EE-556: MATHEMATICS OF DATA: FROM THEORY TO COMPUTATION
LABORATORY FOR INFORMATION AND INFERENCE SYSTEMS
FALL 2021

INSTRUCTOR:
PROF. VOLKAN CEVHER

HEAD TAS
FABIAN LATORRE, ALI KAVIS

HOMEWORK-3

This homework covers Lectures 9-13 in the course and consists of two parts. In the first part, we motivate projection-free convex optimization by comparing the computational costs of the proximal operators $\mathrm{prox}_{\mathcal{X}}$ and the linear minimization oracles $\mathrm{lmo}_{\mathcal{X}}$ for a given set $\mathcal{X}$. We then implement the Frank-Wolfe method for solving a blind image deconvolution problem. In the second part, we use a variant of the Frank-Wolfe method for linearly constrained problems as well as the Vu-Condat primal-dual method to obtain numerical solutions to the classical k-means clustering problem.

1 Crime Scene Investigation with Blind Image Deconvolution (40 pts)

You are working with the local police to help identify a license plate of a car involved in a crime scene investigation. Unfortunately, the CCTV image of the car is blurry. In this exercise, we simulate this scenario with a deblurred license plate image found from the internet.[1]

Deblurring is an instance of the blind deconvolution problem: Given two unknown vectors $\mathbf{x}, \boldsymbol{w} \in \mathbb{R}^L$, we observe their circular convolution $\mathbf{y} = \boldsymbol{w} * \mathbf{x}$, i.e.,

$$y_\ell = \sum_{\ell'=1}^{L} w_{\ell'} x_{\ell-\ell'+1},$$

where the index $\ell - \ell' + 1$ in the sum is understood to be modulo L.

Blind deconvolution seeks to separate $\boldsymbol{w}$ and $\mathbf{x}$, given $\mathbf{y}$. The operative word *blind* comes from the fact that we do not have much prior information about the signals. In this case, what we can assume is that $\boldsymbol{w}$ and $\mathbf{x}$ belong to *known* subspaces of $\mathbb{R}^L$ of dimension K and N, i.e., we write

$$\boldsymbol{w} = \boldsymbol{B}\boldsymbol{h}, \quad \boldsymbol{h} \in \mathbb{R}^K$$
$$\mathbf{x} = \boldsymbol{C}\boldsymbol{m}, \quad \boldsymbol{m} \in \mathbb{R}^N$$

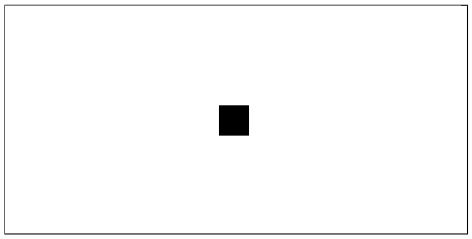
for some $L \times K$ matrix $\boldsymbol{B}$ and $L \times N$ matrix $\boldsymbol{C}$. The columns of these matrices form bases for the subspaces in which $\boldsymbol{w}$ and $\mathbf{x}$ live.

As we have seen in Homework 1, natural images have sparse wavelet expansions. Hence, the image $\mathbf{x}$ can be expressed as $\mathbf{x} = \boldsymbol{C}\boldsymbol{m}$ with $\boldsymbol{C}$ is the matrix formed by a subset of the columns of the wavelet transform matrix. In addition, the blur kernel $\boldsymbol{w}$ is typically due to simple or "sparse" motion, which can be written as $\boldsymbol{w} = \boldsymbol{B}\boldsymbol{h}$ with $\boldsymbol{B}$ is the matrix formed by a subset of the columns of the identity matrix.

In deblurring, $\mathbf{x}$ corresponds to the image we want to recover (i.e., the license plate) and $\boldsymbol{w}$ to a 2D blur kernel. Thus, the 2D convolution $\mathbf{y} = \boldsymbol{w} * \mathbf{x}$ produces a blurred image. We assume that we know or can estimate the support of the blur kernel (i.e., the location of its nonzero elements). In real applications, the support can be estimated by an expert using the physical information such as the distance of object to the focus and the camera, the speed of the camera and/or the object, camera shutter speed, etc.

