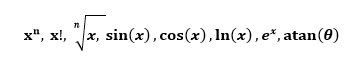
**Lab 7: Iterative Algorithms Using While Loops**

**Introduction:**

Computers, microprocessors, and calculators can add, subtract, multiply, and divide. These operations are done in binary, of course, since numbers are stored in binary.

With this rather limited set of operations, how does your calculator or MATLAB determine values for these functions?



To calculate values for these functions, we need an iterative algorithm or a numerical method that only requires the basic arithmetic operations of addition, subtraction, multiplication, and division. In this lab, we will investigate an iterative algorithm for cube root and an iterative algorithm for ln(x).

**Part A: Cube Root**

There are many, many algorithms for computing the cube root of a number. We will use the Newton-Raphson algorithm. Like all iterative algorithms, this algorithm requires an initial estimate for the solution. The estimate is evaluated for accuracy and if the estimate isn’t accurate enough, the estimate is updated. The process continues (loops) until the estimate is finally accurate enough. The algorithm is illustrated in the flow chart on the following page.

1. In order to understand the algorithm, work through the following table by hand. Assume we want to find the cube root of 27, so **Number = 27**. Assume an initial Estimate = 1.

**Table 1: Newton-Raphson Algorithm**

|  |  |  |
| --- | --- | --- |
| **Initial** | Estimate = 1 *(Clearly not a great guess!)* | Error = abs(27-1^3) = 26 |
|  | **Estimate = 1/3\*(2\*Estimate + Number/Estimate2)** | **Error = abs(Number – Estimate3)** |
| **Iter. 1** | Estimate = 9.6667 | Error = 876.296 |
| **Iter. 2** | Estimate =6.541 | Error = 252.826 |
| **Iter. 3** | Estimate =4.571 | Error =68.508 |
| **Iter. 4** | Estimate =3.478 | Error =15.074 |
| **Iter. 5** | Estimate =3.063 | Error =1.73 |

YES

NO

Input number:

Number

Input initial estimate:

Estimate

Calculate Absolute Value Error:

Error = abs(Number – Estimate3 )

Initialize Count to 0

While Loop

Is Error too big?

Update the Estimate:

Estimate = 1/3\*(2\*Estimate + Number/Estimate2)

Calculate Absolute Value of Error:

Error = abs(Number – Estimate3 )

Output the following:

Estimate of Cube Root

Actual Cube Root

Count (Number of Times thru Loop)

Increment Count by 1

1. Create a new script file

* Using the flow chart as a reference for writing your script, implement the Newton Raphson algorithm for finding the cube root of a number.
* Assume we want the Error no larger than 0.000001 (1e-6).
* Use fprintf statement(s) to output the estimate of the cube root, and the actual cube root (all with 3 places behind decimal point) and the number of times through the while loop (integer value).

1. Test your program using the test cases shown in the table below.

**Table 2: Test Inputs and Results**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Number** | **Initial Estimate** | **Actual Cube Root of Number** | **Estimate of Cube Root** | **Number of Times Through Loop** |
| 27 | 1 | 3.000 | 3.000 | 8 |
| 27 | 2 | 3.000 | 3.000 | 5 |
| 27 | 3 | 3 | 3 | 3 |
| 729345 | 1 | 90.014 | 90.014 | 25 |
| 729345 | 50 | 90.014 | 90.014 | 6 |
| 729345 | 90 | 90.014 | 90.014 | 2 |
| 729345 | 150 | 90.014 | 90.014 | 6 |

1. One of the important considerations of an estimation algorithm like this is how many iterations it takes to get an estimate within the required accuracy.

**Observations: How does the initial guess affect the Number of Iterations?**

**The farther away the initial guess is from the actual cube root of the number, the more loops it will take to get the error down to near zero. So, the closer the number is to the actual cube root, the loop will have to run less amount of times.**

**PASTE SCRIPT FOR PART A HERE:**

clear;clc;

Number = input('What number do you want? ');

Estimate = input('What is the initial estimate number? ');

Error = abs(Number - Estimate^3);

Count = 0;

Actual = Number^(1/3);

while Error > 0.000001

Estimate = 1/3\*((2\*Estimate) + (Number/Estimate^2));

Error = abs((Number)-(Estimate^3));

Count = Count+1;

end

fprintf('The estimate for the cube root of %i is %0.3f\n', Number, Estimate)

fprintf('The actual cube root for %i is %0.3f\n', Number, Actual)

fprintf('Count: %i \n', Count)

**FUN FACT:** Creating fast and accurate algorithms for determining values of functions such as the roots of numbers is a very active area of research. In 2011, Evan O’Dorney (seventeen years old at the time) won first prize in the 2011 Intel Science Talent Search contest. First prize was $100,000 by the way. His project was comparing two different algorithms for computing square root, one that was known to be very, very accurate and a second that was known to be very fast.

**Part B: Natural Log**

The Taylor Series for the ln(x) for any x in the range 0 < x < 2 is given by:

Obviously, we can’t add an infinite number of terms together, but we will use a finite number of terms to get a good estimate for ln(x).

1. In order to see how this algorithm works, fill in Table 3 for x = 1.5.

**Table 3: Iterative Algorithm for ln(x)**

|  |  |  |
| --- | --- | --- |
| **Initial** | **Estimate = 0** | **Difference = abs(Estimate – Previous Estimate)** |
| **k = 1** | **Estimate = Estimate + (x – 1)1 / 1 = 0.5** | **0.5** |
| **k = 2** | **Estimate = Estimate – (x – 1)2 / 2 = 0.375** | **0.125** |
| **k = 3** | **Estimate = Estimate + (x – 1)3 / 3 = 0.4167** | **0.0417** |
| **k = 4** | **Estimate = Estimate (x – 1)4/ 4 = 0.401** | **0.0159** |

**Note: ln(1.5) = 0.4055. You can see that the algorithm is coverging to the actual value as we add more terms!**

**HOW DO WE KNOW WHEN TO QUIT?**

* **For the cube root problem, it was easy. We took our estimate and cubed it. If (Estimate)^3 Number (within our error tolerance), we stopped iterating.**
* **For the natural log problem, things are a bit different. We know that Estimate will eventually get very close to the actual value but we don’t know the actual value. Therefore, we will stop iterating when it looks like the estimates have converged (stopped changing much). So, when the Previous Estimate New Estimate (within our error tolerance), we will stop iterating.**

1. Create a script file to estimate the natural log of a number which is greater than 0 and does not exceed 2 using a finite number of terms from the Taylor Series by doing the following:

* Your program should first prompt the user for the number, x.
* Your program should check and see if x is an invalid number; that is, ***if*** ***x < 0*** **or greater than 2**. Use a **while** loop for this! If the number is invalid, prompt the user to enter a new valid value for x. The **while** loop is nice because it will continue to prompt the user until the user finally enters an acceptable value for x. Some users are a bit dense, and might require a time or two or three or more to finally get it right.
* Initialize the Estimate = 0; the Count (Number of Terms) = 0; and the Difference = 1;
* Your program should then use a **while** loop to calculate the estimate of the natural log of x by doing the following:
* The while loop should continue while Difference exceeds 1e-6.
* Create a variable to save the most recent Estimate: Previous = Estimate
* Update the Estimate by adding a new term from the Taylor Series

**Hint: Look at Table 3. Each iteration, Estimate = Estimate + New Term. The equation for the New Term changes every iteration. It obviously depends on x. See if you can relate the new term to x and Count.**

* Update Difference (Difference = abs(Previous – Estimate))
* Increment the Count by 1
* After the **while** loop, add fprintf statement(s) to display the estimate of the ln(x) with ***6 places*** behind the decimal point, the actual value of ln(x) (remember this is log(x) in MATLAB) with ***6 places*** behind the decimal point, and the number of iterations (or terms), Count as an integer.

1. Test your program to make sure it doesn’t accept invalid values for x. Try negative values, zero, and values above 2 for x. Make sure you used a while loop for this so your program continues to ask for a valid x until a valid value is entered by the user.
2. Now use your program to complete Table 4.

**Table 4: Program Output**

|  |  |  |  |
| --- | --- | --- | --- |
| **x** | **Actual Value for ln(x)** | **Estimate for ln(x)** | **Iterations ( Count)** |
| 0.1 | -2.302585 | -2.302577 | 89 |
| 0.3 | -1.203973 | -1.203971 | 30 |
| 0.5 | -0.693147 | -0.693146 | 16 |
| 1.0 | 0.000000 | 0.000000 | 1 |
| 1.3 | 0.262364 | 0.262364 | 10 |
| 1.5 | 0.405465 | 0.405465 | 16 |
| 1.9 | 0.641854 | 0.641854 | 89 |

**Observations: How does the value for x affect the Number of Iterations?**

As the value of x gets closer to the natural log of x, the loop will need less and less iterations. This will occur as long as the number falls within the parameters of the first while loop.

**PASTE SCRIPT FOR PART B HERE:**

clear;clc;

x = input('What is the value of x? ');

while (x > 2) || (x <= 0)

x = input('Type in a valid number for x ');

end

Estimate = 0;

Count = 0;

Difference = 1;

Logar = log(x);

while Difference > 0.000001

Previous = Estimate;

Count = Count + 1;

if rem(Count,2) == 1

Estimate = Estimate + ((x-1)^Count)/Count;

else

Estimate = Estimate - ((x-1)^Count)/Count;

end

Difference = abs(Previous-Estimate);

end

fprintf('Estimate: %0.6f \n Actual: %0.6f \n Count: %i \n', Estimate, Logar, Count)

**To be turned in: Lab 7 word document with tables filled.**