ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»  
(РУТ (МИИТ))

Институт транспортной техники и систем управления

Кафедра «Управление и защита информации»

ОТЧЁТ  
О ЛАБОРАТОРНОЙ РАБОТЕ №3

По дисциплине «Языки программирования»

ВАРИАНТ 6

Выполнил: ст. гр. ТКИ-141

Винтфельд Рина Дмитриевна

Проверил: к.т.н., доц. Васильева М. А.

(Проверил: к.т.н, доц. Балакина Е. П.)

Москва 2023

Оглавление

[1 Задача 3.1 3](#_Toc153834216)

[1.1 Формулировка задания 3.1 3](#_Toc153834217)

[1.2 Блок-схема алгоритма 3.1 4](#_Toc153834218)

[1.3 Текст программы на языке C 3.1 5](#_Toc153834219)

[1.4 Результаты выполнения программы 3.1 7](#_Toc153834220)

[1.5 Выполнение тестовых примеров 3.1 8](#_Toc153834221)

[1.6 Отметка о выполнении задания в веб-хостинге системы контроля версий 3.1 9](#_Toc153834222)

[2 Задача 3.2 10](#_Toc153834223)

[2.1 Формулировка задания 3.2 10](#_Toc153834224)

[2.2 Блок-схема алгоритма 3.2 11](#_Toc153834225)

[2.3 Текст программы на языке C 3.2 16](#_Toc153834226)

[2.4 Результаты выполнения программы 3.2 20](#_Toc153834227)

[2.5 Выполнение тестовых примеров 3.2 21](#_Toc153834228)

[2.6 Отметка о выполнении задания в веб-хостинге системы контроля версий 3.2 22](#_Toc153834229)

[3 Задача 3.3 23](#_Toc153834230)

[3.1 Формулировка задания 3.3 23](#_Toc153834231)

[3.2 Блок-схема алгоритма 3.3 24](#_Toc153834232)

[3.3 Текст программы на языке C 3.3 27](#_Toc153834233)

[3.4 Результаты выполнения программы 3.3 30](#_Toc153834234)

[3.5 Выполнение тестовых примеров 3.3 31](#_Toc153834235)

[3.6 Отметка о выполнении задания в веб-хостинге системы контроля версий 3.3 32](#_Toc153834236)

1. Задача 3.1
   1. Формулировка задания 3.1

Протабулировать заданную в таблице функцию. Использовать данные в таблице значения шага и интервала в качестве ввода пользователя для решения тестового примера. При невозможности расчёта функции в конкретной точке выводить её значение и надпись, означающую отсутствие решения.

Таблица 1 – Исходные данные

|  |  |  |
| --- | --- | --- |
| Вариант | Функция | Константы |
| 6 |  |  |

* 1. Блок-схема алгоритма 3.1

Блок-схема основного алгоритма представлена ниже (Рисунок 1). Блок-схемы используемых функций представлены ниже (Рисунок 2).

