Project 1 Report: Numerical Methods

Jacky Kheang

September 23, 2019

The Objective of this project is to use numerical methods to find the roots of the following problem:

$$x^2 - 5x + 4 = 0$$

Where the true values are $x = 1$ and $x = 4$. We will assume an error tolerance of $1e-4$ and use a minimum of 8 significant digits. In this report, we will employ five different numerical methods:

1. Bisection Method; [-1.1, 2.5] and [2.0,5.3]

2. Regula-Falsi Method; [-1.1, 2.5] and [2.0,5.3]

3. Fixed Point Method

4. Newton-Raphson Method

5. Secant Method

The bisection and Regula-Falsi methods are classified "bracketing" methods. They find the roots of a function within a certain interval. They have a linear rate of convergence. The other three methods are known as "open" methods of finding a root. The rate of convergence for the fixed point method is linear, Newton-Raphson method is quadratic, and the secant method has a convergence rate of 1.6.

Analysis

The first method we will analyze is the Bisection Method of solving roots. In order to use this method, the function must first be continuous on $[a, b]$, and have a root such that:

$$f(a) * f(b) < 0$$

We must first find the midpoint of $[a, b]$ which is $c_1 = \frac{1}{2}(b - a)$. If the product of $f(a)$ and $f(c_1)$ is negative, then the root will be located in that interval. If not, then the root will be in the interval $[c_1, b]$. The boundaries of our new interval will be our new $[a, b]$. For our second iteration we will conduct the same operations as we did before; Find midpoint c_n and then find the products of the function values. We will keep iterating until $f(c_n) = 0$ or we reach the error tolerance which in our case is $1e - 4$. The function file is as follows:

```
function c = Bisection_Method(f, a, b, n, err)

% c = approximate root calculated
% f = anonymous function f(x)
% a = lower booundary
% b = upper boundary
% n = number of iterations
% err = error tolerance
if nargin < 5 || isempty(err), err = 1e-4; end
if nargin < 4 || isempty(n), n = 20; end
if f(a)*f(b) > 0
    c = 'failure';
    return
end
disp('steps    lower      upper      midpoint   error');
for k=1:n
    c=(a+b)/2; %the first midpont
    if f(c) == 0  %conditional if root is found
        return
    end
    fprintf('%3i    %11.6f%11.6f%11.6f%11.6f\n',k,a,b,c,(b-a)/2);
    if (b-a)/2 < err, return %iterative operation
    end
    if f(b)*f(c) > 0 %selecting bracket for next iteration
        b = c; else a = c; end
end
```

Our error tolerance is calculated by taking the difference of upper and lower boundaries and dividing two. Because the boundaries in our iterative function are supposed to be getting smaller, our answer will become more accurate. Our tolerance is set to 1e-4, meaning if the boundaries are smaller than that value, our function will terminate.
When we run this function into an anonymous function:

```
f=@(x)(x^2-5*x+4);
q = Bisection_Method(f, -1.1, 2.5, [], 1e-4);
```

```
fprintf('Our approximated root is %11.8f'\n,q)
p = Bisection_Method(f, 2.0, 5.3, [], 1e-4);
fprintf('Our approximated root is %11.8f'\n,p)
```

We get these return values:

```
steps     lower       upper       midpoint    error
  1      -1.100000    2.500000    0.700000    1.800000
  2       0.700000    2.500000    1.600000    0.900000
  3       0.700000    1.600000    1.150000    0.450000
  4       0.700000    1.150000    0.925000    0.225000
  5       0.925000    1.150000    1.037500    0.112500
  6       0.925000    1.037500    0.981250    0.056250
  7       0.981250    1.037500    1.009375    0.028125
  8       0.981250    1.009375    0.995312    0.014062
  9       0.995312    1.009375    1.002344    0.007031
 10       0.995312    1.002344    0.998828    0.003516
 11       0.998828    1.002344    1.000586    0.001758
 12       0.998828    1.000586    0.999707    0.000879
 13       0.999707    1.000586    1.000146    0.000439
 14       0.999707    1.000146    0.999927    0.000220
 15       0.999927    1.000146    1.000037    0.000110
 16       0.999927    1.000037    0.999982    0.000055
Our approximated root is  0.99998169
steps     lower       upper       midpoint    error
  1       2.000000    5.300000    3.650000    1.650000
  2       3.650000    5.300000    4.475000    0.825000
  3       3.650000    4.475000    4.062500    0.412500
  4       3.650000    4.062500    3.856250    0.206250
  5       3.856250    4.062500    3.959375    0.103125
  6       3.959375    4.062500    4.010937    0.051562
  7       3.959375    4.010937    3.985156    0.025781
  8       3.985156    4.010937    3.998047    0.012891
  9       3.998047    4.010937    4.004492    0.006445
 10       3.998047    4.004492    4.001270    0.003223
 11       3.998047    4.001270    3.999658    0.001611
 12       3.999658    4.001270    4.000464    0.000806
 13       3.999658    4.000464    4.000061    0.000403
 14       3.999658    4.000061    3.999860    0.000201
 15       3.999860    4.000061    3.999960    0.000101
 16       3.999960    4.000061    4.000011    0.000050
Our approximated root is  4.00001068
```