In this experiment, we use a very rough estimate for the support - a box at the center of the domain, whose size we have roughly tuned. Interestingly, it is possible to make the plate readable even in this setting.

Blurred image of licence plate $\mathbf{y}$


Estimate of the support of blur kernel $\boldsymbol{w}$

[1]https://plus.maths.org/issue37/features/budd/blurredplate.jpg

Reformulating the problem

We now reformulate the blind image deconvolution problem, so that we can apply the constrained optimization algorithms we have seen in the course. Let $\boldsymbol{b}$ be the L-point normalized discrete Fourier transform (DFT) of the observation $\mathbf{y}$, i.e, $\boldsymbol{b} = \mathbf{F}\mathbf{y}$, where F is the DFT matrix. Then, $\boldsymbol{b}$ can be written as $\boldsymbol{b} = \boldsymbol{A}(\mathbf{X})$ where $\mathbf{X} = \boldsymbol{h}\boldsymbol{m}^\top$ and $\boldsymbol{A}$ is a linear operator. Explicit expression of this linear operator $\boldsymbol{A}$ is out of the scope of this homework, c.f., [1] for further details. This reformulation allows us to express $\mathbf{y}$, which is a nonlinear combination of the coefficients of $\boldsymbol{h}$ and $\boldsymbol{m}$, as a linear combination of the entries of their outer product $\mathbf{X} = \boldsymbol{h}\boldsymbol{m}^\top$. Note that given $\boldsymbol{B}$ and $\boldsymbol{C}$, recovering $\boldsymbol{m}$ and $\boldsymbol{h}$ from $\boldsymbol{b}$ is the same as recovering $\mathbf{x}$ and $\boldsymbol{w}$ from $\mathbf{y}$.

Since $\mathbf{X}$ is a rank one matrix, we can use the nuclear norm to enforce approximately low-rank solutions. Then, we can formulate the blind deconvolution problem as follows:

$$\boldsymbol{X}^\star \in \arg\min_{\boldsymbol{X}} \left\{ \frac{1}{2}\|\mathbf{A}(\boldsymbol{X}) - \boldsymbol{b}\|_2^2 : \|\boldsymbol{X}\|_* \leq \kappa,\ \boldsymbol{X} \in \mathbb{R}^{p\times m} \right\}, \tag{1}$$

where $\kappa > 0$ is a tuning parameter.

First, note that our problem is constrained to the nuclear norm ball $\mathcal{X} = \{\boldsymbol{X} : \boldsymbol{X} \in \mathbb{R}^{p\times m}, \|\boldsymbol{X}\|_* \leq \kappa\}$. We have seen in the lectures that we can ensure the iterates of our algorithms stay within $\mathcal{X}$ in one of two ways. In the first way, we use projections, computed via the proximal operator $\text{prox}_{\delta_\mathcal{X}}$. In the second way, we use linear minimization oracles the $\text{lmo}_\mathcal{X}$ within a conditional gradient framework that take simplicial combinations of elements from the set $\mathcal{X}$ whereby producing iterates remaining in $\mathcal{X}$. The following three exercises will help you understand what kind of computations are involved for each of these two operators, and how their computational complexity compares.

Problem 1.1 (9 pts): Computing projections onto $\mathcal{X}$

(a) (2 pts) Recall that given a set $\mathcal{X} \subset \mathbb{R}^{p\times m}$, its corresponding projection operator is given by $\text{proj}_\mathcal{X}(\boldsymbol{Z}) = \arg\min_{\mathbf{X}\in\mathcal{X}}\{\|\mathbf{X} - \mathbf{Z}\|_F^2\},\ \forall \boldsymbol{Z} \in \mathbb{R}^{p\times m}$. Using the definition of the proximal operator given in class, show the equivalence between the projection operator and the proximal operator:

$$\text{proj}_\mathcal{X}(\boldsymbol{Z}) = \text{prox}_{\delta_\mathcal{X}}(\boldsymbol{Z}),$$

where $\delta_\mathcal{X}$ is the indicator function of $\mathcal{X}$.