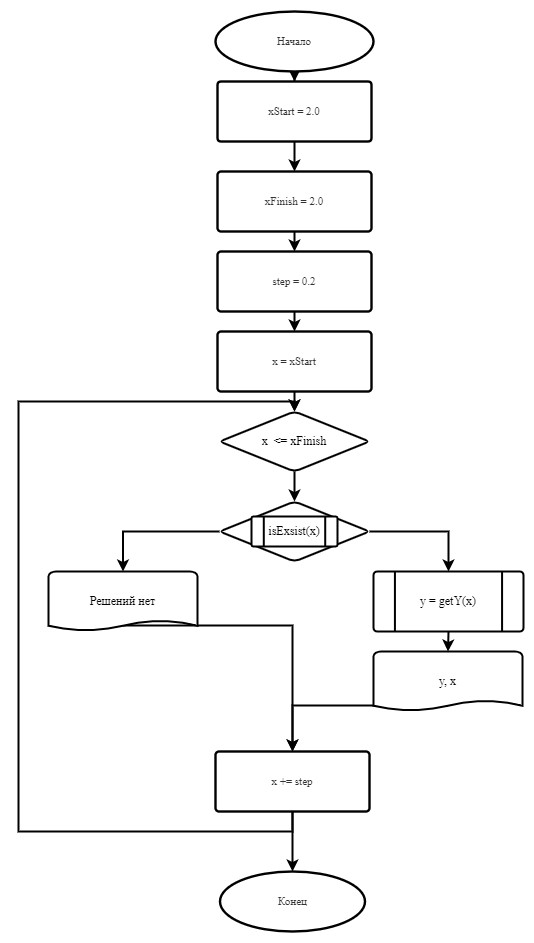


Рисунок 1 - Блок-схема основного алгоритма

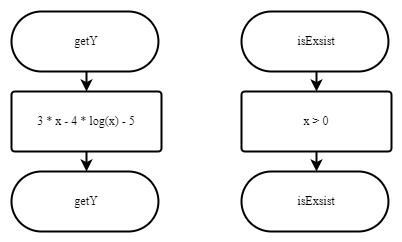


Рисунок 2 - Блок-схема используемой функций

* 1. Текст программы на языке C 3.1

#include <stdio.h>

#include <math.h>

#include <float.h>

#include <stdbool.h>

/\*

\* @brief Существует ли функция в точке x

\* @param x - Аргумент функции

\* @return true, если существует

\*/

bool isExist(const double x);

/\*

\* @brief Расчет функции в точке x

\* @param x - Аргумент функции

\* @return Значение функции в точке x

\*/

double getY(const double x);

/\*

\* @brief является точкой входа в программу

\* @return 0, в случае успеха

\*/

int main() {

double const xStart = 2.0;

double const xFinish = 4.0;

double const step = 0.2;

double x = xStart;

while (x - xFinish <= DBL\_MIN)

{

if (isExist(x))

{

const double y = getY(x);

printf\_s("y = %lf, при x = %lf\n", y, x);

}

else

{

printf\_s("Решений нет при x = %lf\n", x);

}

x += step;

}

return 0;

}

bool isExist(const double x) {

return x > DBL\_MIN;

}

double getY(const double x) {

return 3\*x - 4\*log(x) - 5;

}

* 1. Результаты выполнения программы 3.1

Результаты выполнения программы представлены ниже (Рисунок 3).

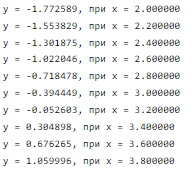


Рисунок 3 - Результат выполнения программы

* 1. Выполнение тестовых примеров 3.1

В программе Photomath выполнен тестовый пример. Результат его выполнения представлен ниже (Рисунок 4, Рисунок 5).



Рисунок 4 – Тестовый пример



Рисунок 5 - Тестовый пример

* 1. Отметка о выполнении задания в веб-хостинге системы контроля версий 3.1

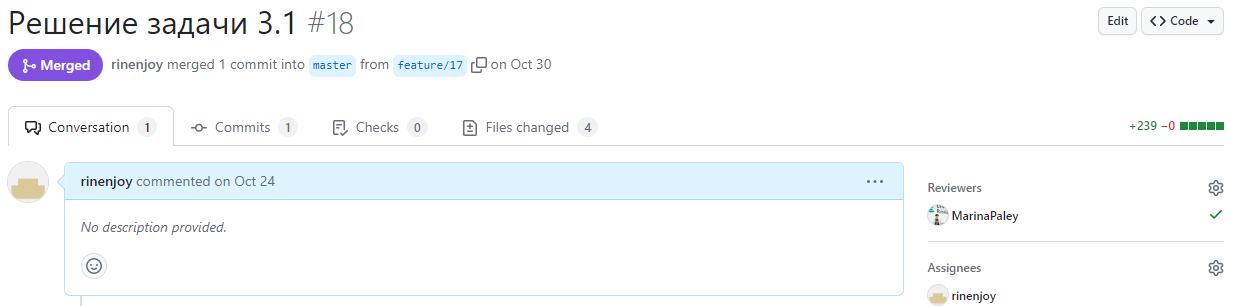


Рисунок 6 - Approved task 3.1

1. Задача 3.2
   1. Формулировка задания 3.2

Составьте две программы:

1. вычислить сумму первых n членов последовательности (k = 1, 2, 3 ..., n).
2. вычислить сумму всех членов последовательности, не меньших заданного числа e.