The next method we will be using is the Regula-Falsi method. This method shares many similarities to the bisection method as they both find the root within a given interval. The technique for it is geometrical in nature. If $f(a_1) * f(b_1) < 0$, then we connect the two points with a straight line where c_1 is the x-intercept of that line. We can find the intercept with:

$$c_1 = b_1 - \frac{b_1 - a_1}{f(b_1) - f(a_1)} f(b_1) = \frac{a_1 f(b_1) - b_1 f(a_1)}{f(b_1) - f(a_1)}$$

By generalizing the above equation we get:

$$c_n = \frac{a_n f(b_n) - b_n f(a_n)}{f(b_n) - f(a_n)}, n = 1, 2, 3, ...$$

The error tolerance or condition for terminating our iterations is:

$$|c_{n+1} - c_n| < \epsilon = 1e - 4$$

The function for the Regula-Falsi Method:

```
function [r, k] = Regula_Falsi(f, a, b, n, err)

if nargin < 5 || isempty(err), err = 1e-4; end
if nargin < 4 || isempty(n), n = 20; end
c = zeros(1,n);
if f(a)*f(b) > 0          %determining if the function has a root
    r = 'failure';          %on the x-intercept and is continuous
    return
end
disp('steps   lower     upper     error');
for k = 1:n
    c(k) = (a*f(b)-b*f(a))/(f(b)-f(a));   %iteration operation
    if f(c(k)) == 0     %conditional if the root is reached
        return
    end
    if f(b)*f(c(k)) > 0             %determining if continuous
        b = c(k); %checking if root is in one of the brackets
    else a = c(k); %if it is not, then it is the other bracket
    end
    c(k+1) = (a*f(b)-b*f(a))/(f(b)-f(a));   %iteration operation
    fprintf('%2i %11.8f %11.8f %11.8f\n',k,a,b,abs((c(k+1)-c(k))))
      ;
    if abs(c(k+1)-c(k)) < err   %checking error tolerance
        r = c(k+1);
        return
    end
end

f=@(x)(x^2-5*x+4);
format long
```

```
[r1,k1]=Regula_Falsi(f,-1.1,2.5,[],1e-4);
fprintf('The approximated root is %11.8f. It took %3i iterations\n
   ',r1,k1);
[r2,k2]=Regula_Falsi(f,2.0,5.3,[],1e-4);
fprintf('The approximated root is %11.8f. It took %3i iterations\n
   ',r2,k2);
steps   lower      upper     error
 1    -1.100000    1.875000  0.44008876
 2    -1.100000    1.434911  0.23913499
 3    -1.100000    1.195776  0.11194441
 4    -1.100000    1.083832  0.04873596
 5    -1.100000    1.035096  0.02054450
 6    -1.100000    1.014551  0.00854249
 7    -1.100000    1.006009  0.00353172
 8    -1.100000    1.002477  0.00145666
 9    -1.100000    1.001021  0.00060021
10    -1.100000    1.000420  0.00024722
11    -1.100000    1.000173  0.00010181
12    -1.100000    1.000071  0.00004192
The approximated root is  1.00002935. It took  12 iterations
steps   lower      upper     error
 1     2.869565    5.300000  0.66678595
 2     3.536351    5.300000  0.30653509
 3     3.842886    5.300000  0.10781288
 4     3.950699    5.300000  0.03422308
 5     3.984922    5.300000  0.01050334
 6     3.995426    5.300000  0.00319000
 7     3.998616    5.300000  0.00096576
 8     3.999581    5.300000  0.00029210
 9     3.999873    5.300000  0.00008832
The approximated root is  3.99996173. It took   9 iterations
```

