Lecture 7:
More Math +
Image Filtering

Administrative

HW0 was due yesterday!

HW1 due a week from yesterday

Cool Talk Today:

AI Seminar

NumPy: A look at the past, present, and future of array computation

Ross Barnowski
Postdoctoral Scholar

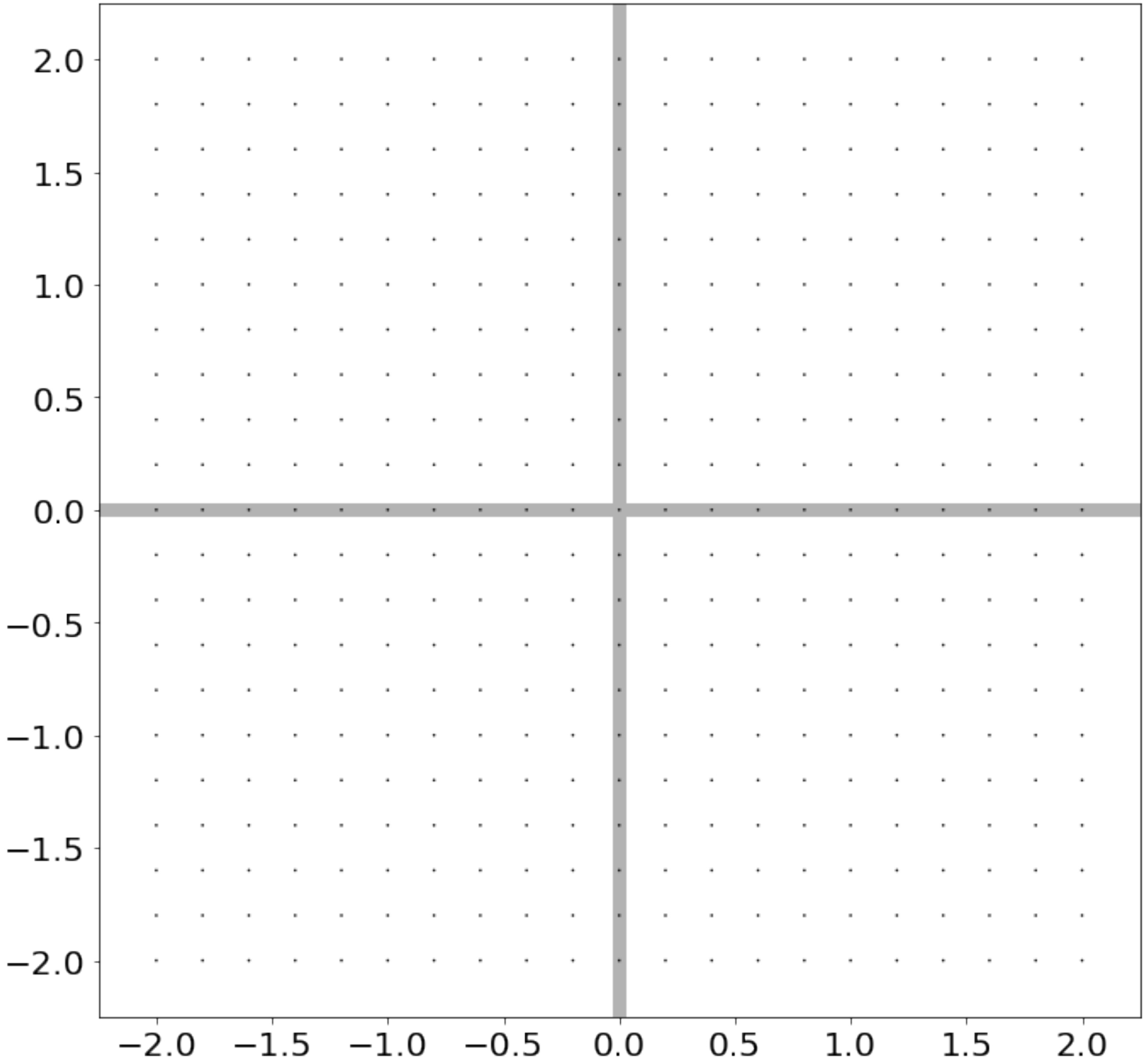
WHERE:	3725 Beyster Building
	MAP
WHEN:	January 30, 2020 @ 1:30 pm - 3:00 pm
	This event is free and open to the public
	ADD TO GOOGLE CALENDAR

https://cse.engin.umich.edu/event/numpy-a-look-at-the-past-present-and-future-of-array-computation

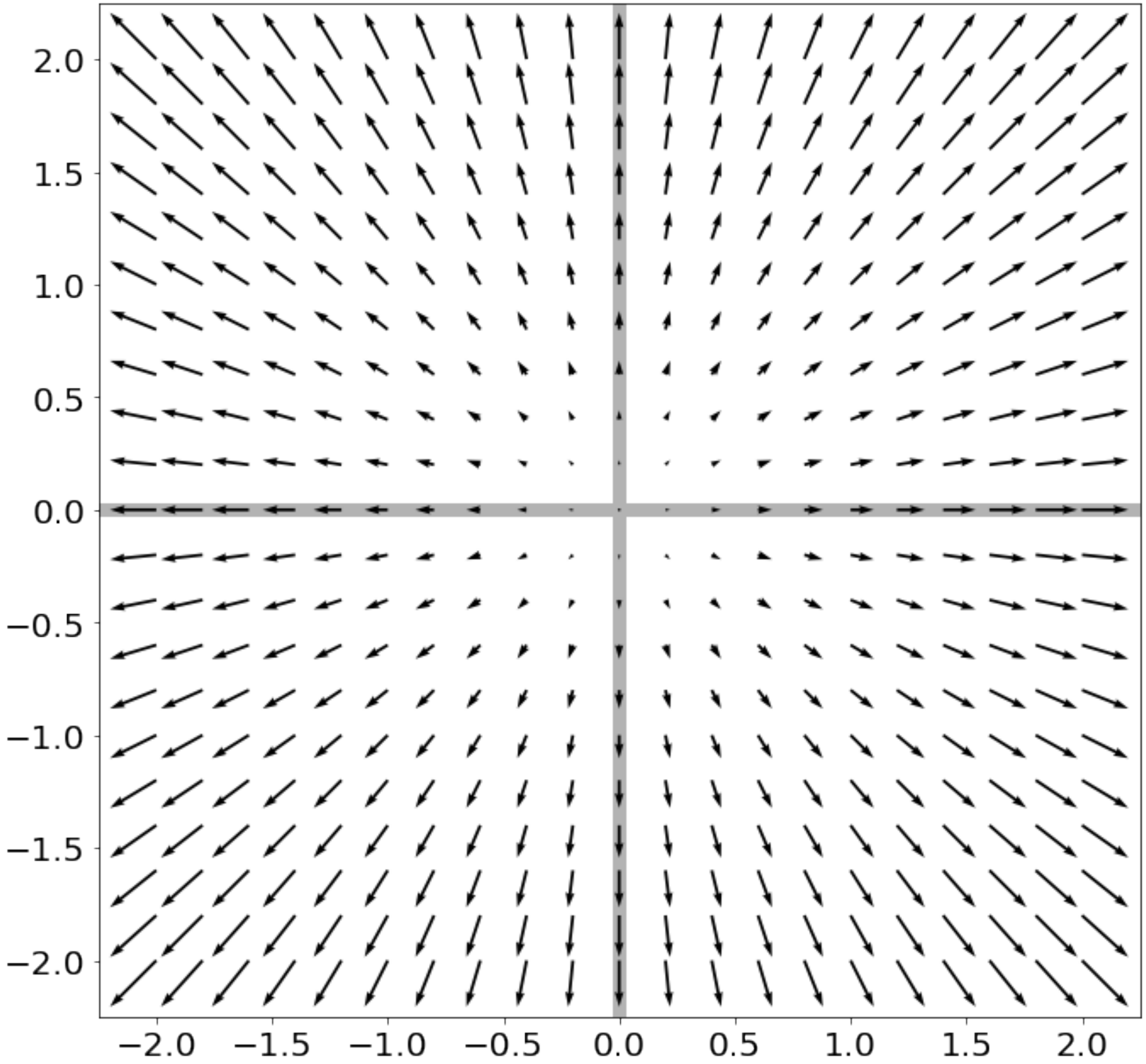
Last Time:
Matrices, Vectorization, Linear Algebra

Eigensystems

- An eigenvector $\boldsymbol{v_i}$ and eigenvalue λ_i of a matrix $\boldsymbol{A}$ satisfy $\boldsymbol{Av_i} = \lambda_i \boldsymbol{v_i}$ ($\boldsymbol{Av_i}$ is scaled by λ_i)
- Vectors and values are always paired and typically you assume $||\boldsymbol{v_i}||^2 = 1$
- Biggest eigenvalue of A gives bounds on how much $f(\boldsymbol{x}) = \boldsymbol{Ax}$ stretches a vector **x**.
- Hints of what people really mean:
  - "Largest eigenvector" = vector w/ largest value
  - "Spectral" just means there's eigenvectors somewhere