Помните о проверке пользовательского ввода. Все результаты вывести на экран. Отчёт дополнить блок-схемой. При вычислении факториалов рекомендуется отказаться от использования рекурсивных методов.

Таблица 2 – Исходные данные

|  |  |
| --- | --- |
| Вариант | Ряд |
| 6 |  |

* 1. Блок-схема алгоритма 3.2

Блок-схема основного алгоритма представлена ниже (Рисунок 8). Блок-схемы используемых функций представлены ниже (Рисунок 9, Рисунок 10, Рисунок 11, Рисунок 12). Расчётная формула(Рисунок 13).

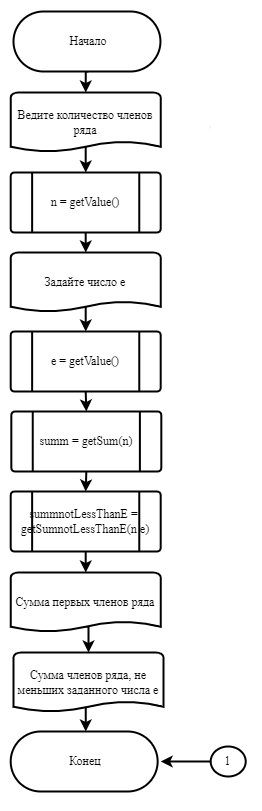


Рисунок 7 - Блок-схема основного алгоритма

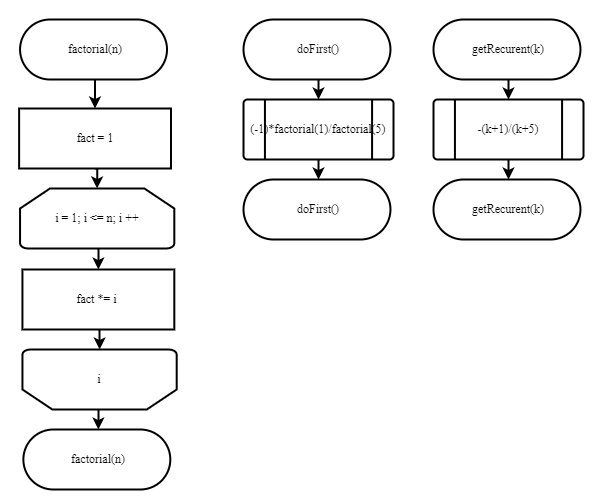


Рисунок 8 - Блок-схема используемых функций

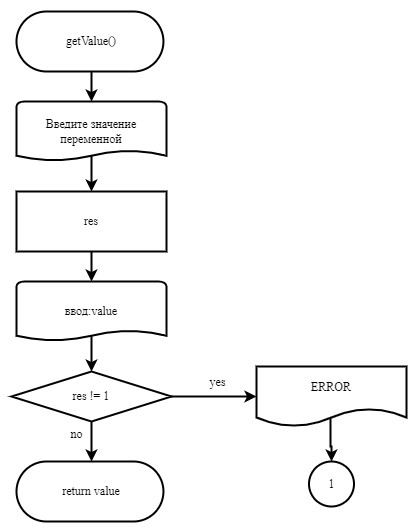


Рисунок 9 - Блок-схема используемой функции

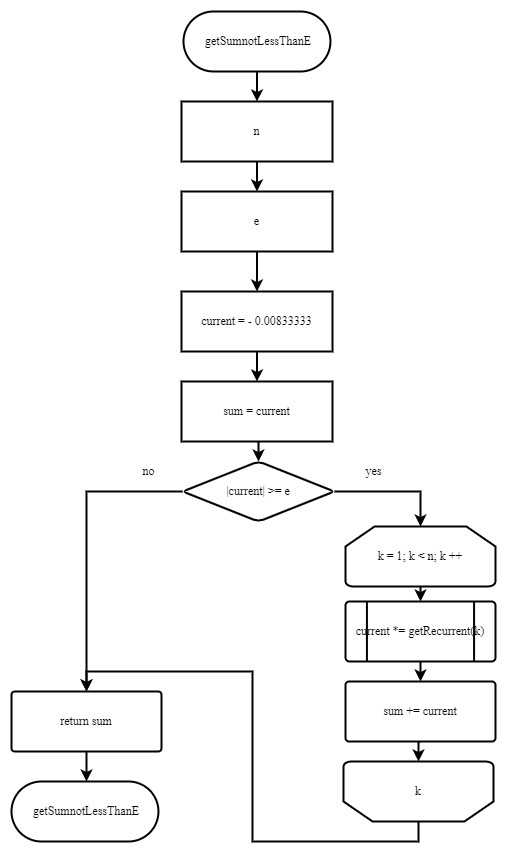


Рисунок 10 - Блок-схема используемой функции

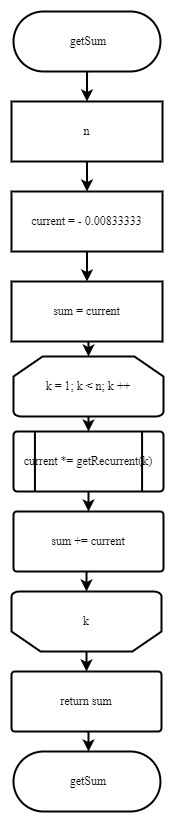


Рисунок 11 - Блок-схема используемой функции

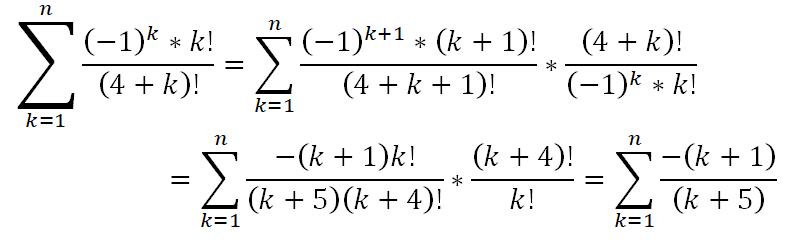


Рисунок 12 - Расчётная формула

* 1. Текст программы на языке C 3.2

#include <stdio.h>

#include <math.h>

#include <errno.h>

#include <stdlib.h>

#include <float.h>

/\*

\* @brief Функция считает факторал

\* @param n = число, от которого надо вычислить факториал

\* @return вычесленное значение

\*/

int factorial(int n);

/\*

\* @brief Функция считает первое значение ряда

\* @param k = 1 - первый член последовательности

\* @return вычесленное значение

\*/

double doFirst();

/\*

\* @brief проверяет пользовательский ввод

\* @return введеное значение

\*/