The next method we will use is the fixed point method. This is an "open" method of finding the roots of a function $f(x)$. First we rewrite $f(x) = 0$ as $x = g(x)$ where $g(x)$ is our iterative function. The point of intersection between $y = g(x)$ and $y = x$ is the fixed point of $g(x)$, which is also the root of $f(x)$. It should also be known that $f(x) = 0$ can have multiple iterative functions. The function file is as follows:

```
function [r, n]=FixedPoint_Method(g, x1, t, err)
    %r=approx fixed point of g(x)
    %n=number of steps/iterations
    %g=an anoonymous function g(x)
    %x=initial point
    %t=maximum iterations
    %err=error tolerance
if nargin<4 || isempty(err), err=1e-4;
end
if nargin <3|| isempty(t), t=20;
end
x(1)=x1;
disp('steps  approximation    error')
for n=1:t
    x(n+1)=g(x(n)); %iterative function
    fprintf(' %3i %11.8f  %11.8f\n', (n), x(n+1),abs(0.5*(x(n+1)-x
       (n))));
    if abs(x(n+1)-x(n))<err %checking error tolerance
       r=x(n+1);
       return
    end
end
r='failure';
```

For the iterative function, we used

$$g(x)_1 = \frac{1}{5}x^2 + 4$$
$$g(x)_2 = 5 - \frac{4}{x}$$

We must run the function file into an anonymous function

```
g1=@(x)(1/5*(x^2+4));
g2=@(x)(5-4/x);
format short
[r1,n1]=FixedPoint_Method(g1,2,[],1e-4);
fprintf('The approximated root is %11.8f. It took %3i iterations\n
   ',r1,n1);
[r2,n2]=FixedPoint_Method(g2,-1,[],1e-4);
fprintf('The approximated root is %11.8f. It took %3i iterations\n
   ',r2,n2);
```

And receive the following tabulated values.

```
steps  approximation    error
   1  1.60000000   0.20000000
   2  1.31200000   0.14400000
   3  1.14426880   0.08386560
   4  1.06187022   0.04119929
   5  1.02551367   0.01817827
   6  1.01033566   0.00758901
   7  1.00415563   0.00309001
   8  1.00166571   0.00124496
   9  1.00066684   0.00049943
  10  1.00026682   0.00020001
  11  1.00010674   0.00008004
  12  1.00004270   0.00003202
The approximated root is  1.00004270. It took  12 iterations
steps  approximation    error
   1  9.00000000   5.00000000
   2  4.55555556   2.22222222
   3  4.12195122   0.21680217
   4  4.02958580   0.04618271
   5  4.00734214   0.01112183
   6  4.00183217   0.00275499
   7  4.00045783   0.00068717
   8  4.00011445   0.00017169
   9  4.00002861   0.00004292
The approximated root is  4.00002861. It took   9 iterations
```

The next method is Newton-Raphson method. In order to use this method, $f(x) = 0$ must be continuous. We start with an initial guess x_1. Then we draw a tangent line at $(x_1, f(x_1))$ on the curve and let its x-intercept be x_2. The general form to find the x-intercept can be found with:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, n = 1, 2, 3, ...$$

Where x_1=initial point. The function file is as follows:

```
function[r, n]=Newton_Method(f, x1, err, t)
%f equals continuous function f(x)
%x1 is initial value
%t is max iterations
%err is error tolerance
%r is approx root
%n is iterations

if nargin<4||isempty(t),t=20;
end
if nargin<3||isempty(err), err=1e-4;
end
disp('step  initial     x(n+1)    error')
fp=matlabFunction(diff(f)); %f'(x)
f=matlabFunction(f); %f(x)
x=zeros(1,t+1);
x(1)=x1;
for n=1:t
    if fp(x(n))==0 %slope=0, method fails
        r='failure';
        return
    end
    x(n+1)=x(n)-f(x(n))/fp(x(n)); %iterative operation
    fprintf('%2i %11.6f %11.6f %11.8f\n',n,x(n),x(n+1),abs((x(n+1)
       -x(n))));
    if abs(x(n+1)-x(n))<err %checking error tolerance
        r=x(n+1);
        return
    end
end
```

