Assignment 2
Large-scale Optimization FTN0452

Li Ju

October 5, 2023

1 Problem 4.2

Given that $\mathbb{E}[\omega] = \mathbb{E}[\omega_i] = \mu$ and $\mathbb{E}[(\omega-\mu)^2] = \mathbb{E}[(\omega_i-\mu)^2] = \sigma^2$ for all $i \in [k]$, we have

$$
\begin{aligned}
f(x^k) &= \frac{1}{2}\mathbb{E}_{\omega_1,\dots,\omega_k,\omega}\left[\left(\frac{1}{k}\sum_{i=1}^{k}\omega_i - \omega\right)^2\right] \\
&= \frac{1}{2}\left\{\frac{1}{k^2}\mathbb{E}_{\omega_1,\dots,\omega_k}\left[\left(\sum_{i=1}^{k}\omega_i\right)^2\right] - \frac{2}{k}\mathbb{E}_{\omega_1,\dots,\omega_k}\left[\sum_{i=1}^{k}\omega_i\right]\mathbb{E}_{\omega}\left[\omega\right] + \mathbb{E}_{\omega}\left[\omega^2\right]\right\} \\
&= \frac{1}{2}\left\{\frac{1}{k^2}\sum_{\substack{i,j\in[k]\\ i\neq j}}\mathbb{E}_{\omega_i}[\omega_i]\mathbb{E}_{\omega_j}[\omega_j] + \frac{1}{k^2}\sum_{i=1}^{k}\mathbb{E}_{\omega_i}[\omega_i^2] - \frac{2\mu}{k}\sum_{i=1}^{k}\mathbb{E}_{\omega_i}\omega_i + \sigma^2 + \mu^2\right\} \\
&= \frac{1}{2}\left(\frac{(k^2-k)\mu^2 + k(\mu^2+\sigma^2)}{k^2} - 2\mu^2 + \sigma^2 + \mu^2\right) \\
&= \frac{\sigma^2}{2k} + \frac{\sigma^2}{2}
\end{aligned}
$$

2 Problem: Sketching

Function `scipy.linalg.clarkson_woodruff_transform` is utilized in our implementation. The code is deferred to Appendix A.1. Here we denote the solution obtained from the sketching method with sketching size m as $\hat{x}_m$, and the exact solution obtained from the original method as $x^\star$.

Experimental results are plotted in Figure 1. We have following observations:

- In terms of function values, with the increase of sketching size from $m = 10^3$ to $m = 10^5$, $f(\hat{x}_m)$ decreases significantly, approaching $f(x^\star)$.
- With the increase of $m = 10^3$ to $m = 10^5$, time cost for solving the problem with sketching method increase from 0.072s to 0.168s. Compared with the time cost by the original method, we gain significant speedup.
- Compared with the exact solution $x^\star$, $\|\hat{x}_x - x^\star\|$ decreases with an increase of sketching size m.

Essentially, matrix sketching provides a trade-off between the accuracy and the computation budget with different sketching sizes.

