Programming Cosine A Maclaurin Series Approach

Gurjot Singh

Las Positas College

Math 2

Olain Olavarrieta

4/30/2022

Abstract

This paper will investigate the usage and explanation of the Maclaurin Series for Cosine as well as to explain generally what the Maclaurin series is as well as how functions are computed during its use. Due to the extremely similar nature of the Maclaurin series and the Taylor series the Taylor series will also be referenced and discussed to provide a reference for the history of the Maclaurin series and to contrast its similarities and differences to the Taylor series. From this the historical basis can be formed to create a Cosine calculator in a programming shell environment written and compiled by the C++ programming language. The C++ programming language specifically has excellent computational prowess in low level numerical manipulation of memory which is especially necessary for the purposes of this paper.

Introduction

The work of the Maclaurin Series expands upon quite closely the work of Brook Taylor 16th century English noble and scholar who actually published and formalized the Taylor series earlier theorized by James Gregory (Maplesoft, n.d.). The Taylor series in essence allows for the approximation of functions on a specified or infinite interval through the use of finite or infinite terms with the basic formula given (Maplesoft, n.d.).

Table 1

shows the Tayor Series formula

$$\text{Taylor Series} \quad (1)\ \sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n$$

figure 1 shows the Tayor Series formula

this is contrasted with the Maclaurin Series which simply evaluates the function at the value zero for the taylor series the similarity is such that many mathematicians still argue about whether to even consider the Maclaurin series at the point x equals zero to be worthy of its own name. This is by virtue of the fact that a Maclaurin series is and always has been in essence a Taylor series expanded upon a specific x value in this case zero. So it is common to hear either the term Maclaurin Series or Taylor series about x equal zero. This is not to say that a Maclaurin series is completely useless; for example its simplicity can be seen in its much more compact writing for a variety of functions such as e to the x.

Table 2

figure 2 shows the Maclaurin Series for e^x

Function to Calculate	(1)	$:f(x) = e^x$
Maclaurin Series	(2)	$:\sum_{n=0}^{\infty} \frac{f^n(0)}{n!}(x)^n$
First Few Terms	(3)	$:\frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} \cdots \frac{x^n}{n!}$
Final Summation	(4)	$:\sum_{n=0}^{\infty} \frac{x^n}{n!}$

The final formula of the function e to the x power shows that the maclaurin series is much more compact and simple to read than a Taylor series compared at another value other than zero.

