EPFL

HPC for numerical methods and data analysis

Fall Semester 2023

Prof. Laura Grigori

Assistant: Mariana Martinez

Session 12 – December 5, 2023

Arnoldi Process and application to GMRES

Exercise 1: Randomized GS

Consider the Randomized Gram-Schmidt algorithm (RGS) from Oleg and Grigori, `https://arxiv.org/pdf/2011.05090.pdf`.

Algorithm 1 Randomized Gram-Schmidt algorithm (RGS)

Input: $n \times m$ W and sketching matrix Ω of size $k \times n$ with $m \leq k \leq n$
Output: $n \times m$ factor Q and $m \times m$ upper triangular factor R.
for $j = 1 : m$ **do**
Sketch $w_i : p_i = \Omega w_i$
Solve the $k \times (i-1)$ least squares problem:

$$R_{1:i-1,i} = \arg\min_y \|S_{i-1}y - p_i\|.$$

Compute the projection of $w_i : q_i' = w_i - Q_{i-1}R_{1:i-1,i}$.
Sketch $q_i' : s_i' = \Omega q_i'$.
Compute the sketched norm $r_{i,i} = \|s_i'\|$.
Scale vector $s_i = s_i'/r_{i,i}$.
Scale vector $q_i = q_i'/r_{i,i}$
end for

Suppose you have an orthonormal set $Q_r = \{q_1, q_2, ..., q_r\}$. Implement this algorithm in such way that it orthogonalizes a vector w_i against Q_r.

The implementation can be found below. Notice that we added an optional input Q to represent the orthonormal set $Q_r = \{q_1, q_2, ..., q_r\}$.

```
# Randomized GS from
# https://arxiv.org/pdf/2011.05090.pdf

import numpy as np
```

```
from numpy.linalg import lstsq, norm
from numpy.random import normal
from math import ceil

def rand_GS(W, k = None, Q = None, Omega = None):
    '''
    Randomized Gram Schmidt. Q is a matrix
    with orthonormal columns, we want to orthogonalize
    the columns of W against the columns of Q.
    Omega is the sketching matrix, if not given
    assumed to be normal
    '''
    n = W.shape[0]
    m = W.shape[1]
    if k is None:
        k = ceil(0.3*min(m,n))
    if Omega is None:
        Omega = normal(loc = 0.0, scale = 1.0, size = (k, n))
    if Q is None:
        start = 1
        Qm = np.empty_like(W)
        end = m
        Sm = np.empty( (k,m) )
        Qm[:, 0] = W[:,0]/norm(W[:,0])
        Sm[:, 0] = Omega@W[:,0]
        Sm[:, 0] = Sm[:,0]/norm(Sm[:,0])
    else:
        start = Q.shape[1]
        Qm = np.empty( (n, m+start) )
        Qm[:, 0:start] = Q # Since these columns are orthonormal already
        end = m+start
        Sm = np.empty( (k, m+start) )
        Sm[:, 0:start] = Omega@Q
    # Start the iteration
    for i in range(start, end):
        pi = Omega@W[:, i-start]
        # Solve the least squares problem
        ri = lstsq(Sm[:, 0:i], pi)[0]
        qiPrime = W[:, i-start] - Qm[:, 0:i]@ri
        siPrime = Omega@qiPrime
        rii = norm(siPrime)
        Sm[:, i] = siPrime/rii
        Qm[:, i] = qiPrime/rii
    return Qm
```

Exercise 2: GMRES

Recall that the Arnoldi process can be used to find eigenvalues. It can also be used to solve systems of equations $Ax^* = b$. The following diagram might be useful:

	$Ax = b$	$Ax = \lambda x$
$A = A^*$	CG	Lanczos
$A \neq A^*$	GMRES CGN BCG et al.	Arnoldi

This was taken from Trefethen and Bau's book `http://www.stat.uchicago.edu/~lekheng/courses/309/books/Trefethen-Bau.pdf`.

At each step GMRES approximates the solution x^* by a vector in the Krylov subspace $x_n \in \mathcal{K}_n$ that minimizes the residual $r_n = b - Ax_n$. This week we are going to implement GMRES from the lecture slides in a sequential fashion. Next week we are going to modify them so that they are implemented using MPI.

a) Implement Algorithm 2.

b) Test this method with the following matrices:

- A from the file `VG_mat.csv` and b from the file `VG_b.csv`
- A from the file `CTT_mat.csv` and b from the file `CTT_b.csv`
- A and $b = f$ from session 9. Such matrix and vector had to do with kernel regression using the MNIST data set.

c) Plot the number of GMRES iterations vs L_2 error

d) Let $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$ be the eigenvalues of A. Plot $\max_j \lambda_j/\lambda_{j+1}$ vs the L_2 error.

