Chapter 2

Application overview (2)

Contents (class version)

2.0 Introduction

This chapter continues the previous chapter in giving an overview of many of the SIPML problems that require iterative algorithms for optimization and that will be addressed later in the course. Again, this chapter focuses on the cost functions, not the algorithms. This chapter has more applications involving **nonconvex** optimization problems.

2.1 Signal processing applications

Patch-based regularization methods

The sparsity-based regularizers in Ch. 1 used dictionaries or sparsifying transforms for the entire image. Especially for data-driven models, often it is more appropriate to synthesize or analyze each **patch** of an image rather than trying to model the entire image.

To write patch-based regularizers mathematically we need a matrix representation of the process of extracting a patch from an image.

Example. Patch extraction is easiest to illustrate in 1D. For a 1D signal of length $N = 8$, a few of the 3×8 matrices for extracting patches from the image with stride=1 are:

$$\boldsymbol{R}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \boldsymbol{R}_2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}, \ldots, \quad \boldsymbol{R}_6 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

If we choose to use **periodic boundary conditions** then we will also use these two patch extraction matrices in this example:

$$\boldsymbol{R}_7 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \boldsymbol{R}_8 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

In math, $\boldsymbol{R}_2 \boldsymbol{x} = \begin{bmatrix} x_2 & x_3 & x_4 \end{bmatrix}^T$, whereas in JULIA code it is simply `x[2:4]`

The generalization to 2D is similar conceptually but more cumbersome to write explicitly and one must think about **lexicographical order**.

To extract a 2D patch of size 2×3, from a $N_1 \times N_2$ image `x` use a JULIA command like:
`x[4:5,6:8]`

If `x` represents a 2D image that is shaped as a 1D array of length $N_1 N_2$, then use index chaining like this to extract a patch:
`reshape(x,N1,N2)[4:5,6:8][:]`

This selection process is a linear operation, so it is possible to write it as a $6 \times N_1 N_2$ matrix $\boldsymbol{R}$ that is all zeros except for one 1 in each row.

For $N_1 = 10$ and $N_2 = 7$, what is the column index for the 1 in the first row of the 6×70 matrix $\boldsymbol{R}$ corresponding to the previous JULIA statement?

A: 27 B: 36 C: 39 D: 54 E: None **??**

Patch-based synthesis regularizer

Let $\boldsymbol{R}_l \in \{0,1\}^{d \times N}$ denote the matrix for extracting the lth of L patches. Assume $\boldsymbol{R}_l \boldsymbol{x} \approx \boldsymbol{D} \boldsymbol{z}_l$ where $\boldsymbol{D}$ is $d \times K$ and $\boldsymbol{z}_l \in \mathbb{F}^K$ is **sparse**. Usually $d \le K$.

Define $\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{z}_1 & \dots & \boldsymbol{z}_L \end{bmatrix} \in \mathbb{F}^{K \times L}$. A typical **patch-based synthesis regularizer** is:

(2.1)

where typically $p = 0, 1$.

Patch-based analysis regularizer

A typical patch-based analysis (or sparsifying transform) regularizer is:

(2.2)

where $\boldsymbol{T}$ is a $K \times d$ sparsifying transform matrix for **vectorized** patches. Often $K \approx d$.

Example. If the patches are piece-wise constant, then a couple reasonable choices for $\boldsymbol{T}$ are

$$\boldsymbol{T}_1 = \begin{bmatrix} -1 & 1 & 0 & \dots & \\ 0 & -1 & 1 & \dots & \\ & & \ddots & \ddots & \\ 0 & \dots & 0 & -1 & 1 \end{bmatrix}, \quad \boldsymbol{T}_2 = \begin{bmatrix} -1 & -1 & 1 & 1 & 0 & \dots & \\ 0 & -1 & -1 & 1 & 1 & 0 & \dots \\ & & \ddots & \ddots & & & \\ 0 & \dots & 0 & -1 & -1 & 1 & 1 \end{bmatrix}, \quad \boldsymbol{T}_3 = \begin{bmatrix} \boldsymbol{T}_1 \\ \boldsymbol{T}_2 \end{bmatrix}.$$

Here $\boldsymbol{T}_1$ is $(d-1) \times d$ and $\boldsymbol{T}_2$ is $(d-2) \times d$ and $\boldsymbol{T}_3$ is $(2d-3) \times d$.

Example. In fact the most minimalist version would be $d = 2$ and $K = 1$ and $\boldsymbol{T} = \begin{bmatrix} -1 & 1 \end{bmatrix}$, which essentially ends up being very similar (but not identical) to a 1D TV regularizer, as discussed in Ch. 1.

