Mech 221: Computer Pre-Lab 3

Hand in the solutions to the two questions in the pre-lab at the *beginning* of the lab.

In the upcoming lab, we will be focusing on the application of Newton's method in finding solutions to equations for which no simple algebraic solution is available. As a final goal of the lab, we will use Newton's method to generate a plot of the solution to a separable differential equation. This pre-lab will cover:

- New MATLAB commands: `format`, `xlabel`, `ylabel` and `title`.
- You will learn how to write a `while` loop in MATLAB that allows you execute the same lines of code until a certain condition is met.
- You will see how to implement a `function` in MATLAB. This is a special .m file that takes inputs and gives outputs.
- Using Newton's method to find the solutions to an equation.

MATLAB commands to know for the lab

`format long` : This will increase the digits of a number shown in MATLAB from 5 to 15 digits. Type `format short` to return to 5 digits.

`xlabel / ylabel` : Typing in `xlabel('string')` will display the word `string` along the x-axis in the current figure. `ylabel('string')` will do the same, but along the y-axis.

`title` : Typing in `title('string')` will display the word `string` as a title to the current figure.

While...end loops

Like `for`, `while` is not a simple command, but allows us to define "while loops". `while` loops are similar to `for` loops, but the condition for their termination is different. A `for` loop is terminated when the specified number of iterations has been reached. A `while` loop is terminated when a condition, supplied by the programmer, is satisfied. A common example of use of a while loop, and one we will require for this lab, is error tolerance. In MATLAB, the structure of a while loop is as follows:

```
while(<condition>)
code to be executed
end
```

where `<condition>` is the Boolean variable (true/false) that will terminate the loop when `<condition>` is "false", and the `code to be executed` are the commands that MATLAB will execute. When MATLAB reaches the end of the `code to be executed` it will re-evaluate `<condition>`, and terminate the loop if "false". It is important that your `condition` will eventually fail. Otherwise your while loop may run forever. If it happens that the while loop does not terminate after a long time, type Ctrl-C and MATLAB will stop the loop.

Consider the following example `while` loop:

```
factor = 1;
while factor > 0.1
   factor = factor/2
end
```

The statement in the `while` loop will be executed 4 times with `factor` displaying successively the values 0.5000, 0.2500, 0.1250, 0.0625.

The `while` condition can involve the following operators

<,<= : less than, less than or equal

>,>= : greater than, greater than or equal

== : equal to (note the double equal sign to distinguish this from the assignment operator)

~= : not equal to

&, | : and, or

Function files

In MATLAB when you want to define a new function (beyond the built-in functions like `sin`, `cos`, `exp`, etc.) you can write an .m file function. An .m file function will accept input parameters and return output to you based on the input. These .m files are different from the "script" .m files you have seen before in that they accept input arguments and keep all variables local

to the function. We will be using the .m file functions to evaluate functions of the form $f(x, y)$ for this lab.

To create an .m file function, we first need to create a new .m file. Then the first line of the file must be:

```
function output = funcname(input1,input2)
```

where `output` is your output variable, `input1, input2` is your list of input variables, and `funcname` is the name you choose for your function. Note that the inputs and output can be vectors and you can specify only one input or more than two inputs and more than one output if needed. After the line above, you include all the lines of code you wish to see executed using your input variables. Your output variables must be defined in this code. You **must** save the .m file as "funcname.m" (the name of the file should match the name of the function you define in it).

As an example, consider the .m file `testfn.m` in the current directory of MATLAB that contains the lines of code

```
function f = testfn(x,y)
f = x*y^2;
```

If you type

```
z=testfn(1,2);
```

then `z` will have the value 4, computed by the function code with the values `x = 1` and `y = 2` passed as inputs.

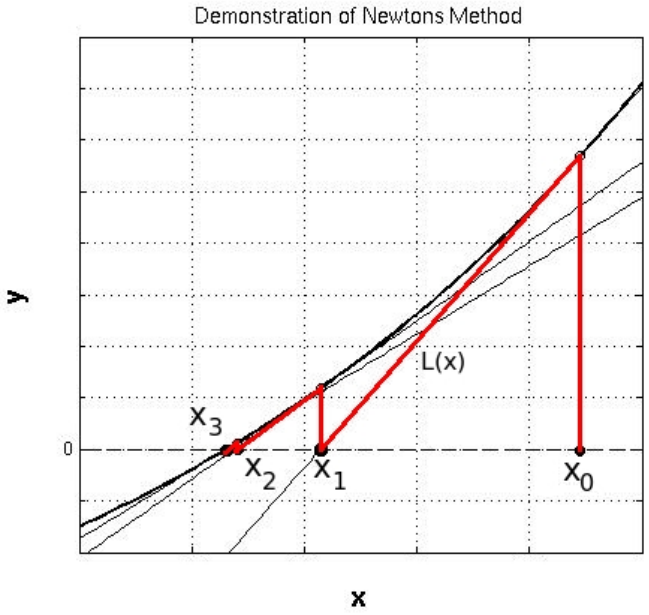
Newton's Method

When it comes to finding the roots of equations, Newton's method is a useful tool to have on your side. First we will give you a brief explanation as to how the method works, then we will give you some information about the performance and limitations of the method.

