Ссылка на видео	

«Изучение методов вычисления синуса»

индивидуальный учебный проект

ВЫПОЛНИЛ:
ученик 10 класса
Калугин Андрей Павлович

НАУЧНЫЙ РУКОВОДИТЕЛЬ:
Ибрагимова Нурай Афиг кызы

Старый Крым, 2023-2024

Содержание

Введение

В области математики и информатики вычисление тригонометрических функций играет первостепенную роль. Одна из таких функций, а именно sin (синус), повсеместно используется в различных областях, начиная с обработки графики и заканчивая научными моделями. Однако, встаёт вопрос: "Какие существуют методы для эффективного вычисления синуса" — вопрос, который далеко не так прост, как может показаться на первый взгляд.

Актуальность темы заключается в том, что такая важная тригонометрическая функция как синус имеет большое количество методов вычисления. К разным целям и оборудованию подходят разные способы реализации функции. Зная оптимальный путь к нахождению значения синуса, можно добиться лучших результатов.

Цель: изучить основные методы для вычисления синуса.

Задачи:

- дать определение функции синус,
- описать свойства синуса,
- рассмотреть основные методы вычисления синуса,
- реализовать основные методы нахождения синуса на языке программирования C++

Объект изучения: тригонометрическая функция sin.

Предмет изучения: способы вычисления синуса.

Использованные методы работы:

- Сравнение
- Анализ
- Синтез
- Моделирование

Глава I. Основные понятия

1.1 Определение синуса острого угла

Тригонометрические функции могут быть определены как отношение сторон прямоугольного треугольника. В частности, синус острого угла равен отношению катета, лежащего напротив данного угла, к гипотенузе.

На рисунке 1 синус угла A — это отношение отрезка BC к отрезку AB.

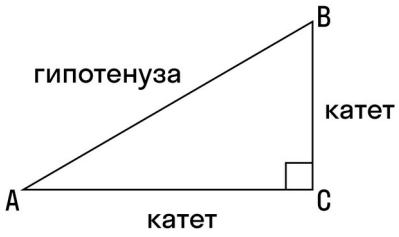

Рисунок 1

1.2 Единичная и числовая окружности, радиан

Единичная окружность — это окружность с радиусом равным 1 и центром в начале координат. Благодаря единичной окружности можно избавиться от лишних коэффициентов при расчётах.

Радиан — это градусная мера угла. Один радиан соответствует углу в окружности, длина дуги которого равна радиусу этой окружности.

Числовая окружность (рисунок 2) — это единичная окружность, на которой каждому действительному числу соответствует точка на ней. Для положительного числа отсчёт происходит против часовой стрелки, а для отрицательного по часовой стрелке. Например, т. к. длина окружности равна 2πR, то точке (0, 1) будет соответствовать $\frac{\pi}{2}$ радиан, $5\frac{\pi}{2}$, $\frac{-3\pi}{2}$ и т. д.

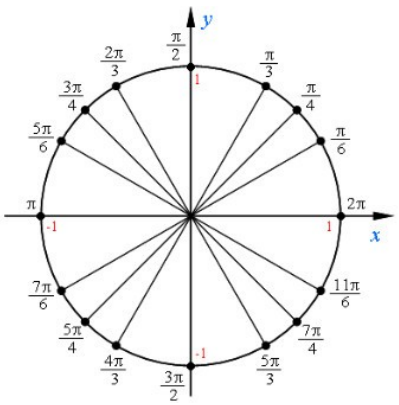

Рисунок 2

1.3 Определение синуса числа

Проведём вектор из начала координат в точку на единичной окружности (рисунок 3). Его длина равна радиусу. Теперь синус может быть определён как вертикальная составляющая этого вектора. Это объясняется тем, что если мы построим прямоугольный треугольник с гипотенузой, совпадающей с данным вектором так, что противолежащий катет вертикален, то синус будет равен отношению противолежащего катета на единичный радиус, что дало бы нам сам противолежащий катет или вертикальную составляющую вектора.

Синус числа равен ординате соответствующей точки на единичной окружности. Как правило, синус числа принимает радианы, но их всегда можно перевести в градусы, если так удобнее.

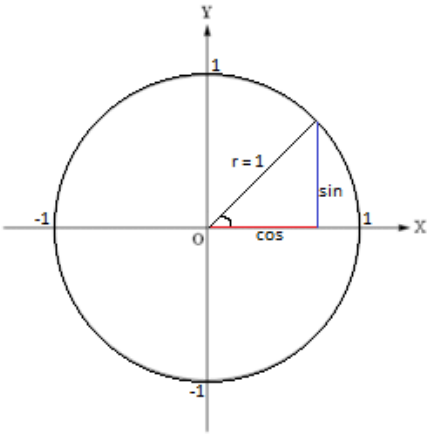

Рисунок 3

1.4 Синусоида

Синусоида — это кривая (график), задаваемая уравнением $y = a + b + \sin(cx + d)$, где a, b, c и d являются постоянными.

Правильной синусоидой называется частный случай синусоиды, в котором a, b, d равны 0, а c = 1. То есть кривая, задаваемая уравнением $y = \sin(x)$ (рисунок 4).

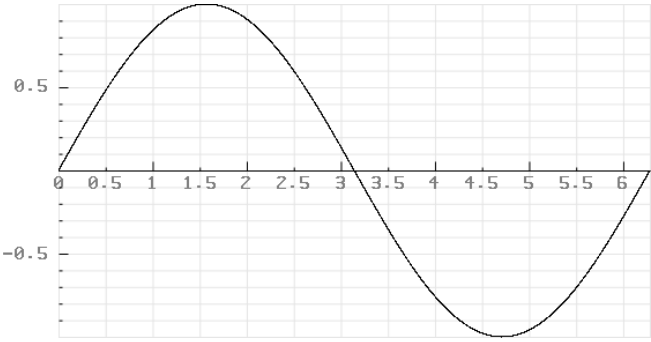

Рисунок 4

Глава II. Свойства синуса

В этой главе мы рассмотрим некоторые свойства синуса и правильной синусоиды, которые пригодятся в дальнейшем.

2.1 Область определения и область значения синуса

Мы уже определили синус для любого числа, следовательно, область определения — все действительные числа.

Если рассматривать синус острого угла, то т. к. гипотенуза всегда больше катета, синус принимает значения (0; 1).Но при рассмотрении синуса числа, заметим, что на единичной окружности максимальная ордината равна 1, а минимальная -1. Следовательно, синус числа принимает значения [-1; 1].