double getValue();

/\*

\* @brief Вычисляет текущий член ряда

\* @param k - Порядковый номер члена ряда

\* @return Значение текущего члена ряда

\*/

double getRecurrent(int k);

/\*

\* @brief Вычисляет сумму ряда до указанного номера члена

\* @param n - Номер члена ряда, до которого нужно вычислить сумму

\* @return Значение суммы ряда

\*/

double getSum(int n);

/\*

\* @brief Вычисляет сумму всех членов ряда, не меньших заданного числа e

\* @param n - Максимальное количество членов ряда для вычисления

\* @param e - Точность вычисления ряда последовательности

\* @return Значение суммы ряда

\*/

double getSumNotLessThanE(int n, double e);

/\*

\* @brief Основная функция программы

\* @return Возвращает 0 в случае успешного выполнения программы

\*/

int main() {

printf\_s("Введите количество членов ряда\n");

int n = getValue();

printf\_s("Задайте число e\n");

double e = getValue();

double summ = getSum(n);

double summNotLessThanE = getSumNotLessThanE(n, e);

printf\_s("Сумма первых членов ряда = %.15lf\n", summ);

printf\_s("Сумма всех членов последовательности, не меньших заданного числа e = %.15lf\n", summNotLessThanE);

return 0;

}

double getValue()

{

double value;

int res = scanf\_s("%lf", &value);

if (res != 1)

{

errno = EIO;

perror("ERROR");

abort();

}

return value;

}

int factorial(int n) {

int fact = 1;

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

{

fact \*= i;

}

return fact;

}

double doFirst() {

return (-1.0) \* factorial(1) / factorial(5);

}

double getRecurrent(int k)