Aggregate (global) sparsity versus patch-wise (local) sparsity

Because the historical significance of the **K-SVD** algorithm [1, 2], per-patch sparsity constraints have been popular. For image patches, regularizing based on aggregate sparsity seems more natural.

Here are dictionary versions of both regularizers for comparison:

(2.3)

(aggregate/global) (patch-wise/local)

There are interesting extensions of such patch-based regularizers, *e.g.*, integration with a **Gaussian mixture model** (**GMM**) for **image segmentation** [3].

Patch-based model learning

In earlier work, the dictionary $\boldsymbol{D}$ in (2.1) and the transform $\boldsymbol{T}$ in (2.2) were defined mathematically, *e.g.*, using the **discrete cosine transform** (**DCT**). More recent work focuses on adaptive methods where $\boldsymbol{D}$ or $\boldsymbol{T}$ are learned from **training data**.

Dictionary learning

Given a collection of L training signals $\boldsymbol{x}_1, \dots, \boldsymbol{x}_L \in \mathbb{F}^d$ (*e.g.*, patches taken from some population of images), the **dictionary learning** problem is to learn a matrix $\boldsymbol{D} = \begin{bmatrix} \boldsymbol{d}_1 & \dots & \boldsymbol{d}_K \end{bmatrix} \in \mathbb{F}^{d \times K}$ consisting of K **atoms** such that each $\boldsymbol{x}_l \approx \boldsymbol{D}\boldsymbol{z}_l$ where the $\boldsymbol{z}_l \in \mathbb{F}^K$ coefficient vectors are **sparse**, either individually or collectively when considering the coefficient matrix $\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{z}_1 & \dots & \boldsymbol{z}_L \end{bmatrix} \in \mathbb{F}^{K \times L}$. A typical dictionary learning optimization problem is:

$$\tag{2.4}$$

where the "arg min" and "min" above are deliberately different.

Typical choices for the sparsity regularizer ϕ are

- $\phi(\boldsymbol{z}) = \|\boldsymbol{z}\|_0$ (global sparsity)
- $\phi(\boldsymbol{z}) = \|\boldsymbol{z}\|_1$ (convex relaxation of $\|\cdot\|_0$)
- $\phi(\boldsymbol{z}) = \chi_{\{\|\boldsymbol{z}\|_0 \le K\}}$ (local sparsity).

If $\mathcal{D}$ is a convex set and we choose $\phi(\boldsymbol{z}) = \|\boldsymbol{z}\|_1$ then (2.4) is a **convex** optimization problem.

A: True B: False **??**

To avoid a scale ambiguity, typically

or

The first $\mathcal{D}$ is a **convex set**. (?)
A: True B: False **??**

If we use the latter choice and also choose the sparsity regularizer $\phi(\boldsymbol{z})$ to be a convex function, then (2.4) is a **biconvex** optimization problem because the cost function $\Psi(\boldsymbol{D}, \boldsymbol{Z})$ is convex in $\boldsymbol{D}$ with $\boldsymbol{Z}$ held fixed, and convex in $\boldsymbol{Z}$ with $\boldsymbol{D}$ held fixed.

To see why a function of the product of two arguments is nonconvex, consider $f(x, y) = xy$ and try
https://www.google.com/search?q=plot+x*y&oq=plot+x*y

Define. A function $f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_K) \mapsto \mathbb{R}$ where $\boldsymbol{x}_k \in \mathbb{F}^{N_k}$ is called **block multi-convex** iff f is convex with respect to each argument when all other arguments are held fixed. In other words, if we define $f_k : \mathbb{F}^{N_k} \mapsto \mathbb{R}$ by $f_k(\cdot) = f(\bar{\boldsymbol{x}}_1, \ldots, \bar{\boldsymbol{x}}_{k-1}, \cdot, \bar{\boldsymbol{x}}_{k+1}, \ldots, \bar{\boldsymbol{x}}_K)$ for any given values of $\bar{\boldsymbol{x}}_j$ for $j \neq k$, then f is **block multi-convex** iff each f_k is **convex** for $k = 1, \ldots, K$.

Sparsifying transform learning

Instead of learning a dictionary $\boldsymbol{D}$ such that $\boldsymbol{x}_l \approx \boldsymbol{D}\boldsymbol{z}_l$, which is a synthesis model, and alternative is to learn a transform $\boldsymbol{T} \in \mathbb{F}^{K \times d}$ such that $\{\boldsymbol{T}\boldsymbol{x}_l\}$ is sparse (either individually or collectively). This is called **sparsifying transform learning**. Often $K = d$ but we also consider other cases here.

