Homework #7, EECS 598-006, W20. Due **Thu. Mar. 13**, by 4:00PM

---

1. [19] **Low-rank matrix sensing and recovery**

The course notes describe an **MM algorithm** for **low-rank matrix completion** (LRMC), a generalization of low-rank matrix approximation. This problem considers the more general **matrix sensing** problem where we observe L linear combinations of the elements of a low-rank $M \times N$ matrix $\boldsymbol{X}$:

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

where $\boldsymbol{y} \in \mathbb{F}^L$ and $\boldsymbol{A}$ is a $L \times (MN)$ sensing matrix $\boldsymbol{\varepsilon} \in \mathbb{F}^L$ is additive noise. A natural formulation of **matrix recovery** is

$$\hat{\boldsymbol{X}} = \underset{\boldsymbol{X} \in \mathcal{X}}{\arg\min}\, \Psi(\boldsymbol{x}), \qquad \Psi(\boldsymbol{x}) = q(\boldsymbol{X}) + R(\boldsymbol{X}), \qquad q(\boldsymbol{X}) = \frac{1}{2}\left\|\boldsymbol{A}\,\mathsf{vec}(\boldsymbol{X}) - \boldsymbol{y}\right\|_2^2,$$

where the regularizer $R(\boldsymbol{X})$ and/or the set $\mathcal{X}$ should enforce or encourage $\boldsymbol{X}$ to be low rank.

There is no closed-form solution for this optimization problem for a general sensing matrix $\boldsymbol{A}$, at least not for useful choices of the regularizer and/or the constraint set $\mathcal{X}$. In general, iterative methods are required to compute $\hat{\boldsymbol{X}}$, as explored in subsequent problems. This problem focuses on the non-iterative special case that is solvable by methods from EECS 551.

(a) [3] If $\boldsymbol{A}$ is a **unitary matrix**, which implies $L = MN$, then there is a closed-form solution for $\hat{\boldsymbol{X}}$ for many typical choices of (unitarily invariant) regularizers $R(\boldsymbol{X})$.
Describe mathematically the solution $\hat{\boldsymbol{X}}$ for the specific case when the constraint set and regularizer are as follows:

$$\mathcal{X} = \left\{\boldsymbol{X} \in \mathbb{F}^{M \times N} \;:\; \mathsf{rank}(\boldsymbol{X}) \le K\right\}, \qquad R(\boldsymbol{X}) = \beta_1 \|\boldsymbol{X}\|_* + \beta_2 \frac{1}{2}\|\boldsymbol{X}\|_{\mathrm{F}}^2.$$

Your solution should be general enough to write JULIA code based on it to solve for $\hat{\boldsymbol{X}}$ given $\boldsymbol{y}$, $\boldsymbol{A}$, β_1, β_2 and K.
Hint: $\|\mathsf{vec}(\boldsymbol{X}) - \boldsymbol{z}\|_2 = \|\boldsymbol{X} - \boldsymbol{Z}\|_{\mathrm{F}}$ where $\boldsymbol{Z} =$ `reshape` $(\boldsymbol{z}, M, N)$, and review an earlier **elastic net** HW problem.

(b) [10] When $\boldsymbol{A}$ is unitary, the vector $\boldsymbol{y}$ has length MN, so we can write $\boldsymbol{y} = \mathsf{vec}(\boldsymbol{Y})$ where $\boldsymbol{Y}$ is a $M \times N$ matrix.
Write a JULIA function that computes $\hat{\boldsymbol{X}}$ given (unitary) $\boldsymbol{A}$, a matrix $\boldsymbol{Y}$, and the parameters β_1,β_2, and K.
Your file should be named `lrms_un.jl` and should contain the following function:

```
"""
    Xh = lrms_un(Y, A ; K, reg1, reg2)

Low-rank matrix recovery: `\\argmin_{X : rank(X) <= K}
 1/2 \\|A vec(X) - vec(Y)\\|_2^2 + reg1 \\|X\\|_* reg2/2 \\|X\\|_F^2`
when `A` is a `MN x MN` unitary matrix.

In
* `Y` `M x N` measurement matrix, where `vec(Y) = A vec(X) + noise`
* `A` `MN x MN` sensing matrix; must be unitary!

Option
* `K` rank constraint; default `min(M,N)`
* `reg1` regularization parameter for nuclear norm; default 1
* `reg2` regularization parameter for Frobenius norm; default 0

Output
* `Xh` `[M,N]` solution to matrix sensing problem when `A` is unitary
"""
function lrms_un(
        Y::AbstractMatrix{<:Number},
        A::AbstractMatrix{<:Number} ;
        K::Int = minimum(size(Y)),
        reg1::Real = 1,
        reg2::Real = 0)
```

Submit your solution to mailto:eecs556@autograder.eecs.umich.edu.