{

return (double)(- 1.0 \* (k + 1) / (k + 5));

}

double getSum(int n)

{

double current = doFirst();

double sum = current;

for (int k = 1; k < n; k++)

{

current \*= getRecurrent(k);

sum += current;

}

return sum;

}

double getSumNotLessThanE(int n, double e)

{

double current = doFirst();

double sum = current;

int k = 1;

if (fabs(current) - e >= DBL\_MIN)

{

for (k; k <= n; k++)

{

current \*= getRecurrent(k);

sum += current;

}

}

return sum;

}

* 1. Результаты выполнения программы 3.2

Результат выполнения программы представлен ниже (Рисунок 14).

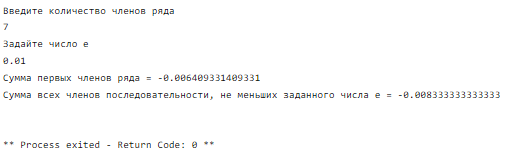


Рисунок 13 - Результат выполнения программы

* 1. Выполнение тестовых примеров 3.2

В приложении Excel выполнены тестовые примеры. Результаты их выполнения представлены ниже (Рисунок 15).

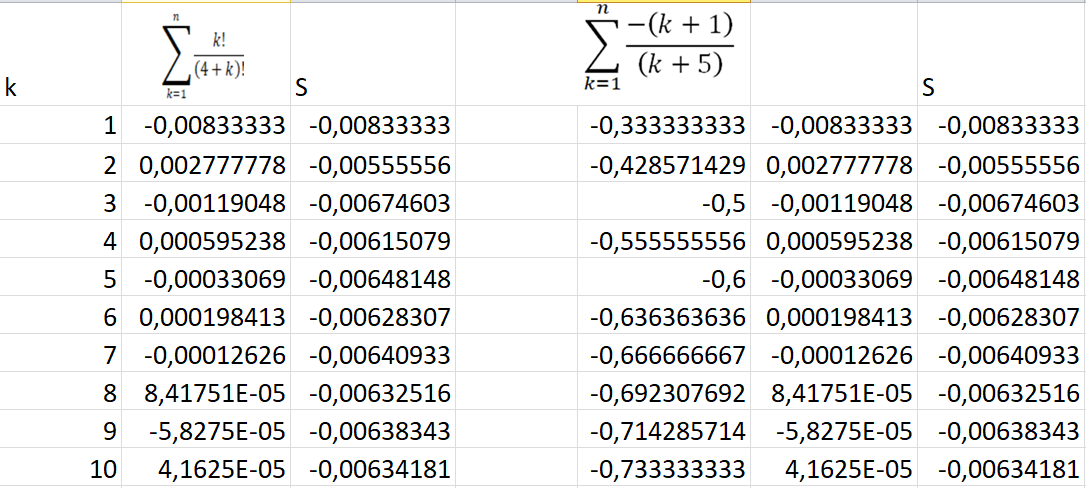


Рисунок 14 – Тестовый расчёт

* 1. Отметка о выполнении задания в веб-хостинге системы контроля версий 3.2

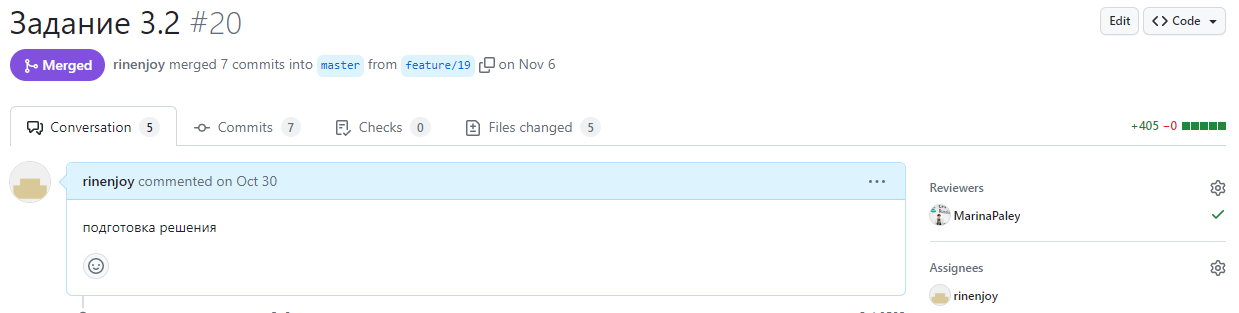


Рисунок 15 - Approved task 3.2

1. Задача 3.3
   1. Формулировка задания 3.3

Протабулировать заданную функцию и сумму функционального ряда разложения этой функции на интервале [*a,b*] и с шагом *h* (шаг и интервал задается в константах). Функциональнй ряд вычисляется по соответствующей рекуррентной формуле с заданной точностью *ɛ*. В результате показать три столбца: значение аргумента, значение функции в данной точке и значение суммы ряда, вычисленное с заданной точностью в данной точке. Два последних столбца должны иметь близкие результаты.

Таблица 3 - Исходные данные

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вар | Функция y | Сумма | Интервал | ɛ |
| 6 |  |  |  | 10-4 |

* 1. Блок-схема алгоритма 3.3

Блок-схема основного алгоритма представлена ниже (). Блок-схемы используемых функций представлены ниже ().

Блок-схема основного алгоритма

Блок-схема используемой функции

* 1. Текст программы на языке C 3.3

#include <stdio.h>

#include <errno.h>

#include <stdlib.h>

#include <float.h>

#include <math.h>

/\*

\* @brief Функция считает сумму ряда с заданной точностью

\* @param x - аргумент

\* @param func - посчитанная функция в данной точке

\* @param e - заданная точность

\* @return сумму

\*/

