**MATH2019 (2020-2021)**
**Introduction to Scientific Computation**
**Coursework 2 Feedback**
**Hand-in date: 14 Dec 2020**

**Coursework details**

- Number of submitted scripts: 216
- Maximum mark possible: 40
- Average mark: 33.3 (= 83%)

**Solution m-files**

Solution m-files can be found on the next few pages. Obviously, many alternatives are possible.

**Feedback**

An individual CW2_MARKS_REPORT for your submission can be found in Moodle (CW1 Submission -> Feedback files), both in html and pdf format. This report contains details on the obtained marks and comments for each output generated by your functions.
***Please have a careful look at your report, and let me know if you notice some unusual marking of your outputs.***
It is very well possible that the automated marking script needs further refining.
Note: In your CW2_MARKS_REPORT, you can now also see the test-case descriptions, just prior to the corresponding outputs.

**General feedback on performance of the class**

Generally, the overall student's performance was very good.
There were many marks >30, and only few marks <20.
**Q1** (forwElimStop_func): mean = 88%.
Well done by most. A 3x3 linear system was tested (as in the example), but also a 4x4 linear system.
**Q2** (forwElimPP_func): mean = 72%
Well done by many. This was one of the lowest scoring questions. A 4x4 linear system was tested (a 3x3 linear system was given as example). While the pivoting may have been wrongly implemented, some marks were still awarded if the linear system was correctly solved (backward-step test).
**Q3** (forwElimLU_func): mean = 94%
Well done by most. This was the highest-scoring question. Tested on a small variation of the given example.
**Q4** (forwElimLU_func with stop): mean = 86%
Well done by most. A 4x4 linear system was tested (a 3x3 linear system was given as example).

**Q5** (jacobi_func): mean = 86%
Well done by most. Tested on a small variation of the given example.
**Q6** (findSystemSizes_func): mean = 74%
Well done by many. This was one of the lowest scoring questions. Tested on a small variation of the given example. For the output p, a variation of 0.1 around the true value was given full marks, while half the marks was given for a variation of 0.67 around the true value.

**Automated marking algorithm**
*Apologies for the delayed return of your marks and feedback.*
It is the first year that we have been using automated marking (for 85% of the submission for CW1 and 20% for CW2). The other parts of the submissions (15% in CW1 and 20% in CW2) were manually marked by the Teaching Assistants (those involving the function descriptions). While a lot of the developed automation algorithm for CW1 was useful for CW2, it did require *significant* manual changes, that was the cause of the significant delay. These changes have now been made so as to minimise the required updates for CW3 and CW4.

**Precise tests carried out on your functions, and generating your outputs:**
To see the tests carried out on your functions, please look into your CW2_MARKS_REPORT, where you can see the test-case descriptions, just prior to the corresponding outputs. You can simply run these tests on your own functions.
If you wish to generate all the marked outputs, download the file GenerateCW1outputsForMarking.m from the Coursework 2 Marking Details Folder on Moodle, into the folder which contains all your m-files. If you wish, you can open this m-file to see how your outputs will be generated, or you can simply run the following in Matlab:

```
   output = GenerateCWoutputsForMarking();
```

Once the above has run, all your outputs are stored in the cell-struct variable "output" (type for example: output{1}, output{1}.sizeB, output{2}, output{2}.Bcol1, etc.)
To see the true outputs, please download the file CW2_TRUEOUTPUTS_FOR_MARKING.mat, from the Coursework 2 Marking Details Folder on Moodle. In Matlab, go to the folder containing the file, then run:

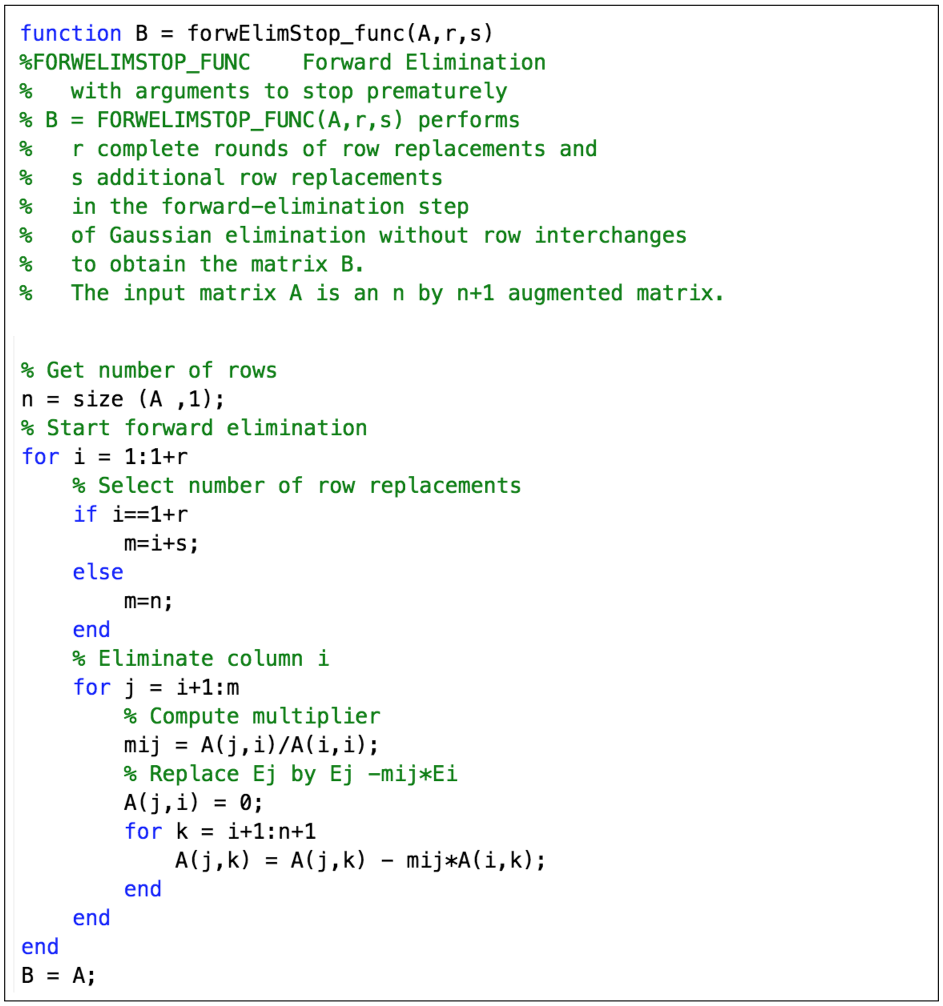
```
   load('CW2_TRUEOUTPUTS_FOR_MARKING.mat','trueOutput');
```

This will load the true outputs into the cell-struct variable "trueOutput".

# MATH2019 (2020-2021)
# Introduction to Scientific Computation
# Coursework 2 Solution

## Contents

## 1 m-file: forwElimStop_func.m (Question 1)



```
function B = forwElimStop_func(A,r,s)
%FORWELIMSTOP_FUNC    Forward Elimination
%   with arguments to stop prematurely
% B = FORWELIMSTOP_FUNC(A,r,s) performs
%   r complete rounds of row replacements and
%   s additional row replacements
%   in the forward-elimination step
%   of Gaussian elimination without row interchanges
%   to obtain the matrix B.
%   The input matrix A is an n by n+1 augmented matrix.


% Get number of rows
n = size (A ,1);
% Start forward elimination
for i = 1:1+r
    % Select number of row replacements
    if i==1+r
        m=i+s;
    else
        m=n;
    end
    % Eliminate column i
    for j = i+1:m
        % 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;
```

## 2 m-file: forwElimPP_func.m (Question 2)

```
function B = forwElimPP_func(A)
%FORWELIMPP_FUNC    Forward Elimination with Partial Pivoting
%   B = FORWELIMPP_FUNC(A) performs the forward-
%   elimination step of Gaussian elimination with
%   row interchanges based on partial pivoting.
%   Input:  n by n+1 augmented matrix A
%   Output: echelon form B of A

% Get n
n = size(A,1);

% Start forward elimination
for i = 1:n-1

    % Partial pivoting: Find row p
    [~,imax] = max(abs(A(i:n,i)));
    p = imax(1)+i-1;

    % Swap row p with row i
    if p~=i
        old_row_p = A(p,:);
        A(p,:) = A(i,:);
        A(i,:) = old_row_p;
    end

    % Eliminate column i
    for j = i+1:n

        % Compute multiplier
        m = A(j,i)/A(i,i);

        % Replace Ej by Ej-m*Ei
        A(j,i) = 0;
        for k = i+1:n+1
          A(j,k) = A(j,k) - m*A(i,k);
        end

    end

%    A
end

% Return output B
B = A;
```