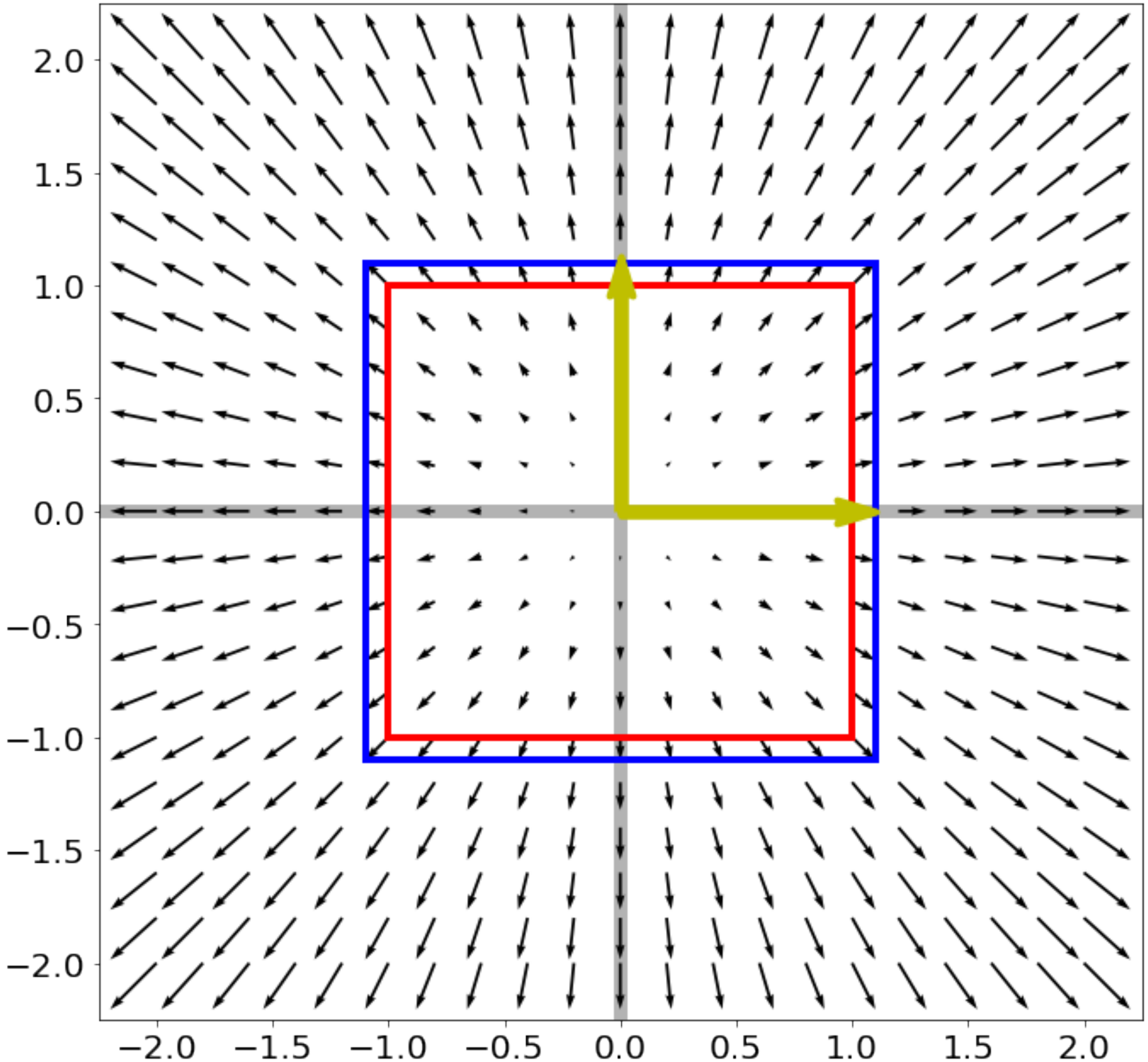
Suppose I have points in a grid



Now I apply f(**x**) = **Ax** to these points

Pointy-end: **Ax** . Non-Pointy-End: **x**

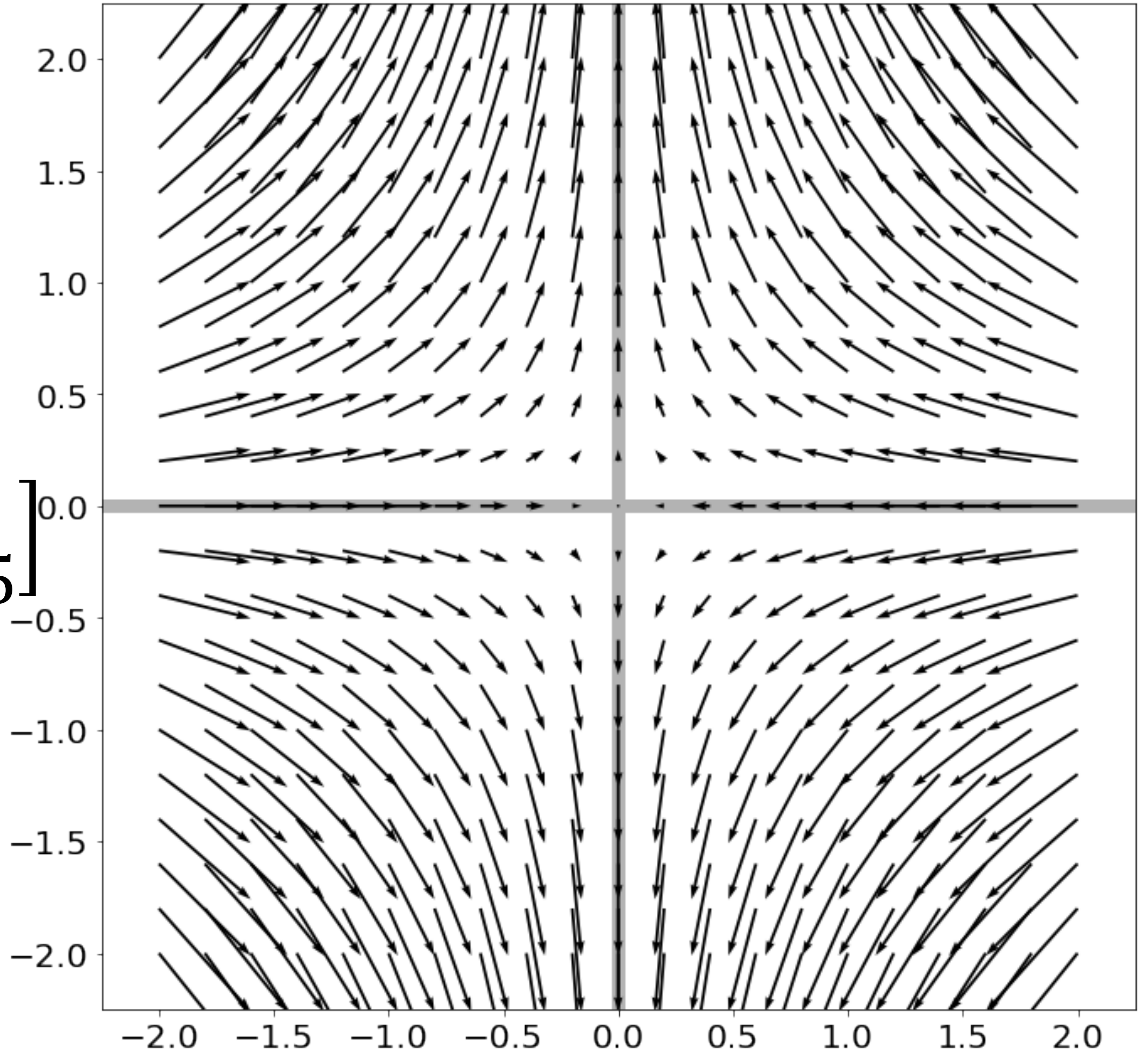
$$A = \begin{bmatrix} 1.1 & 0 \\ 0 & 1.1 \end{bmatrix}$$



Red box – unit square, Blue box – after f(**x**) = **Ax**.

What are the yellow lines and why?

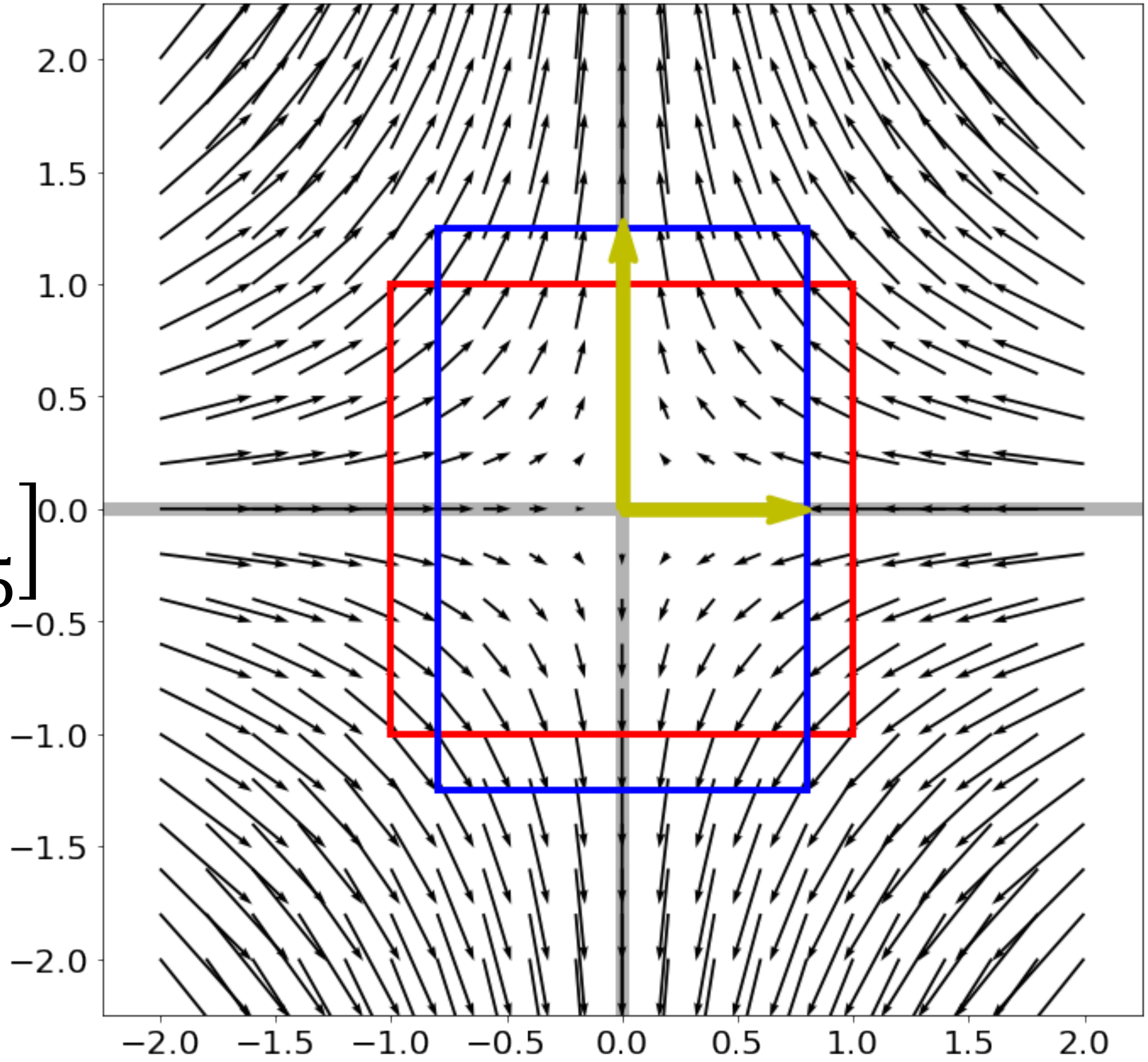
$$A = \begin{bmatrix} 0.8 & 0 \\ 0 & 1.25 \end{bmatrix}$$



Now I apply f(**x**) = **Ax** to these points

Pointy-end: **Ax** . Non-Pointy-End: **x**

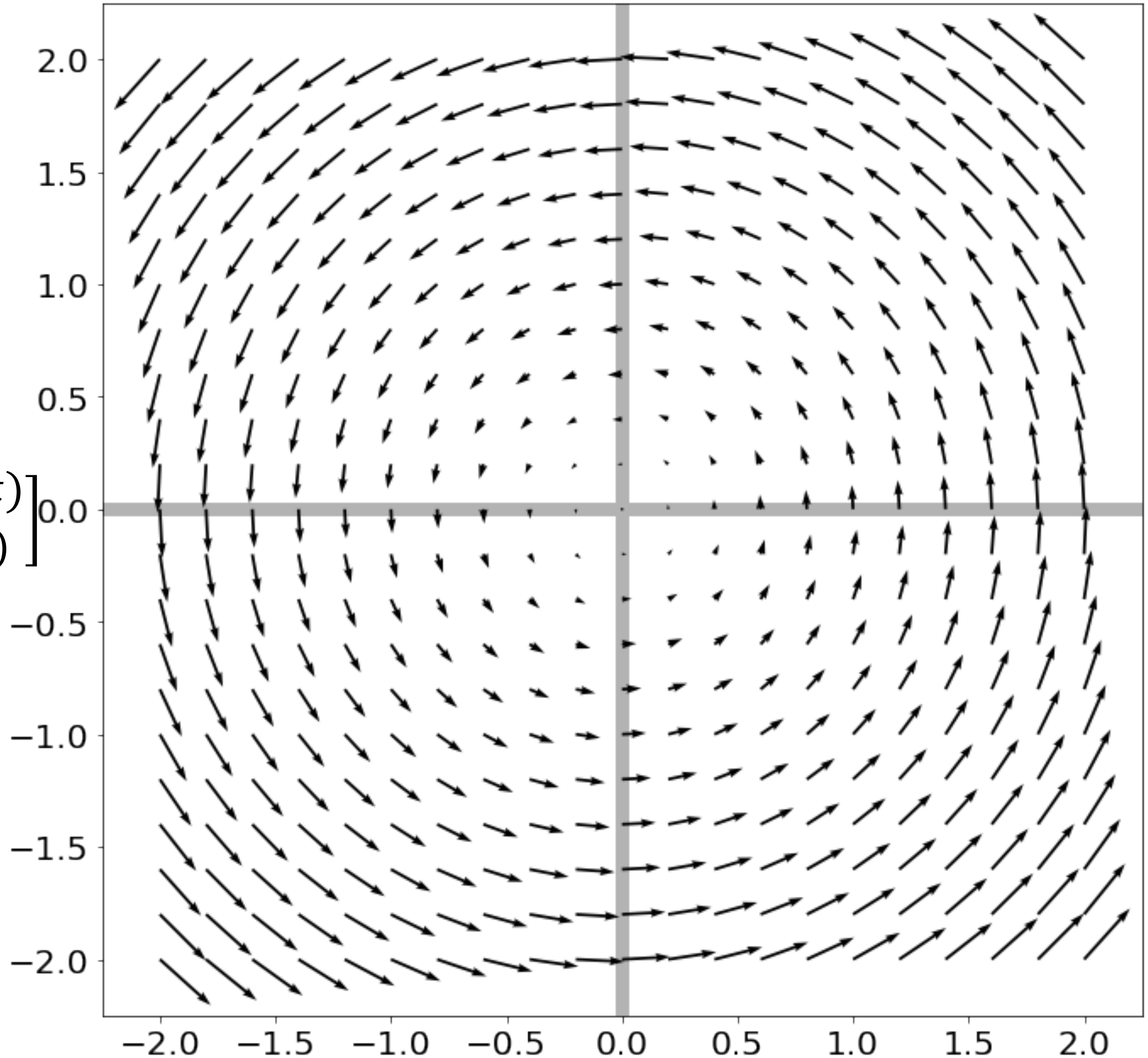
$$A = \begin{bmatrix} 0.8 & 0 \\ 0 & 1.25 \end{bmatrix}$$



Red box – unit square, Blue box – after f(**x**) = **Ax**.

What are the yellow lines and why?

$$A = \begin{bmatrix} \cos(t) & -\sin(t) \\ \sin(t) & \cos(t) \end{bmatrix}$$



Red box – unit square, Blue box – after f(**x**) = **Ax**.

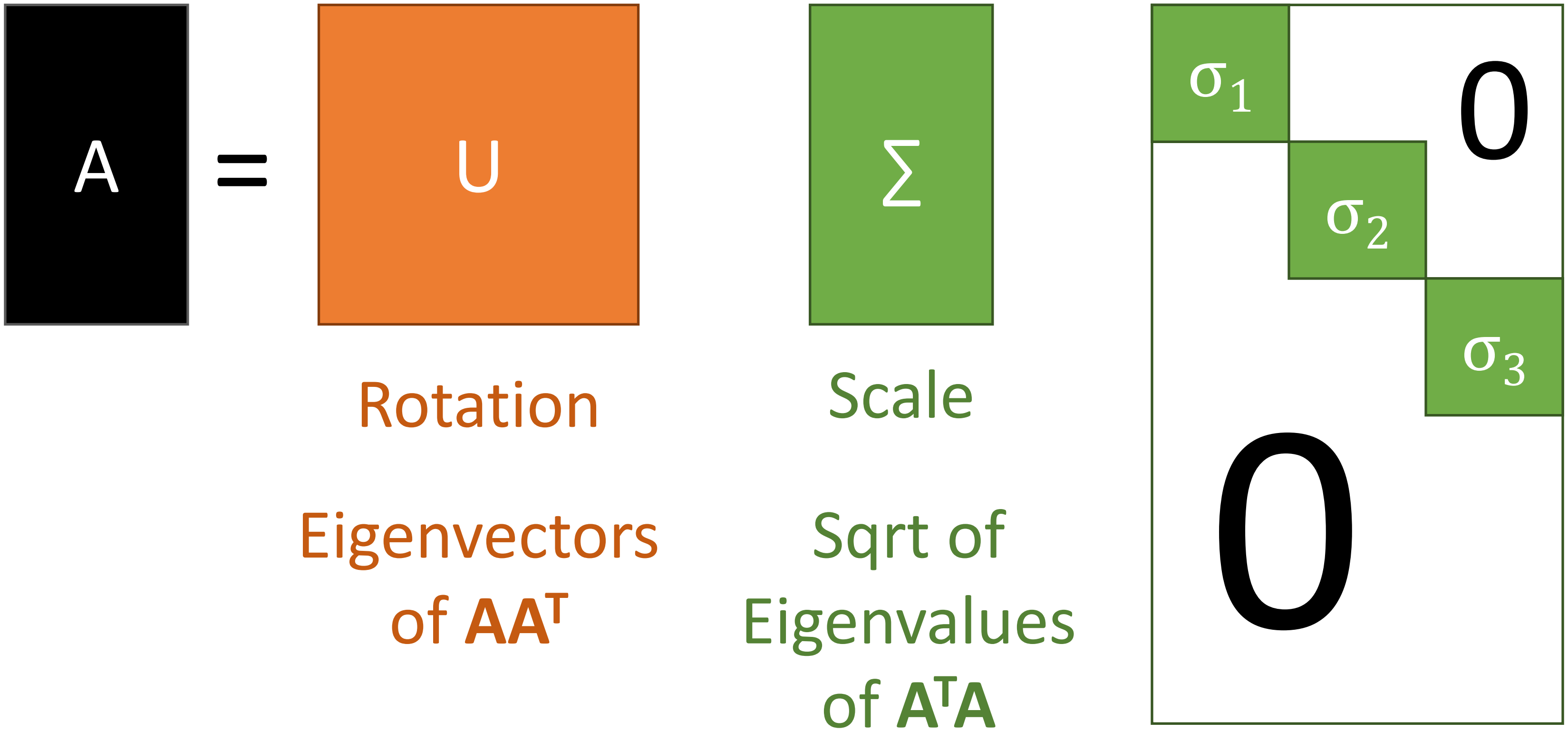
Can we draw any yellow lines?