(c) [3] Is the optimization problem here **strictly convex** when $K = \min(M, N)$ and $\beta_1 > 0$ and $\beta_2 > 0$? Explain.

(d) [3] Find a more general class of non-square matrices $\boldsymbol{A}$ for which this matrix recovery problem has (essentially equally simple) closed-form solutions, and show how to solve for $\hat{\boldsymbol{X}}$ for this more general class. Hint. Think about a family of matrices discussed in EECS 551 that are a generalization of unitary matrices.

---

2. [37] **Low-rank matrix sensing / recovery with an MM algorithm**

The previous problem considered matrix sensing and recovery in the special case when the sensing matrix is unitary. An even simpler special case of that solution is when when the sensing matrix is $\boldsymbol{I}$. This problem considers the case of a general sensing matrix, and develops an MM approach, based on a majorizer that exploits that simple case.

(a) [3] Design a **quadratic majorizer** for the data term $q(\boldsymbol{X})$ in the previous problem that will lead to a MM algorithm where the inner minimization step corresponds to the low-rank matrix approximation solution from the previous problem.

Hint. Your majorizer should be of the form $Q(\boldsymbol{X}; \boldsymbol{X}_k) = \frac{L_q}{2} \left\| \boldsymbol{X} - \tilde{\boldsymbol{X}}_k \right\|_{\mathrm{F}}^2$ for a matrix $\tilde{\boldsymbol{X}}_k$ that depends on $\boldsymbol{X}_k$, $\boldsymbol{A}$, $\boldsymbol{y}$, $\mathsf{vec}(\cdot)$ (*i.e.*, `[:]`) and `reshape`, where L_q is a (best) Lipschitz constant that you must determine.

(b) [3] Describe mathematically the MM update step

$$\boldsymbol{X}_{k+1} = \arg\min_{\boldsymbol{X} \in \mathcal{X}} \phi_k(\boldsymbol{X}), \quad \phi_k(\boldsymbol{X}) = Q(\boldsymbol{X}; \boldsymbol{X}_k) + R(\boldsymbol{X}),$$

for the specific case when the constraint set and (**elastic net** type) regularizer are as follows:

$$\mathcal{X} = \left\{ \boldsymbol{X} \in \mathbb{F}^{M \times N} \ : \ \mathsf{rank}(\boldsymbol{X}) \leq K \right\}, \qquad R(\boldsymbol{X}) = \beta_1 \|\boldsymbol{X}\|_* + \beta_2 \frac{1}{2} \|\boldsymbol{X}\|_{\mathrm{F}}^2.$$

(c) [10] Implement a JULIA function that performs a specified number of MM iterations to find $\hat{\boldsymbol{X}}$ for the constraint set and regularizer given above. Hint. MM is somewhat similar to the ISTA method for matrix completion from 551:
http://web.eecs.umich.edu/~fessler/course/551/julia/demo/09_lrmc_nuc.html
Your file should be named `lrms_mm.jl` and should contain the following function:

```
"""
    (X,out) = lrms_mm(y, A, M::Int, N::Int ; X0, K, ...)

MM algorithm for low-rank matrix sensing: `\\argmin_{X : rank(X) <= K}
 1/2 \\|A vec(X) - y\\|_2^2 + reg1 \\|X\\|_* reg2/2 \\|X\\|_F^2`

In
* `y` vector of `L` measurements, where `y = A vec(X) + noise`
* `A` `L x MN` sensing matrix
* `M,N`    array size

Option
* `X0` `M x N` initial guess; default `zeros(M,N)`
* `K` rank constraint; default `min(M,N)`
* `reg1` regularization parameter for nuclear norm; default 1
* `reg2` regularization parameter for Frobenius norm; default 0
* `niter` # number of MM iterations; default 50
* `func` User-defined function having the following prototype:
`func(X::AbstractMatrix{<:Number}, iter::Int, cost::Function)`

It is evaluated at (X0,0,cost) and then after each iteration.
It should expect the first argument to be a `M x N` matrix,
the 2nd argument is iteration, and the 3rd is a cost function
having prototype `cost(X::AbstractMatrix{<:Number})`

Output
* `X` `M x N` final iterate
* `out`    [func(X0,0,cost), ..., func(X_niter,1,cost)]
"""
```

```
function lrms_mm(
        y::AbstractVector{<:Number},
        A::AbstractMatrix{<:Number},
        M::Int, N::Int ;
        X0::AbstractMatrix{<:Number} = zeros(eltype(y), M, N),
        K::Int = min(M,N),
        reg1::Real = 1,
        reg2::Real = 0,
        niter::Int = 50,
        func::Function =
            (X::AbstractMatrix{<:Number},iter::Int,cost::Function) -> undef,
    )
```

