**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

**к курсовой работе**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: **«Реализация алгоритма CORDIC»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 2309 |  | Савин П. А. |
| Преподаватель |  | Пестерев Д. О. |

Санкт-Петербург

2023

**ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

Студент: Савин П. А.

Группа: 2309

Тема работы: Реализация алгоритма CORDIC

Исходные данные:

Вычислить значения тригонометрических, обратных тригонометрических, гиперболических функций, а также значения экспоненты и натурального логарифма в заданной точке.

Содержание пояснительной записки:

Титульный лист, задание, аннотация, содержание, введение, теоретические сведения, особенности реализации, оценка экспериментально полученных данных, листинг программы, заключение.

Предполагаемый объем пояснительной записки:

Не менее 25 страниц.

Дата выдачи задания: 15.11.2023

Дата сдачи курсовой работы: 24.12.2023

Дата защиты курсовой работы: 27.12.2023

|  |  |  |
| --- | --- | --- |
| Студент гр. 2309 |  | Савин П. А. |
| Преподаватель |  | Пестерев Д. О. |

**АННОТАЦИЯ**

В работе представлен обзор семейства алгоритмов CORDIC для вычисления продвинутых функций в заданной точке. Рассмотрены теоретические аспекты алгоритма, особенности реализации в коде и сравнение полученных значений с более современным методом вычисления — рядами Тейлора.

**SUMMARY**

This paper presents an overview of the family of CORDIC algorithms for computing advanced functions at a given point. Theoretical aspects of the algorithm, peculiarities of implementation in code and comparison of the obtained values with a more modern method of calculation - Taylor series - are considered.

Введение.

Семейство алгоритмов CORDIC является одним из самых используемых методов вычисления сложных операций (тригонометрические и гиперболические функции, квадратные корни, произведение, деление, экспоненты и натуральные логарифмы) на низкоуровневых вычислительных устройствах. Пусть и прошло много лет с момента разработки этого метода (первая итерация алгоритма была представлена в 1956 году Джеком Волдером), он все еще широко используется в микроконтроллерах и программируемых логических интегральных схемах (ПЛИС).

Теоретические сведения.

В основе метода лежит перевод сложных операций в композицию базовых — простые микроконтроллеры обычно не имеют двоичного умножителя, из операций им предоставлено сложение, вычитание, битовый сдвиг и таблица поиска (результаты интерполяции функции, примером такой таблицы является таблица логарифмов).

Известно, что система

обеспечивает поворот вектора на взятый угол .

Если вынести косинус в выражении

и выбрать такой угол поворота, что , можно заметить, что умножение на тангенс в системе сводится к битовому сдвигу на разрядов. Следовательно, если представить некоторый произвольный угол , операцию поворота можно разложить в серию последовательно выполненых элементарных преобразований, при этом косинус не зависит от направления поворота, исходя из равенства . Как это выглядит в конечной реализации:

Возьмем начальный вектор .

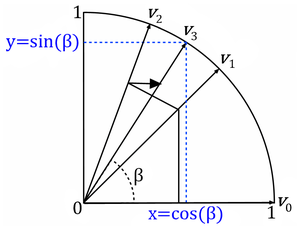
Применим алгоритм: на каждой итерации будем поворачивать исходный вектор — — матрица поворота.

По основным тригонометрическим тождествам:

Тогда матрица поворота принимает следующий вид:

А вектор на каждой итерации будет выглядеть так:

Выглядит это так:



Ограничиваем тангенс до и получаем:

где — так называемый коэффициент масштабирования, а – оператор, определяющий направление поворота (против часовой стрелки или по часовой стрелке) простым образом: если очередной взятый угол больше желаемого, поворачивать будем по часовой стрелке, в противном случае — против часовой стрелки. На деле, коэффициент масштабирования можно опустить для каждой итерации, а потом представить ответ в следующем виде (для косинуса и синуса):

Дополнительные сведения, помогающие в теории:

* Тангенс и котангенс не вычисляются алгоритмом, а находятся делением синуса на косинус и наоборот
* Значения арктангенсов и коэффициентов масштабирования являются для микроконтроллеров теми самыми таблицами поиска, то есть, известны заранее

Для вычисления гиперболических функций используется не арктангенс, а ариатангенс, поэтому число превращается в число . Также был выведен метод «двойных итераций», заключающийся в повторении одного номера два раза. Этот метод был обобщен при рассмотрении вопроса сходимости алгоритма с истинным значением в произвольных системах счисления советскими учеными В. Д. Байковым и В. Б. Смоловым: пусть имеется система счисления с основанием R, тогда для функций sin, cos, arctg достаточно выполнения (R-1) итераций при каждом значении i (i от 0 или 1 до n, где n - разрядность чисел), т.е. для каждого разряда результата. Для функций ln, exp, sh, ch, arth следует выполнять R итераций для каждого значения i. Для функций arcsin и arccos следует выполнять 2\*(R-1) итераций на каждый разряд, т.е. при каждом значении i. Для функций arsh, arch количество итераций составит 2R для каждого i, т.е. для каждого разряда результата.

Особенности реализации.

Поскольку для микроконтроллеров арктангенсы и коэффициенты масштабирования являются таблицами поиска, то и в программе эти значения также предстанут массивами ранее вычисленных значений. Также в программе будет не умножение на конечный коэффициент масштабирования, а изначальное приравнивание к конечному коэффициенту. Дополнительными ограничениями стали диапазон вычислений и ограничение на число итераций: число итераций ограничено разрядность double чисел (64 разряда), а алгоритм работает только в промежутке , поэтому и все вводимые значения должны быть из этого промежутка. Помимо ограничений, обусловимся тем, что в алгоритме все же будет умножение.

Оценка экспериментально полученных данных.

Оценка будет заключаться в сравнении разницы полученных значений алгоритмами CORDIC и рядами Тейлора: ряды обеспечивают точность большего порядка, следовательно, разницу между ними можно считать за оценку ошибки вычисления. Для примера зафиксируем точку 0.5, а для натурального логарифма — 1.5, и посчитаем значения за 1, 2, 5, 10, 15, 20, 25 и 30 итераций двумя способами. Результаты исследования приведны ниже (оси ординат соответствует значение функции в точке, оси абсцисс — число итераций).

Все графики имеют общую тенденцию: судя по данным вычислений, вычисление рядами очень быстро упирается в предел точности хранения double чисел (примерно на 5-6 итерации), при этом CORDIC даже на 30 итерации продолжает приближение к «истинному» значению. Подробные данные приложены в Excel таблице.

Листинг программы.

#include <iostream>

#include <iomanip>

#define M\_PI 3.14159265358979323846

#define M\_E 2.71828182845904523536

/\* pre-calculated table of arctan(2^-i); if i >= 25, arctan(2^-i) is equivalent to 2^-i \*/