Eigenvectors of Symmetric Matrices

- Always n mutually orthogonal eigenvectors with n (not necessarily) distinct eigenvalues
- For symmetric $\boldsymbol{A}$, the eigenvector with the largest eigenvalue maximizes $\frac{\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}}$ (smallest/min)
- So for unit vectors (where $\boldsymbol{x}^T \boldsymbol{x} = 1$), that eigenvector maximizes $\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}$
- A surprisingly large number of optimization problems rely on (max/min)imizing this

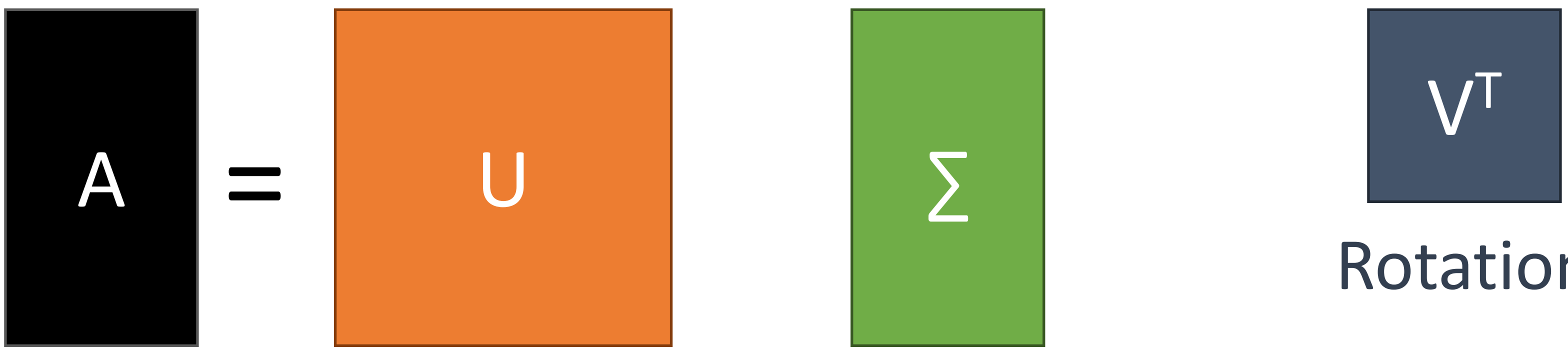
Singular Value Decomposition

Can **always** write a mxn matrix **A** as: $\boldsymbol{A} = \boldsymbol{U\Sigma V^T}$



Singular Value Decomposition

Can **always** write a mxn matrix **A** as: $\boldsymbol{A} = \boldsymbol{U \Sigma V^T}$



Rotation

Rotation

Scale

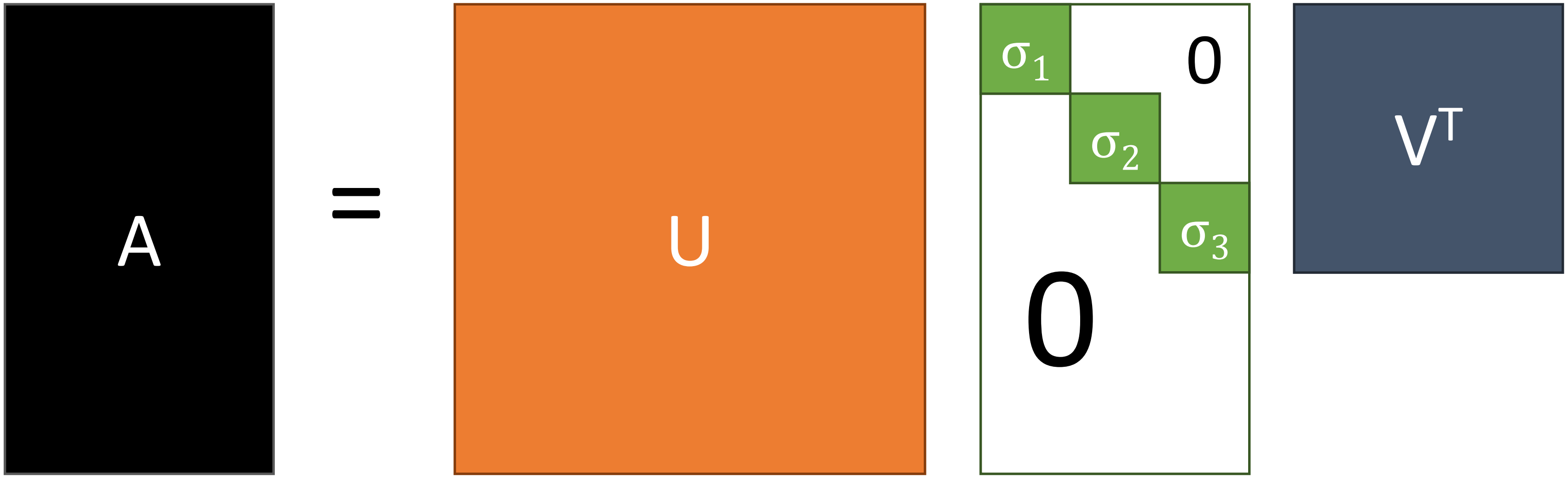
Eigenvectors of **AA^T**

Sqrt of Eigenvalues of **A^{T}A**

Eigenvectors of **A^{T}A**

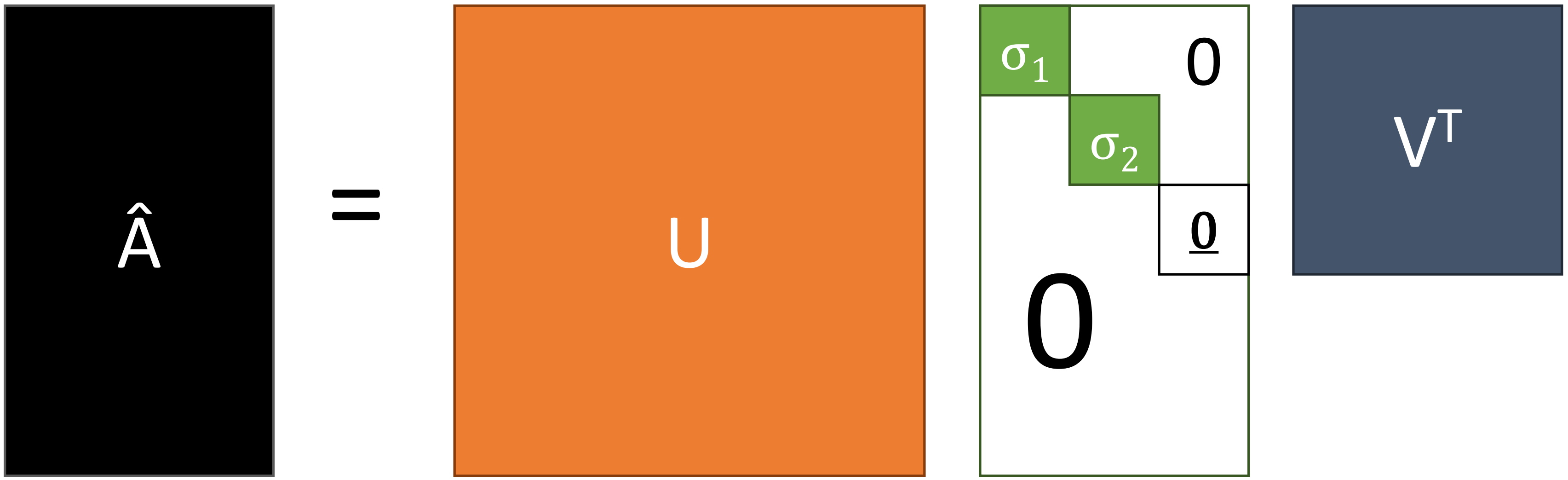
Singular Value Decomposition

- *Every* matrix is a rotation, scaling, and rotation
- Number of non-zero singular values = rank / number of linearly independent vectors
- "Closest" matrix to **A** with a lower rank



Singular Value Decomposition

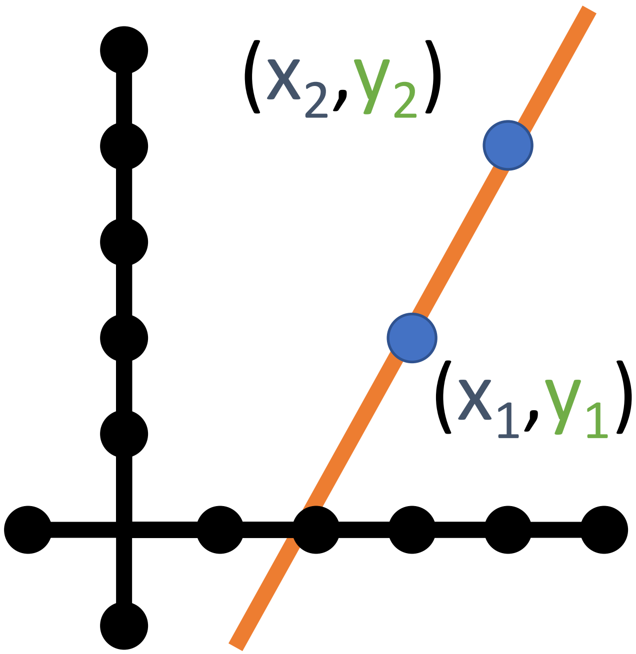
- *Every* matrix is a rotation, scaling, and rotation
- Number of non-zero singular values = rank / number of linearly independent vectors
- "Closest" matrix to **A** with a lower rank



Singular Value Decomposition

- *Every* matrix is a rotation, scaling, and rotation
- Number of non-zero singular values = rank / number of linearly independent vectors
- "Closest" matrix to **A** with a lower rank
- Secretly behind basically many things you do with matrices

Solving Least-Squares



Start with two points (x_i,y_i)

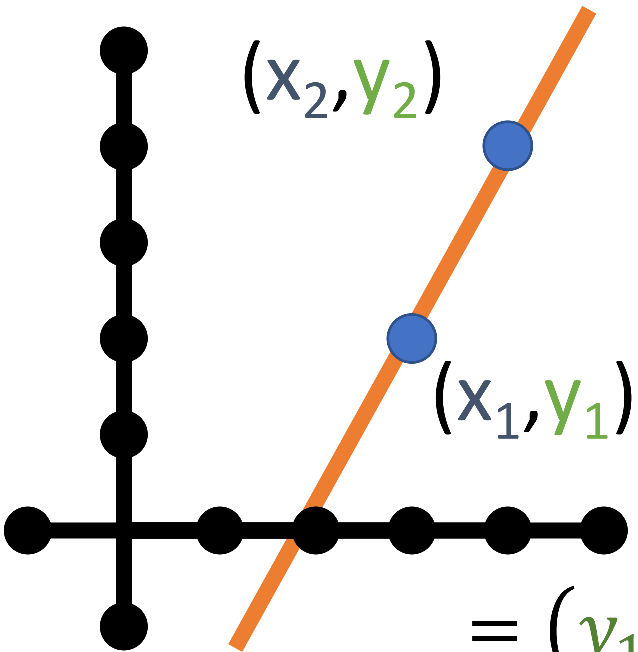
$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{v}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} mx_1 + b \\ mx_2 + b \end{bmatrix}$$

We know how to solve this – invert **A** and find **v** (i.e., (m,b) that fits points)

Solving Least-Squares



Start with two points (x_i, y_i)

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{v}$$

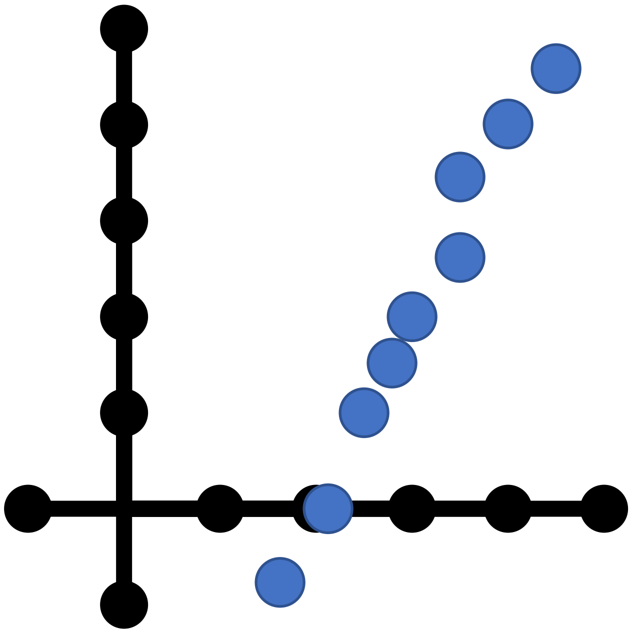
$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix}$$

$$\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{v}\|^2 = \left\| \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} - \begin{bmatrix} mx_1 + b \\ mx_2 + b \end{bmatrix} \right\|^2$$

$$= \left(y_1 - (\underline{mx_1 + b})\right)^2 + \left(y_2 - (\underline{mx_2 + b})\right)^2$$

The sum of squared differences between
the actual value of y and
what the model says y should be.

Solving Least-Squares



Suppose there are n > 2 points

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{v}$$

$$\begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_N & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix}$$

Compute $||y - Ax||^2$ again

$$||\boldsymbol{y} - \boldsymbol{A}\boldsymbol{v}||^2 = \sum_{i=1}^{n} (y_i - (mx_i + b))^2$$

Solving Least-Squares

Given **y**, **A**, and **v** with **y** = **Av** overdetermined

(**A** tall / more equations than unknowns)

We want to minimize $||\boldsymbol{y} - \boldsymbol{A}\boldsymbol{v}||^2$, or find:

$$\boxed{\arg\min_{\boldsymbol{v}}} ||\boldsymbol{y} - \boldsymbol{A}\boldsymbol{v}||^2$$

(The value of x that makes the expression smallest)

Solution satisfies $\left(\boldsymbol{A}^T\boldsymbol{A}\right)\boldsymbol{v}^* = \boldsymbol{A}^T\boldsymbol{y}$

<u>or</u>

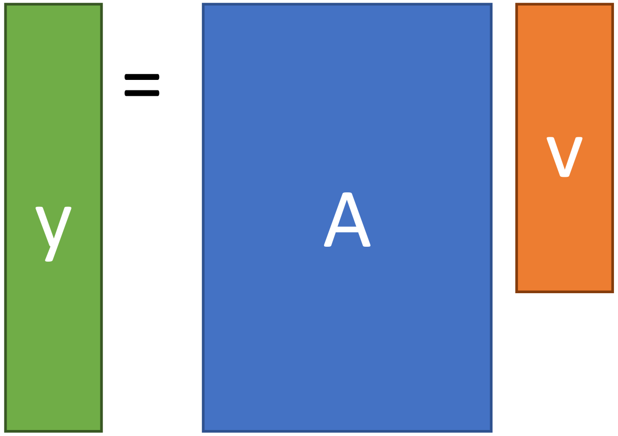
$$\boldsymbol{v}^* = \left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T\boldsymbol{y}$$

(Don't actually compute the inverse!)

