University of Nottingham
UK | CHINA | MALAYSIA

MATH2019 Introduction to Scientific Computation

— Coursework 2 (10%) —

Submission deadline: ***3pm, Monday, 14 Dec 2020*** ~~***7 Dec 2020***~~

Note: This is currently **version 3** of the PDF document.

No more new questions will be added anymore.

This version has some updated text on page 4; See notes in the margin.

This coursework contributes **10%** towards the overall grade for the module.

Rules:

- Each student is to submit their own coursework.
- You are allowed to work together and discuss in small groups (2 to 3 people), but *you must write your own coursework and program all code by yourself*.
- Please be informed of the UoN Academic Misconduct Policy (incl. plagiarism, false authorship, and collusion).

Piazza and Live Computing Classes:

- You are allowed to ask questions on Piazza to obtain clarification of the coursework questions, as well as general Matlab queries. This is certainly encouraged!
- However, **when using Piazza, please ensure that you are not revealing any answers to others**. Also, **please ensure that you are not directly revealing any code that you wrote**. Doing so is considered Academic Misconduct.
- **When in doubt, please simply arrange a 1-on-1 meeting with the demonstrators or lecturers during the Live Computing Classes.**

Coursework Aim & Assessment:

- In this coursework you will develop Matlab code related to algorithms that solve linear systems ~~non-linear equations~~, and you will study some of the algorithms' behaviour.
- A total of 40 points can be obtained in this coursework.

Some Advice:

- You are expected to have basic familiarity with Matlab (in particular: vectors and matrices, plotting, logic and loops, function handles, writing functions and m-files).
- Helpful resources: Matlab Guide (by Tom Wicks), Matlab's Online Documentation
- Write your code as neatly and concisely as possible so that it is easy to follow.
- Add some comments to your scripts that indicate what a piece of code does.

Getting Started:

- Download the contents of the "Coursework 2 Pack" folder from Moodle.

Submission Procedure:

- Submission will open after 26 November 2020.
- To submit, simply upload the required m-files on Moodle. (Your submission will be checked for plagiarism using *turnitin*.)
- Your m-files will be marked (mostly) automatically: This is done by running your functions and comparing their output against the true results.
- You can check the outputs that your m-files generate by running the file `GenerateYourOutputsCW2.m`, which can be found in folder "Coursework 2 Pack" from Moodle.

► **Gaussian elimination algorithm: Forward elimination without pivoting**

To solve a $n \times n$ linear system of equations, consider the Gaussian elimination algorithm consisting of forward elimination (without row interchanges) and backward substitution. Recall from Lecture 4 (video PartA) that the forward-elimination step consists of $n-1$ rounds of row-replacements:

for $i = 1, \ldots, n-1$

  for $j = i+1, \ldots, n$

    $$E_j - \frac{a_{ji}}{a_{ii}} E_i \to E_j$$

  end

end

A Matlab implementation of this is as follows:

```
% Get number of rows
n = size(A,1);
% Start forward elimination
for i = 1:n-1
    % Eliminate column i
    for j = i+1:n
        % Compute multiplier
        mij = A(j,i)/A(i,i);
        % Replace Ej by Ej-mij*Ei
        A(j,i) = 0;
        for k = i+1:n+1
            A(j,k) = A(j,k) - mij*A(i,k);
        end
    end
end
B=A;
```

Note: The following Question 1 and Question 2 can be completed independently. (In other words, you don't have to finish Question 1 to be able to do Question 2.)

1 • Write a *function* m-file called `forwElimStop_func.m`, which implements the above forward elimination method, but it has additional arguments $(\mathtt{r}, \mathtt{s})$ that prematurely stop the forward elimination. The function *must* have the following prototype:

```
function B = forwElimStop_func(A,r,s)
```

where the input `A` is any *augmented* matrix (number of columns is one larger than number of rows), `r` is the number of forward elimination rounds to be completed, `s` is the number of row replacements to be done in the last round, and the output `B` is the resulting matrix.