Again let $\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{z}_1 & \dots & \boldsymbol{z}_L \end{bmatrix} \in \mathbb{F}^{K \times L}$ denote the transform coefficient matrix. A typical transform learning optimization problem is:

(2.5)

where the "arg min" and "min" above are deliberately different and ϕ is some sparsity regularizer.

An alternative formulation that is easier to write but perhaps harder to optimize is

$$\hat{\boldsymbol{T}} = \arg\min_{\boldsymbol{T} \in \mathcal{T}} \Psi(\boldsymbol{T}), \qquad \Psi(\boldsymbol{T}) \triangleq \sum_{l=1}^{L} \phi(\boldsymbol{T}\boldsymbol{x}_l).$$

If $\mathcal{T}$ is a convex set and we choose $\phi(\boldsymbol{z}) = \|\boldsymbol{z}\|_1$ then (2.5) is a **convex** problem in $(\boldsymbol{T}, \boldsymbol{Z})$.
A: True B: False **??**

A HW problem examines the (joint) convexity of the term involving both $\boldsymbol{T}$ and $\boldsymbol{Z}$.

For transform learning, we need to avoid a scale ambiguity *and* make sure the rows of $\boldsymbol{T}$ are not redundant. One approach is to use the following "hard" orthonormality constraint:

With this choice of $\mathcal{T}$ the problem (2.5) is always nonconvex.

Example. Consider the case $K = 1$ and $d = 2$ so $\boldsymbol{T} = \begin{bmatrix} t_1 & t_2 \end{bmatrix}$. Then

$$\mathcal{T} = \left\{ \boldsymbol{T} \in \mathbb{R}^{1 \times 2} \ : \ \boldsymbol{T}\boldsymbol{T}' = \boldsymbol{I}_1 \right\} = \left\{ \begin{bmatrix} t_1 & t_2 \end{bmatrix} \ : \ t_1^1 + t_2^2 = 1 \right\}.$$

Regularized transform learning

An alternative transform learning formulation using a regularizer rather than an orthonormality constraint is:

$$\hat{\boldsymbol{T}} = \underset{\boldsymbol{T} \in \mathbb{F}^{K \times d}}{\arg\min} \min_{\boldsymbol{Z} \in \mathbb{F}^{K \times L}} \Psi(\boldsymbol{T}, \boldsymbol{Z}), \qquad \Psi(\boldsymbol{T}, \boldsymbol{Z}) \triangleq \sum_{l=1}^{L} \frac{1}{2} \|\boldsymbol{T}\boldsymbol{x}_l - \boldsymbol{z}_l\|_2^2 + \alpha \phi(\boldsymbol{z}_l) + \beta R(\boldsymbol{T}), \tag{2.6}$$

where a typical regularizer for $\boldsymbol{T}$ when $K \leq d$ is [4]:

$$R(\boldsymbol{T}; \mu) = \mu \, \mathsf{trace}\{\boldsymbol{T}\boldsymbol{T}'\} - \log \mathsf{det}\{\boldsymbol{T}\boldsymbol{T}'\} \, .$$

This approach is nonconvex in $\boldsymbol{T}$ but has convenient alternating minimization algorithms [5]. Its drawback is that now one much choose additional regularization parameters β and μ, although $\mu = 1$ is a natural choice.

To help understand this $R(\boldsymbol{T}, 1)$, suppose $\boldsymbol{T}$ has **economy SVD** $\boldsymbol{T} = \boldsymbol{U}\boldsymbol{\Sigma}_K\boldsymbol{V}_K'$, so $\boldsymbol{T}\boldsymbol{T}' = \boldsymbol{U}\boldsymbol{\Sigma}_K^2\boldsymbol{U}'$.
Then when $\mu = 1$:

$$R(\boldsymbol{T}, 1) = \mathsf{trace}\{\boldsymbol{\Sigma}_K^2\} - \log \mathsf{det}\{\boldsymbol{\Sigma}_K^2\} = \sum_{k=1}^{K} \left(\sigma_k^2 - 2\log \sigma_k\right).$$

This function is convex in each σ_k. (?)
A: True B: False **??**

Its minimizer is $\sigma_k = 1$. (?)
A: True B: False **??**

So large values of β encourage $\boldsymbol{T}$ to have singular values near which is a kind of "soft" constraint.

Convolutional filter learning

Instead of extracting image patches and learning a *transform* that can sparsify those patches, a related alternative is to learn *filters* that produce sparse outputs when applied to the entire image. Let $\boldsymbol{x}_1, \dots, \boldsymbol{x}_L \in \mathbb{F}^N$ denote training images. We want to learn filters $\boldsymbol{h}_1, \dots, \boldsymbol{h}_K \in \mathbb{F}^R$ such that $\boldsymbol{z}_{l,k} = \boldsymbol{h}_k * \boldsymbol{x}_l$ is **sparse** for all (or most) k and l.