Submit your solution to mailto:eecs556@autograder.eecs.umich.edu.
The function argument `func` here is more convenient for the user than the `fun` in previous problem, because `lrms_mm` provides the cost function to the calling routine so the caller can evaluate it if desired. If the user wants to evaluate the time, the cost function and the NRMSE each iteration, then the user should pass a function like this:
`func = (X,iter,cost) -> [time(), cost(X), norm(X-Xtrue)]`
Computing the cost function takes extra time, so if the user wants speed then they can ignore `cost` argument and use something like: `func = (X,iter,cost) -> [time(), norm(X-Xtrue)]`

(d) [3] Write a JULIA script that self-tests your `lrms_mm` function as follows.
- Generate a random unitary matrix $\boldsymbol{A}$, a random low-rank matrix $\boldsymbol{X}$, and a data vector $\boldsymbol{y} = \boldsymbol{A}\,\mathsf{vec}(\boldsymbol{X}) + \boldsymbol{\varepsilon}$.
- Compute the minimizer $\hat{\boldsymbol{X}}$ using that analytical solution from the previous HW problem.
- Apply your MM algorithm to the data and plot the NRMSD $\left\|\boldsymbol{X}_k - \hat{\boldsymbol{X}}\right\|_{\mathrm{F}} / \left\|\hat{\boldsymbol{X}}\right\|_{\mathrm{F}}$ versus iteration k.

The purpose of this part is to help encourage you to write your own test cases for your optimization algorithm code, especially when there is a reference analytical solution.
Submit a screenshot of your test code to gradescope.

(e) [3] Submit the plot of NRMSD to gradescope. NRMSD should converge to near zero in one iteration!

(f) [0] The test in the previous two parts is imperfect because if your solution to the previous problem is incorrect and you use that solution in your MM algorithm then your NRMSD will likely go to zero anyway. Can you think of another test that avoids that flaw?

(g) [3] Write a script that applies 500 iterations of your MM algorithm to the following synthetic data and makes the plots for the next two parts.

```
seed!(0); M = 60; N = 30; Kt = 5; L = 3*Kt*(M+N); reg1 = 0.01; reg2 = 0
Xtrue = svd(randn(M,Kt)).U * Diagonal(1:Kt) * svd(randn(N,Kt)).U'
A = randn(L,M*N); sig = 0.01; y = A * Xtrue[:] + sig * randn(L)
```

As shown in the code above, use $\beta_1 = 0.01$ and $\beta_2 = 0$. Use $K = 9$ for the rank constraint in $\mathcal{X}$, even though the true $\mathsf{rank}(\boldsymbol{X}) = 5$, because in practice one will not know the exact rank. Use the default zero initialization for $\boldsymbol{X}_0$.
Submit a screenshot of your script to gradescope.

(h) [3] Plot the cost function versus iteration on the usual log10 scale.

(i) [3] Plot the NRMSE versus iteration on the usual log10 scale.