To recall these algorithms:

Algorithm 2 Algorithm 2 GMRES with MGS

Input: $A, x_0, b, \max_{\text{iter}} = m$
Output: x^* approximation to the solution of $Ax^* = b$
  $r_0 = b - Ax_0$, $\beta = \|r_0\|_2, q_1 = r_0/\beta$
  for $j = 1 : m - 1$ **do**
    $w_{j+1} = Aq_j$
    MGS to orthogonalize w_{j+1} against $\{q_1, ..., q_j\}$
    Obtain $[r_0, AQ_j] = Q_{j+1}[\|r_0\|_2 e_1 \bar{H}_j]$
  end for
  Solve $y_m = \arg\min_y \|\beta e_1 - \bar{H}_m y\|2$

You can find the data here: `https://www.dropbox.com/scl/fi/fissuao7legmz5cv588sv/Data.zip?rlkey=ccjry33l8bmfaxqwp0odwsby3&dl=0`

The implementation of the necessary functions is here:

```
# Algorithm 2 or GMRES with Modified Gram Shmidt

import numpy as np
from numpy.linalg import lstsq, norm


def arnoldi_cgs(A, b, x0, m, r0 = None, beta = None, q = None):
    '''
    Algorithm 1, Arnoldi
    '''
    if r0 is None and beta is None and q is None:
        r0 = b - A@x0
        beta = norm(r0)
        q = r0/beta
    # Define
    Hbar = np.zeros((m+1, m))
    Q = np.zeros((A.shape[1], m))
    Q[:, 0] = q
    for j in range(0, m):
        w = A@q
        hij = [np.dot(w, Q[:, i]) for i in range(j+1)]
        Hbar[0:(j+1), j] = hij
        for i in range(j+1):
            w = w - hij[i]*Q[:, i]
        Hbar[j+1, j] = norm(w)
        if Hbar[j+1, j]<1e-10:
            break
        q = w/Hbar[j+1, j]
        Q[:, j+1] = q
    H = Hbar[0:m, :]
    return H, Q

def mgs(W, Q = None):
    '''
    Modified Gram Schmidt. If Q is not none,
    we want to orthogonalize the columns of V with respect
    to the columns in Q. We assume Q is orthonormal
    '''
    if(len(W.shape) == 1):
        W = np.reshape(W, (W.shape[0], 1))
    n = W.shape[0]
    m = W.shape[1]
    if Q is None:
        start = 1
        Qm = np.empty_like(W)
        end = m
        Qm[:, 0] = W[:,0]/norm(W[:, 0])
    else:
        start = Q.shape[1]
        Qm = np.empty( (n, m+start) )
        Qm[:, 0:start] = Q
        end = m+start
    # Start the iteration
    for i in range(start, end):
        Qm[:, i] = W[:, i-start]
        for j in range(0, i):
```

```
            Qm[:, i] = Qm[:, i] - np.dot( Qm[:, j], Qm[:, i] )*Qm[:, j]
        Qm[:, i] = Qm[:, i]/norm(Qm[:, i])
    return Qm

def gmres_mgs(A, b, x0, m, tol):
    '''
    GMRES algorithm with MGS, algorithm 2 from slides
    '''
    x = x0
    r = b - A@x
    beta = norm(r)
    q = r/beta
    e1 = np.zeros((m+1, ))
    e1[0] = 1.0
    # Define
    Hbar = np.zeros((m+1, m))
    Q = np.zeros((A.shape[1], m+1))
    Q[:, 0] = q
    for j in range(m):
        print(j)
        w = A@q
        # Arnoldi
        hij = [np.dot(w, Q[:, i]) for i in range(j+1)]
        Hbar[0:(j+1), j] = hij
        for i in range(j+1):
            w = w - hij[i]*Q[:, i]
        Hbar[j+1, j] = norm(w)
        # Least squares
        ym = lstsq( Hbar[:, 0:(j+1)], beta*e1 )[0]
        x = x0 + Q[:, 0:(j+1)]@ym
        if norm( A@x - b  )<tol:
            print("Residual tolerance reached")
            break
        if Hbar[j+1, j]<1e-12:
            print("Hbar[j+1, j] too small")
            break
        q = w/Hbar[j+1, j]
        Q[:, j+1] = q
    return x, j, norm(A@x - b)
```

