МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**“УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”**

Факультет информационных систем и технологий

Кафедра “Вычислительная техника”

Дисциплина “Высокопроизводительные вычисления”

**Лабораторная работа №2**

Исследование алгоритмических и программных методов ускорения реализации функции вещественных переменных

Вариант 3

Выполнил:

студент гр. ИВТАПбд-31

Вершинин Д. В.

Проверил:

Негода В.Н.

Ульяновск, 2019

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.

Функция, заданная вариантом ***cos(x)***, Погрешность вычислений не должна превышать

**1. Анализ разложения в ряды функций, фигурирующих в варианте задания.**

Разложение косинуса в степенной ряд:

Область сходимости:

**2. Разработка процедуры-функции контроля**

Чтобы найти длину ряда, нужно написать функцию, которая будет сравнивать результат нашей функции с эталоном – *cos(x).* По условиям задания диапазон аргумента x принадлежит промежутку [0, 1). Пройдемся по данному промежутку с шагом равным max\_delta = 2-22 и в каждой точке будем проверять хватает ли нам итераций. Погрешность вычислений не должна превышать:

Фрагмент кода определяющий достаточное количество членов ряда:

float max\_delta = (float)pow(2, -22);

int test\_n = 0;

//циклический проход по диапазону аргумента х

for (double i = 0.0; i < 1; i += max\_delta) {

while (!flverify(i, fl\_cycle\_nogorner)) {

test\_n++;

}

}

// функция для сравнения с эталоном

int flverify(float fl, PFLOAT p) {

float etalon = cos(fl);

float test\_cos = p(fl);

float delta = fabs(etalon - test\_cos);

if (delta < max\_delta)

return 1;

return 0;

}

//вычисление функции без оптимизаций

float fl\_cycle\_nogorner(float x) {

float result = 0;

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

result += pow(-1, i) \* (pow(x, 2 \* i) / (factorials[2 \* i]));

}

return result;

}

В результате проверки переменная test\_n принимает значение 5, следовательно, можно сделать вывод о том, что необходимая точность вычислений достигается при 5 членах ряда.

**3. Исследование времени вычисления для данных типа float**

Функция FlMath() вызывает функцию ***cos()*** из библиотеки ***math.h***.

float FlMath(float x) {

float res = cos(x);

return res;

}

Наивные реализации вычисления функции (без использования схемы Горнера).

Функция fl\_cycle\_nogorner() содержит наивную реализацию циклического вычисления функции без использования схемы Горнера.

float fl\_cycle\_nogorner(float x) {

float result = 0;

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

result += pow(-1, i) \* (pow(x, 2 \* i) / (factorials[2 \* i]));

}

return result;

}

Функция fl\_nocycle\_nogorner содержит наивную бесцикловую реализацию вычисления функции без использования схемы Горнера:

float fl\_nocycle\_nogorner(float x) {

return pow(-1, 0) \* (pow(x, 2 \* 0) / (factorials[2 \* 0])) +

pow(-1, 1) \* (pow(x, 2 \* 1) / (factorials[2 \* 1])) +

pow(-1, 2) \* (pow(x, 2 \* 2) / (factorials[2 \* 2])) +

pow(-1, 3) \* (pow(x, 2 \* 3) / (factorials[2 \* 3])) +

pow(-1, 4) \* (pow(x, 2 \* 4) / (factorials[2 \* 4])) +

pow(-1, 5) \* (pow(x, 2 \* 5) / (factorials[2 \* 5]));

}

Для реализации вычисления функции с использованием схемы Горнера выполним преобразование ряда по схеме Горнера для 5 членов ряда:

Функция fl\_nocycle\_gorner содержит бесцикловую реализацию вычисления функции с использованием схемы Горнера. В качестве коэффициентов *a* используются численные константы.

float fl\_nocycle\_gorner(float x) {

float xPow = x \* x;

return 1 -

xPow \* (((1 / (factorials[2])) -

xPow \* ((1 / (factorials[4])) -

xPow \* ((1 / (factorials[6])) -

xPow \* ((1 / (factorials[8])) -

xPow \* ((1 / (factorials[10]))))))));