Collect the filters into a matrix $\boldsymbol{H} \triangleq \begin{bmatrix} \boldsymbol{h}_1 & \dots & \boldsymbol{h}_K \end{bmatrix} \in \mathbb{F}^{R \times K}$ and let $\boldsymbol{Z}$ denote the collection of the $\{\boldsymbol{z}_{l,k}\}$ outputs. Then convolutional filter learning is typically posed as an optimization problem like [6, 7]:

where $\phi(\cdot)$ is some sparsity regularizer such as the 1-norm.

If ϕ is convex, then $\Psi(\boldsymbol{H}, \boldsymbol{Z})$ is **jointly convex** in $\boldsymbol{H}$ and $\boldsymbol{Z}$ (?)
Hint. Letting $\boldsymbol{X}_l$ denote the $N \times R$ or $(N + R - 1)$ matrix (depending on boundary conditions) consisting of patches from $\boldsymbol{x}_l$:

$$\|\boldsymbol{h}_k * \boldsymbol{x}_l - \boldsymbol{z}_{l,k}\| = \|\boldsymbol{X}_l \boldsymbol{h}_k - \boldsymbol{z}_{l,k}\| .$$

A: True B: False **??**

If one used $\mathcal{H} = \mathbb{F}^{R \times K}$ then the problem would have the trivial solution $\hat{\boldsymbol{H}} = \boldsymbol{0}$.

To ensure diversity, one can use the following (scaled) orthogonality constraint

This constraint is related to a **tight frame** condition [6, 7].

There are also **synthesis** versions of convolutional model learning [8].

An early approach used a **Markov random field** perspective and constrained the filters to have mean zero [9, 10], but no other diversity constraint.

Blind deconvolution

Another signal processing optimization problem that is often **biconvex** is **blind deconvolution** [11–14].

In 1D, the typical model here os

$$\boldsymbol{y} = \boldsymbol{h} * \boldsymbol{x} + \boldsymbol{\varepsilon},$$

where $\boldsymbol{h} \in \mathbb{F}^K$ is the **point-spread function** (**PSF**) of the unknown blur, $\boldsymbol{x} \in \mathbb{F}^N$ is the unknown image, and $\boldsymbol{y} \in \mathbb{F}^{N+K-1}$ is the observed blurry image. This is an under-determined problem with $N + K$ unknowns.

A typical regularized optimization formulation is

$$(\hat{\boldsymbol{x}}, \hat{\boldsymbol{h}}) = \arg\min_{\boldsymbol{x},\boldsymbol{h}} \Psi(\boldsymbol{x}, \boldsymbol{h}), \quad \Psi(\boldsymbol{x}, \boldsymbol{h}) \triangleq \frac{1}{2} \|\boldsymbol{y} - \boldsymbol{h} * \boldsymbol{x}\|_2^2 + R_1(\boldsymbol{x}) + R_2(\boldsymbol{h}),$$

where the image regularizer $R_1(\boldsymbol{x})$ might be **TV** is $\boldsymbol{x}$ is piece-wise constant, and the blur regularizer $R_2(\boldsymbol{h})$ might be $\|\boldsymbol{h}\|_1$ if the blur PSF is sparse, *e.g.*, for camera shake.

Out-of-focus blur is another common PSF model.

The cost function above is **biconvex** because of the $\boldsymbol{h} * \boldsymbol{x}$ term.

Phase retrieval (Read)

An classic signal processing problem is that of **phase retrieval** [15–18], where the measurement model is:

$$y_i = \left|[\boldsymbol{A}\boldsymbol{x}]_i\right|^2 + \varepsilon_i.$$

Historically the $M \times N$ matrix $\boldsymbol{A}$ corresponded to a Fourier transform operation, but there have been many generalizations.

The nonlinearity due to the absolute value greatly complicates estimation, and generally necessitates iterative algorithms, even for small problem sizes.

There has been much recent work due to **convex** formulations [19] and sparsity models [20–22] and more [23].

For Gaussian noise, a typical nonconvex optimization formulation is [24]:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \sum_i \frac{1}{2} \left| y_i - \left|[\boldsymbol{A}\boldsymbol{x}]_i\right|^2 \right|^2 + \beta R(\boldsymbol{x}),$$

but the literature has many variations. This cost function is quite nonconvex due to the absolute value.

