HPC for numerical methods and data analysis

Fall Semester 2023

Prof. Laura Grigori

Assistant: Mariana Martinez

Session 7 – October 31, 2023

Randomized SVD

Exercise 1: SRHT

In the context of overdetermined least-squares problems, we need to find $x \in \mathbb{R}^n$ such that it minimizes:

$$\|Wx - b\|_2^2,$$

where $W \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, m > n$. There is a class of randomized algorithms for solving this problem based on sketching method. Sketching methods involve using a random matrix $\Omega \in \mathbb{R}^{l \times m}$ to project the data W (and maybe also b) to a lower dimensional space with $l \ll m$. Then they approximately solve the least-squares problem using the sketch ΩW (and/or Ωb). One relaxes the problem to finding a vector x so that

$$\|Wx - b\| \leq (1 + \varepsilon)\|Wx^* - b\|,$$

where x^* is the optimal solution. The overview of sketching applied to solve linear least squares is:

a) Sample/build a random matrix Ω

b) Compute ΩA and Ωb

c) Output the exact solution to the problem $\min_x \|(\Omega W)x - (\Omega)b\|_2$.

Given a data matrix, $W \in \mathbb{R}^{m \times n}$, we want to reduce the dimensionality of W by defining a random orthonormal matrix $\Omega \in \mathbb{R}^{l \times m}$ with $l \ll m$. For $m = 2^q, q \in \mathbb{N}$, the Subsampled Randomized Hadamard Transform (SRHT) algorithm defined a $l \times m$ matrix as:

$$\Omega = \sqrt{\frac{m}{l}} P H_m D,$$

where:

- $D \in \mathbb{R}^{m \times m}$ is a diagonal matrix whose elements are independent random signs, i.e. it's diagonal entries are just -1 or 1.

- $H \in \mathbb{R}^{m\times m}$ is a **normalized** Walsh-Hadamard matrix. If you're going to use a library that implements this transform then check that it implements the normalized Walsh-Hadamard matrix. This matrix is defined recursively as:

$$H_m = \begin{bmatrix} H_{m/2} & H_{m/2} \\ H_{m/2} & -H_{m/2} \end{bmatrix} \qquad\qquad H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$
$$H = \frac{1}{\sqrt{m}} H_m \in \mathbb{R}^{m\times m}.$$

- $P \in \mathbb{R}^{l\times m}$ is a subset of randomly sampled l columns from the $m \times m$ identity matrix. The purpose of using P is to uniformly sample r columns from the rotated data matrix $X_{\text{rot}} = H_m D X$.

The following theorem help us get an idea for the size of l.

Theorem 1 (Subsampled Randomized Hadamard Transform) *Let $\Omega = \sqrt{\frac{m}{l}} P H_m D$ as previously defined. Then if*

$$l \geq \mathcal{O}((\varepsilon^{-2}\log(n))(\sqrt{n} + \sqrt{\log m})^2)$$

with probability $0,99$ for any fixed $U \in \mathbb{R}^{m\times n}$ with orthonormal columns:

$$\|I - U^\top \Omega \Omega^\top U\|_2 \leq \varepsilon.$$

Further, for any vector $x \in \mathbb{R}^m, \Omega x$ can be computed in $\mathcal{O}(n \log l)$ time.

Choose a data set from [https://www.kaggle.com/datasets?tags=13405-Linear+Regression]. Compare the randomized least squares fit using SRHT vs the deterministic least squares fit. Use the previous theorem to estimate l. *Hint: you can use the fast Hadamard transform from scipy or pytorch*

