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 $\text{prox}_{\mathcal{X}}$ and the linear minimization oracles $\text{lmo}_{\mathcal{X}}$ 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

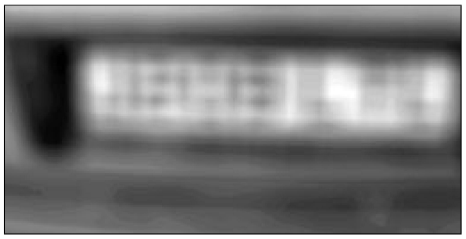
$$\boldsymbol{w} = \boldsymbol{B}\boldsymbol{h}, \quad \boldsymbol{h} \in \mathbb{R}^K$$
$$\mathbf{x} = \boldsymbol{C}\boldsymbol{m}, \quad \boldsymbol{m} \in \mathbb{R}^N$$

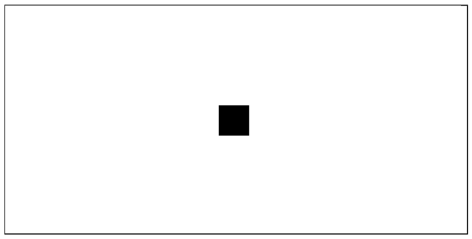
for some $L \times K$ matrix $\boldsymbol{B}$ and $L \times N$ matrix $\boldsymbol{C}$. The columns of these matrices form bases for the subspaces in which $\boldsymbol{w}$ and $\mathbf{x}$ live.

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:

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

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

First, note that our problem is constrained to the nuclear norm ball $\mathcal{X} = \{\boldsymbol{X} : \boldsymbol{X} \in \mathbb{R}^{p\times m}, \|\boldsymbol{X}\|_* \le \kappa\}$. 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 $\mathrm{prox}_{\delta_\mathcal{X}}$. In the second way, we use linear minimization oracles the $\mathrm{lmo}_\mathcal{X}$ 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 $\mathrm{proj}_\mathcal{X}(\boldsymbol{Z}) = \arg\min_{\mathbf{X}\in\mathcal{X}}\{\|\mathbf{X} - \mathbf{Z}\|_F^2\},\ \forall \boldsymbol{Z} \in \mathbb{R}^{p\times m}$. Using the definition of the proximal operator given in class, show the equivalence between the projection operator and the proximal operator:

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

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:

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

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: $\mathbf{z}^* = \mathrm{proj}_\mathcal{X}(\mathbf{x}) \iff \langle \mathbf{x} - \mathbf{z}^*, \mathbf{z} - \mathbf{z}^* \rangle \le 0,\ \forall \mathbf{z} \in \mathcal{X}$.

(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 $\{\mathbf{x} : \mathbf{x} \in \mathbb{R}^s, \|\mathbf{x}\|_1 \le \kappa\}$ with radius κ. Show that the projection of this matrix onto the nuclear norm ball $\mathcal{X} = \{\boldsymbol{X} : \boldsymbol{X} \in \mathbb{R}^{p\times m}, \|\boldsymbol{X}\|_* \le \kappa\}$ can be computed by projecting σ onto the ℓ_1 norm ball, i.e.,

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

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

(Hint: Use Mirsky's inequality: $\|\boldsymbol{X} - \mathbf{Z}\|_F \ge \|\boldsymbol{\Sigma}_\boldsymbol{X} - \boldsymbol{\Sigma}_\mathbf{Z}\|_F$, 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 $O(\min(m^2p, mp^2))$, 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, $\min_{\boldsymbol{X}}\{f(\boldsymbol{X}) : \boldsymbol{X} \in \mathcal{X}, \ \mathcal{X}$ - convex, compact$\}$ as follows:

Frank-Wolfe's algorithm

```
1. Choose $X^0 \in \mathcal{X}$.
2. For $k = 0, 1, \ldots$ 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

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

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

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

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.