(b) (3 pts) The projection operator of convex sets has an interesting and useful property: it is non-expansive. Mathematically, we write:

$$\|\text{proj}_\mathcal{X}(\mathbf{x}) - \text{proj}_\mathcal{X}(\mathbf{y})\| \leq \|\mathbf{x} - \mathbf{y}\|, \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^p, \tag{2}$$

where $\|\cdot\|$ denotes the usual Euclidean norm and $\mathcal{X}$ is a non-empty, closed and convex set. For keeping things simple, in this point we use the space of vectors $\mathbb{R}^p$ in which $\mathcal{X} \in \mathbb{R}^p$ is a closed convex set.

Informally, (2) means that the distance between the *projections* of the two points onto $\mathcal{X}$ will be *no greater* than the distance between the points themselves. Conversely, for non-convex sets, this does not hold (you can try building a counterexample for a doughnut-shaped set).

Prove inequality (2) starting from the equivalent characterization of the Euclidean projection: $\mathbf{z}^* = \text{proj}_\mathcal{X}(\mathbf{x}) \iff \langle \mathbf{x} - \mathbf{z}^*, \mathbf{z} - \mathbf{z}^* \rangle \leq 0,\ \forall \mathbf{z} \in \mathcal{X}$.

(c) (4 pts) Let $\mathbf{Z} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ be the singular value decomposition of $\mathbf{Z} \in \mathbb{R}^{p\times m}$. Denote the diagonal of $\boldsymbol{\Sigma} \in \mathbb{R}^{s\times s}$ by a vector $\sigma \in \mathbb{R}^s$, where $s = \min\{p, m\}$. Let σ^{ℓ_1} be the projection of σ onto the ℓ_1-norm ball $\{\mathbf{x} : \mathbf{x} \in \mathbb{R}^s, \|\mathbf{x}\|_1 \leq \kappa\}$ with radius κ. Show that the projection of this matrix onto the nuclear norm ball $\mathcal{X} = \{\boldsymbol{X} : \boldsymbol{X} \in \mathbb{R}^{p\times m}, \|\boldsymbol{X}\|_* \leq \kappa\}$ can be computed by projecting σ onto the ℓ_1 norm ball, i.e.,

$$\text{proj}_\mathcal{X}(\boldsymbol{Z}) = \boldsymbol{U}\boldsymbol{\Sigma}^{\ell_1}\boldsymbol{V}^\top,$$

where $\Sigma^{\ell_1} \in \mathbb{R}^{s\times s}$ denotes the diagonal matrix with diagonal σ^{ℓ_1}.

(Hint: Use Mirsky's inequality: $\|\boldsymbol{X} - \mathbf{Z}\|_F \geq \|\boldsymbol{\Sigma}_\boldsymbol{X} - \boldsymbol{\Sigma}_\mathbf{Z}\|_F$, where $\boldsymbol{\Sigma}_\boldsymbol{X}, \boldsymbol{\Sigma}_\boldsymbol{Z} \in \mathbb{R}^{s\times s}$ are the diagonal matrices of the singular values of $\boldsymbol{X}, \boldsymbol{Z}$ respectively.)

