МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Вычислительная техника»

Дисциплина «Высокопроизводительные вычисления»

**Лабораторная работа №2.**

**Исследование алгоритмических и программных методов**

**ускорения реализации функций вещественных переменных**

Выполнил:

студент группы ИВТАПбд-31

Кондратьев П.С.

Проверил:

Негода В. Н.

Ульяновск, 2019

Оглавление

[Варианты заданий 3](#_Toc5224002)

[Цель работы: 3](#_Toc5224003)

[Анализ поставленной задачи: 4](#_Toc5224004)

[Программная реализация 5](#_Toc5224005)

[Примечание 12](#_Toc5224006)

[Приложение 1. 13](#_Toc5224007)

[Приложение 2. 14](#_Toc5224008)

# Варианты заданий

|  |  |  |
| --- | --- | --- |
| № | Функция | Значение N, задающее предельную погрешность |
| 11 | *ch x* | 22 |

# Цель работы:

Изучение методов реализации функций от вещественных переменных, представленных степенными рядами. Приобретение умений и навыков варьирования соотношения «затраты памяти – время реализации» в рамках этих методов.

Данная цель предполагает решение следующих основных задач:

1. Изучение общей схемы алгоритма разложения ряда, заданное по варианту.
2. Написание нескольких программных реализаций (цикловых, без цикловых) и сравнение их производительности.

# Анализ поставленной задачи:

Заданная функция: ch(x)

Заданная точность: E = 2-22 ≈ 0,0000002384185791015625

Разложение заданной функции в ряд выполняется по следующей формуле:



По заданию лабораторной работы задан отрезок для исследования ряда [0,1]. Так как точка x=0 не входит в область определения функции, то рассмотрим разложение ch(x) на интервале (0,1].

# Программная реализация

**Разработка процедуры-функции контроля**

Напишем функцию обеспечивающую проверку корректности реализации вычисления ch(x).

/\* Проверяет погрешность функции для заданного аргумента. \*/

int flverify(PFLOAT p) {

float val, et, x;

int kr = 0;

for (x = 0; x < 1.0f; x += E) {

val = p(x); // Вычисляем значение проверяемой функции.

et = (exp(x) + exp(-x)) / 2; // Вычисляем значение эталонной функции.

if (fabs(val - et) < E) {

if (fabs(val - et) < E && kr == 1200) {

cout << "lx=" << x << " ";

return false;

}

kr++;

}

}

return true;

}

lx=0.000286102 FS\_math:0

lx=0.000286102 simple\_cycle\_no\_gorner:0

simple\_cycle\_gorner:1

simple\_no\_cycle\_gorner:1

lx=0.000286102 simple\_no\_cycle\_no\_gorner:0

Таб. 1. Результаты проверки корректности реализации вычисления ch(x)

В ходе испытаний получен диапазон, на котором достигается заданная точность. Заданная точность обеспечивается на промежутке (0.000286102; 1].

**Исследование времени вычисления для данных типа float**

Разработка на языке Си набора процедур реализации функции для случая использования данных типа float. В этот набор включаются следующие процедуры:

**Процедура FS1** с использованием вызовов функции, фигурирующей в качестве первого слагаемого в выражении варианта задания на лабораторную работу

float fS1(float x) {

return (exp(x) + exp(-x)) / 2;

}

**Процедура simple\_cycle\_no\_gorner** с циклом, построенным без использования схемы Горнера

/\* Определяет ch числа по разложению в ряд с циклом без схемы Горнера.\*/

float simple\_cycle\_no\_gorner(float x) {

float p, // Степень аргумента.

member, // Член ряда.

result; // Значение косеканса.

int n, // Подфакториальное выражение.

i; // Номер члена ряда.

result = member = p = 1; // Задаём нулевой член.

/\* Суммируем члены ряда. \*/

for(i = 1, n = 1; i < 15; i++, n++) {

p \*= x \* x;

member = p / factr(n \* 2); // Вычисляем член.

result += member; // Прибавляем член к сумме ряда.

}

return result;

}

Данная реализация наиболее затратная по времени, т.к приходиться в цикле вычислять каждый члена ряда.

**Процедура simple\_cycle\_gorner** для многочлена с циклом на основе схемы Горнера

float simple\_cycle\_gorner(float x) {

float s, p = x \* x; /\*Сумма первых членов ряда.\*/

unsigned i; /\*Номер члена ряда.\*/

s = A[14]; /\*Задаём нулевой член.\*/

for (i = 13; i >= 1; i--) /\*Суммируем члены ряда.\*/

s = s \* p + A[i];

return s \* x + 1.0f;

}

Плюсы данной реализации в том, что не требуется вычислять коэффициенты при членах ряда и возводить x в степень. Недостатком является обращение к массиву.

