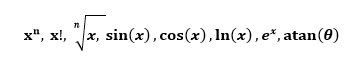
**Taylor Series: Sin(x)**

**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. Taylor series is one method for calculating the sine of an angle.

The Taylor series for sin(x) around x = 0 is:

Complete Table 1 by computing the Taylor series for the given angle and given number of terms. Use either MATLAB or a calculator. The values in the table can then be used for testing your program.

Notes:

5! is computed in MATLAB using the function ***factorial***(5).

Three terms means: it does not refer to the power.

**Table 1: Taylor Series Estimate**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **x** | **sin(x)** | **1 Term** | **2 Terms** | **3 Terms** | **4 Terms** | **5 Terms** |
| π/2 |  |  |  |  |  |  |
| π |  |  |  |  |  |  |

**Pre-work Exercise:** If x = 7π/2, the estimate for sin(x) using the first 14 terms of the Taylor series above gives an estimated value of 1.15667. What would the estimate be using 15 terms?

(Note: make good use of the result already given for 14 terms).

**Problem 1:** In this problem, you will be writing a MATLAB program that will compute an estimate for the sin of an angle based on the Taylor series expansion for sin(x) around x = 0. The student (user) will have to input the angle (in radians) and the number of terms of the Taylor series to include. The program will then iterate through the Taylor series. The program should output the actual value for the sin of the angle, the estimate based on the Taylor series, and the absolute value of the estimation error (use **abs**). Use ***fprintf*** to display the values. Use a separate line for each value (\n) and display each value using 6 places behind the decimal point with appropriate text identifying what the value is (i.e., actual or estimate or error).

1. Based on the problem statement, decide what the inputs and outputs are.

Inputs Outputs

1. Create a flow chart for this program. When the flowchart is completed, have it checked by your instructor prior to proceeding to writing code.
2. In MATLAB, create a new script and write your program. Test the program when it is finished using your results from Table 1.

**Problem 2:** In the previous problem, the user specified both the angle and the number of terms to include for the Taylor series estimation. Write a new program where the user only specifies the angle and the Taylor series is iterated until the absolute value (**abs**) of the estimation error is less than 1e6. This program should display (output) the actual sin value, the estimated value, the absolute value of the error, and the number of terms needed in the Taylor series to achieve the required accuracy. Again, use ***fprintf*** to display the outputs (one output per line with appropriate identifying text). Display the actual sin value and estimated sin value using 6 places behind the decimal point, the absolute value of the error using 8 places behind the decimal point, and the number of terms as an integer (%d).

***Note: Don’t overwrite your function from Problem 1 – start a new one. You can certainly copy code from the original function to save typing time.***

1. In MATLAB, create a new script file and write your program. Test the program when it is finished.
2. Once you have verified that the program is working, use your program to complete

Table 2. The first value is provided as a check for your program.

**Table 2: Taylor Series Estimation for sin(x)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **angle** | **Actual** | **Estimate** | **Estimation Error** | **Number of Terms Required for Taylor Series** |
| π/4 | 0.707107 | 0.707106 | 0.00000031 | 4 |
| π/2 |  |  |  |  |
| π |  |  |  |  |
| π |  |  |  |  |
| 2π |  |  |  |  |
| 7π/2 |  |  |  |  |
| 8.5π |  |  |  |  |

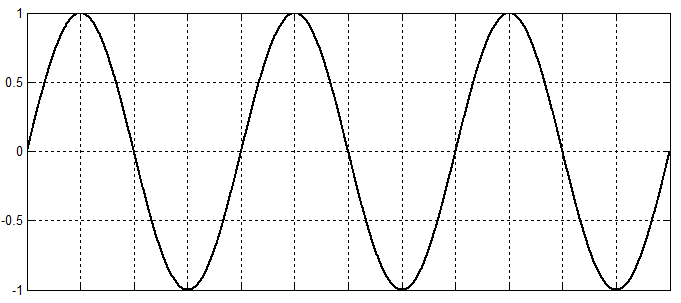
What happens with 8.5π? Why?

***Note: Clearly, the number of required terms gets larger as the angle moves away from 0. One way to reduce the number of terms would be to expand the Taylor series around an angle closer to the one we are trying to estimate. A look-up table would provide starter values. Another approach is to make use of some of the symmetry properties of sine waves to reduce the number of terms needed. Most programs and calculators use a combination of these two approaches. For now, we will just use the symmetry properties to reduce the number of terms needed.***

**Problem 3:**

From the previous problem, you know it takes no more than 13 terms to provide a good estimate when the angle ranges from π to +2π, but as the angle drifts further from this range, more and more terms are required to get a good estimate. In fact, your program completely failed for 8.5π and would also fail for any angles with absolute value larger than 8π.

**sin(x)**



2π 3π/2 π π/2 0 π/2 π 3π/2 2π 5π/2 3π 7π/2 4π

**Property:** sin(x) = sin(x + k(2π)) where k is any integer.

In words, sin(x) is a periodic function that repeats every 2π. Angles that are 2π apart or integer multiples of 2π apart have exactly the same sin value. So, sin(8.5π) = sin(8.5π2π2π2π2π) = sin(0.5π). The program you wrote in Problem 2 can’t handle 8.5π, but it can certainly handle 0.5π and both angles have the exact same sine value. So, if we can translate a large angle into a smaller angle with the exact same sine value, our program would work O.K.

MATLAB has a function called **rem** (remainder after division). If x is any angle, the command

>> x\_equiv = rem(x,2\*pi)

divides angle x by 2π and assigns the remainder to x\_equiv. So angle x\_equiv will have a value between 2π and +2π and will be exactly some integer multiple of 2π away from the original x value. This means x\_equiv will have exactly the same sine value as the original angle, x.

To convince yourself of this, complete Table 3 below by dividing each angle by 2π and recording the remainder. Notice that every one of your remainders are integer multiples of 2π from the original angle so the sin of the “remainder angles” will be identical to the sin of the original angle.

**Table 3: The rem function**

|  |  |  |
| --- | --- | --- |
| **x** | **x\_equiv= rem(x,2\*pi)** | **x – x\_equiv** |
| 7.5π | 1.5π | 6π |
| 7.5π |  |  |
| 3.25π |  |  |
| 5.25π |  |  |
| 4.25π |  |  |

Now, modify the program for ***Problem 1*** (*save it under a new name*) to take the incoming angle and translate it to an equivalent angle between 2π and 2π before computing the Taylor series estimate using 13 terms. Fill in Table 4 below (you should not get NaN for 8.5π.

**Table 4: Taylor Series Estimate**

|  |  |  |  |
| --- | --- | --- | --- |
| **angle** | **Actual** | **Estimate (13 terms)** | **Estimation Error** |
| 8π |  |  |  |
| 8.5π |  |  |  |
| 43.7π |  |  |  |
| π |  |  |  |