|  |
| --- |
| Московский Государственный университет, факультет вычислительной математики и кибернетики |
| Вычисление корней уравнений и определённых интегралов |
|  |
|  |
| **Владимир Терёшин** |
| **21.02.2012** |

|  |
| --- |
|  |

Оглавление

[Постановка задачи 2](#_Toc317624633)

[Математическое обоснование 2](#_Toc317624634)

[Описание программы 4](#_Toc317624635)

[Тестирование 5](#_Toc317624636)

[Программа на языке СИ 5](#_Toc317624637)

[Полученные результаты 9](#_Toc317624638)

[Анализ ошибок 9](#_Toc317624639)

[Использованная литература 9](#_Toc317624640)

# Постановка задачи

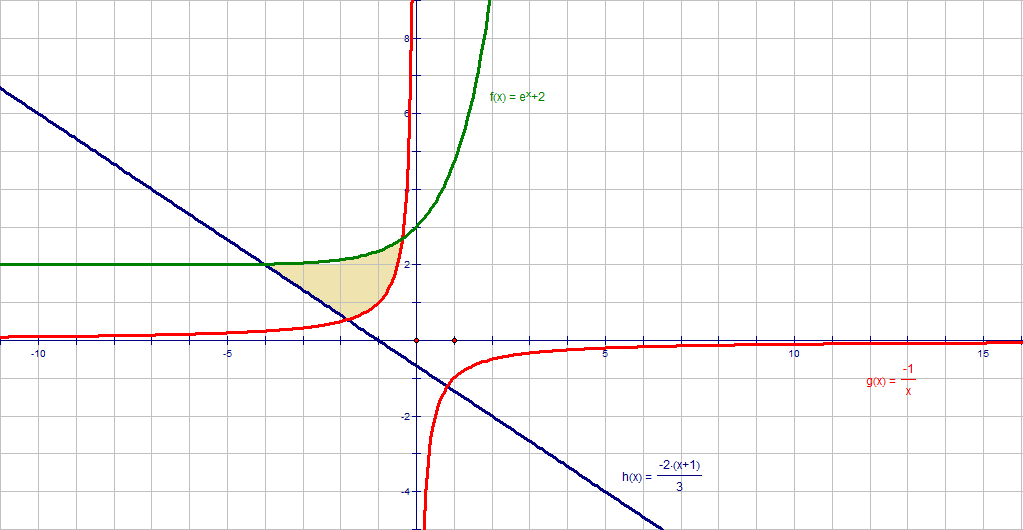
Необходимо вычислить площадь криволинейного треугольника, ограниченного тремя данными функциями.

При этом необходимо найти точки попарного пересечения графиков этих функций методом Ньютона (касательных) и вычислить алгебраическую сумму определённых интегралов на полученных промежутках. Определённые интегралы необходимо вычислять методом Симпсона.

Данные функции:

1. .

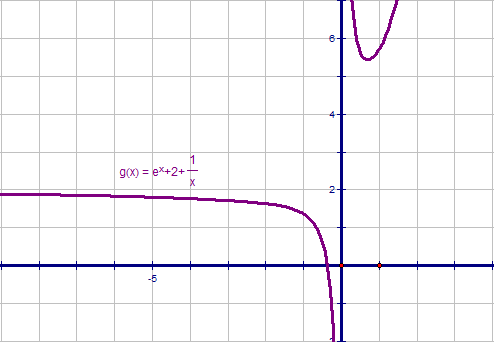
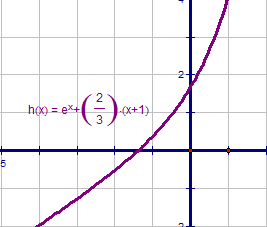
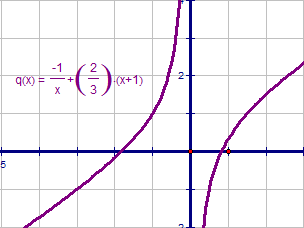
# Математическое обоснование



Бежевым цветом выделена область, площадь которой нам необходимо вычислить. Заметим, что данные функции на этом участке принимают положительные значения, следовательно, мы можем вычислить требуемую площадь как алгебраическую сумму определённых интегралов данных функций.

Очевидно, что необходимое пересечение первой и второй функции находится на интервале [-1.0; -0.1], первой и третьей на [-6.0; -3.0], второй и третьей на [-5.0; -1.0].

Несложными вычислениями проверяем, что первые и вторые производные разностей данных функций на данном промежутке сохраняют свой знак:

1. **
2. **
3. 

Выбор и

Заметим, что максимальная абсолютная погрешность вычисления корня будет равна как раз , а максимальная абсолютная погрешность вычисления интеграла , где M = 10, что заведомо превосходит максимальные значения функций на необходимых промежутках.