Можно сразу сделать вывод о размахе правильной синусоиды. Напомню, что размах — это разность максимального и минимального значений. Итак, размах равен 2.

2.2 Периодичность синуса

Для начала определим периодическую функцию. Функция f(x) называется периодической с периодом T, если $T!=0$ и $f(x-T)=f(x)=f(x+T)$. Стоит заметить, что для периодической функции также справедливо $f(x-T\cdot n)=f(x)=f(x+T\cdot n)$ для натуральных n. Поэтому надо различать основной период с остальными. Основной период — это наименьший положительный период.

Как мы уже видели, одной точке на числовой окружности соответствует бесконечное множество чисел. Так для точки (1, 0) соответствуют 0, 2π, 4π, -6π и т. д.

Заметим, что если к углу прибавить целый оборот (2π или 360 градусов), то итоговое значение останется прежним. Таким образом, синус — периодическая функция.

Основной период синуса равен 2π. $\sin(x-2\pi)=\sin(x)=\sin(x+2\pi)$. На самом деле периодичность синусоиды не заканчивается на этом. Рассмотрим график синусоиды в значениях [0; π] (полупериод).

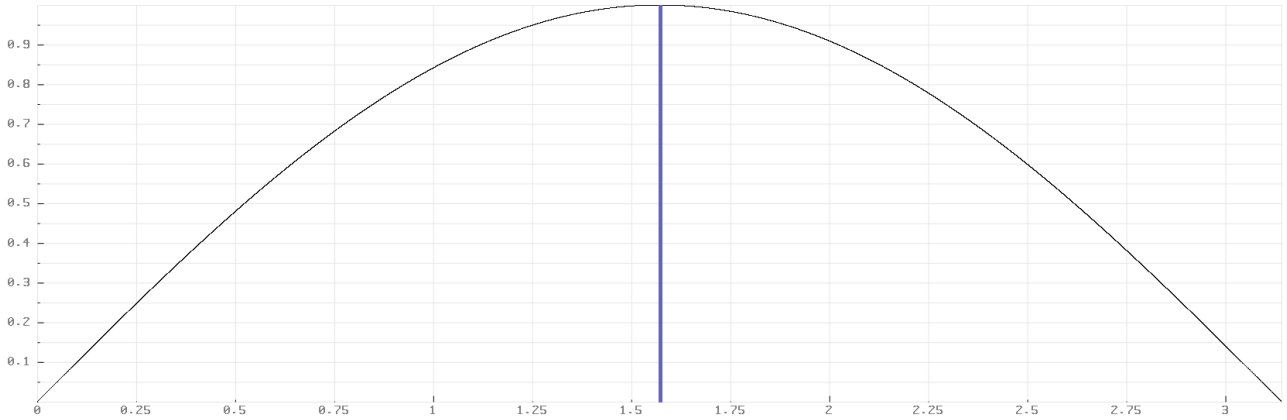

Рисунок 5

На графике (рисунок 5) видно, что левая половина полупериода симметрична правой. Это объясняется тем, что после пересечения отметки $\frac{\pi}{2}$, синус, достигнув максимального значения, начинает уменьшаться, принимая те же значения, что и до. Теперь рассмотрим график одного полного периода синусоиды.

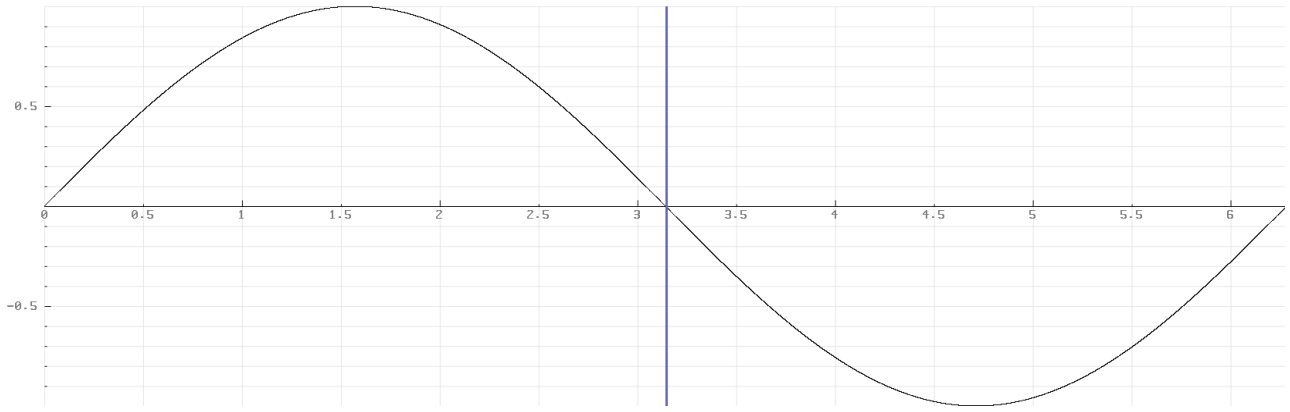

Рисунок 6

Заметим, что левый полупериод симметричен правому (рисунок 6). Фактически, если угол больше нуля, то нечётные полупериоды принимают положительные значения, а чётные - отрицательные. Если угол меньше нуля - наоборот.

Эти свойства позволят нам в дальнейшем упростить нахождение синуса больших углов, сводя все вычисления в промежуток $[0; \frac{\pi}{2}]$.

2.3 Производная синуса

Производная функции — это отношение изменения значения функции к изменению её аргумента с условием, что изменение аргумента стремится к нулю. Первая производная функции характеризует как быстро функция растёт в данной точке. Вторая производная функции — это производная первой производной, то есть она характеризует как быстро растёт скорость изменения изначальной функции. Третья производная — это производная второй производной и т. д.

Например, пусть f(t) — зависимость координаты тела от времени. Первая производная данной функции есть скорость тела, а вторая — ускорение.

Существуют разные способы записи производной функции. Пусть f(x) — изначальная функция, тогда её производную можно записать как:

$f^{(1)}(x_0)$ или $f'(x_0)$ общий вид записи: $f^{(n)}(x)$ — нотация Лагранжа.

x_0 — точка, в окрестности которой мы ищем производную.

$\frac{d}{dx}(x_0)$ или $\frac{df}{dx}(x_0)$ общий вид записи: $\frac{d^n f}{dx^n}(x_0)$ — нотация Лейбница.