2.2 Machine learning applications

Low-rank approximation (review) (Read)

If $\boldsymbol{Y} \in \mathbb{F}^{M \times N}$ is a given data matrix with rank r having **SVD** $\boldsymbol{Y} = \boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{V}'$ and $\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{\varepsilon}$ where we believe $\mathsf{rank}(\boldsymbol{X}) \le K$ and $\boldsymbol{\varepsilon}$ denotes a $M \times N$ noise matrix, then EECS 551 discussed many methods for **low-rank approximation** that balance between data-fit and model complexity, all having the general form:

$$\hat{\boldsymbol{X}} = \underset{\boldsymbol{X} \in \mathbb{F}^{M \times N}}{\arg\min} \tag{2.7}$$

for some **unitarily invariant** matrix norm $\|\cdot\|$, such as the **Frobenius norm** $\|\cdot\|_{\mathrm{F}}$, and for some unitarily invariant **regularizer** $R(\boldsymbol{X})$, where $R : \mathbb{F}^{M \times N} \mapsto \mathbb{R}$, such as $R(\boldsymbol{X}) = \chi_{\{\mathsf{rank}(\boldsymbol{X}) \le K\}}$ or $R(\boldsymbol{X}) = \mathsf{rank}(\boldsymbol{X})$ or $R(\boldsymbol{X}) = \|\boldsymbol{X}\|_*$. Any minimizer has the form

$$\hat{\boldsymbol{X}} = \sum_{k=1}^{r} \hat{w}_k \, \boldsymbol{u}_k \boldsymbol{v}_k' = \boldsymbol{U}_r \hat{\boldsymbol{\Sigma}}_r \boldsymbol{V}_r' \text{ where } \hat{\boldsymbol{\Sigma}}_r = \mathsf{diag}\{\hat{w}_k\}, \quad \hat{w}_k = h_k(\sigma_1, \dots, \sigma_r; \beta), \quad \boldsymbol{Y} = \boldsymbol{U}_r \boldsymbol{\Sigma}_r \boldsymbol{V}_r', \tag{2.8}$$

for some **shrinkage** or **thresholding** function $h_k(\cdot; \beta)$ that depends on the data-fit norm and the regularizer. The matrix product $\boldsymbol{U}_r \hat{\boldsymbol{\Sigma}}_r \boldsymbol{V}_r'$ forms the **compact SVD** of $\hat{\boldsymbol{X}}$ where $\boldsymbol{U}_r$ is $M \times r$ and $\boldsymbol{\Sigma}_r$ is $r \times r$ and $\boldsymbol{V}_r$ is $N \times r$. Typically only $K \ll r$ elements of $\hat{\boldsymbol{\Sigma}}_r$ are nonzero so one can also write $\hat{\boldsymbol{X}}$ as a product of $N \times K$, $K \times K$ and $K \times N$ matrices: $\hat{\boldsymbol{X}} = \boldsymbol{U}_K \hat{\boldsymbol{\Sigma}}_K \boldsymbol{V}_K'$.

Unfortunately, this elegant approach does not scale to problems where both M and N are large, even if the rank K is very small, because of the **SVD** operation.

Matrix factorization approach

To overcome this limitation of SVD-based approaches, one can take a **matrix factorization** approach by choosing a desired (maximum) rank K and expressing $\hat{\boldsymbol{X}}$ directly as

$$\hat{\boldsymbol{X}} =$$

where now $\boldsymbol{U}$ and $\boldsymbol{V}$ should be simply interpreted as latent factors, not as matrices with singular vectors. With this formulation, a typical optimization problem for finding $\boldsymbol{U}$ and $\boldsymbol{V}$, and hence $\hat{\boldsymbol{X}}$, looks like

$$\begin{aligned} \hat{\boldsymbol{X}} &= \\ (\hat{\boldsymbol{U}}, \hat{\boldsymbol{V}}) &= \\ \Psi(\boldsymbol{U}, \boldsymbol{V}) &\triangleq \end{aligned} \tag{2.9}$$

where one must select appropriate regularizers for $\boldsymbol{U}$ and $\boldsymbol{V}$.

If R_1 and R_2 are each convex cost functions, then the cost function is:
A: biconvex in $(\boldsymbol{U}, \boldsymbol{V})$ B: jointly convex in $(\boldsymbol{U}, \boldsymbol{V})$ C: none of these **??**

We will discuss regularizers and **alternating minimization** algorithms for Ψ that involve no SVD operations and scale to large problems.

We do not need a regularizer like $\|\boldsymbol{X}\|_* = \|\boldsymbol{U}\boldsymbol{V}\|_*$ here because that **nuclear norm** regularizer is a surrogate for the rank constraint $\chi_{\{\mathsf{rank}(\boldsymbol{X}) \le K\}}$ and $\mathsf{rank}(\boldsymbol{U}\boldsymbol{V}) \le K$ by construction!