**Процедура simple\_no\_cycle\_gorner** с бесцикловой реализацией функции на основе выражения, представляющего схему Горнера многочлена.

float simple\_no\_cycle\_gorner(float x) {

float p = x \* x;

return 1.0f + (((((((((((((A[14] \* p + A[13]) \*p + A[12]) \* p + A[11]) \* p + A[10]) \* p + A[9]) \* p + A[8]) \* p + A[7]) \*p + A[6]) \* p +

A[5]) \* p + A[4]) \* p + A[3]) \* p + A[2]) \* p + A[1])\*x;

}

Преимущества данной реализации весьма значительны. Не требуется вычислять коэффициенты при членах ряда, возводить x в степень, также благодаря отсутствию цикла все команды выполняются последовательно.

**Процедура simple\_no\_cycle\_no\_gorner** с бесцикловой реализацией функции на основе выражения ряда без схемы Горнера.

float simple\_no\_cycle\_no\_gorner(float x) {

float p, member, result;

result = member = p = 1;

/\* Суммируем члены ряда. \*/

int n = 1;

p \*= x \* x;

member = p / factr(n \* 2);

result += member;

n = 2;

p \*= x \* x;

member = p / factr(n \* 2);

result += member;

n = 3;

p \*= x \* x;

member = p / factr(n \* 2);

n = 4;

p \*= x \* x;

member = p / factr(n \* 2);

result += member;

result += member;

n = 5;

p \*= x \* x;

member = p / factr(n \* 2);

result += member;

n = 6;

p \*= x \* x;

member = p / factr(n \* 2);

result += member;

n = 7;

p \*= x \* x;

member = p / factr(n \* 2);

result += member;

n = 8;

p \*= x \* x;

member = p / factr(n \* 2);

result += member;

n = 9;

p \*= x \* x;

member = p / factr(n \* 2);

result += member;

n = 10;

p \*= x \* x;

member = p / factr(n \* 2);

result += member;

n = 11;

p \*= x \* x;

member = p / factr(n \* 2);

result += member;

n = 12;

p \*= x \* x;

member = p / factr(n \* 2);

result += member;

n = 13;

p \*= x \* x;

member = p / factr(n \* 2);

result += member;

n = 14;

p \*= x \* x;

member = p / factr(n \* 2);

result += member;

n = 15;

p \*= x \* x;

member = p / factr(n \* 2);

result += member;

return result;

}

Данная реализация должна повысить скорость работы, за счёт того, что в функции нет циклов и все команды выполняются последовательно, но если сравнивать со схемой Горнера, затраты времени побольше, так как приходиться вычислять степени x.

Проведение измерений затрат времени вычисления функции. Составим сводную таблицу скорости работы различных реализаций. Оценку будем производить в клоках на один вызов функции.

|  |  |
| --- | --- |
| Функция | Время работы, нс |
| Release |
| FS1 | 21 |
| simple\_cycle\_no\_gorner | 763 |
| simple\_cycle\_gorner | 14 |
| simple\_no\_cycle\_gorner | 13 |
| simple\_no\_cycle\_no\_gorner | 800 |

Таб. 2 Результаты замеров времени для чисел типа float

Легко видеть, что реализации без использования циклов превосходят реализации использующие циклы. Также видно, что реализации с использованием схемы Горнера превосходят наивные. Реализация схемы Горнера без цикла выигрывает у всех других реализаций.

**Разработка функций обработки чисел с фиксированной точкой**

Число с фиксированной запятой (англ. fixed-point number) — формат представления вещественного числа в памяти ЭВМ в виде целого числа. При этом само число x и его целочисленное представление x′ связаны формулой: x= x′ \*z, где z — цена (вес) младшего разряда.

Учитывая требуемый порядок точности видно, что данные придётся хранить в типе long long.

Для работы с числами с фиксированной точкой использовались inline (встроенные) функции. Inline функция избавляет процессор прыгать в ячейку, по адресу которой начинается эта функция. Сам смысл inline состоит в том, чтобы вместо вызова функции подставить ее тело (код функции) в место, где она вызывается.

long long Z = 1 << N; //цена младшего разряда

//Встраиваемая функция преобразование числа формата float в формат с фиксированной точкой :

//Встраиваемая функция преобразование числа в формате с фиксированной точкой в формат float:

inline float fixToFloat(long long arg) {

return (float)arg / Z;

}

//Встраиваемая функция умножения чисел с фиксированной точкой::

inline long long fixMn(long long arg1, long long arg2) {

long long rs = arg1 \* arg2;

rs = rs / Z;

return rs;

}

Пояснения: При умножении двух чисел типа float, число знаков после запятой в результате будет равно сумме знаков после запятой в обоих множителях, поэтому, когда мы умножаем числа типа fix, результат необходимо умножить на цену младшего разряда.

