EPFL

HPC for numerical methods and data analysis

Fall Semester 2023

Prof. Laura Grigori

Assistant: Mariana Martinez

Session 8 – November 7, 2023

Randomized low rank approximation

For $A \in \mathbb{R}^{n\times n}$ SPSD (symmetric positive semidefinite) we want to implement a randomized algorithm that approximates this matrix.

Exercise 1: Create test matrices

Build the following test matrices. Here $n = 10^3$ and $R \in \{5, 10, 20\}$.

Low-Rank and PSD Noise. Let A be in the following form:

$$A = \text{diag}(1, ..., 1, 0, ..., 0) + \xi n^{-1} W,$$

where there are R initial 1's followed by zeros, $W \in \mathbb{R}^{n\times n}$ has a Wishart distribution, $W \sim \text{WISHART}(n, n)$. That is $W = GG^\top$, where $G \in \mathbb{R}^{n\times n}$ is a standard normal matrix. The parameter ξ controls the signal-to-noise ratio. Consider three examples, $\xi = 10^{-4}$, $\xi = 10^{-2}$, and $\xi = 10^{-1}$.

Polynomial Decay. Let A be in the following form:

$$A = \text{diag}(1, ..., 1, 2^{-p}, 3^{-p}, ..., (n - R + 1)^{-p},$$

where there are R initial 1's. Let $p \in \{0.5, 1, 2\}$.

Exponential Decay. Let A be in the following form:

$$A = \text{diag}(1, ..., 1, 10^{-q}, 10^{-2q}, ..., 10^{-(n-R)q}),$$

where there are R initial 1's and the parameter $q > 0$ controls the rate of exponential decay. Let $q \in \{0.1, 0.25, 1\}$.

Exercise 2: Randomized Nyström

For $A \in \mathbb{R}^{n\times n}$ SPSD and a sketching $\Omega_1 \in \mathbb{R}^{n\times l}$, randomized Nyström approximation computes:

$$\tilde{A}_{\text{Nyst}} = (A\Omega_1)(\Omega_1^\top A\Omega_1)^\dagger(\Omega_1^\top A),$$

Algorithm 1 Randomized Nyström

Input: $A \in \mathbb{R}^{n\times n}$, $l \in \mathbb{N}$, sketching $\Omega_1 \in \mathbb{R}^{n\times l}$
Output: Approximation $\tilde{A}_{\text{Nyst}} = \hat{U}\Sigma^2\hat{U}^\top$
Compute $C = A\Omega_1$
Compute $B = \Omega_1^\top C$ and its Cholesky factorization $B = LL^\top$
Compute $Z = CL^{-\top}$
Compute the QR factorization of $Z = QR$
Compute the SVD factorization of $R = \tilde{U}\Sigma\tilde{V}^\top$
Compute $\hat{U} = Q\tilde{U}$
Output factorization $\tilde{A}_{\text{Nyst}} = \hat{U}\Sigma^2\hat{U}^\top$

where $(\cdot)^\dagger$ denotes the pseudoinverse. Consider the following algorithm:
Do the following:

a) Plot the singular values of the matrices built in exercise 1

b) Explain the idea behind Nyström factorization and possible problems with algorithm 1

c) Implement algorithm 1

d) For each of the test matrices, plot the singular values of B, compute the condition number of this matrix and explain why this might be a problem

e) Relate the condition number of B with computational difficulties when computing $Z = CL^{-\top}$

f) Propose a stable algorithm for computing Z in the test matrices

g) Plot the relative error, $\text{rel}(A, \tilde{A}_{\text{Nyst}})$

h) Comment on the relationship between the relative error with the condition number of A, the condition number of B and the computation of Z

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import norm, qr, cholesky, inv, svd, matrix_rank, lstsq, cond
import pdb
from scipy.linalg import ldl

plt.ion()

# Build synthetic test matrices


def psd_Noise(R, n, xi, seed = 166297):
    '''
    Build a nxn matrix A such that
        A = diag(1, 1, 0, ..., 0) + xi/n W
    '''
    np.random.seed(seed)
    W = np.random.normal(loc= 0.0, scale = 1.0, size = [n, n])
    W = W@np.transpose(W)
    diagA = np.zeros((n))
```

