Numerical Methods
Lab 5

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

This week, we will continue programming some actual numerical methods. Each new lab will have all of the previous labs' material and append new information where necessary.

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 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 (not 0 as in PYTHON) 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.

2.2
Manipulating Matrices

o 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!

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:

```
1  if condition1
2          .
3          statementsA
4          .
5  elseif condition2
6          .
7          statementsB
8          .
9  else
10         .
11         statementsC
12         .
13 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

Some Useful MATLAB Built-in Commands

Command	Result
`zeros(m,n)`	Creates an $m \times n$ matrix consisting of all zeros.
`ones(m,n)`	Creates an $m \times n$ matrix consisting of all ones.
`eye(m,n)`	Creates an $m \times n$ Identity matrix.
`X = diag(v,k)`	Creates a diagonal matrix with elements of v on the kth diagonal. $k = 0$ is the main diagonal.
`v = diag(X)`	Returns a vector of the main diagonal of matrix X.
`transpose(X)`	Returns the transpose of matrix X.
`trace(X)`	Returns the sum of the elements on the main diagonal of X.
`inv(X)`	Returns the inverse of the square matrix X.
`det(X)`	Returns the determinant of the square matrix X.
`[V, D] = eig(X)`	Returns the eigenvalues and eigenvectors of matrix X.
`fliplr(X)`	Returns X with rows preserved and columns flipped in the left/right direction.
`flipud(X)`	Returns X with columns preserved and rows flipped in the up/down direction.
`tril(X,k)`	Extracts the lower triangular part of the matrix X, consisting of the elements on and below the kth diagonal.
`triu(X,k)`	Extracts the upper triangular part of the matrix X, consisting of the elements on and above the kth diagonal.
`linspace(a,b,n)`	Returns a vector of n equally spaced points between (and including) a and b.

Main Exercises

The due date for the lab submission is to be announced.

Consider the initial value problem

$$\frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0 \tag{†}$$

and do the exercises below. You may want to check how your numerical method is performing against the exact solution using the function $f(x, y) = (1 - \frac{4}{3}x)y$, where $y(0) = 1$ and $x \in [0, 3]$.

Euler Method : $y_{k+1} = y_k + hf(x_k, y_k)$

Improved Euler Method: $y_{k+1} = y_k + \frac{h}{2}\left(f(x_k, y_k) + f(x_k + h, y_k + hf(x_k, y_k))\right)$

Midpoint Method : $y_{k+1} = y_k + hf(x_k + \frac{h}{2}, y_k + \frac{h}{2}f(x_k, y_k))$

Runge kutta Method (Order 4): $y_{k+1} = y_k + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$,

where $k_1 = hf(x_k, y_k)$, $\quad k_2 = hf(x_k + \frac{h}{2}, y_k + \frac{k_1}{2})$,

$k_3 = hf(x_k + \frac{h}{2}, y_k + \frac{k_2}{2})$, $\quad k_4 = hf(x_k + h, y_k + k_3)$

5.1

Exercise 1

Program Euler's Method for solving ODEs. Your function should take as five inputs, the function `f` given in equation (†) (passed as a function handle), `x0`, the initial x value, `y0`, the initial y value, `N`, the number of subintervals and `xf` the final x value to be evaluated. The output should be an $(N+1) \times 2$ matrix with the first column representing the x values and the second column, the corresponding y values. The first row then would be the initial condition. The first line of your function should look like this:

```
1  function [x,y] = Euler(f,x0,y0,N,xf)
```

5.2

Exercise 2

Program Improved Euler's Method (or Heun's Method) for solving ODEs. Your function should take as five inputs, the function `f` given in equation (†) (passed as a function handle), `x0`, the initial x value, `y0`, the initial y value, `N`, the number of subintervals and `xf` the final x value to be evaluated. The output should be an $(N+1) \times 2$ matrix with the first column representing the x values and the second column, the corresponding y values. The first row then would be the initial condition. The first line of your function should look like this:

```
1  function [x,y] = Heun(f,x0,y0,N,xf)
```

5.3

Exercise 3

Program the Midpoint Method for solving ODEs. Your function should take as five inputs, the function f given in equation (†) (passed as a function handle), x0, the initial x value, y0, the initial y value, N, the number of subintervals and xf the final x value to be evaluated. The output should be an $(N+1) \times 2$ matrix with the first column representing the x values and the second column, the corresponding y values. The first row then would be the initial condition. The first line of your function should look like this:

```
function [x,y] = Midpoint(f,x0,y0,N,xf)
```

5.4

Exercise 4

Program the Runge-Kutta 4th order Method for solving ODEs. Your function should take as five inputs, the function f given in equation (†) (passed as a function handle), x0, the initial x value, y0, the initial y value, N, the number of subintervals and xf the final x value to be evaluated. The output should be an $(N+1) \times 2$ matrix with the first column representing the x values and the second column, the corresponding y values. The first row then would be the initial condition. The first line of your function should look like this:

```
function [x,y]= rk4(f,x0,y0,N,xf)
```

5.5

Exercise 5a

Plot the solution to the initial value problem (†) obtained in Exercises 1 using N=30 and N=300 on the same graph. Also, plot the exact solution on the very same graph. The plot of the three graphs should help underscore the importance of using a small step size.

5.6

Exercise 5b

Plot the solution to the initial value problem (†) obtained in Exercises 1-4 on the same graph. Choose N=300. You will need to use markers to differentiate between plots.

_____

5.7

Exercise 6

Program the Runge-Kutta 4th order method for solving an initial value problem for any system of n linear first order ODEs. Your function should take as four inputs, the function f (passed as a function handle), a vector xspan = [x0 xf] where x0 and xf are the initial and final x values, $\vec{y0}$, a row vector of initial values for y and its derivation at x_0 and N, the number of subintervals. The output should be x, a $(N+1) \times 1$ column vector and y, an $(N+1) \times n$ matrix with the first column representing the

y values, the second column, the y' values, ..., the last column, the $y^{(n-1)}$ values. Note that $y^{(n-1)}$ means the $(n-1)$th derivative of y. The first row then would be the initial condition. The first line of your function should look like this:

```
function [x,y]= RK4system(f⃗,xspan,y⃗0,N)
```

NB: Your function must be able to return the desired output, given any system of n first order ordinary differential equations, where $n \geq 2$.

Using your code solve the third order ordinary differential equation

$$\frac{d^3 y}{dx^3} + y\frac{d^2 y}{dx^2} = 0, \quad 0 \leq x \leq 5 \quad y(0) = 0; \ y'(0) = 0 \ y''(0) = 1.$$

Plot (x, y')

5.8

Exercise 7a

Consider the boundary value problem

$$\frac{d^2 y}{dx^2} = p(x)\frac{dy}{dx} + q(x)y + r(x), \ a \leq x \leq b \quad y(x_0) = y_0; \ y(x_f) = y_f$$

and do the exercises below. You may want to check how your numerical method is performing by solving the above boundary value problem using the function $p(x) = -3$, $q(x) = 2$ and $r(x) = 2x+3$. Choose $0 \leq x \leq 1$, $y(0) = 2; \ y(1) = 1$. Do plot the graph of y against x for the below exercise. Use markers to indicate plots.

5.9

Exercise 7b

Program the method of linear shooting for solving boundary value problems. Your function should take as seven inputs, the functions f1, defining a system of first order ordinary differential equations for the inhomogeneous equation and f2, for the homogeneous equation (both are passed as a function handle), x0, the initial x value, y_0, the initial y value, y_f, the final y value, N, the number of subintervals and xf the final x value to be evaluated. The output should be x, a $(N+1) \times 1$ column vector and y, an $(N+1) \times 2$ matrix with the first column representing the y values and the second column, the corresponding y' values. The first line of your function should look like this:

```
function [x,y] = shootingmethod(f⃗1,f⃗2,x0,y0,yf,N, xf)
```

5.10

Exercise 7c

Program the finite difference method to solve the boundary value problem outlined in Exercise 7a above. Your function should take as five inputs, x0, the initial x value, y_0, the initial y value, y_f, the final y value, N, the number of interior points and xf the final x value to be evaluated. The output should be x, a $(N+2) \times 1$ column vector and y, an $(N+2) \times 1$ column vector representing the y values. The first line of your function should look like this:

```
3   function [x,y] = findifference(x0,y0,yf,N,xf)
```