Процедура генерации массива аргументов в двух форматах.

/\*Выполняет инициализацию коэффициентов схемы Горнера.\*/

void initGorner() {

int n, /\*Подфакториальное выражение.\*/

i, /\*Номер члена ряда.\*/

x;

/\* Расчёт коэффициентов. \*/

for (i = 1, n = 1, x = 0; i < 15; i++, n++, x += E) {

A[i] = 1.0f / factr(n \* 2); /\*Вычисляем коэффициент.\*/

FixA[i] = floatToFix(A[i]);

}

}

**Исследование времени вычисления для данных с фиксированной точкой**

Предварительно разрабатывается функция верификации int fixverify (PFIX p),

/\* Проверяет погрешность функции для заданного аргумента. \*/

int flverify(PFLOAT p) {

float val, et, x;

int kr = 0;

for (x = 0; x < 1.0f; x += E) {

val = p(x); // Вычисляем значение проверяемой функции.

et = (exp(x) + exp(-x)) / 2; // Вычисляем значение эталонной функции.

if (fabs(val - et) < E) {

if (fabs(val - et) < E && kr == 1200) {

cout << "lx=" << x << " ";

return false;

}

kr++;

}

}

return true;

}

**FixCyclGorner на основе реализации схемы Горнера с циклом**

/\* Определяет ch числа с помощью циклов и схемы Горнера. \*/

fix FixCyclGorner(fix x) {

fix s, p; /\*Сумма первых членов ряда.\*/

int i; /\*Номер члена ряда.\*/

s = FixA[14]; /\*Задаём нулевой член.\*/

p = fixMn(x, x);

for (i = 13; i >= 1; i--) /\*Суммируем члены ряда.\*/

s = fixMn(s, p) + FixA[i];

return (fixMn(s, x) + 1.0f);

}

**FixNoCyclGorner на основе бесцикловой реализации схемы Горнера**

/\* Определяет ch числа с помощью схемы Горнера без циклов. \*/

fix FixNoCyclGorner(fix x) {

fix x2 = fixMn(x, x);

return (fixMn(fixMn(fixMn(fixMn(fixMn(fixMn(fixMn(fixMn(fixMn(fixMn(fixMn(fixMn(fixMn(fixMn(FixA[14], x2) + FixA[13], x2) + FixA[12], x2) +

+FixA[11], x2) + FixA[10], x2) + FixA[9], x2) + FixA[8], x2) + FixA[7], x2) + FixA[6], x2) +

FixA[5], x2) + FixA[4], x2) + FixA[3], x2) + FixA[2], x2) + FixA[1], x) + 1.0f);

}

Проведём исследования времени работы:

|  |  |
| --- | --- |
| Функция | Время работы, нс |
| Release |
| FixCyclGorner | 13 |
| FixNoCyclGorner | 6 |

Таб. 3 Результаты замеров времени для чисел в формате фиксированной точкой

По данной таблице видно, что время работы с фиксированной точкой меньше, чем с плавающей. Это объясняется тем, что арифметические операции над числами с плавающей запятой выполняются существенно медленнее, чем над целыми.

Для быстродействия было решено получать значения унижения коэффициентов схемы Горнена, до того, как рассчитывать время работы, чтобы не затрачивать время на умножение.

void test(fix x) {

fix x2 = fixMn(x, x);

for (int i = 14; i >= 1; i--) {

temp[i] = temp[i + 1] + fixMn(FixA[i], x2);

}

}

fix TestFixNoCyclGorner(fix x) {

fix x2 = fixMn(x, x);

return (temp[14] + temp[13] + temp[12] + temp[11] + temp[10] + temp[9] + temp[8] + temp[7] + temp[6] + temp[5] + temp[4] + temp[3] + temp[2] + temp[1] + 1.0f);

}

**Исследование таблично-алгоритмических реализаций функций**

В основе таблично-алгоритмического метода лежит разбиение интервала аппроксимации функции f(x) на равные промежутки величиной h (рис.1):

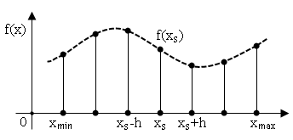


Рис.1. Разбиение интервала аппроксимации

В результате разбиения образуются значения аргумента xs, которые в дальнейшем называются табличными. Для каждого xs вычисляется значение функции f(xs), которое также называется табличным. Полученные значения xs и f(xs) сводятся в таблицу, которая размещается в памяти компьютера (рис.2). Если таблично-алгоритмический метод использует и производные вычисляемой функции, то аналогичным образом формируется таблица значений производных. Табличные значения функции и производных получают один раз на этапе разработки алгоритма.