}

Функция fl\_cycle\_gorner содержит циклическую реализацию вычисления функции с использованием схемы Горнера.

float fl\_cycle\_gorner(float x) {

float res = 0;

res = (double)1.0 / factorials[8] - x\*x / factorials[10];

for (int i = 3; i >= 0; i--) {

res \*= x\*x;

res = (double)1.0 / (factorials[2 \* i]) - res;

}

return res;

}

Проверим точность вычисления для данных функций с помощью ранее написанной функции test().

cout << "Checking " << test(fl\_cycle\_gorner);

cout << test(fl\_cycle\_nogorner);

cout << test(fl\_nocycle\_gorner);

cout << test(fl\_nocycle\_nogorner);



Для каждой функции проверка вернула значение 1, следовательно каждая функция работает корректно и возвращает ответ с необходимой точностью.

Функция для измерения затрат времени. Замеры проводятся с помощью функции clock(), выполняется 1 000 000 вычислений функции при одинаковом значении аргумента *х*:

int time\_fl(int count, PFLOAT p) {

int time;

clock\_t time\_start = clock();

for (int i = 0; i < count; i++)

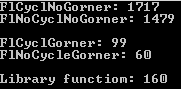
p(1);

time = (clock() - time\_start);

return time;

}

Сравнение времени работы функций, работающих с типом float:



|  |  |
| --- | --- |
| FlCyclNoGorner | 1717 |
| FlNoCyclNoGorner | 1479 |
| FlCyclGorner | 99 |
| FlNoCyclGorner | 60 |
| Library function | 160 |

Можно заметить, что бесцикловые реализации функции работают быстрее аналогичных циклических реализаций. А при разложении по схеме Горнера функции работают быстрее стандартной функции из библиотеки <math.h>

**4. Разработка макросов обработки чисел с фиксированной точкой.**

Определение типа с фиксированной точкой и макросы для перевода чисел из типа с плавающей запятой в фиксированную и наоборот.

typedef uint32\_t PFIX;

#define float\_to\_fix(a) ((PFIX)((a) \* (1LL<<28)))

#define fix\_to\_float(a) ((a) / (double)(1LL<<28))

Макросы умножения и деления чисел с фиксированной запятой:

#define fix\_mul(a, b) ((PFIX)(((int64\_t)(a) \* (b)) >> 28))

#define fix\_div(a, b) ((PFIX)(((int64\_t)(a) << 28) / (b)))

Указатель на функцию с фиксированной запятой:

typedef PFIX(\*FUNC\_FIX)(PFIX);

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

Для начала напишем функцию верификации для работы с числами с фиксированной запятой. Для этого адаптируем ранее написанную функцию flverify() для работы с числами с фиксированной запятой.

int fixverify(float x, FUNC\_FIX p) {

float etalon = cos(x);

PFIX test\_cos = p(float\_to\_fix(x));

float test\_cos\_fl = fix\_to\_float(test\_cos);

float delta = fabs(etalon - test\_cos\_fl);

if (delta < max\_delta)

return 1;

return 0;

}

Теперь напишем тестирующую функцию, которая будет запускать верификацию для чисел из промежутка значений аргумента *х*.

bool test\_fix(FUNC\_FIX p) {

for (float i = max\_delta; i < 1; i += max\_delta) {

if (fixverify(i, p) == 0)

return false;

}

return true;

}

Теперь адаптируем ранее написанные реализации вычисления функции на основе схемы Горнера с циклом и бесцикловой реализации схемы Горнера для работы с числами с фиксированной запятой.