Несмотря на то, что запись в виде деления, на самом деле имеется в виду производная функции f(x) в точке x_0. Есть и другие способы записи производной. Мы же будем использовать обе нотации там, где они более уместны.

Запишем определение производной с помощью предела $\lim\limits_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \lim\limits_{\Delta x \to 0} \frac{\Delta f(x)}{\Delta x} = f'(x)$.

Итак, первая производная синуса — это косинус. $f(x) = \sin(x)$, $f'(x) = \cos(x)$. В свою очередь производная косинуса есть синус со знаком минус, следовательно, вторая производная синуса — это отрицательный синус.

$g(x) = \cos(x)$, $g'(x) = -\sin(x)$, $f''(x) = -\sin(x)$. Производная отрицательного синуса — это косинус со знаком минус, а его производная — это синус. Выходит, что четвёртая производная синуса есть сам синус $\frac{d^4 f}{dx^4} = \sin(x)$.

Глава III. Методы вычисления синуса

3.1 Табличные значения

Табличный метод — это эффективный и, пожалуй, самый простой способ получения приближенных значений синуса и других функций. Он основан на хранении предварительно вычисленных значений для некоторых аргументов.

Здесь очень помогает то, что все вычисления можно свести в промежуток от 0 до 90 градусов, ведь благодаря этому таблица получается относительно небольшой. После того, как таблица составлена, мы можем для любого заданного угла использовать интерполяцию между ближайшими значениями из таблицы.

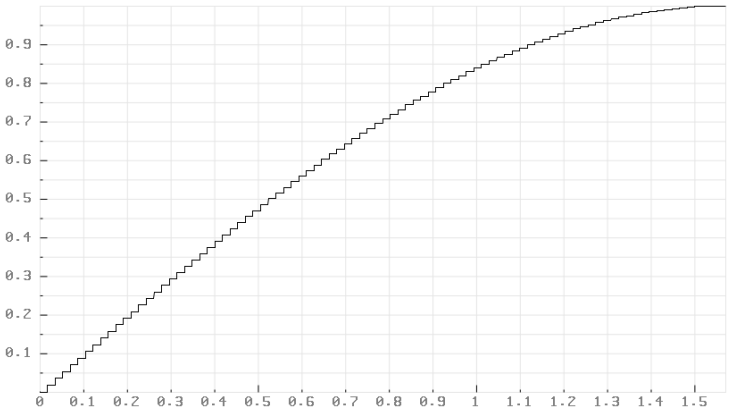

Рисунок 7

Интерполяция — это нахождение неизвестных промежуточных значений функции, по набору известных значений. Существуют разные методы интерполяции, например, простейший из них — это линейная интерполяция. Она соединяет соседние точки прямолинейными отрезками.

На рисунке 7 изображён график функции, использующей табличный метод вычисления синуса с новым значением для каждого целого градуса. Так, для угла 80.4° значение то же, что и для 80°. На рисунке 8 изображён график той же функции, но для нахождения промежуточных значений используется линейная интерполяция.

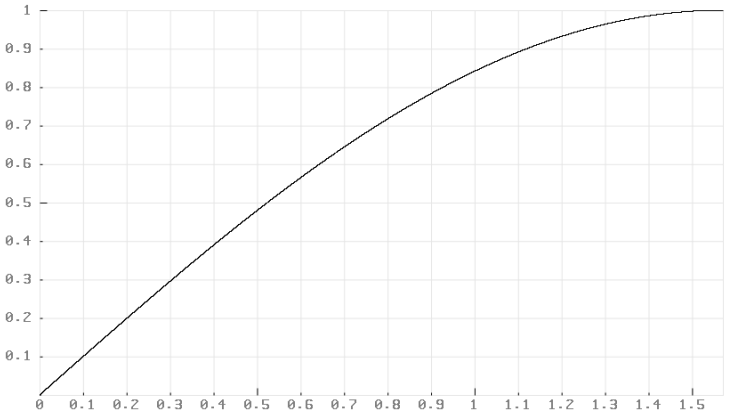

Рисунок 8

Так как данный метод не требует сложных операций, то вычисления происходят весьма быстро. Очевидно, что точность таких вычислений напрямую зависит от количества элементов в таблице. Поэтому его можно использовать в тех ситуациях, где вычислительные ресурсы ограничены, но есть достаточно памяти для хранения значений.

3.2 Ряд Тейлора

Ряд Тейлора — это разложение функции на сумму степенных функций. То есть ряд Тейлора — это многочлен.

Пусть $f(x)=a_0+a_1x+a_2x^2+...+a_{n-1}x^{n-1}+a_nx^n$. Продифференцируем (найдём производные) данный многочлен n раз:

$f'(x)=a_1+2a_2x+...+(n-1)a_{n-1}x^{n-2}+na_nx^{n-1}$,

$f''(x)=1\cdot2\cdot a_2+...+(n-2)(n-1)a_{n-1}x^{n-3}+(n-1)na_nx^{n-1}$ и т. д. Тогда:

$a_0=\frac{f(0)}{0!}$, $a_1=\frac{f'(0)}{1!}$, $a_2=\frac{f''(0)}{2!}$, $a_n=\frac{f^n(0)}{n!}$. Подставим это в изначальное определение f(x):

$f(x)=\frac{f(0)}{0!}+\frac{f'(0)}{1!}x+\frac{f''(0)}{2!}x^2+...+\frac{f^{n-1}(0)}{(n-1)!}x^{n-1}+\frac{f^n(0)}{n!}x^n$. Данное выражение называется формулой Маклорена. Это частный случай при x=0. Если дифференцировать в точке a, то получим формулу в общем виде, которая называется формулой Тейлора:

$f(x)=f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+...+\frac{f^{n-1}(a)}{(n-1)!}(x-a)^{n-1}+\frac{f^n(a)}{n!}(x-a)^n$.

Если функцию можно дифференцировать в точке a конечное количество раз k, то сумма конечна. Если же функция бесконечно дифференцируемая, то и сумма бесконечна.

$f(x)=\sum_{n=0}^{k}\frac{f^n(a)}{n!}(x-a)^n$, или $f(x)=\sum_{n=0}^{+\infty}\frac{f^n(a)}{n!}(x-a)^n$.

