THE CHINESE UNIVERSITY OF HONG KONG

Department of Mathematics

2024-25 Term 2 MATH2221A Mathematics Laboratory II

Lab Assignment 6 Suggested Solutions

- Full Mark: 40

1. (8 marks) Consider $A = \begin{pmatrix} 2 & 0 & 2 & 5 \\ 0 & 3 & 1 & 3 \\ 1 & 0 & 3 & 0 \\ 1 & 2 & 1 & 5 \end{pmatrix}$.

   Use suitable MATLAB built-in functions to find the trace, determinant, rank, and reduced row-echelon form of A. Write down both the commands and the answers obtained in the box below.

Solution:

```
>> A = [2, 0, 2, 5; 0, 3, 1, 3; 1, 0, 3, 0; 1, 2, 1, 5];
>> trace(A)
ans =
    13

>> det(A)
ans =
   -4.0000

>> rank(A)
ans =
     4

>> rref(A)
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

2. The **QR Iteration** method is an iterative scheme for computing all eigenvalues of an $n \times n$ matrix A based on the use of QR factorization. In this question, we will implement the QR Iteration method and compare the results with the built-in eigenvalue functions in MATLAB.

   (a) (8 marks) Write a MATLAB function `e = QRIteration(A)` that takes a $n \times n$ matrix A as input and performs the following tasks:

   - Repeat the following steps for 100 times:
     - Compute the QR factorization of A to get $QR = A$.
     - Replace A with RQ (i.e. multiply Q and R in the reverse order to form a new matrix).
     - If the matrix 2-norm of `A-triu(A)` is less than 10^{-4} (i.e. A is close enough to an upper triangular matrix), terminate the iterations.
   - After the above procedure is completed, the diagonal entries of the latest A should correspond to the eigenvalues of the original A (automatically in descending order in magnitude). Store the diagonal entries as a $n \times 1$ vector `e` and return `e` as output.

   Include the code file `QRIteration.m` in your submission.

   Solution:

```
function e = QRIteration(A)
for i = 1:100
    [Q,R] = qr(A);
    A = R*Q;
    if norm(A-triu(A),2) < 1e-4
        break;
    end
end
e = diag(A);
end
```

   (b) (4 marks) Write a MATLAB script `q2b.m` to do the following:

   - Construct a matrix `A = sqrt(magic(10))` (remark: `magic` is a built-in function that generates a magic square).

- Run the `QRIteration` function with input `A` and record the resulting eigenvalues `e`.
- Run the built-in eigenvalue computation function in MATLAB to get the 5 largest magnitude eigenvalues of `A` and record them as `e2`.
- Compute the vector 2-norm difference between `e(1:5)` and the vector `e2`.

Include the code file `q2b.m` in your submission.

Solution:

```
A = sqrt(magic(10));
e = QRIteration(A);
e2 = eigs(A,5);
norm(e(1:5)-e2)
```

Answer = `1.4150e-10`

Remark: this number may vary slightly on different computers with different MATLAB versions.

3. It is well-known that the Gram–Schmidt process transforms a given set of n linearly independent real column vectors $\{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n\}$ into an *orthonormal* set of vectors $\{\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_n\}$ (i.e. with the dot product of any two of them satisfying $\mathbf{q}_i \cdot \mathbf{q}_j = 1$ if $i = j$ and 0 if $i \neq j$). In this question, we will consider two numerical algorithms for the Gram–Schmidt process and compare their performance.

   (a) (8 marks) In the **Classical Gram–Schmidt (CGS)** algorithm, we first get an **orthogonal** set of vectors $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\}$ without normalization:

$$\begin{cases} \mathbf{v}_1 = \mathbf{a}_1, \\ \mathbf{v}_2 = \mathbf{a}_2 - \left(\dfrac{\mathbf{v}_1 \cdot \mathbf{a}_2}{\mathbf{v}_1 \cdot \mathbf{v}_1}\right) \mathbf{v}_1, \\ \mathbf{v}_3 = \mathbf{a}_3 - \left(\dfrac{\mathbf{v}_1 \cdot \mathbf{a}_3}{\mathbf{v}_1 \cdot \mathbf{v}_1}\right) \mathbf{v}_1 - \left(\dfrac{\mathbf{v}_2 \cdot \mathbf{a}_3}{\mathbf{v}_2 \cdot \mathbf{v}_2}\right) \mathbf{v}_2, \\ \quad \vdots \\ \mathbf{v}_n = \mathbf{a}_n - \left(\dfrac{\mathbf{v}_1 \cdot \mathbf{a}_n}{\mathbf{v}_1 \cdot \mathbf{v}_1}\right) \mathbf{v}_1 - \left(\dfrac{\mathbf{v}_2 \cdot \mathbf{a}_n}{\mathbf{v}_2 \cdot \mathbf{v}_2}\right) \mathbf{v}_2 - \cdots - \left(\dfrac{\mathbf{v}_{n-1} \cdot \mathbf{a}_n}{\mathbf{v}_{n-1} \cdot \mathbf{v}_{n-1}}\right) \mathbf{v}_{n-1}. \end{cases} \tag{1}$$

Once we have obtained $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$, we can get an **orthonormal** set of vectors $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n\}$ by normalizing each of them:

$$\mathbf{q}_1 = \frac{\mathbf{v}_1}{\|\mathbf{v}_1\|_2}, \ \mathbf{q}_2 = \frac{\mathbf{v}_2}{\|\mathbf{v}_2\|_2}, \ \ldots, \ \mathbf{q}_n = \frac{\mathbf{v}_n}{\|\mathbf{v}_n\|_2}. \tag{2}$$

Write a MATLAB function `Q = CGS(A)` that takes a $m \times n$ matrix A (storing n column vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$) as input and performs the following tasks:

- If $\text{rank}(A) \neq n$ (i.e. the column vectors are not all linearly independent), return an empty matrix as output.
- If $\text{rank}(A) = n$, return a $m \times n$ matrix Q (storing the n orthonormal column vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$) as output based on the CGS algorithm (i.e. Eq. (1) and Eq. (2)).

  Hint: You may utilize nested for-loops for implementing Eq. (1), where one of them loops through $i = 1, 2, \ldots, n$ (for different lines in Eq. (1)) and one loops through $j = 1, 2, \ldots, i-1$ (for the operations within each line in Eq. (1)).

Include the code file `CGS.m` in your submission.

Solution:

```
function Q = CGS(A)
[m,n] = size(A);
if rank(A) ~= n
    Q = [];
else
    % Eq. (1)
    V = zeros(m,n);
    for i = 1:n
        V(:,i) = A(:,i);
        for j = 1:(i-1)
            V(:,i) = V(:,i) - ...
                dot(V(:,j),A(:,i))/dot(V(:,j),V(:,j))*V(:,j);
        end
    end
    % Eq. (2)
```

