Homework 2
Large-scale optimization

Jens Sjölund Sebastian Mair

Deadline: October 18th, 2023 at 13.15

Problem 4.2

Verify the formula

$$f(x^k) = \frac{1}{2}\mathbb{E}_{\omega_1,\ldots,\omega_k,\omega}\left[\left(\frac{1}{k}\sum_{j=1}^{k}\omega_j - \omega\right)^2\right] = \frac{1}{2k}\sigma^2 + \frac{1}{2}\sigma^2, \tag{5.14}$$

given that the mean of the random variable ω is μ and its variance is σ^2. Note that the random variables $\omega_1, \ldots, \omega_k, \omega$ all follow the same distribution, and all random variables in this expression are independent.

Problem: Sketching

Let $A \in \mathbb{R}^{N\times n}$ and $b \in \mathbb{R}^N$. Consider the problem

$$\min_x \; f(x) := \frac{1}{N}\|Ax - b\|^2 \tag{0.1}$$

which can be naively solved via the normal equations $A^\mathsf{T}Ax = A^\mathsf{T}b$.

Test the sketching idea for the problem outlined above on (preprocessed) song data[1]. Choose an appropriate sketching approach and use various sketch sizes to analyze and plot (i) $f(x)$, (ii) $f(x_{\text{sketch}})$, (iii) $\|x - x_{\text{sketch}}\|$, and (iv) the time to sketch and solve the sketched problem as functions of the sketch size. Also compare against (v) the time to solve the original problem.

Note: You might run into the problems applying Gaussian and Rademacher sketch matrices. Consider using CountSketch[2].

Problem: Portfolio optimization[3]

Background. Suppose that you are working in finance and that you are faced with $n = 15000$ assets, such as stocks, that you could invest in. The assets have mean return $\mu \in \mathbb{R}^n$ and return covariance $\Sigma \in \mathbb{S}^n_{++}$ (i.e. a symmetric and strictly positive definite $n \times n$ matrix).

[1]The file `song_preprocessed.npz` can be loaded via `numpy` and contains A and b. The file can be found here: https://uppsala.box.com/s/dckj9eux29pvhglted8n2q78j3zxvcjg

[2]https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.clarkson_woodruff_transform.html

[3]Modification of one of the additional exercises for Stanford EE364a, Convex Optimization, that do not appear in Boyd and Vandenberghe 2004.

Problem description. Your task is to create a normalized portfolio $w \in \mathbb{R}^n$ (with negative entries meaning short positions) that balances the risk of the portfolio against the expected return by solving the following portfolio optimization problem:

$$\begin{array}{ll} \underset{w,y}{\text{minimize}} & \frac{1}{2} w^\mathsf{T} D w + \frac{1}{2} y^\mathsf{T} Q y - \mu^\mathsf{T} w \\ \text{subject to} & \mathbf{1}^\mathsf{T} w = 1, \quad F^\mathsf{T} w = y, \end{array}$$

where $y = F^\mathsf{T} w$ represents the factor exposures for the portfolio w.

We consider a so-called "k-factor risk model", which means that the return covariance has the form $\Sigma = FQF^\mathsf{T} + D$, where $F \in \mathbb{R}^{n \times k}$ is the factor loading matrix, $Q \in \mathbb{S}^k_{++}$ is the factor covariance matrix, and D is a diagonal matrix with positive entries. We assume that there are $k = 30$ factors.

The solution of this constrained optimization problem can be found by solving the corresponding KKT system (you don't have to understand why, we will get there later in the course):

$$\begin{pmatrix} D & 0 & \mathbf{1} & F \\ 0 & Q & 0 & -I \\ \mathbf{1}^\mathsf{T} & 0 & 0 & 0 \\ F^\mathsf{T} & -I & 0 & 0 \end{pmatrix} \begin{pmatrix} w \\ y \\ \nu \\ \kappa \end{pmatrix} = \begin{pmatrix} \mu \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

and extracting the corresponding w.

However, your competitors are using the same model, and if they can find the solution faster, they will invest before you and drive up the prices (at your expense) and vice versa. During lunch, you overheard a conversation between peers at a competing firm who said their code was easily 100x faster than the naïve baseline. The question is: can you beat them?