```
Homotopy Conditional Gradient Method (HCGM)
1. Choose $x^0 \in \mathcal{X}$ and $\beta_0 > 0$
2. For $k = 1, 2, \ldots$ perform:
    $\gamma_k = 2/(k+1)$, and $\beta_k = \beta_0/\sqrt{k+1}$
    $v_k = \beta_k \nabla f(x_k) + A_1^\top(A_1(x_k) - b_1) + A_2^\top(A_2(x_k) - b_2) + (x_k - \mathrm{proj}_{\mathcal{K}}(x_k))$
    $\hat{x}^k := \mathrm{argmin}_{x \in \mathcal{X}} \langle v_k, x \rangle$
    $x^{k+1} := (1 - \gamma_k)x^k + \gamma_k \hat{x}^k$
3.Output: $x^{k+1}$
```

Algorithm 2. Another option for solving this problem is Vu-Condat method we have seen in Lecture 12. For solving the problem

$$\min_x f(x) + g(A(x)) + h(x),$$

where, $f(x) = \langle \boldsymbol{C}, x \rangle$ and $h(x) = \delta_{\mathcal{X}}(x)$. Moreover, $g(A(x)) = g_1(A_1(x)) + g_2(A_2(x)) + \delta_{\mathcal{K}}(Bx)$ for a suitably defined g and A. In particular, define

$$z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} A_1 x \\ A_2 x \\ Bx \end{bmatrix}, \quad A = \begin{bmatrix} A_1 \\ A_2 \\ B \end{bmatrix} \quad \Rightarrow \quad z = A(x), \tag{5}$$

and

$$g(z) = \delta_{\{b_1\}}(z_1) + \delta_{\{b_2\}}(z_2) + \delta_{\mathcal{K}}(z_3). \tag{6}$$

Vu-Condat algorithm iterates as follows (recall that g^* denotes the conjugate function of g):