While the tests and plots are here:

```
from GMRES_MGS import *
from numpy.linalg import eig
import matplotlib.pyplot as plt

plt.ion()

# For the Vico Greengard paper:
# Note that this takes a while to run
# VG_mat = np.genfromtxt("VG_mat.csv", dtype=complex)
# VG_mat = np.real(VG_mat)
# VG_b = np.genfromtxt("VG_b.csv", dtype=complex)
# VG_b = np.real(VG_b)

# For the close-to-touching interactions paper
CTT_mat = np.genfromtxt("CTT_mat.txt", delimiter = ',')
CTT_b = np.genfromtxt("CTT_b.csv", delimiter = ',')
```

```
# For the MNIST data set
RBF_b = np.genfromtxt("RBF_b.csv", delimiter = ',')
RBF0_mat = np.genfromtxt("RBF0_mat.csv", delimiter = ',')
RBF1_mat = np.genfromtxt("RBF1_mat.csv", delimiter = ',')
RBF2_mat = np.genfromtxt("RBF2_mat.csv", delimiter = ',')

# Get eigenvalues
maxgap = 4*[0]
# eig_VG = eig(VG_mat)[0]
# maxgap[0] = max([eig_VG[i]/eig_VG[i+1] for i in range(len(eig_VG)-1) ] )
eig_CTT = eig(CTT_mat)[0]
maxgap[0] = max([eig_CTT[i]/eig_CTT[i+1] for i in range(len(eig_CTT)-1) ] )
eig_RBF0 = eig(RBF0_mat)[0]
maxgap[1] = max([eig_RBF0[i]/eig_RBF0[i+1] for i in range(len(eig_RBF0)-1) ] )
eig_RBF1 = eig(RBF1_mat)[0]
maxgap[2] = max([eig_RBF1[i]/eig_RBF1[i+1] for i in range(len(eig_RBF1)-1) ] )
eig_RBF2 = eig(RBF2_mat)[0]
maxgap[3] = max([eig_RBF2[i]/eig_RBF2[i+1] for i in range(len(eig_RBF2)-1) ] )

# Use GMRES
ms = [10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100]
tol = 1e-10
l = len(ms)
#iterVG = 10*[0]
iterCTT = l*[0]
iterRBF0 = l*[0]
iterRBF1 = l*[0]
iterRBF2 = l*[0]
#errVG = l*[0]
errCTT = l*[0]
errRBF0 = l*[0]
errRBF1 = l*[0]
errRBF2 = l*[0]


for i in range(l):
    #VG_x, j, err = gmres_mgs(VG_mat, VG_b, VG_b, ms[i], tol)
    #iterVG[i] = j
    #errVG[i] = err
    CTT_x, j, err = gmres_mgs(CTT_mat, CTT_b, CTT_b, ms[i], tol)
    iterCTT[i] = j
    errCTT[i] = err
    RBF0_x, j, err = gmres_mgs(RBF0_mat, RBF_b, RBF_b, ms[i], tol)
    iterRBF0[i] = j
    errRBF0[i] = err
    RBF1_x, j, err = gmres_mgs(RBF1_mat, RBF_b, RBF_b, ms[i], tol)
    iterRBF1[i] = j
    errRBF1[i] = err
    RBF2_x, j, err = gmres_mgs(RBF2_mat, RBF_b, RBF_b, ms[i], tol)
    iterRBF2[i] = j
    errRBF2[i] = err

# Plot
plt.figure(figsize=(8, 6), dpi=80)
plt.loglog(iterCTT, errCTT, c = "#00d4e9", marker = 'o', label = 'Close-to-touching')
plt.loglog(iterRBF0, errRBF0, c = "#004dff", marker = 'o', label = 'RBF0')
plt.loglog(iterRBF1, errRBF0, c = "#0a00ad", marker = 'o', label = 'RBF1')
plt.loglog(iterRBF2, errRBF0, c = "#242b4f", marker = 'o', label = 'RBF2')
```

```
plt.legend()
plt.title("Number of GMRES iterations and L2 error")
plt.xlabel("GMRES iterations")
plt.ylabel("L2 error")


errsLast = [errCTT[-1], errRBF0[-1], errRBF1[-1], errRBF2[-1]]
plt.figure(figsize=(8, 6), dpi=80)
plt.scatter(maxgap, errsLast, c = "#7100e9", marker = 'o')
plt.title("Maximum spectral gap and L2 error")
plt.xlabel(r'$\max_{i} \frac{\lambda_{i}}{\lambda_{i+1}}$')
plt.ylabel("L2 error")
```