• Test your function by considering, for example, the system used in Lecture 4, and typing in the command window:

```
A = [2 3 4 2; 4 9 10 6; 6 15 20 12]
r = 0
s = 1
B = forwElimStop_func(A,r,s)
```

The results for this example should be as follows:

$(r,s) = (0,0):\quad B = \begin{bmatrix} 2 & 3 & 4 & 2 \\ 4 & 9 & 10 & 6 \\ 6 & 15 & 20 & 12 \end{bmatrix}$ $\qquad$ $(r,s) = (1,0):\quad B = \begin{bmatrix} 2 & 3 & 4 & 2 \\ 0 & 3 & 2 & 2 \\ 0 & 6 & 8 & 6 \end{bmatrix}$

$(r,s) = (0,1):\quad B = \begin{bmatrix} 2 & 3 & 4 & 2 \\ 0 & 3 & 2 & 2 \\ 6 & 15 & 20 & 12 \end{bmatrix}$ $\qquad$ $(r,s) = (1,1):\quad B = \begin{bmatrix} 2 & 3 & 4 & 2 \\ 0 & 3 & 2 & 2 \\ 0 & 0 & 4 & 2 \end{bmatrix}$

$(r,s) = (0,2):\quad B = \begin{bmatrix} 2 & 3 & 4 & 2 \\ 0 & 3 & 2 & 2 \\ 0 & 6 & 8 & 6 \end{bmatrix}$ $\qquad$ $(r,s) = (2,0):\quad B = \begin{bmatrix} 2 & 3 & 4 & 2 \\ 0 & 3 & 2 & 2 \\ 0 & 0 & 4 & 2 \end{bmatrix}$

• Also test your function when combined with the backward subtitution step: The relevant function m-file, `backwSub_func.m`, can be downloaded from Moodle (Coursework 2 Pack). Typing the following into the command window should give the solution of the system ($\mathtt{x} = [-1/2; 1/3; 1/2]$):

```
A = [2 3 4 2; 4 9 10 6; 6 15 20 12]
r = 2
s = 0
B = forwElimStop_func(A,r,s)
x = backwSub_func(B)
```

• Also add a brief description at the top of your `forwElimStop_func` m-file. This description should become visible, whenever one types: `help forwElim_func`. **Hint:** See also Section 9.5 of the Matlab Guide for writing a "help" for user-defined functions.

Marks can be obtained for your `forwElimStop_func` m-file for generating the required output, for certain set(s) of inputs. The correctness of the following will be checked: **[8 / 40]**

- The size and values of `B`
- The correctness of `x`, when combined with `backwSub_func`
- The output of "help forwElimStop_func".

(Run the file `GenerateYourOutputsCW2.m` to see these outputs generated using the set of inputs from Part 1. Note that in marking your work, different input(s) may be used.)

► **Gaussian elimination algorithm: Forwarding elimination with *partial pivoting***

Now consider Gaussian elimination with forward elimination using *partial pivoting* as discussed in Lecture 5 (video PartC). Recall that the pivoting is carried out at the start of each round before performing the usual row operations.

2 • Write a *function* m-file called `forwElimPP_func.m`, which implements forward

elimination with the partial pivoting strategy. The function *must* have the following prototype:

```
function B = forwElimPP_func(A)
```

where the input `A` is any *augmented* matrix (number of columns is one larger than number of rows), and the output `B` is the final matrix (in echelon form).

Hint: To find a maximum in some vector (or matrix), you can use the Matlab function `max` (see `help max` for more info). For the absolute value use `abs`.

← Hint added on **17 Nov 2020**

• Test your function by considering, for example, the system used in Lecture 4, and typing in the command window:

```
A = [2 3 4 2; 4 9 10 6; 6 15 20 12]
B = forwElimPP_func(A)
```

which should do the forward elimination by interchanging rows 1 and 3, then rows 2 and 3.

Hint: It may help to start from the given Matlab implementation for forward elimination, and to add "`disp(A)`" right before the final "`end`", so that you can see `A` at the end of each elimination round. (Don't forget to remove this again once ready.)

• Also test your function when combined with the backward subtitution step: The relevant function m-file, `backwSub_func.m`, can be downloaded from Moodle (Coursework 2 Pack). Typing the following into the command window should give the solution of the system ($\mathtt{x} = [-1/2; 1/3; 1/2]$):

```
A = [2 3 4 2; 4 9 10 6; 6 15 20 12]
B = forwElimPP_func(A)
x = backwSub_func(B)
```

• Also add a brief description at the top of your `forwElimPP_func` m-file. This description should become visible, whenever one types: `help forwElimPP_func`.

Hint: See also Section 9.5 of the Matlab Guide for writing a "help" for user-defined functions.

[8 / 40]

Marks can be obtained for your `forwElimPP_func` m-file for generating the required output, for certain set(s) of inputs. The correctness of the following will be checked:

- The size and values of `B`
- The correctness of `x`, when combined with `backwSub_func`
- The output of "help forwElimPP_func".

(Run the file `GenerateYourOutputsCW2.m` to see these outputs generated using the set of inputs from Part 2. Note that in marking your work, different input(s) may be used.)

► LU factorisation

Consider the LU factorisation of an $n \times n$ matrix A (see Lecture 6). Theorem 6.19 provides the general form for the matrix L and U.

