Chapter 6

Alternating minimization

Contents (class version)

6.0 Introduction

Although the proximal methods in the previous chapter are quite flexible and useful, there are many cost functions of interest that are not smooth and also not "prox friendly." So we need additional tools.

Furthermore, all of the methods discussed so far update all elements of the optimization variable simultaneously. For many optimization problems it can be easier to update just some of the variables at a time.

This chapter develops **alternating minimization** algorithms that are suitable for such problems.

These methods have a long history in both optimization and signal/image processing. In image processing, an early approach was **iterated conditional modes** (**ICM**) [1].

These **coordinate descent** methods can be very useful, but also can have limitations for certain nonsmooth cost functions.

This chapter illustrates the methods by focusing on SIMPL applications, drawing from Ch. 1 and Ch. 2.

6.1 Signal processing applications

We begin with signal and image processing applications.

Compressed sensing using synthesis sparsity models

Consider a linear measurement model assuming **synthesis** sparsity using a (usually wide) dictionary $\boldsymbol{D}$:

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\varepsilon}, \qquad \boldsymbol{x} \approx \boldsymbol{D}\boldsymbol{z}, \qquad \boldsymbol{z} \text{ is sparse.}$$

(If $\boldsymbol{A} = \boldsymbol{I}$, then this is a **denoising** problem; if $\boldsymbol{A}$ is wide then it is compressed sensing.)

For these assumptions, two related but distinct ways to estimate $\boldsymbol{x}$ are:

$$\hat{\boldsymbol{x}} = \boldsymbol{D}\hat{\boldsymbol{z}}, \quad \hat{\boldsymbol{z}} = \arg\min_{\boldsymbol{z}} \frac{1}{2} \|\boldsymbol{A}\boldsymbol{D}\boldsymbol{z} - \boldsymbol{y}\|_2^2 + \beta \|\boldsymbol{z}\|_1 \tag{6.1}$$

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta R(\boldsymbol{x}), \quad R(\boldsymbol{x}) = \tag{6.2}$$

A potential advantage of (6.2) is that $\hat{\boldsymbol{x}}$ need not be in the range of $\boldsymbol{D}$.

We could write the second one (6.2) as this joint optimization problem:

$$(\hat{\boldsymbol{x}}, \hat{\boldsymbol{z}}) = \arg\min_{\boldsymbol{x},\boldsymbol{z}} \Psi(\boldsymbol{x}, \boldsymbol{z}), \quad \Psi(\boldsymbol{x}, \boldsymbol{z}) \triangleq \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta \left(\frac{1}{2} \|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{z}\|_2^2 + \alpha \|\boldsymbol{z}\|_1 \right). \tag{6.3}$$

There are at least 3 distinct ways to pursue this joint optimization problem.

- Rewrite the cost function in LASSO form (with diagonally weighted 1-norm) in terms of $\tilde{\boldsymbol{x}} = (\boldsymbol{x}, \boldsymbol{z})$

$$\Psi(\tilde{\boldsymbol{x}}) = \frac{1}{2} \|\boldsymbol{B}\tilde{\boldsymbol{x}} - \boldsymbol{b}\|_2^2 + \beta\alpha \|\boldsymbol{W}\tilde{\boldsymbol{x}}\|_1, \quad \boldsymbol{B} \triangleq \begin{bmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \sqrt{\beta}\boldsymbol{I} & -\sqrt{\beta}\boldsymbol{D} \end{bmatrix}, \quad \boldsymbol{b} \triangleq \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{0} \end{bmatrix}, \quad \boldsymbol{W} \triangleq \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}, \tag{6.4}$$

  and then apply any **proximal method** from Ch. 5, *e.g.*, POGM.
- If $\boldsymbol{D}$ is unitary, then we can solve analytically for the minimizer over $\boldsymbol{z}$ and substitute back in to get a cost function in terms of $\boldsymbol{x}$ only, where the regularizer (in this case) involves a **Huber function** (as seen in HW):

$$R(\boldsymbol{x}) = \boldsymbol{1}' \psi_{.}(\boldsymbol{D}'\boldsymbol{x}; \alpha).$$

  Then we can apply any gradient-based method (such as line search OGM) to that cost function. However, this approach is inapplicable in the general case of interest where $\boldsymbol{D}$ is wide (over-complete).
- Apply **alternating minimization** or **alternating descent**, aka **block coordinate minimization** (**BCM**) or **block coordinate descent** (**BCD**) to the "two block" cost function $\Psi(\boldsymbol{x}, \boldsymbol{z})$.

Here, both BCM and BCD start with some initial guesses $\boldsymbol{x}_0$ and $\boldsymbol{z}_0$ and then alternate updates. Reasonable initial guesses are application dependent, but one option is $\boldsymbol{x}_0 = c\boldsymbol{A}'\boldsymbol{y}$ for some constant c and $\boldsymbol{z}_0 = \boldsymbol{0}$.

BCM

For this “two block” cost function, the **BCM** algorithm is:

`for` $k =$ `0:niter-1`

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \underset{\boldsymbol{x}}{\arg\min}\, \Psi(\boldsymbol{x}, \boldsymbol{z}_k) \\ \boldsymbol{z}_{k+1} &= \underset{\boldsymbol{z}}{\arg\min} \end{aligned} \tag{6.5}$$

If implemented as written above, then this approach decreases the cost function monotonically:

$$\Psi(\boldsymbol{x}_{k+1}, \boldsymbol{z}_{k+1}) \leq$$

For convenience, the joint cost function (6.3) is repeated here:

$$\Psi(\boldsymbol{x}, \boldsymbol{z}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta \left(\frac{1}{2} \|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{z}\|_2^2 + \alpha \|\boldsymbol{z}\|_1 \right).$$

To apply **BCM** (6.5) to this cost function, the updates required are:

$$\text{regularized LS}: \quad \boldsymbol{x}_{k+1} = \underset{\boldsymbol{x}}{\arg\min}\, \Psi(\boldsymbol{x}, \boldsymbol{z}_k) =$$

$$\textbf{sparse coding}: \quad \boldsymbol{z}_{k+1} = \underset{\boldsymbol{z}}{\arg\min}\, \Psi(\boldsymbol{x}_{k+1}, \boldsymbol{z}) =$$

The **sparse coding** step above involves a **proximal operation**. (?)
A: True B: False $\boxed{??}$

For certain applications, the $\boldsymbol{x}$ update above is easy:
- image denoising: $\boldsymbol{A} = \boldsymbol{I}$
- single-coil MRI with Cartesian sampling: $\boldsymbol{A}'\boldsymbol{A}$ is circulant
- image inpainting: $\boldsymbol{A}'\boldsymbol{A}$ is diagonal
- image super-resolution: $\boldsymbol{A}'\boldsymbol{A}$ is block diagonal with small blocks

In those cases, inverting $\boldsymbol{A}'\boldsymbol{A} + \beta\boldsymbol{I}$ is easy, *i.e.*, $O(N)$ or $O(N \log N)$.
But for general $\boldsymbol{A}$ that inverse is $O(N^3)$ to compute exactly and infeasible if $\boldsymbol{A}$ is large, so one would have to apply an iterative method like CG for the $\boldsymbol{x}$ update.

However, regardless of $\boldsymbol{A}$, the $\boldsymbol{z}$ update above is a LASSO problem that requires an iterative solution in general, except for certain cases like when $\boldsymbol{D}$ happens to be unitary. We could use POGM to solve that inner LASSO problem but then one might ask why not just apply POGM to the joint LASSO problem (6.4)? One answer is that (6.4) involves $\boldsymbol{A}$ which can be very large and expensive, whereas the $\boldsymbol{z}$ update above involves only $\boldsymbol{D}$ which might be much faster.

BCD

For any finite number of inner iterations of CG for the $\boldsymbol{x}$ update, or a proximal method like POGM for the $\boldsymbol{z}$ update, the BCM algorithm above does not provide an exact minimization, so the name BCM would then seem inappropriate. When using a small number (perhaps just one) inner iteration for the $\boldsymbol{x}$ and/or $\boldsymbol{z}$ updates, a more appropriate term is **block coordinate descent** (**BCD**), which, for a two-block problem, is:

```
for $k =$ 0:niter
    Find $\boldsymbol{x}_{k+1}$ s.t. $\Psi(\boldsymbol{x}_{k+1}, \boldsymbol{z}_k) \leq \Psi(\boldsymbol{x}_k, \boldsymbol{z}_k)$
    Find $\boldsymbol{z}_{k+1}$ s.t. $\Psi(\boldsymbol{x}_{k+1}, \boldsymbol{z}_{k+1}) \leq \Psi(\boldsymbol{x}_{k+1}, \boldsymbol{z}_k)$.     (6.6)
```

Clearly this algorithm monotonically decreases the cost function by design: $\Psi(\boldsymbol{x}_{k+1}, \boldsymbol{z}_{k+1}) \leq \Psi(\boldsymbol{x}_k, \boldsymbol{z}_k)$.

Applying BCD (6.6) to (6.3) by using one GD update for $\boldsymbol{x}$ and one PGM update for $\boldsymbol{z}$ leads to the following simple algorithm:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \frac{\gamma}{\|\boldsymbol{A}\|_2^2 + \beta} \left(\boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{y}) + \beta(\boldsymbol{x}_k - \boldsymbol{D}\boldsymbol{z}_k)\right) \\ \boldsymbol{z}_{k+1} &= \operatorname{soft}_.\left(\boldsymbol{z}_k - \frac{1}{L}\boldsymbol{D}'(\boldsymbol{D}\boldsymbol{z}_k - \boldsymbol{x}_{k+1}); \frac{\alpha}{L}\right), \quad L = \end{aligned} \tag{6.7}$$

For any $0 < \gamma < 2$, the above algorithm is appropriately named BCD. (?)
A: True B: False $\boxed{??}$

Depending on the relative compute effort of working with $\boldsymbol{A}$ and $\boldsymbol{D}$, one could apply multiple inner GD iterations and/or PGM iterations and it will still be a BCD algorithm (for appropriate step sizes as shown above). Alternatively one could use MM updates with diagonal majorizers to avoid computing the spectral norms.

If we use multiple iterations of FGM or OGM for the $\boldsymbol{x}$ update, and/or multiple iterations of FISTA or POGM for the $\boldsymbol{z}$ update in (6.7), then the resulting algorithm is appropriately named BCD. (?)
A: True B: False $\boxed{??}$

The distinction between BCM and BCD is typically disregarded in the literature, but these notes will strive to use the terms appropriately for clarity.

Compressed sensing with analysis regularizer

Assuming that $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\varepsilon}$ and $\boldsymbol{T}\boldsymbol{x}$ is sparse, for some sparsifying transform matrix $\boldsymbol{T}$, the most natural formulation of analysis regularization is the following challenging optimization problem:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta \|\boldsymbol{T}\boldsymbol{x}\|_1 . \tag{6.8}$$

This is simple to solve only in special cases such as when the sparsifying transform $\boldsymbol{T}$ is unitary.
The literature is full of *approximate* solutions to (6.8), discussed next.

Corner rounding

The non-differentiability of the 1-norm is the primary challenge of (6.8), so one "approximate" approach is simply to replace the 1-norm with a smooth convex approximation ψ:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \Psi(\boldsymbol{x}), \quad \Psi(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta \boldsymbol{1}' \psi .(\boldsymbol{T}\boldsymbol{x}). \tag{6.9}$$

When ψ is smooth, we can apply fast methods like OGM and CG to Ψ easily. A reasonable choice for ψ is the Fair potential function, or the hyperbola $|z| \approx \sqrt{z^2 + \epsilon^2} - \epsilon$, and these can approximate the 1-norm very closely by taking δ or ϵ small enough. However, as ϵ decreases the global Lipschitz constant of $\Psi(\boldsymbol{x})$ increases, leading to slow convergence of methods like OGM that depend on the global Lipschitz constant. In contrast, CG can still work well because its step size is adaptive and depends only on the curvature along the search direction.

Variable splitting regularizer

Another option is to "split" the $\boldsymbol{T}$ matrix from the 1-norm term using a penalty function as follows:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta R(\boldsymbol{x}), \quad R(\boldsymbol{x}) = \min_{\boldsymbol{z}} \frac{1}{2} \|\boldsymbol{T}\boldsymbol{x} - \boldsymbol{z}\|_2^2 + \alpha \|\boldsymbol{z}\|_1 . \tag{6.10}$$

As α approaches 0, the solution more strongly enforces $\boldsymbol{z} \approx \boldsymbol{T}\boldsymbol{z}$ and the solution more closely approximates the original formulation (6.8). As shown in a HW problem, this formulation is mathematically equivalent to the corner rounding approach (6.9) with ψ chosen as a Huber function.

