandwidth due to communication of adaptive parameters can often be justified given the level of improvement (*e.g.*, JPEG adaptive bit allocation).

Inter-frame coding is very useful in applications such as HDTV where the sequence of frames has significant temporal correlation.

A general rule is that as the bits/pixel decreases, the complexity increases. The major design trade-off usually is between the

complexity of the encoder vs. decoder given a particular communications bandwidth.

Based on the way errors propagate, the decoder must be designed to be robust against channel noise. Error-correcting codes are often used to reduce the effect of channel noise.

Bibliography

- [1] M. A. Tekalp. *Digital video processing*. New York: Prentice-Hall, 1996.
- [2] J. S. Lim. *Two-dimensional signal and image processing*. New York: Prentice-Hall, 1990. ISBN: 978-0139353222.
- [3] J. Max. “Quantizing for minimum distortion”. In: *IEEE Trans. Info. Theory* 6.1 (Mar. 1960), 7–12.
- [4] S. P. Lloyd. “Least squares quantization in PCM”. In: *IEEE Trans. Info. Theory* 28.2 (Mar. 1982), 129–37.
- [5] R. M. Gray. “Quantization noise spectra”. In: *IEEE Trans. Info. Theory* 36.6 (Nov. 1990), 1220–44.
- [6] A. Gersho. “Asymptotically optimal block quantization”. In: *IEEE Trans. Info. Theory* 25.4 (July 1979), 373–80.
- [7] M. Aharon, M. Elad, and A. Bruckstein. “K-SVD: an algorithm for designing overcomplete dictionaries for sparse representation”. In: *IEEE Trans. Sig. Proc.* 54.11 (Nov. 2006), 4311–22.
- [8] Y. Linde, A. Buzo, and R. Gray. “An algorithm for vector quantizer design”. In: *IEEE Trans. Comm.* 84-95.28 (Jan. 1980), p. 1.
- [9] M. Thorpe, F. Theil, A. M. Johansen, and N. Cade. “Convergence of the k -means minimization problem using Γ -convergence”. In: *SIAM J. Appl. Math.* 75.6 (2015), 2444–74.
- [10] D. Arthur and S. Vassilvitskii. “K-means++: The advantages of careful seeding”. In: *Proc. 18th Annual ACM-SIAM Symp. Disc. Alg. (SODA)*. 2007, 1027–35.
- [11] O. Bachem, M. Lucic, S. H. Hassani, and A. Krause. “Approximate K-means++ in sublinear time”. In: *Conf. on Artificial Intelligence (AAAI)*. 2016.
- [12] E. C. Chi and K. Lange. “Splitting methods for convex clustering”. In: *J. Computational and Graphical Stat.* 24.4 (2015), 994–1013.
- [13] V. J. Mathews and P. J. Hahn. “Vector quantization using the L_∞ distortion measure”. In: *IEEE Signal Proc. Letters* 4.2 (Feb. 1997), 33–35.

- [14] A. Cohen, I. Daubechies, O. G. Guleryuz, and M. T. Orchard. “On the importance of combining wavelet-based nonlinear approximation with coding strategies”. In: *IEEE Trans. Info. Theory* 48.7 (July 2002), 1895–1921.
- [15] A. Gyorgy and T. Linder. “On the structure of optimal entropy-constrained scalar quantizers”. In: *IEEE Trans. Info. Theory* 48.2 (Feb. 2002), 416–27.
- [16] P. A. Chou, T. Lookabaugh, and R. M. Gray. “Entropy-constrained vector quantization”. In: *IEEE Trans. Acoust. Sp. Sig. Proc.* 37.1 (Jan. 1989), 31–42.
- [17] L. Roberts. “Picture coding using pseudo-random noise”. In: *Information Theory, IRE Transactions on* 8.2 (Feb. 1962), 145–54.
- [18] I. T. Jolliffe. *Principal component analysis*. Springer, 2002.
- [19] R. Vidal, Y. Ma, and S. Sastry. “Generalized principal component analysis (GPCA)”. In: *IEEE Trans. Patt. Anal. Mach. Int.* 27.12 (Dec. 2005), 1945–59.
- [20] M. O. Ulfarsson and V. Solo. “Vector l_0 sparse variable PCA”. In: *IEEE Trans. Sig. Proc.* 59.5 (May 2011), 1949–58.
- [21] C. Eckart and G. Young. “The approximation of one matrix by another of lower rank”. In: *Psychometrika* 1.3 (1936), 211–8.
- [22] G. Golub, A. Hoffman, and G. Stewart. “A generalization of the Eckart-Young-Mirsky matrix approximation theorem”. In: *Linear Algebra and its Applications* 88 (Apr. 1987), 317–27.
- [23] E. J. Candès, C. A. Sing-Long, and J. D. Trzasko. “Unbiased risk estimates for singular value thresholding and spectral estimators”. In: *IEEE Trans. Sig. Proc.* 61.19 (Oct. 2013), 4643–57.
- [24] B. A. Olshausen and D. J. Field. “Emergence of simple-cell receptive field properties by learning a sparse code for natural images”. In: *Nature* 381 (June 1996), 607–9.
- [25] G. K. Wallace. “The JPEG still picture compression standard”. In: *IEEE Trans. Consumer Electronics* 38.1 (Feb. 1992), xviii–xxxiv.