Для вычисления площади криволинейного треугольника нам необходимо вычислить алгебраическую сумму трёх интегралов, таким образом, общая абсолютная погрешность не превышает .

Таким образом, т.к. , то можно взять .

# Описание программы

Константы

|  |  |  |
| --- | --- | --- |
| Название | Значение | Назначение |
| EPS | 10e-3 | Определяет необходимую точность |
| INTEGRAL\_P |  | Константа для метода Симпсона из правила Рунге |
| LEFT\_TO\_RIGHT | 0 | Обозначение направления движения в методе касательных |
| RIGHT\_TO\_LEFT | 1 | Обозначение направления движения в методе касательных |

Функции

|  |  |  |
| --- | --- | --- |
| Название | Параметры | Назначение |
| f1, f2, f3 | double x | Значение соответственно первой, второй и третьей данные функции |
| der\_f1, der\_f2, der\_f3 | double x | Соответственно производные первой, второй и третьей функций |
| get\_direction | function f, function g, function der\_f, function der\_g, double left\_bound, double right\_bound | Определение направление движения в методе касательных для нахождения корней разности f и g с производными der\_f и der\_g на промежутке [left\_boud; right\_bound] |
| get\_root | function f, function g, function der\_f, function der\_g, double left\_bound, double right\_bound | Нахождение корней разности f и g с производными der\_f и der\_g на промежутке [left\_bound; right\_bound] методом Ньютона (касательных) |
| get\_integral | function f, double left\_bound, double right\_bound | Нахождение определённого интеграла f на промежутке [left\_bound; right\_bound] методом Cимпсона |

Где тип function – указатель на функцию, принимающую единственный double и возвращающую double.

Выходными данными программы являются точки попарного пересечения графиков данных функций и требуемый результат – площадь криволинейного треугольника, ограниченного ими. Какие-либо входные данные отсутствуют.

# Тестирование

Тестирование функции get\_root

|  |  |  |
| --- | --- | --- |
| Тест | Результат работы программы | Примерный правильный ответ |
|  | -1.898829 | -1.89882892 |
|  | 2.302585 | 2.302585 |
|  | 0.523599 | 0.523599 |

Таким образом, функция get\_root работает нормально на тестовых входных данных, будем считать её правильной.

Тестирование функции get\_integral

|  |  |  |
| --- | --- | --- |
| Тест | Результат работы программы | Примерный правильный ответ |
|  | 32.399987 | 32.4 |
|  | 0.958852 | 0.958851 |
| на [0.5; 0.6] | 0.182322 | 0.182322 |

Таким образом, функция get\_integral работает нормально на тестовых входных данных, будем считать её правильной.

# Программа на языке СИ

#include <stdio.h>

#include <math.h>

const double EPS = 10e-3, INTEGRAL\_P = 1.0 / 15.0;

const int RIGHT\_TO\_LEFT = 1, LEFT\_TO\_RIGHT = 0;

typedef double(\*function)(double);

double abs\_val(double t) {

return t < 0 ? -t : t;

}

double min(double t1, double t2) {

return t1 < t2 ? t1 : t2;

}

double max(double t1, double t2) {

return t1 > t2 ? t1 : t2;

}

double f1(double x) {

return exp(x) + 2.0;

}

double f2(double x) {

return -1.0 / x;

}

double f3(double x) {

return -2.0 \* (x + 1.0) / 3.0;

}

double der\_f1(double x) {

return exp(x);

}

double der\_f2(double x) {

return 1 / (x \* x);

}

double der\_f3(double x) {

return - 2.0 / 3.0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*TESTING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TESTING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

double zero(double x) {

return 0;

}

double root\_test\_f1(double x) {

return -x \* x \* x \* x + 13;

}

double root\_test\_f2(double x) {

return exp(x) - 10;

}

const double PI = acos(-1.0);

double root\_test\_f3(double x) {

return sin(x) - 0.5;

}

double der\_zero(double x) {

return 0;

}

double der\_root\_test\_f1(double x) {

return -4 \* x \* x \* x;

}

double der\_root\_test\_f2(double x) {

return exp(x);

}

double der\_root\_test\_f3(double x) {

return cos(x);

}

//Integral test functions

double integral\_test\_f1(double x) {

return -x \* x \* x \* x + 13.0;

}

double integral\_test\_f2(double x) {

return cos(x);

}

double integral\_test\_f3(double x) {

return 1.0 / x;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*TESTING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TESTING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int get\_direction(function f, function g,

function der\_f, function der\_g,

double left\_bound, double right\_bound) {

double med = (left\_bound + right\_bound) / 2.0, med\_der = der\_f(med) - der\_g(med);

double substr = (f(med) - g(med)) - (f(right\_bound) - g(right\_bound) + f(left\_bound) - g(left\_bound)) / 2.0;

if (substr > 0 && med\_der > 0 || substr < 0 && med\_der < 0)

return LEFT\_TO\_RIGHT;

else

return RIGHT\_TO\_LEFT;