Balanced analysis/synthesis regularization (Read)

Recall that for the synthesis sparsity model, where we assume $\hat{\boldsymbol{x}} \approx \boldsymbol{D}\hat{\boldsymbol{z}}$, a typical formulation is

$$\hat{\boldsymbol{x}} = \boldsymbol{D}\hat{\boldsymbol{z}}, \qquad \hat{\boldsymbol{z}} = \arg\min_{\boldsymbol{z}} \frac{1}{2} \|\boldsymbol{A}\boldsymbol{D}\boldsymbol{z} - \boldsymbol{y}\|_2^2 + \beta \|\boldsymbol{z}\|_1 .$$

If $\boldsymbol{D}$ is a **Parseval tight frame**, for which $\boldsymbol{D}\boldsymbol{D}' = \boldsymbol{I}$, an alternative formulation that is called the **balanced model** [2–6] is:

$$\hat{\boldsymbol{x}} = \boldsymbol{D}\hat{\boldsymbol{z}}, \qquad \hat{\boldsymbol{z}} = \arg\min_{\boldsymbol{z}} \frac{1}{2} \|\boldsymbol{A}\boldsymbol{D}\boldsymbol{z} - \boldsymbol{y}\|_2^2 + \beta \|\boldsymbol{z}\|_1 + \alpha \frac{1}{2} \|(\boldsymbol{I} - \boldsymbol{D}'\boldsymbol{D})\boldsymbol{z}\|_2^2 . \tag{6.11}$$

This cost function is essentially equivalent to (5.1).
- When $\alpha = 0$, this reverts to the synthesis model.

- As $\alpha \to \infty$, the additional term enforces the constraint $\boldsymbol{z} = \boldsymbol{D}'\boldsymbol{D}\boldsymbol{z} \Longrightarrow \|\boldsymbol{z}\|_1 = \|\boldsymbol{D}'\boldsymbol{D}\boldsymbol{z}\|_1 = \|\boldsymbol{D}'\boldsymbol{x}\|_1$, which is a particular type of analysis regularizer $\|\boldsymbol{T}\boldsymbol{x}\|_1$, where $\boldsymbol{T} = \boldsymbol{D}'$.
- For any finite α, this formulation is yet another approximation to the analysis regularizer formulation, and even then only in the case of a Parseval tight frame.

Results in [6] for compressed sensing MRI using shift-invariant wavelets for $\boldsymbol{D}$ showed no empirical performance advantages of the "balanced" approach over an analysis regularizer.

If $\boldsymbol{D}$ above is unitary, then the "balanced" formulation (6.11) is equivalent to a usual synthesis (?) and/or analysis (?) formulations?

A: T,T B: T,F C: F,T D: F,F $\boxed{??}$

Optimization strategies

We have at least three viable options for optimizing the variable split form (6.10).

- Rewrite as a **joint cost function** in terms of $\tilde{\boldsymbol{x}} = (\boldsymbol{x}, \boldsymbol{z})$ and apply any proximal method like **POGM** to that joint cost function:

$$\Psi(\tilde{\boldsymbol{x}}) = \frac{1}{2}\|\boldsymbol{B}\tilde{\boldsymbol{x}} - \boldsymbol{b}\|_2^2 + \beta\alpha\|\boldsymbol{W}\tilde{\boldsymbol{x}}\|_1, \quad \boldsymbol{b} \triangleq \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{0} \end{bmatrix}, \quad \boldsymbol{W} \triangleq \begin{bmatrix} \boldsymbol{0} & \\ & \boldsymbol{I} \end{bmatrix}, \quad \boldsymbol{B} \triangleq$$

- Minimize over $\boldsymbol{z}$ and plug back in leading to a Huber function regularizer in $\boldsymbol{x}$. Then apply any gradient-based method like line-search **OGM** to optimize over $\boldsymbol{x}$.
  Letting ψ denote that Huber function, so $R(\boldsymbol{x}) = \mathbf{1}' \psi_{.}(\boldsymbol{T}\boldsymbol{x}; \alpha)$, the gradient here is
  $$\nabla \Psi(\boldsymbol{x}) = \boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}) + \beta \boldsymbol{T}' \dot{\psi}_{.}(\boldsymbol{T}\boldsymbol{x}; \alpha),$$
  for which there is no non-iterative solution if set to zero, unless $\alpha = 0$, so iterative method are required.
- Write a two-block cost function and apply **BCD** or **BCM** to it; here is that cost function:
  $$\Psi(\boldsymbol{x}, \boldsymbol{z}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta \left(\frac{1}{2} \|\boldsymbol{T}\boldsymbol{x} - \boldsymbol{z}\|_2^2 + \alpha \|\boldsymbol{z}\|_1 \right).$$

The $\boldsymbol{z}$ update is "easy," *i.e.*, non-iterative even for large problems. (?)
A: True B: False $\boxed{??}$

In general the $\boldsymbol{z}$ update is

$$\boldsymbol{z}_{k+1} =$$

For the denoising case where $\boldsymbol{A} = \boldsymbol{I}$, the $\boldsymbol{x}$ update is typically "easy," *i.e.*, non-iterative even for large problems. (?)
A: True B: False $\boxed{??}$

In general the $\boldsymbol{x}$ update is

$$\boldsymbol{x}_{k+1} =$$

One case where this update is easy is when both $\boldsymbol{A}'\boldsymbol{A}$ and $\boldsymbol{T}'\boldsymbol{T}$ are *circulant*, because then we can use FFT operations for the inverse. $\boldsymbol{A}'\boldsymbol{A}$ is circulant, *e.g.*, for denoising, for single-coil Cartesian MRI, for deblurring with periodic boundary conditions. $\boldsymbol{T}'\boldsymbol{T}$ circulant when $\boldsymbol{T}$ is unitary and when $\boldsymbol{T}$ corresponds to finite differences with periodic boundary conditions, Often the Hessian is Toeplitz, *i.e.*, approximately circulant, and we can use a circulant preconditioner for CG.

Sparse coding revisited with multi-block BCM

Recall that the **sparse coding** step of **BCM** for synthesis-based regularization is a LASSO problem:

$$\hat{\boldsymbol{z}} = \underset{\boldsymbol{z} \in \mathbb{F}^K}{\arg\min} \frac{1}{2} \|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{z}\|_2^2 + \alpha \|\boldsymbol{z}\|_1 ,$$

where $\boldsymbol{D} \in \mathbb{F}^{N \times K}$, for which there is no closed-form solution, so iterative methods like POGM are required.

Recall this inner minimization arose from a "two-block" joint cost function $\Psi(\boldsymbol{x}, \boldsymbol{z})$ with corresponding two-way alternating minimization. Another approach is to think of the cost function as having $K + 1$ blocks: $\Psi(\boldsymbol{x}, z_1, \ldots, z_K)$, and to implement BCM or BCD by updating one of these (now much smaller) blocks at a time:

```
for $t =$ 0:niter      (outer loop over iteration)
    $\boldsymbol{x}^{(t+1)} = \underset{\boldsymbol{x} \in \mathbb{F}^N}{\arg\min} \Psi\left(\boldsymbol{x}, z_1^{(t)}, \ldots, z_K^{(t)}\right)$
    for $k =$ 1:K (inner loop over coefficients)
        $z_k^{(t+1)} = \underset{z_k \in \mathbb{F}}{\arg\min} \Psi\left(\boldsymbol{x}^{(t+1)}, z_1^{(t+1)}, z_2^{(t+1)}, \ldots, z_{k-1}^{(t+1)}, z_k, z_{k+1}^{(t)}, \ldots, z_K^{(t)}\right).$
```

As written, this algorithm is guaranteed to monotonically decrease Ψ. (?)

A: True B: False $\boxed{??}$

Note that $\boldsymbol{D}\boldsymbol{z} = \sum_{j=1}^{K} \boldsymbol{d}_j z_j$ where $\boldsymbol{d}_j = D[:, j]$.
Now focus on updating one z_k coefficient by defining

$$f_k(z_k) = g(z_1, \ldots, z_{k-1}, z_k, z_{k+1}, \ldots, z_K) = \frac{1}{2} \left\| \boldsymbol{x} - \sum_{j \neq k} \boldsymbol{d}_j z_j - \boldsymbol{d}_k z_k \right\|_2^2 + \alpha \left(\sum_{j \neq k} |z_j| + |z_k| \right).$$

Define $\boldsymbol{r} \triangleq \boldsymbol{x} - \sum_{k=1}^{K} \boldsymbol{d}_k z_k^{(t)}$ and $\boldsymbol{r}_k \triangleq \boldsymbol{r} + \boldsymbol{d}_k z_k^{(t)}$, then ignoring constants independent of z_k:

$$\begin{aligned} f_k(z_k) &= \frac{1}{2} \|\boldsymbol{r}_k - \boldsymbol{d}_k z_k\|_2^2 + \alpha |z_k| + c_1 \\ &= \frac{1}{2} \|\boldsymbol{r}_k\|_2^2 - \mathrm{real}\{\boldsymbol{r}_k' \boldsymbol{d}_k z_k\} + \frac{1}{2} |z_k|^2 \|\boldsymbol{d}_k\|_2^2 + \alpha |z_k| \\ &= \end{aligned}$$

Thus the update for z_k is

$$z_k^{(t+1)} = \mathsf{soft}\left(\tilde{\boldsymbol{d}}_k' \boldsymbol{r}_k, \alpha / \|\boldsymbol{d}_k\|_2^2 \right).$$

For an efficient implementation we keep $\boldsymbol{r}$ updated as a state vector, something like this:

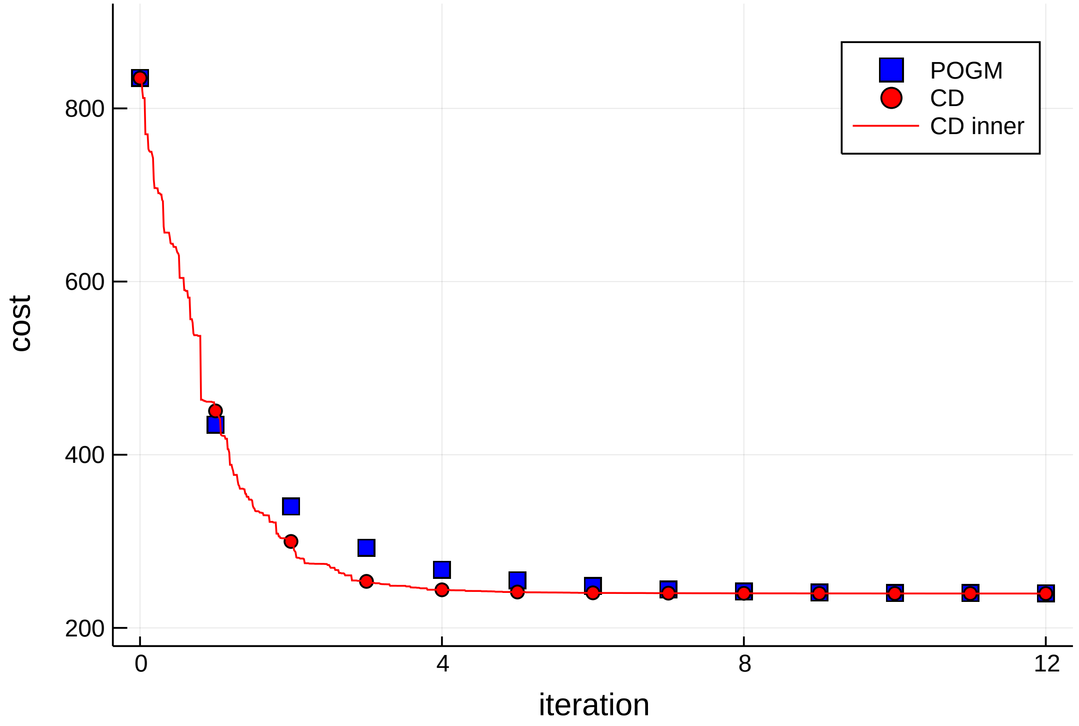
$$\boldsymbol{r} = \boldsymbol{r}_k - \boldsymbol{d}_k z_k^{(t+1)}.$$

Here is JULIA code

```
# argmin_x 0.5*|A x - y|^2 + reg |x|_1
function sparse_code_cd(y, A, x0::AbstractVector{<:Number}, reg::Real)
  N = length(x0)
  vr = y - A * x0 # residual vector
  norm2 = [norm(A[:,n])^2 for n=1:N] # normalize
  D = hcat([A[:,n]/norm2[n] for n=1:N]...) # normalize
  x = copy(x0)

  for iter=1:niter # outer loop over iteration
    for n=1:N # inner loop over elements
      an = A[:,n]
      vr += an * x[n] # r_k
      x[n] = soft(D[:,n]'*vr, reg/norm2[n])
      vr -= an * x[n] # full residual again
    end
  end
  return x
end
```

Example. Same data as HW that compared POGM and CLS, where $N = 50$, $K = 99$ and 32/99 coefficients are zero. CD converges faster than POGM in terms of reducing cost per iteration. But wall time?



CD approach to $\boldsymbol{x}$ update

Recall that the $\boldsymbol{x}$ update in general required inverting a large matrix involving $\boldsymbol{A}'\boldsymbol{A}$ for the synthesis form:

$$\Psi(x_1, \ldots, x_N, \boldsymbol{z}) = \Psi(\boldsymbol{x}, \boldsymbol{z}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta \left(\frac{1}{2} \|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{z}\|_2^2 + \alpha \|\boldsymbol{z}\|_1 \right).$$

A multi-block approach, aka **coordinate descent**, can also avoid this matrix inverse. The update for x_n is:

$$x_n^{(t+1)} = \operatorname*{arg\,min}_{x_n \in \mathbb{F}} \Psi\left(x_1^{(t+1)}, \ldots, x_{n-1}^{(t+1)}, x_n, x_{n+1}^{(t)}, \ldots, x_N^{(t)} \right) = \operatorname*{arg\,min}_{x_n \in \mathbb{F}} f_n(x_n)$$
$$f_n(x_n) = \frac{1}{2} \left\| \boldsymbol{A}\boldsymbol{x}^{(t,n)} + \boldsymbol{a}_n \left(x_n - x_n^{(t)}\right) - \boldsymbol{y} \right\|_2^2 + \beta \frac{1}{2} \left\| \boldsymbol{x}^{(t,n)} + \boldsymbol{a}_n \left(x_n - x_n^{(t)}\right) - \boldsymbol{D}\boldsymbol{z} \right\|_2^2,$$

where $\boldsymbol{a}_n = \boldsymbol{A}[:, n]$ and $\boldsymbol{x}^{(t,n)} = \left(x_1^{(t+1)}, \ldots, x_{n-1}^{(t+1)}, x_n^{(t)}, x_{n+1}^{(t)}, \ldots, x_N^{(t)} \right)$ contains all the most recent values.

