Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет прикладной математики и информатики
Кафедра вычислительной математики и программирования

Курсовой проект
По дисциплине
«Вычислительные системы»
1 семестр

Задание 3:
Вещественный тип. Приближенные значения.
Табулирование функций.

Студент:	Тузова К.К.
Группа:	М8О-109Б-22
Преподаватель:	
Подпись:	
Оценка:	
Дата:	04.01.2022

Москва
2021

Содержание

1 Что нужно сделать?

Нужно было составить программу, которая вычисляет значения заданной в варианте функции в точках на интервале [a, b], при этом значения должны вычисляться двумя способами, а результаты работы программы должны выводиться в виде таблицы. Первый способ вычисления - подсчёт с помощью формулы Тейлора. Знаем что складывая элементы многочлена Тейлора, являющегося разложением функции, можно приблизиться к значению этой функции в заданной точке. В качестве второ способа для нахождения значения функции нужно использовать встроенные средства языка Си.

В каждом варианте, предлагаемом студентам, даны значения концов отрезка. Он разбивается на n равных частей, а значения функции считаются соответственно на + 1 точках из данного отрезка [a, b].

При вычислениях через формулу Тейлора нужно обеспечить точность $\varepsilon * k$, где машинное эпсилон, а k - некий коэффициент. Программа должна сама определять ε, этого и начнём.

2 Решение

2.1 Определение машинного ε

Эта задача общая для любых вариантов, для нахождения машинного ε обратимся к тому, что это такое. Итак, машинное ε – это некоторое очень маленькое число, являющееся минимальной разницей между двумя числами, которую может определить компьютер. Иными словами, если к числу a = 1 прибавить число b < ε , т компьютер посчитает, что 1 = 1 + b.

Чтобы рассчитать значение ε можно использовать следующую функцию:

```
double find_eps() {
   double e = 1.0;
   while (1.0+e>1.0)
      e /= 2.0;
   return e;
}
```

Тут всё просто, сначала инициализируем переменную типа double и присваиваем ей значение 1, внутри цикла while происходит уменьшение значения ε до тех пор, пока компьютер не сможет различать 1 и 1 + ε. В этом и во всех других случаях объявления переменных double, судя по всему, нет разницы между double *var* = a.0 и double *var* = a, по крайней мере компилятор gcc 10.2 преобразует оба варианта в один и тот же код.

2.2 Вариант 17

В этом варианте следующая функция $\frac{1}{2}\ln(x)$, подсчитать ее значения нужно на отрезке [0,2;0,7]. Ряд Тейлора для этой функции – $\frac{x-1}{x+1} + \frac{1}{3}(\frac{x-1}{x+1})^3 + \ldots + \frac{1}{2n+1}(\frac{x-1}{x+1})^{2n+1}$. С помощью встроенных средств языка Си значения этой функции вычисляются так:

```
double integrated_func(double x) {
   return (1.0/2.0)*log(x);
}
```

Подключив библиотеку math.h, можно создать функцию, куда передается значение точки x , в которой и вычисляется значение функции. Теперь подсчёт значения функции с помощью формулы Тейлора. Раскладывать ничего не надо, ряд уже дан в условии задачи.

```
double taylor(unsigned int n, double x) {
   double func_sum = 0;
   for(unsigned i = 0; i <= n; i++)
      func_sum +=(1.0/(2.0*i+1))*pow((x-1)/(x+1), 2*i + 1);
   return func_sum;
```

В принципе, тут тоже используются команды из math.h, но мы вычисляется не функцию на прямую, а сумму ряда Тейлора. В функцию передается количество членов ряд, которые суммируются, и точка *x*, в которой подсчитывается функция. Чем больше членов, тем, следовательно, сумма ряда ближе к значению, рассчитанному встроенными методами Си. В циклеfor складываются члены до тех пор, пока их количество не достигнет значения числа *n.*

```
struct Line step(unsigned iteration, double x, double eps, double (*taylor)(unsigned, double), double
(*integrated_func)(double)){
   unsigned c = 0;
   double taylor_var = taylor(c, x);
   while ((fabs(taylor_var - integrated_func(x)) > eps * 100) && c < 100){
      taylor_var = taylor(c, x);
      c++;
   }
   struct Line ans = {.iteration = iteration, .x = x, .integrated = integrated_func(x), .taylor = taylor_var};
   return ans;
}

void print_table_line(struct Line line) {
   printf("%.13f\t| %.13f\t| %.13f\t| %d\n", line.x, line.taylor, line.integrated, line.iteration);
}

void print_table(unsigned int n, double a, double b, double (*taylor)(unsigned int, double),
            double (*integrated_func)(double)) {
   double delta = (b - a) / n;
```

```
    double eps = find_eps();
    double x = a;
    printf("x%14s\t| taylor%9s\t| integrated%5s\t| iteration\n", "", "", "");
    printf("______________________________________________________________________________\n");
    for(unsigned i = 0; i <= n; i++, x += delta){
        print_table_line(step(i, x, eps, taylor, integrated_func));
    }
    printf("______________________________________________________________________________\n");
}
```