Problem 1.2 (4 pts): Computing the linear minimization oracle of $\mathcal{X}$

Problem 1.1 shows that projection onto the nuclear norm ball requires computing the singular value decomposition. The computational complexity of the singular value decomposition is $O(\min(m^2 p, mp^2))$, which can easily become a computational bottleneck if m

or p are large. This bottleneck increased the popularity of algorithms that leverage the linear minimization oracle (lmo) instead (e.g., [5, 11]):

$$\mathrm{lmo}_{\mathcal{X}}(\boldsymbol{Z}) = \arg\min_{\boldsymbol{X}\in\mathcal{X}} \langle \boldsymbol{X}, \boldsymbol{Z}\rangle \qquad \text{where} \qquad \langle \boldsymbol{X}, \boldsymbol{Z}\rangle = \mathrm{Tr}(\boldsymbol{Z}^\top \boldsymbol{X}).$$

Note that $\mathrm{lmo}_{\mathcal{X}}(\boldsymbol{Z})$ is not single valued in general. With abuse of terminology, when we say that we compute the lmo, we actually mean that we compute an instance $\boldsymbol{X}$ such that $\boldsymbol{X} \in \mathrm{lmo}_{\mathcal{X}}(\boldsymbol{Z})$.

Show that the $\mathrm{lmo}_{\mathcal{X}}$ when $\mathcal{X}$ is the nuclear norm ball: $\mathcal{X} = \{\boldsymbol{X} : \boldsymbol{X} \in \mathbb{R}^{p\times m}, \|\boldsymbol{X}\|_* \leq \kappa\}$ gives the following output:

$$-\kappa \mathbf{u}\mathbf{v}^T \in \mathrm{lmo}_{\mathcal{X}}(\boldsymbol{Z}),$$

where $\mathbf{u}$ and $\mathbf{v}$ are the left and right singular vectors that correspond to the largest singular value of $\boldsymbol{Z}$.

(Hint: By definition $\kappa\mathbf{u}\mathbf{v}^T \in \mathcal{X}$. You just need to show $\langle \boldsymbol{X}, \boldsymbol{Z}\rangle \geq \langle -\kappa\mathbf{u}\mathbf{v}^T, \boldsymbol{Z}\rangle$ for all $\boldsymbol{X} \in \mathcal{X}$.)

Problem 1.3 (8 pts): Comparing the scalability of $\mathrm{proj}_{\mathcal{X}}(\boldsymbol{Z})$ and $\mathrm{lmo}_{\mathcal{X}}(\boldsymbol{Z})$

In this exercise, we will compare the execution time of $\mathrm{proj}_{\mathcal{X}}(\boldsymbol{Z})$ and $\mathrm{lmo}_{\mathcal{X}}(\boldsymbol{Z})$ on two datasets provided to you in the codes. These datasets consist of the ratings given by MovieLens users to movies in a given list. The 100k dataset consists of 100,000 ratings from 1000 users on 1700 movies. The 1M dataset consists of 1 million ratings from 6000 users on 4000 movies.

As you likely figured out already from the numbers above, users do not rate all of the movies, and therefore, we model the ratings as entries of a low-rank matrix, where rows correspond to different users and columns correspond to different movies. A classical task in machine learning is to predict the value of the missing entries, which is called the matrix completion problem.

Many other tasks can be formulated as convex minimization problems, constrained to the nuclear-norm ball, which captures a low rank model since it is the atomic norm of rank-1 matrices (see Lecture 4). A good optimization algorithm must ensure feasibility in a scalable way: For instance, the famous Netflix competition data consists of 100480507 ratings that 480189 users gave to 17770 movies (much bigger than the datasets above). Projecting a matrix of this size onto the nuclear-norm ball is indeed demanding.

Based on your derivations in 1.1.b and 1.2, you will

(a) (4 pts) Implement the projection operator as a function called `proj_nuc` in the file `prb_1.3a.py` (or alternatively, the *jupyter notebook* with the same name). You can use the helper function `proj_L1` from the file `proj_L1.py`.

Set $\kappa = 5000$ and measure the computation time of the projection operator with the 100k and the 1M MovieLens datasets. You can do this by running the script `prb_1.3a.py` (or alternatively, the *jupyter notebook* with the same name), which loads the datasets, constructs the data matrix, and times the evaluation of the projection operator. Write the values you get in your report. Run your script at least for 5 times and report the average timing.