In-class group work on $\boldsymbol{x}$ update:

Sparse coding for tight frames

The **sparse coding** problem is:

$$\hat{\boldsymbol{z}} = \operatorname*{arg\,min}_{\boldsymbol{z} \in \mathbb{F}^K} \frac{1}{2} \|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{z}\|_2^2 + R(\boldsymbol{z}),$$

for some regularizer such as $R(\boldsymbol{z}) = \alpha \|\boldsymbol{z}\|_1$ or $R(\boldsymbol{z}) = \alpha \|\boldsymbol{z}\|_0$.
So far we have discussed 2 ways to approach this optimization problem:

- proximal methods like **POGM** that update *all* coefficients $\boldsymbol{z}$ simultaneously,
- multi-block **BCM** where we update *one* coefficient z_k at a time, sequentially.

These two options represent two extremes of parallel versus sequential; there are also "in-between" options.

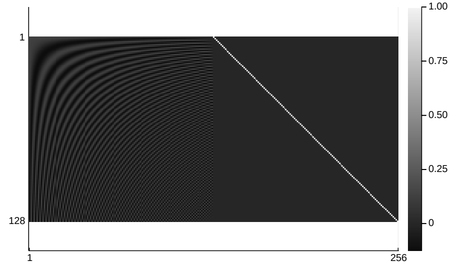
Consider the case where the dictionary $\boldsymbol{D}$ is a **tight frame** consisting of two $N \times N$ unitary matrices: $\boldsymbol{D} = \begin{bmatrix} \boldsymbol{D}_1 & \boldsymbol{D}_2 \end{bmatrix}$. In the usual case where R is **additively separable**, we can rewrite the sparse coding problem as:

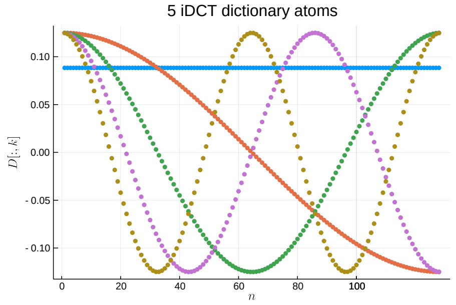
$$(\hat{\boldsymbol{z}}_1, \hat{\boldsymbol{z}}_2) = \operatorname*{arg\,min}_{\boldsymbol{z}_1, \boldsymbol{z}_2 \in \mathbb{F}^N}$$

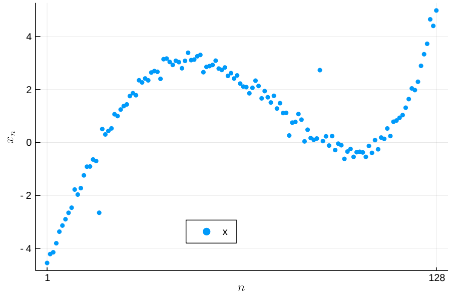
There is still no joint closed-form solution here. But because $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$ are unitary, it is very easy to perform two-block BCM where we alternate between updating $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$. The $\boldsymbol{z}_1$ update is $\operatorname{prox}_R(\boldsymbol{D}_1'(\boldsymbol{x} - \boldsymbol{D}_2\boldsymbol{z}_2))$.

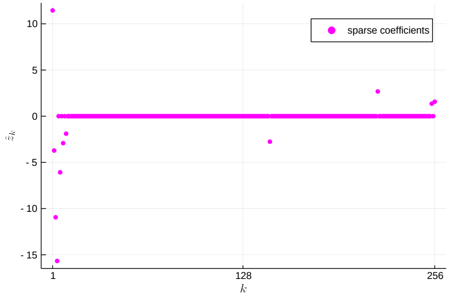
Example.

An in-class task will focus on a waves+spikes application for signals that are smooth + some impulses, where $\boldsymbol{D}_1$ is the (inverse) **DCT** matrix and $\boldsymbol{D}_2 = \boldsymbol{I}$, both of which are unitary matrices.



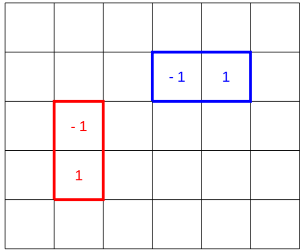




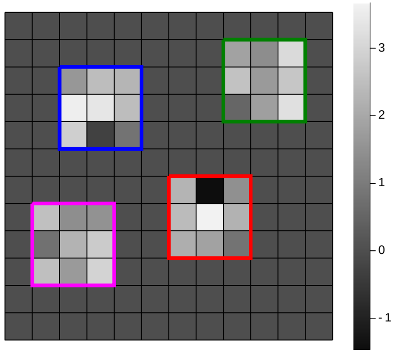


Patch-based regularization: analysis form

Using **TV** regularizer $R(\boldsymbol{x}) = \|\boldsymbol{T}\boldsymbol{x}\|_1$
where $\boldsymbol{T}$ is 1st-order finite-differences
$\equiv$ **patches** of size 2×1.



Larger patches provide more context
for distinguishing signal from noise.



cf. CNN approaches

Patch-based regularizers:
- **synthesis** models
- **analysis** methods

Especially for data-driven models, often it is more appropriate to analyze / regularize each **patch** of an image rather than trying to model the entire image.

For the model "$\boldsymbol{T}\boldsymbol{R}_p\boldsymbol{x}$ tends to be sparse," a typical patch-based analysis (or sparsifying transform) regularizer is:

$$\hat{\boldsymbol{x}} = \mathop{\arg\min}_{\boldsymbol{x}\in\mathbb{F}^N} \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{y}\|_2^2 + \beta R(\boldsymbol{x}), \qquad R(\boldsymbol{x}) = \min_{\boldsymbol{Z}\in\mathbb{F}^{K\times P}} \sum_{p=1}^{P} \frac{1}{2}\|\boldsymbol{T}\boldsymbol{R}_p\boldsymbol{x}-\boldsymbol{z}_p\|_2^2 + \alpha\phi(\boldsymbol{z}_p), \tag{6.12}$$

where $\boldsymbol{T}$ is a $K\times d$ sparsifying transform matrix for **vectorized** patches and $\boldsymbol{Z} = [\boldsymbol{z}_1 \ \ldots \ \boldsymbol{z}_K]$. Often $K \approx d$. Here $\phi(\cdot)$ is a sparsity regularizer such as $\|\cdot\|_0$ or $\|\cdot\|_1$ or $\|\boldsymbol{W}\cdot\|_0$ for some diagonal weighting matrix $\boldsymbol{W}$ like we used with the wavelet transform.

Example. 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. When we use (6.12), we are hoping to outperform methods like TV. (In 2D we would need both 2×1 and 1×2 patches.)

We write $\boldsymbol{R}_p$ as a matrix above, but $\boldsymbol{R}_p\boldsymbol{x}$ is yet another linear operation that we implement efficiently in code, not as matrix-vector multiplication.

If $\boldsymbol{X}$ is a 2D image of size $N_x \times N_y$ and $\boldsymbol{x} = \text{vec}(\boldsymbol{X})$ and we want to use a patch size $p_x \times p_y$, for which $N = N_x N_y$ and $d = d_x d_y$, the code for computing $\boldsymbol{R}_1\boldsymbol{x}$ is `reshape(x, Nx, Ny)[1:px,1:py]` and for $\boldsymbol{R}_2\boldsymbol{x}$ is `reshape(x, Nx, Ny)[((1:px)+1),1:py]` etc. (See Ch. 2.)

In practice one could use something like MATLAB's `im2col` function to extract all the patches. (See next page for more memory efficient way.) There are many versions online for JULIA:

https://discourse.julialang.org/t/what-is-julias-im2col/14066
https://github.com/pluskid/Mocha.jl/blob/master/benchmarks/native-im2col/im2col-bm.jl
https://github.com/outyang/MatlabFun.jl/blob/master/im2col.jl

BCD/BCM for patch-based analysis regularization

To perform the joint optimization problem (6.12), BCD/BCM are natural algorithm choices.
We alternate between updating the image $\boldsymbol{x}$ and updating the sparse coefficients $\boldsymbol{z}$.

- The Hessian of (6.12) w.r.t. $\boldsymbol{x}$ is $\boldsymbol{A}'\boldsymbol{A} + \beta \boldsymbol{D}$ where $\boldsymbol{D} \triangleq \sum_{p=1}^{P} \boldsymbol{R}_p' \boldsymbol{T}' \boldsymbol{T} \boldsymbol{R}_p$.

  If that Hessian does not happen to have an easy inverse, what algorithm is the most natural choice for updating $\boldsymbol{x}$?
  A: GD B: (P)SD C: (P)CG D: OGM E: POGM $\boxed{??}$

  If $\boldsymbol{T}$ is unitary, then $\boldsymbol{D} = \sum_{p=1}^{P} \boldsymbol{R}_p' \boldsymbol{R}_p$ is a $N \times N$ diagonal, where the nth diagonal element is the number of patches that contain the nth pixel. If we choose patches with stride=1 and periodic boundary conditions, then that number is always d, the patch size, so $\boldsymbol{D} = d\boldsymbol{I}$. Otherwise it is at most d, and $\boldsymbol{D} \preceq d\boldsymbol{I}$.
  So if $\boldsymbol{A}'\boldsymbol{A}$ is also diagonal (*e.g.*, denoising, inpainting, single-coil Cartesian MRI), then the $\boldsymbol{x}$ update is an exact minimization.

- The $\boldsymbol{z}_p$ updates are an **embarrassingly parallel** proximal operation:

$$\boldsymbol{z}_p^{(t+1)} =$$

In JULIA this operation is simply: `Z = mapslices(prox, T*Xpatch, dims=1)`
where `Xpatch` $= \begin{bmatrix} \boldsymbol{R}_1\boldsymbol{x} & \ldots \boldsymbol{R}_P\boldsymbol{x} \end{bmatrix} \in \mathbb{F}^{d\times P}$ and `prox` is a `Function` that computes the proximity operator of $\alpha\phi(\cdot)$.
If ϕ is additively separable, then the code simplifies further to `Z = prox.(T*Xpatch)`

Practical implementation (Read)

As written, this BCD/BCM approach would be memory intensive because it stores $\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{z}_1 & \ldots \boldsymbol{z}_P \end{bmatrix}$.
Careful implementation can reduce the memory greatly, at least when $\boldsymbol{T}$ is unitary.

Consider the part of the regularizer in (6.12) involving $\boldsymbol{x}$ at iteration t:

$$f(\boldsymbol{x}) \triangleq \sum_{p=1}^P \frac{1}{2} \left\| \boldsymbol{T}\boldsymbol{R}_p\boldsymbol{x} - \boldsymbol{z}_p^{(t)} \right\|_2^2 \Longrightarrow \nabla f(\boldsymbol{x}) = \sum_{p=1}^P \boldsymbol{R}_p'\boldsymbol{T}' \left(\boldsymbol{T}\boldsymbol{R}_p\boldsymbol{x} - \boldsymbol{z}_p^{(t)}\right) = \boldsymbol{D}\left(\boldsymbol{x} - \tilde{\boldsymbol{x}}^{(t)}\right)$$

$$\tilde{\boldsymbol{x}}^{(t)} \triangleq \boldsymbol{D}^{-1} \sum_{p=1}^P \boldsymbol{R}_p'\boldsymbol{T}'\boldsymbol{z}_p^{(t)} = \boldsymbol{D}^{-1} \sum_{p=1}^P \boldsymbol{R}_p'\boldsymbol{T}' \operatorname{prox}_{\alpha\phi}\left(\boldsymbol{T}\boldsymbol{R}_p\boldsymbol{x}^{(t-1)}\right).$$

This summation form means that we can extract one (or several) of the patches from $\boldsymbol{x}^{(t-1)}$ at time, apply the transform, threshold, and inverse transform, and put the result into an **accumulator** $\tilde{\boldsymbol{x}}^{(t)}$ that is the same size as $\boldsymbol{x}$, and finally apply $\boldsymbol{D}^{-1}$ (typically just $1/d$). We never need to store all of $\boldsymbol{Z}$.

