3450:427/527 Applied Numerical Methods I, Kreider, Summer 2017

Homework Set 1

Each problem is worth 10 points.

Due date: Thursday 15 June

The purpose of these problems is to acquaint you with features of MATLAB that we use extensively in this class. Review the MATLAB Overview, paying particular attention to sections 3, 4 and 5. For each problem, submit both the code and the output by using the publish feature. Use as little paper as necessary.

Download MATLAB from My Akron. Log on to your account and look for Download Software for Home Use box.

Set Preferences. To optimize the display in the command window, click on 'Files' in the upper left. Under 'Command Window Preferences', set the 'Numeric display' to compact (which eliminates blank lines in the output) and set the 'Numeric format' to long g (long means the default display of a number uses many digits and g means that MATLAB chooses between scientific notation (e format) and floating point notation (f format)). No points for this, but do it anyway.

1. Dot Operators. All variables in MATLAB are considered to be matrices, so x*y, for example, is treated as matrix multiplication. There are plenty of calculations, however, where x and y are vectors that we think of as sets of scalars, and we want to perform scalar operations on the individual elements. MATLAB has a set of 'dot operators' to do this, for operations other than addition and subtraction (which are the same for vectors and scalars). Run this code to see how these operators work. Line 3 gives an error message – learn to recognize this so you can fix your code quickly. First, type these lines in the command window to see what happens. Then put the commands in a script file with line 3 omitted or commented. Submit the script file and output for the problem.

```
x = [1 2 3 4]
y = [2 1 -2 3]
x*y
x.*y
x./y
x.^y
x+y
x-y
```

2. Plotting Functions by the Build-A-Vector Method. Sometimes, we want a quick plot of a function with no intention of using the function later. The plot command requires a vector of x values and a vector of y values. It is convenient in this case to create the y vectors directly without using MATLAB's data structures for functions. As an example,

```
h = 1;
x = 0:h:10;
y = x.^2;
figure(1)
plot(x,y)
h = .01;
x = 0:h:10;
y = x.^2; % note that when you change the x vector, you need to recompute the y vect
figure(2)
plot(x,y)
```

Figure 1 is in a window underneath Figure 2's window. See the MATLAB Overview, section 2, for the description of what's happening with x. I call h the resolution; it is important because the plot is a set of straight line segments connecting the dots in the input vectors, clearly seen in the first plot. This is called a linear spline, discussed in Chapter 5. Your assignment is to plot $f(x) = \sqrt{x^3 + 1}\sin x$ on the interval $[5, 10]$ with poor resolution ($h = 1$) and good resolution ($h = 0.01$). To print the plots, you have 2 choices. You can print from the plot window (full page for each plot, don't do this for assignments) or you can save the plot as a jpg file (click on File in the plot window, Save As and use File Type JPEG image). Then you can insert the figure in your Word document and resize it so you don't waste paper.

3. Sometimes, you want multiple graphs on one plot. Do one of following 2 options for the function in problem 2, again on $[5, 10]$. Use dots for the poor resolution graph.

```
helpwin plot

% First approach
x1 = 0:10; y1 = x1.^2;
x2 = 0:.01:10; y2 = x2.^2;
plot(x1,y1,'o',x2,y2)  % the 'o' is an optional third parameter
                       % that formats the graph of x1,y1

% Second approach
x = 0:10; y = x.^2;
figure(1)
hold on
plot(x,y,'o')
x = 0:.01:10; y = x.^2;
plot(x,y)
```

4. Sometimes we want to define a function to use several times, or in different ways, in a code. MATLAB allows us to define 'anonymous functions'; the name of the function is called the handle. There is a specific syntax that must be used in the definition: the independent variables follow the at sign in parentheses:

```
f = @(x) x.^2+1;
g = @(x,y) x ./ (y.^3+x+1);
```

Here, x and y are dummy variables that have no specific values. Here are some examples of how to use these functions:

```
f = @(x) x.^2+1;
f(3)
x=[1,2,3]
f(x)
x=0:.01:5; y = f(x);
figure(1)
plot(x,y)
figure(2)
plot(x,f(x))
g = @(x,y) x ./ (y.^3+x+1);
g(1,2)
g(5,6)
g([1 2],[5 6])
```

Your assignment is to define $f(x) = \dfrac{e^x \sin x}{x^2+1}$ as an anonymous function and plot it on $[3, 7]$ using poor resolution and good resolution (same h values as before) in one plot, with dots for the poor resolution, as in the previous problem.

5. In the previous example, the values $f(3)$ and $[f(1), f(2), f(3)]$ were displayed in the command window using the numeric format we specified in preferences. Sometimes we want more control over the format, so we use the fprintf command (section 1 of the Overview). As an example,

```
f = @(x) x.^2+1;
x = .6221;
fprintf('f(%6.5f) = %12.4e \n',x,f(x))
fprintf('f(%6.5f) = %12.4e \n',3.706,f(3.706))
```

Note that changing the output display does not change the value stored in MATLAB. Your assignment is to print the values of $f(x) = \sin(x^3)$ at the points $x = 5.201, -8323.6, 0.0003$ twice, in floating point (f format) with 8 digits after the decimal and then in scientific notation (e format) with 10 digits after the decimal. Check the help page (type helpwin fprintf) for details. Notice that the floating point format is not appropriate for the third x; get in the habit of printing in scientific notation to see the magnitude of a number.

6. A third way to define functions is to create a separate file for the function. This is used for more complicated functions, or algorithms, that require more than one line of code. The function uses only dummy variables and cannot be run directly. Instead you create a script file in which the inputs are defined and then the function is called. As an example, one way to approximate the derivative of a function at a point x is called the first order forward difference method (FD1):

$$f'(x) = \frac{f(x+h) - f(x)}{h}$$

To implement this, we create a function file called FD1.m, shown below. To signify that this file defines a function, the first line must have the proper syntax: function [outputs] = name(inputs).

```
function d = FD1(f,x,h)
d = (f(x+h)-f(x))/h;
```

A sample script that uses this simple function is

```
f = @(x) x.^2;
x = 3;
exactderiv = 2*x;
h1 = .1; h2 = .01;
d1 = FD1(f,x,h1); d2 = FD1(f,x,h2);
fprintf('The approximate derivative at %f using h=%f is %e\n',x,h1,d1)
fprintf('The approximate derivative at %f using h=%f is %e\n',x,h2,d2)
```

Your assignment follows the same structure. Write a function file that computes the roots of a quadratic using the form for x^+ and x^-1 that are least susceptible to cancellation error. Also write a script file to find the roots of

$$\begin{aligned} x^2 + 1.888888888888888x - .740740740740740740 &= 0 \\ x^2 - 12.00035802468123x + 36.00214817704637 &= 0 \end{aligned}$$

Enter the coefficients with 16 digits to get double precision accuracy!

For reference, your function file will have the structure

```
function [xp,xm] = quadform(a,b,c)
if (b<0)
    xp =
    xm =
else
    xm =
    xp =
end
```