```
Vu-Condat method
1. Choose $x^0 \in \mathcal{X}, y^0 \in \mathbb{R}^{2p+p^2}$ and $\tau, \sigma > 0$
2. For $k = 1, 2, \ldots$ perform:
    $x^{k+1} = \mathrm{prox}_{\tau h}(x^k - \tau(\nabla f(x^k) + A^\top y^k))$
    $\bar{x}^{k+1} = 2x^{k+1} - x^k$
    $y^{k+1} = \mathrm{prox}_{\sigma g^*}(y^k + \sigma A(\bar{x}^{k+1}))$
3.Output: $x^k$
```

Dataset: We use a similar setup described by [6]. We use the fashion-MNIST data in [10] which is released as a possible replacement for the MNIST handwritten digits . Each data point is a 28x28 grayscale image, associated with a label from 10 classes. Figure 2 depicts 3 row samples from each class. Classes are labeled from 0 to 9. First, we extract the meaningful features from this dataset using a simple 2 layers neural network with a sigmoid activation. Then, we apply neural network to 1000 test samples from the same dataset, which gives us a vector $\mu \in \mathbb{R}^{10}$ where each entry represents the probability being in that class. Then, we form the pairwise distance matrix $\boldsymbol{C}$ by using this probability vectors.[3]

PROBLEM 2.1: METHODS FOR CLUSTERING THE FASHION-MNIST DATA (40 PTS)

(a) (5 pts) Show that the domain $\mathcal{X} = \{\boldsymbol{X} : \mathrm{Tr}(\boldsymbol{X}) \leq \kappa,\ \boldsymbol{X} \in \mathbb{C}^{p \times p},\ \mathbf{X} \geq 0\}$ is a convex set. For this purpose, apply the definition of set convexity.

(b) (10 pts) Given a linear inclusion constraint $Tx \in \mathcal{Y}$, the corresponding quadratic penalty function is given by

$$\mathrm{QP}_{\mathcal{Y}}(x) = \mathrm{dist}^2(Tx, \mathcal{Y}) = \min_{y \in \mathcal{Y}} \|y - Tx\|^2.$$

Write down the constraints in (4) in the quadratic penalty form and **show that** the penalized objective takes the form

$$f(x) + \frac{1}{2\beta}\|A_1(x) - b_1\|^2 + \frac{1}{2\beta}\|A_2(x) - b_2\|^2 + \frac{1}{2\beta}\mathrm{dist}^2(x, \mathcal{K}),$$

and **show that** the gradient of the penalized objective is equal to v_k/β in the algorithm.
(Hint: You can write $\mathrm{dist}^2(Tx, \mathcal{Y}) = \|y^* - Tx\|^2$, where $y^* = \arg\min_{y \in \mathcal{Y}} \|y - Tx\|^2$. and take the derivative with respect to $\mathbf{X}$ without worrying about y^* depending on $\mathbf{X}$, thanks to Danskin's theorem (*cf.*, Lecture 10).)

[3]In the code, you do not need to worry about any of the processing details mentioned here. You are directly given the matrix $\mathbf{C}$.

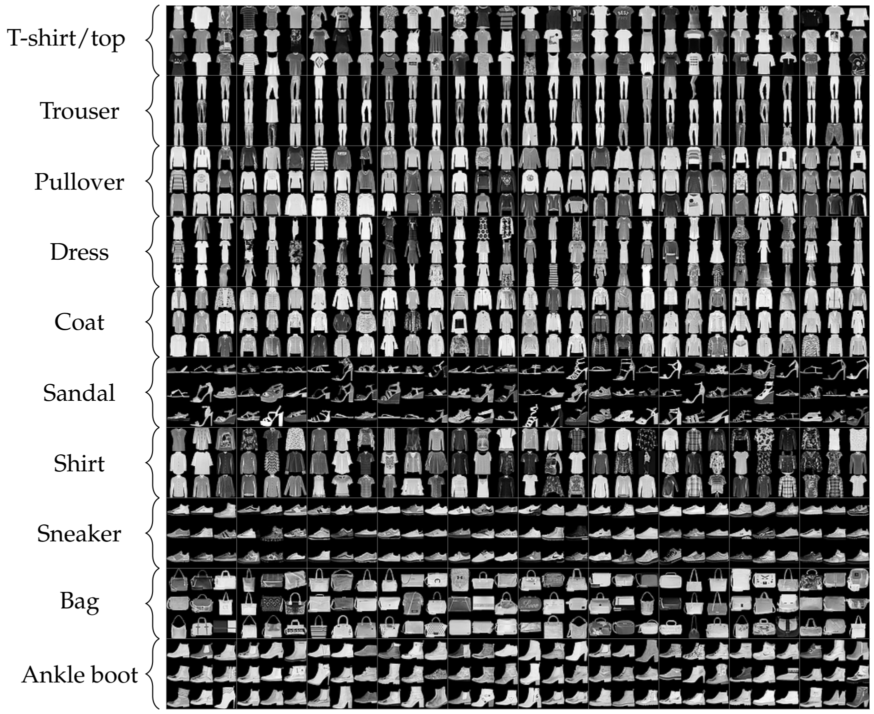


Figure 1: Fashion MNIST data

(c) (5 pts) Write down v_k explicitly by deriving the gradient and projection specific to Problem (4).

(d) (5 pts) Using the definitions we used for g and A in (5) and (6), show that the y^{k+1} update step of Vu-Condat can be written in the form

$$y^{k+1} := \begin{bmatrix} y_1^{k+1} \\ y_2^{k+1} \\ y_3^{k+1} \end{bmatrix} = \begin{bmatrix} y_1^{k} \\ y_2^{k} \\ y_3^{k} \end{bmatrix} + \sigma \begin{bmatrix} A_1\tilde{x}^{k+1} - b_1 \\ A_2\tilde{x}^{k+1} - b_2 \\ \tilde{x}^{k+1} - \mathrm{proj}_{\mathcal{K}}(\sigma^{-1}y_3^k + \tilde{x}^{k+1}) \end{bmatrix},$$

and

$$A^\top y^{k+1} = A^\top y^k + \sigma(A_1^\top(A_1(\tilde{x}^{k+1}) - b_1) + A_2^\top(A_2(\tilde{x}^{k+1}) - b_2) + \tilde{x}^{k+1} - \mathrm{proj}_{\mathcal{K}}(\sigma^{-1}y_3^k + \tilde{x}^{k+1})),$$

where the vector y in the dual domain can be written in the form $y = [y_1\ ,\ y_2\ ,\ y_3]^\top$ with $y_1, y_2 \in \mathbb{R}^p$ and $y_3 \in \mathbb{R}^{p^2}$ (see also (5)).

(Hint: Use Moreau's decomposition to write the update using prox_g instead of prox_{g^*}. In particular