double get\_sum(double x, double func, double e);

/\*

\* @brief проверяет введенное значение

\* @param massage - сообщение для пользователя

\* @return переменную

\*/

double get\_value(const char\* massage);

/\*

\* @brief Заданная функция

\* @param x - аргумент функции

\* @return Значение функции в заданной точке

\*/

double function(double x);

/\*

\* @brief проверяет правильность ввода

\* @return введенное значение в случае успешной проверки, код ошибки в случае неправильного ввода

\*/

double get\_step();

/\*

\* @brief Точка входа в программу

\* @return 0, в случае успешного завершения программы

\*/

int main() {

double const a = 0.1;

double const b = 1.0;

//printf\_s("Enter step\n");

double step = 0.1;

//get\_step();

double const e = 0.001;

//get\_value("Введите е");

printf\_s("Argument\tFunction\tSumma\n");

for (double i = a; i - b <= DBL\_EPSILON; i += step)

{

double sum = get\_sum(i, function(i), e);

printf\_s("%.2lf\t\t%lf\t%lf\n", i, function(i), sum);

}

return 0;

}

double get\_step()

{

double step;

int res = scanf\_s("%lf", &step);

if (res != 1 || step - 1.0 > - DBL\_EPSILON)

{

errno = EIO;

perror ("ERROR");

abort();

}

return step;

}

double get\_sum(double x, double func, double const e)

{

double current = 1.0;

double sum = current;

double n = 1;

while (current - e > - DBL\_EPSILON )

{

current \*= (double)((pow(x,2)\*(2\*n + 3))/ ((n + 1) \* (2 \* n + 1)));

sum += current;

n = n + 1.0;

//printf\_s("\n%d\n%lf\n", n, current);

}

return sum;

}

double get\_value(const char\* massage)

{

printf\_s("\n%s\t", massage);

double value = 0.0;

int res = scanf\_s("%lf", &value);

if (res != 1)

{

errno = EIO;

perror("ERROR");

abort();

}

return value;

}

double function(double x) {

return ((1 + 2 \* pow(x, 2)) \* exp(pow(x, 2)));

}

* 1. Результаты выполнения программы 3.3

Результат выполнения программы представлены ниже (Рисунок 16)

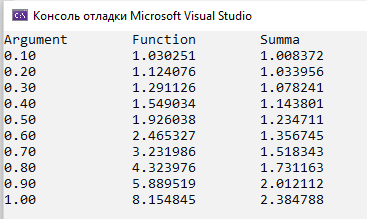


Рисунок - Результат выполнения программы

* 1. Выполнение тестовых примеров 3.3

На выполнены тестовые примеры. Результаты их выполнения представлены ниже (Рисунок 17, Рисунок 18, Рисунок 19, Рисунок 20, Рисунок 21, Рисунок 22, Рисунок 23, Рисунок 24, Рисунок 25, Рисунок 26).