(b) (4 pts) Implement the lmo with $\mathcal{X}$ as a function called `lmo_nuc` in the file `prb_1.3b.py` (or alternatively, the *jupyter notebook* with the same name). Set $\kappa = 5000$ and measure the computation time for the 100k and 1M Movielens datasets. You can do so by running the script `prb_1.3b.py` (or alternatively, the *jupyter notebook* with the same name), which loads the datasets, constructs the data matrix, and times the evaluation of the lmo.

Write the values you get in your report. Run your script at least for 5 times and report the average timing. Compare these values with the computation time of the projection operator from Problem 1.3.a.

Problem 1.4 (19 pts): Frank-Wolfe for blind image deconvolution

We will apply the Frank-Wolfe algorithm to solve the optimization problem given in (1). The Frank-Wolfe algorithm is one of the earliest algorithms that avoids projections. Instead of projections, it leverages lmos (for a very good survey see [5]):

$$\mathrm{lmo}(\nabla f(\boldsymbol{Z})) = \arg\min_{\boldsymbol{X}\in\mathcal{X}} \langle \nabla f(\boldsymbol{Z}), \boldsymbol{X}\rangle,$$

where $\mathcal{X} = \{\boldsymbol{X} : \|\boldsymbol{X}\|_* \leq \kappa,\ \boldsymbol{X} \in \mathbb{R}^{p\times m}\}$ as in Part 1. It applies to the generic constrained minimization template with a smooth objective function, $\min_{\boldsymbol{X}}\{f(\boldsymbol{X}) : \boldsymbol{X} \in \mathcal{X},\ \mathcal{X}\text{ - convex, compact}\}$ as follows:

Frank-Wolfe's algorithm

```
1. Choose $\boldsymbol{X}^0 \in \mathcal{X}$.
2. For $k = 0, 1, \dots$ perform:
        $\hat{\mathbf{X}}^k := \mathrm{lmo}(\nabla f(\boldsymbol{X}^k))$,
        $\mathbf{X}^{k+1} := (1-\gamma_k)\mathbf{X}^k + \gamma_k \hat{\mathbf{X}}^k$,
   where $\gamma_k := 2/(k+2)$.
```

(a) (4 pts) Recall that the Frank-Wolfe algorithm applies only for smooth objectives. Show that the objective function is smooth in the sense its gradient is Lipschitz continuous.

(b) (15 pts) Complete the missing lines of the `frank_wolfe` function in the `test_deblur.py` file (or alternatively, the *jupyter notebook* with the same name). We provide you the linear operators that you need to compute the lmo in the code. Note that we do not need to store and use the linear operator $\boldsymbol{A}$ in the ambient dimensions. In fact, for storage and arithmetic efficiency, we should avoid explicitly writing $\boldsymbol{A}$. You can find more details about this aspect as comments in the code.

Test your implementation using the script `test_deblur.py` (or alternatively, the *jupyter notebook* with the same name). Tune the parameter κ until the license plate number becomes readable. What is the license plate number? You can also tune the support of the blur kernel and try to get better approximations.

2 Hands-on experiment 2: k-means Clustering by Semidefinite Programming (SDP) (40 pts)

Clustering is an unsupervised machine learning problem in which we try to partition a given data set into k subsets based on distance between data points, or in general based on their similarity. Hence, we seek to find k centers and to assign each data point to one of the centers such that the sum of the square distances between them are minimal [7]. This problem is known to be NP-hard.

Mathematical formulation: Given a set of n points in a $d-$dimensional Euclidean space, denoted by

$$S = \{\mathbf{s}_i = (s_{i1}, \cdots, s_{id})^\top \in \mathbb{R}^d \;\; i = 1, \cdots, n\},$$

we seek to find an assignment of the n points into k disjoint clusters $\mathcal{S} = (S_1, \cdots, S_k)$ whose centers are $\mathbf{c}_j$ $(j = 1, \cdots, k)$ based on the total sum-of-squared Euclidean distances from each point $\mathbf{s}_i$ to its assigned cluster centroid $\mathbf{c}_i$, i.e.,