}

double get\_root(function f, function g,

function der\_f, function der\_g,

double left\_bound, double right\_bound, double EPS) {

double cur\_point = get\_direction(f, g, der\_f, der\_g, left\_bound, right\_bound) == RIGHT\_TO\_LEFT ? right\_bound : left\_bound;

while (1) {

double dx = (f(cur\_point) - g(cur\_point)) / (der\_f(cur\_point) - der\_g(cur\_point));

if (abs\_val(dx) < EPS)

break;

cur\_point -= dx;

}

return cur\_point;

}

double get\_integral(function f, double left\_bound, double right\_bound, double EPS) {

double last;

int n, i, j;

for (n = 5, i = 0; ; n \*= 2, i++) {

double cur\_res = 0.0, dx = (right\_bound - left\_bound) / n;

for (j = 0; j < n; j++)

cur\_res += dx \* (f(left\_bound + j \* dx) + 4 \* f(left\_bound + (j + 0.5) \* dx) + f(left\_bound + (j + 1.0) \* dx)) / 6.0;

if (i && INTEGRAL\_P \* abs\_val(cur\_res - last) < EPS)

return last;

last = cur\_res;

}

return last;

}

int main(void) {

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*TESTING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TESTING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

printf("Root tests:\n");

printf("Root test #1: %.6Lf\n", get\_root(root\_test\_f1, zero, der\_root\_test\_f1, der\_zero, -3.0, -0.1, EPS / 10e5));

printf("Root test #2: %.6Lf\n", get\_root(root\_test\_f2, zero, der\_root\_test\_f2, der\_zero, 0.0, 3.0, EPS / 10e5));

printf("Root test #3: %.6Lf\n", get\_root(root\_test\_f3, zero, der\_root\_test\_f3, der\_zero, 0.0, PI / 2 - 10e-4, EPS / 10e5));

printf("\nIntegral tests:\n");

printf("Integral test #1: %.6Lf\n", get\_integral(integral\_test\_f1, -2.0, 1.0, EPS / 10e3));

printf("Integral test #2: %.6Lf\n", get\_integral(integral\_test\_f2, -0.5, 0.5, EPS / 10e3));

printf("Integral test #3: %.6Lf\n", get\_integral(integral\_test\_f3, 0.5, 0.6, EPS / 10e3));

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*TESTING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TESTING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

double int\_1\_2 = get\_root(f1, f2, der\_f1, der\_f2, -1.0, -0.1, EPS / 10e5);

double int\_1\_3 = get\_root(f1, f3, der\_f1, der\_f3, -6.0, -3.0, EPS / 10e5);

double int\_2\_3 = get\_root(f2, f3, der\_f2, der\_f3, -5.0, -1.0, EPS / 10e5);

printf("\nIntersections:\n");

printf("f1 and f2: %.9Lf\n", int\_1\_2);

printf("f1 and f3: %.9Lf\n", int\_1\_3);

printf("f2 and f3: %.9Lf\n", int\_2\_3);

double res = get\_integral(f1, min(int\_1\_2, int\_1\_3), max(int\_1\_2, int\_1\_3), EPS / 10e3) -

get\_integral(f2, min(int\_1\_2, int\_2\_3), max(int\_1\_2, int\_2\_3), EPS / 10e3) -

get\_integral(f3, min(int\_1\_3, int\_2\_3), max(int\_1\_3, int\_2\_3), EPS / 10e3);

printf("\nResult square: %.9Lf\n", res);

return 0;

}

# Полученные результаты

Root tests:

Root test #1: -1.898829

Root test #2: 2.302585

Root test #3: 0.523599

Integral tests:

Integral test #1: 32.399987

Integral test #2: 0.958852

Integral test #3: 0.182322

Intersections:

f1 and f2: -0.371819243

f1 and f3: -4.026748330

f2 and f3: -1.822875655

Result square: 3.563716239

Сравнивая полученные результаты с результатами ручного подсчёта видно, что ошибка составила меньше , что намного лучше требуемой точности .

# Анализ ошибок

Первой и самой значимой ошибкой было не читать с самого начала методичку, т.к., как оказалось, писать все функции и методы необходимо было строго по образцу. Поэтому пришлось переделывать программы в последний день.

Второй ошибкой было написание отчёта в Microsoft Office Word 2010. По той простой причине, что он не предназначен ни для чего чуть большего, чем для печати копированного текста из Википедии. Поэтому ошибка – плохое знание вёрстки в LaTeX.

# Использованная литература

1. Трифонов Н.П., Пильщиков В.Н. Задания практикума на ЭВМ (1 курс)
2. Wikipedia