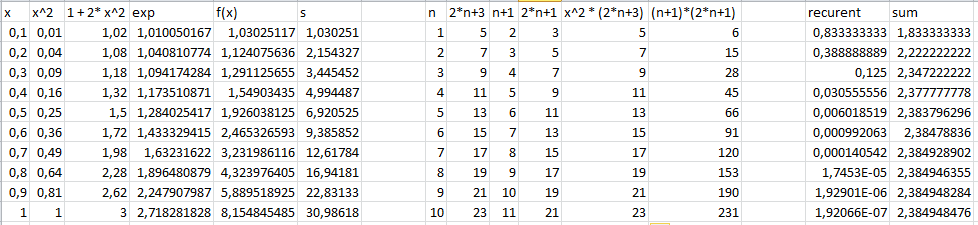


Рисунок – Тестовый расчёт при x = 1

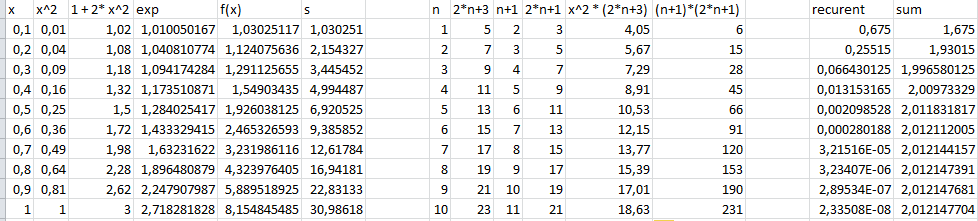


Рисунок - Тестовый расчёт при x = 0.9

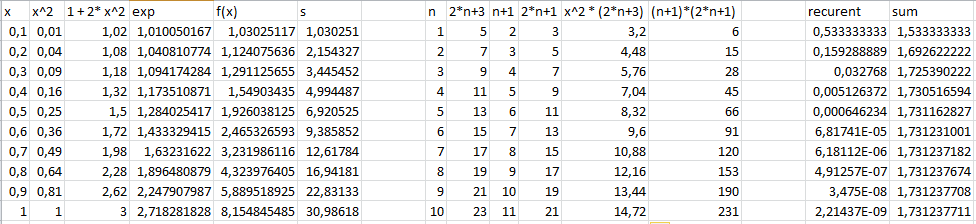


Рисунок - Тестовый расчёт при x = 0.8

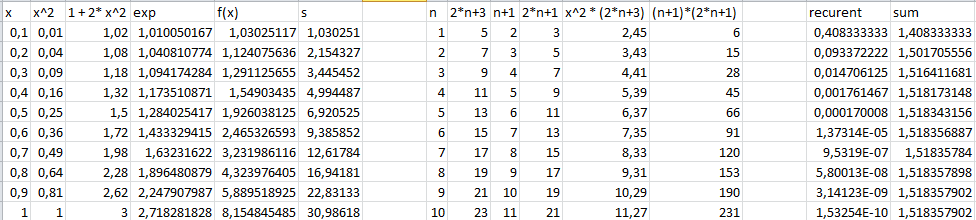


Рисунок - Тестовый расчёт при x = 0.7

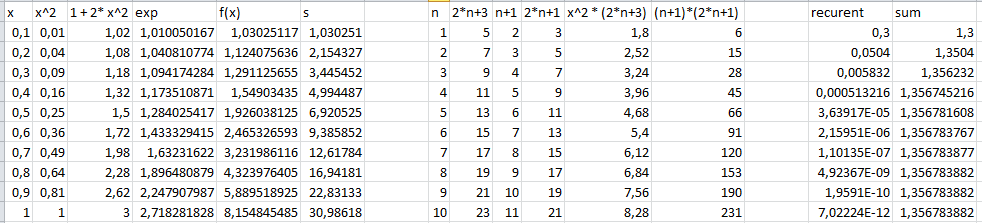


Рисунок - Тестовый расчёт при x = 0.6

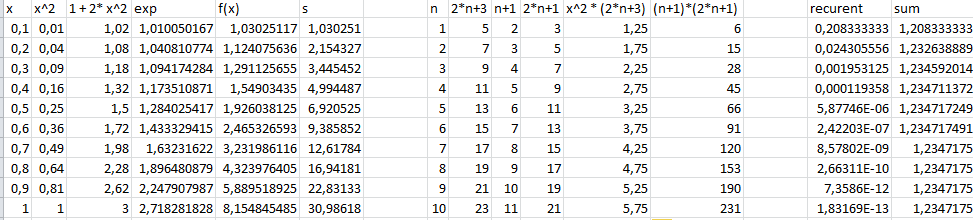