```
    diagA[0:R] = 1
    A = np.diag(diagA) + (xi/n)*W
    return A

def poly_decay(R, n, p):
    '''
    Build a nxn matrix A such that
       A = diag(1, .., 1, 2^(-p), ..., (n-R+1)^(-p)
    '''
    diagA = np.ones(n)
    diagA[R:] = [(i)**(-p) for i in range(2, n-R+2)]
    A = np.diag(diagA)
    return A

def exp_decay(R, n, q):
    '''
    Build a nxn matrix A such that
       A = diag(1, ..., 1, 10^(-q), ..., 10^(-(n-R)q)
    '''
    diagA = np.ones(n)
    diagA[R:] = [10**(-i*q) for i in range(1, n-R+1)]
    A = np.diag(diagA)
    return A

def setColNames(axes, cols = ['PSD + Noise', 'Poly Decay', 'Exp Decay'],
                rows = ['R = 5', 'R = 10', 'R = 20']):
    '''
    for the grid of plots set column and row names
    '''
    if(type(axes[0]) != np.ndarray):
        for i in range(len(axes)):
            axes[i].set(xlabel = cols[i], ylabel = rows[0])
    else:
        for ax, col in zip(axes[0], cols):
            ax.set_title(col)
        for ax, row in zip(axes[:,0], rows):
            ax.set_ylabel(row, rotation=0, size='large')

def setIndividualTitles(axes, titles =  ['PSD + Noise', 'Poly Decay', 'Exp Decay']):
    '''
    Set individual titles for subplots
    '''
    if(type(axes[0]) != np.ndarray):
        axes = axes.reshape(1, len(axes))
    for i in range(axes.shape[1]):
        axes[0, i].set_title( titles[i] )

def randNystrom(A, Omega, returnExtra = True):
    '''
    Randomized Nystrom
    Option to return the singular values of B and rank of A
    '''
    m = A.shape[0]
    n = A.shape[1]
    l = Omega.shape[1]
    C = A@Omega
    B = np.transpose(Omega)@C
    try:
        # Try Cholesky
```

```
        L = cholesky(B)
        Z = lstsq(L, np.transpose(C))[0]
        Z = np.transpose(Z)
    except np.linalg.LinAlgError as err:
        # Do LDL Factorization
        lu, d, perm = ldl(B)
        # Question for you: why is the following line not 100% correct?
        lu = lu@np.sqrt(np.abs(d))
        # Does this factorization actually work?
        L = lu[perm, :]
        Cperm = C[:, perm]
        Z = lstsq(L, np.transpose(Cperm))[0]
        Z = np.transpose(Z)
    Q, R = qr(Z)
    U_t, Sigma_t, V_t = svd(R)
    Sigma_t = np.diag(Sigma_t)
    U = Q@U_t
    if returnExtra:
        S_B = cond(B)
        rank_A = matrix_rank(A)
        return U, Sigma_t@Sigma_t, np.transpose(U), S_B, rank_A
    else:
        return U, Sigma_t@Sigma_t, np.transpose(U)



n = 10**3
l = 50
Rs = [5, 10, 20]
xis = [10**(-4), 10**(-2), 10**(-1)]
ps = [0.5, 1, 2]
qs = [0.1, 0.25, 1]
Omega = np.random.normal(loc= 0.0, scale = 1.0, size = [n, l])

colors = ['#003aff', '#ff8f00', '#b200ff']

# Grid of plots
fig_diag, axs_diag = plt.subplots(nrows=3, ncols=3, figsize=(12, 8))
fig_diag.suptitle('Diagonal entries of A')

fig_err, axs_err = plt.subplots(nrows=1, ncols=3, figsize=(12, 8))
fig_err.suptitle('Relative error of approximation')

fig_condB, axs_condB = plt.subplots(nrows=1, ncols=3, figsize=(12, 8))
fig_condB.suptitle('Condition number of B')

fig_condA, axs_condA = plt.subplots(nrows=1, ncols=3, figsize=(12, 8))
fig_condA.suptitle('Condition number of A')

for i in range(3):
    # Iterate through the Rs
    R = Rs[i]
    err_noise = 3*[0]
    err_poly = 3*[0]
    err_exp = 3*[0]
    conB_noise = 3*[0]
    conB_poly = 3*[0]
    conB_exp = 3*[0]
    conA_noise = 3*[0]
```

