Homework #4, EECS 598-006, W20. Due **Thu. Feb. 06**, by 4:00PM

1. [3] **Cost functions for sparsity models**

Consider the inverse problem measurement model $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\varepsilon}$ where the latent vector $\boldsymbol{x} \in \mathbb{F}^N$ is thought to be the sum of two signal components, a foreground signal $\boldsymbol{f} \in \mathbb{F}^N$ and a background signal $\boldsymbol{b} \in \mathbb{F}^N$. We expect $\boldsymbol{f}$ to be well represented by a sparse linear combination of atoms from a $N \times K$ **dictionary** $\boldsymbol{D}$, and we expect $\boldsymbol{b}$ to be a very smooth function. Write down a **cost function** and **optimization problem** for estimating $\boldsymbol{x}$, where the cost function should use the stated signal model properties. Annotate your cost function to explain where your solution captures the different properties.

2. [6] **Convexity of transform learning**

A previous HW problem showed that the cost function $g(\boldsymbol{x}, \boldsymbol{z}) = \|\boldsymbol{T}\boldsymbol{x} - \boldsymbol{z}\|_2^2$ is jointly convex in $(\boldsymbol{x}, \boldsymbol{z})$, and this property is important for regularization with **transform sparsity** models.

Now **transform learning** involves the cost function $f(\boldsymbol{T}, \boldsymbol{Z}) = \sum_{l=1}^{L} \|\boldsymbol{T}\boldsymbol{x}_l - \boldsymbol{z}_l\|_2^2$, where $\boldsymbol{Z} \triangleq \begin{bmatrix} \boldsymbol{z}_1 & \dots & \boldsymbol{z}_L \end{bmatrix} \in \mathbb{F}^{K \times L}$, where $\boldsymbol{x}_l \in \mathbb{F}^d$ and $\boldsymbol{T} \in \mathbb{F}^{K \times d}$. This problem examines convexity of this cost function.

(a) [0] Show to yourself that you can rewrite the cost function as follows:

$$f(\boldsymbol{T}, \boldsymbol{Z}) \triangleq \sum_{l=1}^{L} \|\boldsymbol{T}\boldsymbol{x}_l - \boldsymbol{z}_l\|_2^2 = \|\boldsymbol{T}\boldsymbol{X} - \boldsymbol{Z}\|_{\mathrm{F}}^2,$$

where $\boldsymbol{X} \triangleq \begin{bmatrix} \boldsymbol{x}_1 & \dots & \boldsymbol{x}_L \end{bmatrix} \in \mathbb{F}^{d \times L}$. This Frobenius norm form may be helpful.

(b) [3] Show that f is **jointly convex** in $\boldsymbol{T}$ and $\boldsymbol{Z}$.

(c) [0] Convince yourself that the cost function is **strictly convex** in $\boldsymbol{Z}$ when $\boldsymbol{T}$ is held fixed to any value.

(d) [0] State the necessary and sufficient condition on matrix $\boldsymbol{A}$ such that $\Psi(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$ is **strictly convex** in $\boldsymbol{x}$.

(e) [3] If we hold $\boldsymbol{Z}$ fixed (to any value), then the cost function is of course convex in $\boldsymbol{T}$, but is it **strictly convex** in $\boldsymbol{T}$? The answer depends on the training data $\boldsymbol{X}$. (For example, if $\boldsymbol{X} = \boldsymbol{0}$, then definitely the cost function is *not* strictly convex in $\boldsymbol{T}$.) Find a fairly simple necessary and sufficient condition on $\boldsymbol{X}$ that determines whether the cost function is strictly convex.
Hint. My solution uses $\mathrm{vec}(\cdot)$ and properties of vec of matrix products that were derived in a previous HW problem. A starting point is $\|\boldsymbol{A}\|_{\mathrm{F}} = \|\mathrm{vec}(\boldsymbol{A})\|_2$. There probably are other approaches too.

3. [12] **Descent directions and minimizers on $\mathbb{C}^N$**

Consider $\Psi : \mathbb{C}^N \mapsto \mathbb{R}$ defined by $\Psi(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$ where $\boldsymbol{A} \in \mathbb{C}^{M \times N}$ and $\boldsymbol{y} \in \mathbb{C}^M$, and define $\boldsymbol{g}(\boldsymbol{x}) \triangleq \boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})$.