$$y^{k+1} = \mathrm{prox}_{\sigma g^*}(y^k + \sigma A(\tilde{x}^{k+1})) = y^k + \sigma A(\tilde{x}^{k+1}) - \sigma \mathrm{prox}_{\sigma^{-1}g}(\sigma^{-1}y^k + A(\tilde{x}^{k+1}))$$

The remaining steps are to use (5) and find how to compute prox_g when g is in the decomposed form given in (6). End of Hint)

(e) (15 pts)

▹ Complete the missing lines in the function definitions of `HCGM` and `PDHG`, which implements Homotopy CGM and Vu-Condat algorithms, respectively. Run `HCGM` (5000-iterations) and `PDHG` (1000-iterations) to solve the k-means clustering problem.

Remarks:

- For simplicity, there is only one penalty parameter β_k in the HCGM Algorithm. However, in practice, one can have different penalty parameters for different constraints. In our case, we advise you to **multiply by 1000 the term**$(x_k - \mathbf{proj}_{\mathcal{K}}(x_k))$**)** in Algorithm 1, in order to obtain a better practical convergence. This basically corresponds to having different penalty parameters for different constraints.

- A similar observation applies tor the Vu-Condat algorithm: it is possible to use different dual step sizes $\{\sigma_1, \sigma_2, \ldots\}$. In our case, we advise you to **multiply the step-size for** y_3 **by** 10^4 to obtain a better practical convergence.

▹ Plot the convergence results of both algorithms using `plot_comp` function.

▹ What are the final objective values? Are they below the optimal value provided to you? If yes, explain the reason.

▹ Using the function `value_kmeans`, compute and report the k-means value before and after running both algorithms.

▹ Run the function `kmeans` a few times and report the k-means value obtained by Llyod's algorithm. Compare it with the ones obtained by rounding the solution of convex methods `HCGM` and `PDHG` (the rounding procedure is already given to you as the function `sdp_rounding`, you just need to run it). Comment on the result.

(Hint: Note that when $\mathcal{X}$ is as given in (3) and $\mathbf{X} \not\succeq 0$, $\kappa uu^\top \in \text{lmo}_\mathcal{X}(X)$, where u is the eigenvector corresponding to the smallest eigenvalue of X. Otherwise, if $\mathbf{X} \succeq 0$, then the LMO returns $\mathbf{0}$.)

(Hint: $\text{prox}_h(\cdot)$ step in the Vu-Condat algorithm reduces to projection onto the set $\mathcal{X}$ given in equation (3). Note also that $\text{proj}_\mathcal{X}(\boldsymbol{Z}) = \boldsymbol{U}\boldsymbol{\Sigma}^{\Delta_p}\boldsymbol{U}^\top$, where $\boldsymbol{U}$ is the matrix containing the eigenvectors of $\boldsymbol{Z}$ and $\boldsymbol{\Sigma}^{\Delta_p}$ is the diagonal matrix containing the projection of eigenvalues of $\boldsymbol{Z}$ to the simplex with radius κ.)

3 Hands-on experiment 3: Computing a geometric embedding for the Sparsest Cut Problem via Semidefinite Programming (20 pts)

The Uniform Sparsest Cut problem (USC) aims to find a bipartition $(S, \bar{S})$ of the nodes of a graph $G = (V, E)$, $|V| = p$, which minimizes the quantity

$$\frac{E(S, \bar{S})}{|S|\,|\bar{S}|},$$

where $E(S, \bar{S})$ is the number of edges connecting S and $\bar{S}$, and $|S|$ is the number of nodes in S. This problem is of broad interest, with applications in areas such as VLSI layout design, topological design of communication networks and image segmentation. Relevant to machine learning, it appears as a subproblem in hierarchical clustering algorithms [4, 3].