```
    Q = zeros(m,n);
    for i = 1:n
        Q(:,i) = V(:,i)/norm(V(:,i),2);
    end
end
end
```

(b) (4 marks) It turns out that rounding errors may easily accumulate throughout the orthogonalization process in the CGS algorithm and affect the orthogonality of the resulting vectors. To remedy this issue, the **Modified Gram–Schmidt (MGS)** algorithm is considered.

Specifically, throughout the orthogonalization process, when $k \geq 2$, instead of directly computing $\mathbf{v}_k$ using $\mathbf{a}_k$ as in Eq. (1)

$$\mathbf{v}_k = \mathbf{a}_k - \left(\frac{\mathbf{v}_1 \cdot \mathbf{a}_k}{\mathbf{v}_1 \cdot \mathbf{v}_1}\right)\mathbf{v}_1 - \left(\frac{\mathbf{v}_2 \cdot \mathbf{a}_k}{\mathbf{v}_2 \cdot \mathbf{v}_2}\right)\mathbf{v}_2 - \cdots - \left(\frac{\mathbf{v}_{k-1} \cdot \mathbf{a}_n}{\mathbf{v}_{k-1} \cdot \mathbf{v}_{k-1}}\right)\mathbf{v}_{k-1},$$

in MGS we will compute $\mathbf{v}_k$ via $(k-1)$ sub-steps as follows:

$$\begin{cases} \mathbf{v}_k^{(1)} = \mathbf{a}_k - \left(\dfrac{\mathbf{v}_1 \cdot \mathbf{a}_k}{\mathbf{v}_1 \cdot \mathbf{v}_1}\right)\mathbf{v}_1, \\ \mathbf{v}_k^{(2)} = \mathbf{v}_k^{(1)} - \left(\dfrac{\mathbf{v}_2 \cdot \mathbf{v}_k^{(1)}}{\mathbf{v}_2 \cdot \mathbf{v}_2}\right)\mathbf{v}_2, \\ \mathbf{v}_k^{(3)} = \mathbf{v}_k^{(2)} - \left(\dfrac{\mathbf{v}_3 \cdot \mathbf{v}_k^{(2)}}{\mathbf{v}_3 \cdot \mathbf{v}_3}\right)\mathbf{v}_3, \\ \qquad \vdots \\ \mathbf{v}_k^{(k-2)} = \mathbf{v}_k^{(k-3)} - \left(\dfrac{\mathbf{v}_{k-2} \cdot \mathbf{v}_k^{(k-3)}}{\mathbf{v}_{k-2} \cdot \mathbf{v}_{k-2}}\right)\mathbf{v}_{k-2}, \\ \mathbf{v}_k = \mathbf{v}_k^{(k-1)} = \mathbf{v}_k^{(k-2)} - \left(\dfrac{\mathbf{v}_{k-1} \cdot \mathbf{v}_k^{(k-2)}}{\mathbf{v}_{k-1} \cdot \mathbf{v}_{k-1}}\right)\mathbf{v}_{k-1}. \end{cases} \tag{3}$$

In other words, every line in Eq. (1) will be replaced with a set of sub-steps as described in Eq. (3). After performing Eq. (1) with this modification, we perform the same normalization procedure as in Eq. (2).

Write a MATLAB function `Q = MGS(A)` that takes a $m \times n$ matrix A (storing n column vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$) as input and performs the following tasks:

- If $\text{rank}(A) \neq n$ (i.e. the column vectors are not all linearly independent), return an empty matrix as output.
- If $\text{rank}(A) = n$, return a $m \times n$ matrix Q (storing the n orthonormal column vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$) as output based on the MGS algorithm. *Hint: Your code should be largely similar to the one in part (a) except for some minor change in one of the for-loops.*

Include the code file `MGS.m` in your submission.

Solution:

```
function Q = MGS(A)
[m,n] = size(A);
if rank(A) ~= n
    Q = [];
else
    % Eq. (1) with modification
    V = zeros(m,n);
    for i = 1:n
        V(:,i) = A(:,i);
        for j = 1:(i-1)
            % modification based on Eq. (3)
            % replacing A(:,i) with V(:,i)
            V(:,i) = V(:,i) - ...
                dot(V(:,j),V(:,i))/dot(V(:,j),V(:,j))*V(:,j);
        end
    end
    % Eq. (2)
    Q = zeros(m,n);
    for i = 1:n
        Q(:,i) = V(:,i)/norm(V(:,i),2);
    end
end
end
```

(c) (8 marks) Consider the following $n \times n$ matrix A_n:

$$A_n = 10^{-6} I_n + H_n$$
$$= \begin{pmatrix} 10^{-6} & & & \\ & 10^{-6} & & \\ & & \ddots & \\ & & & 10^{-6} \end{pmatrix} + \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{n+1} \\ \vdots & & \ddots & & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \frac{1}{n+2} & \cdots & \frac{1}{2n-1} \end{pmatrix},$$

where I_n is the $n \times n$ identity matrix and H_n is an $n \times n$ matrix with its (i, j)-th entry being $\dfrac{1}{i + j - 1}$ for all $1 \leq i, j \leq n$. Note that H_n is also known as the Hilbert matrix and can be created using the MATLAB built-in `hilb` function.

Write a MATLAB script `q3c.m` to do the following:

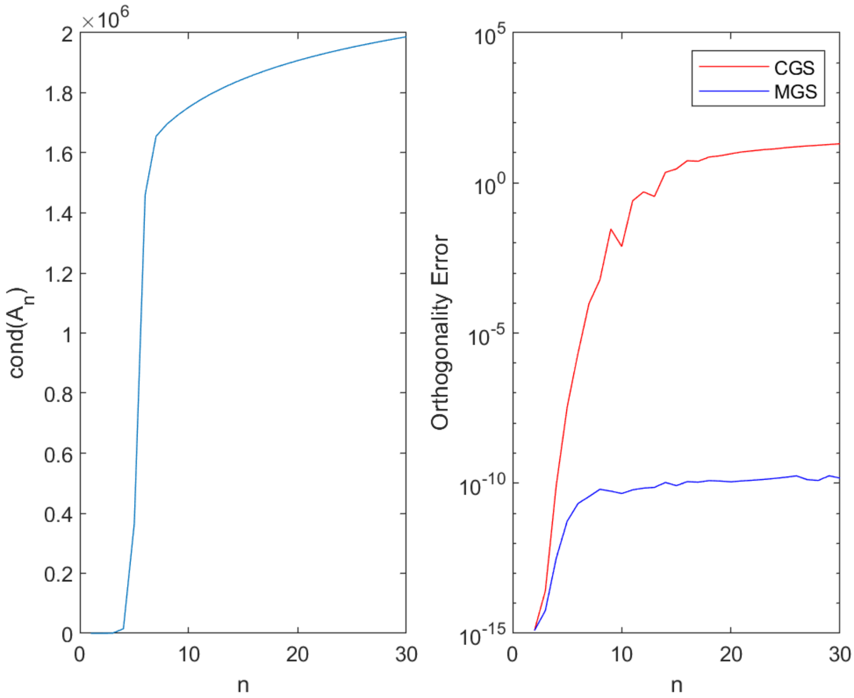
- For $n = 1, 2, \ldots, 30$,
  - Construct the matrix A_n and record the condition number $\kappa(A_n)$ (with default 2-norm used).
  - Run the `CGS` function with input A_n to obtain the matrix Q and record the matrix 2-norm error $\|Q^T Q - I_n\|_2$.
  - Run the `MGS` function with input A_n to obtain the matrix Q and record the matrix 2-norm error $\|Q^T Q - I_n\|_2$.
- Create a MATLAB figure with two subplots:
  - Subplot 1: A plot of the condition number $\kappa(A_n)$ versus n for all $n = 1, 2, \ldots, 30$. Label the x-axis as "n" and the y-axis as "cond(A_n)".
  - Subplot 2: Two semilogy plots of $\|Q^T Q - I_n\|_2$ versus n (for all $n = 1, 2, \ldots, 30$) for both CGS and MGS in the same figure. Use <u>red solid lines</u> for CGS and <u>blue solid lines</u> for MGS. Add the legends "CGS" and "MGS". Label the x-axis as "n" and the y-axis as "Orthogonality Error".

Include the code file `q3c.m` in your submission.

Solution:

```
N = 30;
error_CGS = zeros(N,1);
error_MGS = zeros(N,1);
k_A = zeros(N,1);
for n = 1:N
    A = 10^(-6)*eye(n)+hilb(n);
    k_A(n) = cond(A);

    Q = CGS(A);
    error_CGS(n) = norm(Q'*Q-eye(n),2);

    Q = MGS(A);
    error_MGS(n) = norm(Q'*Q-eye(n),2);
end

figure;

subplot(1,2,1);
plot(1:N,k_A);
xlabel('n');
ylabel('cond(A_n)');

subplot(1,2,2);
semilogy(1:N,error_CGS,'r-');
hold on;
semilogy(1:N,error_MGS,'b-');
legend('CGS','MGS');
xlabel('n');
ylabel('Orthogonality Error');
```

The figure is as follows:



Remark: The error values may vary slightly on different computers with different MATLAB versions. They may also vary with different mathematically equivalent commands used in the computation, such as `dot(u,v)` *and* `u.*v`*. due to the imperfection of finite precision arithmetics of the computer. We will grade this question based on the correctness of the codes instead of the exact values here.*

End of Lab Assignment