float fS1(float x) {

return (exp(x) + exp(-x)) / 2;

}

float fS2(float x) {

return exp(x) / 2 - exp(-x) / 2;

}

float fS3(float x) {

return exp(x) / 2 + exp(-x) / 2;

}

float delta;

int szTable = 2221;

float table[3000][4], val[3000];

void stat(int szTable) {

delta = 1.0f / szTable;

cout << szTable << " " << (FixNoCyclGorner(floatToFix(bX)) < E ? 1 : 0) << " " << (fS1(bX) \* delta < E ? 1 : 0)

<< " " << (fS2(bX) \* delta \* delta < E ? 1 : 0) << " " << (fS3(bX) \* delta \* delta \* delta < E ? 1 : 0);

cout << "\n";

}

void genTable() {

delta = 1.0 / szTable;

float x = 0;

for (int i = 0; i < szTable; ++i, x += delta) {

val[i] = x;

table[i][0] = FixNoCyclGorner(floatToFix(x));

table[i][1] = fS1(x);

}

}

float acosTable(float x) {

int ind = (szTable - 1) \* x;

float h = x - val[ind];

return table[ind][0] + table[ind][1] \* h;

}

Замерим время работы в клоках на один запуск функции для таблично-алгоритмической реализации.

|  |  |
| --- | --- |
| Функция | Время работы, нс |
| Release |
| acosTable | 3 |

Таб. 4 Время работы таблично-алгоритмической реализации

Видно, что табличная реализация в конфигурации Release быстрее, бесцикловых реализаций схемы Горнера. Это понятно, так как в табличной реализации рассчитывается значение выражения: a0+a1\*x.

# Примечание

Параметры машины, в среде которой проводились измерения, приведены в **Приложение 1**.

Сведенье о результатах замеров времени вычисления, приведены в **Приложение 2**.

# Приложение 1.

Сводная таблица характеристик, тестируемых ЭВМ;

|  |  |
| --- | --- |
| Наименование характеристик | Компьютер 1 |
| Тип процессора | DualCore Intel Core i5-7200U |
| Исходная частота | 2500 МГц |
| Максимальная частота | 3100 MHz |
| Архитектура процессора | Kaby Lake-U |
| Cash-памяти | Кэш L1 — 2x32+2x32 Кб per core  Кэш L2 — 256x2 Кб per core (On-Die, ECC, Full-Speed)  Кэш L3 — 3 Мб (On-Die, ECC, Full-Speed) |
| Тип сокета | BGA1356 |
| Напряжение питания процессора | 0.6 V |
| Мощность процессора | 15 W |
| Количество ядер/потоков | 2/4 |
| Производитель | Asus |
| Материнская плата | Asus VivoBook X556UQK |
| Наличие и тип слотов PCI | PCI  PCI Express 2.0 |
| Тип видеокарты (встроенная, дополнительная) | Intel HD Graphics 620  NVIDIA GeForce 940MX |
| Тип шины | Встроено  PCI-устройство 8086-5916 / 1043-1490 (Rev 02) |
| Объем памяти видеокарты | 1 Гб  2 Гб |
| Физическая память ЭВМ | HGST HTS721010A9E630 (1 ТБ, 7200 RPM, SATA-III) |
| Оперативная память (объем, тип и скорость) | 8Гб SO-DIMM DDR4 15-15-15-35 4-50-17-8 2T  DDR3 SO-DIMM DDR4  2x 1ГГц (2.13ГГц) |
| Драйвер видеокарты | 23.20.16.4905 |
| Интерфейсы ввода-вывода | USB 2.0, USB 3.0, USB type C, HDMI, microSD, Audio 3.5, LAN, VGA |
| ОС | Windows 10 |

# Приложение 2.

|  |  |
| --- | --- |
| Функция | Время работы, нс |
| FS1 | 21 |
| simple\_cycle\_no\_gorner | 763 |
| simple\_cycle\_gorner | 14 |
| simple\_no\_cycle\_gorner | 13 |
| simple\_no\_cycle\_no\_gorner | 800 |
| FixCyclGorner | 13 |
| FixNoCyclGorner | 6 |
| acosTable | 3 |

Таб. 5 Итоговая таблица замеров времени

В ней можно отметить следующие закономерности: чем более качественные оптимизации мы используем, тем быстрее работает функция. Таблично-алгоритмическая реализация выигрывает по скорости. Тем не менее она проигрывает по памяти, так как требует массива предварительно подсчитанных результатов. Также стоит отметить, что реализации, с фиксированной точкой обходят соответствующие реализации с плавающей точкой.