## 3 m-file: forwElimLU_func.m (Question 3 & 4)

```
function [L,U] = forwElimLU_func(A)
%FORWELIMLU_FUNC    Forward Elimination LU Factorisation
%   [L,U] = FORWELIMLU_FUNC(A) performs the forward-
%   elimination step of Gaussian elimination
%   without row interchanges to obtain the
%   echelon form U of A, and in addition
%   computes the lower triangular matrix L,
%   so that L*U = A.
%   The input matrix A is an n by n matrix.
%   When A has a (close to) zero pivot,
%   it stops elimination early and returns
%   the incomplete L and U.



% Extract number of rows n from A
n = size(A,1);

% Initialise L (ones on diagonal)
L = diag(ones(1,n));

% Start Gaussian elimination loop
for i = 1:n-1

    % Eliminate column i
    for j = i+1:n
        if abs(A(i,i)) < 1e-8
            break
        end
        % Compute multiplier
        mij = A(j,i)/A(i,i);

        % Update L
        L(j,i) = mij;

        % Replace Ej by Ej-m*Ei
        A(j,i) = 0;
        for k = i+1:n
          A(j,k) = A(j,k) - mij*A(i,k);
        end

    end

%    A
end

% Return output U
U = A;
```

## 4 m-file: jacobi_func.m (Question 5)

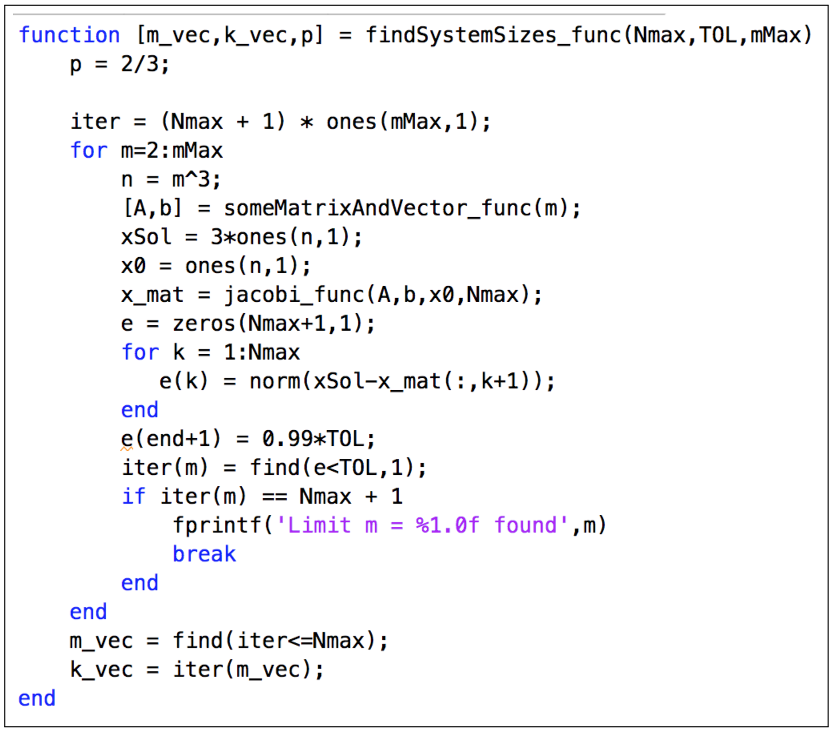
```
function x_mat = jacobi_func(A,b,x0,Nmax)
%JACOBI_FUNC  Jacobi Iteration Method
%   x_mat = JACOBI_FUNC(A,b,x0,Nmax) performs
%   the Jacobi iteration method
%   for the solution of
%   the linear system Ax = b,
%   for Nmax iterations,
%   where A is an nxn matrix,
%         b is an nx1 vector,
%        x0 is an nx1 starting vector.
%   Output: x_mat is an nx(Nmax+1) matrix
%           containing all iteration
%           vectors x_k, k=0,1,...,Nmax.

x_mat = zeros(length(x0),Nmax+1);
x_mat(:,1) = x0;

L = -tril(A,-1);
U = -triu(A, 1);
Dinv = diag(1./diag(A));
T = Dinv*(L+U);
c = Dinv*b;


for i=1:Nmax
    x_mat(:,i+1) = T*x_mat(:,i) + c;
end
```

## 5 m-file: findSystemSizes_func.m (Question 6)



```
function [m_vec,k_vec,p] = findSystemSizes_func(Nmax,TOL,mMax)
    p = 2/3;

    iter = (Nmax + 1) * ones(mMax,1);
    for m=2:mMax
        n = m^3;
        [A,b] = someMatrixAndVector_func(m);
        xSol = 3*ones(n,1);
        x0 = ones(n,1);
        x_mat = jacobi_func(A,b,x0,Nmax);
        e = zeros(Nmax+1,1);
        for k = 1:Nmax
           e(k) = norm(xSol-x_mat(:,k+1));
        end
        e(end+1) = 0.99*TOL;
        iter(m) = find(e<TOL,1);
        if iter(m) == Nmax + 1
            fprintf('Limit m = %1.0f found',m)
            break
        end
    end
    m_vec = find(iter<=Nmax);
    k_vec = iter(m_vec);
end
```