Notes for lab 2

In homework 2, you are asked to write programs which will find the zero a function. One of the inputs of such a program must be the function in question. In the lab I discussed a method to pass the name of a function to another function in Matlab. I will now repeat the details here.

First I will write a function which will take in a value x and return $3x + 5$. Here is the m-file for this function.

```
function y=f1(x)

y=3.*x+5;

return
```

- The semi-colon blocks output. If it where not there, every time we called the function, it would output the result. We probably don't want this if we are going to call the function many times as in a root finding program. It may be useful to remove the semi-colon when looking for bugs though.
- The name of the function is f1 and it **must** be saved as "f1.m". The file should also be in the same directory you call Matlab from. If you are using the gui interface, the working directory should be listed in the toolbar.
- `y` will contain the value we wish to return. We must set `y` before we return.
- I used the .'s for the commands so I can pass a vector to the function and get a vector back. This is very useful for plotting.

We can test the function by calling it from the command line in Matlab.

```
f1(3)

ans =

    14
```

As you can see it works. Now we write a function which will take another function as one of it's arguments. The function we will write will take two arguments. The name of a function and a value x and return the function evaluated at $2x$. Lets call the new function fat2x (we must save it as fat2x.m). Here is the Matlab code.

```
function y=fat2x(f,x)

y=feval(f,2*x);

return
```

How does this function work? Lets call it and see.

```
fat2x('f1',1.5)
ans =
    14
```

Note: when we call fat2x, we must enclose the function name is single quotes. We are really passing a string variable which is the name of the function to be evaluated.

The point is that we may pass any function we wish. So when you write the root finding subroutines, you can set them up so that you just pass the name of a function (which is also a program you must write) and the other required information and it will return the root. In the program we can use feval to find the values of the function at the required points.

I hope this is clear.

In this assignment, you will also need to understand loops (while loops and maybe for loops) as well as conditionals (if then else end). On the course home page, there is a good link to a tutorial (the last one is good for this). I recommend you check it out.

To get you started, I worked out a bisection function for you in the lab. You will need to adapt these ideas to write a Newton's method routine. Here is a listing,

```
function [y,i]=bisect(f,a,b,tol)

if feval(f,a)*feval(f,b)>0
    error('Bad Interval Choice');
end
i=0;
while(abs(a-b)>tol)
    c=(a+b)/2;
    if feval(f,a)*feval(f,c)<0
        b=c;
    else
        a=c;
    end
    i=i+1;
end

y=c;
end
```

In your code, you should make sure that there are no problems which could creep up. In particular if one of the guesses lands on a root, we should be careful.

Here is the result of running the code on the function f1.m

```
f1(-1)

ans =

     2

f1(-5)

ans =

   -10


[y,iter]=bisect('f1',-1,-5,0.0001)
y =
   -1.6667
iter =
    16
```

When coding up Newton's method, you will need to supply 2 functions for input. The function we wish to find the zero of and the derivative of that function. You will have to find the derivatives of each function by hand and write a program to evaluate the derivative.

Calling your routine should look something like

```
[y iter]=newton('f1','f1p',x0,tol)
```

Here f1.m will be a program to evaluate a function and f1p.m will be a program to evaluate its derivative. Another point to keep in mind when coding you Newton's method is that the conditional for the while loop will look something like

```
while(abs((x1-x0)/x0)>tol)
```