Putting the function file into an anonymous function:

```
syms x;
f=x^2-5*x+4;
[r1, n1]=Newton_Method(f, 0);
fprintf('The approximated root is %11.8f. It took %3i iterations\n
    ',r1,n1);
[r2, n2]=Newton_Method(f,8);
```

```
fprintf('The approximated root is %11.8f. It took %3i iterations\n
   ',r2,n2);
```

We will receive the following values:

```
step  initial     x(n+1)    error
 1    0.000000    0.800000  0.80000000
 2    0.800000    0.988235  0.18823529
 3    0.988235    0.999954  0.01171893
 4    0.999954    1.000000  0.00004578
The approximated root is  1.00000000. It took   4 iterations
step  initial     x(n+1)    error
 1    8.000000    5.454545  2.54545455
 2    5.454545    4.358042  1.09650350
 3    4.358042    4.034497  0.32354488
 4    4.034497    4.000388  0.03410931
 5    4.000388    4.000000  0.00038771
 6    4.000000    4.000000  0.00000005
The approximated root is  4.00000000. It took   6 iterations
```

The Newton-Raphson method is fast but it is also unstable.

The last method we will use is the secant method. It has a similar idea the Newton-Raphson method, except instead taking the derivative it uses the difference quotient as shown below:

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n), n = 2, 3, 4..., x_1, x_2, = \text{initial points}$$

The function file for the secant method is as follows:

```
ffunction[r,n] = Secant_Method(f,x1,x2,err,t)
%f is function
%x1 is first point
%x2 is second point
%err is error
%n is #iterations
if nargin<5 || isempty(t), t=20;
end
if nargin<4 || isempty(err), err=1e-4;
end
f=matlabFunction(f);
x=zeros(1,t+1);
disp('steps    x1           x2          error')
for n=2:t
    if x1==x2 %slope req 2 diff points
        r='failure';
        return
    end
    x(1)=x1;
    x(2)=x2;
    x(n+1)=x(n)-((x(n)-x(n-1))/(f(x(n))-f(x(n-1))))*f(x(n));
    fprintf('%3i %11.8f %11.8f %11.8f\n',n,x(n),x(n+1),abs(x(n+1)-
       x(n)) )

    if abs(x(n+1)-x(n))<err %error check
        r=x(n+1);
        return
    end
end
```

Putting the function file into an anonymous function:

```
syms x
f=x^2-5*x+4;
[r1,n1]=Secant_Method(f,-6,2);
fprintf('The approximated root is %11.8f. It took %3i iterations\n
   ',r1,n1);
[r2,n2]=Secant_Method(f,3,6);
fprintf('The approximated root is %11.8f. It took %3i iterations\n
   ',r2,n2);
```

Return values:

```
steps    x1          x2          error
  2  2.00000000  1.77777778  0.22222222
  3  1.77777778  0.36363636  1.41414141
  4  0.36363636  1.17314488  0.80950851
  5  1.17314488  1.03181523  0.14132965
  6  1.03181523  0.99802914  0.03378609
  7  0.99802914  1.00002111  0.00199198
  8  1.00002111  1.00000001  0.00002110
The approximated root is  1.00000001. It took   8 iterations
steps    x1          x2          error
  2  6.00000000  3.50000000  2.50000000
  3  3.50000000  3.77777778  0.27777778
  4  3.77777778  4.04878049  0.27100271
  5  4.04878049  3.99616491  0.05261558
  6  3.99616491  3.99993856  0.00377365
  7  3.99993856  4.00000008  0.00006152
The approximated root is  4.00000008. It took   7 iterations
```

Concluding Thoughts

The purpose of this project is familiarize ourselves with the numerical methods of finding the root of a function. It is important to understand how each method works and how to use it should we need to find the roots of a function that cannot be found analytically. They each have characteristics that are useful in their own right such as rate of convergence. For example, the bracketed methods have a rate of convergence of one, but are relatively stable. Meanwhile the Newton-Raphson has a rate of convergence of two, but it can be unstable.