```
    conA_poly = 3*[0]
    conA_exp = 3*[0]
    for j in range(3):
        # Iterate through the parameters of the matrices
        xi = xis[j]
        p = ps[j]
        q = qs[j]
        A_noise = psd_Noise(R, n, xi)
        A_poly = poly_decay(R, n, p)
        A_exp = exp_decay(R, n, q)

        # Loglog the diagonals
        axs_diag[i, 0].loglog(np.arange(n), np.diag(A_noise),
                              c = colors[j], label = r'$\xi=$' + str(xi))
        axs_diag[i, 0].legend(loc='upper left')
        axs_diag[i, 1].loglog(np.arange(n), np.diag(A_poly),
                              c = colors[j], label = r'$p=$' + str(p))
        axs_diag[i, 1].legend(loc='upper left')
        axs_diag[i, 2].loglog(np.arange(n), np.diag(A_exp),
                              c = colors[j], label = r'$q=$' + str(q))
        axs_diag[i, 2].legend(loc='upper left')

        # Randomized Nystrom
        U_noise, S2_noise, UT_noise, S_B_noise, rankA_noise = randNystrom(A_noise, Omega)
        U_poly, S2_poly, UT_poly, S_B_poly, rankA_poly = randNystrom(A_poly, Omega)
        U_exp, S2_exp, UT_exp, S_B_exp, rankA_exp = randNystrom(A_exp, Omega)
        # Save errors to loglog
        err_noise[j] = norm(U_noise@S2_noise@UT_noise - A_noise)/norm(A_noise)
        err_poly[j] = norm(U_poly@S2_poly@UT_poly - A_poly)/norm(A_poly)
        err_exp[j] = norm(U_exp@S2_exp@UT_exp - A_exp)/norm(A_poly)
        conB_noise[j] = S_B_noise
        conB_poly[j] = S_B_poly
        conB_exp[j] = S_B_exp
        conA_noise[j] = cond(A_noise)
        conA_poly[j] = cond(A_poly)
        conA_exp[j] = cond(A_exp)
    # Loglog
    axs_err[0].loglog(xis, err_noise, c = colors[i], label = 'R= ' + str(R))
    axs_err[1].loglog(ps, err_poly, c = colors[i], label = 'R= ' + str(R))
    axs_err[2].loglog(qs, err_exp, c = colors[i], label = 'R= ' + str(R))
    axs_condB[0].loglog(xis, conB_noise, c = colors[i], label = 'R= ' + str(R))
    axs_condB[1].loglog(ps, conB_poly, c = colors[i], label = 'R= ' + str(R))
    axs_condB[2].loglog(qs, conB_exp, c = colors[i], label = 'R= ' + str(R))
    axs_condA[0].loglog(xis, conA_noise, c = colors[i], label = 'R= ' + str(R))
    axs_condA[1].loglog(ps, conA_poly, c = colors[i], label = 'R= ' + str(R))
    axs_condA[2].loglog(qs, conA_exp, c = colors[i], label = 'R= ' + str(R))

# Add legends and format titles

axs_err[0].legend(loc='upper left')
axs_err[1].legend(loc='upper left')
axs_err[2].legend(loc='upper left')
axs_condB[0].legend(loc='upper left')
axs_condB[1].legend(loc='upper left')
axs_condB[2].legend(loc='upper left')
axs_condA[0].legend(loc='upper left')
axs_condA[1].legend(loc='upper left')
axs_condA[2].legend(loc='upper left')
```

```
setColNames(axs_diag)
setColNames(axs_err, cols = [r'$\xi$', 'p', 'q'], rows = ['Rel err'])
setIndividualTitles(axs_err)
setColNames(axs_condB, cols = [r'$\xi$', 'p', 'q'], rows = [r'$\kappa$'])
setIndividualTitles(axs_condB)
setColNames(axs_condA, cols = [r'$\xi$', 'p', 'q'], rows = [r'$\kappa$'])
setIndividualTitles(axs_condA)
```