**Lab 7: Solving Nonlinear Equations using**

**the Newton-Raphson Method**

**Background**

Suppose we want to solve the equation f(x) = 0. Let xo be an initial guess for the solution. The Newton-Raphson method uses this initial guess to iterate to a “better” solution as follows:

x1 = x0 – f(xo)/f’(xo)

The updated guess, x1, can then be used to iterate to an even “better” solution as follows:

x2 = x1 – f(x1)/f’(x1)

This pattern continues so that the new guess depends on the previous guess as follows:

xn+1 = xn – f(xn)/f’(xn)

The iterative algorithm runs until |f(xn)| < ε (some small specified value) or until the number of iterations exceeds some specified value indicating that the algorithm doesn’t appear to be converging.

So, the algorithm works as follows:

1. Make an initial guess, x1
2. Is │f(x1)│ < ε? That is, is the guess close enough to a solution? If yes, finish. If not, go to step 3.
3. Iterate to a new guess using the Newton-Raphson algorithm.
4. Is │f(xn+1)│ < ε? If yes, finish. If not, return to step 3 unless the number of iterations exceeds some specified maximum.

The Newton-Raphson method is very useful in solving both single equations and systems of non-linear equations. In this lab, we will be applying to algorithm to find solutions to single equations only.

**Note: Do not use symbolic expressions in any of your scripts for this lab.**

**Part A**

Suppose we want to estimate the fifth root of 80.5. This is equivalent to solving the equation f(x) = x5 – 80.5 = 0. A decent initial guess might be 2.5 since 25 = 32 and 35 = 243.

For this particular function, work out what the update algorithm would be and complete the expression below:

|  |
| --- |
| Xn+1 = X\_n - (x^5 – 80.5)/(5\*x^4) |

Complete Table 1 for an initial guess of 2.5 then check values with your T.A.

**Table 1: Estimate of 5th Root of 80.5**

|  |  |  |
| --- | --- | --- |
| **Number of Iterations** | **5th Root Estimate Using**  **Newton Raphson** | **Evaluate Accuracy:**  │**f(xn)** │ **= ?** |
| 0 | X1 = 2.5 | 17.1563 |
| 1 | X2 =2.4121600000 | 1.1639828506 |
| 2 | X3 =2.4052837425 | 0.0066173571 |
| 3 | X4 =2.4052442013 | 0.0000002176 |

After the 3rd iteration, the estimate of the fifth root of 80.5 is accurate to within how many places behind the decimal point?

|  |
| --- |
| It is accurate within 6 decimal places behind the decimal. |

What if the initial estimate for the fifth root was zero (look back at your update equation)?

|  |
| --- |
| The derivative would be zero when plugging in the x value and you would be dividing by zero. |

So what does this mean you have to be check before updating your estimate?

|  |
| --- |
| You have to check that the derivative is not equal to zero in order to continue updating the estimate. |

**Part B**

Write a MATLAB script that utilizes the Newton Raphson algorithm to search for the fifth root of any number entered by the user to within four places behind the decimal point (i.e., │f(xn)│ < 0.0001). The script should do the following:

* Prompt the user for the number to find the 5th root of
* Prompt the user for an initial guess
* Use a while loop to iterate through the Newton Raphson algorithm as long as the required accuracy hasn’t been reached (i.e., abs(f(xn)) > 0.0001) **and** the number of iterations is below 100. Save all of the estimates in a vector.
* Within the while loop, before updating the estimate, check to see if f’(xn)is zero. If it is, modify the value of xn just a bit to avoid division by zero in the Newton Raphson update equation. For example, you could just add a small value (xn = xn + 0.01) before the update whenever the derivative at xn is zero.
* After the end of the while loop, add an fprintf statement to display the number of iterations performed.
* If the number of iterations is less than 100, use an fprintf statement to output the final estimate (with five places behind the decimal point) for the 5th root of the number entered by the user. Otherwise, tell the user that the algorithm failed to converge and a better initial guess should be used.
* Plot each of the estimates on the y-axis. On the x-axis, just put 0:N where N is the number of iterations. Include axis labels and a title.