PFIX fix\_nocycle\_gorner(PFIX x) {

PFIX double\_x = fix\_mul(x, x);

PFIX res = d -

fix\_mul(double\_x, koef[2] -

fix\_mul(double\_x, koef[4] -

fix\_mul(double\_x, koef[6] -

fix\_mul(double\_x, koef[8] -

fix\_mul(double\_x, koef[10])))));

return res;

}

PFIX fix\_cycle\_gorner(PFIX x) {

PFIX pow = fix\_mul(x, x);

float flpow = fix\_to\_float(pow);

PFIX temp = float\_to\_fix(flpow / factorials[10]);

PFIX res = koef[8] - temp;

for (int i = 3; i >= 0; i--) {

res = fix\_mul(pow, res);

res = koef[2 \* i] - res;

}

return res;

}

Проверим написанные функции на корректность.

cout << test\_fix(fix\_nocycle\_gorner);

cout << test\_fix(fix\_cycle\_gorner) << endl;

Для данных функций возвращается значение 1, следовательно, функции корректны и возвращают ответ с необходимой точностью.

Функция для измерения затрат времени. Замеры проводятся с помощью функции clock(), выполняется 1 000 000 вычислений функции при одинаковом значении аргумента *х*:

int time\_fix(int count, FUNC\_FIX p) {

int time;

clock\_t time\_start = clock();

for (int i = 0; i < count; i++) {

p(1);

}

time = (clock() - time\_start);

return time;

}

Сравнение времени работы функций, работающих с числами с фиксированной запятой:



|  |  |
| --- | --- |
| FixNoCycleGorner | 41 |
| FixCycleGorner | 62 |
| Library function | 140 |

Можно увидеть, что бесцикловая реализация вновь работает быстрее циклической, они также работают быстрее библиотечной реализации и методов, работающих с числами с плавающей запятой.

**6. Исследование таблично-алгоритмических реализаций функций.**

Перед реализацией таблично-алгоритмического метода определим оптимальные размеры таблицы.

int test\_size;

void defineSize() {

float delta = 1.0 / test\_size;

cout << test\_size << " " << (fix\_nocycle\_gorner(float\_to\_fix(0.99)) < max\_delta ? 1 : 0)

<< " " << (abs(-sin(0.99)) \* delta < max\_delta ? 1 : 0)

<< " " << (abs(-cos(0.99)) \* delta \* delta < max\_delta ? 1 : 0);

cout << "\n";

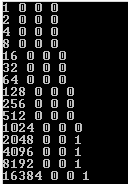
}

for (int i = 0; i < 15; i++) {

test\_size = pow(2, i);

defineSize();

}



При размерности таблицы в 2048 элементов функция представляется рядом a[0] + a[1]\*x

Разработка таблично-алгоритмической реализации.

int szTable = 2048;

float table[2050][2], val[2050];

void genTable() {

float delta = 1.0 / szTable;

float x = 0;

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

val[i] = x;

table[i][0] = fix\_to\_float(fix\_nocycle\_gorner(float\_to\_fix(x)));

table[i][1] = -sin(x);

}

}

float calcTable(float x) {

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

float h = x - val[ind];

float res = table[ind][0] + table[ind][1] \* h;

return res;

}

Для сравнения напишем таблично-алгоритмическую реализацию, которая будет работать с таблицей размером в 256 ячеек. Длина вычисляемого ряда соответственно увеличится до a[0] + a[1] \* x + a[2] \* x \* x. Ожидается, что он будет работать медленнее таблицы на 2048 значений.

int szTable2 = 256;

float table2[260][3], val2[260];

void genTable2() {

float delta = 1.0 / szTable2;

float x = 0;

for (int i = 0; i < szTable2; ++i, x += delta) {

val2[i] = x;

table2[i][0] = fix\_to\_float(fix\_nocycle\_gorner(float\_to\_fix(x)));

table2[i][1] = -sin(x);

table2[i][2] = -cos(x);

}

}

float calcTable2(float x) {

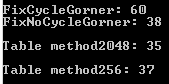
int ind = (szTable2 - 1) \* x;

float h = x - val2[ind];

return table2[ind][0] + h \* (table2[ind][1] + table2[ind][2] \* h);

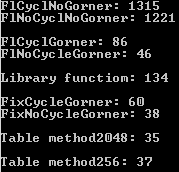
}

Сравним время вычисления функции для двух табличных методов и методов с фиксированной запятой.



Видно, что табличная реализация для таблицы на 2048 ячеек немного опережает методы с фиксированной запятой и вторую таблицу из-за того, что вычисления сокращаются до выражения a0 + a1\*x.