Начальная функция print_table, она принимает на ввод количество отрезков - n, отрезок [a, b], ссылки на функцию тейлора - taylor и интегрированную - integrated_func. Она начинает с расчёта “дельты” - шага между точками, зависящий от введённого n, на которое делится начальный отрезок данный в варианте. Потом функция ищет машинный ε с помощью вышеуказанной функции. Затем написан вывод “шапки” таблицы. После написан цикл, где находится основная рабочая часть программы, для удобства она вынесена в отдельную функцию step(её опишем чуть позже).

Функция print_table_line отвечает лишь за вывод результата работы функции step в виде строки таблицы, в ней стоит обратить внимание лишь на форматированный вывод, чтоб добится необходимой точности вывода, иначе будет выведено меньшее количество знаков после запятой.

Структура Line создана для удобства передачи и хранения данных из функции step т.к. они потенциально могут пригодиться в будущем, а так же чтоб вывесть вывод из функции с главной логикой и облегчить её читаемость.

Функция step принимает номер итерации - iteration(только сохраняется в структуре, в логике функции участие не принимает), точка для которой происходят вычисления - x, машинный ε - eps, ссылки на функцию тейлора - taylor и интегрированную - integrated_func. Именно тут проверяется сходимость ряда к значению функции, разница между ними должна быть не большееε *100, если разница слишком большая, увеличивается длина ряда тейлора - c, оно уже упоминалось виде n в функции taylor.

При этом по условию c ограничена сверху числом 100 => выходим из цикла.

В итоге получаем таблицу с точками x с шагом (b - a) / n, где n вводимое с клавиатуры число, так будет выглядеть main:

```
int main() {
    double a = 0.2, b = 0.7;
    unsigned n;
    scanf("%d", &n);
    print_table(n, a, b, taylor, integrated_func);
    return 0;
}
```

Концы отрезка закреплены вариантом, вводить с клавиатуры их не требуется, поэтому при запуске программы на стандартный ввод подается только количеств отрезков. Для этого варианта задания получаем на вывод ровную табличку, где-то при заданной точности вывода результаты двух способов равны, а где-то значение по Тейлору меньше настоящего, но точность всё же очень хорошая.

```
10
x              | taylor               | integrated           | iteration
-----------------------------------------------------------------------
0.2000000000000 | -0.8047189562170      | -0.8047189562171      | 0
0.2500000000000 | -0.6931471805599      | -0.6931471805599      | 1
0.3000000000000 | -0.6019864021630      | -0.6019864021630      | 2
0.3500000000000 | -0.5249110622493      | -0.5249110622493      | 3
0.4000000000000 | -0.4581453659371      | -0.4581453659371      | 4
0.4500000000000 | -0.3992538481089      | -0.3992538481089      | 5
0.5000000000000 | -0.3465735902800      | -0.3465735902800      | 6
0.5500000000000 | -0.2989185003778      | -0.2989185003778      | 7
0.6000000000000 | -0.2554128118830      | -0.2554128118830      | 8
0.6500000000000 | -0.2153914580462      | -0.2153914580462      | 9
0.7000000000000 | -0.1783374719694      | -0.1783374719694      | 10
-----------------------------------------------------------------------
```

Число 10 наверху таблицы – количество отрезков. Другими словами, функция в данном случае рассчитана для 11 точек.

3 Вывод

Я написала программу, вычисляющую значения в разных точках отрезка при помощи встроенных методов языка Си и с помощью формулы Тейлора. Точность вычисления через формулу Тейлора составляет 100* ε, где $\varepsilon \approx 10^{-16}$. Сложность программы составляет O(n), так как для каждой точки разбиения программа выполняет конечно число шагов, не превышающее определённое значение, которое зависит от n.