Task. Start from the code provided in Appendix A, which creates the problem data and solves the KKT system in the naïve, slow, way. Your task is to use block elimination to implement a more efficient solver. You are not allowed to make any additional imports. If done correctly, your method should be at least 10 times faster than the naïve baseline.

Problem: MIMO Channel Estimation

In wireless communications, the amount of data that can be transmitted over a radio link can be substantially increased by using multiple transmitting and receiving antennas. Such systems, generally referred to as multiple-input and multiple-output (MIMO), are an essential element of modern standards for wireless communication.

In its simplest form, MIMO communication can be represented mathematically as a vector-valued signal $x \in \mathbb{R}^n$ transmitted over a linear channel that is represented by the channel matrix $H \in \mathbb{R}^{m \times n}$ and corrupted by independent and identically distributed Gaussian noise ϵ, such that the received vector $y \in \mathbb{R}^m$ is given by

$$y = Hx + \epsilon.$$

(We ignore the fact that, in reality, these entities would all be complex-valued.) The channel H captures a range of phenomena including interference, fading, and reflections. In most cases it is therefore impossible to find an accurate analytical expression for it. Instead, the channel matrix is often estimated by transmitting a set of k known pilot signals $P = \left(p_1, \ldots, p_k\right)$, recording the received signals $Y = \left(y_1, \ldots, y_k\right)$, and solving the optimization problem

$$H^* = \underset{H}{\arg\min} \, \|HP - Y\|_F^2. \tag{1}$$

(a) Show that each row of the channel matrix can be determined independently of the others by solving a least-squares problem.

(b) Suppose that $m = n$, $k = 5n$, and that P is a dense, unstructured, matrix of full rank. Show that the channel estimation problem (1) can be solved with $\mathcal{O}(n^3)$ flops using a Cholesky-based factor-solve method.

(c) Implement the method from part (b) using the functions `cholesky` and `solve_triangular` imported from `scipy.linalg` and use it to solve the problem generated by the code in Appendix B.

Submission instructions

Please submit a **single pdf document** that includes all *detailed derivations*, *justifications*, *code*, and *result figures*. Your solutions can be written on a computer, e.g., using LaTeX, or be handwritten, as long as they are readable.

All code snippets provided by us are written in Python. You can choose to implement the task in another language but you have to take care to port the given code yourself. Random states can then be neglected.

Your submission will be peer-reviewed. To pass the review, you should have **at least 70% correct**. Your reviewer will then comment on your submission and rate it according as *strong accept*, *accept*, *borderline accept*, *borderline reject*, or *reject* together with a justification.

References

Boyd, Stephen P and Lieven Vandenberghe (2004). *Convex optimization.* Cambridge university press.

A Code for Portfolio Optimization

```
import numpy as np
import scipy.sparse as sps
import scipy.sparse.linalg as splinalg
import time

# generate some random data
n = 15000 # number of assets
k = 30 # number of factors
np.random.seed(0)
F = np.random.randn(n,k)
F = np.matrix(F)
d = 0.1 + np.random.rand(n)
d = np.matrix(d).T
Q = np.random.randn(k)
Q = np.matrix(Q).T
Q = Q * Q.T + np.eye(k)
Sigma = np.diag(d.A1) + F*Q*F.T
mu = np.random.rand(n)
mu = np.matrix(mu).T

# the slow way, solve full KKT
t = time.time()
kkt_matrix = np.vstack((np.hstack((Sigma, np.ones((n,1)))),
                        np.hstack((np.ones(n), [0.]))))
wnu = np.linalg.solve(kkt_matrix, np.vstack((mu, [1.])))
print("Elapsed time for naive method is %f seconds." % (time.time() - t))
wslow = wnu[:n]

# fast method
t = time.time()

YOUR CODE HERE (return wfast)

print("Elapsed time is %f seconds." % (time.time() - t))
rel_err = np.sqrt(np.sum((wfast-wslow).A1**2)/np.sum(wslow.A1**2))
print(rel_err)
```

B Code for MIMO Channel Estimation

```
import numpy as np
from scipy.linalg import cholesky, solve_triangular
import scipy.sparse

# Create synthetic data
np.random.seed(0)
n = 5000
m = n
k = 5 * n

H = np.random.randn(m, n)
P = scipy.sparse.random(n, k, density=0.01).toarray()
Y = H @ P + np.random.randn(m, k)

assert np.linalg.matrix_rank(P) == n
```