Clearly $\nabla^2 f = \boldsymbol{D}$ so we can also write

$$f(\boldsymbol{x}) = \frac{1}{2} \left\| \boldsymbol{x} - \tilde{\boldsymbol{x}}^{(t)} \right\|_{\boldsymbol{D}}^2 + c,$$

so the $\boldsymbol{x}$ update is simply

$$\boldsymbol{x}^{(t+1)} = \arg\min_{\boldsymbol{x}} \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta \frac{1}{2} \left\| \boldsymbol{x} - \tilde{\boldsymbol{x}}^{(t)} \right\|_{\boldsymbol{D}}^2 = (\boldsymbol{A}'\boldsymbol{A} + \beta \boldsymbol{D})^{-1} (\boldsymbol{A}'\boldsymbol{y} + \beta \boldsymbol{D} \tilde{\boldsymbol{x}}^{(t)}).$$

When $\boldsymbol{D} = d\boldsymbol{I}$ this simplifies to

$$\boldsymbol{x}^{(t+1)} = (\boldsymbol{A}'\boldsymbol{A} + \beta d \boldsymbol{I})^{-1} (\boldsymbol{A}'\boldsymbol{y} + \beta d \tilde{\boldsymbol{x}}^{(t)}). \tag{6.13}$$

(Read)

Example. For single-coil Cartesian MRI, where $\boldsymbol{A} = \boldsymbol{P}\boldsymbol{F}$ where $\boldsymbol{F}^{-1} = \frac{1}{N}\boldsymbol{F}'$ and where $\boldsymbol{P}$ is the $K \times N$ sample selection matrix, for which $\boldsymbol{P}'\boldsymbol{P}$ is diagonal, the update (6.13) further simplifies to

$$\begin{aligned}
\boldsymbol{x}^{(t+1)} &= (\boldsymbol{F}'\boldsymbol{P}'\boldsymbol{P}\boldsymbol{F} + \beta d \boldsymbol{I})^{-1}(\boldsymbol{F}'\boldsymbol{P}'\boldsymbol{y} + \beta d \tilde{\boldsymbol{x}}^{(t)}) = (\boldsymbol{F}'\boldsymbol{P}'\boldsymbol{P}\boldsymbol{F} + \beta \frac{d}{N}\boldsymbol{F}'\boldsymbol{F})^{-1}(\boldsymbol{F}'\boldsymbol{P}'\boldsymbol{y} + \beta d \tilde{\boldsymbol{x}}^{(t)}) \\
&= \boldsymbol{F}^{-1}(\boldsymbol{P}'\boldsymbol{P} + \beta \frac{d}{N}\boldsymbol{I})^{-1}(\boldsymbol{F}')^{-1}(\boldsymbol{F}'\boldsymbol{P}'\boldsymbol{y} + \beta d \tilde{\boldsymbol{x}}^{(t)}) = \boldsymbol{F}^{-1}(\boldsymbol{P}'\boldsymbol{P} + \beta \frac{d}{N}\boldsymbol{I})^{-1}(\boldsymbol{P}'\boldsymbol{y} + \beta \frac{d}{N}\boldsymbol{F}\tilde{\boldsymbol{x}}^{(t)}) \\
&= \boldsymbol{F}^{-1}\boldsymbol{v}, \quad v_k = \begin{cases} \frac{1}{1+\beta d/N}([\boldsymbol{P}'/\boldsymbol{y}]_k + (\beta d/N)[\boldsymbol{F}\tilde{\boldsymbol{x}}^{(t)}]_k), & k \in \Omega \\ [\boldsymbol{F}\tilde{\boldsymbol{x}}^{(t)}]_k, & k \notin \Omega. \end{cases}
\end{aligned}$$

From this expression, small β seems desirable, *i.e.*, $\beta d/N \ll 1$.

(Read)

If $\boldsymbol{T}$ is not unitary, then in general the matrix $\boldsymbol{D}$ above is not diagonal but we still have that

$$\boldsymbol{D}\boldsymbol{x} = \sum_{p=1}^{P} \boldsymbol{R}_p'\boldsymbol{T}'\boldsymbol{T}\boldsymbol{R}_p\boldsymbol{x}$$

and this summation could be done incrementally (one or several patches at a time) instead of extracting all patches at once, to save memory.

Nonlinear models for patches based on artificial **neural networks** are a recent trend [7].

Sparsifying transform learning

So far we have considered a dictionary $\boldsymbol{D}$ or a sparsifying transform $\boldsymbol{T}$ to be "given," but often we want to **learn** $\boldsymbol{T}$ from training data. Given a set of training examples (typically image patches) $\boldsymbol{X} \triangleq \begin{bmatrix} \boldsymbol{x}_1 & \ldots & \boldsymbol{x}_L \end{bmatrix} \in \mathbb{F}^{d \times L}$, we want to find a transform $\boldsymbol{T} \in \mathbb{F}^{K \times d}$ such that the transform coefficients $\{\boldsymbol{z}_l = \boldsymbol{T}\boldsymbol{x}_l\}$ are typically **sparse**. This process is called **sparsifying transform learning**. Often $K = d$ but we also consider other cases here. Let $\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{z}_1 & \ldots & \boldsymbol{z}_L \end{bmatrix} \in \mathbb{F}^{K \times L}$ denote the transform coefficient matrix. A typical transform learning optimization problem is [8, 9]:

$$\hat{\boldsymbol{T}} = \operatorname*{arg\,min}_{\boldsymbol{T} \in \mathcal{T}} \min_{\boldsymbol{Z} \in \mathbb{F}^{K \times L}} \Psi(\boldsymbol{T}, \boldsymbol{Z}), \qquad \Psi(\boldsymbol{T}, \boldsymbol{Z}) \triangleq \tag{6.14}$$

where the "arg min" and "min" above are deliberately different and ϕ is some sparsity regularizer like $\|\cdot\|_0$. An alternative formulation that looks simpler (no α choice), but perhaps harder to optimize, is:

$$\hat{\boldsymbol{T}} = \operatorname*{arg\,min}_{\boldsymbol{T} \in \mathcal{T}} \Psi(\boldsymbol{T}), \quad \Psi(\boldsymbol{T}) = \sum_{l=1}^{L} \phi(\boldsymbol{T}\boldsymbol{x}_l).$$

For transform learning, we need to avoid a scale ambiguity *and* and we would like to ensure that the rows of $\boldsymbol{T}$ are not redundant. So one natural approach is to use the following (row) orthonormality constraint:

$$\mathcal{T} = \left\{ \boldsymbol{T} \in \mathbb{F}^{K \times d} \,:\, \boldsymbol{T}\boldsymbol{T}' = \boldsymbol{I}_K \right\}. \tag{6.15}$$

With this choice of $\mathcal{T}$ the problem (6.14) is always nonconvex, even if ϕ is the convex 1-norm.

For the constraint (6.15) to hold, we must have (choose most general correct condition):
A: $K \leq d$ B: $K \geq d$ C: $K = d$ D: $K \neq d$ E: $d, K \in \mathbb{N}$ $\boxed{\textbf{??}}$

Some authors consider tall $\boldsymbol{T}$, called an "over-complete" transform [10].

Example. The following matrix $\boldsymbol{T} \in \mathcal{T}$ satisfies the constraint for $K = 2$ and $d = 5$:

$$\boldsymbol{T}_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix}.$$

Think of each row as a filter that we hope can sparsify patches extracted from images.

This example $\boldsymbol{T}_2$ is in $\mathcal{T}$, but still has some redundancy in it from a filtering perspective. If we use this $\boldsymbol{T}_2$ as part of a regularizer where we we extract signal patches of size 5×1 with a stride of 1 pixel (see Ch. 2), then we would get the same results using this simpler sparsifying transform

$$\boldsymbol{T}_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \end{bmatrix},$$

with β adjusted by a factor of 2.
How to design $\mathcal{T}$ to encourage less redundancy is an active research topic. See [11] for a Fourier approach.

Two-block BCM for transform learning

There is no closed-form joint solution for $\boldsymbol{T}$ and $\boldsymbol{Z}$ in the transform learning problem (6.14), but its form suggests applying two-block **BCM**. Repeating (6.14) here for convenience:

$$\Psi(\boldsymbol{T}, \boldsymbol{Z}) \triangleq \sum_{l=1}^{L} \frac{1}{2} \|\boldsymbol{T}\boldsymbol{x}_l - \boldsymbol{z}_l\|_2^2 + \alpha\phi(\boldsymbol{z}_l), \text{ s.t. } \boldsymbol{T} \in \mathcal{T}.$$

- The $\boldsymbol{Z}$ update is an **embarrassingly parallel** proximal operation:

$$\boldsymbol{z}_l^{(t+1)} =$$

  In JULIA this operation is simply: `Z = mapslices(prox, T*X, dims=1)`
  where `prox` is a `Function` that computes the proximity operator of $\alpha\phi(\cdot)$.
  If ϕ is additively separable, then the code simplifies further to `Z = prox.(T*X)`
- Now consider the $\boldsymbol{T}$ update:

$$\hat{\boldsymbol{T}} = \arg\min_{\boldsymbol{T}\in\mathcal{T}} \sum_{l=1}^{L} \|\boldsymbol{T}\boldsymbol{x}_l - \boldsymbol{z}_l\|_2^2 = \qquad (6.16)$$

  Because of the $\boldsymbol{X}$ multiplying $\boldsymbol{T}$, this is not a proximal operator.
  It is *almost* a problem you have solved already! HW1#9
  What is the name of this problem? $\boxed{??}$
  Why $\boxed{??}$ Why almost? $\boxed{??}$

Square transform learning: T update

The solution for the $\boldsymbol{T}$ update (6.16) in the square case ($K = d$) uses the following **SVD**:

$$\hat{\boldsymbol{T}} = \boldsymbol{U}\boldsymbol{V}', \quad \underbrace{\boldsymbol{Z}}_{d \times L}\underbrace{\boldsymbol{X}'}_{L \times d} = \underbrace{\boldsymbol{U}}_{d \times d}\underbrace{\boldsymbol{\Sigma}}_{d \times d}\underbrace{\boldsymbol{V}'}_{d \times d}. \tag{6.17}$$

As a sanity check: $\hat{\boldsymbol{T}}\hat{\boldsymbol{T}}' = \boldsymbol{U}\boldsymbol{V}'\boldsymbol{V}\boldsymbol{U}' = \boldsymbol{I}_d$.

Does the use of an SVD here preclude large-scale problems?
Typically $K = d \ll L$, *e.g.*, $K = d = 8^2$ for 8×8 patches whereas L can greatly exceed 10^6.
The most expensive part is the matrix multiplication $\boldsymbol{Z}\boldsymbol{X}'$ that is $O(d^2 L)$, whereas the $d \times d$ SVD is $O(d^3)$.
For 8×8 patches doing a $8^2 \times 8^2$ SVD is trivial.

Here is a summary of the BCM algorithm for (square) transform learning with $K = d$:
- Apply the **proximal operator** (*e.g.*, soft or hard thresholding) to each column of $\boldsymbol{T}\boldsymbol{X}$ to get new $\boldsymbol{Z}$.
- Apply orthogonal **Procrustes method** using SVD of the product $\boldsymbol{Z}\boldsymbol{X}'$ to get new $\boldsymbol{T}$.
- Repeat until convergence.

See [8] for some convergence theory for this alternating method, even for $\phi(\boldsymbol{z}) = \|\boldsymbol{z}\|_0$.

This method for updating $\boldsymbol{T}$,$\boldsymbol{Z}$ should be named **BCD** instead of **BCM**. (?)
A: True B: False $\boxed{??}$

Non-square transform learning

In the non-square case, specifically where $K < d$, the generalized orthogonal Procrustes solution from EECS 551 and HW1#9 (`stiefl`) is inapplicable. Those solutions were for the case where $\boldsymbol{T}$ is tall ($K \geq d$ here), with the constraint $\boldsymbol{T}'\boldsymbol{T} = \boldsymbol{I}_d$.

The derivation there does not generalize readily to handle $\boldsymbol{T}\boldsymbol{T}' = \boldsymbol{I}_K$. To synopsize the issue, consider:

$$\begin{aligned}\|\boldsymbol{T}\boldsymbol{X} - \boldsymbol{Z}\|_{\mathrm{F}}^2 &= \operatorname{trace}\{(\boldsymbol{T}\boldsymbol{X} - \boldsymbol{Z})' (\boldsymbol{T}\boldsymbol{X} - \boldsymbol{Z})\} \\ &= \operatorname{trace}\{\boldsymbol{X}'\boldsymbol{T}'\boldsymbol{T}\boldsymbol{X}\} - 2\,\mathrm{real}\{\operatorname{trace}\{\boldsymbol{T}\boldsymbol{X}\boldsymbol{Z}'\}\} + \operatorname{trace}\{\boldsymbol{Z}'\boldsymbol{Z}\}.\end{aligned}$$

The previous solution used the constraint $\boldsymbol{T}'\boldsymbol{T} = \boldsymbol{I}_d$ to simplify the first term. Here we have the constraint $\boldsymbol{T}\boldsymbol{T}' = \boldsymbol{I}_K$ that does not help simplify the first term in general.
In the square case, $\boldsymbol{T}'\boldsymbol{T} = \boldsymbol{I} \iff \boldsymbol{T}\boldsymbol{T}' = \boldsymbol{I}$, so the previous solution applies, but not more generally.

One possible approach is to use the trace circular commutative property to write the first term as

$$\begin{aligned}\operatorname{trace}\{\boldsymbol{X}'\boldsymbol{T}'\boldsymbol{T}\boldsymbol{X}\} &= \operatorname{trace}\{\boldsymbol{T}\boldsymbol{X}\boldsymbol{X}'\boldsymbol{T}'\} \leq \operatorname{trace}\{\boldsymbol{T}\boldsymbol{\Pi}\boldsymbol{T}'\} + \operatorname{trace}\{(\boldsymbol{T} - \boldsymbol{T}_k)(\rho\boldsymbol{I}_d - \boldsymbol{\Pi})(\boldsymbol{T} - \boldsymbol{T}_k)'\} \\ &= -2\,\mathrm{real}\{\operatorname{trace}\{\boldsymbol{T}(\rho\boldsymbol{I}_d - \boldsymbol{\Pi})\boldsymbol{T}_k'\}\} + \operatorname{trace}\{\boldsymbol{T}_k(\rho\boldsymbol{I}_d - \boldsymbol{\Pi})\boldsymbol{T}_k'\} + \rho\,\underbrace{\operatorname{trace}\{\boldsymbol{T}\boldsymbol{T}'\}}_{K},\end{aligned}$$

where $\boldsymbol{\Pi} \triangleq \boldsymbol{X}\boldsymbol{X}'$ is the $d \times d$ data covariance matrix and ρ is its spectral radius and $\boldsymbol{T}_k$ is the current transform estimate. This inequality can be the basis for a (possibly novel) **MM** approach.