В зависимости от искомой функции и её аргумента, существует так называемая область схождения, то есть область, в пределах которой ряд действительно приближается к правильному значению. Эта область — круг, центр которого — точка a. Например, ряд Тейлора для натурального логарифма $f(x)=\ln(x+1)$ при a = 0, сходится при $-1<x\le1$. Аргумент (x+1) составлен так, чтобы f(0) = 0. Если же $x>1$ или $x\le-1$, то ряд расходится с реальными значениями функции. На рисунке 9 видно, как в области схождения функция и ряд Тейлора сходятся, а вне — расходятся. А также, что чем больше n (количество слагаемых), тем ближе приблизительные значения к реальным.

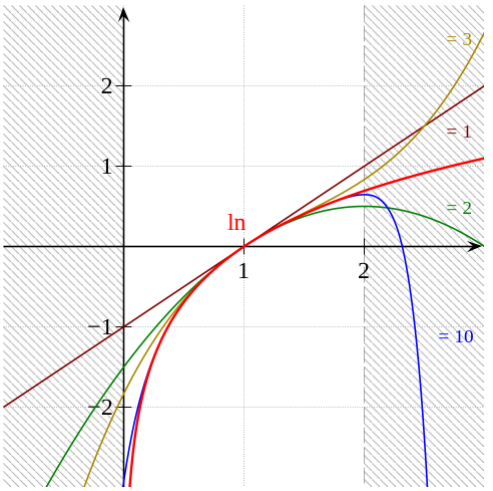

Рисунок 9

Синус можно дифференцировать бесконечное количество раз в любой точке. Следовательно, ряд Тейлора для этой функции — это бесконечная сумма. Как мы уже знаем: $\sin'(x)=\cos(x)$, $\sin''(x)=-\sin(x)$, $\sin'''(x)=-\cos(x)$, $\sin^4(x)=\sin(x)$. Воспользуемся формулой Маклорена зная, что: $\sin(0)=0$ и $\cos(0)=1$;

$\sin(x)=x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}...=\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)!}x^{2n+1}$.

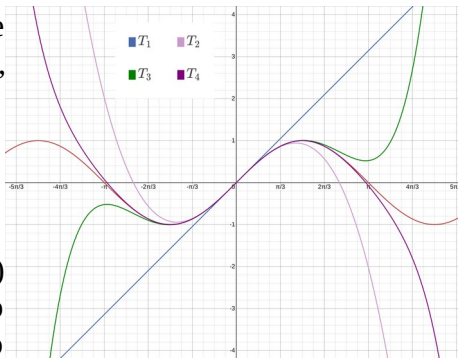

Рисунок 10

На рисунке 10 изображены графики синуса (красный) и рядов Тейлора с 1, 2, 3, 4 слагаемыми. Можно заметить, что линия T1 (синий) — это просто прямая y = x. Это потому, что первая производная $\sin(x_0)$ при $x_0=0$ равна 1.

3.3 CORDIC алгоритм

CORDIC (COordinate Rotation DIgital Computer — цифровой вычислитель координат вращения) — итеративный метод, широко используемый при вычислениях тригонометрических, гиперболических, логарифмических и других сложных функций. Сводит вычисления к простым операциям сложения и сдвига. Это особенно полезно в случаях, когда аппаратные ресурсы ограничены, например, в микроконтроллерах, не обладающих операцией умножения.

CORDIC алгоритм вращает вектор (1, 0) на заданный угол φ. Ордината данного вектора и есть синус угла φ. Поэтому важно понимать как координаты нового вектора связаны с изначальным.

Пусть угол между осью x и изначальным вектором равен θ. Тогда ордината нового вектора равна $\sin(\theta+\varphi)$, а абсцисса - $\cos(\theta+\varphi)$. Используя формулы $\cos(\theta+\varphi)=\cos(\theta)\cos(\varphi)-\sin(\theta)\sin(\varphi)$ и $\sin(\theta+\varphi)=\sin(\theta)\cos(\varphi)+\sin(\varphi)\cos(\theta)$ получаем, что: $x_n=x_{n-1}\cos(\varphi)-y_{n-1}\sin(\varphi)$ и $y_n=y_{n-1}\cos(\varphi)+x_{n-1}\sin(\varphi)$, где x_n и y_n — координаты вектора (x_{n-1}, y_{n-1}) после вращения. Вынесем $\cos(\varphi)$ как общий множитель:
$x_n=\cos(\varphi)(x_{n-1}-y_{n-1}\tan(\varphi))$ и $y_n=\cos(\varphi)(y_{n-1}+x_{n-1}\tan(\varphi))$.

Идея данного заключается метода в том, чтобы вращать вектор на углы, тангенсы которых кратны половине, то есть $\tan(\varphi)=\frac{1}{2^{i-1}}$ для натуральных i. Таким образом умножение на тангенс сводится к делению на степени двойки, что в двоичной системе счисления заменяется битовым сдвигом вправо на $i-1$ бит. Таким образом: $\varphi_i=\arctan\left(\frac{1}{2^{i-1}}\right)$, $\varphi_1=\arctan\left(\frac{1}{2^0}\right)=45°=\frac{\pi}{4}$. Если суммарный угол вращения больше заданного угла, то вращение происходит по часовой стрелке, если меньше — против.

Установив фиксированное количество итераций, можно заранее посчитать углы вращения. Поскольку косинус является чётной функцией, то есть $\cos(x)=\cos(-x)$, то умножение на $\cos(\varphi)$ можно вынести в конец вычислений как заранее вычисленную константу.

$\tan(\varphi)$	φ
1	45°
1/2	26.565°
1/4	14.036°
1/8	7.125°

Глава IV. Практическая часть.

Перейдём к программной реализации изученных методов вычисления синуса. Язык программирования – C++. Ссылка на репозиторий GitHub, где можно найти все исходные файлы: https://github.com/DuyhaBeitz/ComputingSine/

Для некоторых простых математических функций, которых нет в стандартной библиотеке, например факториал или знак числа, был создан файл «basic_math.cpp». Большинство примеров исходного кода является только частью готовой программы и не будет работать в изоляции.

4.1 Встроенная функция

В стандартной библиотеке C++ функция sin() перегружена для типов float, double и long double. Функция принимает значение в радианах и возвращает синус. Пример использования:

```
#include <iostream>
#include <cmath>

int main()
{
	std::cout << "sin(M_PI / 2) = " << std::sin(M_PI / 2);

	std::cout << "\nsin(M_PI) = " << std::sin(M_PI);

	double angle_degrees = 30.0;
	std::cout << "\nsin(30 degrees) = " << std::sin(angle_degrees * M_PI / 180.0);

	return 0;
}
```

M_PI — это приблизительное значение числа пи. M_PI и sin() определены в заголовочном файле <cmath>. Программа выводит значения синуса для углов M_PI / 2 и M_PI в радианах, а также для угла в 30 градусов, конвертированного в радианы.

Вывод программы:

```
sin(M_PI / 2) = 1
sin(M_PI) = 1.22465e-16
sin(30 degrees) = 0.5
```

Синус числа пи равен нулю, но т. к. мы используем приблизительное значение пи, мы получаем число приблизительно равное нулю, то есть $1.22465 * 10^{-16} \approx 0$. В случае M_PI / 2 произошло округление до 1.

Итак, выходное значение имеет тот же тип данных что и входное, функция принимает значение в радианах. Этих правил мы будем придерживаться при собственной реализации синуса.

4.2 Табличные значения

Реализуем функцию синус с помощью табличного метода. Сперва нужно создать таблицу:

```
for (int n = 0; n < 91; n++) {
	double angle = n * M_PI / 180.0;
	std::cout << std::sin(angle) << ",\n";
}
```

Для таблицы я использую значения синуса углов от 0° до 90° включительно. Каждый целый градус переводится в радианы и передаётся функции sin(), после чего значение выводится на экран.

Вывод программы:

```
0,
0.0174524,
0.0348995,
...
0.999391,
0.999848,
1,
```

Наша функция будет принимать значения в радианах типа double, конвертировать в градусы, брать целую часть и возвращать значение из таблицы. Но входное значение нужно также привести в промежуток [0°, 90°]. Для этого создадим функцию reduce_angle(), принимающую значение угла в радианах типа double, и возвращающую угол в промежутке $[0; \frac{\pi}{2}]$, синус которого по модулю равен синусу изначального угла.

```
double reduce_angle(double angle) {
	double abs_angle = std::abs(angle);

	while (abs_angle > M_PI) {
		abs_angle -= M_PI;
	}

	if (abs_angle > M_PI/2) {
		abs_angle = M_PI - abs_angle;
	}
	return abs_angle;
}
```

При этом мы теряем информацию о знаке синуса. Например, пусть входное значение — это $\frac{3\,M\_PI}{2}$, тогда выходное значение будет равно $\frac{M\_PI}{2}$, но $\sin\left(\frac{3\,M\_PI}{2}\right)=-1$, а $\sin\left(\frac{M\_PI}{2}\right)=1$. Поэтому создадим вспомогательную функцию sine_sign(), которая будет определять знак синуса входного угла:

```
int sine_sign(double angle) {
	double abs_angle = std::abs(angle);
	bool half_period_even = (int(abs_angle / M_PI) % 2 == 0);

	if (!half_period_even) return -Sign(angle);

	return Sign(angle);
}
```

Переменная half_period_even означает является ли полупериод, в котором находится угол чётным или нет. Если да, то значение синуса в этом периоде будет положительным, если нет — отрицательным, важно понимать, что отсчёт происходит от 0 включительно, то есть, если abs_angle < M_PI, то угол находится в нулевом периоде. Но это работает только для положительного угла, для отрицательного всё наоборот, поэтому результат умножается на знак входного значения. Эти две функции находятся в файле «sine_helper.cpp»

Теперь работу функции можно показать на блок-схеме (схема 1):

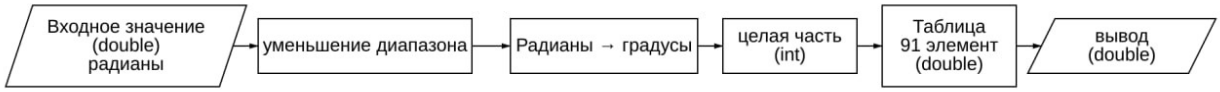

Схема 1

Определяем таблицу:

```
double table[91] = {
        0.0,
        0.0174524,
        ...
        0.999848,
        1.0
}
```

Определяем функцию:

```
double table_sine(double x) {
        int reduced_x = Degrees(reduce_angle(x));
        return table[reduced_x] * sine_sign(x);
}
```

Функция Radians() конвертирует градусы в радианы, а функция Degrees() - наоборот. Эти две функции определены в файле «basic_math.cpp»

Переменная reduced_x — это угол в промежутке [0°, 90°], функция возвращает соответствующее ей значение в таблице, умноженное на знак синуса входного угла.

С помощью функции draw_func(), определённой в файле «draw_func.cpp», можно рисовать график функции, передавая её как аргумент. График данной функции выглядит так (рисунок 11):