static const double arctan\_table[64] = {

7.853981633974483E-1,4.636476090008061E-1,2.449786631268642E-1,1.243549945467614E-1,

6.241880999595735E-2,3.123983343026828E-2,1.562372862047683E-2,7.812341060101111E-3,

3.906230131966972E-3,1.953122516478819E-3,9.765621895593194E-4,4.882812111948983E-4,

2.441406201493618E-4,1.220703118936702E-4,6.103515617420878E-5,3.05175781155261E-5,

1.525878906131576E-5,7.62939453110197E-6,3.814697265606496E-6,1.907348632810187E-6,

9.536743164059609E-7,4.768371582030889E-7,2.38418579101558E-7,1.192092895507807E-7,

5.960464477539055E-8,2.98023223876953E-8,1.490116119384766E-8,7.450580596923828E-9,

3.725290298461914E-9,1.862645149230957E-9,9.313225746154785E-10,4.656612873077393E-10,

2.328306436538696E-10,1.164153218269348E-10,5.820766091346741E-11,2.91038304567337E-11,

1.455191522836685E-11,7.275957614183426E-12,3.637978807091713E-12,1.818989403545856E-12,

9.094947017729282E-13,4.547473508864641E-13,2.273736754432321E-13,1.13686837721616E-13,

5.684341886080801E-14,2.842170943040401E-14,1.4210854715202E-14,7.105427357601002E-15,

3.552713678800501E-15,1.77635683940025E-15,8.881784197001252E-16,4.440892098500626E-16,

2.220446049250313E-16,1.110223024625157E-16,5.551115123125783E-17,2.775557561562891E-17,

1.387778780781446E-17,6.938893903907228E-18,3.469446951953614E-18,1.734723475976807E-18,

8.673617379884035E-19,4.336808689942018E-19,2.168404344971009E-19,1.084202172485504E-19 };

/\* pre-calculated table of K(n) where K(n) is product(1 / sqrt(1 + 2^-2\*i)) for i from 0 to n - 1 \*/

static const double k\_table[64] = {

7.071067811865475E-1,6.324555320336759E-1,6.135719910778963E-1,6.088339125177524E-1,

6.076482562561681E-1,6.073517701412958E-1,6.072776440935258E-1,6.072591122988925E-1,

6.07254479332562E-1,6.072533210898748E-1,6.072530315291339E-1,6.072529591389442E-1,

6.072529410413965E-1,6.072529365170094E-1,6.072529353859126E-1,6.072529351031383E-1,

6.072529350324446E-1,6.072529350147711E-1,6.072529350103526E-1,6.072529350092478E-1,

6.072529350089715E-1,6.072529350089023E-1,6.072529350088848E-1,6.072529350088803E-1,

6.072529350088791E-1,6.072529350088786E-1,6.072529350088783E-1,6.072529350088781E-1,

6.072529350088778E-1,6.072529350088776E-1,6.072529350088773E-1,6.072529350088771E-1,

6.072529350088768E-1,6.072529350088765E-1,6.072529350088762E-1,6.072529350088759E-1,

6.072529350088756E-1,6.072529350088753E-1,6.07252935008875E-1,6.072529350088746E-1,

6.072529350088743E-1,6.07252935008874E-1,6.072529350088736E-1,6.072529350088733E-1,

6.072529350088729E-1,6.072529350088725E-1,6.072529350088721E-1,6.072529350088717E-1,

6.072529350088713E-1,6.072529350088709E-1,6.072529350088705E-1,6.0725293500887E-1,

6.072529350088696E-1,6.072529350088692E-1,6.072529350088687E-1,6.072529350088683E-1,

6.072529350088678E-1,6.072529350088673E-1,6.072529350088668E-1,6.072529350088663E-1,

6.072529350088658E-1,6.072529350088653E-1,6.072529350088648E-1,6.072529350088642E-1 };