Here is an alternative approach that seems simpler.

Suppose we want to learn $K < d$ filters. We can still estimate a $d \times d$ matrix $\boldsymbol{T}$, but ignore the last $d - K$ "dummy" rows of the matrix. To truly ignore those rows, we need to ignore the corresponding rows of $\boldsymbol{Z}$ that correspond to the products of the dummy rows of $\boldsymbol{T}[(K+1):d,:]$ with $\boldsymbol{X}$, *i.e.*, we want the first K rows of $\boldsymbol{Z}$ to be sparse, but we do not care about the remaining rows. Thus, we define the following (possibly novel) cost function

$$\Psi(\boldsymbol{T}, \boldsymbol{Z}) = \frac{1}{2} \|\boldsymbol{T}\boldsymbol{X} - \boldsymbol{Z}\|_{\mathrm{F}}^2 + \alpha \qquad (6.18)$$

(The idea here is somewhat similar to the HW problem where we apply sparsity regularization to the wavelet detail coefficients only.)

With this approach, the $\boldsymbol{Z}$ update applies hard thresholding to the first $K < d$ rows of $\boldsymbol{T}\boldsymbol{X}$ and leaves untouched the remaining rows: `Z[1:K,:]  .= hard.(T[1:K,:]*X)`

Then the $\boldsymbol{T}$ update is simply the standard orthogonal Procrustes solution in (6.17).

The potential advantage of the MM approach is that the matrix $\boldsymbol{Z}$ requires storing only a $K \times L$ matrix whereas the dummy rows approach appears to require storing a $d \times L$ matrix. If $d = 8^3$ for a 3D imaging problem and we are content learning, say, 32, filters, that is a 16-fold difference in storage that could be significant. The advantage of the dummy rows approach is that we can easily reuse the `stief1` code instead of writing a new MM algorithm.

Convergence properties of both approaches would require investigation, but most likely the proofs in [8] could be adapted.

BCM for non-square transform learning

Here is a summary of a **BCM** algorithm for transform learning with $K < d$.

- Apply the **proximal operator** (*e.g.*, soft or hard thresholding) to the first K rows of $\boldsymbol{T}\boldsymbol{X}$ to update $\boldsymbol{Z}$.
- Apply the **orthogonal Procrustes method** using SVD of the product $\boldsymbol{Z}\boldsymbol{X}'$ to update $\boldsymbol{T}$.
- Repeat until convergence, then perhaps keep just $\hat{\boldsymbol{T}}[1:K,:]$

Why perhaps? Because to use $\hat{\boldsymbol{T}}$ as a patch-based regularizer, we might want to use the same trick:

$$\hat{\boldsymbol{x}} = \operatorname*{arg\,min}_{\boldsymbol{x}\in\mathbb{F}^N} \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{y}\|_2^2 + \beta R(\boldsymbol{x}), \qquad R(\boldsymbol{x}) = \min_{\boldsymbol{Z}\in\mathbb{F}^{d\times L}} \sum_{l=1}^{L} \frac{1}{2}\left\|\hat{\boldsymbol{T}}\boldsymbol{R}_l\boldsymbol{x}-\boldsymbol{z}_l\right\|_2^2 + \alpha\phi(\boldsymbol{W}\boldsymbol{z}_l), \tag{6.19}$$

where $\boldsymbol{W} = \text{Diag}\{\boldsymbol{w}\},\ \boldsymbol{w} = \begin{bmatrix}\mathbf{1}_K \\ \mathbf{0}_{d-K}\end{bmatrix}.$

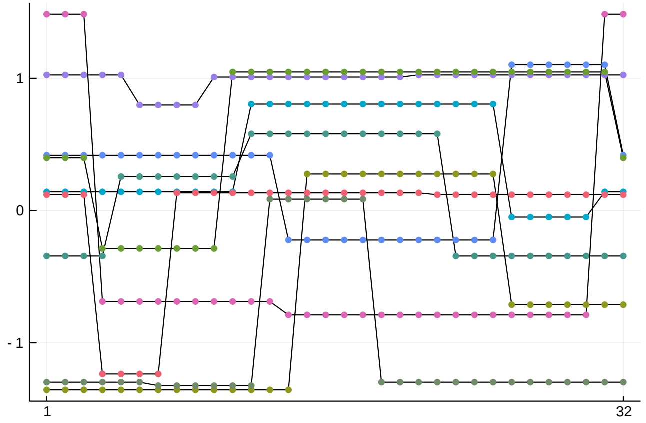
This way $\hat{\boldsymbol{T}}'\hat{\boldsymbol{T}} = \boldsymbol{I}$, simplifying the update, while we essentially ignore the last $d-K$ rows of $\boldsymbol{Z}$ by not enforcing sparsity there. To be explicit, for the above $\boldsymbol{W}$ we have:

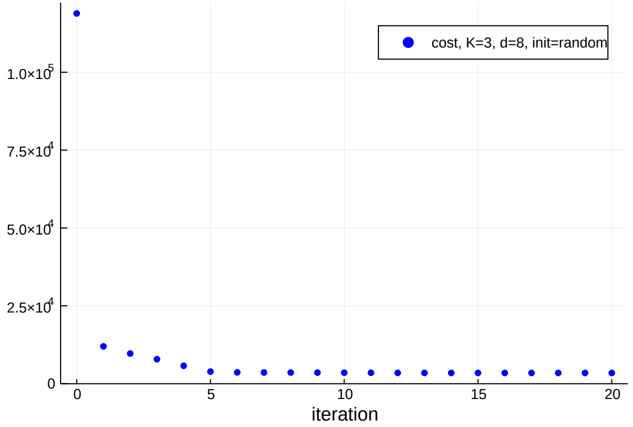
$$R(\boldsymbol{x}) = \min_{\boldsymbol{Z}\in\mathbb{F}^{d\times L}} \sum_{l=1}^{L} \frac{1}{2}\left\|\hat{\boldsymbol{T}}\boldsymbol{R}_l\boldsymbol{x}-\boldsymbol{z}_l\right\|_2^2 + \alpha\phi(\boldsymbol{W}\boldsymbol{z}_l) = \min_{\boldsymbol{Z}\in\mathbb{F}^{K\times L}} \sum_{l=1}^{L} \frac{1}{2}\left\|\hat{\boldsymbol{T}}[1:K,:]\boldsymbol{R}_l\boldsymbol{x}-\boldsymbol{z}_l\right\|_2^2 + \alpha\phi(\boldsymbol{z}_l).$$

This dummy-row trick may be novel. (If you see it in the literature please let me know.)

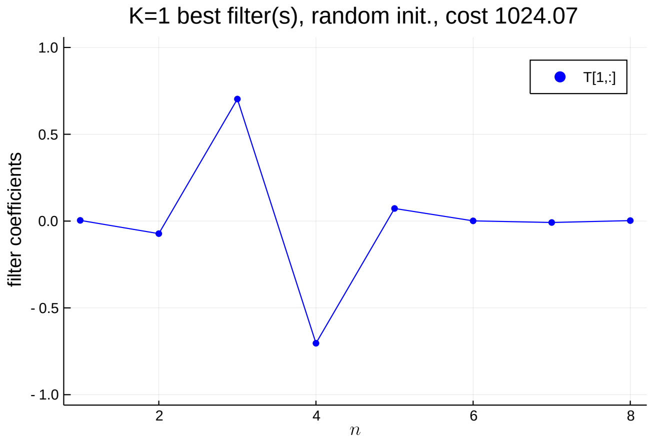
Example. Here we consider an ensemble of 2^{11} 1D piecewise-constant signals of length $N = 32$, several of which are illustrated in the left figure below.

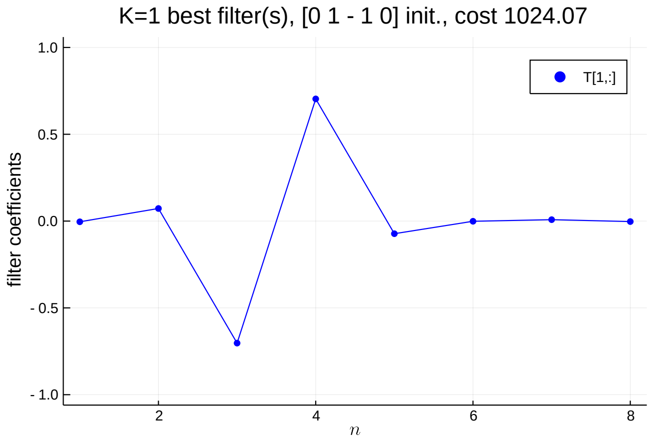
From these signals I extracted all $5 \cdot 10^4$ patches of size $d = 8 \times 1$, and then discarded all patches that are completely uniform because they seem to contain little useful information. There were about $L = 3 \cdot 10^4$ "interesting" patches for training. The right figure shows the cost function (6.18) decreasing with each BCM update, for the case $K = 1$.





Here are learned filters $\hat{\boldsymbol{T}}$ for $K = 1$ For the left figure $\boldsymbol{T}_0$ was randomly initialized, and for the right figure the first row of $\boldsymbol{T}_0$ was $[-1\ 1\ 0 \ldots 0]$. The consistency of the shape (up to a sign flip) is interesting.



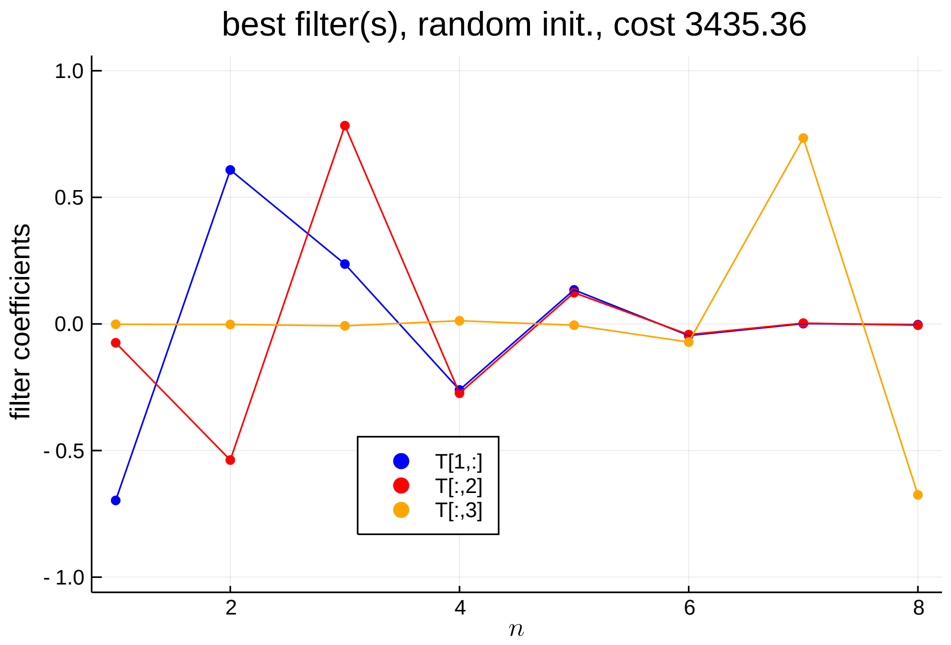


The filter values $\hat{\boldsymbol{T}}$ are [-0.0 -0.05 0.7 -0.71 0.07 0.0 -0.01 0.0]

I used $\alpha = 0.4$, for which about 5% of the $\boldsymbol{Z}[1, :]$ values were nonzero.

Because all the training signals are piecewise-constant (by design), it is unsurprising that the learned filter is close to a finite difference filter. But it is not exactly $[1\ -1]$, so it would be interesting to compare denoising using a regularizer based on this learned filter to that based on TV. Increasing L did not change the filter estimates.