When is Least-Squares Possible?

Given **y**, **A**, and **v**. Want **y** = **Av**

Want n outputs, have n knobs to fiddle with, every knob is useful if A is full rank.

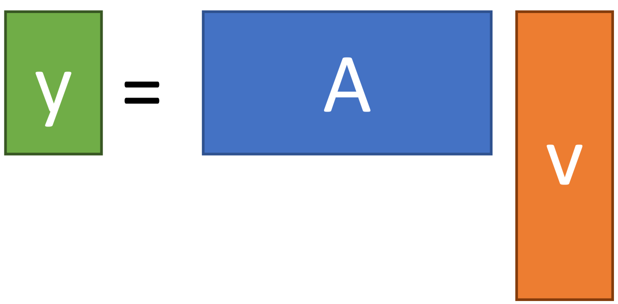


A: rows (outputs) > columns (knobs). Thus can't get precise output you want (not enough knobs). So settle for "closest" knob setting.

When is Least-Squares Possible?

Given **y**, **A**, and **v**. Want **y** = **Av**

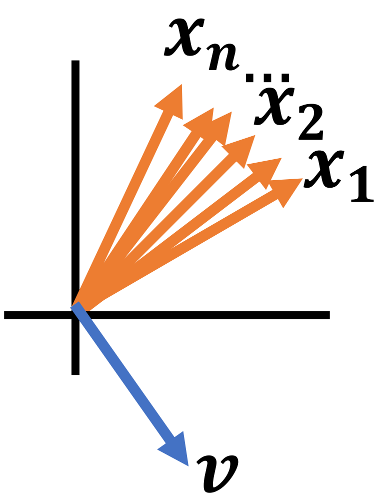
Want n outputs, have n knobs to fiddle with, every knob is useful if A is full rank.



A: columns (knobs) > rows (outputs). Thus, any output can be expressed in infinite ways.

Homogeneous Least-Squares

Given a set of unit vectors (aka directions) $x_1, \ldots, x_n$ and I want vector v that is as orthogonal to all the x_i as possible (for some definition of orthogonal)



Stack x_i into **A**, compute **Av**

$$Av = \begin{bmatrix} - & x_1^T & - \\ & \vdots & \\ - & x_n^T & - \end{bmatrix} v = \begin{bmatrix} x_1^T v \\ \vdots \\ x_n^T v \end{bmatrix}$$

0 if orthog

Compute $\|Av\|^2 = \boxed{\sum_i^n \left(x_i^T v\right)^2}$

Sum of how orthog. v is to each x

Homogenous Least-Squares

- A lot of times, given a matrix **A** we want to find the **v** that minimizes $||\boldsymbol{Av}||^2$.
- I.e., want $\mathbf{v}^* = \arg\min_{\boldsymbol{v}} ||\boldsymbol{Av}||_2^2$
- What's a trivial solution?
- Set **v = 0** $\rightarrow$ **Av = 0**
- Exclude this by forcing v to have unit norm

Homogenous Least-Squares

Let's look at $||\boldsymbol{Av}||_2^2$

$||\boldsymbol{Av}||_2^2 =$ Rewrite as dot product

$||\boldsymbol{Av}||_2^2 = (\mathbf{Av})^{\mathrm{T}}(\mathbf{Av})$ Distribute transpose

$||\boldsymbol{Av}||_2^2 = \boldsymbol{v}^{\boldsymbol{T}}\boldsymbol{A}^{\boldsymbol{T}}\mathbf{Av} = \mathbf{v}^{\mathrm{T}}\left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)\mathbf{v}$

We want the vector minimizing this quadratic form

Where have we seen this?

Homogenous Least-Squares

Ubiquitious tool in vision:

$$\arg\min_{\|\boldsymbol{v}\|^2=1} \|\boldsymbol{A}\boldsymbol{v}\|^2$$

→ **(1)** "Smallest"* eigenvector of $\boldsymbol{A}^T\boldsymbol{A}$

(2) **"**Smallest" right singular vector of $\boldsymbol{A}$

For min → max, switch smallest → largest

*Note: $\boldsymbol{A}^T\boldsymbol{A}$ is positive semi-definite so it has all non-negative eigenvalues

Derivatives

Derivatives

Remember derivatives?

Derivative: rate at which a function f(x) changes at a point as well as the direction that increases the function

Given quadratic function f(x)

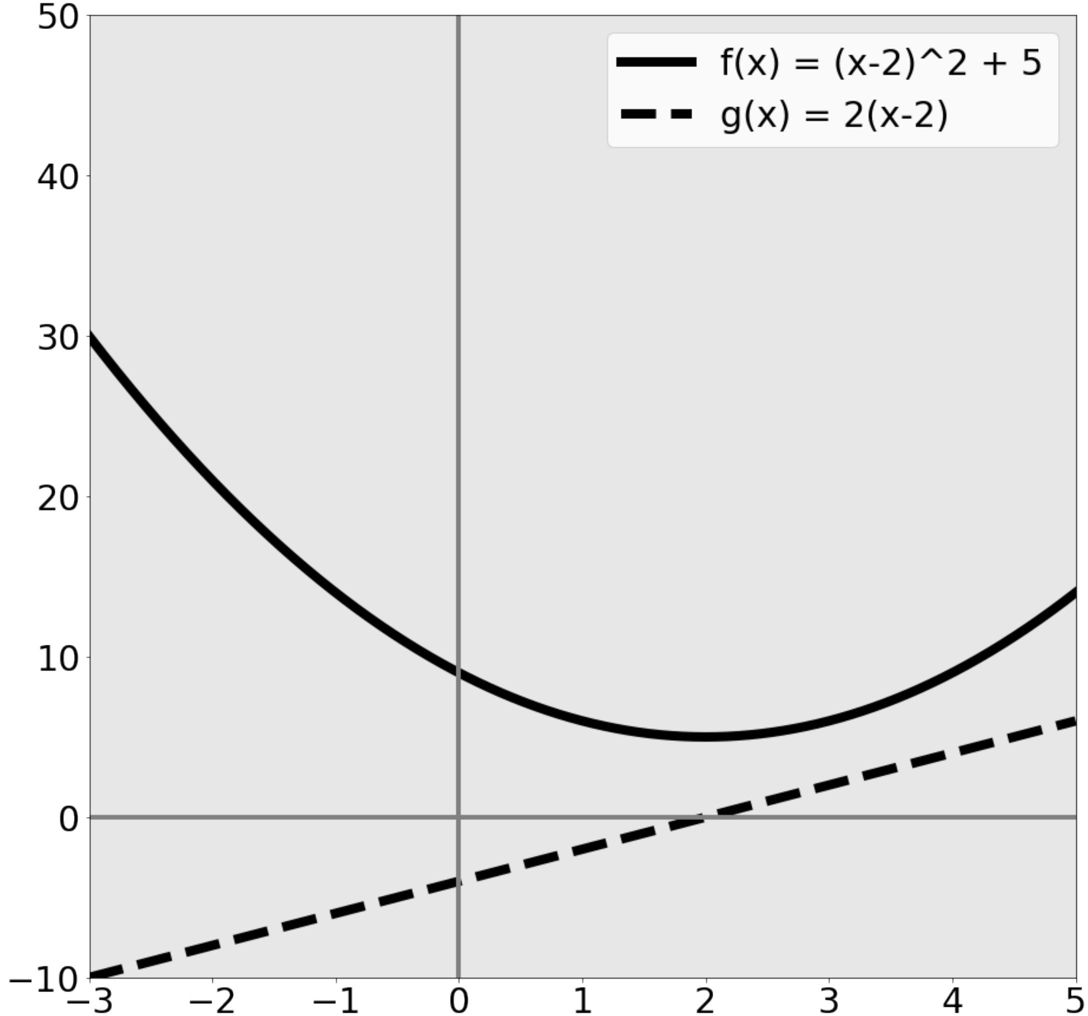
$$f(x,y) = (x-2)^2 + 5$$

$f(x)$ is function

$$g(x) = f'(x)$$

aka

$$g(x) = \frac{d}{dx} f(x)$$



Given quadratic function f(x)

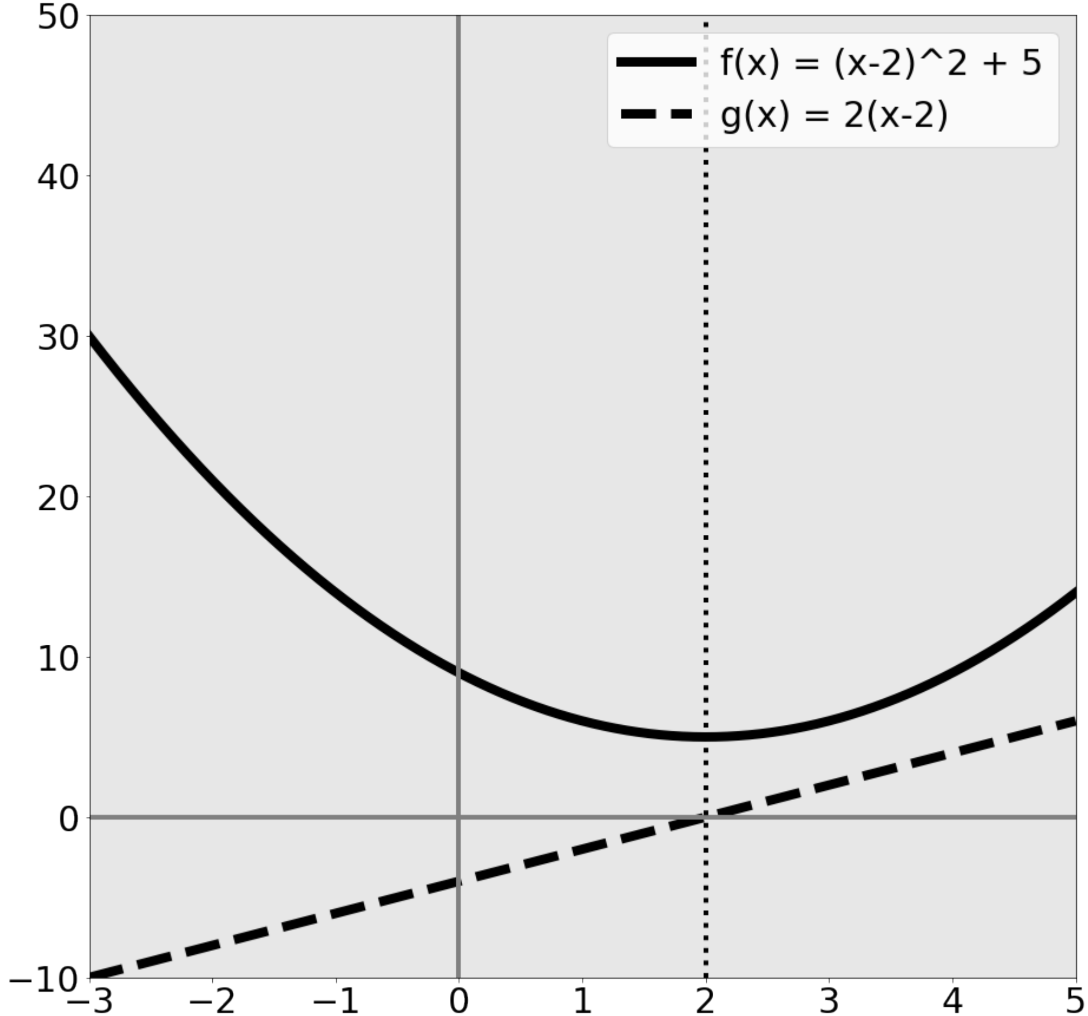
$$f(x, y) = (x-2)^2 + 5$$

What's special about x=2?

$f(x)$ minim. at 2
$g(x) = 0$ at 2

a = minimum of f $\rightarrow$
$g(a) = 0$

Reverse is ***<u>not true</u>***



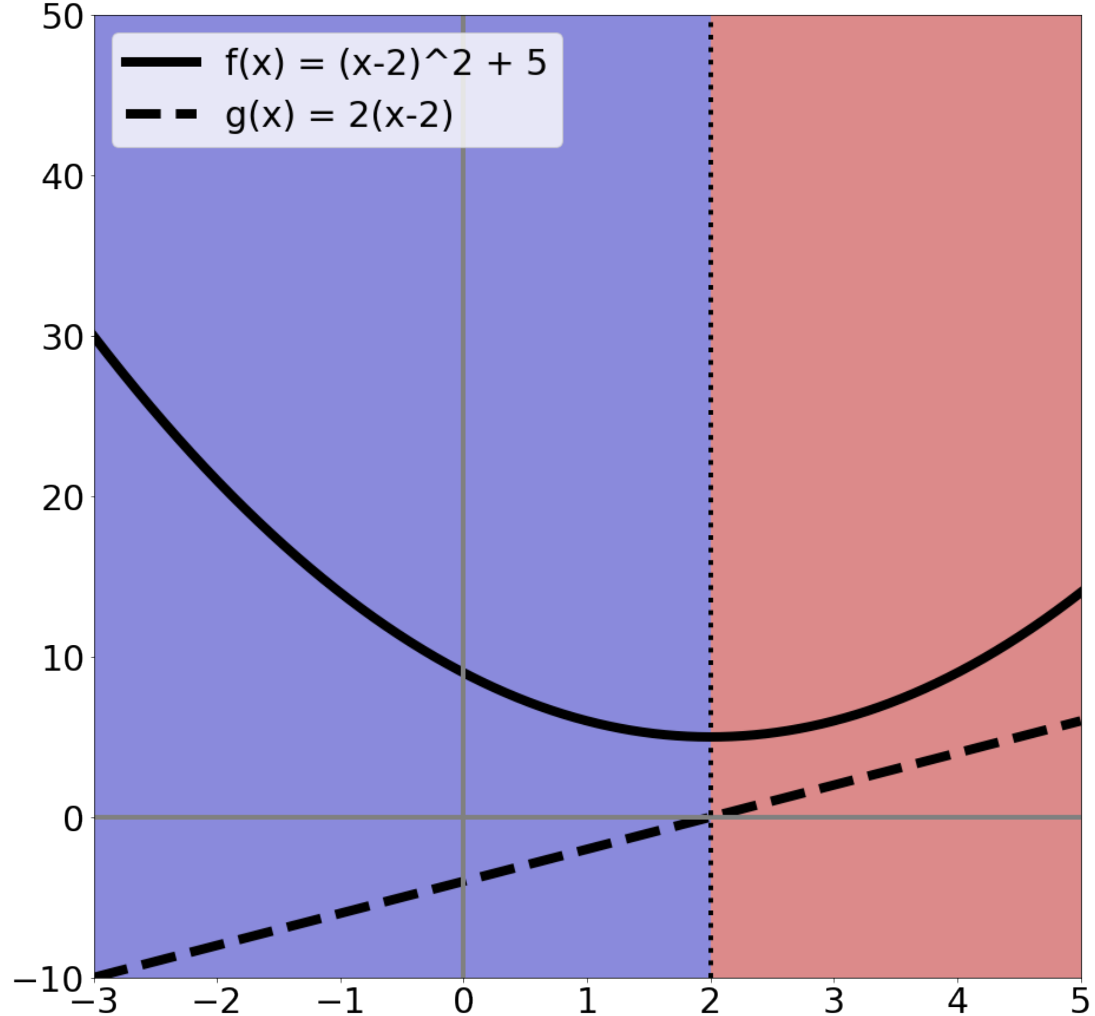
Rates of change

$$f(x, y) = (x-2)^2 + 5$$

Suppose I want to increase f(x) by changing x:

Blue area: move left
Red area: move right

Derivative tells you direction of ascent and rate



Calculus to Know

- Really need intuition
- Need chain rule
- Rest you should look up / use a computer algebra system / use a cookbook
- Partial derivatives (and that's it from multivariable calculus)

Partial Derivatives

- Pretend other variables are constant, take a derivative. That's it.
- Make our function a function of two variables

$$f(x) = (x-2)^2 + 5$$

$$\frac{\partial}{\partial x} f(x) = 2(x-2) * 1 = 2(x-2)$$

$$f_2(x,y) = (x-2)^2 + 5 + \boxed{(y+1)^2}$$

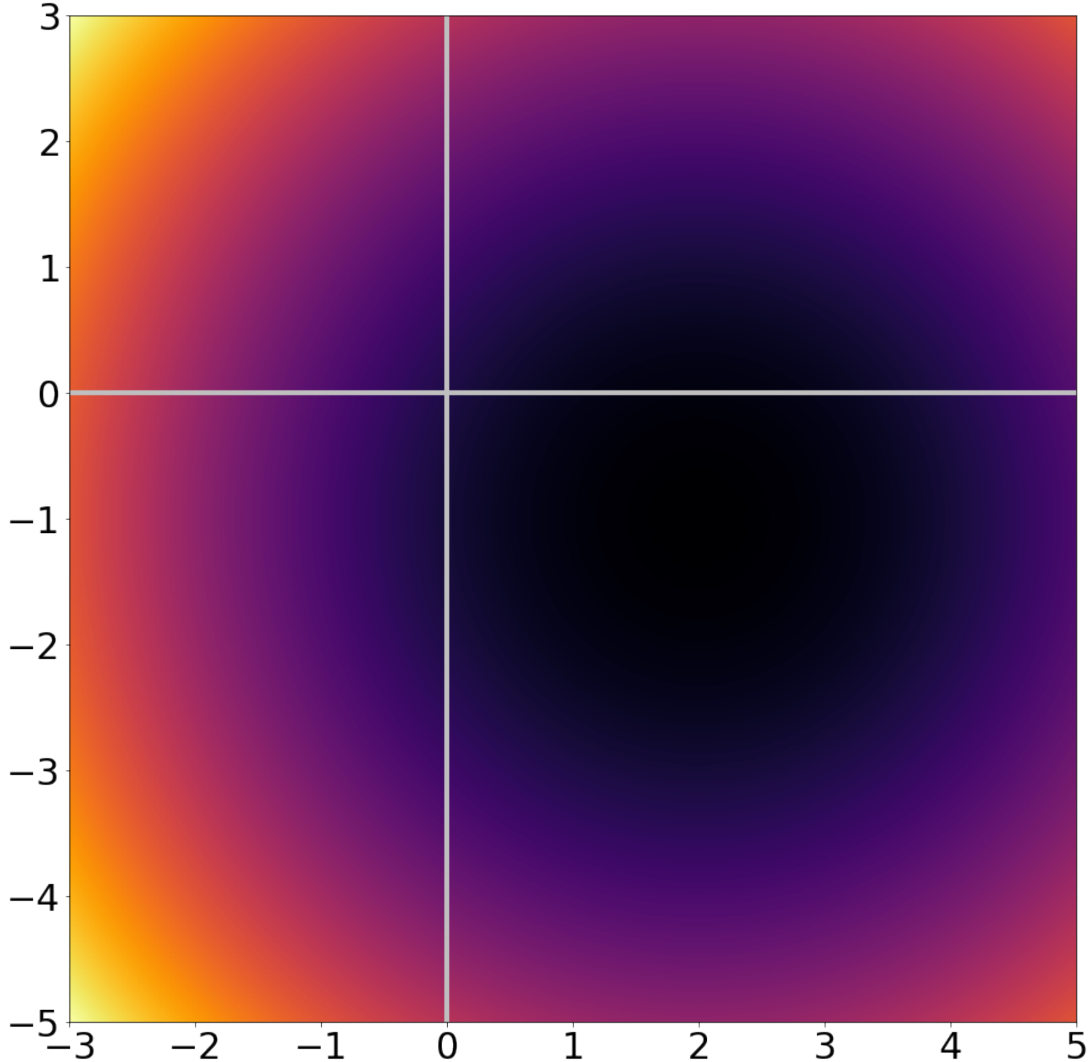
Pretend it's constant → derivative = 0

$$\frac{\partial}{\partial x} f_2(x) = 2(x-2)$$

Zooming Out

$$f_2(x, y) = (x - 2)^2 + 5 + (y + 1)^2$$

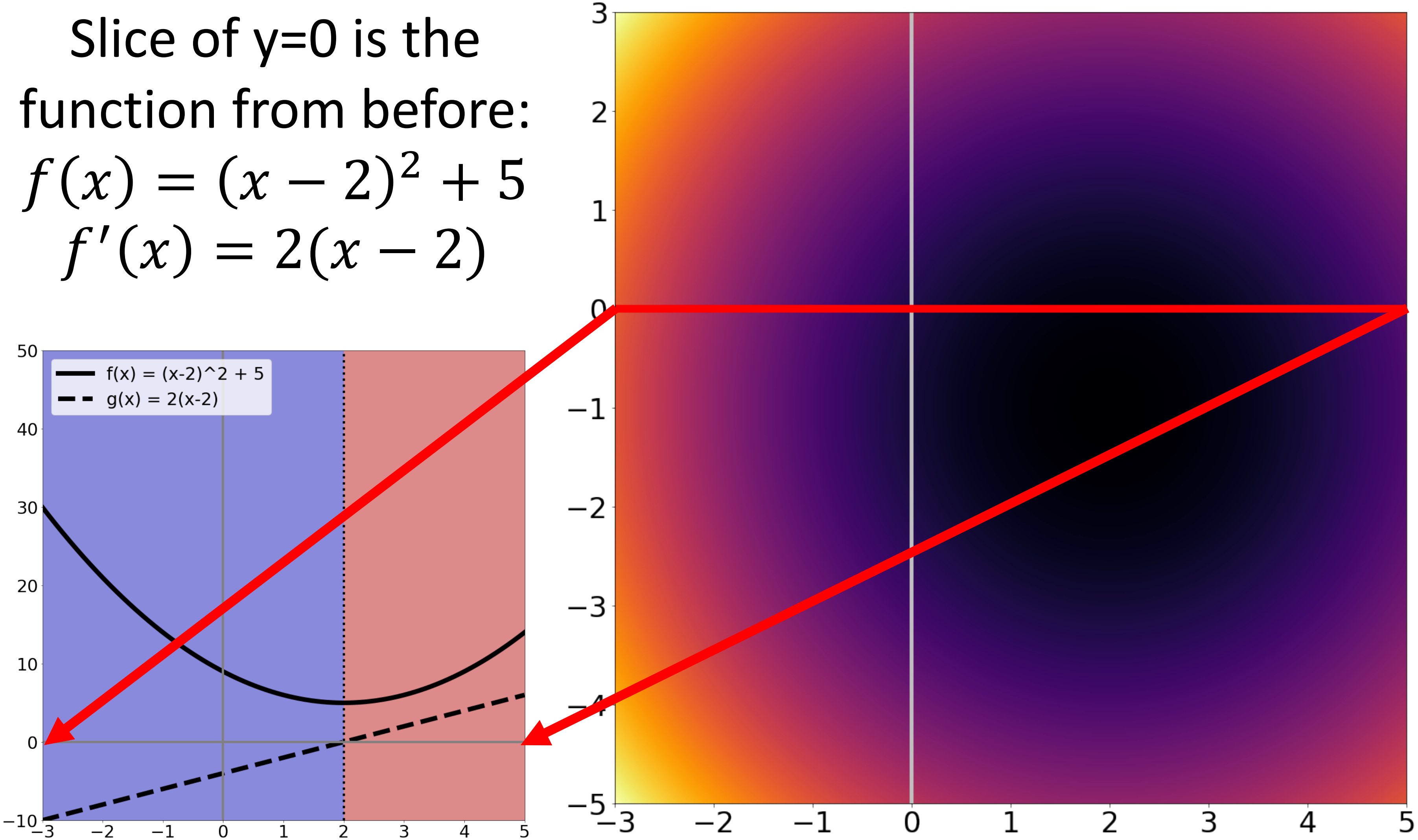
Dark = f(x,y) low
Bright = f(x,y) high



Taking a slice of
$$f_2(x, y) = (x-2)^2 + 5 + (y+1)^2$$

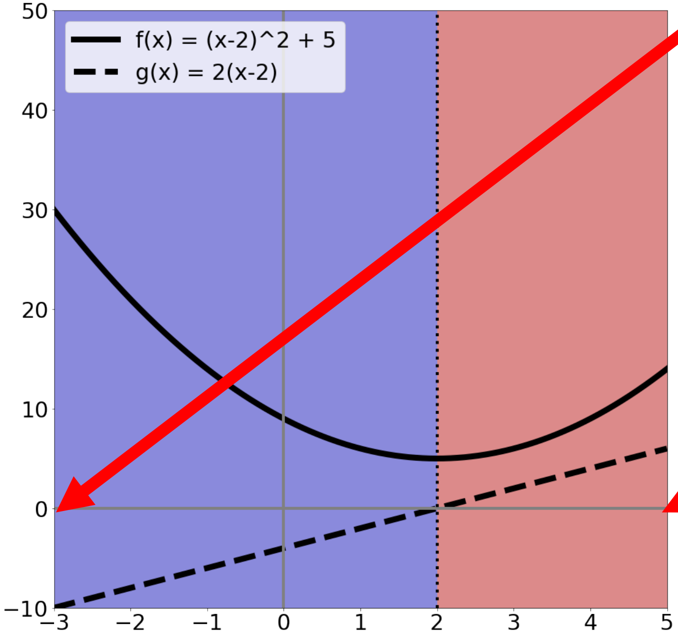
Slice of y=0 is the function from before:

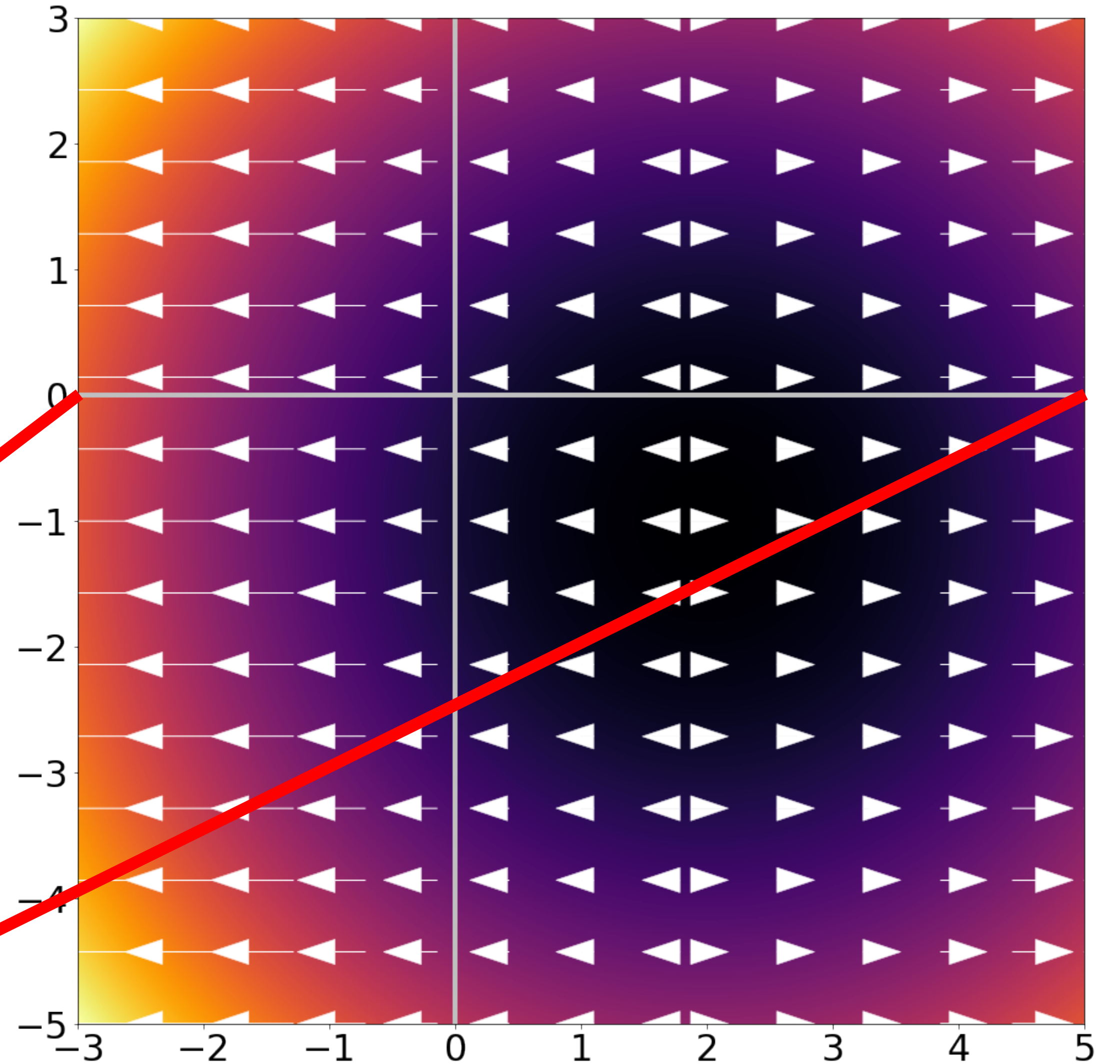
$$f(x) = (x-2)^2 + 5$$
$$f'(x) = 2(x-2)$$