3 • Write a *function* m-file called `forwElimLU_func.m`, which computes the LU factorisation of a given A. The function *must* have the following prototype:

```
function [L,U] = forwElimLU_func(A)
```

where the input `A` is any *square* matrix, the output `U` is the final echelon form (an upper triangular matrix) as obtained by forward elimination (without row exchanges), and the output `L` is the corresponding lower-triangular matrix as defined in Theorem 6.19

Hint: It may help to start from the given Matlab implementation for forward elimination on page 2 of this PDF.

• Test your function by considering, for example, the matrix used in Lecture 6 and typing in the command window:

```
A = [1 1 0; 2 1 -1; 0 -1 -1]
[L,U] = forwElimLU_func(A)
```

Marks can be obtained for your `forwElimLU_func` m-file for generating the required output, for certain set(s) of inputs. The correctness of the following will be checked: **[6 / 40]**

• The size and values of `L` and `U`

• The correctness of the product `LU`

(Run the file `GenerateYourOutputsCW2.m` to see these outputs generated using the set of inputs from Part 3. Note that in marking your work, different input(s) may be used.)

4 • Add additional code to your `forwElimLU_func.m`, so that it does not divide by (almost) zero when applied to certain matrices, but instead stops prematurely. In other words, stop the forward elimination when it encounters in the current pivot position (a_{ii}) a value very close to zero. In particular, the elimination must stop when $|a_{ii}| < $ 1e-8. When stopping early, ensure the output of `forwElimLU_func.m` is simply the incomplete `L` and `U` matrices obtained thus far.

• Test your function by considering, for example, the following:

```
A = [1 1 0; 0 0 -1; 1 -1 -1]
[L,U] = forwElimLU_func(A)
```

which should return $\mathtt{L} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$ and $\mathtt{U} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & -2 & -1 \end{bmatrix}$.

• Also add a brief description at the top of your `forwElimLU_func` m-file. This description should become visible, whenever one types: `help forwElimLU_func`.

Marks can be obtained for your `forwElimLU_func` m-file for generating the required output, for certain set(s) of inputs. The correctness of the following will be checked: **[6 / 40]**

- The size and values of `L` and `U`, when presented with a matrix `A` having (close to) zero pivot.
- The correctness of the corresponding product `LU`
- The output of "help forwElimLL_func".

(Run the file `GenerateYourOutputsCW2.m` to see these outputs generated using the set of inputs from Part 4. Note that in marking your work, different input(s) may be used.)

The below was added on 24 Nov 2020.

► Jacobi method

To solve a linear system of equations, $A\mathbf{x} = \mathbf{b}$, we now consider iterative techniques (see Lecture 7), in particular, we focus on the Jacobi method.

5 • Write a *function* m-file called `jacobi_func.m`, which computes the approximation vectors $\{\mathbf{x}^{(k)}\}$ according to Jacobi's method. The function *must* have the following prototype:

```
function x_mat = jacobi_func(A,b,x0,Nmax)
```

where the input consists of an $(n \times n)$ matrix `A`, an $(n \times 1)$ vector `b`, an $(n \times 1)$ vector `x0` (a given initial approximation), and `Nmax`, which is the total number of iterations to be performed by the method. The output `x_mat` is a matrix of size $n \times (\texttt{Nmax}+1)$ that stores all approximation vectors $\mathbf{x}^{(k)}$, that is, column $k+1$ of matrix `x_mat` contains $\mathbf{x}^{(k)}$, for $k = 0, 1, \ldots, \texttt{Nmax}$.

Hint: You are welcome to use any of Matlab's built-in functions (such as `diag`, `tril`, `triu`, "\", `inv`, etc) in order to obtain D, L and U (in the decomposition $A = D - L - U$), and subsequently compute the iteration matrix T and vector $\mathbf{c}$. (See also last video recording of Lecture 7.)

• Test your function by considering, for example, the problem studied in Lecture 7, and typing in the command window:

```
A = [4 -1 0; -1 8 -1; 0 -1 4]
b = [48 ; 12 ; 24]
x0 = [1 ; 1 ; 1]
Nmax = 10
x_mat = jacobi_func(A,b,x0,Nmax)
```

• As usual, also add a brief description at the top of your `jacobi_func` m-file.

[6 / 40]

Marks can be obtained for your `jacobi_func` m-file for generating the required output, for certain set(s) of inputs. The correctness of the following will be checked:

- The size and values of `x_mat`
- The output of "`help jacobi_func`".