Computing such a bipartition is NP-hard and intense research has gone into designing efficient approximation algorithms for this problem. In the seminal work of [2] an $O(\sqrt{\log p})$ approximation algorithm is proposed for solving USC, which relies on finding a *well-spread* ℓ_2^2 geometric representation of G where each node $i \in V$ is mapped to a vector $\mathbf{v}_i$ in $\mathbb{R}^p$. In this experimental section we focus on solving the SDP that computes this geometric embedding.

The canonical formulation of the SDP is

$$\begin{aligned}
\boldsymbol{X}^\star \in \arg\min_{\boldsymbol{X}} \Big\{ \langle \boldsymbol{C}, \boldsymbol{X} \rangle : p\ \text{Tr}(\mathbf{X}) - \text{Tr}(\mathbf{1}_{p\times p}\mathbf{X}) = \frac{p^2}{2}, & \qquad \longleftarrow \quad \equiv A(\mathbf{X}) = \frac{p^2}{2} \\
\mathbf{X}_{i,j} + \mathbf{X}_{j,k} - \mathbf{X}_{i,k} - \mathbf{X}_{j,j} \le 0,\ \forall\, i \ne j \ne k \ne i \in V, & \qquad \longleftarrow \quad \equiv B_{i,j,k}(\mathbf{X}) \in \mathcal{K} = (-\infty, 0] \qquad (7) \\
\underbrace{\text{Tr}(\boldsymbol{X}) \le p,\ \boldsymbol{X} \in \mathbb{R}^{p\times p},\ \mathbf{X} \succeq 0}_{\mathcal{X}} \Big\}, & \qquad \longleftarrow \quad \mathbf{X} \in \mathcal{X} \text{ (the SDP cone of bounded trace)}
\end{aligned}$$

where $\boldsymbol{C}$ represents the Laplacian of graph G and $\mathbf{X}_{i,j} = \langle \mathbf{v}_i, \mathbf{v}_j \rangle$ gives the geometric embedding of the nodes.

We can rewrite the optimization problem (7) as

$$\min_{\mathbf{X}\in\mathcal{X}} f(\mathbf{X}) + g(A(\mathbf{X})) \qquad \text{subject to} \qquad B_{i,j,k}(\mathbf{X}) \in \mathcal{K},\ \forall\, i \ne j \ne k \ne i \in V, \tag{8}$$

where $f(\mathbf{X}) = \langle \boldsymbol{C}, \boldsymbol{X} \rangle$ and $g(\cdot) = \delta_{\left\{\frac{p^2}{2}\right\}}(\cdot)$ is the indicator function of singleton $\left\{\frac{p^2}{2}\right\}$.

(a) (5 pts) How many constraints does problem (7) have (as a function of p)? How does this number compare to the one of Problem (3)?

N.B.1: In Part 2 the constraints are expressed in matrix form, while here they are listed individually. Make sure to take this into account (e.g., the constraint $\mathbf{X} \ge 0$ in Part 2 is applied *for each* entry).

N.B.2: You can respond to this question by either computing the exact number of constraints, or by identifying the correct order of magnitude (big-O notation).

(b) (5 pts) Express the constraints in (8) in quadratic penalty form and write down the corresponding penalized objective function.

(c) (10 pts) We will now observe the behavior of HCGM on three graphs from the Network Repository dataset [8]:

- **G1:** `mammalia-primate-association-13` with 25 nodes,
- **G2:** `55n-insecta-ant-colony1-day37` with 55 nodes and
- **G3:** `insecta-ant-colony4-day10` with 102 nodes.

Based on your calculation in point a), give an estimate of the number of constraints for each dataset above.

We provide most of the code for solving this problem in file `Test_Sparsest_Cut.ipynb`. Fill in the few missing parts, run the algorithm for each dataset (you can cook your dinner in the meantime) and add the resulting plots to your report. What do you notice about the running times of the algorithm for the three problem instances? What are the potential bottlenecks to applying this method to large graphs?

One way to address the issues you identified above, especially if low accuracy suffices, is to resort to stochastic algorithms (the reasoning here is similar to the one which stands behind GD vs. SGD). Such an example are the methods proposed in [9], where the framework of HCGM is used in conjunction with stochastic gradients and variance reduction for alleviating some of the shortcomings of the full-batch method you implemented above. A brief presentation of these methods is provided in the supplementary section of Lecture 13.