Here is the case where we learn $K = 3$ filters:



Memory efficient implementation of transform learning (Read)

The BCM algorithm for **transform learning** on p. 6.32 is elegant in its simplicity, but as written it is memory inefficient because it must store both the $d \times L$ matrix $\boldsymbol{X}$ and the $K \times L$ matrix $\boldsymbol{Z}$, where L can be enormous. Furthermore, typically we form the columns of $\boldsymbol{X}$ from overlapping patches from training images, and the patch extraction process (with stride=1) increases the memory of an image with N pixels to a set of patches with dN elements, so by a factor of d. Careful implementation can avoid these memory issues. First we rewrite the two BCM steps:

$$\begin{aligned} \boldsymbol{z}_l^{(t+1)} &= \operatorname{prox}_{\alpha\phi}\left(\boldsymbol{T}^{(t)} \boldsymbol{x}_l\right), \quad l = 1, \ldots, L \\ \boldsymbol{T}^{(t+1)} &= \underset{\boldsymbol{T} \in \mathcal{T}}{\arg\min} \left\| \boldsymbol{T}\boldsymbol{X} - \boldsymbol{Z}^{(t+1)} \right\|_{\mathrm{F}}^2 = \boldsymbol{U}\boldsymbol{V}', \quad \boldsymbol{Z}^{(t+1)}\boldsymbol{X}' = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}'. \end{aligned}$$

The key is to implement the matrix product $\boldsymbol{Z}^{(t+1)}\boldsymbol{X}'$ with an **accumulator**:

$$\boldsymbol{Z}^{(t+1)}\boldsymbol{X}' = \sum_{l=1}^{L} \boldsymbol{z}_l^{(t+1)} \boldsymbol{x}_l' = \sum_{l=1}^{L} \operatorname{prox}_{\alpha\phi}\left(\boldsymbol{T}^{(t)}\boldsymbol{x}_l\right) \boldsymbol{x}_l'. \tag{6.20}$$

In this form, we never need to store all of the $\{\boldsymbol{z}_l\}$ coefficients. Furthermore, if we are extracting each patch $\boldsymbol{x}_l$ from a set of training images, then we never need to store the individual patches; we just loop over each training image, then loop over each patch in those images; we extract one patch, multiply it by $\boldsymbol{T}^{(t)}$, apply the proximal operator, make the outer product with the patch, and accumulate using `+=`.

One small drawback of using (6.20) is that without all of $\boldsymbol{X}$ and $\boldsymbol{Z}$ available, one cannot compute the cost function $\Psi(\boldsymbol{T}, \boldsymbol{Z})$ for any $\boldsymbol{T}$ and $\boldsymbol{Z}$. However, if $\boldsymbol{T}$ is unitary, then the key term in cost is

$$\begin{aligned} f(\boldsymbol{T}, \boldsymbol{Z}) &= -2\,\mathrm{real}\{\mathsf{trace}\{\boldsymbol{Z}(\boldsymbol{T}\boldsymbol{X})'\}\} + \mathsf{trace}\{\boldsymbol{Z}\boldsymbol{Z}'\} \\ &= -2\,\mathrm{real}\left\{\mathsf{trace}\left\{\sum_{l=1}^L \boldsymbol{z}_l(\boldsymbol{T}\boldsymbol{x}_l)'\right\}\right\} + \mathsf{trace}\left\{\sum_{l=1}^L \boldsymbol{z}_l\boldsymbol{z}_l'\right\} \\ &= -2\,\mathrm{real}\left\{\sum_{l=1}^L \langle \boldsymbol{z}_l,\, \boldsymbol{T}\boldsymbol{x}_l\rangle\right\} + \sum_{l=1}^L \|\boldsymbol{z}_l\|_2^2 = \sum_{l=1}^L \|\boldsymbol{z}_l - \boldsymbol{T}\boldsymbol{x}_l\|_2^2 + c. \end{aligned}$$

If needed, we can use this expression to evaluate $f(\boldsymbol{T}^{(t)}, \boldsymbol{Z}^{(t+1)})$ while computing (6.20).

For an extension to multi-layer transform learning see [12].

Dictionary learning via two-block BCD

Problem statement: Given training data $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \dots & \boldsymbol{x}_L \end{bmatrix} \in \mathbb{F}^{N \times L}$, typically patches extracted from images, find a **dictionary** $\boldsymbol{D} \in \mathbb{F}^{d \times K}$ such that $\boldsymbol{x}_l \approx \boldsymbol{D} \boldsymbol{z}_l$ where $\boldsymbol{z}_l \in \mathbb{F}^K$ is (typically) a **sparse** coefficient vector. Optimization formulation:

$$\hat{\boldsymbol{D}} = \mathop{\arg\min}_{\boldsymbol{D} \in \mathcal{D}} \min_{\boldsymbol{Z} \in \mathbb{F}^{K \times l}} \Psi(\boldsymbol{D}, \boldsymbol{Z}), \quad \Psi(\boldsymbol{D}, \boldsymbol{Z}) \triangleq$$

where $\|\boldsymbol{Z}\|_0 = \sum_{l=1}^L \|\boldsymbol{z}_l\|_0$ is the *aggregate* non-sparsity, *i.e.*, the total number of nonzero coefficients. This cost function Ψ is nonconvex due to the product $\boldsymbol{D}\boldsymbol{Z}$.
(It would be **biconvex** if we used $\|\text{vec}(\boldsymbol{Z})\|_1$.)

To avoid the scale ambiguity, we focus on this typical choice for the set admissible dictionaries:

$$\mathcal{D} = \left\{ \boldsymbol{D} \in \mathbb{F}^{d \times K} \ : \ \|\boldsymbol{d}_k\|_2 = 1, \ k = 1, \dots, K \right\}.$$

To minimize Ψ, one could attempt two-block **BCD**, alternating between updating $\boldsymbol{D}$ and $\boldsymbol{Z}$.

- The $\boldsymbol{D}$ update is a (nonconvex) constrained problem and one could apply **gradient projection** for it (see HW). If we replace $\|\boldsymbol{d}_k\|_2 = 1$ with $\|\boldsymbol{d}_k\|_2 \leq 1$ then the $\boldsymbol{D}$ update would be a convex constrained problem.
- The $\boldsymbol{Z}$ update is a set of L separate sparse coding problems, If we used the 1-norm, then it would be convex and one could apply **POGM** in parallel to each column of $\boldsymbol{Z}$. This is a simple use of parallel

processing. For the 0-norm, there is no convergence guarantee of momentum methods like POGM (to my knowledge), so it seems safer to use **PGM**, corresponding to iterative hard thresholding:

$$\boldsymbol{z}_l^{(t+1)} = \mathsf{hard}_{.}\left(\boldsymbol{z}_l^{(t)} - \frac{1}{\|\boldsymbol{D}\|_2^2}\boldsymbol{D}'\left(\boldsymbol{D}\boldsymbol{z}_l - \boldsymbol{x}_l\right), \frac{\beta}{\|\boldsymbol{D}\|_2^2}\right), \; l = 1, \ldots, L.$$

Both of these updates require 1 or more inner iterations, so overall it is a **BCD** approach.

Dictionary learning via multi-block BCM (SOUP-DIL)

Instead of updating the entire dictionary $\boldsymbol{D}$ and the entire coefficient vector $\boldsymbol{Z}$ simultaneously, we can instead think of the multi-block cost function $\Psi(\boldsymbol{d}_1, \ldots, \boldsymbol{d}_K, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_K)$ where $\boldsymbol{C} = \boldsymbol{Z}' = [\boldsymbol{c}_1 \; \ldots \; \boldsymbol{c}_K] \in \mathbb{F}^{L \times K}$, and write the product in the following sum-of-outer-products (**SOUP**) form:

$$\begin{bmatrix}\boldsymbol{D}\boldsymbol{z}_1 & \ldots & \boldsymbol{D}\boldsymbol{z}_L\end{bmatrix} = \boldsymbol{D}\boldsymbol{Z} = \boldsymbol{D}\boldsymbol{C}' =$$

With this formulation, a useful multi-block **BCM** approach is to update one atom $\boldsymbol{d}_k$ at a time, and then update the corresponding coefficient vector $\boldsymbol{c}_k$, and loop sequentially through $k = 1, \ldots, K$.

Updating the variables in matched pairs, *e.g.*, in the order: $\boldsymbol{d}_1, \boldsymbol{c}_1, \boldsymbol{d}_2, \boldsymbol{c}_2, \ldots, \boldsymbol{d}_K, \boldsymbol{c}_K$, seems to accelerate convergence. The next pages describe those updates.

Dictionary atom update

To update $\boldsymbol{d}_k$, define the residual matrix

$$\boldsymbol{R} = \boldsymbol{X} - \sum_{j \neq k} \boldsymbol{d}_j \boldsymbol{c}_j' \tag{6.21}$$

then the update is

$$\boldsymbol{d}_k^{(t+1)} = \underset{\boldsymbol{d}_k \in \mathbb{F}^d \,:\, \|\boldsymbol{d}\|_2 = 1}{\arg\min} \frac{1}{2} \|\boldsymbol{R} - \boldsymbol{d}_k \boldsymbol{c}_k'\|_{\mathrm{F}}^2$$

where

Sparse coefficient update

The update for $\boldsymbol{c}_k$ is group work:

$$\boldsymbol{c}_k^{(t+1)} = \underset{\boldsymbol{c}_k \in \mathbb{F}^L}{\arg\min} \frac{1}{2} \|\boldsymbol{R} - \boldsymbol{d}_k \boldsymbol{c}_k'\|_{\mathrm{F}}^2 + \beta \|\boldsymbol{c}_k\|_0 .$$

Because $\|\boldsymbol{d}_k\|_2 = 1$:

$$\|\boldsymbol{R} - \boldsymbol{d}_k \boldsymbol{c}_k'\|_{\mathrm{F}}^2 = \mathsf{trace}\{(\boldsymbol{R} - \boldsymbol{d}_k \boldsymbol{c}_k')(\boldsymbol{R} - \boldsymbol{d}_k \boldsymbol{c}_k')'\} = \|\boldsymbol{c}_k\|_2^2 - 2\,\mathrm{real}\{\boldsymbol{d}_k' \boldsymbol{R} \boldsymbol{c}_k\} + c_1 = \|\boldsymbol{c}_k - \boldsymbol{R}' \boldsymbol{d}_k\|_2^2 + c_2$$

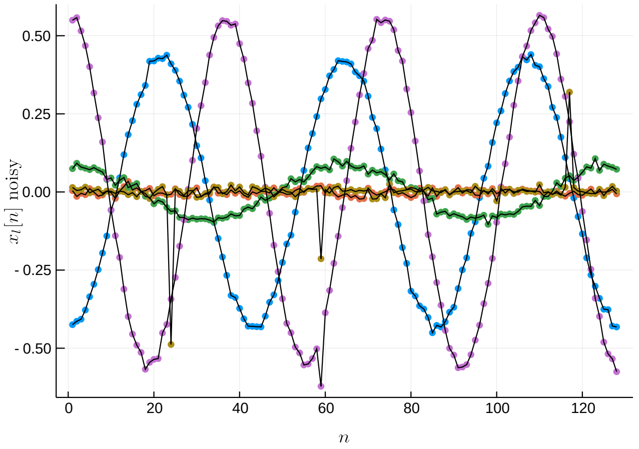
$$\boldsymbol{c}_k^{(t+1)} = \mathsf{hard}_{.}(\boldsymbol{R}' \boldsymbol{d}_k, \beta).$$

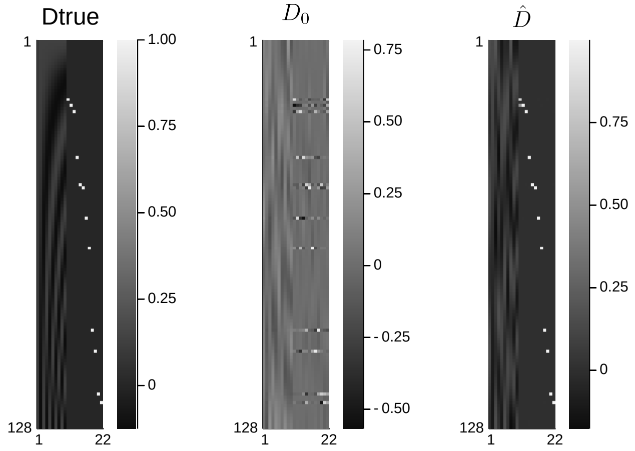
The main drawback of this SOUP approach is that it is less parallelizable than the two-block BCD approach.

This SOUP alternating approach is:

A: BCD B: BCM C: Neither $\boxed{??}$

Example. An in-class group activity will apply this SOUP **dictionary learning** method to signals consisting of a smooth part and some added spikes. The left plot shows some example signals. The right plot shows the true dictionary, the initial dictionary estimate $\boldsymbol{D}_0$ (from PCA) and the estimated dictionary $\hat{\boldsymbol{D}}$.





Joint update of atom and coefficients (Read)

Previously we focused on alternating between updating one atom $\boldsymbol{d}_k$ and the corresponding coefficient vector $\boldsymbol{c}_k$, ala SOUP. Can we update them concurrently? Starting with the residual (6.21) and dropping subscript k for simplicity, we want to solve (or descend)

$$\underset{\boldsymbol{c}\in\mathbb{F}^L,\ \boldsymbol{d}\in\mathbb{F}^d \,:\, \|\boldsymbol{d}\|_2\leq 1}{\arg\min} f(\boldsymbol{d},\boldsymbol{c}), \quad f(\boldsymbol{d},\boldsymbol{c}) \triangleq \frac{1}{2}\|\boldsymbol{d}\boldsymbol{c}' - \boldsymbol{R}\|_{\mathrm{F}}^2 \overset{\mathrm{c}}{=} \frac{1}{2}\|\boldsymbol{d}\|_2^2\,\|\boldsymbol{c}\|_2^2 - \boldsymbol{d}'\boldsymbol{R}\boldsymbol{c}.$$

The Hessian of this cost function is

$$\nabla^2 f = \begin{bmatrix} \|\boldsymbol{c}\|^2\,\boldsymbol{I}_d & 2\boldsymbol{d}\boldsymbol{c}' - \boldsymbol{R} \\ 2\boldsymbol{c}\boldsymbol{d}' - \boldsymbol{R}' & \|\boldsymbol{d}\|^2\,\boldsymbol{I}_L \end{bmatrix}. \tag{6.22}$$

Consider the scalar case $(d = L = 1)$ and suppose $R = 0$. Then $\nabla^2 f = \begin{bmatrix} c^2 & 2dc \\ 2cd & d^2 \end{bmatrix}$ which has eigenvalues $\lambda = (d^2 + c^2 + \sqrt{(d^2 - c^2)^2 + 16d^2c^2})/2$, so $\rho(\nabla^2 f)$ is unbounded as c increases. Thus the claim in [13] that ∇f is jointly global Lipschitz is incorrect in general.