(a) [3] Show that if $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$, then $\boldsymbol{x}$ is a minimizer of Ψ, *i.e.*, $\Psi(\boldsymbol{x}) \leq \Psi(\boldsymbol{x} + \boldsymbol{z}),\ \forall \boldsymbol{z} \in \mathbb{C}^N$.
Hint. Let $\boldsymbol{r} = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}$ and note that $\boldsymbol{A}'\boldsymbol{r} = \boldsymbol{0}$.

(b) [3] Show the converse of (a): if $\hat{\boldsymbol{x}}$ is a minimizer of $\Psi(\boldsymbol{x})$ over $\mathbb{C}^N$, then $\boldsymbol{g}(\hat{\boldsymbol{x}}) = \boldsymbol{0}$.
Hint. Examine $\Psi(\hat{\boldsymbol{x}} + \boldsymbol{z})$ for $\boldsymbol{z} \triangleq -\alpha \boldsymbol{g}(\hat{\boldsymbol{x}}) = -\alpha \boldsymbol{A}'\boldsymbol{r}$ with $\boldsymbol{r} = \boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{y}$.

(c) [3] Show that $\boldsymbol{d} = -\boldsymbol{P}\boldsymbol{g}(\boldsymbol{x})$ is a **descent direction** for Ψ at $\boldsymbol{x}$ when $\boldsymbol{P}$ is a positive definite matrix.
Hint. Examine $\Psi(\boldsymbol{x} + \epsilon \boldsymbol{d})$.
Thus for the purposes of solving optimization problems with Ψ, it is reasonable to write $\nabla \Psi(\boldsymbol{x}) = \boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})$ even in the complex case, despite Ψ not being differentiable.

(d) [3] Determine (without proof) a descent direction for the cost function used for **edge-preserving image recovery** on $\mathbb{C}^N$:

$$\Psi(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta \boldsymbol{1}_K' \, \psi.(\boldsymbol{T}\boldsymbol{x})$$

for some $K \times N$ matrix $\boldsymbol{T}$, where $\psi(z) = \delta^2 \log\cosh(|z/\delta|)$. Hint. Use [wiki].

4. [31] **Complex edge-preserving image denoising**

(a) [3] Here you will use the **descent direction** derived in the previous problem to do 2D edge-preserving **image denoising**, where we want to recover $\boldsymbol{x}$ from the model $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{\varepsilon}$ using the optimization problem

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x} \in \mathbb{C}^N} \Psi(\boldsymbol{x}), \quad \Psi(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{y} - \boldsymbol{x}\|_2^2 + \beta R(\boldsymbol{x}), \quad R(\boldsymbol{x}) = \sum_k \psi([\boldsymbol{C}\boldsymbol{x}]_k, \delta),$$

where ψ denotes the **Fair potential**, and $\boldsymbol{C}$ denotes the 2D first-order finite-differencing matrix.
Following the conjecture in the course notes, determine the Lipschitz constant for the descent direction of Ψ.

(b) [10] Write a JULIA function that uses your `gd` code for GD to minimize this cost function. Your function must return $\hat{\boldsymbol{x}}$, the cost function evaluated at each iteration, and the usual optional `out` array if the user requests. (You will need this array below to compute the NRMSE each iteration.) Your function must be able to handle **complex** images.
Your function must work for large-scale problems, so it *cannot* use expensive and memory hungry operations like `svd` `svdvals` `eigen` `eigvals` `opnorm` etc.
Hint. The functions `spdiagm` and `kron` and `I(n)` are useful, though other ways to implement $\boldsymbol{C}$ are faster.
Your file should be named `dn2cx.jl` and should contain the following function:

```julia
"""
    (x,cost,out) = dn2cx(y::AbstractMatrix ; x0::AbstractMatrix = y,
     reg::Real = 1, del::Real = 2, niter::Int = 100,
     fun::Function = (x,iter) -> undef)

Perform 2D edge-preserving image denoising using GD,
to "solve" the minimization problem
`argmin_x 1/2 |y - x|^2 + reg * sum_k pot([C x]_k,del)`
where `pot()` is the Fair potential with parameter `del`
and `C` denotes the 2D first-order finite differencing matrix.

This code is (must be) general enough to handle complex-valued images!
(Uses "gd" function from previous problem.)

In
* `y` 2D noisy grayscale image of size `[M,N]`, possibly complex-valued

Option
* `x0` 2D initial guess of size `[M,N]`; default = `y`
* `niter` # number of iterations; default `100`
* `reg` regularization parameter; default `1`
* `del` potential function parameter; default `2`
* `fun` user-defined function to be evaluated with two arguments `(x,iter)`
    evaluated at `(x0,0)` and then after each iteration

Out
* `x` 2D final iterate image of size `[M,N]`
* `cost` `[niter+1]` cost function each iteration
* `out` `[niter+1]` `[fun(x0,0), fun(x1,1), ..., fun(x_niter,niter)]`
"""
function dn2cx(y::AbstractMatrix ;
    x0::AbstractMatrix = y,
    reg::Real = 1,
    del::Real = 2,
    niter::Int = 100,
    fun::Function = (x,iter) -> undef)
```