Example. Denote the set of $M \times K$ matrices with nonnegative elements by

$$\mathcal{N}_M = \left\{ \boldsymbol{U} \in \mathbb{R}^{M \times K} \ : \ u_{mk} \ge 0, \ m = 1, \dots, M, \ k = 1, \dots, K \right\}.$$

Then **nonnegative matrix factorization**, (**NMF**) or (**NNMF**), is the matrix factorization problem in (2.9) above where $R_1(\boldsymbol{U}) = \chi_{\mathcal{N}_M}(\boldsymbol{U})$ and R_2 is defined similarly [25–27].

The $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{V}$ approach is sometimes called the **Burer-Monteiro** style parameterization after [28].

Low-rank matrix completion (Read)

As discussed in EECS 551, **matrix completion** problem involves noisy missing data and a sampling mask:

$$Y_{ij} = \begin{cases} X_{ij} + \varepsilon_{ij}, & (i,j) \in \Omega \\ 0, & \text{otherwise}, \end{cases} \qquad M_{ij} \triangleq \begin{cases} 1, & (i,j) \in \Omega \\ 0, & \text{otherwise}. \end{cases} \tag{2.10}$$

We want to find $\hat{\boldsymbol{X}}$ where $\boldsymbol{M} \odot \hat{\boldsymbol{X}} \approx \boldsymbol{M} \odot \boldsymbol{Y}$ and also $\hat{\boldsymbol{X}}$ is low-rank. A typical **low-rank matrix completion** (**LRMC**) formulation is

$$\hat{\boldsymbol{X}} = \underset{\boldsymbol{X} \in M \times N}{\arg\min} \frac{1}{2} \| \boldsymbol{M} \odot (\boldsymbol{X} - \boldsymbol{Y}) \|_{\mathrm{F}}^2 + \beta R(\boldsymbol{X}), \tag{2.11}$$

where R is a rank constraint or rank penalty or nuclear norm penalty, etc. Other than in trivial cases such as when $\boldsymbol{Y}$ is zero, this optimization problem *always* requires iterative algorithms to solve.

EECS 551 described **majorize-minimize** (**MM**) algorithms for this family of optimization problems. These MM algorithms have an inner step of the form:

$$\boldsymbol{X}_{k+1} = \underset{\boldsymbol{X} \in \mathbb{F}^{M \times N}}{\arg\min} \frac{1}{2} \left\| \boldsymbol{X} - \bar{\boldsymbol{X}}_k \right\|^2 + \beta R(\boldsymbol{X}), \quad \bar{\boldsymbol{X}}_k = \bar{\boldsymbol{M}} \odot \boldsymbol{X} + \boldsymbol{M} \odot \boldsymbol{Y}.$$

Unfortunately, this inner form is a **low-rank matrix approximation** problem and the standard solutions use **SVD** operations. Likewise, **ADMM** algorithms for this problem also use an inner SVD step.

This class will discuss **matrix factorization** approaches that scale to problems where both M and N are large, as long as the rank K is small enough.

Matrix sensing / matrix recovery

The measurement model (2.10) involves **samples** of the elements of the matrix $\boldsymbol{X}$, with additive noise.

A more general measurement model that arises in a variety of applications (including dynamic MRI image reconstruction) involves linear combinations of the elements of $\boldsymbol{X} \in \mathbb{F}^{M \times N}$ [29, 30]. Suppose we have a vector $\boldsymbol{y} \in \mathbb{F}^L$ of measurements, each of which is a linear combination of the elements of $\boldsymbol{X}$, plus noise. The concise way to write this model is

where $\boldsymbol{\varepsilon} \in \mathbb{F}^L$ is a noise vector and where $\mathcal{A} : \mathbb{F}^{M \times N} \mapsto \mathbb{F}^L$ is a **linear map** or **linear operator** that maps a $M \times N$ matrix into a vector in $\mathbb{F}^L$.
The goal in such **matrix sensing** or **matrix recovery** problems is to recover $\boldsymbol{X}$ from $\boldsymbol{y}$.

The **operator** $\mathcal{A}$ is *not* a matrix, but it is linear. To write this model in matrix form we use

The size of the matrix $\boldsymbol{A}$ here is:
A: $M \times N$ B: $(LM) \times N$ C: $(LN) \times M$ D: $L \times (MN)$ E: none of these **??**

Example.

$$\mathcal{A}(\boldsymbol{X}) = \begin{bmatrix} \mathbf{1}_M' \boldsymbol{X} \mathbf{1}_N \\ 7X_{1,1} - X_{3,1} \end{bmatrix} \implies \boldsymbol{A} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 7 & 0 & -1 & 0 & 0 & 0 \end{bmatrix}.$$

The **matrix completion** problem is a special case of the **matrix sensing** problem where roughly

$$\mathcal{A}(\boldsymbol{X}) = \mathsf{vec}(\boldsymbol{M} \odot \boldsymbol{X}) \,.$$

Actually that expression is not quite accurate because in matrix sensing L is the number of observed measurements. In matrix completion that number is

$$L = \sum_i \sum_j M_{ij} = \mathbf{1}_M' \boldsymbol{M} \mathbf{1}_N .$$

It is not easy to write $\mathcal{A}$ in terms of $\boldsymbol{M}$ in "math," but it is easy to write the operation in JULIA:

```
X[M]
```

where `M` is a `Bool` array.