Table 3

figure 3 shows the Taylor Series for e^x

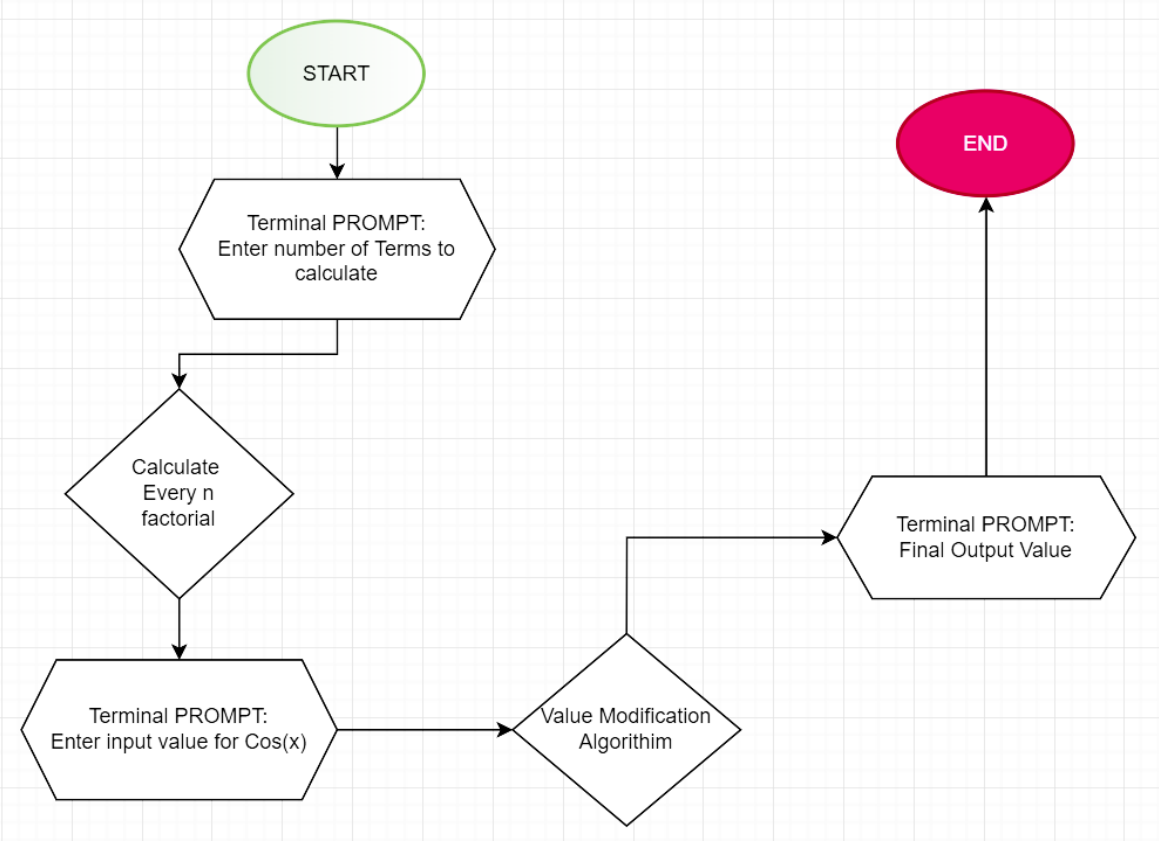
Function to Calculate	(1) :	$f(x) = e^x$
Taylor Series at x = 3	(2) :	$\sum_{n=0}^{\infty} \frac{f^n(3)}{n!}(x-3)^n$
First Few Terms	(3) :	$\frac{e^3(x-3)^0}{0!} + \frac{e^3(x-3)^1}{1!} + \frac{e^3(x-3)^2}{2!} \cdots \frac{e^3(x-3)^n}{n!}$
Final Summation	(4) :	$\sum_{n=0}^{\infty} \frac{e^3(x-3)^n}{n!}$

Historical Origins

As mentioned already the basis of naming the Taylor series centered at x equals zero's justification is debated but the main basis for its naming after the Scottish mathematician Colin Maclaurin was because of his use in proving the inflection points as well as maximum and minimum values of problems as described in his work "A Treatise on Fluxions" of infinitely differentiable functions (Maclaurin, 1742). By which an algorithm so to speak was developed through his application of the series. For example if at any point in a function input x is equal a positive number whose derivative at that point is zero then there is either a maximum or minimum of said function at this point depending on whether or not the second derivative is positive or negative so if negative the input value is a maximum and a minimum if the second derivative is positive. It is on this basis for which the Taylor series centered at zero was coined after Colin Maclaurin but only by later mathematicians because Colin Maclaurin never attributed it to himself due to it being known to several mathematicians before him. To better and more accurately calculate cosine as is done in many digital platforms the development of a Macluarin Series calculator for Cosine using C++ a very popular programming language.

Table 4

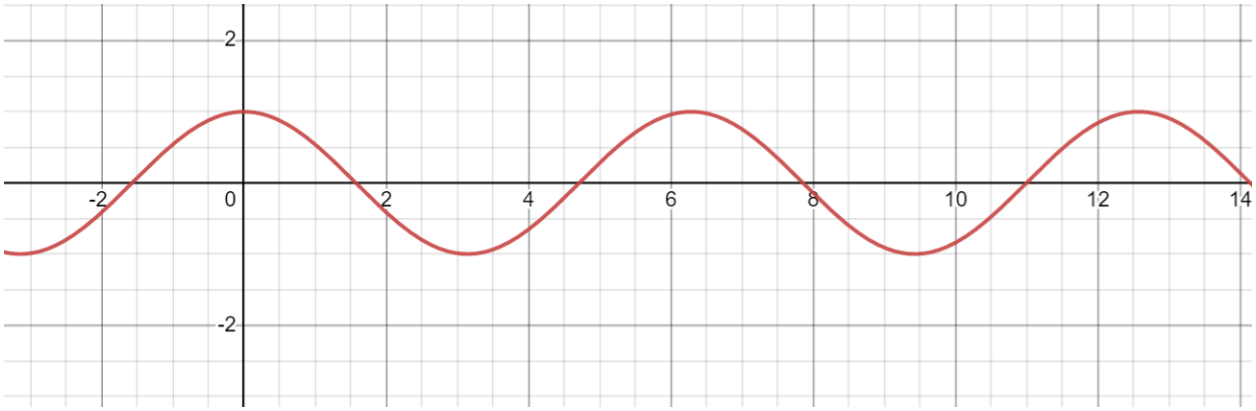
figure 4 shows Cosine Calculator in flowchart form



The flowchart in figure 4 explains quite clearly in its labeling the steps that the program is taking to create the final calculator for a cosine except the Value Modification Algorithm. The Cosine function is inherently repeating, meaning that every output value repeats constantly as the inputs grow larger. So instead of having to calculate and manage every input and output value the user input can be shifted to a domain that contains every output value which is closer to zero. This is known as the reference angle and through it more accurate calculations can be done.

Table 5

figure 5 shows Cosine Function graphed by Desmos



VALUE MODIFICATION ALGORITHM

(0) $After\ continuous\ subtraction\ of\ 2\pi$

(1) $if\ (0 \leq\ value \leq \frac{\pi}{2})\ =>\ Output: cos(value)$

(2) $if\ (\frac{\pi}{2} \leq\ value \leq \pi)\ =>\ Output: cos(\pi - value)$

(3) $if\ (\pi \leq\ value \leq \frac{3\pi}{2})\ =>\ Output: -1 * cos(value - \pi)$

(4) $if\ (\frac{3\pi}{2} \leq\ value \leq 2\pi)\ =>\ Output: cos(2\pi - value)$

The value modification algorithm as established above gives the proper reference angle essentially meaning that it will account for the negative and positive values in the output.

Error Analysis

Because many complex computations will not always give conclusive results it is important to know the error of an output in this case of a Maclaurin series an upper bound of error can be produced by the Remainder Estimation Theorem which will not give the exact error of a measurement but simple the maximum possible error of a series

REMAINDER ESTIMATION THEOREM

(1) Basic Formula : $\left|R_n(x) \leq M\frac{|x-a|^{n+1}}{(n+1)!}\right|$

(2) Where $M \geq f^{(n+1)}(x)\ on\ domain\ [0, input]$

In the Remainder Estimation Theorem M refers to some constant greater than the derivative of degree n+1 and since cosine never exceeds the value one where it is the greatest this can be used as the M in the formula.

Table 6

figure 6 shows a terminal with cosine calculated at x= 10 along with it error

```
For How any terms will you calculate the Maclaurin Seres of Cosine?
7

For what value would you like x to be for cos(x)
10



Your Answer is -0.839071529076452216067139033839

With an error no greater than 0.000000297278578486870131361666

For How any terms will you calculate the Maclaurin Seres of Cosine?
```