**7. Формирование итоговых результатов**



|  |  |
| --- | --- |
| FlCyclNoGorner | 1315 |
| FlNoCyclNoGorner | 1221 |
| FlCyclGorner | 86 |
| FlNoCyclGorner | 46 |
| Library function | 134 |
| FixNoCycleGorner | 60 |
| FixCycleGorner | 38 |
| Table method2048 | 35 |
| Table method256 | 37 |

Из таблицы видно, что бесцикловые реализации работают быстрее циклических, также заметно, что использование схемы Горнера на порядок повышает скорость работы алгоритма и они начинают работать быстрее стандартной библиотечной реализации. Еще одна оптимизация, повышающая прирост скорости – использование чисел с фиксированной запятой, вместо стандартного типа float. Самую высокую скорость показывает таблично алгоритмическая реализация алгоритма для таблицы размером 2048 ячеек.

**Исходный код**

#include <stdio.h>

#include <math.h>

#include <conio.h>

#include <time.h>

#include <iostream>

using namespace std;

typedef float(\*PFLOAT)(float);

typedef uint32\_t PFIX;

float max\_delta = (float)pow(2, -22);

int test\_n = 0;

int n = 5;

float factorials[100];

PFIX fix\_factorials[100];

PFIX koef[100];

#define float\_to\_fix(a) ((PFIX)((a) \* (1LL<<28)))

#define fix\_to\_float(a) ((a) / (double)(1LL<<28))

#define fix\_mul(a, b) ((PFIX)(((int64\_t)(a) \* (b)) >> 28))

#define fix\_div(a, b) ((PFIX)(((int64\_t)(a) << 28) / (b)))

PFIX d = float\_to\_fix(1);

typedef PFIX(\*FUNC\_FIX)(PFIX); // Указатель на функцию фикс. точка

double factorial(int x) {

if (factorials[x] == 0) {

factorials[x] = (x == 0 ? 1 : factorial(x - 1) \* x);

}

return factorials[x];

}

int time\_fl(int count, PFLOAT p) {

int time;

clock\_t time\_start = clock();

for (int i = 0; i < count; i++)

p(0.999);

time = (clock() - time\_start);

return time;

}

float fl\_cycle\_nogorner(float x) {

float result = 0;

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

result += pow(-1, i) \* (pow(x, 2 \* i) / (factorials[2 \* i]));

}

return result;

}

float fl\_cycle\_gorner(float x) {

float res = 0;

res = (double)1.0 / factorials[8] - x\*x / factorials[10];

for (int i = 3; i >= 0; i--) {

res \*= x\*x;

res = (double)1.0 / (factorials[2 \* i]) - res;

}

return res;

}

PFIX fix\_cycle\_gorner(PFIX x) {

PFIX pow = fix\_mul(x, x);

float flpow = fix\_to\_float(pow);

PFIX temp = float\_to\_fix(flpow / factorials[10]);

PFIX res = koef[8] - temp;

for (int i = 3; i >= 0; i--) {

res = fix\_mul(pow, res);

res = koef[2 \* i] - res;

}

return res;

}

int flverify(float fl, PFLOAT p) {

float etalon = cos(fl);

float test\_cos = p(fl);

float delta = fabs(etalon - test\_cos);

if (delta < max\_delta)

return 1;

return 0;

}

bool test(PFLOAT p) {

for (float i = max\_delta; i < 1; i += max\_delta) {

if (flverify(i, p) == 0)

return false;

}

return true;

}

float FlMath(float x) {

float res = cos(x);

return res;

}

int fixverify(float x, FUNC\_FIX p) {

float etalon = cos(x);

PFIX test\_cos = p(float\_to\_fix(x));

float test\_cos\_fl = fix\_to\_float(test\_cos);

float delta = fabs(etalon - test\_cos\_fl);

if (delta < max\_delta)

return 1;

return 0;

}

bool test\_fix(FUNC\_FIX p) {

for (float i = max\_delta; i < 1; i += max\_delta) {

if (fixverify(i, p) == 0)

return false;

