Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»

Факультет: «Информационные технологии и прикладная математика»

Курсовая работа

Дисциплина: «Вычислительная системы»

I семестр

Задание 3: «Вещественный тип. Приближенные вычисления. Табулирование функций»

|  |  |
| --- | --- |
| Группа: | М8О-108Б-18, №12 |
| Студент: | Коростелев Дмитрий Васильевич |
| Преподаватель: | Поповкин Александр Викторович |
| Оценка: |  |
| Дата: | 06.12.2018 |

Москва, 2018

**Содержание**

1. Задание.............................................................................................................2
2. Решение............................................................................................................2
3. Алгоритм..........................................................................................................3
4. Пояснение к алгоритму...................................................................................3
5. Код программы................................................................................................4
6. Результат работы программы.........................................................................5
7. Заключение.......................................................................................................8
8. Список использованной литературы.............................................................9
9. **Задание**

Составить программу на Си, которая печатает таблицу значений элементарной функции, вычисленной двумя способами: по формуле Тейлора и с помощью встроенных функций языка программирования. В качестве аргументов таблицы взять точки разбиения отрезка [a, b] на n равных частей (n + 1 точка, включая концы отрезка), находящихся в рекомендованной области хорошей точности формулы Тейлора. Вычисления по формуле Тейлора проводить по экономной в сложностном смысле схеме с точностью ε×k, где ε — машинное эпсилон аппаратно реализованного вещественного типа для данной ЭВМ, а k — экспериментально подбираемый коэффициент, обеспечивающий приемлемую сходимость. Число итераций должно ограничиваться сверху числом порядка 100. Программа должна сама определять машинное ε и обеспечивать корректные размеры генерируемой таблицы.

Вариант 12

|  |
| --- |
| Функция:  Отрезок: [0,0 ; 1,0]  Ряд: |

1. **Решение**

Суть задания состоит в том, что нужно реализовать вычисление заданной функции двумя способами на языке Си: разложением в ряд по формуле Тейлора и с помощью встроенных команд языка Си.

Вычисление функции с помощью формулы Тейлора происходит итерационно, на каждой итерации вычисляется значение шага, которое в последствии прибавляется к общей сумме. При решении данного задания стоит также задуматься о точности вычислений, так как некоторые результаты будут представлять из себя иррациональный числа, а как мы знаем бесконечные иррациональные числа нельзя представить в виде одной переменной. Отсюда следует вывод, что при решении задания следует найти специальную, минимально возможную дробную переменную - «машинное эпсилон». Машинное эпсилон – это минимальная разница между числами, которую компьютер в состоянии различить, то есть соблюдается условие 1.0 + ℇ ≠ 1.0 , при этом любое число меньшее ℇ будет восприниматься компьютером как 0, а вследствие будет выполняться выше указанное равенство.

1. **Алгоритм**
2. Считать число, на которое будет разбит заданный интервал.
3. Считать коэффицинет на который будет умножено машинное эпсилон для задания интервала допустимой погрешности.
4. Посчитать «машинное эпсилон»
5. Итерационно считать значение заданной функции с помощью встроенных команд и библиотек языка Си и с помощью ряда Тейлора.
6. Вывести полученные значения в виде таблицы.
7. **Пояснение к алгоритму.**
8. Подсчет «машинного эпсилона» будем производить, деля изначальное значение «машинного эпсилона» равное 1 на 2 до того момента, пока не выполнится равенство 1.0 + ℇ = 1.0.
9. Для подсчет значения функции с помощью встроенных библиотек будем использовать функцию pow(double x, double y), которая принимает две вещественные переменные x и y, и возводит x в степень y, функция реализована в библиотеке math.h
10. Поиск значения по Тейлору представляет из себя цикл while, условием выполнения которого является разность между значением функции, вычисленной ранее и значением по Тейлору, полученным на данной прошлой итерации, вычисления продолжаются до того момента пока разность этих значений по модулю будет меньше допустимой погрешности ℇ\*k.
11. В процессе работы алгоритма используются также функции fabs, log и fact. Функции fabs и log описаны в библиотеке math.h и возвращают значение аргумента по модуля и значение натурального логарифма соответственно.
12. **Код программы**

dmitry@dmitry-VirtualBox:~/lubs$ cat 3kp.c

#include<stdio.h>

#include<math.h>

double Epsilon() {

double epsilon = 1.0;

while ( 1.0 + epsilon / 2.0 > 1.0) {

epsilon /= 2.0;

