UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG
School of Computer Science and Applied Mathematics

Numerical Methods
Lab 1

08 February 2018

Instructions

- Read all the instructions carefully.
- MATLAB has a help file for every function if you get stuck.
- There are also numerous sources available on the internet, Google is your friend!

SECTION 1

Introduction

The first lab is to serve as either a refresher to those who have used MATLAB before, or an introduction to those who haven't.

We will now begin programming some actual numerical methods. Before we can do that, we will need to cover some basic core MATLAB structures. First we will look at vectors and Matrices

SECTION 2

Vectors and Matrices

Vectors and matrices are entered in square brackets. Row vectors are written with their elements separated by spaces or commas (e.g. `[1 2 3]` or `[1,2,3]`), column vectors have their elements separated by semicolons (e.g. `[1;2;3]`). Matrices are entered similarly, e.g. e.g. `[1 2 3; 4 5 6]`. A vector can be transposed by using the ' operator: e.g. `[1 2 3]'`. Matrices and vectors can be added `(+)` and multiplied `(*)` just as in mathematics. A matrix or vector can also be pairwise-multiplied and divided (`.*` and `./` respectively) by a matrix or vector of the same dimensions. The linear system $Ax = b$ can be solved by typing `A\b`.

2.1 Indexing

An element of a vector is referenced by an index in round brackets. Indices start from 1 and can also be specified by a vector, e.g. `x(2)` returns the second element in the vector x, while `x([2 4])` returns the second and fourth elements.

Elements of a matrix are referred to by two comma separated indices, the first refers to the row and the second refers to the column, e.g. `A(1,2)`. An index colon may be used to refer to whole rows or columns of a matrix. E.g. `A(:,1)` refers to the first column of the matrix A, while `A(2,:)` refers to the second row of the matrix A. It is also possible to access parts of a matrix, for example, `A([1 3],:)` refers to the first and third row of matrix `A`, and `A(1:3,2:4)` refers to row 1 to row 3 and column 2 to column 4 of matrix A.

2.2 Manipulating Matrices

By using the relevant index, elements in vectors and matrices can be changed or deleted. To change the 4th element in a vector, x to the value 3, use:

```
>> x(4) = 3
```

and to delete it, use:

```
>> x(4) = []
```

To change or delete more than one element use a vector of indices to indicate the positions of the elements that you wish to remove. E.g.

```
>> x([1 3 6]) = []
```

Vectors and matrices can also be concatenated (joined). E.g. if y and z are vectors, then

```
>> x = [y z]
```

creates a new vector x with all the elements of y and z in order.

Below is an example of a function that will convert four binary digits to a decimal number:

```
function [x] = FromBinary4(a,b,c,d)
        % FromBinary4 converts 4 binary digits to a decimal number
        % INPUTS: a,b,c,d (0 or 1) are the digits of the binary number
        % OUTPUTS: x, the decimal number corresponding to (abcd)_2
                x = a*8+b*4+c*2+d;
end
```

2.3 Script Files

It is also possible in MATLAB to create a `.m` file without the function header. Such a file is known as a script file and serves the purpose of collecting together a number of commands you might otherwise

have typed in the command window. You can run a script file from the editor by pressing the play button (or typing F5). A script can also be run from the Command Window by typing its name.

A good example of how to use a script is to call a function with specific values of its input variables. For example:

```
% BinaryTester.m
% This script calls FromBinary4.m with different inputs and displays
% the results neatly on the screen

d1 = 1; d2 = 0; d3 = 1; d4 = 0; % binary digits
result1 = FromBinary4(d1,d2,d3,d4);     % result 1 is (1010)_2
d4 = 1;                                 % change a digit
result2 = FromBinary4(d1,d2,d3,d4);     % result 2 is (1011)_2
result3 = FromBinary4(1,1,1,1); % result 3 is (1111)_2

disp('First Answer:');
disp(result1);
disp('---');
disp('Second Answer:');
disp(result2);
disp('---');
disp('Third Answer:');
disp(result3);
disp('---');
```

If you run the script above, you'll notice that the workspace window updates itself to contain the new variables created in the script: `d1, d2, d3, d4`, `result1`, `result2` and `result3`. This is because running the script is exactly the same thing as typing each of the commands in the script, one at a time, in the command window.