Submit your solution to `mailto:eecs556@autograder.eecs.umich.edu`.
Hint. Note that the inputs $\boldsymbol{y}$ and $\boldsymbol{x}_0$ and the output $\hat{\boldsymbol{x}}$ are all 2D images, but GD is designed to work with vectors. You will need to use `[:]` and `reshape`.

(c) [3] Apply your 2D denoising method `dn2cx` with $\delta = 2$ and $\beta = 12$ to the 2D noisy signal generated by the following code, using 300 iterations.

```
using Random: seed!
using MIRT: jim
using Plots: plot

tmp = [
  zeros(1,20);
  0 1 0 0 0 0 1 0 0 0 1 1 1 1 0 1 1 1 1 0;
  0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 1 0 0;
  0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0;
  0 0 1 1 1 1 0 0 0 0 1 1 0 0 0 0 0 1 1 0;
  zeros(1,20)
]';
xtrue = kron(10 .+ 80*tmp, ones(9,9))
xtrue = xtrue + 1im * reverse(xtrue, dims=1) # make a complex image
seed!(0) # add complex noise:
y = xtrue + 20 * (randn(size(xtrue)) + 1im * randn(size(xtrue)))
clim = [0,100]
plot(jim(real.(xtrue), title="x real", clim=clim),
  jim(imag.(xtrue), title="x imag", clim=clim),
  jim(real.(y), title="y real", clim=clim),
  jim(imag.(y), title="y imag", clim=clim))
```

Submit a screenshot of your plotting code for the next two parts to gradescope.

(d) [3] Make a plot of $\log_{10}(\Psi(\boldsymbol{x}_k))$ versus iteration k to confirm that your method is working and that we have enough iterations.

(e) [3] Make a plot of the NRMSE $\|\boldsymbol{x}_k - \boldsymbol{x}_{\text{true}}\| \,/\, \|\boldsymbol{x}_{\text{true}}\|$ versus iteration k to see how the error evolves.
A single call to your `dn2cx` function should suffice to get the data needed for both of these plots!

(f) [3] Make images of the real and imaginary parts of $\boldsymbol{x}_{\text{true}}$, $\boldsymbol{y}$, $\hat{\boldsymbol{x}}$, and $\hat{\boldsymbol{x}} - \boldsymbol{x}_{\text{true}}$.
This will be 8 total images so group them into two separate figures with 4 images each (one for the real part, one for the imaginary part).
To display grayscale images, use the `jim` function in the `MIRT` library as shown above.
For more examples, see:
http://web.eecs.umich.edu/~fessler/course/551/julia/demo/09_lrmc_nuc.html
To put multiple axes into a single plot (like `subplot` in MATLAB), use the example above or something like this:
`p1 = jim(...); p2 = jim(...); plot(p1, p2)`

(g) [3] Does this cost function Ψ have a unique minimizer $\hat{\boldsymbol{x}}$? Explain why or why not.

(h) [3] Does the cost function $\Psi(\boldsymbol{x}_k)$ decrease monotonically as k increases?
Does the NRMSE function decrease monotonically as k increases?
Discuss whether or not these two sequences are guaranteed to decrease monotonically.

---

5. [3] **Line-search for smooth inverse problems**

Consider a large-scale inverse problems having the general cost function $\Psi(\boldsymbol{x}) = \sum_{j=1}^J f_j(\boldsymbol{B}_j \boldsymbol{x})$ discussed in the course notes. Assume each f_j function is **convex** and has a **Lipschitz continuous** gradient. For later use in implementing an efficient **line search**, let $h_k(\alpha) \triangleq \sum_{j=1}^J f_j\left(\boldsymbol{u}_j^{(k)} + \alpha \boldsymbol{v}_j^{(k)}\right)$. Let L_j denote a Lipschitz constant for the gradient of f_j. Determine a Lipschitz constant of the derivative of h_k.