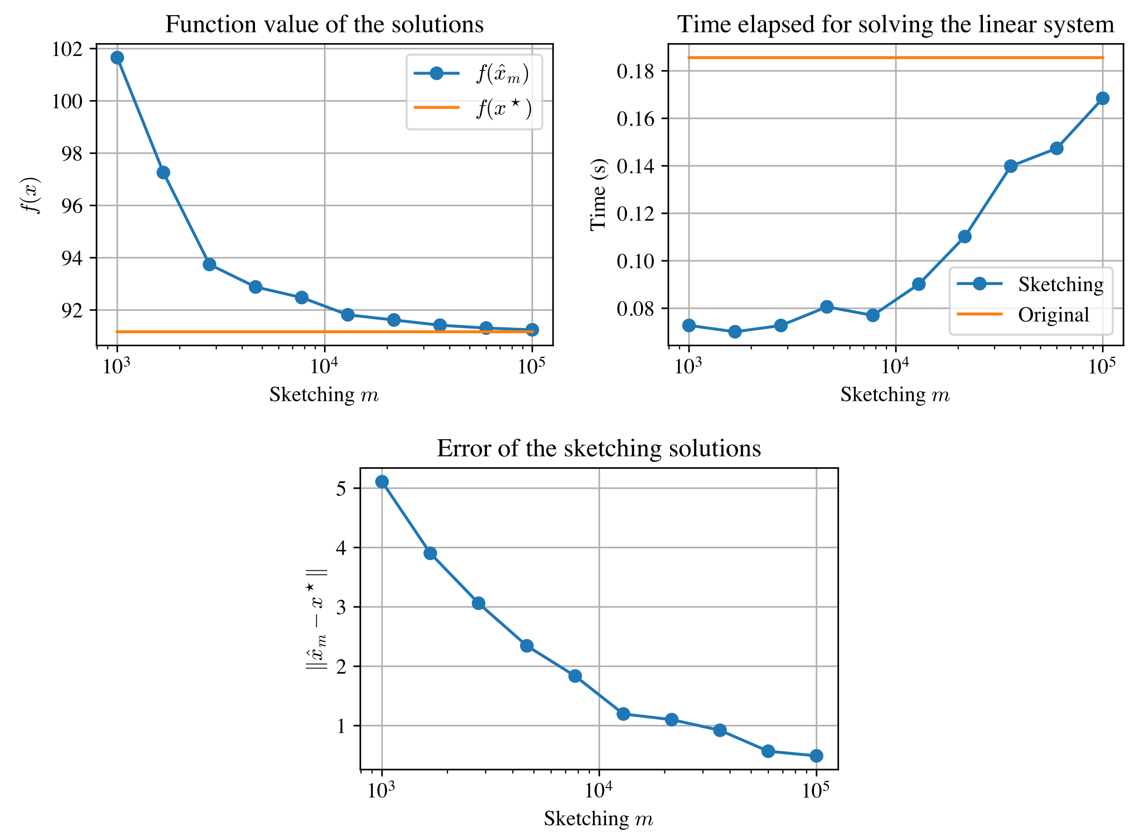

Figure 1: Performance of the sketching method: Function values, time elapsed for solving the problem and error of the sketching solutions are compared with different sketching size m.

3 Problem: Portfolio Optimization

To solve the KKT system in a more efficient way, we use block elimination, taking full advantages of the structure of the system.

Derivation Let

$$\underset{(n+k)\times(n+k)}{A_{11}} = \begin{pmatrix} D & 0 \\ 0 & Q \end{pmatrix} \qquad \underset{(n+k)\times(k+1)}{A_{12}} = \begin{pmatrix} \mathbf{1} & F \\ 0 & -I \end{pmatrix} \qquad \underset{(k+1)\times(n+k)}{A_{21}} = \begin{pmatrix} \mathbf{1}^\top & 0 \\ F^\top & -I \end{pmatrix} \qquad \underset{(k+1)\times(k+1)}{A_{22}} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

and

$$\underset{(n+k)\times 1}{x_1} = \begin{pmatrix} w \\ y \end{pmatrix} \qquad \underset{(k+1)\times 1}{x_2} = \begin{pmatrix} \nu \\ \kappa \end{pmatrix} \qquad \underset{(n+k)\times 1}{b_1} = \begin{pmatrix} \mu \\ 0 \end{pmatrix} \qquad \underset{(k+1)\times 1}{b_2} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

We have $A_{11}^{-1} = \begin{pmatrix} D^{-1} & 0 \\ 0 & Q^{-1} \end{pmatrix}$. Then we have

$$\underset{(n+k)\times(k+1)}{A_{11}^{-1}A_{12}} = \begin{pmatrix} D^{-1}\mathbf{1} & D^{-1}F \\ 0 & -Q^{-1} \end{pmatrix} \qquad \underset{(n+k)\times 1}{A_{11}^{-1}b^1} = \begin{pmatrix} D^{-1}\mu \\ 0 \end{pmatrix}$$