Because matrix completion is a special case of matrix sensing, we can also write its measurement model as

$$\boldsymbol{y} = \boldsymbol{A}\, \mathsf{vec}(\boldsymbol{X}) + \boldsymbol{\varepsilon} .$$

In this form, each row of $\boldsymbol{A}$ has:
A: exactly one 1 and the rest zeros B: at most one 1 and the rest zeros C: none of these **??**

For problems of moderate size, a typical matrix sensing optimization formulation is

or equivalently

Typical solutions involve assuming that $\boldsymbol{X}$ is low-rank, and there are **MM** and **ADMM** iterative algorithms (and more) that involve inner steps that are **low-rank matrix approximation** problems that involve **SVD** operations.

We will discuss **matrix factorization** approaches that scale to large problems.

Group activity: write down what you think the matrix factorization approaches look like.

2.3 Summary

This chapter and the preceding chapter describe several SIPML applications with optimization formulations that do not have closed-form solutions and thus require iterative algorithms to solve.

There are other problems of interest that we may discuss if time permits

- manifold problems like subspace learning
- machine learning like probabilistic PCA
- CNN training
- learned regularizers like BCD-net, momentum net

All of these problems, like most in this chapter, are nonconvex, and there is growing progress on guarantees for certain nonconvex optimization problems; see:
https://m.huffpost.com/us/entry/9563882/amp

The next chapter begins to discuss iterative optimization methods. We will start with the simpler cost functions (quadratic), then move to convex and smooth problems, then convex and non-smooth problems, and then eventually to non-convex problems.

We will not discuss **direct search** methods such as the **Nelder-Mead simplex algorithm** [31, 32] because such methods do not scale well to large-scale problems.

Bibliography

[1] 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 (cit. on p. 2.7).

[2] R. Rubinstein, M. Zibulevsky, and M. Elad. *Efficient implementation of the K-SVD algorithm using batch orthogonal matching pursuit*. Tech. Report 40, CS Department, Technion. 2008 (cit. on p. 2.7).

[3] J. Caballero, W. Bai, A. N. Price, D. Rueckert, and J. V. Hajnal. "Application-driven MRI: joint reconstruction and segmentation from undersampled MRI data". In: *Medical Image Computing and Computer-Assisted Intervention*. 2014, 106–13 (cit. on p. 2.7).

[4] S. Ravishankar and Y. Bresler. "Learning sparsifying transforms". In: *IEEE Trans. Sig. Proc.* 61.5 (Mar. 2013), 1072–86 (cit. on p. 2.12).

[5] S. Ravishankar and Y. Bresler. "l_0 sparsifying transform learning with efficient optimal updates and convergence guarantees". In: *IEEE Trans. Sig. Proc.* 63.9 (May 2015), 2389–404 (cit. on p. 2.12).

[6] I. Y. Chun and J. A. Fessler. *Convolutional analysis operator learning: acceleration and convergence*. 2018 (cit. on pp. 2.14, 2.15).

[7] I. Y. Chun and J. A. Fessler. "Convolutional analysis operator learning: acceleration and convergence". In: *IEEE Trans. Im. Proc.* 29.1 (Jan. 2020), 2108–22 (cit. on pp. 2.14, 2.15).

[8] I. Y. Chun and J. A. Fessler. "Convolutional dictionary learning: acceleration and convergence". In: *IEEE Trans. Im. Proc.* 27.4 (Apr. 2018), 1697–712 (cit. on p. 2.15).

[9] 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 (cit. on p. 2.15).

[10] S. Roth and M. J. Black. "Fields of experts". In: *Intl. J. Comp. Vision* 82.2 (Jan. 2009), 205–29 (cit. on p. 2.15).

[11] D. Kundur and D. Hatzinakos. "Blind image deconvolution". In: *IEEE Sig. Proc. Mag.* 13.3 (May 1996), 43–64 (cit. on p. 2.16).

[12] D. Kundur and D. Hatzinakos. "Blind image deconvolution revisited". In: *IEEE Sig. Proc. Mag.* 13.6 (Nov. 1996), 61–63 (cit. on p. 2.16).