Source Code

```
//============================================================================
// Name        : Maclaurin Series Calculator
// Author      : Gurjot Singh
// Version     : 23
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//============================================================================

#include <iostream>
#include <cmath>
#include <math.h>
#include <bits/stdc++.h>

short int terms;
unsigned long long int factorial [50]; // first 11 factorials pre calculated


 double finalval;
long double input;
short int accuracy = 30;
long double error;

void clearcin() {

	std::cin.clear();
	std::cin.ignore(100000000, '\n');

}

void setfactorials() {
	factorial[0] = 1;

	for (int i = 1; i <= terms; i++) {

		factorial[i] = std::tgamma(2*i + 1);

	}
}


void StartUP() {
	std::cout << std::endl;
	std::cout
			<< "For How any terms will you calculate the Maclaurin Seres
of Cosine?"
			<< std::endl;
	std::cin >> terms;
	if(terms >8) {
		terms = 8;

	}
	setfactorials();
	std::cout << std::endl;
	std::cout << "For what value would you like x to be for cos(x)"
			<< std::endl;
	std::cin >> input;

}

void Output(long double somevalue) {
	std::cout << std::endl;
	std::cout << std::endl;
	std::cout << std::endl;
```

```
	std::cout << std::fixed << std::setprecision(accuracy) << "Your Answer is
"<< somevalue << std::endl;
	std::cout << std::endl;
	error = pow(input,terms+1)/ (std::tgamma(terms+2));
	std::cout << "With an error no greater than " << error << std::endl;
}

bool zero_pi_2(long double inputval) {
	if ((0 < inputval) or (0 == inputval)) {
		if ((inputval < M_PI / 2) || (inputval == M_PI / 2)) {
			return true;
		}
		else {
			return false;
		}
	}
	else {
		return false;
	}
}

bool pi_by2_and_pi(long double inputval) {
	if ((M_PI / 2 < inputval) or (M_PI / 2 == inputval)) {
		if ((inputval < M_PI) || (inputval == M_PI)) {
			return true;
		}
		else {
			return false;
		}
	}
	else {
		return false;
	}
}

bool pi_and_3pi_by2(long double inputval) {
	if ( (  M_PI < inputval) or (M_PI  == inputval) ) {
		if ((inputval < 3*M_PI/2) || (inputval == 3*M_PI/2)) {
			return true;
		}
		else {
			return false;
		}
	}
	else {
		return false;
	}
}

bool threepi2_by2pi(long double inputval) {


	if ( (  3*M_PI/2 < inputval) or (3*M_PI/2  == inputval) ) {
		if ((inputval < 2*M_PI) || (inputval == 2*M_PI)) {
			return true;
		}
		else {
			return false;
		}
	}
	else {
		return false;
	}
}
```

```
int main() {

	int a = 2;
	int negativeone = -1;

	while (true) { /****/

		finalval = 0;

		StartUP();

		if (input < 0) {
			input = input * negativeone;
		}
		while(input > (2*M_PI)) {
			input = input - (2*M_PI);
		}

		if (zero_pi_2(input)) {
			for (int e = 0; factorial[e] != 0; e++) {
				finalval = finalval+ (pow(negativeone, e) * pow(input, a
* e)) / factorial[e];
			}
			Output(finalval);
		}


		else if (pi_by2_and_pi(input)) {
			input = M_PI - input;

			for (int e = 0; e <= terms; e++) {
				finalval = finalval+ (pow(negativeone, e) * pow(input, a
* e)) / factorial[e];
			}
			finalval = finalval * negativeone;
			Output(finalval);
		}

		else if (pi_and_3pi_by2(input)) {
			input =  input - M_PI;

			for (int e = 0; e <= terms; e++) {
				finalval += (pow(negativeone, e) * pow(input, a * e)) /
factorial[e];
			}
			finalval = finalval * negativeone;

			Output(finalval);
		}

		else if (threepi2_by2pi(input)) {
			input =  2*M_PI - input ;

			for (int e = 0; e <= terms; e++) {
				finalval += (pow(negativeone, e) * pow(input, a * e)) /
factorial[e];
			}
			finalval = finalval * negativeone;

			Output(finalval);
		}
		clearcin();

	} /****/
```

```
        return 0;
}
```

References

Maclaurin, C. (1742). *A Treatise of Fluxions. In Two Books by Colin MACLAURIN - First edition - 1742 - from SOPHIA RARE BOOKS (SKU: 4222)*. Biblio.Com; Printed by T.W. and T. Ruddimans.https://www.biblio.com/book/treatise-fluxions-two-books-maclaurin-colin/d/1034960224#:~:text=%22The%20Treatise%20of%20Fluxions%20of%201742%20was%20Maclaurin%27s,explained%20and%20many%20new%20results%20introduced%20and%20proved.

MacLean, M. (2007, October 7). *Some Notes on Taylor Polynomials and Taylor Series*. University of British Columbia. https://personal.math.ubc.ca/~nagata/sci1/taylornotes.pdf

Maplesoft. (n.d.). *Taylor Series - Math Terms & Solutions - Maplesoft*. Maplesoft - Software for Mathematics, Online Learning, Engineering. Retrieved May 10, 2022, from https://www.maplesoft.com/ns/math/taylor-series.aspx#:~:text=The%20concept%20of%20a%20Taylor,mathematician%20Brook%20Taylor%20in%201715.

Burnette, C. (n.d.). *Proof of Taylor's Remainder Estimation Theorem*. Https://Burnettecd.Files.Wordpress.Com/. Retrieved May 17, 2022, from https://burnettecd.file.wordpress.com/2019/04/proof-of-taylors-remainder-estimation-theorem.pdf