Taking a slice of
$$f_2(x,y) = (x-2)^2 + 5 + (y+1)^2$$

$\frac{\partial}{\partial x} f_2(x,y)$ is rate of change & direction in x dimension





Zoming Out

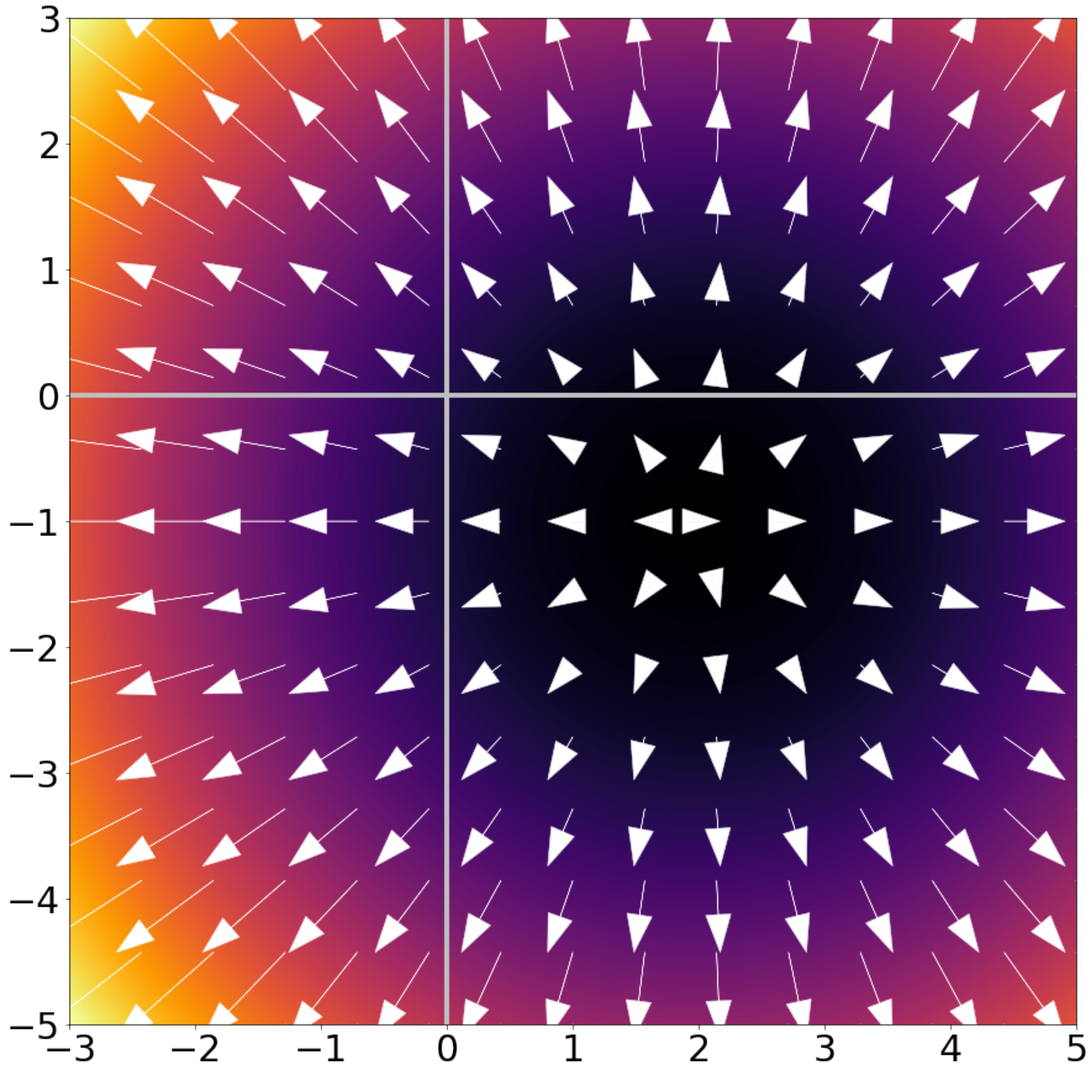
$$f_2(x,y) = (x-2)^2 + 5 + (y+1)^2$$

Gradient/Jacobian: Making a vector of

$$\nabla_f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

gives rate and direction of change.

Arrows point OUT of minimum / basin.

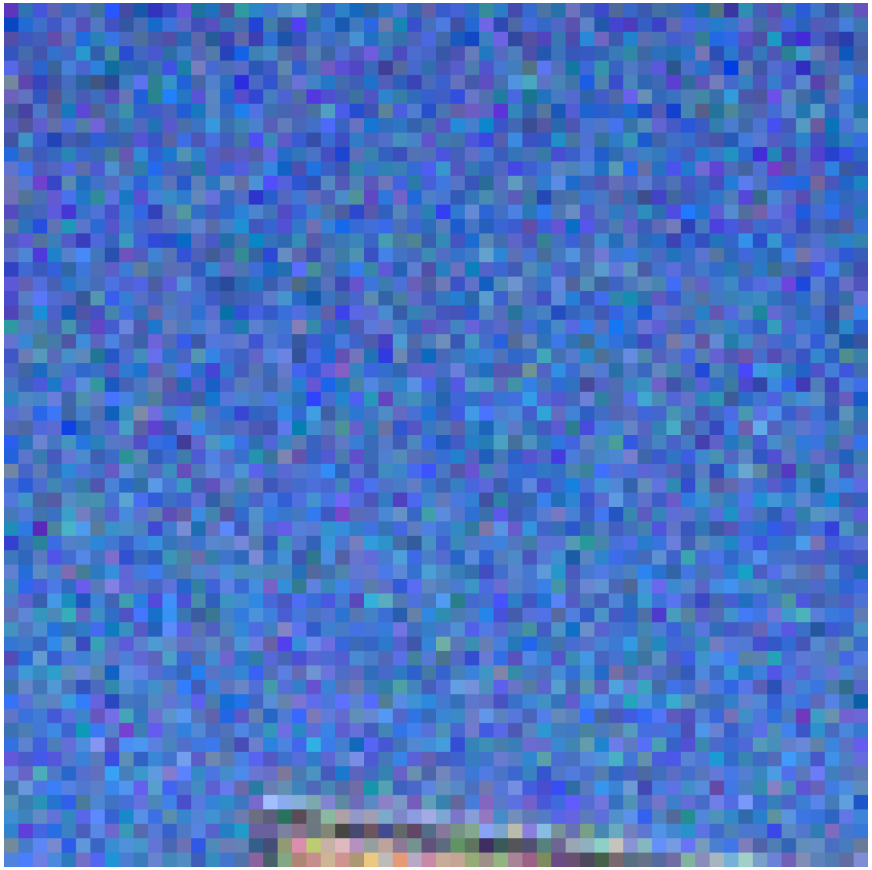


What Should I Know?

- Gradients are simply partial derivatives per-dimension: if $\boldsymbol{x}$ in $f(\boldsymbol{x})$ has n dimensions, $\nabla_f(x)$ has n dimensions
- Gradients point in direction of ascent and tell the rate of ascent
- If a is minimum of $f(\boldsymbol{x}) \rightarrow \nabla_{\mathrm{f}}(\mathrm{a}) = \mathbf{0}$
- Reverse is not true, especially in high-dimensional spaces

Image Filtering

A Noisy Image



Cleaning it up

- We have noise in our image
- Let’s replace each pixel with a *weighted* average of its neighborhood
- Weights are *filter kernel*

	Out	

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Slide Credit: D. Lowe

1D Case

Signal	10	12	9	11	10	11	12
Filter	1/3	1/3	1/3				
Output		10.33					

1D Case

Signal							
	10	12	9	11	10	11	12

Filter			
	1/3	1/3	1/3

Output		
	10.33	10.66

1D Case

Signal	10	12	9	11	10	11	12
Filter			1/3	1/3	1/3		
Output		10.33	10.66	10			

1D Case

Signal	10	12	9	11	10	11	12
Filter				1/3	1/3	1/3	
Output		10.33	10.66	10	10.66		

1D Case

Signal	10	12	9	11	10	11	12
Filter					1/3	1/3	1/3
Output		10.33	10.66	10	10.66	11	

Applying a 2D Filter

Input

I11	I12	I13	I14	I15	I16
I21	I22	I23	I24	I25	I26
I31	I32	I33	I34	I35	I36
I41	I42	I43	I44	I45	I46
I51	I52	I53	I54	I55	I56

Filter

F11	F12	F13
F21	F22	F23
F31	F32	F33

Output

O11	O12	O13	O14
O21	O22	O23	O24
O31	O32	O33	O34

Applying a 2D Filter

Input & Filter

F11	F12	F13	I14	I15	I16
F21	F22	F23	I24	I25	I26
F31	F32	F33	I34	I35	I36
I41	I42	I43	I44	I45	I46
I51	I52	I53	I54	I55	I56

Output

O11

O11 = I11*F11 + I12*F12 + … + I33*F33

Applying a 2D Filter

Input & Filter

I11	F11	F12	F13	I15	I16
I21	F21	F22	F23	I25	I26
I31	F31	F32	F33	I35	I36
I41	I42	I43	I44	I45	I46
I51	I52	I53	I54	I55	I56

Output

O11	O12

$$O12 = I12 * F11 + I13 * F12 + \ldots + I34 * F33$$

Applying a 2D Filter

Input

I11	I12	I13	I14	I15	I16
I21	I22	I23	I24	I25	I26
I31	I32	I33	I34	I35	I36
I41	I42	I43	I44	I45	I46
I51	I52	I53	I54	I55	I56

Filter

F11	F12	F13
F21	F22	F23
F31	F32	F33

Output

How many times can we apply a 3x3 filter to a 5x6 image?

Applying a 2D Filter

Input

I11	I12	I13	I14	I15	I16
I21	I22	I23	I24	I25	I26
I31	I32	I33	I34	I35	I36
I41	I42	I43	I44	I45	I46
I51	I52	I53	I54	I55	I56

Filter

F11	F12	F13
F21	F22	F23
F31	F32	F33

Output

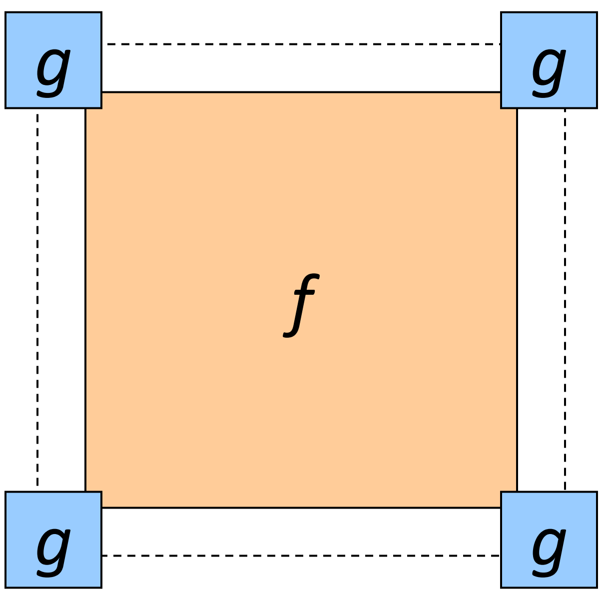
O11	O12	O13	O14
O21	O22	O23	O24
O31	O32	O33	O34

$$Oij = Iij*F11 + Ii(j+1)*F12 + \ldots + I(i+2)(j+2)*F33$$

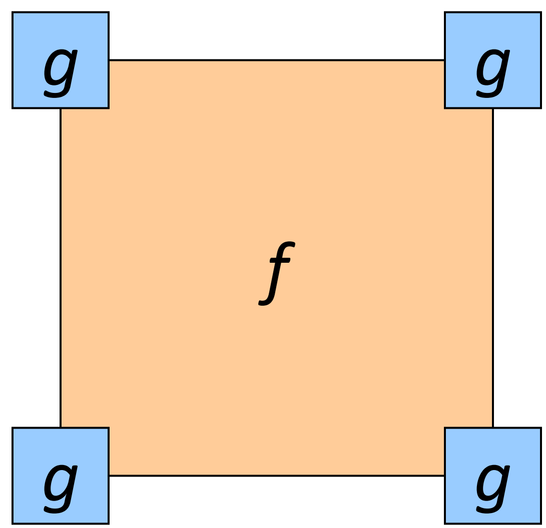
Edge Cases

Convolution doesn't keep the whole image.
Suppose **f** is the image and **g** the filter.

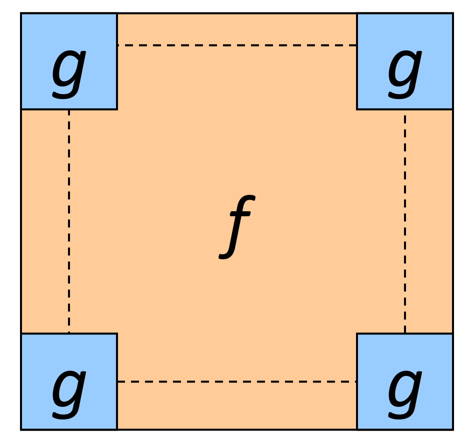
Full: Any part of g touches f.