Рисунок - Тестовый расчёт при x = 0.5

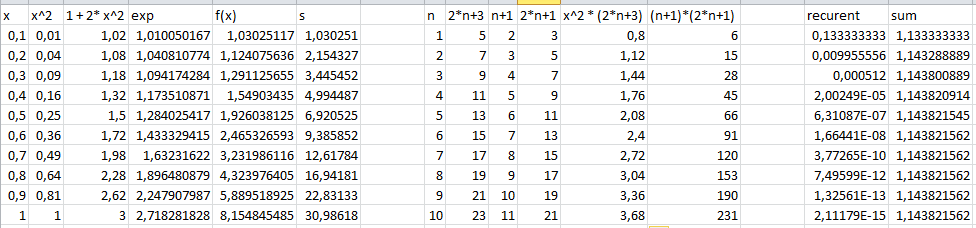


Рисунок - Тестовый расчёт при x = 0.4

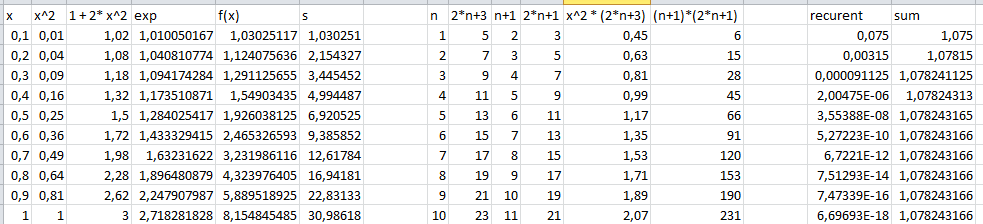


Рисунок - Тестовый расчёт при x = 0.3

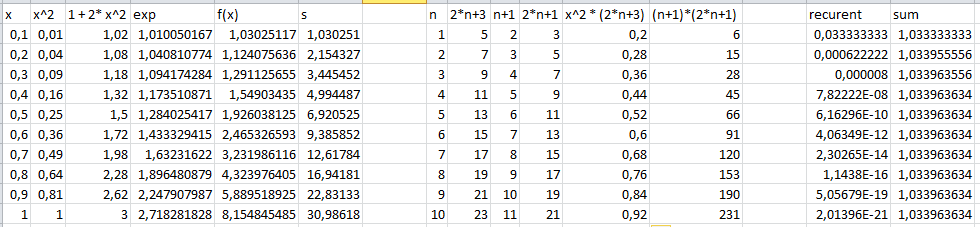


Рисунок - Тестовый расчёт при x = 0.2

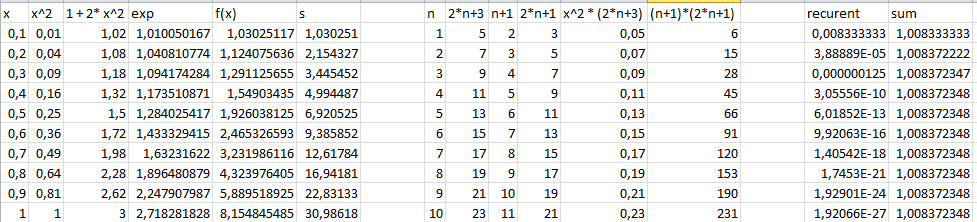


Рисунок - Тестовый расчёт при x = 0.1

* 1. Отметка о выполнении задания в веб-хостинге системы контроля версий 3.3

Approved task 3.3