(j) [3] Apply your MM to data $\boldsymbol{y}$ generated as follows:

```
using Random: seed!
fun = (x,p) -> p == 1 ? x == 1 : [x >= p; fun(x - p*(x >= p), p/2)]
tmp = [0 14 1 1 1 14 0 0 0 9 15 8 8 4 8 8 15 9 0]
tmp = hcat([Int.(fun(v, 2^3)) for v in tmp]...)
tmp = [zeros(Int, 1,19); tmp; zeros(Int, 1,19)]'
Xtrue = kron(10 .+ 80*tmp, ones(5,5)); (M,N) = size(Xtrue)
seed!(0); L = 6*(M+N); A = randn(L, M*N); sig = 5
y = A * Xtrue[:] + sig * randn(L);
```

This is a more realistic problem where you have to select the rank K, the regularization parameters β_1, β_2, and the number of iterations yourself. I found parameters that gave a SNR of over 45dB, where SNR $= 20\log_{10}\left(\|\boldsymbol{X}\|_{\mathrm{F}} / \left\|\hat{\boldsymbol{X}} - \boldsymbol{X}\right\|_{\mathrm{F}}\right)$. You should strive for results at least that good. Submit a picture of your $\hat{\boldsymbol{X}}$ and your error image $\hat{\boldsymbol{X}} - \boldsymbol{X}$ and report what values you used for all parameters.

(k) [3] Speculate about what is the minimum possible value of L here (in terms of K, M and N). How close to that minimum is L in the previous part?

---

3. [3] Page 4.13 of the course notes derives a quadratic majorizer for the data term of the LASSO problem. That derivation has two constants: c_1 and c_2. Determine the values of those constants. (The constants themselves are not so important, but it is important that you are able to understand all of the steps of those derivations.)

---

4. [3] Suppose $\Psi : \mathbb{R}^N \mapsto \mathbb{R}$ has a $\boldsymbol{S}$-Lipschitz gradient, for an invertible matrix $\boldsymbol{S}$. We have a search direction $\boldsymbol{d}$ and we want to perform the **line search** $\arg\min_{\alpha\in\mathbb{R}} h(\alpha),\ h(\alpha) = \Psi(\boldsymbol{x}_k + \alpha\boldsymbol{d})$. Determine a Lipschitz constant for $\dot{h}$.

Hint: $\langle \boldsymbol{x},\, \boldsymbol{z}\rangle = \langle \boldsymbol{S}^{-1}\boldsymbol{x},\, \boldsymbol{S}'\boldsymbol{z}\rangle$.

Optional: generalize to the case where $\Psi : \mathbb{C}^N \mapsto \mathbb{R}$.

Optional problem(s)

---

5. [0] **GD convergence in 1 step**

If $f(x)$ is a 1D convex quadratic function whose derivative has (best) Lipschitz constant L, then the GD algorithm with step size $1/L$ is the same as Newton's method and it converges in one iteration from any starting point, *i.e.*,
$x_k = x_* = \arg\min_x f(x),\ \forall k \in \mathbb{N}$.
Prove or disprove the following converse: if $f(x)$ is a 1D smooth convex function whose derivative has (best) Lipschitz constant L, and the GD algorithm with step size $1/L$ converges in one iteration from any starting point, *i.e.*, $x_1 = x_* = \arg\min_x f(x)$, then $f(x)$ is a quadratic function.

---

6. [0] **Lipschitz continuity**

A function $\boldsymbol{g}$ is $\boldsymbol{S}$-Lipschitz continuous for an invertible matrix $\boldsymbol{S}$ iff $\left\|\boldsymbol{S}^{-1}\left(\boldsymbol{g}(\boldsymbol{x}) - \boldsymbol{g}(\boldsymbol{z})\right)\right\|_2 \le \left\|\boldsymbol{S}'\left(\boldsymbol{x} - \boldsymbol{z}\right)\right\|_2,\ \forall \boldsymbol{x}, \boldsymbol{z}$.
(a) Prove that $\boldsymbol{g}$ is **Lipschitz continuous** with some Lipschitz constant L and specify L. Hint. All norms are equivalent.
(b) Is your L the best Lipschitz constant? Prove or disprove with a counter-example.

---

7. [0] **Continuity of rank1 matrix approximation**

For any $M \times N$ matrix $\boldsymbol{X}$ with $M, N > 1$, let $R_1(\boldsymbol{X})$ denote the best rank-1 approximation of $\boldsymbol{X}$ in the Frobenius norm sense. Determine whether $R_1(\cdot)$ is a **Lipschitz continuous** function and, if so, find its **best Lipschitz constant**.

Optional challenge: generalize to the low-rank matrix approximation method that uses singular-value soft-thresholding.