A script is very different from a function:

- Functions **cannot** access or edit any variables created in the command window. Variables created or modified within a function are **not** created or modified in the workspace outside of that fuction. The only way that the outside world can communicate with the function is by passing it INPUTS, and the only way a function can communicate with the outside world is by returning OUTPUTS.
- Any variable created in a script **will** be on the workspace, and any variable modified in a script **will also** be modified in the workspace. A script cannot take inputs and produce outputs. Especially as programs get larger, this is extremely undesirable behaviour. You should be very cautious about creating a script to do the job of a function!

SECTION 3

Loop Constructs

3.1 For Loops

The for loop is used to repeat a collection of statements a fixed number of times. The most common form (used in a script file, not on the command line) is:

```
for indexvar = j:m:k
        statements
end
```

This will execute the statements a number of times with the index variable taking on the values from `j` to `k`, in increments of `m`. MATLAB uses a default increment of 1 if you leave out the increment `m`, i.e. `j:k` is equivalent to using `j:1:k`. A more general form of the for loop is given by:

```
for indexvar = v
        statements
end
```

where `v` is any vector. In this case, the index will take on the values of each vector element in turn. Using either form, the index variable can be used by the statements section of the loop, but changing its value will not cause the loop to skip steps.

3.2 While Loops

The while loop is a construct that repeats statements while a condition remains true. It is used to repeat a collection of statements a variable number of times. The general form of the while loop is:

```
while condition
        statements
end
```

Note that the variables that make up the condition being tested must be changed by the statements, otherwise the while loop will **run forever!**

3.3 Nested Loops

A nested loop is a loop within a loop: an inner loop within the body of an outer one. These loops may be for or while loops. For example, the following construction shows two nested for loops:

```
for indexvar1 = j:k
        for indexvar2 = l:m
                statements
        end
end
```

Notice that within each loop, indentation (extra spacing) is used to indicate the level of nesting - this enhances readability of the code and is good coding practice. The index ranges of the inner loops may depend on the index values of the outer loops.

3.4 If-Else Loops

The `if-elseif-else` statement allows conditional execution of code fragments depending on one or more conditional expressions. The construction of such loops like the following:

```
if condition1
        .
        statementsA
        .
elseif condition2
        .
        statementsB
        .
else
        .
        statementsC
        .
end
```

Both the `elseif` and `else` clauses are optional and more than one `elseif` clause is allowed. Remember - as always the construction should terminate with an `end`.

SECTION 4
Main Exercises

The due date for this weeks submissions will be announced, again you will receive further instruction via email - please check your student mail regularly.

4.1 Exercise 1

Program the Gaussian elimination method with no partial pivoting and back substitution for single systems. Your function should take in a coefficient matrix A, and a single vector b. Your function should return the answer vector x. That is, your code should solve $Ax = b$. The first line of your function should look like:

```
function x = gaussElimination(A,b)
```

4.2 Exercise 2

Program Gaussian elimination with partial pivoting and back substitution for multiple systems. Your function should take in a coefficient matrix A, and a series of input vectors b as a matrix B. Your

function should return a series of vector x as a matrix X. That is, your code should solve AX=B. The first line of your function should look like:

```
function X = gaussMultipleSystems(A,B)
```

4.3

Exercise 3

Write a function that performs LU-factorisation (permuted LU factorisation) of a matrix A. Then solve the problem described in Exercise 2 using LU-factorisation (permuted LU factorisation) of a matrix A. Remember, you only need to do the LU-factorisation of a matrix A ONCE. The pseudocode for the forward elimination of the Gaussian Elimination procedure looks like:

Algorithm 1 Gaussian Elimination - Forward Elimination

for $k = 1$ **to** $n-1$ **do**
  for $i = k+1$ **to** n **do**
    for $j = k$ **to** n **do**
      $a_{ij} \leftarrow a_{ij} - (a_{ik}/a_{kk})a_{kj}$
    end for
  end for
end for

Note - You will still need to implement the backwards substitution component of the algorithm.