A Python script with the solutions is found below:

```
import numpy as np
from numpy.linalg import norm, lstsq
from pandas import read_csv
from numpy.random import normal
from math import ceil, log, sqrt, floor
import matplotlib.pyplot as plt
import time
from random import sample
import random
import torch
from hadamard_transform import hadamard_transform

plt.ion()

# For SRHT sketching applied to a least squares problem
# we report the following quantities:
##### Time taken to solve the full problem
##### Time taken to solve the compressed problem
##### Residual norm full problem
##### Residual norm compressed problem
##### Relative error in the spectral norm
```

```
# We are going to read the data (which was previously downloaded)
# We just want to work with certain columns, not all of them
d = read_csv("ParisHousing.csv")
b = d.price
b = b.values
d.drop(['hasYard', 'hasPool', 'floors', 'cityCode', 'numPrevOwners',
        'made', 'basement', 'attic', 'garage', 'hasGuestRoom'], axis = 1)
A = d.values
# But we need to make sure m is a power of 2
m = int(2**(floor(log(A.shape[0])/log(2))))
A = A[0:m, :]
b = b[0:m]

# Now that we have out set up
n = A.shape[1]
nRuns = 10
sigma = 0.99
epsilon = np.array([100, 10, 5, 2, 1, 0.5, 0.1])
rVec = np.ceil( (log(n)/(epsilon**2))*(sqrt(n) + log(m))**2).astype('int')
# Notice that some r's might be bigger than m

timeF = np.empty_like(epsilon)
timeC = np.empty_like(epsilon)
resF = np.empty_like(epsilon)
resC = np.empty_like(epsilon)
relErrSpec = np.empty_like(epsilon)

for k in range(len(epsilon)):
    eps = epsilon[k]
    r = min(m, rVec[k])
    tF = 0
    tC = 0
    rC = 0
    rES = 0
    for run in range(nRuns):
        # Begin with the compressed problem
        ts = time.time()
        d = np.array([1 if random.random() < 0.5 else -1 for i in range(m)])
        D = np.diag(sqrt(m/r)*d)
        P = sample(range(m), r)
        omega = D
        omega = np.array([ hadamard_transform(torch.from_numpy(omega[:, i])).numpy() for i in range(m
        omega = np.transpose(omega)
        omega = omega[P, :]
        omegaA = omega@A
        omegab = omega@b
        xPrime = lstsq(omegaA, omegab)
        xPrime = xPrime[0]
        tC += time.time() - ts
        # Now for the full problem
        ts = time.time()
        xStar = lstsq(A, b)
        xStar = xStar[0]
        tF += time.time() - ts
        # Report desired quantities for the randomized part
        rC += norm(omegaA@xPrime - omegab)
        rES += abs(norm(omegaA) - norm(A))/norm(A)
```

```
    # Save averages
    timeF[k] = tF/nRuns
    timeC[k] = tC/nRuns
    resF[k] = norm(A@xStar - b)
    resC[k] = rC/nRuns
    relErrSpec[k] = rES/nRuns


###
### Plot plot plot
# Time
plt.figure(figsize=(8, 6), dpi=80)
plt.loglog(epsilon, timeF, c = "#003aff", marker = 'o',
           label = "Full problem")
plt.loglog(epsilon, timeC, c = "#00b310", marker = '*',
           label = "Compressed problem")
plt.legend()
plt.title(r'$\varepsilon$' +
          ", time taken to build and compute")
plt.xlabel(r'$\varepsilon$')
plt.ylabel("Time, s")

# Norm of residual
plt.figure(figsize=(8, 6), dpi=80)
plt.loglog(epsilon, resF, c = "#003aff", marker = 'o',
           label = "Full problem")
plt.loglog(epsilon, resC, c = "#00b310", marker = '*',
           label = "Compressed problem")
plt.legend()
plt.title(r'$\varepsilon$' + ", norm of residual")
plt.xlabel(r'$\varepsilon$')
plt.ylabel("Norm of residual")

# Relative error in spectral norm
plt.figure(figsize=(8, 6), dpi=80)
plt.loglog(epsilon, relErrSpec, c = "#5400b3", marker = 'o',
           label = "Relative error")
plt.loglog(epsilon, epsilon, c = '#676b74', linestyle='dashed',
           label = r'$\varepsilon$')
plt.legend()
plt.title(r'$\varepsilon$' + ", relative error spectral norm " +
          r'$| \|\Omega A\|_2 - \|A\|_2 |/\| A\|_2$')
plt.xlabel(r'$\varepsilon$')
plt.ylabel(r'$| \|\Omega A\|_2 - \|A\|_2 |/\| A\|_2$')
```

Exercise 2: Randomized SVD

Consider the following algorithm to compute a randomized SVD factorization:
Remember the following theorem:

Theorem 2 *If Ω_1 is chosen to be i.i.d. $\mathcal{N}(0,1)$, $k, p \geq 2$, then the expectation with respect to the random matrix Ω_1 is:*

$$\mathbb{E}(\|A - Q_1 Q_1^\top A\|_2) \leq \left(1 + \frac{4\sqrt{k+p}}{p-1}\sqrt{\min(m,n)}\right)\sigma_{k+1}(A)$$

and the probability that the error satisfies

Algorithm 1 Randomized SVD q = 1

Input: $A \in \mathbb{R}^{m\times n}$, desired rank k, $l = p + k$
Output: Approximation $A_k = Q_1 U, \Sigma, V$
Sample an $n \times l$ test matrix Ω_1 with intependend mean-zero, unit-variance Gaussian entries.
Compute $Y = (AA^\top)A\Omega_1$
Construct $Q_1 \in \mathbb{R}^{m\times l}$ with columns forming an orthonormal basis for the range of Y.
Compute $B = Q_1^\top A, B \in R^{l\times n}$
Compute the rank-k truncated SVD of B as $U\Sigma V^\top, U \in \mathbb{R}^{l\times k}, V \in \mathbb{R}^{n\times k}$

$$\|A - Q_1 Q_1^\top A\|_2 \le \left(1 + 11\sqrt{k+p}\sqrt{\min(m,n)}\right)\sigma_{k+1}(A)$$

is at least $1 - 6/p^p$. *For* $p = 6$, *the probability becomes* $0,99$.