}

return epsilon;

}

long long fact(int count) {

long long fct = 1;

for (int i = 1; i <= count; i++)

fct = fct \* i;

return fct;

}

int main(void) {

double epsilon = Epsilon();

printf("Enter the number of split points for the segment: ");

int n;

scanf("%d", &n);

printf("Enter the error factor k: ");

int k;

scanf("%d", &k);

printf("\nEpsilon = %0.16lf\n",epsilon);

printf("Step = %lf\n\n", 1.0 / n);

printf("PointX\t Taylor\t Function\t Iteration\t\n");

int iteration;

long double function, x, taylor;

for (int i = 0; i < n + 1; i++) {

iteration = 1;

taylor = 1;

x = i \* (1.0 / n);

function = pow(3, x);

//printf("%lf", function);

while ((fabs(function - taylor) >= (epsilon\*k)) && iteration < 100) {

//printf("%d", factorial(iteration));

unsigned long long f = fact(iteration);

taylor += (powl(log(3), iteration)/f) \* pow(x,iteration);

//printf("%lld\n", f);

iteration++;

}

printf("%.5Lf\t %.10Lf\t %.10Lf\t %d\t\n",x,taylor,function,iteration);

}

return 0;

}

1. **Результат работы программы**

dmitry@dmitry-VirtualBox:~/lubs$

dmitry@dmitry-VirtualBox:~/lubs$ ./3kp.out

Enter the number of split points for the segment: 100

Enter the error factor k: 10

Epsilon = 0.0000000000000002

Step = 0.010000

PointX Taylor Function Iteration

0.00000 1.0000000000 1.0000000000 1

0.01000 1.0110466919 1.0110466919 7

0.02000 1.0222154133 1.0222154133 7

0.03000 1.0335075120 1.0335075120 8

0.04000 1.0449243511 1.0449243511 8

0.05000 1.0564673085 1.0564673085 8

0.06000 1.0681377774 1.0681377774 9

0.07000 1.0799371664 1.0799371664 9

0.08000 1.0918668996 1.0918668996 9

0.09000 1.1039284169 1.1039284169 10

0.10000 1.1161231740 1.1161231740 10

0.11000 1.1284526429 1.1284526429 10

0.12000 1.1409183116 1.1409183116 10

0.13000 1.1535216847 1.1535216847 10

0.14000 1.1662642834 1.1662642834 10

0.15000 1.1791476457 1.1791476457 11

0.16000 1.1921733265 1.1921733265 11

0.17000 1.2053428979 1.2053428979 11

0.18000 1.2186579496 1.2186579496 11

0.19000 1.2321200886 1.2321200886 11

0.20000 1.2457309396 1.2457309396 11

0.21000 1.2594921455 1.2594921455 12

0.22000 1.2734053673 1.2734053673 12

0.23000 1.2874722841 1.2874722841 12

0.24000 1.3016945938 1.3016945938 12

0.25000 1.3160740130 1.3160740130 12

0.26000 1.3306122771 1.3306122771 12

0.27000 1.3453111411 1.3453111411 12

0.28000 1.3601723788 1.3601723788 12

0.29000 1.3751977840 1.3751977840 13

0.30000 1.3903891703 1.3903891703 13

0.31000 1.4057483712 1.4057483712 13

0.32000 1.4212772404 1.4212772404 13

0.33000 1.4369776522 1.4369776522 13

0.34000 1.4528515016 1.4528515016 13

0.35000 1.4689007046 1.4689007046 13

0.36000 1.4851271982 1.4851271982 13

0.37000 1.5015329408 1.5015329408 13

0.38000 1.5181199126 1.5181199126 13

0.39000 1.5348901156 1.5348901156 14

0.40000 1.5518455739 1.5518455739 14

0.41000 1.5689883339 1.5689883339 14

0.42000 1.5863204647 1.5863204647 14

0.43000 1.6038440582 1.6038440582 14

0.44000 1.6215612294 1.6215612294 14

0.45000 1.6394741168 1.6394741168 14

0.46000 1.6575848823 1.6575848823 14