First consider the following problem. We wish to calculate the root (or roots) of a given function $f(x)$. (i.e. We wish to find the value(s) $x = x^*$ such that $f(x^*) = 0$). How can we do this? First, choose a point near the root. We will call it x_0. Next, make a linear approximation to $f(x)$ at the point $x = x_0$, giving us:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) = L(x) \tag{1}$$

Now, we will find the root of the linear approximation. The idea behind Newton's Method is that since our initial guess was close to x^*, the root of the linear approximation must be close to x^* as well. We will now find the root to the linear approximation and call it x_1.



$$L(x_1) = 0 \quad = \quad f(x_0) + f'(x_0)(x_1 - x_0) \tag{2}$$

$$\rightarrow x_1 \quad = \quad x_0 - \frac{f(x_0)}{f'(x_0)} \tag{3}$$

Now we have a better approximation x_1 for the root. At this point you may have noticed that we can apply this algorithm iteratively as the following.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{4}$$

As n gets large we expect x_n to approach the root, x^*. If this happens then we say that x_n converges to x^*. The strength of Newton's Method is that it can give very accurate answers with only a few iterations. We will demonstrate this with the following example.

Example: Find the positive, real 6th root of π.

Although this can be found with our hand calculator directly, let's consider the answer to be the value x that is a root of the following equation and apply Newton's method.

$$f(x) = x^6 - \pi = 0 \tag{5}$$

Given that we are looking for the positive root, we will take an initial guess of $x_0 = 2$ (This is clearly a bad guess since $2^6 = 64 >> \pi$, but in this case it will not matter). We calculate $f'(x) = 6x^5$, and apply our algorithm:

$$\begin{aligned}
x_{n+1} &= x_n - \frac{x_n^6 - \pi}{6x_n^5} \\
x_0 &= 2 \\
x_1 &= 2 - \frac{64 - \pi}{192} = 1.683029128 \\
x_2 &= 1.441298261 \\
x_3 &= 1.285265836 \\
x_4 &= 1.220345924 \\
x_5 &= 1.210411669 \\
x_5 &= 1.210203332 \\
x_6 &= 1.210203242 \\
x_7 &= 1.210203242
\end{aligned}$$

From this we can say that $\pi^{1/6} \approx 1.210203242$. (This calculation was done with a calculator.)

The weakness of Newton's method is that are situations where it will fail (not converge to the desired root). For example, Newton's method may converge to a different root if your initial guess is not close enough. It is also possible that the iterates will oscillate between values that are not roots, or tend to infinity in absolute value (blow up).

Assessing the Convergence of Newton's Method

If Newton's Method converges, then the iterates tend to a root and the values of

$$|x_n - x_{n-1}|$$

tend to zero. If Newton's Method fails then the quantity above almost always stays large. As a *Rule of Thumb*, for problems that are well-scaled, if

$$|x_n - x_{n-1}| < \epsilon \tag{6}$$

then $|x_n - x^*| < \epsilon$. That is if the difference between successive iterates satisfies a desired small tolerance then the approximation to that root also satisfies that tolerance. Iterating until (6) is reached can be done with a `while` loop. Two further, more rigorous checks can be done

1. If x_n is your final iterate, $f(x_n)$ should be approximately zero. This is an important check when debugging your method.

2. If it is very important that x_n be accurate to the root x^* to tolerance ϵ, compute $f(x_n + \epsilon)$ and $f(x_n - \epsilon)$. If these values have opposite signs then by the Intermediate Value Theorem, there must be a root of f within the desired tolerance from x_n. Draw a picture to convince yourself of this result.

Question 1: Solving a Differential Equation

Consider the following initial value problem:

$$\frac{dy}{dx} = \frac{2x}{ye^y}\left(1 + \frac{1}{(x^2+1)}\right), y(0) = 1 \tag{7}$$

- Solve this initial value problem. *Hint:* This differential equation is separable). You will be able to separate variables, integrate both sides and solve for the integration constant C using the initial condition. This gives a solution in an implicit form

  $$G(y) = F(x) + C \quad \text{or} \quad G(y) - F(x) - C = 0$$

  with C determined. *Hand in this solution (Show all your work).*

You are unlikely to be able to solve the DE above fully, that is to find an explicit solution (an analytic solution in the form $y(x)$). It is possible for this problem although the solution involves the Lambert-W function. This is where Newton's method comes in handy. We can use Newton's method to find the solution $y(x)$ at any given value x. This can be done for all x values on a grid and so a plot of the solution can be generated.

Question 2: Application of Newton's Method

- Recall your solution for the initial value problem in Question #1. We want to find $y(1)$ for this solution to high accuracy. Set $x = 1$ in your implicit formula from Question #1, then rewrite the relationship that $y = y(1)$ must satisfy in the form $q(y) = 0$.
- Write the formula for Newton iterations to get increasingly accurate approximations of y.
- Calculate an accurate approximation to y at $x = 1$, using Newton's Method with the formula derived above until you have 5 significant digits in your approximate answer. Take your initial guess to be $y_0 = 1.8$. Show carefully that your approximation of $y(1)$ has 5 significant digits, using the bracketing argument described above.
- *Hand in the function $q(y)$ you found above, the formula you find for the Newton iterates to this problem, list the iterates y_n you obtain and show your reasoning that your answer is correct to 5 significant figures.*

Coming up in Lab #3

Completing the pre-lab will prepare you for the lab, in which the following will be covered:

- Debugging MATLAB program code.
- Writing function .m files in MATLAB.
- Finding solutions to separable differential equations using Newton's Method in MATLAB. It is important to understand the steps leading to the solution of a separable differential equation with this technique.
- Significant MATLAB coding will be required.