/\* pre-calculated table of arctanh(2^-(i+1)); \*/

static const double arctanh\_table[64] = {

5.493061443340549E-1,2.554128118829953E-1,1.25657214140453E-1,6.258157147700301E-2,

3.126017849066699E-2,1.562627175205221E-2,7.812658951540421E-3,3.906269868396826E-3,

1.95312748353255E-3,9.765628104410358E-4,4.882812888051128E-4,2.441406298506386E-4,

1.220703131063297E-4,6.103515632579116E-5,3.051757813447389E-5,1.525878906368418E-5,

7.629394531398012E-6,3.814697265643522E-6,1.907348632814768E-6,9.536743164065301E-7,

4.768371582032244E-7,2.384185791015377E-7,1.192092895507544E-7,5.960464477539066E-8,

2.980232238766821E-8,1.490116119384766E-8,7.450580596923828E-9,3.725290298407704E-9,

1.862645149176747E-9,9.313225745612684E-10,4.656612873077393E-10,2.32830643613212E-10,

1.164153218303229E-10,5.820766086010433E-11,2.910383045694546E-11,1.455191517420968E-11,

7.275957559986552E-12,3.637978752884913E-12,1.818989349336575E-12,9.094946475630264E-13,

4.547472966764072E-13,2.27373675443245E-13,1.136867835115106E-13,5.684341886080882E-14,

2.842165522029559E-14,1.421080050509343E-14,7.105427357601014E-15,3.552713678800504E-15,

1.776302629291627E-15,8.881242095915012E-16,4.440349997414384E-16,2.21990394816407E-16,

1.110223024625157E-16,5.545694112263355E-17,2.770136550700464E-17,6.938893903907228E-18,

6.938893903907228E-18,0.0E0,0.0E0,0.0E0,0.0E0,0.0E0,0.0E0,0.0E0 };

/\* pre-calculated table of indexes where double iteration mode is used \*/

static const int arctan\_twice\_index[4] = { 3,12,39,120 };

/\* pre-calculated table of K(n) where K(n) is product(1 / sqrt(1 + 2^-2\*(i + 1))) for i from 0 to n - 1 \*/

static const double kh\_table[64] = {

1.154700538379252E0,1.192569587999888E0,1.201997162280557E0,1.206710876642441E0,

1.207300522842615E0,1.207447925385481E0,1.207484775458747E0,1.207493987941918E0,

1.207496291060514E0,1.207496866840026E0,1.207497010784895E0,1.207497046771112E0,

1.207497064764221E0,1.207497067013359E0,1.207497067575644E0,1.207497067716215E0,

1.207497067751358E0,1.207497067760143E0,1.20749706776234E0,1.207497067762889E0,

1.207497067763026E0,1.20749706776306E0,1.207497067763069E0,1.207497067763071E0,

1.207497067763072E0,1.207497067763072E0,1.207497067763072E0,1.207497067763072E0,

1.207497067763072E0,1.207497067763072E0,1.207497067763072E0,1.207497067763072E0,

1.207497067763072E0,1.207497067763072E0,1.207497067763072E0,1.207497067763072E0,

1.207497067763072E0,1.207497067763072E0,1.207497067763072E0,1.207497067763072E0,

1.207497067763072E0,1.207497067763072E0,1.207497067763072E0,1.207497067763072E0,

1.207497067763071E0,1.207497067763071E0,1.207497067763071E0,1.207497067763071E0,

1.207497067763071E0,1.207497067763071E0,1.207497067763071E0,1.207497067763071E0,

1.207497067763071E0,1.207497067763071E0,1.207497067763071E0,1.207497067763071E0,

1.207497067763071E0,1.207497067763071E0,1.207497067763071E0,1.207497067763071E0,

1.207497067763071E0,1.207497067763071E0,1.207497067763071E0,1.207497067763071E0 };

double cos\_cordic(double t, int n) {

double x = k\_table[n - 1];

double xt;

double y = 0.0;

double z = 0.0;

double k = 1.0;

int i;

if (t < 0 || t > M\_PI / 2) {

return 0.0;

}

for (i = 0; i < n; ++i) {

xt = x;

if (t > z) {

x -= (y \* k);

y += (xt \* k);

z += (i >= 25 ? k : arctan\_table[i]);

}

else {

x += (y \* k);

y -= (xt \* k);

z -= (i >= 25 ? k : arctan\_table[i]);

}

k \*= 0.5;

}

return x;

}

double sin\_cordic(double t, int n) {

double x = k\_table[n - 1];

double xt;

double y = 0.0;

double z = 0.0;

double k = 1.0;

int i;

if (t < 0 || t > M\_PI / 2) {

return 0.0;

}

for (i = 0; i < n; ++i) {

xt = x;

if (t > z) {

x -= (y \* k);

y += (xt \* k);

z += (i >= 25 ? k : arctan\_table[i]);

}

else {

x += (y \* k);

y -= (xt \* k);

z -= (i >= 25 ? k : arctan\_table[i]);

}

k \*= 0.5;

}

return y;

}

double atan2\_cordic(double y, double x, int n) {

double xt;

double z = 0.0;

double k = 1.0;

int i;

if (x < 0.0 || y < 0.0 || y > x) {

return 0.0;

}

for (i = 0; i < n; ++i) {

xt = x;

if (y > 0.0) {

x += (y \* k);

y -= (xt \* k);

z += (i >= 25 ? k : arctan\_table[i]);

}

else {

x -= (y \* k);

y += (xt \* k);

z -= (i >= 25 ? k : arctan\_table[i]);

}

k \*= 0.5;

}

return z;

}

double cosh\_cordic(double t, int n) {

double x = kh\_table[n];

double y = 0;

double z = 0;

double k = 0.5;

double xt;

int i;

int twice\_ind\_ind = 0;

int twice\_ind = arctan\_twice\_index[twice\_ind\_ind];

if (t > 1.11 || t < -1.11) {

return 0.0;

}

for (i = 0; i < n; i++) {

xt = x;

if (t > z) {

x += y \* k;

y += xt \* k;

z += (i >= 25 ? k : arctanh\_table[i]);

}

else {

x -= y \* k;

y -= xt \* k;

z -= (i >= 25 ? k : arctanh\_table[i]);

}

if (i == twice\_ind) {

xt = x;

if (t > z) {

x += y \* k;

y += xt \* k;

z += (i >= 25 ? k : arctanh\_table[i]);

}

else {

x -= y \* k;

y -= xt \* k;

z -= (i >= 25 ? k : arctanh\_table[i]);

}

twice\_ind\_ind++;

twice\_ind = arctan\_twice\_index[twice\_ind\_ind];

}

k \*= 0.5;

}

return x;

}

double sinh\_cordic(double t, int n) {

double x = kh\_table[n];

double y = 0;

double z = 0;

double k = 0.5;

double xt;

int i;

int twice\_ind\_ind = 0;

int twice\_ind = arctan\_twice\_index[twice\_ind\_ind];

if (t > 1.11 || t < -1.11) {

return 0.0;

}

for (i = 0; i < n; i++) {

xt = x;

if (t > z) {

x += y \* k;

y += xt \* k;

z += (i >= 25 ? k : arctanh\_table[i]);

}

else {

x -= y \* k;

y -= xt \* k;

z -= (i >= 25 ? k : arctanh\_table[i]);

}

if (i == twice\_ind) {

xt = x;

if (t > z) {

x += y \* k;

y += xt \* k;

z += (i >= 25 ? k : arctanh\_table[i]);

}

else {

x -= y \* k;

y -= xt \* k;

z -= (i >= 25 ? k : arctanh\_table[i]);

}

twice\_ind\_ind++;

twice\_ind = arctan\_twice\_index[twice\_ind\_ind];

}

k \*= 0.5;

}

return y;

}

double exp\_cordic(double t, int n) {

double x = kh\_table[n];

double y = 0;

double z = 0;

double k = 0.5;

double xt;

int i;

int twice\_ind\_ind = 0;

int twice\_ind = arctan\_twice\_index[twice\_ind\_ind];

if (t > 1.11 || t < -1.11) {

return 0.0;

}

for (i = 0; i < n; i++) {

xt = x;

if (t > z) {

x += y \* k;

y += xt \* k;

z += (i >= 25 ? k : arctanh\_table[i]);

}

else {

x -= y \* k;

y -= xt \* k;

z -= (i >= 25 ? k : arctanh\_table[i]);

}

if (i == twice\_ind) {

xt = x;

if (t > z) {

x += y \* k;

y += xt \* k;

z += (i >= 25 ? k : arctanh\_table[i]);

}

else {

x -= y \* k;

y -= xt \* k;

z -= (i >= 25 ? k : arctanh\_table[i]);

}

twice\_ind\_ind++;

twice\_ind = arctan\_twice\_index[twice\_ind\_ind];

}

k \*= 0.5;

}

return x + y;

}

double atanh2\_cordic(double y, double x, int n) {

double z = 0;

double k = 0.5;

double xt;

int i;

int twice\_ind\_ind = 0;

int twice\_ind = arctan\_twice\_index[twice\_ind\_ind];

if (y < 0 || x < 0) {

return 0.0;

}

for (i = 0; i < n; i++) {

xt = x;

if (y > 0) {

x -= y \* k;

y -= xt \* k;

z += (i >= 25 ? k : arctanh\_table[i]);

}

else {

x += y \* k;

y += xt \* k;

z -= (i >= 25 ? k : arctanh\_table[i]);

}

if (i == twice\_ind) {

xt = x;

if (y > 0) {

x -= y \* k;

y -= xt \* k;

z += (i >= 25 ? k : arctanh\_table[i]);

}

else {

x += y \* k;

y += xt \* k;

z -= (i >= 25 ? k : arctanh\_table[i]);

}

twice\_ind\_ind++;

twice\_ind = arctan\_twice\_index[twice\_ind\_ind];

}

k \*= 0.5;

}

return z;

}

double ln\_cordic(double x, int n) {

return 2.0 \* atanh2\_cordic((x - 1.0) / (x + 1.0), 1, n);

}

int main() {

double t;

double x;

double y;

int i;

std::cout << "Input the theta angle (in range [-pi/2;pi/2]): ";

std::cin >> t;

std::cout << "Input the x coordinate (non-negative number): ";

std::cin >> x;

std::cout << "Input the y coordinate (non-negative number): ";

std::cin >> y;

std::cout << "Input the number of iterations (non-negative number up to 64): ";

std::cin >> i;

std::cout << std::setprecision(15);

std::cout << "sin(x) = " << sin\_cordic(t, i) << std::endl;

std::cout << "cos(x) = " << cos\_cordic(t, i) << std::endl;

std::cout << "tg(x) = " << sin\_cordic(t, i) / cos\_cordic(x, i) << std::endl;

std::cout << "ctg(x) = " << cos\_cordic(t, i) / sin\_cordic(x, i) << std::endl;

std::cout << "arctg(y/x) = " << atan2\_cordic(y, x, i) << std::endl;

std::cout << "sh(x) = " << sinh\_cordic(t, i) << std::endl;

std::cout << "ch(x) = " << cosh\_cordic(t, i) << std::endl;

std::cout << "arth(y/x) = " << atanh2\_cordic(y, x, i) << std::endl;

std::cout << "exp(x) = " << exp\_cordic(t, i) << std::endl;

std::cout << "ln(x) = " << ln\_cordic(t, i) << std::endl;

return 0;

}

Заключение.

Алгоритмы CORDIC до сих пор очень широко используемы: машинная графика, цифровая обработка сигналов, решение прикладных задач в науке, в частности, вычислительной математике, навигации. Эффективность по памяти и по вычислительным затратам в условиях сильных ограничений по мощностям, и при этом вычисляемое значение имеет высокую точность — в этом и суть CORDIC.

Список использованной литературы.

А. В. Захаров, В. М. Хачумов «Алгоритмы CORDIC. Современное состояние

и перспективы», стр. 354-359. URL: <http://skif.pereslavl.ru/%7Eabram/share/CD-ROM-final/e-book/e-book/1-4/03-Zakharov-Algoritmy-CORDIC-p-353.pdf>

В. Д. Байков, В. Б. Смолов «Специализированные процессоры: итерационные алгоритмы и структуры», стр. 134-143

Infineon Technologies, AP16105 Application Note «XC166 family: Software implementation of Trigonometric functions using CORDIC Algorithm». URL: <https://www.infineon.com/dgdl/AP1610510_CORDIC_Algorithm.pdf?fileId=db3a304313719f4f011372b47428008b>

Ссылка на репозиторий: <https://github.com/pashkev14/A_DS_CW>