(Run the file `GenerateYourOutputsCW2.m` to see these outputs generated using the set of inputs from Part 5. Note that in marking your work, different input(s) may be used.)

► **Jacobi method: Find solvable system sizes**

Next, consider the following interesting system of equations. Let $n = m^3$, where m is an integer at least equal to 2 (hence n is at least 8). Consider the system $A\mathbf{x} = \mathbf{b}$, where A is an $n \times n$ symmetric matrix with nonzero values on 7 of its diagonals: It has the values 6 on the main diagonal, and -1 on the diagonals that are 1, m, and m^2 above and below the main diagonal. The vector $\mathbf{b}$ has a value 9 as its first entry, 6 for the first m entries except the 1st, and 3 for the first m^2 entries except the first m. This pattern in $\mathbf{b}$ is mirrored from below. Here is an illustration of A and $\mathbf{b}$:

$$A = \begin{bmatrix} 6 & -1 & & -1 & & -1 & & \\ -1 & 6 & -1 & & -1 & & -1 & \\ & -1 & \ddots & \ddots & & \ddots & & \ddots \\ -1 & & \ddots & & & & & \\ & \ddots & & & & & & \\ & & & & & & & \\ -1 & & & & & & & \\ & \ddots & & & & & & \\ & & & & & & & \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} 9 \\ 6 \\ \vdots \\ 6 \\ 3 \\ \vdots \\ 3 \\ 0 \\ \vdots \\ \\ \vdots \\ 0 \\ 3 \\ \vdots \\ 3 \\ 6 \\ \vdots \\ 6 \\ 9 \end{pmatrix}.$$

Note that the solution is $\mathbf{x} = \begin{pmatrix} 3 \\ 3 \\ \vdots \\ 3 \end{pmatrix}$ (the $n \times 1$ vector with all entries $= 3$).

Fortunately, a Matlab function m-file, `someMatrixAndVector_func.m`, that builds this matrix and vector for arbitrary input m can be downloaded from Moodle (Coursework 2 Pack), and can be combined with the Jacobi method as follows:

```
m = 2
[A,b] = someMatrixAndVector_func(m)
n = m^3
x0 = ones(n,1);
Nmax = 10
x_mat = jacobi_func(A,b,x0,Nmax)
```

Now comes the interesting thing! When one tries to solve matrix A using the Jacobi method, one observes convergence for any m(!), but it becomes slower for larger system sizes. Assuming we only wish to have at most `Nmax` iterations in the Jacobi method, a natural question is: *What is the largest solvable system?*

6⋆ • To address the above question, write a *function* m-file called `findSystemSizes_func.m`, which computes all possible values of $m \geq 2$ for which the Jacobi method approximates the exact solution in at most `Nmax` iterations up to a certain tolerance `TOL`, that is,

$$\left\|\mathbf{x} - \mathbf{x}^{(k)}\right\|_2 \leq \texttt{TOL} \qquad \text{for some } k \leq \texttt{Nmax}.$$

The function *must* have the following prototype:

```
function [m_vec,k_vec,p] = findSystemSizes_func(Nmax,TOL,mMax)
```

In your implementation, for simplicity, you only need to consider values of m such that $2 \leq m \leq$ `mMax`. Also, you can assume that the initial vector in the Jacobi method is always the vector with ones, i.e., $\mathbf{x}^{(0)} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$. Ensure that the output `m_vec` is the vector of all valid values of m, in increasing order, and that `k_vec` is the corresponding vector of smallest iteration indices k for which the above criterion is satisfied.

• Furthermore, there is one additional output requested: The scalar `p`. Based on your observations, this is simply your estimate for the power in the observed relationship $\texttt{k} \approx C\, n^{\texttt{p}}$, where $n = m^3$. Note that this relationship links the number of required iterations to the true system size n. You can simply assign this value somewhere within the m-file.

Hint: You are welcome to use Matlab's `norm` to compute 2-norms. You may also find useful: `find`, `max`, `min`, and other Matlab's inbuilt functions.

• Test your function by considering, for example:

```
Nmax = 200
TOL  = 10^(-2)
mMax = 10
[m_vec,k_vec,p] = findSystemSizes_func(Nmax,TOL,mMax)
```

Marks can be obtained for your `findSystemSizes_func` m-file for generating the required outputs, for certain set(s) of inputs. The correctness of the following will be checked: **[6 / 40]**

- The values in `m_vec` and `k_vec`
- Your estimate of `p` (an estimate with 2 significant figures is sufficient)
- Also, your function should complete in a reasonable time.

(Run the file `GenerateYourOutputsCW2.m` to see these outputs generated using the set of inputs from Part 6.

► Finished!