Then we derive Schur complements $S := A_{22} - A_{21}A_{11}^{-1}A_{12}$ and $\tilde{d} := b_2 - A_{21}A_{11}^{-1}b_1$:

$$\underset{(k+1)\times(k+1)}{S} = -A_{21}A_{11}^{-1}A_{12} = -\begin{pmatrix} \mathbf{1}^\top D^{-1}\mathbf{1} & \mathbf{1}^\top D^{-1}F \\ F^\top D^{-1}\mathbf{1} & F^\top D^{-1}F + Q^{-1} \end{pmatrix}$$

$$\underset{(k+1)\times 1}{\tilde{b}} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} - \begin{pmatrix} \mathbf{1}^\top & 0 \\ F^\top & -I \end{pmatrix} \begin{pmatrix} D^{-1}\mu \\ 0 \end{pmatrix} = \begin{pmatrix} 1 - \mathbf{1}^\top D^{-1}\mu \\ -F^\top D^{-1}\mu \end{pmatrix}$$

Then we have $x_2 = S^{-1}\tilde{b}$ and $x_1 = A_{11}^{-1}(b_1 - A_{12}x_2)$.

Computation Since we have analytical form of S and $\tilde{b}$, we can compute them directly. There are tricks to reduce computation cost:

- Computing D^{-1}: Taking advantages of the diagonality of D, the computation cost of D^{-1} can be reduced from $\mathcal{O}(n^3)$ to $\mathcal{O}(n)$.
- Computing $D^{-1}\mathbf{1}, D^{-1}F$ and $D^{-1}\mu$: Since D^{-1} is diagonal, the computation cost of $\underset{n\times n}{D^{-1}}\underset{n\times m}{M}$ can be reduced from $\mathcal{O}(n^2m)$ to $\mathcal{O}(nm)$.

It could be further optimized to compute Q^{-1} and S^{-1} using Cholesky decomposition, taking advantages of their positive-definite-ness. However, by timing the executing time, it is observed that computing S is the bottleneck of the performance (with given Q^{-1}), but not inverting Q and S. It is not worthy to optimize the non-dominating part of the code.

The implementation of the optimized method is deferred to Appendix A.2. The optimized code reduces the executing time from 12.4213 ± 0.3207 seconds to 0.0051 ± 0.0004 seconds, with a speedup of $\times 2400$.

4 Problem: MIMO Channel Estimation

Notation For matrix M, we denote the ith row and the jth column vector of M as $M_{i,*}^\top$ and $M_{*,j}$, respectively. The element at ith row and jth column of M is denoted as $M_{i,j}$. For vector v, we denote the ith element as v_i.

4.1 a).

The optimization problem can be restructured as

$$\begin{aligned}
H^\star = \begin{pmatrix} H_{1,*}^{\star\top} \\ \vdots \\ H_{m,*}^{\star\top} \end{pmatrix} &= \arg\min_H \|HP - Y\|_F^2 = \arg\min_H \|P^\top H^\top - Y^\top\|_F^2 \\
&= \arg\min_H \sum_{i=1}^m \sum_{j=1}^k \begin{pmatrix} P^\top H_{1,*} - Y_{1,*} \\ \vdots \\ P^\top H_{m,*} - Y_{m,*} \end{pmatrix}_{i,j}^2 \\
&= \begin{pmatrix} \left[\arg\min_{H_{1,*}} \sum_{j=1}^k (P^\top H_{1,*} - Y_{1,*})_j^2\right]^\top \\ \vdots \\ \left[\arg\min_{H_{m,*}} \sum_{j=1}^k (P^\top H_{1,*} - Y_{m,*})_j^2\right]^\top \end{pmatrix} \\
&= \begin{pmatrix} \left[\arg\min_{H_{1,*}} \|P^\top H_{1,*} - Y_{1,*}\|^2\right]^\top \\ \vdots \\ \left[\arg\min_{H_{m,*}} \|P^\top H_{m,*} - Y_{m,*}\|^2\right]^\top \end{pmatrix}
\end{aligned}$$

