(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 $\kappa$ 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:

$$\begin{cases} \text{Assignment step:} & \mathbf{s}_i \text{ belongs to cluster } j, \text{ where } j := \operatorname{argmin}_{j \in [1,k]} \|\mathbf{s}_i - \mathbf{c}_j\| \\ \text{Update each cluster center:} & \mathbf{c}_j = \frac{1}{|S_j|} \sum_{i=1}^{|S_j|} \mathbf{s}_i^j \end{cases}$$

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

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

where $C \in \mathbb{R}^{p \times p}$ is the Euclidean distance matrix between the data points. $\operatorname{Tr}(\boldsymbol{X}) \leq \kappa$ 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 $\operatorname{Tr}(\boldsymbol{X}) = \|\boldsymbol{X}\|_*$ 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:

$$\begin{cases} \gamma_k & = 2/(k+1), \text{ and } \beta_k = \beta_0/\sqrt{k+1} \\ v_k & = \beta_k \nabla f(x_k) + A_1^\top (A_1(x_k) - b_1) + A_2^\top (A_2(x_k) - b_2) + (x_k - \text{proj}_{\mathcal{K}}(x_k)) \\ \hat{x}^k & := \text{argmin}_{x \in \mathcal{X}} \langle v_k, x \rangle \\ x^{k+1} & := (1-\gamma_k) x^k + \gamma_k \hat{x}^k \end{cases}$$

**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:

$$\begin{cases} x^{k+1} & = \text{prox}_{\tau h}(x^k - \tau(\nabla f(x^k) + A^\top y^k)) \\ \bar{x}^{k+1} & = 2x^{k+1} - x^k \\ y^{k+1} & = \text{prox}_{\sigma g^*}(y^k + \sigma A(\bar{x}^{k+1})) \end{cases}$$

**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 $\mathcal{X} = \{\boldsymbol{X} : \text{Tr}(\boldsymbol{X}) \leq \kappa, \ \boldsymbol{X} \in \mathbb{C}^{p \times p}, \ \mathbf{X} \succeq 0\}$ 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

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

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

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

and **show that** the gradient of the penalized objective is equal to $v_k/\beta$ in the algorithm.
(Hint: You can write $\text{dist}^2(Tx, \mathcal{Y}) = \|y^* - Tx\|^2$, 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 $\mathbf{C}$.

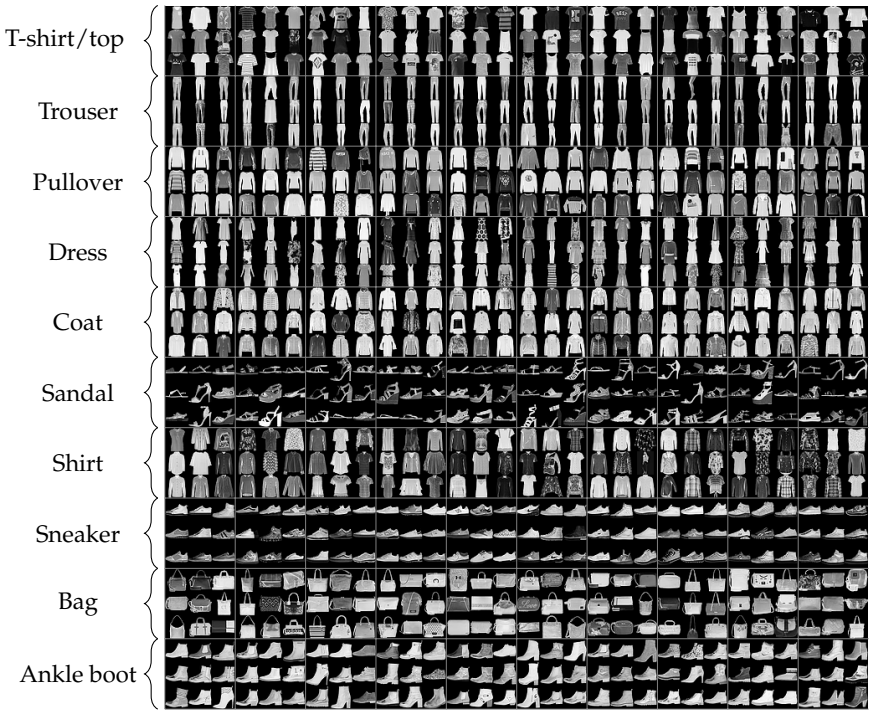


Figure 1: Fashion MNIST data

(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 $\mathrm{prox}_g$ instead of $\mathrm{prox}_{g^*}$. 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 $\mathrm{prox}_g$ 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 $\beta_k$ 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 $\mathbf{X} \not\succeq 0$, $\kappa uu^\top \in \mathrm{lmo}_{\mathcal{X}}(X)$, 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: $\mathrm{prox}_h(\cdot)$ step in the Vu-Condat algorithm reduces to projection onto the set $\mathcal{X}$ given in equation (3). Note also that $\mathrm{proj}_{\mathcal{X}}(\boldsymbol{Z}) = \boldsymbol{U}\boldsymbol{\Sigma}^{\Delta_p}\boldsymbol{U}^\top$, 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 $\kappa$.)

## 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* $\ell_2^2$ 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

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

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

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

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 $\mathbf{X} \succeq 0$ 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).