[13] G. Harikumar and Y. Bresler. "Perfect blind restoration of images blurred by multiple filters: theory and efficient algorithms". In: *IEEE Trans. Im. Proc.* 8.2 (Feb. 1999), 202–19 (cit. on p. 2.16).

[14] Y. Li, K. Lee, and Y. Bresler. "Identifiability and stability in blind deconvolution under minimal assumptions". In: *IEEE Trans. Info. Theory* 63.7 (July 2017), 4619–33 (cit. on p. 2.16).

[15] R. W. Gerchberg and W. O. Saxton. “A practical algorithm for the determination of phase from image and diffraction plane pictures”. In: *Optik* 35.2 (Apr. 1972), 237–46 (cit. on p. 2.17).

[16] J. R. Fienup. “Reconstruction of an object from the modulus of its Fourier transform”. In: *Optics Letters* 3.1 (July 1978), 27–9 (cit. on p. 2.17).

[17] J. R. Fienup. “Phase retrieval algorithms: a comparison”. In: *Appl. Optics* 21.15 (Aug. 1982), 2758–69 (cit. on p. 2.17).

[18] J. R. Fienup. “Phase retrieval algorithms: a personal tour [Invited]”. In: *Appl. Optics* 52.1 (Jan. 2013), 45–56 (cit. on p. 2.17).

[19] E. J. Candes, T. Strohmer, and V. Voroninski. “PhaseLift: exact and stable signal recovery from magnitude measurements via convex programming”. In: *Comm. Pure Appl. Math.* 66.8 (Aug. 2013), 1241–74 (cit. on p. 2.17).

[20] Y. C. Eldar and S. Mendelson. “Phase retrieval: Stability and recovery guarantees”. In: *Applied and Computational Harmonic Analysis* 36.3 (May 2014), 473–94 (cit. on p. 2.17).

[21] Y. Shechtman, A. Beck, and Y. C. Eldar. “GESPAR: efficient phase retrieval of sparse signals”. In: *IEEE Trans. Sig. Proc.* 62.4 (Feb. 2014), 928–38 (cit. on p. 2.17).

[22] M. Iwen, A. Viswanathan, and Y. Wang. “Robust sparse phase retrieval made easy”. In: *Applied and Computational Harmonic Analysis* 42.1 (Jan. 2017), 135–42 (cit. on p. 2.17).

[23] M. N’Gom, M-B. Lien, N. M. Estakhri, T. B. Norris, E. Michielssen, and R. R. Nadakuditi. “Controlling light transmission through highly scattering media using semi-definite programming as a phase retrieval computation method”. In: *Nature* 7.1 (2017) (cit. on p. 2.17).

[24] D. S. Weller, A. Pnueli, G. Divon, O. Radzyner, Y. C. Eldar, and J. A. Fessler. “Undersampled phase retrieval with outliers”. In: *IEEE Trans. Computational Imaging* 1.4 (Dec. 2015), 247–58 (cit. on p. 2.17).

[25] D. D. Lee and H. S. Seung. “Learning the parts of objects by non-negative matrix factorization”. In: *Nature* 401.6755 (1999), 788–91 (cit. on p. 2.20).

[26] P. Fernsel and P. Maass. “A survey on surrogate approaches to non-negative matrix factorization”. In: *Viet. J. Math* (2018) (cit. on p. 2.20).

[27] X. Fu, K. Huang, N. D. Sidiropoulos, and W-K. Ma. “Nonnegative matrix factorization for signal and data analytics: identifiability, algorithms, and applications”. In: *IEEE Sig. Proc. Mag.* 36.2 (Mar. 2019), 59–80 (cit. on p. 2.20).

[28] S. Burer and R. D. C. Monteiro. “A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization”. In: *Mathematical Programming* 95.2 (Feb. 2003), 329–57 (cit. on p. 2.20).

[29] R. Ge, C. Jin, and Y. Zheng. “No spurious local minima in nonconvex low rank problems: A unified geometric analysis”. In: *Proc. Intl. Conf. Mach. Learn*. Vol. 70. 2017, 1233–42 (cit. on p. 2.22).

[30] R. Y. Zhang, Cedric Josz, S. Sojoudi, and J. Lavaei. "How much restricted isometry is needed in nonconvex matrix recovery?" In: *Neural Info. Proc. Sys.* 2018 (cit. on p. 2.22).

[31] J. A. Nelder and R. Mead. "A simplex method for function minimization". In: *Computer Journal* 7.4 (1965), 308–13 (cit. on p. 2.25).

[32] T. G. Kolda, R. M. Lewis, and V. Torczon. "Optimization by direct search: new perspectives on some classical and modern methods". In: *SIAM Review* 45.3 (2003), 385–482 (cit. on p. 2.25).