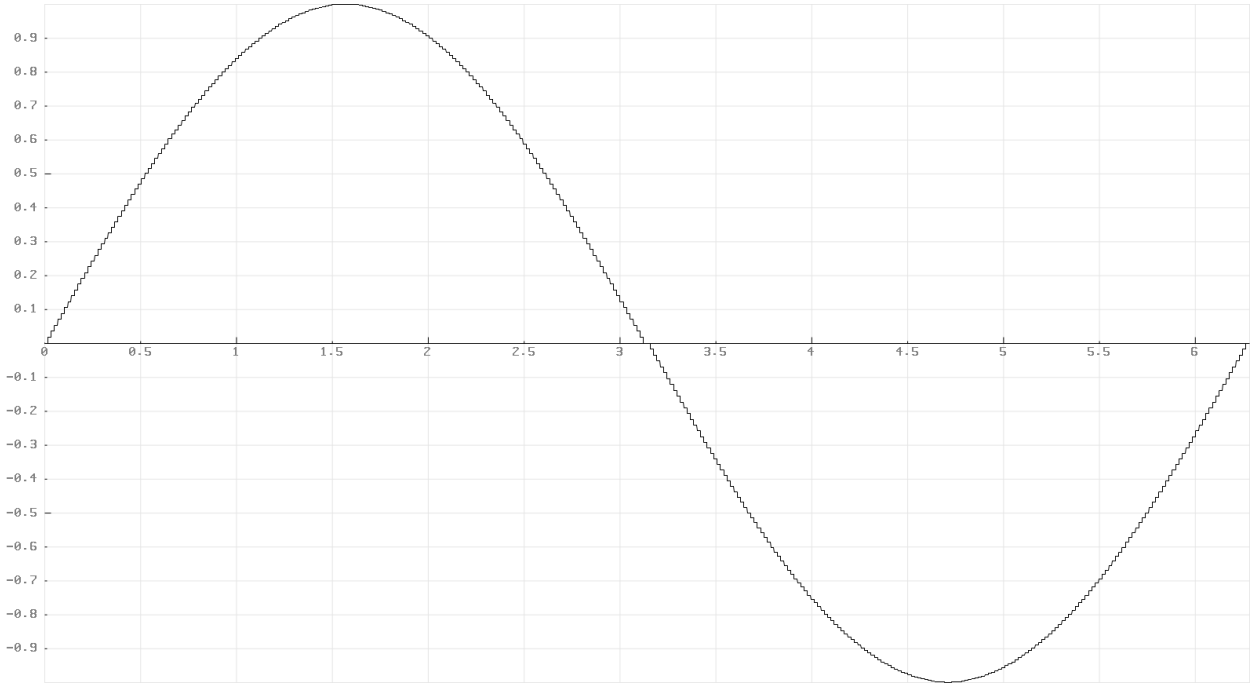

Рисунок 11

Линия зубчатая, т. к. промежуточные (нецелые) значения не имеют соответствующих им значений в таблице, так, для угла 15.6° результат тот же, что и для 15°. То есть т. к. reduced_x – int, то только целые углы получают новое значение. Для того,

чтобы сделать функцию более "гладкой", что соответствует реальной синусоиде, можно использовать линейную интерполяцию.

Как мы уже знаем, линейная интерполяция соединяет точки прямолинейными отрезками. Пусть x_1 — целое значение угла в градусах, $x_2 = x_1 + 1°$ и x — абсцисса точки (x, y) между точками $(x_1, f(x_1))$ и $(x_2, f(x_2))$, тогда y можно выразить через известные переменные при помощи подобия треугольников: $y = f(x_2)\frac{x - x_1}{1°}$ как частный случай если $x_1 = 0$. В общем виде формула выглядит так: $y = f(x_1) + (f(x_2) - f(x_1))\frac{x - x_1}{1°}$ (рисунок 12). Для упрощения чтения введём переменные:

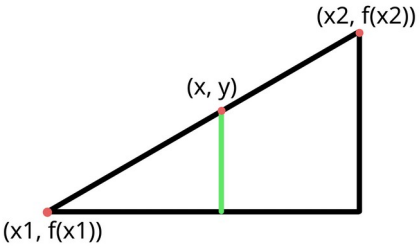

Рисунок 12

$f(x_2) - f(x_1)$ — это разность ординат (y_change),

$\frac{x - x_1}{1°}$ — это коэффициент подобия (k).

Таким образом, $y = f(x_1) + \text{y\_change} \cdot k$. Если уменьшить диапазон, то т. к. информация о знаке синуса теряется: $y = \text{sine\_sign(x)} \times (f(x_1) + \text{y\_change} \cdot k)$.

```
double table_sine_interp(double x) {
	double reduced_x = reduce_angle(x);

	double x1 = Radians(int(Degrees(reduced_x)));
	double x2 = Radians(int(Degrees(reduced_x) + 1));

	double y_change = table_sine(reduced_x + Radians(1)) - table_sine(reduced_x);
	double k = (reduced_x - x1) / Radians(1);

	return sine_sign(x) * (table_sine(reduced_x) + y_change * k);
}
```

Функция table_sine_interp() так же принимает и возвращает значения в радианах. Теперь график функции выглядит так (рисунок 13):

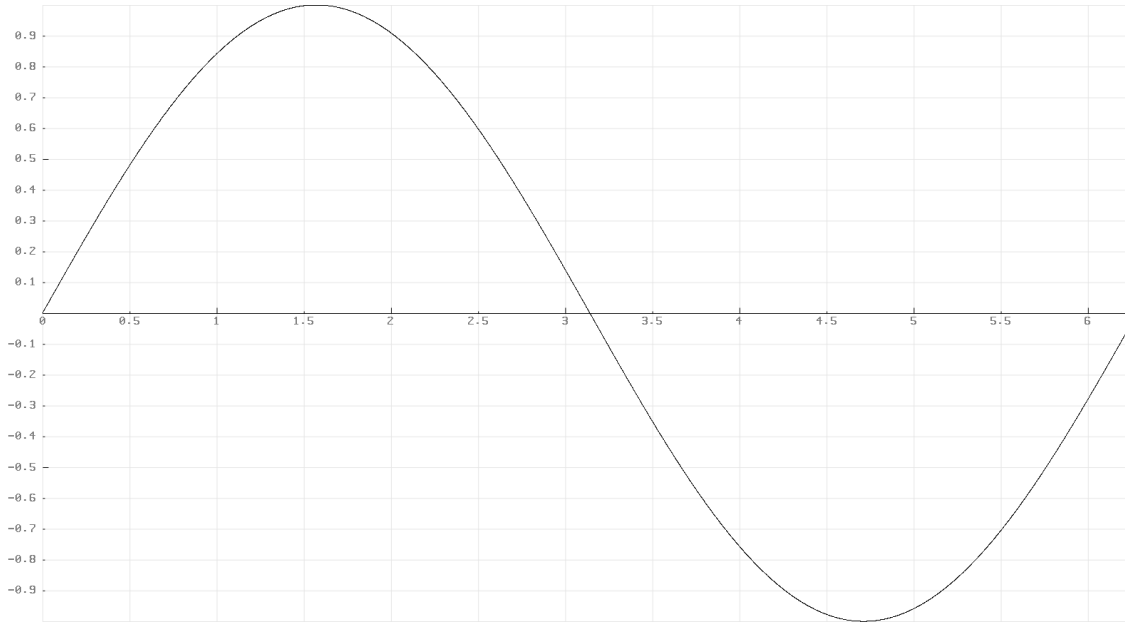

Рисунок 13

4.3 Ряд Тейлора

Чтобы изменить точность вычислений алгоритма, который использует табличные значения, нам приходится менять тип интерполяции или количество элементов в таблице. Ряд Тейлора позволяет позволяет динамично изменять точность вычислений, изменяя количество слагаемых ряда. Поэтому вводится новая переменная n — количество слагаемых ряда.

Формулу $\sin(x)=\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)!}x^{2n+1}$ для удобства разобьём на три переменные:

$a=(-1)^n$, $b=x^{2n+1}$, $c=(2n+1)!$. Теперь каждое слагаемое равно $\frac{ab}{c}$.

