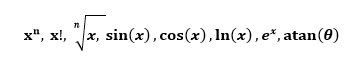
**Lab 8: Iterative Algorithms**

**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 square root and an iterative algorithm for ln(x).

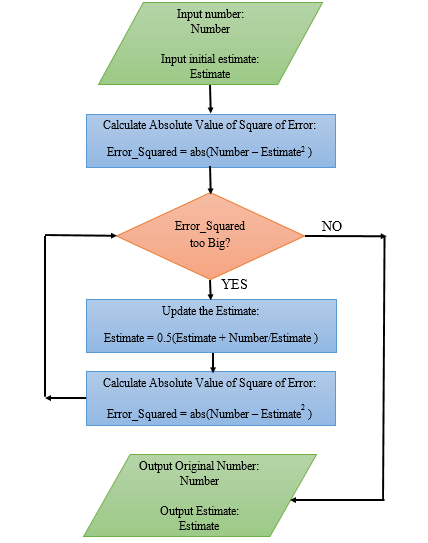
**Part A: Square Root**

There are several algorithms for computing the square root of a number. We are going to look at a very simple algorithm that has been around since the time of the Babylonians and is therefore referred to as the Babylonian Method. 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. Take a look at the flow chart then come back to this step. In order to understand the algorithm, work through the following table by hand first. Assume we want to find the square root of 8, so Number = 8. Assume an initial estimate of Estimate = 1.

**Table 1: Babylonian Method**

|  |  |  |
| --- | --- | --- |
| **Initial** | Estimate = | Error\_Squared = |
|  | **Estimate = 0.5(Estimate + Number/Estimate)** | **Error\_Squared = abs(Number – Estimate2)** |
| **Iter. 1** | Estimate = | Error\_Squared = |
| **Iter. 2** | Estimate = | Error\_Squared = |
| **Iter. 3** | Estimate = | Error\_Squared = |
| **Iter. 4** | Estimate = | Error\_Squared = |



1. Download the template script file from Blackboard. Re-name the file Lab9A\_*YourLastName*. Write a script file to estimate the square root of a number using the Babylonian Method.

* Use the flow chart as a reference for writing your script.
* Assume we want an error no larger than 0.001 so the Error\_Squared should be no larger than 1e-6.
* Use a single fprintf statement to output the original number and the estimate of the square root with 3 places behind the decimal point.

1. Create a set of test inputs for your program. Pick three numbers and any initial estimate for the square root and enter them into the table below. Calculate the actual square root and enter it in the table with at least 3 places behind the decimal point. Now run your program and enter the result. Fix your program if necessary.

**Table 2: Test Inputs and Results**

|  |  |  |  |
| --- | --- | --- | --- |
| **Number** | **Initial Estimate** | **Actual Square Root of Number** | **Program Output** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

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. Of course, the number of iterations will also depend on how close the original guess is. Modify your script to count the number of iterations through the while loop. Add another fprintf statement at the end of the script to display the numbers of iterations as an integer.
2. Run your program for the values shown in Table 3 and enter your results in the table.

**Table 3: Effect of Initial Guess on Number of Iterations**

|  |  |  |  |
| --- | --- | --- | --- |
| **Number** | **Initial Estimate** | **Estimate of Square Root** | **Number of Iterations** |
| **12345678.9** | **1** |  |  |
| **12345678.9** | **100** |  |  |
| **12345678.9** | **1000** |  |  |
| **12345678.9** | **4000** |  |  |

**Comment:** Computing fast and accurate algorithms for determining values of functions such as square root 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 an estimate for ln(x).

1. In order to see how this algorithm works, fill in Table 4 for x = 1.25.

**Table 4: Iterative Algorithm for ln(x)**

|  |  |
| --- | --- |
| **Initial** | **Estimate = 0** |
| **k = 1** | **Estimate = Estimate + (x – 1)1 / 1 =** |
| **k = 2** | **Estimate = Estimate – (x – 1)2 / 2 =** |
| **k = 3** | **Estimate = Estimate + (x – 1)3 / 3 =** |
| **k = 4** | **Estimate = Estimate (x – 1)4/ 4 =** |

**Note: ln(1.25) = 0.2231. The algorithm provides a pretty accurate estimate after only four iterations for this particular number, x = 1.25.**

1. Download the template script file from Blackboard. Re-name the file Lab9B\_*YourLastName*. Write 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.

* Your program should first prompt the user for the number,x, and for the number of desired terms, N.
* Your program should check and see if x is an invalid number; that is, ***x is*** ***less than or equal to 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 the number. The **while** loop is nice because it will continue to prompt the user until the user finally enters an acceptable value of x.
* Your program should then use a **for** loop (**for** k = 1:N) to calculate the estimate of the natural log using N terms. **Remember: in MATLAB, natural log is log.**

**Hint: Look at Table 4. Each iteration, Estimate = Estimate + New Term. The equation for the New Term changes every iteration. It obviously depends on x. See if you can figure out how to relate the equation to k (the index variable for your loop) also.**

* After the **for** loop, add an fprintf statement to display the estimate of the ln(x) with ***8 places*** behind the decimal point.

1. Test your program to make sure it doesn’t accept invalid values for x. Try negative values, zero, and values above 2.
2. Test your program using the values you computed by hand in Table 4. That is, choose x = 1.25 and try 1, 2, 3, and 4 terms.
3. Now use your program to complete Table 5.

**Table 5: Program Output**

|  |  |  |  |
| --- | --- | --- | --- |
| **x** | **Number of Terms** | **Actual Value for ln(x)** | **Estimate for ln(x)** |
| 1.5 | 1 | 0.40546511 |  |
| 1.5 | 3 | 0.40546511 |  |
| 1.5 | 6 | 0.40546511 |  |
| 1.5 | 7 | 0.40546511 |  |
| 1.5 | 10 | 0.40546511 |  |
| 1.5 | 20 | 0.40546511 |  |