Test your code for each of the cases shown in Table 2 and enter your results into the table as well. Paste plots for the 2nd and 3rd test case only.

**Table 2: Estimated 5th Roots Using Newton Raphson**

|  |  |  |  |
| --- | --- | --- | --- |
| **Number** | **Initial Guess** | **Final Estimate of 5th Root** | **Number of Iterations** |
| **80.5** | **2.5** | **2.405244** | **3** |
| **80.5** | **1** | **2.405244** | **13** |
| **80.5** | **0** | **2.405244** | **96** |
| **43,000,000** | **30** | **3.362743** | **5** |
| **43,000,000** | **10** | **3.362743** | **20** |
| **43,000,000** | **0** | **Could Not Converge** | **100** |

**PASTE PLOT for Number = 80.5 and Initial Guess = 1**



**PASTE PLOT for Number = 80.5 and Initial Guess = 0**



How does the initial guess affect the number of iterations?

|  |
| --- |
| The farther the guess is from the actual fifth root the more iterations it takes for the program to get closer to the actual value. |

**PASTE MATLAB CODE FOR PART B HERE:**

%Kyle O'Connor

%

%Lab 7

%

%2/24/2016

%

home;

clear all;

clc;

num = input('What number would you like to find the 5th root of?');

guess = input('What is your initial guess?');

count = 0;

if guess == 0;

guess = guess + 0.01;

end

func = guess^5 - num;

deriv = 5\*guess^4;

while((abs(func) > 0.0001) && (count < 100))

guess = guess - (func/deriv);

func = guess^5-num;

deriv = 5\*guess^4;

count = count + 1;

approx(count) = func;

if deriv == 0

guess = guess + 0.01;

else

deriv = 5\*guess^4;

end

end

fprintf('Iterations: %i\n', count);

if count < 100

fprintf('Estimate: %i\n', guess)

else

fprintf('Could not converge with current guess')

end

plot(1:count,approx,'ro')

**Part C WORK THIS ON YOUR OWN!**

Each Bearcat widget sells for $4.00. The cost ($) to produce x Bearcat widgets is:

Use the Newton Raphson algorithm to find out how many Bearcat widgets should be produced to break even. In other words, find x such that f(x) = Cost – Sales = 0.

Complete the equation for f(x)

|  |
| --- |
| f(x) = (1000 + 2\*x + 3\*x^(2/3)) – 4x |

Complete the equation for the Newton Raphson Update:

|  |
| --- |
| xn+1 = xn – (1000 + 2\*x + 3\*x^(2/3)-4\*x)/(2+2x^(-1/3)) |

Write a MATLAB script to solve for the number of widgets needed to break even then run your script.

How many Bearcat widgets are needed to break even?

|  |
| --- |
| # Bearcat Widgets = 608 widgets |

Check your answer by calculating Cost and Sales.

|  |
| --- |
| Cost = $2340.40 Sales = $2340.40 |

**PASTE MATLAB CODE FOR PART C HERE:**

%Kyle O'Connor

%

%Lab 7

%

%2/24/2016

%

home;

clear all;

clc;

x = 2;

func = (1000 + (2\*x) + 3\*(x^(2/3))-(4\*x));

deriv = (2\*(x^(-1/3))-2);

x = x - (func/deriv);

counter = 1;

while(abs(func) > 0.0001 && counter < 100)

counter = counter+1;

x = x - (func/deriv);

func = (1000 + (2\*x) + 3\*(x^(2/3))-(4\*x));

deriv = (2\*(x^(-1/3))-2);

end

cost = (1000 + (2\*x) + 3\*(x^(2/3)));

sales = 4\*x;

profit = cost - sales;

fprintf('Answer check sales - cost = %0.2f', profit)