0.47000 1.6758957118 1.6758957118 14

0.48000 1.6944088155 1.6944088155 14

0.49000 1.7131264277 1.7131264277 14

0.50000 1.7320508076 1.7320508076 15

0.51000 1.7511842393 1.7511842393 15

0.52000 1.7705290321 1.7705290321 15

0.53000 1.7900875209 1.7900875209 15

0.54000 1.8098620662 1.8098620662 15

0.55000 1.8298550549 1.8298550549 15

0.56000 1.8500689000 1.8500689000 15

0.57000 1.8705060412 1.8705060412 15

0.58000 1.8911689452 1.8911689452 15

0.59000 1.9120601060 1.9120601060 15

0.60000 1.9331820449 1.9331820449 15

0.61000 1.9545373114 1.9545373114 15

0.62000 1.9761284830 1.9761284830 16

0.63000 1.9979581656 1.9979581656 16

0.64000 2.0200289939 2.0200289939 16

0.65000 2.0423436319 2.0423436319 16

0.66000 2.0649047729 2.0649047729 16

0.67000 2.0877151398 2.0877151398 16

0.68000 2.1107774858 2.1107774858 16

0.69000 2.1340945944 2.1340945944 16

0.70000 2.1576692800 2.1576692800 16

0.71000 2.1815043878 2.1815043878 16

0.72000 2.2056027947 2.2056027947 16

0.73000 2.2299674094 2.2299674094 16

0.74000 2.2546011724 2.2546011724 16

0.75000 2.2795070570 2.2795070570 16

0.76000 2.3046880692 2.3046880692 17

0.77000 2.3301472483 2.3301472483 17

0.78000 2.3558876671 2.3558876671 17

0.79000 2.3819124324 2.3819124324 17

0.80000 2.4082246853 2.4082246853 17

0.81000 2.4348276015 2.4348276015 17

0.82000 2.4617243919 2.4617243919 17

0.83000 2.4889183029 2.4889183029 17

0.84000 2.5164126167 2.5164126167 17

0.85000 2.5442106516 2.5442106516 17

0.86000 2.5723157629 2.5723157629 17

0.87000 2.6007313427 2.6007313427 17

0.88000 2.6294608207 2.6294608207 17

0.89000 2.6585076643 2.6585076643 17

0.90000 2.6878753795 2.6878753795 17

0.91000 2.7175675108 2.7175675108 18

0.92000 2.7475876419 2.7475876419 18

0.93000 2.7779393962 2.7779393962 18

0.94000 2.8086264369 2.8086264369 18

0.95000 2.8396524679 2.8396524679 18

0.96000 2.8710212339 2.8710212339 18

0.97000 2.9027365211 2.9027365211 18

0.98000 2.9348021572 2.9348021572 18

0.99000 2.9672220125 2.9672220125 18

1.00000 3.0000000000 3.0000000000 18

1. **Заключение**

Таким образом, на практике удалось убедится в следующем: вычисления сложных функций, результатом которых являются вещественные числа, проходят с определенной точностью близкой к значению машинного эпсилона, стоит отметить, что значение машинного эпсилона на каждом компьютере будет разным, также значение эпсилона зависит от типа переменной, кроме того, можно убедится, в том, что вычисление функций по ряду Тейлора не является эффективным, так как, вычисления занимают много времени, а сложность написания алгоритма в несколько раз выше нежели использование встроенных функций.

1. **Список использованной литературы**
2. Брайан Керниган, Деннис Ритчи. Язык программирования Си. Второе издани: Серия книг по программированию от Prentice Hall. 2009. – 292с.:ил.
3. Машинный эпсилон [Электронный ресурс]// https://toster.ru/ URL: <https://toster.ru/q/16099>
4. Формула Тейлора. Ряд Тейлора [Электронный ресурс]// https://studfiles.net/ URL: https://studfiles.net/preview/3220166