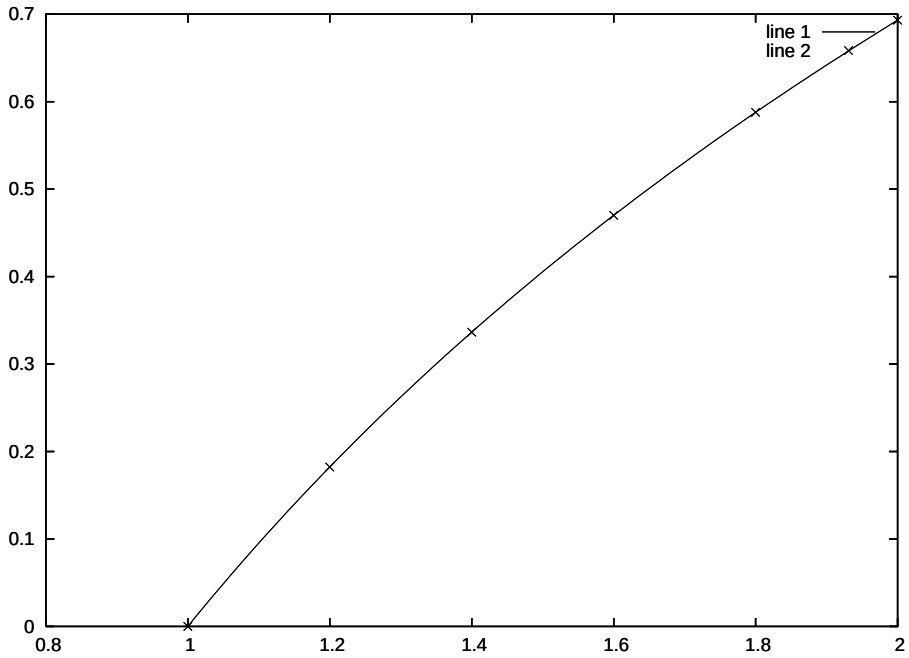
If you execute this line without putting a value in x1, the computer will give you an error. So for Newton's method, you must do the first iteration outside of the loop. The remaining iterations will be inside the loop. Please refer to the pseudo code I gave during the lecture.

Finally in the lab I worked out a divided difference table to interpolate some data. I then plot the polynomial using the nested evaluation form discussed in class.

```
x=linspace(1,2,5)'

x =

    1.0000
    1.2500
    1.5000
    1.7500
    2.0000

y=log(x)

y =

         0
    0.2231
    0.4055
    0.5596
    0.6931

F=zeros(5)

F =

     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0

F(:,1)=y

F =

         0         0         0         0         0
    0.2231         0         0         0         0
    0.4055         0         0         0         0
    0.5596         0         0         0         0
    0.6931         0         0         0         0

for i=2:5
for j=i:5
F(j,i)=(F(j,i-1)-F(j-1,i-1))/(x(j)-x(j-i+1));
end
end

F

F =

         0         0         0         0         0
    0.2231    0.8926         0         0         0
    0.4055    0.7293   -0.3266         0         0
    0.5596    0.6166   -0.2254    0.1349         0
    0.6931    0.5341   -0.1650    0.0806   -0.0544

x1=linspace(1,2);
```

```
size(x1)

ans =

     1   100

v=F(5,5).*ones(1,100);
for i=4:-1:1
v=v.*(x1-x(i))+F(i,i);
end
plot(x1,v)
hold on
plot(x,y,'x')


plot(x,y,'x')
plot(x1,log(x1),'g')
```

And I also include the graph produced.



In the homework, I don't require you to use Matlab to make the divided difference table. You can do it by hand and use Matlab to plot the polynomials. This might be easier, but if you have the time and the inclination, I would recommend writing a program to get some practice.

Finally I plotted some of the Chebyshev polynomials. Here is the matlab session:

```
octave:3> x=linspace(-1,1);
octave:4> t1=x;
octave:5> t2=2.*x.*t1-1;
octave:6> t3=2.*x.*t2-t1;
octave:7> t4=2.*x.*t3-t2;
octave:8> t5=2.*x.*t4-t3;
octave:9> plot(x,t1)
octave:10> hold on
octave:11> plot(x,t2)
octave:12> plot(x,t3)
octave:13> plot(x,t4)
```

```
octave:14> plot(x,t5)
octave:15> print("cheb.eps")
octave:16> diary off
```

Here is the graph: