MA228 NUMERICAL ANALYSIS - EXERCISE SHEET 1

Joe Tilley 1403899

January 21, 2016

Question 1

There are a variety of stopping criteria we can use for the bisection method. For a function $f(x)$ with a defined tolerance ϵ and a sequence of estimates $\{x_n\}_{n\geq 1}$, we will look at the following stopping criteria:

1. $|f(x_n)| < \epsilon$ – the residual size
2. $|x_n - x_{n-1}| < \epsilon$ – the increment size
3. $\dfrac{|x_n - x_{n-1}|}{|x_n|} < \epsilon$ – the relative change

That is, the algorithm will end once the criteria has been satisfied. The following is this code for the criteria 1.

```
function BisectionMethod(a, b, func, tol, nmax)
    % a is the lower estimate
    % b is the upper estimate
    % func is the function with one root between a and b, and must be a string ...
        , for example: 'x^2-2'
    % tolerance is the value at which when the stopping criteria reaches, the ...
        algorithm ends
    % nmax is the max number of iterations permitted

    %initial setup
    format long
    starta = a;                                  %remembers start values
    startb = b;
    func = strcat('@(x) ',func);                 %converts func into form of ...
        anonymous function
    f = str2func(func);                          %converts the string to an ...
        anonymous function
    stop = inf;                                  %stopping number set to ...
        infinity to bypass while loop
    estimates = [];                              %estimates is initially an ...
        empty list
    iter = 1;
    success = false;                             %unless changed, the method ...
        has failed
```

```
18
19      %whilst iteration number is below the maximum
20      while iter ≤ nmax
21          x = (a+b)/2;                                                %find the midpoint
22          stop = abs(f(x));                                           %calulate stopping ...
                number (error)
23          estimates = [estimates; iter x];                            %store this result
24          if f(x)==0 || stop < tol                                    %if we have found a ...
                root or stopping number below tolerance
25              disp('   Iteration           Root Estimate')            %title for results
26              disp(estimates)                                         %show iteration ...
                    number and root estimate
27              success = true;                                         %method has ...
                    succeeded, prevents error message
28              break                                                   %end the loop
29          end
30          if f(a)*f(x) > 0                                    %if lower estimate and ...
                midpoint are on same side of x-axis
31              a = x;                                          %let lower estimate be midpoint
32          else                                                %if lower estimate and ...
                midpoint are on opposite side of x-axis
33              b = x;                                          %let upper estimate be midpoint
34          end
35          iter = iter + 1;                                        %add 1 to iterations
36      end
37      if success == false                                         %method has failed, ...
            show error message
38          display('Method Failed')
39      end
40  end
```

If we wish to change the stopping criteria to stopping criteria 2 (the increment size), we need only change line 22 of the code. However, this criteria is only defined well for $n > 1$, so we must add in a simple if statement. So, line 22 is replaced by the following:

```
22  if iter > 1          %if the stopping number (error) will be well defined then ...
        calculate it (else will be left as infinite)
23      stop = abs(estimates(iter,2)-estimates(iter-1,2));
24  end
```

where there are two arguments in 'estimates' are because it is an $n \times 2$ matrix, as seen above in the output. The code can be modified similarly to create stopping criteria 3, keeping the majority of the code, but editing the definition of 'stop', and ensuring there are no division by zero errors. The code is as follows:

```
22  if iter > 1;                            %if the stopping number (error) will be ...
        well defined then calculate it (else will be left as infinite)
23      if estimates(iter,2) == 0           %prevents division by zero error
24          stop = inf;                     %ensures algorithm continues
25      else
26          stop = abs(estimates(iter,2)-estimates(iter-1,2))/abs(estimates(iter,2));
27      end
28  end
```

Question 2

As in Question 1, we can consider the same three stopping criteria. The following is this code for the criteria 1.

```
function NewtonMethod(x0, func, funcderiv, tol, nmax)
    % x0 is the initial value
    % func is the function with a root, and must be a string , for example: 'x^2-2'
    % funcderiv is the derivative of func, and must be a string , for example: ...
        '2*x'
    % tolerance is the value at which when the stopping criteria reaches, the ...
        algorithm ends
    % nmax is the max number of iterations permitted

    %initial setup
    format long
    func = strcat('@(x) ',func);                  %converts func into form of ...
        anonymous function
    f = str2func(func);                           %converts the string to an ...
        anonymous function
    funcderiv = strcat('@(x) ',funcderiv);        %converts func into form of ...
        anonymous function
    fdash = str2func(funcderiv);                   %converts the string to an ...
        anonymous function
    iter = 1;                                     %iteration number
    x(iter) = x0;                                 %x0 is the first value in list ...
        of estimates
    estimates = [iter-1 x(iter)];                 %first value is '0th iteration'
    stop = inf;                                   %stopping number set to ...
        infinity to bypass while loop
    success = false;                              %unless changed, the method has ...
        failed

    %whilst iteration number is below the maximum
    while iter ≤ nmax
        if abs(fdash(x(iter)))<eps                %prevents division by zero error
            break                                 %method has failed, exit while loop
        end
        x(iter+1) = x(iter) - f(x(iter))/fdash(x(iter));    %perform the next ...
            iteration
        estimates = [estimates; iter x(iter+1)];            %stores results
        stop = abs(f(x(iter+1)));                           %calulate stopping ...
            number (error)
        if f(x(iter+1))==0 || stop < tol                    %if we have found a ...
            root or stopping number below tolerance
            disp('   Iteration           Root Estimate')    %title for results
            disp(estimates)                                 %show iteration ...
                number and root estimate
            success = true;                                 %method has ...
                succeeded, prevents error message
            break                                           %exit the loop
        end
        iter = iter + 1;            %add 1 to iterations
    end
```

```
36     if success == false             %method has failed, show error message
37         display('Method Failed')
38     end
39 end
```

I have included the initial value x_0 in the results as the '0^{th} iteration'. Like with the bisection method, we can easily modify the code to change the stopping criteria by changing a single line (line 27). To implement stopping criteria 2, we use the following code Hence, Line 29 is replaced by the following:

```
27 if iter > 1                                      %if the stopping number (error) ...
       will be well defined then calculate it
28     stop = abs(estimates(iter,2)-estimates(iter-1,2));
29 end
```

where the two arguments in 'estimates' are because it is an $n+1 \times 2$ matrix, as seen above in the output. Stopping criteria 3 can be modified similarly, keeping the majority of the code, but editing the definition of 'stop', and ensuring there are no division by zero errors. The code is as follows:

```
27 if iter > 1;             %if the stopping number (error) will be well defined ...
       then calculate it
28     if abs(estimates(iter,2)) < eps        %prevents division by zero error
29         stop = inf;                        %ensures algorithm continues
30     else
31         stop = abs(estimates(iter,2)-estimates(iter-1,2))/abs(estimates(iter,2));
32     end
33 end
```

Question 3

The function $f(x) = e^{-x} - x$ has a root in the range $(0,1)$. In fact, the exact value of the root is $W(1) = 0.567143290409784$, where W is the Lambert W function. We can perform both bisection method and Newton method to find an approximation to this root. We must also specify the stopping criteria we must use for each of these methods. I have chosen to use stopping criteria 1, the residual size, for both methods. For the function $f(x) = e^{-x} - x$, and so $f'(x) = -e^{-x} - 1$, in the range $(0,1)$ (so $a = 0$ and $b = 1$), if we let $\epsilon = 10^{-3}$, and $n_{max} = 100$ (very large so the algorithm does not fail and we will assume this will be n_{max} from now on), the result of the bisection method is as follows

```
>> BisectionMethod(0,1,'exp(-x)-x',10^(-3), 100)
   Iteration           Root Estimate
   1.000000000000000   0.500000000000000
   2.000000000000000   0.750000000000000
   3.000000000000000   0.625000000000000
   4.000000000000000   0.562500000000000
   5.000000000000000   0.593750000000000
   6.000000000000000   0.578125000000000
   7.000000000000000   0.570312500000000
```

```
  8.000000000000000   0.566406250000000
  9.000000000000000   0.568359375000000
 10.000000000000000   0.567382812500000
```

Using Newton method for the same function and using its derivative, but with initial value $x_0 = 0$ (in fact, any value in the range 0 to 1 can be used, though I have chosen this somewhat randomly) and using the same tolerance of $\epsilon = 10^{-3}$, we get

```
>> NewtonMethod(0,'exp(-x)-x','-exp(-x)-1',10^(-3),100)
  Iteration           Root Estimate
                  0                   0
  1.000000000000000   0.500000000000000
  2.000000000000000   0.566311003197218
  3.000000000000000   0.567143165034862
```

Using a smaller tolerance, $\epsilon = 10^{-7}$, we will get more iterations. The code is the following using bisection method

```
>> BisectionMethod(0,1,'exp(-x)-x',10^(-7), 100)
  Iteration           Root Estimate
  1.000000000000000   0.500000000000000
  2.000000000000000   0.750000000000000
  3.000000000000000   0.625000000000000
  4.000000000000000   0.562500000000000
  5.000000000000000   0.593750000000000
  6.000000000000000   0.578125000000000
  7.000000000000000   0.570312500000000
  8.000000000000000   0.566406250000000
  9.000000000000000   0.568359375000000
 10.000000000000000   0.567382812500000
 11.000000000000000   0.566894531250000
 12.000000000000000   0.567138671875000
 13.000000000000000   0.567260742187500
 14.000000000000000   0.567199707031250
 15.000000000000000   0.567169189453125
 16.000000000000000   0.567153930664063
 17.000000000000000   0.567146301269531
 18.000000000000000   0.567142486572266
 19.000000000000000   0.567144393920898
 20.000000000000000   0.567143440246582
 21.000000000000000   0.567142963409424
 22.000000000000000   0.567143201828003
 23.000000000000000   0.567143321037292
```

And again for Newton method, using the same tolerance

```
>> NewtonMethod(0,'exp(-x)-x','-exp(-x)-1',10^(-7), 100)
  Iteration           Root Estimate
                  0                   0
  1.000000000000000   0.500000000000000
  2.000000000000000   0.566311003197218
  3.000000000000000   0.567143165034862
```

```
    4.000000000000000   0.567143290409781
```

Lastly, I will set the tolerance to be incredibly small, $\epsilon = 10^{-15}$, the bisection method gives the result (I have cut a lot of iterations out to save paper)

```
>> BisectionMethod(0,1,'exp(-x)-x',10^(-15), 100)
    Iteration           Root Estimate
    1.000000000000000   0.500000000000000
    2.000000000000000   0.750000000000000
    3.000000000000000   0.625000000000000
    4.000000000000000   0.562500000000000
    5.000000000000000   0.593750000000000
    ...
   46.000000000000000   0.567143290409788
   47.000000000000000   0.567143290409781
   48.000000000000000   0.567143290409785
   49.000000000000000   0.567143290409783
   50.000000000000000   0.567143290409784
```

And again for Newton method, using the same tolerance, gives the results

```
NewtonMethod(0,'exp(-x)-x','-exp(-x)-1',10^(-15), 100)
    Iteration           Root Estimate
                    0                   0
    1.000000000000000   0.500000000000000
    2.000000000000000   0.566311003197218
    3.000000000000000   0.567143165034862
    4.000000000000000   0.567143290409781
    5.000000000000000   0.567143290409784
```

Question 4

Looking at the results, it is extremely obvious that the Newton method converges to the root quicker than the bisection method, ($\epsilon = 10^{-3}$ gave 10 to 3 iterations, $\epsilon = 10^{-7}$ gave 23 to 4 iterations, and $\epsilon = 10^{-15}$ gave 50 to 5 iterations in bisection to Newton method respectively). While the bisection method converges linearly (as proven in lectures), the Newton method does not. Newton method converges quadratically, $O(n^2)$, and this can be proven via the taylor series. If we let function f have a root α (so $f(\alpha) = 0$), then we can expand $f(\alpha)$ by the taylor series at x_n (assuming it has a continuous second derivative near α) as

$$0 = f(\alpha) = f(x_n) + f'(x_n)f(\alpha - x_n) + \frac{f''(\xi_n)}{2}(\alpha - x_n)^2$$

for some ξ_n between α and x_n. Manipulating this gives

$$\frac{f(x_n)}{f'(x_n)} + (\alpha - x_n) = -\frac{f''(\xi_n)}{2f'(x_n)}(\alpha - x_n)^2$$

The error ϵ_n is given by $|x_n - \alpha|$, and noting that by the Newton method, we have let

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

then we can rewrite the manipulated taylor series as

$$\alpha - x_{n+1} = -\frac{f''(\xi_n)}{2f'(x_n)}(\alpha - x_n)^2 \quad \Rightarrow \quad \epsilon_{n+1} = \left|\frac{f''(\xi_n)}{2f'(x_n)}\right| \epsilon_n^2$$

after having considered absolute values, hence proving the quadratic convergence of Newton method.

If we wish to explore the rate of convergence for each of the algorithms in my implementation, we should consider plotting the graph of ϵ_n on the y-axis against ϵ_{n-1} on the x-axis for the function $f(x) = e^{-x} - x$ from Question 3. We will take the tolerance $\epsilon =$eps, the smallest positive computer number, to get the maximum number of iterations possible before the algorithm terminates. Furthermore, we can find the exact error at the n^{th} iteration by calculating $|x_n - W(1)|$, and storing these values, then generating our plots from this. Since the bisection method follows the relationship $\epsilon_n = \frac{\epsilon_{n-1}}{2}$ (proven in lectures but can be seen intuitively by halving the range each iteration), we expect a straight line of gradient 2, whereas Newton method will produce a quadratic curve.

The problem is that producing this graph leads to a cluster of results around $(0, 0)$, since the errors tend to 0, making the graph indecipherable. Instead, we can take logs to show our results. By plotting $\log(\epsilon_n)$ on the y-axis and $\log(\epsilon_{n-1})$ on the x-axis, for bisection method we have $\log(\epsilon_n) = \log(\epsilon_{n-1}) + \log(\frac{1}{2})$ by taking logs of bisection's error relationship, so we expect a straight line of gradient 1 (which would display linearity). We can show Newton method's quadratic convergence, as we expect a gradient of very near 2, since taking logs of the error relationship gives $\log(\epsilon_n) = 2\log(\epsilon_{n-1}) + 2\log(\left|\frac{f''(\xi_n)}{2f'(x_n)}\right|)$ for some ξ_n between the root and x_n. Figure 1 is the plot of these results for $f(x) = e^{-x} - x$ with a tolerance of eps.

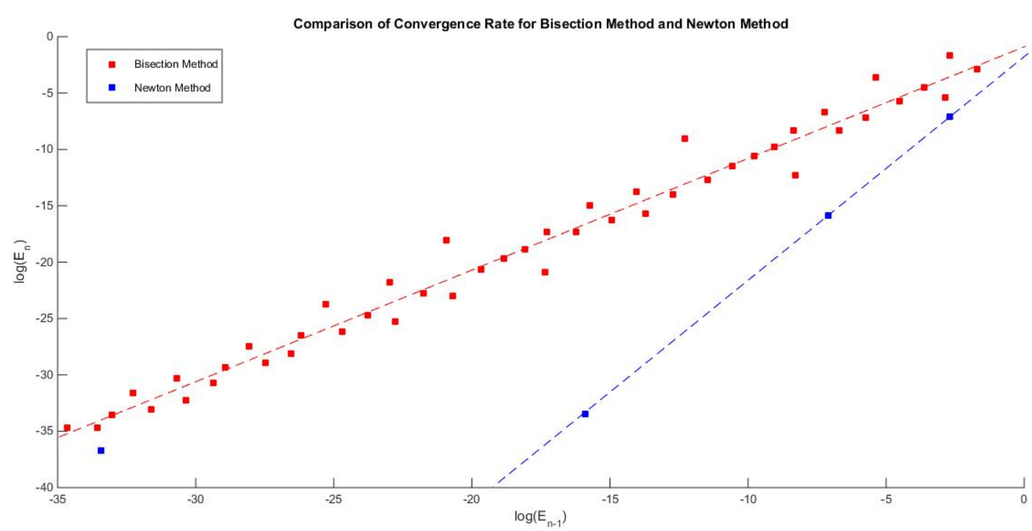

Figure 1: A graph displaying the rate of convergence for bisection method and Newton method

As we can see from the graph, the bisection method asymptotically tends towards a line of gradient 1, displaying its linear convergence, and the Newton method forms points along a line of gradient 2 (the outlier point is caused by the error of the initial value, so can be excluded), so displays its quadratic convergence.