Same: Output is same size as f



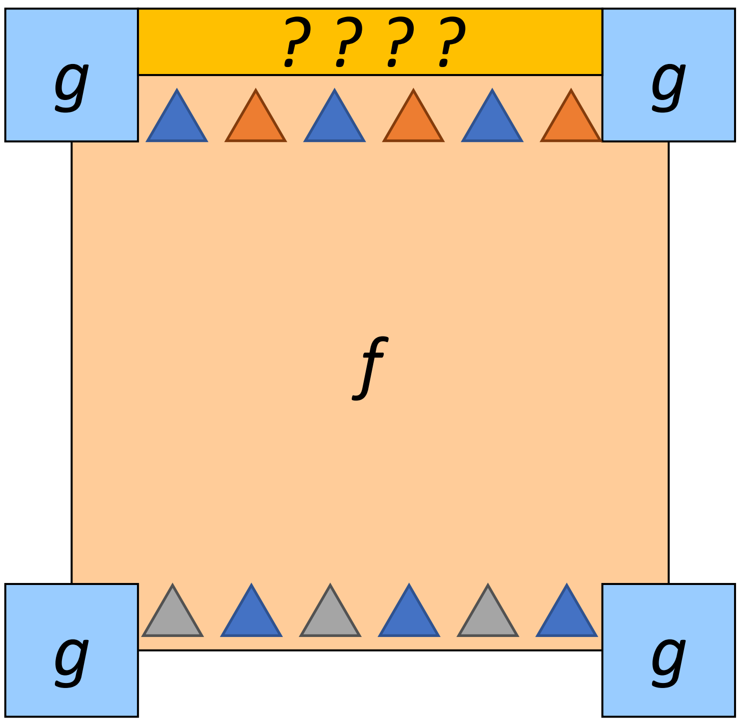
Valid: Filter doesn't fall off edge



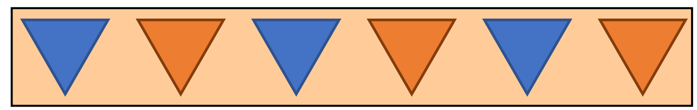
f/g Diagram Credit: D. Lowe

Edge Cases

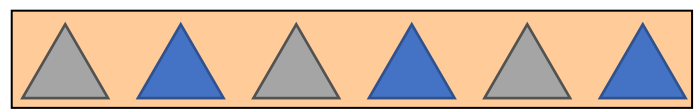
What to about the "?" region?



Symm: fold sides over



Circular/Wrap: wrap around



pad/fill: add value, often 0

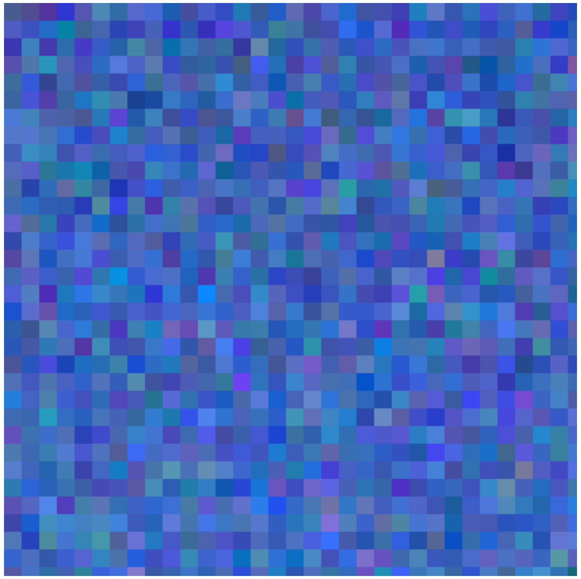
f/g Diagram Credit: D. Lowe

Edge Cases: Does It Matter?

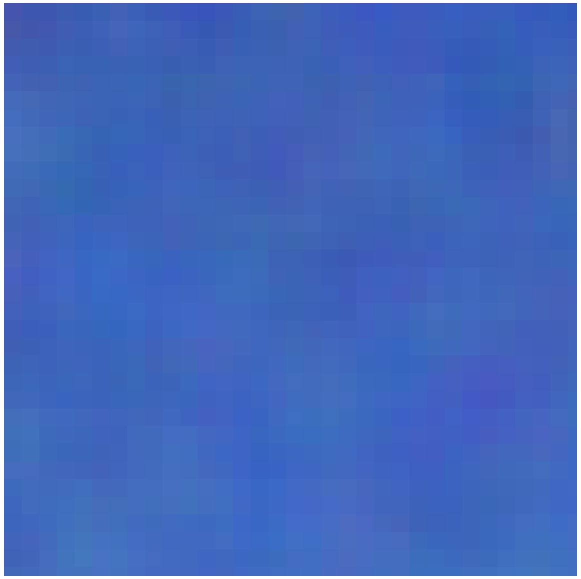
(I've applied the filter per-color channel)

Which padding did I use and *why*?

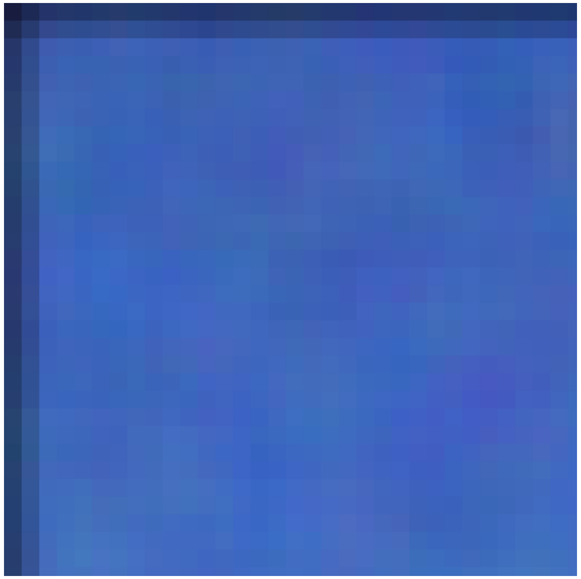
Input Image



Box Filtered ???



Box Filtered ???

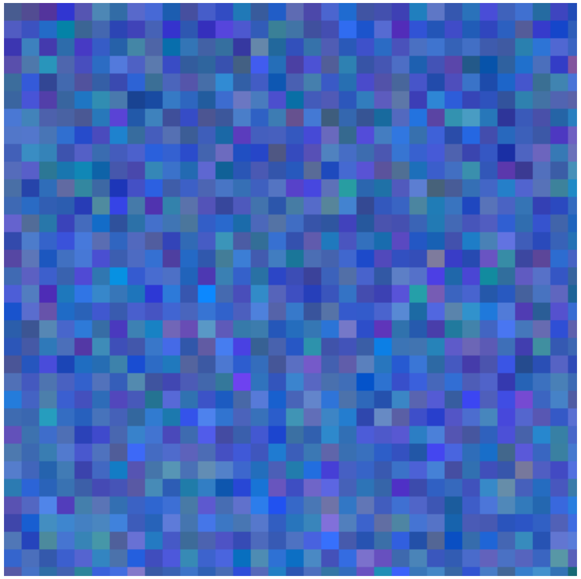


Note – this is a zoom of the filtered, not a filter of the zoomed

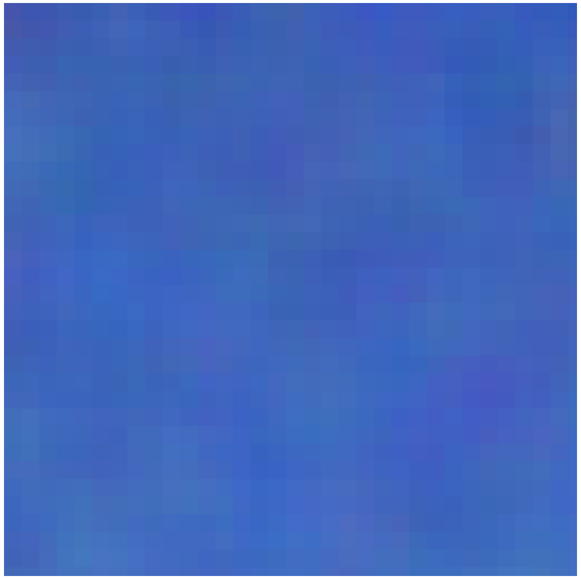
Edge Cases: Does It Matter?

(I've applied the filter per-color channel)

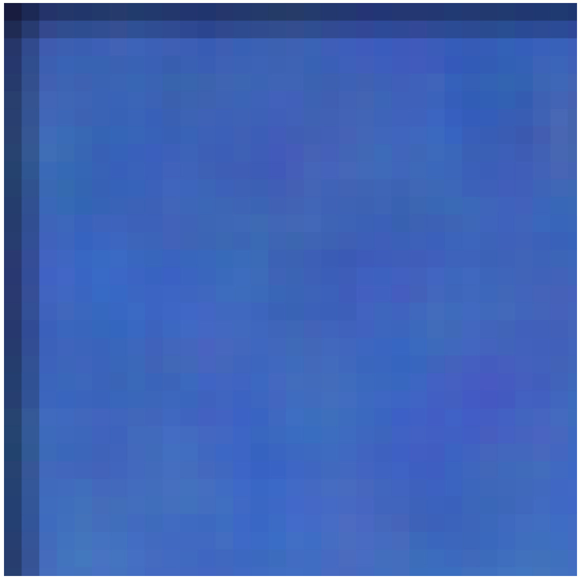
Input Image



Box Filtered Symm Pad

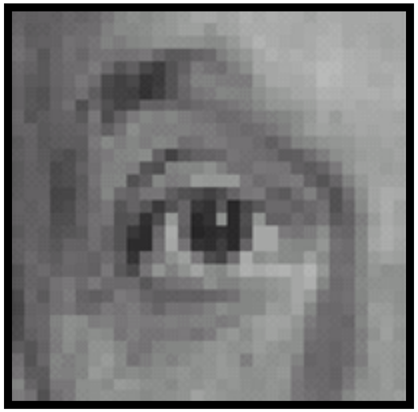


Box Filtered Zero Pad



Note – this is a zoom of the filtered, not a filter of the zoomed

Practice with Linear Filters



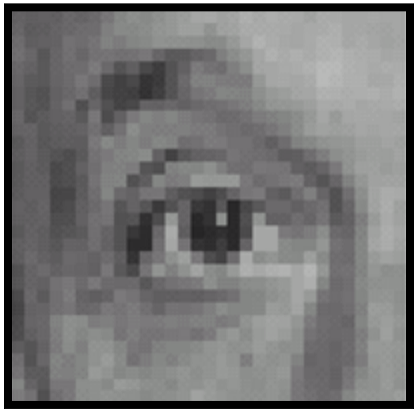
Original

0	0	0
0	1	0
0	0	0

?

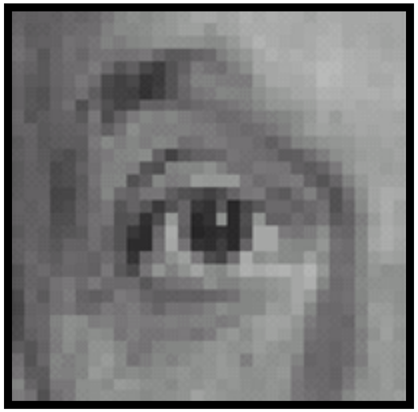
Slide Credit: D. Lowe

Practice with Linear Filters



Original

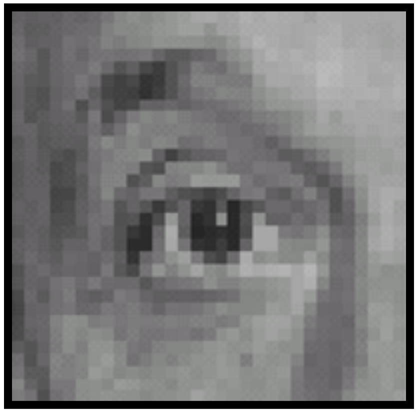
0	0	0
0	1	0
0	0	0



The Same!

Slide Credit: D. Lowe

Practice with Linear Filters



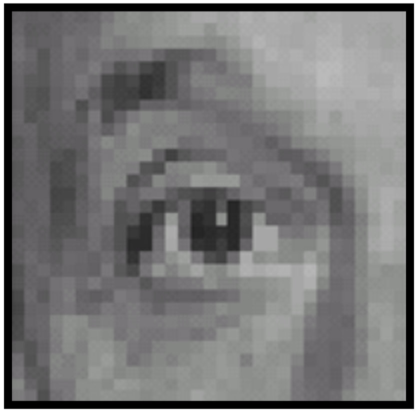
Original

0	0	0
0	0	1
0	0	0

?

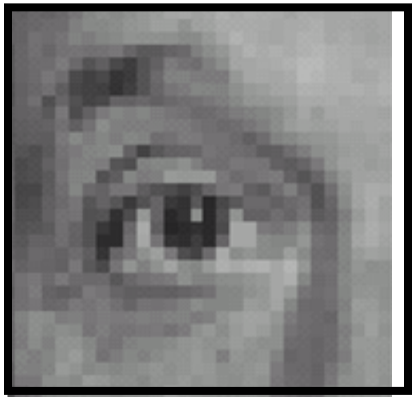
Slide Credit: D. Lowe

Practice with Linear Filters



Original

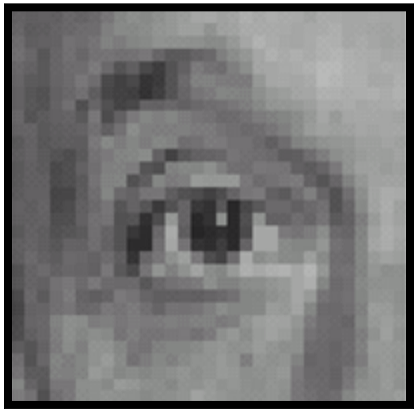
0	0	0
0	0	1
0	0	0



Shifted
<u>LEFT</u>
1 pixel

Slide Credit: D. Lowe

Practice with Linear Filters



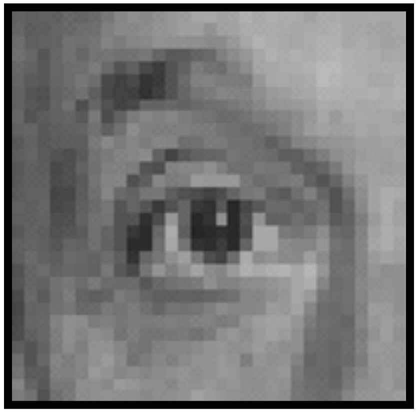
Original

0	1	0
0	0	0
0	0	0

?

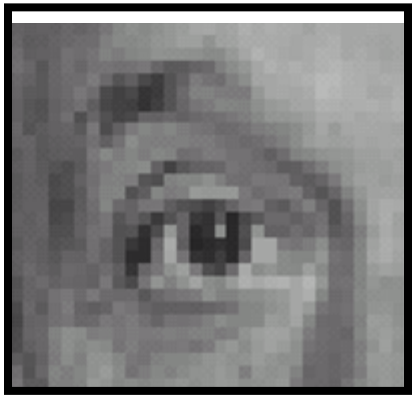
Slide Credit: D. Lowe

Practice with Linear Filters



Original

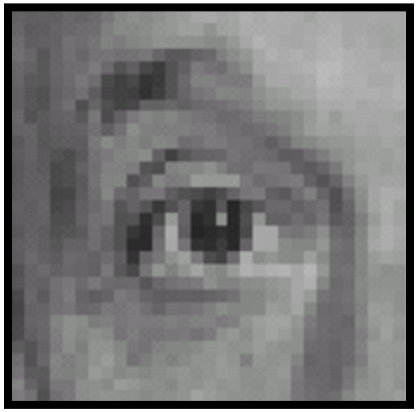
0	1	0
0	0	0
0	0	0



Shifted
DOWN
1 pixel

Slide Credit: D. Lowe

Practice with Linear Filters



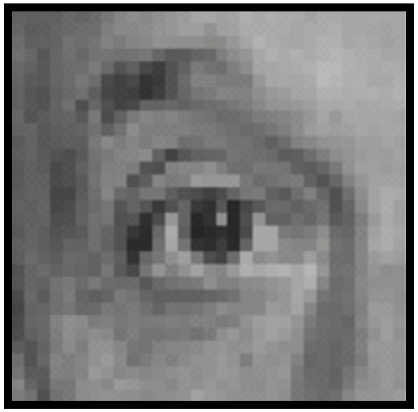
Original

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

?