Thus, the ith row of $H^\star$ can be independently obtained by solving the least-square problem $H^\star_{i,*} = \arg\min_{H_{i,*}} \|P^\top H_{i,*} - Y_{i,*}\|^2$.

4.2 b).

With $m = n$ and $k = 5n$, the least square problem of each row is an over-determined system, and the solution is given by the linear system $PP^\top H^\star_{i,*} = PY_{i,*}, \forall i \in [m]$. Given that P is a dense matrix with full rank, we recognize that $PP^\top$ is positive definite. Thus we having following algorithm:

Algorithm 1 Pseudo code for Problem MIMO channel estimation

Require: P, Y
  Compute $A = PP^\top$ ▷ flops: $5n^3$
  Factorize $A = LL^\top$ with Cholesky decomposition ▷ flops: $1/3n^3$
  while $i \in [m]$ **do**
    Forward substitution: Solve $Lz_1 = PY_{i,*}$ ▷ flops: $n^2 + 5n^2$
    Forward substitution: Solve $L^\top H^\star_{i,*} = z_i$ ▷ flops: n^2
  end while ▷ total flops: $7n^3$
  $H^\star = (H^\star_{1,*}, \cdots, H^\star_{m,*})^\top$

The problem thus can be solved within $12.34n^3$, i.e. $\mathcal{O}(n^3)$, flops.

4.3 c).

The implementation of Algorithm 1 is deferred to Appendix A.3.

References

A Code

A.1 Sketching

```
import numpy as np
import time
from scipy.linalg import clarkson_woodruff_transform
import matplotlib.pyplot as plt


# load data
data = np.load("as2/song_preprocessed.npz")
A, b = data["A"], data["b"]


def get_value(A, b, x):
    return 1/b.shape[0] * np.linalg.norm(A @ x - b)**2


# 1. solve the linear system with original method
def original_solve(A, b):
    t = time.time()
    # solve the linear system
    ATA = A.T @ A
    ATb = A.T @ b
    x = np.linalg.solve(ATA, ATb)
    ori_time = time.time() - t
    return x, ori_time


# 2. solve the linear system with sketching method
def sketch_solve(A, b, sketch_size):
    t = time.time()
    # generate sketching matrix
    SAb = clarkson_woodruff_transform(
        np.hstack((A, b.reshape(-1, 1))),
        sketch_size)
    SA, Sb = SAb[:, :-1], SAb[:, -1]

    # solve the linear system
    SATSA = SA.T @ SA
    SATb = SA.T @ Sb
    x_sketch = np.linalg.solve(SATSA, SATb)
    sketch_time = time.time() - t
    return x_sketch, sketch_time


# 3. run experiments
x, ori_time = original_solve(A, b)

results = {}

sketch_sizes = [int(np.power(10, i)) for i in np.linspace(3, 5, 10)]
print(sketch_sizes)
```

```python
for sketch_size in sketch_sizes:
    x_sketch, sketch_time = sketch_solve(A, b, sketch_size)
    results[sketch_size] = {
        "value": get_value(A, b, x_sketch),
        "time": sketch_time,
        "error": np.linalg.norm(x - x_sketch)
    }

# 4. plot the results
# 1). function value: compare original and sketching of different sizes
plt.figure()
plt.plot(sketch_sizes, [results[sketch_size]["value"]
                        for sketch_size in sketch_sizes],
         '-o',
         label=r"$f(\hat{x}_m)$")
plt.plot(sketch_sizes, [get_value(A, b, x)] * len(sketch_sizes),
         label=r"$f(x^\star)$")
plt.xlabel(r"$\log m$")
plt.ylabel(r"$f(x)$")
plt.title("Function value of the solutions")
plt.xscale("log")
plt.legend()
plt.savefig("as2/sketch_value.pdf", dpi=300)
plt.close()

# 2). time: compare original and sketching of different sizes
plt.figure()
plt.plot(sketch_sizes, [results[sketch_size]["time"]
                        for sketch_size in sketch_sizes],
         '-o',
         label="Sketching")
plt.plot(sketch_sizes, [ori_time] * len(sketch_sizes),
         label="Original")
plt.xlabel(r"$\log m$")
plt.ylabel("Time (s)")
plt.title("Time elapsed for solving the linear system")
plt.xscale("log")
plt.legend()
plt.savefig("as2/sketch_time.pdf", dpi=300)
plt.close()

# 3). error: compare original and sketching of different sizes
plt.figure()
plt.plot(sketch_sizes, [results[sketch_size]["error"]
                        for sketch_size in sketch_sizes],
         '-o')
plt.xlabel(r"$\log m$")
plt.ylabel(r"$\| \hat{x}_m - x^\star \|$")
plt.title("Error of the sketching solutions")
plt.xscale("log")
plt.savefig("as2/sketch_error.pdf", dpi=300)
plt.close()
```

A.2 Portfolio Optimization

```python
import numpy as np
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(f"Elapsed time for naive method is {(time.time() - t)} seconds.")
wslow = wnu[:n]

# fast method: solve the linear system with block matrix
t = time.time()

dinv = np.asarray(1./d)
Qinv = np.linalg.inv(Q)

# 1. formulation step:
dinvF = np.multiply(dinv, F)
dinvmu = np.multiply(dinv, mu)

# Compuet S and £\tilde{d}£
s11 = np.array([[dinv.sum()]])
s12 = np.ones((1, n))@dinvF
s21 = s12.T
s22 = F.T @ dinvF + Qinv


S = -np.vstack((np.hstack((s11, s12)),
                np.hstack((s21, s22))))

# Compute £\tilde{b}£
tilde_b = np.concatenate(
    (
```

```
        np.array([[1 - dinvmu.sum()]]),
        -F.T@dinvmu,
    )
)

# 2. solving step:

# Solve the linear system
# 1). solve Sx_2 = \tilde{b}
x_2 = np.linalg.solve(S, tilde_b)

# 2). solve x1 (i.e. (w, y) ) using x2
b1 = np.concatenate((mu, np.zeros((k, 1))))

A12 = np.vstack(
    (np.hstack((np.ones((n, 1)), F)),
     np.hstack((np.zeros((k, 1)), -np.eye(k))))
)

right_hand_side = b1 - A12@x_2

wfast = np.multiply(dinv, right_hand_side[:n])

print(f"Time for solving is {(time.time() - t)} seconds.")
rel_err = np.sqrt(np.sum((wfast-wslow).A1**2)/np.sum(wslow.A1**2))
print(f"Error: {rel_err}")
```

A.3 MIMO channel estimation

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

t = time.time()
# 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

print(f"Time elapsed for generating: {time.time() - t}s")

t = time.time()
PT = P.T.copy()
A = P @ PT

# factorize A with cholesky decomposition
L = cholesky(A, lower=True)
```

```python
# solve the linear systems
Z = solve_triangular(
    L, P@y.T, lower=True, check_finite=False)
H_star = solve_triangular(
    L.T, Z, lower=False, check_finite=False).T

print(f"Time elapsed for solving: {time.time() - t}s")

# # use naive method to compute the residual
# H_naive = np.linalg.solve(A, P@y.T).T

# # compute the residual
# error = np.linalg.norm(H_star.flatten() - H_naive.flatten())
# print(f"Error: {error}")
```