}

return true;

}

float fl\_nocycle\_gorner(float x) {

float xPow = x \* x;

return 1 -

xPow \* (((1 / (factorials[2])) -

xPow \* ((1 / (factorials[4])) -

xPow \* ((1 / (factorials[6])) -

xPow \* ((1 / (factorials[8])) -

xPow \* ((1 / (factorials[10]))))))));

}

PFIX fix\_nocycle\_gorner(PFIX x) {

PFIX double\_x = fix\_mul(x, x);

PFIX res = d -

fix\_mul(double\_x, koef[2] -

fix\_mul(double\_x, koef[4] -

fix\_mul(double\_x, koef[6] -

fix\_mul(double\_x, koef[8] -

fix\_mul(double\_x, koef[10])))));

return res;

}

float fl\_nocycle\_nogorner(float x) {

return pow(-1, 0) \* (pow(x, 2 \* 0) / (factorials[2 \* 0])) +

pow(-1, 1) \* (pow(x, 2 \* 1) / (factorials[2 \* 1])) +

pow(-1, 2) \* (pow(x, 2 \* 2) / (factorials[2 \* 2])) +

pow(-1, 3) \* (pow(x, 2 \* 3) / (factorials[2 \* 3])) +

pow(-1, 4) \* (pow(x, 2 \* 4) / (factorials[2 \* 4])) +

pow(-1, 5) \* (pow(x, 2 \* 5) / (factorials[2 \* 5]));

}

int time\_fix(int count, FUNC\_FIX p) {

int time;

clock\_t time\_start = clock();

for (int i = 0; i < count; i++) {

p(1);

}

time = (clock() - time\_start);

return time;

}

int szTable = 2048;

float table[2050][2], val[2050];

int test\_size;

void defineSize() {

float delta = 1.0 / test\_size;

cout << test\_size << " " << (fix\_nocycle\_gorner(float\_to\_fix(0.99)) < max\_delta ? 1 : 0)

<< " " << (abs(-sin(0.99)) \* delta < max\_delta ? 1 : 0)

<< " " << (abs(-cos(0.99)) \* delta \* delta < max\_delta ? 1 : 0);

cout << "\n";

}

void genTable() {

float delta = 1.0 / szTable;

float x = 0;

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

val[i] = x;

table[i][0] = fix\_to\_float(fix\_nocycle\_gorner(float\_to\_fix(x)));

table[i][1] = -sin(x);

}

}

float calcTable(float x) {

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

float h = x - val[ind];

float res = table[ind][0] + table[ind][1] \* h;

return res;

}

int main()

{

for (int i = 0; i < 11; i++)

factorial(i);

for (int i = 0; i < 11; i++) {

koef[i] = float\_to\_fix(1.0 / factorials[i]);

}

for (int i = 0; i < 15; i++) {

test\_size = pow(2, i);

defineSize();

}

cout << "Checking " << test(fl\_cycle\_gorner);

cout << test(fl\_cycle\_nogorner);

cout << test(fl\_nocycle\_gorner);

cout << test(fl\_nocycle\_nogorner);

cout << test\_fix(fix\_nocycle\_gorner);

cout << test\_fix(fix\_cycle\_gorner) << endl;

printf("FlCyclNoGorner: %d\n", time\_fl(1000000, fl\_cycle\_nogorner));

printf("FlNoCyclNoGorner: %d\n", time\_fl(1000000, fl\_nocycle\_nogorner));

printf("\nFlCyclGorner: %d\n", time\_fl(1000000, fl\_cycle\_gorner));

printf("FlNoCycleGorner: %d\n", time\_fl(1000000, fl\_nocycle\_gorner));

printf("\nLibrary functiom: %d\n", time\_fl(1000000, FlMath));

printf("\nFixCycleGorner: %d\n", time\_fix(1000000, fix\_cycle\_gorner));

printf("FixNoCycleGorner: %d\n", time\_fix(1000000, fix\_nocycle\_gorner));

genTable();

printf("\nTable method: %d\n", time\_fl(1000000, calcTable));

\_getch();

return 0;

}