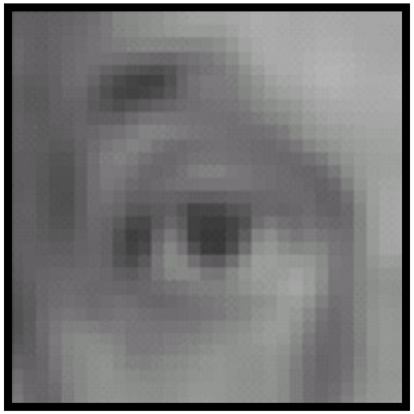
Slide Credit: D. Lowe

Practice with Linear Filters



Original

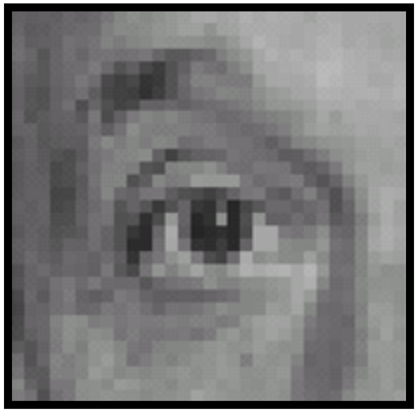
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9



Blur
(Box Filter)

Slide Credit: D. Lowe

Practice with Linear Filters



Original

0	0	0
0	2	0
0	0	0

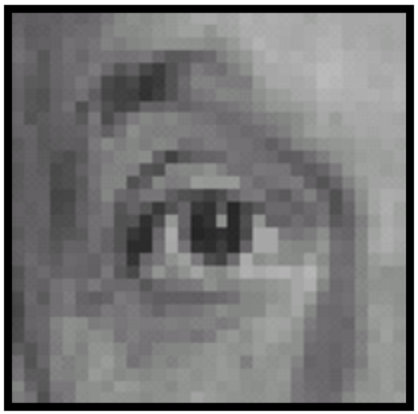
-

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

?

Slide Credit: D. Lowe

Practice with Linear Filters



Original

0	0	0
0	2	0
0	0	0

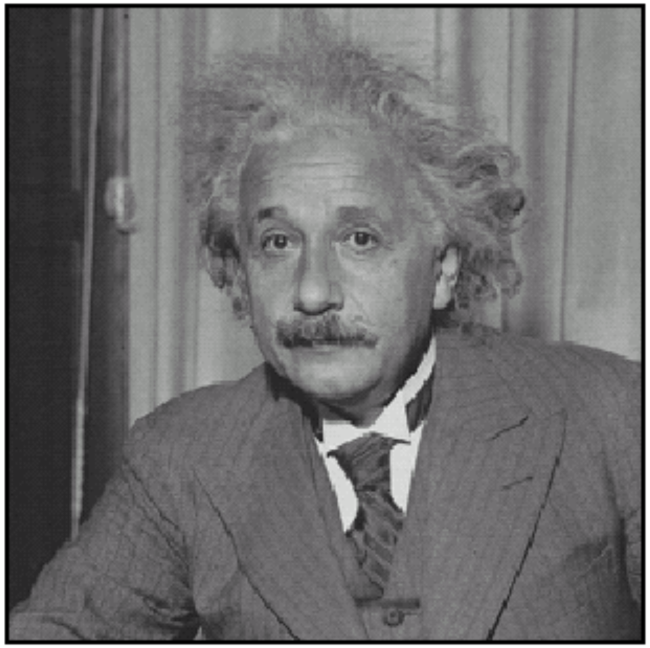
\-

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

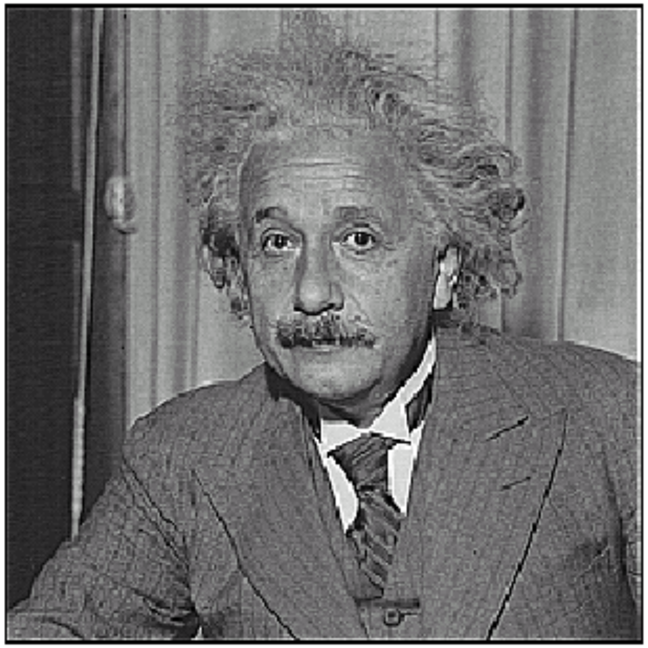
Sharpened
(Acccentuates difference from local average)

Slide Credit: D. Lowe

Sharpening



before



after

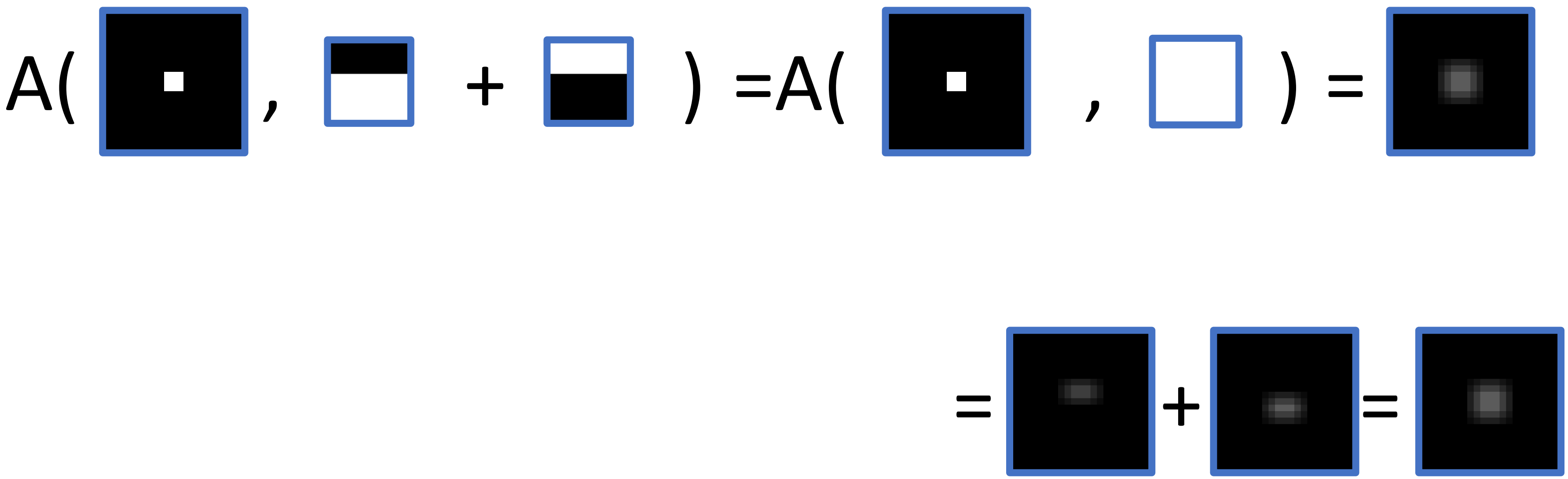
Slide Credit: D. Lowe

Properties: Linear

Assume: I image f1, f2 filters

Linear: apply(I,f1+f2) = apply(I,f1) + apply(I,f2)

I is a white box on black, and f1, f2 are rectangles



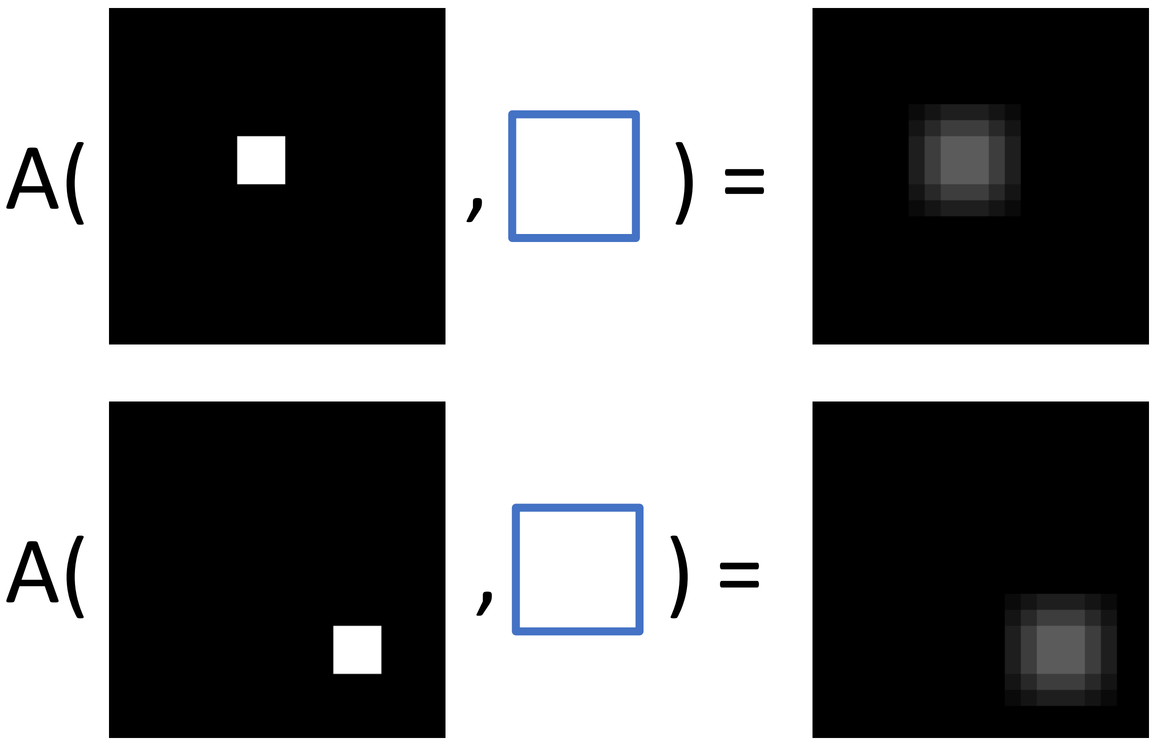
Note: I am showing filters un-normalized and blown up. They're a smaller box filter (i.e., each entry is 1/(size^2))

Properties: Shift-Invariant

Assume: I image, f filter

Shift-invariant: shift(apply(I,f)) = apply(shift(I,f))

Intuitively: only depends on filter neighborhood



Note: "Shift-Invariant" is standard terminology, but I think "Shift-Equivariant" is more correct

Annoying Terminology

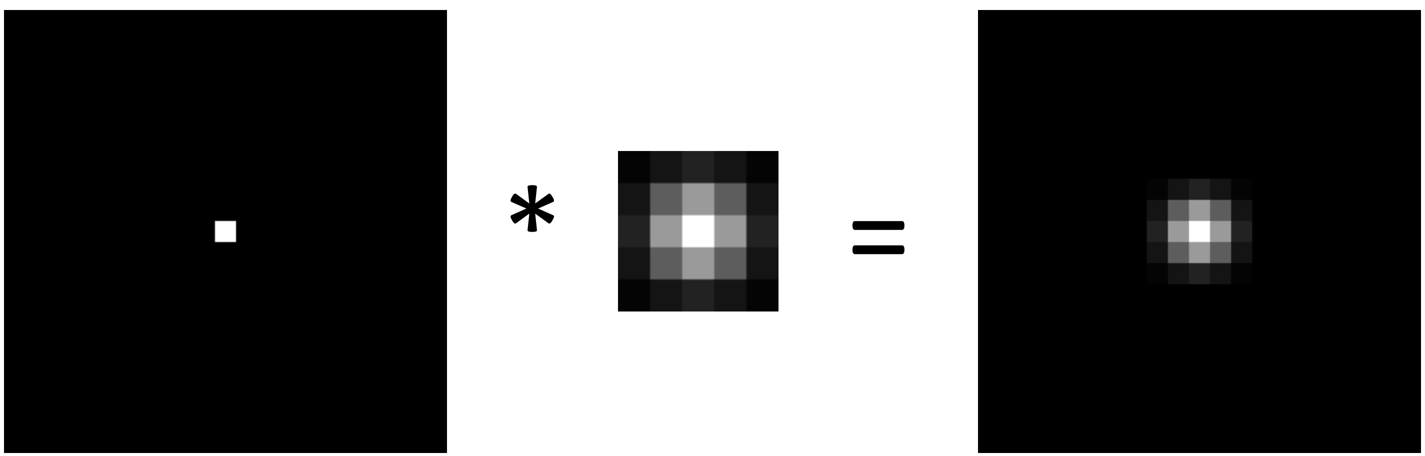
Often called “convolution”. *Actually* cross-correlation.

Cross-Correlation
(Slide filter over image)

Convolution
(Flip filter, then slide it)

Properties of Convolution

- Any shift-invariant, linear operation is a convolution ($\star$)
- Commutative: $f \star g = g \star f$
- Associative: $(f \star g) \star h = f \star (g \star h)$
- Distributes over +: $f \star (g + h) = f \star g + f \star h$
- Scalars factor out: $kf \star g = f \star kg = k\,(f \star g)$
- Identity (a single one with all zeros):



Property List: K. Grauman

Next Time:
More Image Filtering, Edge Detection