$$f(S, \mathcal{S}) = \sum_{j=1}^{k} \sum_{i=1}^{|S_j|} \|\mathbf{s}_i^j - \mathbf{c}_j\|^2, \qquad (k\text{-means value})$$

where $|S_j|$ is the number of points in S_j, and $\mathbf{s}_i^j$ is the i^{th} point in S_j. The most popular algorithm for k-means is by Llyod

```
Lloyd's algorithm for k-means
1. Choose initial cluster centers $\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_k$
2. Repeat until convergence:
    { Assignment step:  $\mathbf{s}_i$ belongs to cluster $j$, where$j := \text{argmin}_{j\in[1,k]}\|\mathbf{s}_i - \mathbf{c}_j\|$
    { Update each cluster center:   $\mathbf{c}_j = \frac{1}{|S_j|}\sum_{i=1}^{|S_j|}\mathbf{s}_i^j$
```

Note that the algorithm converges to local optimal points, so (k-means value) can be arbitrarily bad depending on the initialization of the cluster centers.

SDP relaxation of the problem: The work [7] proposes an SDP-relaxation to approximately solve the aforementioned model-free $k-$means clustering problem. The resulting optimization problem[2] takes the standard semidefinite programming form

$$\boldsymbol{X}^\star \in \arg\min_{\boldsymbol{X}} \Big\{ \langle \boldsymbol{C}, \boldsymbol{X} \rangle : \underbrace{\boldsymbol{X}\mathbf{1} = \mathbf{1}}_{A_1(\boldsymbol{X})=b_1}, \underbrace{\boldsymbol{X}^\top\mathbf{1} = \mathbf{1}}_{A_2(\boldsymbol{X})=b_2}, \underbrace{\boldsymbol{X} \geq 0}_{B(\boldsymbol{X})\in\mathcal{K}}, \underbrace{\mathrm{Tr}(\boldsymbol{X}) \leq \kappa, \; \boldsymbol{X} \in \mathbb{R}^{p\times p}, \; \mathbf{X} \succeq 0}_{\mathcal{X}} \Big\}, \tag{3}$$

where $C \in \mathbb{R}^{p\times p}$ is the Euclidean distance matrix between the data points. $\mathrm{Tr}(\boldsymbol{X}) \leq \kappa$ enforces approximately low-rank solutions, the linear inclusion constraint $\boldsymbol{X} \geq 0$ is element-wise nonnegativity of $\boldsymbol{X}$, the linear equality constraints $\boldsymbol{X}\mathbf{1} = \mathbf{1}$ and $\boldsymbol{X}^\top\mathbf{1} = \mathbf{1}$ require row and column sums of $\boldsymbol{X}$ to be equal to 1's, and $\mathbf{X} \succeq 0$ means that $\mathbf{X}$ is positive semi-definite. Recall that $\mathrm{Tr}(\boldsymbol{X}) = \|\boldsymbol{X}\|_*$ for any positive semi-definite matrix $\boldsymbol{X}$.

Algorithm 1. The SDP in (3) can be reformulated as

$$\min_{x\in\mathcal{X}} \quad f(x) + g_1(A_1(x)) + g_2(A_2(x)) \qquad \text{subject to} \qquad B(x) \in \mathcal{K} \tag{4}$$

where $f(x) = \langle \boldsymbol{C}, x \rangle$ is a smooth convex function, $g_1 = \delta_{\{b_1\}}(\cdot)$ is the indicator function of singleton $\{b_1\}$, $g_2 = \delta_{\{b_2\}}(\cdot)$ is the indicator function of singleton $\{b_2\}$ and $\mathcal{K}$ is the positive orthant for which computing the projection is easy.

Note that the classical Frank-Wolfe method does not apply to this problem due to nonsmooth terms g_1, g_2. In the sequel, we will attempt to solve this problem with the HomotopyCGM method proposed in [11] to handle the non-smooth problems with a conditional gradient based method. The algorithm to solve the problem in (4) is given below.

[2]See section (2) of [7] for details of this relaxation and Lecture 13 for a brief introduction.