4 Guidelines for the preparation and the submission of the homework

Work on your own. Do not copy or distribute your codes to other students in the class. Do not reuse any other code related to this homework. Here are few warnings and suggestions for you to prepare and submit your homework.

- This homework is due at 11:59, 24 December, 2021
- Submit your work before the due date. Late submissions are not allowed and you will get 0 point from this homework if you submit it after the deadline.
- Your final report should include detailed answers and it needs to be submitted in PDF format.
- The PDF file can be a scan or a photo. Make sure that it is eligible.
- The results of your simulations should be presented in the final report with clear explanation and comparison evaluation.
- We provide some Python scripts that you can use to implement the algorithms, but you can implement them from scratch using any other convenient programming tool (in this case, you should also write the codes to time your algorithm and to evaluate their efficiency by plotting necessary graphs).
- Even if you use the Python scripts that we provide, you are responsible for the entire code you submit. Apart from completing the missing parts in the scripts, you might need to change some written parts and parameters as well, depending on your implementations.
- The codes should be well-documented and should work properly. Make sure that your code runs without error. If the code you submit does not run, you will not be able to get any credits from the related exercises.
- Compress your codes and your final report into a single ZIP file, name it as `ee556_2020_hw3_NameSurname.zip`, and submit it through the moodle page of the course.

References

[1] Ahmed, A., Recht, B., and Romberg, J. Blind deconvolution using convex programming. *IEEE Transactions on Information Theory 60*, 3 (2014), 1711–1732.

[2] Arora, S., Rao, S., and Vazirani, U. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM (JACM) 56*, 2 (2009), 5.

[3] Chatziafratis, V., Niazadeh, R., and Charikar, M. Hierarchical clustering with structural constraints. *arXiv preprint arXiv:1805.09476* (2018).

[4] Dasgupta, S. A cost function for similarity-based hierarchical clustering. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing* (2016), pp. 118–127.

[5] JAGGI, M. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *Proc. 30th Int. Conf. Machine Learning* (2013).

[6] MIXON, D. G., VILLAR, S., AND WARD, R. Clustering subgaussian mixtures by semidefinite programming. *arXiv preprint arXiv:1602.06612* (2016).

[7] PENG, J., AND WEI, Y. Approximating k-means-type clustering via semidefinite programming. *SIAM journal on optimization 18*, 1 (2007), 186–205.

[8] ROSSI, R. A., AND AHMED, N. K. The network data repository with interactive graph analytics and visualization. In *AAAI* (2015).

[9] VLADAREAN, M.-L., ALACAOGLU, A., HSIEH, Y.-P., AND CEVHER, V. Conditional gradient methods for stochastically constrained convex minimization. In *International Conference on Machine Learning* (2020), PMLR, pp. 9775–9785.

[10] XIAO, H., RASUL, K., AND VOLLGRAF, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[11] YURTSEVER, A., FERCOQ, O., LOCATELLO, F., AND CEVHER, V. A conditional gradient framework for composite convex minimization with applications to semidefinite programming. *arXiv preprint arXiv:1804.08544* (2018).