Определим функцию:

```
double taylor_siries_sine(double x) {
	unsigned n = 1; //замените своим значением
	double siries_value = 0.0;

	for (unsigned i = 0; i < n; i++)
	{
		int a = std::pow(-1, i);
		double b = std::pow(x, 2 * i + 1);
		unsigned c = Factorial(2 * i + 1);
		siries_value += a * b / c;
	}
	return siries_value;
}
```

Вот как выглядят графики данной функции с n = 1, 2, 3, 4 (рисунок 14, 15, 16, 17):

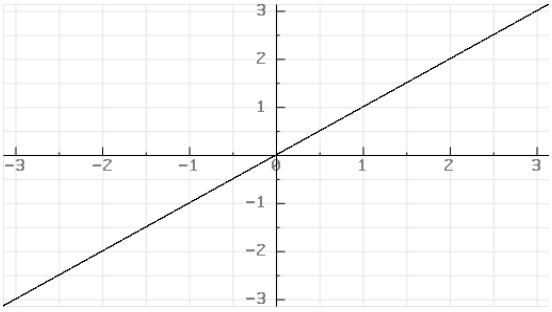

Рисунок 14. n = 1

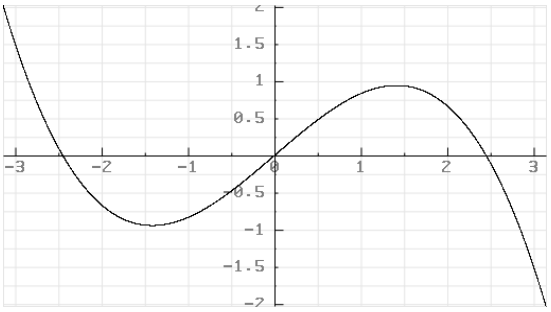

Рисунок 15. n = 2

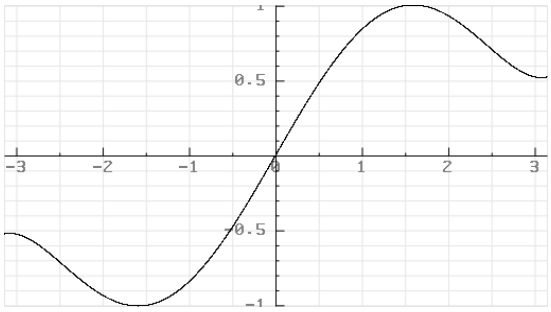

Рисунок 16. n = 3

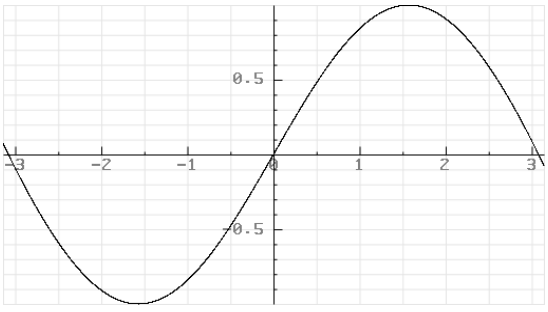

Рисунок 17. n = 4

Чем дальше аргумент от 0, тем сильнее расходятся ряд и функция. Чтобы это исправить, воспользуемся функциями для уменьшения диапазона из предыдущего параграфа.

```
double taylor_siries_sine(double x) {
	unsigned n = 1; //замените своим значением

	double reduced_x = reduce_angle(x);

	double siries_value = 0.0;
	for (unsigned i = 0; i < n; i++)
	{
		int a = std::pow(-1, i);
		double b = std::pow(reduced_x, 2 * i + 1);
		unsigned c = Factorial(2 * i + 1);
		siries_value += a * b / c;
	}
	return siries_value * sine_sign(x);
}
```

Теперь даже с небольшим значением n, график функции будет выглядеть как правильная синусоида.

На рисунке 18 изображён график данной функции с n = 3. Даже со столь малым значением n, функция выдаёт хороший результат. Более того, не стоит давать n слишком большое значение: каждое следующее слагаемое ряда вносит всё меньшее уточнение на промежутке $[0; \frac{\pi}{2}]$, а также при через-чур больших n, программа прервётся, выдав ошибку.

Данный метод может быть использован в ситуациях, когда требуется большая точность вычислений, хотя это и не самый эффективный вариант.

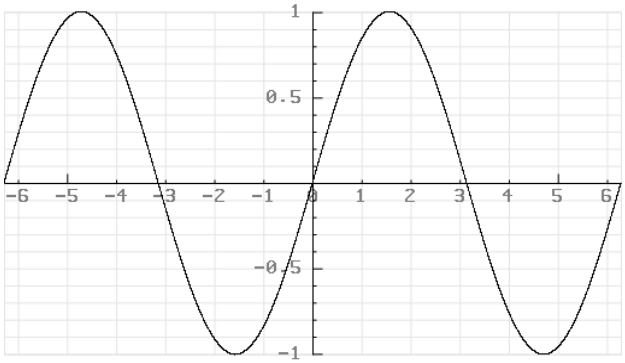

Рисунок 18

4.4 CORDIC алгоритм

Точность данного метода напрямую зависит от количества итераций. Так как от количества итераций зависит количество заранее просчитанных углов вращения, динамично менять точность невозможно. В моей реализации данного метода количество итераций равно 15.

Создадим программу, которая выводит углы вращения и константу k - произведение косинусов данных углов, с помощью формулы $\varphi_i = \arctan\left(\frac{1}{2^{i-1}}\right)$:

```
int main() {
	std::cout.precision(20);
	std::cout << std::fixed;

	double k = 1;
	for (int i = 0; i < 15; i++) {
		double angle = std::atan(std::pow(2, -i));
		std::cout << angle << ",\n";
		k *= std::cos(angle);
	}
	std::cout << "\nk = " << k << '\n';
}
```

atan() — это функция арктангенс, определённая в заголовке <cmath>. Выходные значения для углов вращения представлены в радианах.