Construct a rank$-k$ approximation with $k = 10, p = 6$ to a matrix $A \in \mathbb{R}^{m\times 2m}$ via its SVD:

$$A = U^{(A)}\Sigma^{(A)}V^{(A)\top},$$

where:

- $U \in \mathbb{R}^{m\times m}$ is a Hadamard matrix
- $V \in \mathbb{R}^{2m\times 2m}$ is a Hadamard matrix
- $\Sigma \in \mathbb{R}^{m\times 2m}$ is a diagonal matrix whose diagonal entries are defined as:

  $$\Sigma_{jj} = \sigma_j = (\sigma_{k+1})^{\lfloor j/2\rfloor/5},$$

  for $j = 1, 2, ..., 9, 10$ and

  $$\Sigma_{jj} = \sigma_j = \sigma_{k+1}\frac{m-j}{m-11},$$

  for $j = 11, 12, ..., m-1, m$. Thus $\sigma_1 = 1$ and $\sigma_k = \sigma_{k+1}$.

Test this algorithm for $m = 2^{11}$, $\sigma_{k+1} = 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001$. Plot the decay of the singular values of A and compare such decay with the accuracy of the approximation, $\|A - Q_1 Q_1^\top A\|_2$. Compare it with the theorem presented above.
A Python script with the solutions is found below:

```
import numpy as np
from numpy.linalg import svd, qr, norm
import matplotlib.pyplot as plt
from scipy.linalg import hadamard
from math import log, sqrt, floor
import torch
from hadamard_transform import hadamard_transform

# So that the plots are "interactive" when we run this script
plt.ion()

def SVD_rand(A, k, p):
    '''
```

```
    Randomized SVD with q = 1
    IN :
           A          : mxn matrix to be factorized
           k          : order of approximation
           p          : such that l = p + k
    OUT :
           U          : approximated left singular vectors
           Sigma      : approximated singular values
           V          : approximated right singular vectors
    '''
    m = A.shape[0]
    n = A.shape[1]
    l = p+k
    # STEP 1
    # Using a random number generator form a i.i.d. Gaussian matrix
    Omega1 = np.random.normal(loc= 0.0, scale = 1.0, size = [n, l])
    Y = (A@np.transpose(A))@A@Omega1
    # Construct Q1
    Q1, R = qr(Y)
    # Compute B
    B = np.transpose(Q1)@A
    # Compute th rank-k truncated SVD of B
    U, Sigma, V = svd(B)
    U = U[:, 0:k]
    Sigma = Sigma[0:k]
    V = V[:, 0:k]
    U = Q1@U
    return Q1, U, Sigma, V


def buildA(m, sigma_k1, k = 10):
    '''
    From Rokhlin, Szlam, Tygert paper A Randomized Algorithm For Principal Component
    Analysis, build test matrix A of size mx(2m). We use the fast Hadamard transform
    IN:  m         : number of desired rows in matrix A
         sigma_k1  : (k+1)th biggest singular value of A
         k         : where we are going to truncate the approximation of A
    OUT: A         : matrix with desired structure

    QUESTION: Can we build A faster? Notice that Sigma is just a diagonal matrix.
    Also notice that we can use the fast Hadamard transform to build A.
    If you can, change this function so that it builds A faster!
    '''
    U = (1/sqrt(m))*hadamard(m)
    V = (1/sqrt(2*m))*hadamard(2*m)
    firstSig = [sigma_k1**(floor(j/2)/5) for j in range(1, k+1)]
    sigmas = firstSig + [sigma_k1*(m - j)/(m - 11) for j in range(k+1, m+1)]
    Sigma = np.zeros((m, 2*m))
    np.fill_diagonal(Sigma, sigmas)
    return U@Sigma@np.transpose(V), sigmas

# Test
m = 2**11
k = 10
p = 6
sigma_k1S = [0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001]
errorApprox = np.empty(6)
errorApproxRel = np.empty(6)
errTh = np.empty(6)
```

```
for s in range(6):
    sigma = sigma_k1S[s]
    # Build A
    A, sigmas = buildA(m, sigma, k)
    # Randomized SVD
    Q1, U, S, V = SVD_rand(A, k, p)
    Sigma = np.zeros((U.shape[1], S.shape[0]))
    np.fill_diagonal(Sigma, S)
    # Plot the decay of the singular values
    plt.figure(figsize=(8, 6), dpi=80)
    plt.loglog(np.arange(m), sigmas, marker = 'o', c = "#0800ff")
    plt.title("Decay on singular values for " + r"$\sigma_{k+1} = $" + str(sigma))
    plt.xlabel("k")
    plt.ylabel(r"$\sigma_{k}$")
    # Save the error of the approximation
    errTh[s] = norm( A - Q1@np.transpose(Q1)@A)


# Plot error from theorem
plt.figure(figsize=(8, 6), dpi=80)
plt.loglog(sigma_k1S, errTh, marker = 'o', c = "#ff8f00")
plt.title(r"$| \| A - Q_1Q_1^{\top}A\|  $" + " and decay on singular values")
plt.xlabel(r"$\sigma_{k+1}$")
plt.ylabel(r"$| \| A - Q_1Q_1^{\top}A\|  $")
```