However, in practical problems it seems reasonable to assume (or impose) that the sparse coefficients are bounded by some finite maximum value: $\|\boldsymbol{c}\|_\infty \leq \bar{z}$, *cf.* [13, 14]. In the scalar case this leads to a simple upper bound on the spectral radius of the Hessian.

Challenge: using $\|\boldsymbol{d}\|_2 \leq 1$ and $\|\boldsymbol{c}\|_\infty \leq \bar{z}$, find an upper bound on the spectral radius of the Hessian in (6.22). It is fine to assume $1 \leq \bar{z}$. Perhaps using $\|\!|\cdot|\!\|_1$ could simplify.

6.2 Machine learning applications

Low-rank approximation for large-scale problems

Given $\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{\varepsilon} \in \mathbb{F}^{M \times N}$ where $\boldsymbol{\varepsilon}$ denotes a $M \times N$ noise matrix and we believe $\operatorname{rank}(\boldsymbol{X}) \leq K$.

EECS 551 used truncated/thresholded SVD methods requiring $O(MN^2)$ operations that do not scale to large problems where both M and N are large, even if the rank K is very small. We can overcome this problem using **alternating minimization** (two-block BCD/BCM) methods.

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

$$\hat{\boldsymbol{X}} = \underbrace{\boldsymbol{U}}_{M \times K} \underbrace{\boldsymbol{V}}_{K \times N},$$

where now $\boldsymbol{U}$ and $\boldsymbol{V}$ should be simply interpreted as 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:

$$\begin{aligned}
\hat{\boldsymbol{X}} &= \hat{\boldsymbol{U}}\hat{\boldsymbol{V}} \\
(\hat{\boldsymbol{U}}, \hat{\boldsymbol{V}}) &= \underset{\boldsymbol{U} \in \mathbb{F}^{M \times K},\ \boldsymbol{V} \in \mathbb{F}^{K \times N}}{\arg\min} \Psi(\boldsymbol{U}, \boldsymbol{V}) \\
\Psi(\boldsymbol{U}, \boldsymbol{V}) &\triangleq \frac{1}{2} \|\boldsymbol{Y} - \boldsymbol{U}\boldsymbol{V}\|_{\mathrm{F}}^2 + \beta_1 R_1(\boldsymbol{U}) + \beta_2 R_2(\boldsymbol{V}),
\end{aligned}$$

where one must select appropriate regularizers for $\boldsymbol{U}$ and $\boldsymbol{V}$. The data term here is **biconvex**.

Ambiguities

Scale ambiguity:

Factorization ambiguity for invertible $\boldsymbol{P}$:

- These ambiguities might not matter if we just want the final $\hat{\boldsymbol{X}}$.
- They might matter for optimization where if the iterates do not converge.

Constraining $\boldsymbol{U}$ to have orthonormal columns resolves the above scale (?) and factorization (?) ambiguity.

A: F,F B: F,T C: T,F D: T,T $\boxed{??}$

Two-block BCM with unitary constraint

One way to resolve the ambiguity is to constrain $\boldsymbol{U}$ to have orthonormal columns:

$$R_1(\boldsymbol{U}) = \chi_{\mathcal{V}_K(\mathbb{F}^M)}(\boldsymbol{U}),$$

where the **Stiefel manifold** is $\mathcal{V}_K(\mathbb{F}^M) = \left\{\boldsymbol{Q} \in \mathbb{F}^{M \times K} \;:\; \boldsymbol{Q}'\boldsymbol{Q} = \boldsymbol{I}_K\right\}$. The optimization problem becomes:

$$\hat{\boldsymbol{X}} = \hat{\boldsymbol{U}}\hat{\boldsymbol{V}}, \quad (\hat{\boldsymbol{U}}, \hat{\boldsymbol{V}}) = \underset{\boldsymbol{U}\in\mathbb{F}^{M\times K},\, \boldsymbol{V}\in\mathbb{F}^{K\times N}}{\arg\min} \Psi(\boldsymbol{U},\boldsymbol{V}), \quad \Psi(\boldsymbol{U},\boldsymbol{V}) \triangleq \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{U}\boldsymbol{V}\|_{\mathrm{F}}^2 + \chi_{\mathcal{V}_K(\mathbb{F}^M)}(\boldsymbol{U}).$$

A two-block **BCM** for this problem is:

$$\boldsymbol{U}_{t+1} = \underset{\boldsymbol{U}\,:\,\boldsymbol{U}'\boldsymbol{U}=\boldsymbol{I}_K}{\arg\min} \|\boldsymbol{Y} - \boldsymbol{U}\boldsymbol{V}_t\|_{\mathrm{F}}^2 = \tilde{\boldsymbol{U}}\tilde{\boldsymbol{V}}', \qquad \underbrace{\boldsymbol{Y}}_{M\times N}\underbrace{\boldsymbol{V}_t'}_{N\times K} = \underbrace{\tilde{\boldsymbol{U}}_K}_{M\times K}\,\boldsymbol{\Sigma}_K\,\underbrace{\tilde{\boldsymbol{V}}'}_{K\times K}. \qquad O(MK^2)$$

$$\boldsymbol{V}_{t+1} = \underset{\boldsymbol{V}\in\mathbb{F}^{K\times N}}{\arg\min} \|\boldsymbol{Y} - \boldsymbol{U}_{t+1}\boldsymbol{V}\|_{\mathrm{F}}^2 \Longrightarrow \boldsymbol{V}_{t+1}[:,n] = \boldsymbol{U}_{t+1}'\boldsymbol{Y}[:,n] \Longrightarrow \boldsymbol{V}_{t+1} = \boldsymbol{U}_{t+1}'\boldsymbol{Y}, \qquad O(MNK)$$

by solving separate LS problems for each column of $\boldsymbol{V}$, because

$$\|\boldsymbol{Y} - \boldsymbol{U}_{t+1}\boldsymbol{V}\|_{\mathrm{F}}^2 = \sum_{n=1}^{N} \|\boldsymbol{Y}[:,n] - \boldsymbol{U}_{t+1}\boldsymbol{V}[:,n]\|_2^2.$$

- Convergence?
  Cost function decreases every iteration, bounded below, $\Longrightarrow$ cost converges.
  Convergence of iterates is an active research area.
  Yes, Under some RIP conditions [15]
- Other regularizers or constraints?
  Sparsity (even more structure than low-rank!) $R_2(\boldsymbol{V}) = \|\mathrm{vec}(\boldsymbol{V})\|_1$

group work: $\boldsymbol{V}$ update using **PGM** (because we have so many tools now)

Fused LASSO / generalized LASSO

(Read)

The **fused LASSO** is a generalization of the LASSO problem used in machine learning for regression problems where some features are known to be related [17]. The cost function has the form

$$\Psi(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta \|\boldsymbol{x}\|_1 + \gamma \|\boldsymbol{T}\boldsymbol{x}\|_1 ,$$

for some special matrix $\boldsymbol{T}$ that involves finite differences between correlated features.

This is a challenging optimization problem.

A "dual path algorithm" [18] [19] [20] is available in R:
https://rdrr.io/cran/genlasso/man/fusedlasso.html

However, the documentation states: "Hence it is not advisable to run fusedlasso2d on image denoising problems of large scale, as the dual solution path is computationally infeasible. It should be noted that a faster algorithm for the 2d fused LASSO solution path (when the predictor matrix is the identity), which begins at the dense end of the path, is available in the `flsa` package."

A related cost function called **generalized LASSO** simply uses $\beta = 0$ above.
https://rdrr.io/cran/genlasso/man/genlasso.html

These challenges motivate the AL/ADMM methods described in Ch. 7.

Alternating minimization for 0-norm in biconvex form (Read)

Consider the very challenging sparsity-constrained optimization problem

$$\arg\min_{\boldsymbol{x}\in\mathbb{F}^N} f(\boldsymbol{x}) \text{ s.t. } \|\boldsymbol{T}\boldsymbol{x}\|_0 \leq K. \tag{6.23}$$

Nothing we have covered so far solves this type of problem for a general matrix $\boldsymbol{T}$.
If f has a Lipschitz gradient, then when $\boldsymbol{T} = \boldsymbol{I}$ we could make a proximal gradient method for (6.23), which would involve a form of **iterative hard thresholding**.

For $\boldsymbol{z} \in \mathbb{R}^N$, we can write the 0-norm as the following (**convex**!) optimization problem solution [21]:

$$\|\boldsymbol{z}\|_0 = \min_{-\mathbf{1}_N \leq \boldsymbol{u} \leq \mathbf{1}_N} \|\boldsymbol{u}\|_1 \text{ s.t. } \|\boldsymbol{z}\|_1 = \langle \boldsymbol{u}, \boldsymbol{z} \rangle .$$

The unique solution is $\boldsymbol{u}_* = \boldsymbol{u}_*(\boldsymbol{z}) = \operatorname{sign}.(\boldsymbol{z})$, for which $\|\boldsymbol{u}_*\|_1 = \|\boldsymbol{z}\|_0$.

More generally (thanks to Katherine Banas for helping me see this), for $\boldsymbol{z} \in \mathbb{F}^N$, we can write the 0-norm as the following (**convex**!) optimization problem solution:

$$\|\boldsymbol{z}\|_0 = \min_{\boldsymbol{u}\in\mathbb{F}^N : \|\boldsymbol{u}\|_\infty \leq 1} \|\boldsymbol{u}\|_1 \text{ s.t. } \|\boldsymbol{z}\|_1 = \langle \boldsymbol{u}, \boldsymbol{z} \rangle . \tag{6.24}$$

The unique solution is $\boldsymbol{u}_* = \boldsymbol{u}_*(\boldsymbol{z}) = \operatorname{sign}.(\boldsymbol{z})$, for which $\|\boldsymbol{u}_*\|_1 = \|\boldsymbol{z}\|_0$.
The form (6.24) is general enough to cover both $\mathbb{R}$ and $\mathbb{C}$ cases.

Thus we can rewrite the original problem (6.23) as:

$$\arg\min_{\boldsymbol{x}\in\mathbb{F}^N} \min_{\boldsymbol{u}\in\mathbb{F}^N} f(\boldsymbol{x}) + \chi_{\mathcal{C}_K}(\boldsymbol{u}) \text{ s.t. } \|\boldsymbol{T}\boldsymbol{x}\|_1 = \langle \boldsymbol{u},\, \boldsymbol{T}\boldsymbol{x}\rangle, \qquad \mathcal{C}_K = \left\{\boldsymbol{u}\in\mathbb{F}^N \;:\; \|\boldsymbol{u}\|_\infty \le 1,\ \|\boldsymbol{u}\|_1 \le K\right\}. \tag{6.25}$$

One can verify that the constraint set $\|\boldsymbol{z}\|_1 = \langle \boldsymbol{u},\, \boldsymbol{z}\rangle$ is convex, so (6.25) is **biconvex** [22].

When $\|\boldsymbol{u}\|_\infty \le 1$, we have $\langle \boldsymbol{u},\, \boldsymbol{z}\rangle \le \sum_{n=1}^N |z_n| = \|\boldsymbol{z}\|_1$.
Thus $\{\boldsymbol{z} \;:\; \|\boldsymbol{z}\|_1 = \langle \boldsymbol{u},\, \boldsymbol{z}\rangle\} = \{\boldsymbol{z} \;:\; \|\boldsymbol{z}\|_1 \le \langle \boldsymbol{u},\, \boldsymbol{z}\rangle\}$.
Now if $\boldsymbol{z}$ and $\boldsymbol{w}$ are both in this set then $\alpha\boldsymbol{z} + \beta\boldsymbol{w}$ is also in this set for $0 \le \alpha \le 1$ and $\beta = 1-\alpha$,
because $\|\alpha\boldsymbol{z} + \beta\boldsymbol{w}\|_1 \le \alpha\|\boldsymbol{z}\|_1 + \beta\|\boldsymbol{w}\|_1 \le \alpha\langle \boldsymbol{u},\, \boldsymbol{z}\rangle + \beta\langle \boldsymbol{u},\, \boldsymbol{w}\rangle = \langle \alpha\boldsymbol{u} + \beta\boldsymbol{w},\, \boldsymbol{z}\rangle$.

Still, that constraint seems challenging, so one can replace the constraint in (6.25) with a penalty on the gap:

$$\arg\min_{\boldsymbol{x}\in\mathbb{F}^N} \min_{\boldsymbol{u}\in\mathbb{F}^N} f(\boldsymbol{x}) + \chi_{\mathcal{C}_K}(\boldsymbol{u}) + \mu\left(\|\boldsymbol{T}\boldsymbol{x}\|_1 - \langle \boldsymbol{u},\, \boldsymbol{T}\boldsymbol{x}\rangle\right). \tag{6.26}$$

This is a **biconvex** problem. One can increase μ as the iterations proceed to (asymptotically) enforce the constraint. The obvious approach here is **alternating minimization** and there are convergence results in [21].