Вывод программы:

```
0.78539816339744827899949086713604629039764404296875,
0.46364760900080609351547877849952783435583114624023,
...
0.00012207031189367020785306594543584424172877334058,
0.00006103515617420877259350145416227917394280666485,

k = 0.607252935385913628074661119172
```

Определим таблицу, хранящую углы вращения:

```
double angles_lut[15] = {
        0.78539816339744827899949086713604629039764404296875,
        0.46364760900080609351547877849952783435583114624023,
        ...
        0.00012207031189367020785306594543584424172877334058,
        0.00006103515617420877259350145416227917394280666485,
};
```

Функция будет принимать значение в радианах типа double, вращать вектор и возвращать его ординату. Алгоритм её работы в виде блок-схемы представлен на схеме 2:

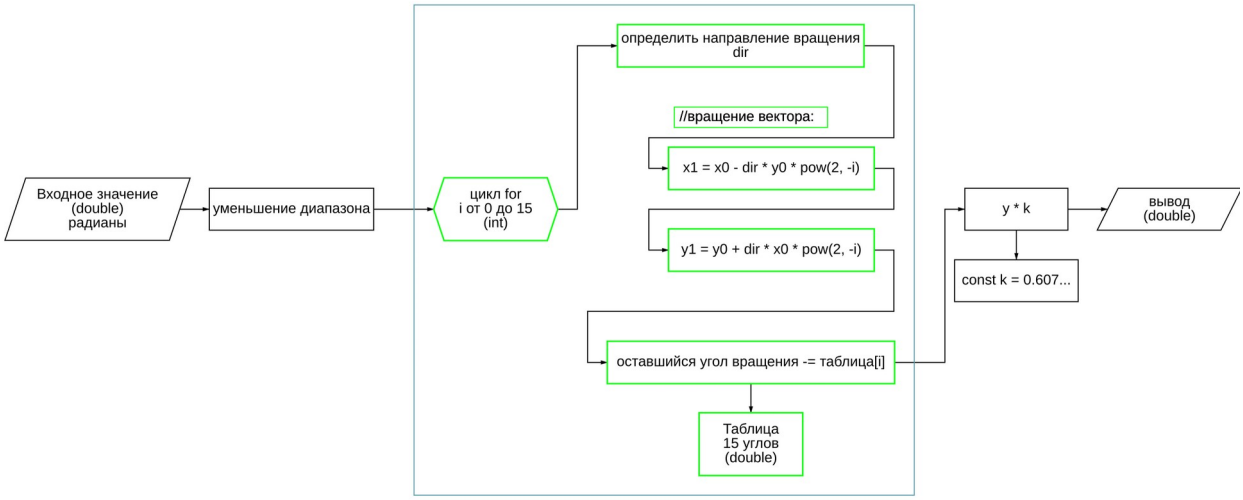

Схема 2

Определим функцию:

```
double cordic_sine(double angle) {
        //уменьшение диапазона
        const double reduced_angle = reduce_angle(angle);
        const int func_sign = sine_sign(angle);

        //константы для CORDIC алгоритма
        constexpr double K = 0.607252935385914;
        constexpr int iterations = 15;

        double x = 1.0, y = 0.0, z = reduced_angle;
...
```

Переменная z — это оставшийся угол вращения, изначально он равен входному значению. Если z > 0, то суммарный угол вращения меньше входного, вращение будет происходить против часовой стрелки, если же z < 0, то вектор повернулся больше чем

надо, и вращение будет происходить по часовой стрелке. В наших формулах поворота вектора, положительное направление вращения соответствует вращению по часовой стрелке. Следовательно, направление равно знаку z.

```
…
        for (int i = 0; i < iterations; ++i) {
                //выбор направления
                int direction = Sign(z);

                //вращение вектора
                double new_x = x - direction * y * std::pow(2, -i);
                double new_y = y + direction * x * std::pow(2, -i);

                x = new_x;
                y = new_y;
                z -= direction * angles[i];
        }
        return y * K * func_sign;
}
```

Так как в C++ нет стандартного числа с фиксированной точкой, а числа с плавающей запятой не поддерживают битовый сдвиг, приходится умножать на 2^{-i}. В файле «cordic_fixed_point.cpp» данная функция использует числа с фиксированной точкой, благодаря чему вычисления происходят быстрее.

Так как CORDIC алгоритм использует только базовые операции сложения и сдвига, его можно использовать в ситуациях, когда технические ограничения не позволяют использовать операторы умножения, деления, возведения в степень и т. д.

Заключение

Главной целью было изучение основных методов вычисления синуса. В рамках данной проектной работы были рассмотрены различные методы вычисления синуса, а также изучены его свойства и применены в соответствующих реализациях. В процессе изучения были представлены и реализованы следующие методы:

1. Табличный метод

2. Ряд Тейлора

3. CORDIC метод

Каждый из предложенных методов имеет свои преимущества и недостатки, и выбор зависит от конкретной задачи. Например, ряд Тейлора обеспечивает высокую точность вычислений, но является весьма ресурсозатратным.

Результаты данной работы могут быть использованы учениками, учителями и студентами для:

1. В учебных целях метод ряда Тейлора может быть использован для демонстрации разложения функции в ряд. Это часто применяется в курсах математики и численных методов.

2. Табличный метод может быть использован как пример применения интерполяции для нахождения всей функции, зная некоторое количество её точек.

3. CORDIC метод можно использовать для объяснения построения алгоритмов или как пример того, как можно вращать вектор с помощью базовых операций.

Итак, в ходе работы цель была достигнута, а также были выполнены все задачи, а именно:

1. Было дано определение синусу угла и синусу числа.

2. Были описаны некоторые свойства синуса, такие как: его производная, периодичность и т. д.

3. Были рассмотрены основные методы вычисления синуса.

4. Основные методы были реализованы на языке программирования C++.

Все исходные файлы, изображения и т. д. можно найти на репозитории на GitHub: https://github.com/DuyhaBeitz/ComputingSine/

Список литературы

Исследование численных методов для синуса и косинуса [Текст] / Строганов Ю.В., Пудов Д.Ю., Сиденко А.Г. // Новые информационные технологии в автоматизированных системах. 2018. №21. URL: (https://cyberleninka.ru/article/n/issledovanie-chislennyh-metodov-dlya-sinusa-i-kosinusa).

Approximating sin(x) to 5 ULP with Chebyshev polynomials [Электронный ресурс], — (https://mooooo.ooo/chebyshev-sine-approximation/).

Формулы Маклорена и Тейлора [Электронный ресурс], — (https://www.webmath.ru/poleznoe/formules_8_19.php).

Mercator series [Электронный ресурс], — (https://en.wikipedia.org/wiki/Mercator_series).

Taylor series [Электронный ресурс], — (https://en.wikipedia.org/wiki/Taylor_series).

Derivative [Электронный ресурс], — (https://en.wikipedia.org/wiki/Derivative).

CORDIC [Электронный ресурс], — (https://en.wikipedia.org/wiki/CORDIC).

CORDIC Algorithm [Электронный ресурс], — (https://youtu.be/m1e8IbDsIKw?si=UuKnD7S3-xcOGWHW).