Interpretation (Read)

For any finite $\mu > 0$, we can rewrite the penalized formulation (6.26) as:

$$\arg\min_{\boldsymbol{x}} f(\boldsymbol{x}) + \mu R(\boldsymbol{T}\boldsymbol{x}), \quad R(\boldsymbol{z}) \triangleq \min_{\boldsymbol{u} \in \mathcal{C}_K} \left(\|\boldsymbol{z}\|_1 - \langle \boldsymbol{u},\, \boldsymbol{z} \rangle \right).$$

We can simplify the expression for R as follows:

$$R(\boldsymbol{z}) = \|\boldsymbol{z}\|_1 - g(\boldsymbol{z}), \quad g(\boldsymbol{z}) \triangleq \max_{\boldsymbol{u} \in \mathcal{C}_K} \langle \boldsymbol{u},\, \boldsymbol{z} \rangle = \texttt{sum(sort(abs(z))[1:K])}.$$

(This is related to the "order weighted L1" (**OWL**) regularizer [23].) Thus

$$R(\boldsymbol{z}) = \mu\, \texttt{sum(sort(abs(z))[K+1:end])}. \tag{6.27}$$

So this regularizer penalizes the $N - K$ smallest (in magnitude) values of $\boldsymbol{z} = \boldsymbol{T}\boldsymbol{x}$, so as μ increases those values will be thresholded to zero, which is what (6.23) requires! To help see this, note that

$$\|\boldsymbol{z}\|_0 \le K \iff \texttt{norm(sort(abs(z))[K+1:end],1) == 0}$$

This analysis is helpful for insight, but the sorting operation in (6.27) is very nonconvex and hard for optimization. In contrast, the biconvex set up (6.26) seems much simpler for optimization.

6.3 Convergence properties

BCD converges in 1 iteration in the rate cases where the cost function is block separable (decoupled):

$$\Psi(\boldsymbol{x}_1, \dots, \boldsymbol{x}_B) = \sum_{b=1}^{B} \Psi_b(\boldsymbol{x}_b) \,.$$

Convergence results (for limit points) under weak assumptions are given in [24, p. 268]. See also [25–34].

Convergence of the coordinate descent method for strictly convex, twice-differentiable cost functions is analyzed in detail in [35], including consideration of **box constraints**. Powell demonstrates that uniqueness of the "arg min" step is important to ensure convergence [36].

Convergence rate analysis, including constrained cases, is given in [31, 37].

For "pure" **coordinate descent** (**CD**), where we update one parameter at a time in sequence, if the cost function is twice differentiable then there is an asymptotic linear convergence rate. Specifically, when $\boldsymbol{x}_n$ is near the minimizer $\hat{\boldsymbol{x}}$:

$$\|\boldsymbol{x}_{n+1} - \hat{\boldsymbol{x}}\|_{\boldsymbol{H}^{1/2}} \leq \rho(\boldsymbol{M}) \, \|\boldsymbol{x}_n - \hat{\boldsymbol{x}}\|_{\boldsymbol{H}^{1/2}} \,, \quad \boldsymbol{M} = \boldsymbol{I} - [\boldsymbol{D} + \boldsymbol{L}]^{-1} \boldsymbol{H},$$

where $\boldsymbol{H} = \nabla^2 \Psi(\hat{\boldsymbol{x}}) = \boldsymbol{L} + \boldsymbol{D} + \boldsymbol{L}'$ where $\boldsymbol{D}$ is diagonal and $\boldsymbol{L}$ is lower triangular. The analysis is similar to that of the **Gauss-Seidel method** for solving a linear system of equations.

Analysis of the convergence rate of "pure" CD is a special case of the analysis of **SAGE** in [38].

For a BPGM version with **momentum** see [34, 39].

For analysis of inexact proximal BCD see [40].

Generalizations for non-smooth and non-convex functions are in [29, 40, 41]. This is an evolving area because of growing interest in BCD methods.

In particular, there is considerable recent focus on **biconvex** problems like those involving **matrix factorization** $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{V}$. One can show that that some such problems have no spurious local minima [42–44].

Global convergence guarantees for dictionary learning appear in [45], despite many saddle points, for random initialization.

For randomized block selection, see [46].

6.4 Summary

Alternating minimization methods have a multitude of applications. They also provide a nice context for course review because they use many previous methods (gradient, MM, proximal) as intermediate steps.

Limitations of BCD methods are difficulty with **parallelism** (if one uses too many blocks with too small sizes) and getting stuck at non-stationary points for non-smooth cost functions.

Bibliography

[1] J. Besag. "On the statistical analysis of dirty pictures". In: *J. Royal Stat. Soc. Ser. B* 48.3 (1986), 259–302 (cit. on p. 6.3).

[2] J-F. Cai, R. H. Chan, and Z. Shen. "A framelet-based image inpainting algorithm". In: *Applied and Computational Harmonic Analysis* 24.2 (Mar. 2008), 131–49 (cit. on p. 6.11).

[3] J-F. Cai, R. Chan, L. Shen, and Z. Shen. "Restoration of chopped and nodded images by framelets". In: *SIAM J. Sci. Comp.* 30.3 (2008), 1205–27 (cit. on p. 6.11).

[4] J-F. Cai, R. H. Chan, L. Shen, and Z. Shen. "Convergence analysis of tight framelet approach for missing data recovery". In: *Applied and Computational Harmonic Analysis* 31.1 (Oct. 2009), 87–113 (cit. on p. 6.11).

[5] J-F. Cai and Z. Shen. "Framelet based deconvolution". In: *J. Comp. Math.* 28.3 (May 2010), 289–308 (cit. on p. 6.11).

[6] Y. Liu, J-F. Cai, Z. Zhan, D. Guo, J. Ye, Z. Chen, and X. Qu. "Balanced sparse model for tight frames in compressed sensing magnetic resonance imaging". In: *PLoS One* 10.4 (2015), 1–19 (cit. on pp. 6.11, 6.12).

[7] D. Gilton, G. Ongie, and R. Willett. "Learned patch-based regularization for inverse problems in imaging". In: *Proc. Intl. Wkshp. Comp. Adv. Multi-Sensor Adapt. Proc.* 2019, 211–5 (cit. on p. 6.28).

[8] 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 pp. 6.29, 6.32, 6.34).

[9] B. Wen, S. Ravishankar, L. Pfister, and Y. Bresler. “Transform learning for magnetic resonance image reconstruction: from model-based learning to building neural networks”. In: *IEEE Sig. Proc. Mag.* 37.1 (Jan. 2020), 41–53 (cit. on p. 6.29).

[10] Z. Li, S. Xie, W. Chen, and Z. Yang. “Overcomplete transform learning with the log regularizer”. In: *IEEE Access* 6 (2018), 65239–49 (cit. on p. 6.30).

[11] L. Pfister and Y. Bresler. “Learning filter bank sparsifying transforms”. In: *IEEE Trans. Sig. Proc.* 67.2 (Jan. 2019), 504–19 (cit. on p. 6.30).

[12] S. Ravishankar and B. Wohlberg. “Learning multi-layer transform models”. In: *Allerton Conf. on Comm., Control, and Computing*. 2018 (cit. on p. 6.40).

[13] G-J. Peng. “Joint and direct optimization for dictionary learning in convolutional sparse representation”. In: *IEEE Trans. Neural Net. Learn. Sys.* (2019) (cit. on p. 6.46).

[14] G-J. Peng. “Adaptive ADMM for dictionary learning in convolutional sparse representation”. In: *IEEE Trans. Im. Proc.* 28.7 (July 2019), 3408–422 (cit. on p. 6.46).

[15] P. Jain, P. Netrapalli, and S. Sanghavi. “Low-rank matrix completion using alternating minimization”. In: *ACM Symp. Theory Comp.* 2013, 665–74 (cit. on p. 6.50).

[17] R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight. “Sparsity and smoothness via the fused LASSO”. In: *J. Royal Stat. Soc. Ser. B* 67.1 (Feb. 2005), 91–108 (cit. on p. 6.51).

[18] H. Hoefling. *A path algorithm for the Fused Lasso signal approximator*. 2009 (cit. on p. 6.51).

[19] R. J. Tibshirani and J. Taylor. “The solution path of the generalized LASSO”. In: *Ann. Stat.* 39.3 (June 2011), 1335–71 (cit. on p. 6.51).

[20] T. B. Arnold and R. J. Tibshirani. “Efficient implementations of the generalized LASSO dual path algorithm”. In: *J. Computational and Graphical Stat.* 25.1 (2016), 1–27 (cit. on p. 6.51).

[21] G. Yuan and B. Ghanem. *Sparsity constrained minimization via mathematical programming with equilibrium constraints*. 2018 (cit. on pp. 6.52, 6.53).

[22] A. Bechensteen, L. Blanc-Feraud, and G. Aubert. “New methods for l2-l0 minimization and their applications to 2D single-molecule localization microscopy”. In: *Proc. IEEE Intl. Symp. Biomed. Imag.* 2019, 1377–81 (cit. on p. 6.53).

[23] M. A. T. Figueiredo and R. D. Nowak. “Ordered weighted l1 regularized regression with strongly correlated covariates: Theoretical aspects”. In: *aistats*. 2016, 930–8 (cit. on p. 6.54).

[24] D. P. Bertsekas. *Nonlinear programming*. 2nd ed. Belmont: Athena Scientific, 1999 (cit. on p. 6.55).

[25] P. Tseng. “Convergence of a block coordinate descent methods for nondifferentiable minimization”. In: *J. Optim. Theory Appl.* 109 (2001), 475–94 (cit. on p. 6.55).

[26] Y. Nesterov. “Efficiency of coordinate descent methods on huge-scale optimization problems”. In: *SIAM J. Optim.* 22.2 (2012), 341–62 (cit. on p. 6.55).

[27] M. W. Jacobson and J. A. Fessler. “An expanded theoretical treatment of iteration-dependent majorize-minimize algorithms”. In: *IEEE Trans. Im. Proc.* 16.10 (Oct. 2007), 2411–22 (cit. on p. 6.55).

[28] A. Beck and L. Tetruashvili. “On the convergence of block coordinate descent type methods”. In: *SIAM J. Optim.* 23.4 (2013), 2037–60 (cit. on p. 6.55).

[29] Y. Xu and W. Yin. “A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion”. In: *SIAM J. Imaging Sci.* 6.3 (2013), 1758–89 (cit. on pp. 6.55, 6.56).

[30] J. Bolte, S. Sabach, and M. Teboulle. “Proximal alternating linearized minimization for nonconvex and nonsmooth problems”. In: *Mathematical Programming* 146.1 (Aug. 2014), 459–94 (cit. on p. 6.55).

[31] S. Yun. “On the iteration complexity of cyclic coordinate gradient descent methods”. In: *SIAM J. Optim.* 24.3 (2014), 1567–80 (cit. on p. 6.55).

[32] K. Khare and B. Rajaratnam. *Convergence of cyclic coordinatewise l1 minimization*. 2015 (cit. on p. 6.55).

[33] Z. Shi and R. Liu. *A better convergence analysis of the block coordinate descent method for large scale machine learning*. 2016 (cit. on p. 6.55).

[34] Y. Xu and W. Yin. “A globally convergent algorithm for nonconvex optimization based on block coordinate update”. In: *J. of Scientific Computing* 72.2 (Aug. 2017), 1–35 (cit. on pp. 6.55, 6.56).

[35] Z. Q. Luo and P. Tseng. “On the convergence of the coordinate descent method for convex differentiable minimization”. In: *J. Optim. Theory Appl.* 72.1 (Jan. 1992), 7–35 (cit. on p. 6.55).

[36] M. J. D. Powell. “On search directions for minimization algorithms”. In: *Mathematical Programming* 4.1 (1973), 193–201 (cit. on p. 6.55).

[37] Z-Q. Luo and P. Tseng. “On the convergence rate of dual ascent methods for linearly constrained convex minimization”. In: *Math. Oper. Res.* 18.4 (Nov. 1993), 846–67 (cit. on p. 6.55).

[38] J. A. Fessler and A. O. Hero. “Space-alternating generalized expectation-maximization algorithm”. In: *IEEE Trans. Sig. Proc.* 42.10 (Oct. 1994), 2664–77 (cit. on p. 6.56).

[39] 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 p. 6.56).

[40] E. Chouzenoux, J-C. Pesquet, and A. Repetti. “A block coordinate variable metric forward-backward algorithm”. In: *J. of Global Optimization* 66.3 (Nov. 2016), 457–85 (cit. on p. 6.56).

[41] M. Razaviyayn, M. Hong, and Z. Luo. "A unified convergence analysis of block successive minimization methods for nonsmooth optimization". In: *SIAM J. Optim.* 23.2 (2013), 1126–53 (cit. on p. 6.56).

[42] M. Hardt. "Understanding alternating minimization for matrix completion". In: *Foundations of Computer Science (FOCS)*. 2014, 651–60 (cit. on p. 6.56).

[43] R. Ge, J. D. Lee, and T. Ma. "Matrix completion has no spurious local minimum". In: *Neural Info. Proc. Sys.* 2016, 2973–81 (cit. on p. 6.56).

[44] 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. 6.56).

[45] D. Gilboa, S. Buchanan, and J. Wright. *Efficient dictionary learning with gradient descent*. 2018 (cit. on p. 6.56).

[46] J. Diakonikolas and L. Orecchia. "Alternating randomized block coordinate descent". In: *Proc. Intl. Conf. Mach. Learn*. Vol. 80. 2018, 1224–32 (cit. on p. 6.56).