Вычисление корней уравнений и определённых интегралов

Мингалёв Олег

Московский государственный университет им. М. В. Ломоносова,

Факультет вычислительной математики и кибернетики,

101 группа

oleg@mingalev.net

23 февраля 2014 г.

Оглавление

1 Постановка задачи **2**

1.1 Общая задача 2

1.2 Исследуемые функции 2

2 Используемые численные методы **3**

2.1 Решение уравнений: Метод касательных 3

2.2 Вычисление определённого интеграла: Формула трапеций 4

3 Аналитическая часть **5**

3.1 Приближённый поиск корней 5

3.2 Обоснование применимости метода поиска корней 6

4 Тестирование **7**

5 Результаты счёта **8**

A Исходный код **9**

Глава 1

Постановка задачи

1.1 Общая задача

С заданной точностью ε вычислить площадь плоской фигуры, ограниченной тремя кривыми, уравнения которых определяются функциями $y = f_1(x)$, $y = f_2(x)$ и $y = f_3(x)$.

При решении задачи необходимо:

- С некоторой точностью ε_1 вычислить абсциссы точек пересечения кривых.
- Представить площадь заданной фигуры как алгебраическую сумму определённых интегралов и вычислить эти интегралы с некоторой точностью ε_2.

1.2 Исследуемые функции

$$f_1(x) = 0.6x + 3, \qquad f_2(x) = (x-2)^3 - 1, \qquad f_3(x) = 3/x$$

$$\varepsilon = 10^{-3}$$

Глава 2

Используемые численные методы

2.1 Решение уравнений: Метод касательных

Основная идея метода заключается в следующем: задаётся начальное приближение вблизи предположительного корня, после чего строится касательная к исследуемой функции в точке приближения, для которой находится пересечение с осью абсцисс. Эта точка и берётся в качестве следующего приближения. И так далее, пока не будет достигнута необходимая точность.

Пусть $f(x) : [a, b] \to \mathbb{R}$ — определённая на отрезке $[a, b]$ и дифференцируемая на нём вещественнозначная функция. Тогда формула итеративного исчисления приближений может быть выведена следующим образом:

$$f'(x_n) = \tan\alpha = \frac{\Delta y}{\Delta x} = \frac{f(x_n) - 0}{x_n - x_{n+1}} = \frac{-f(x_n)}{x_{n+1} - x_n}$$

где α — угол наклона касательной в точке x_n.

Следовательно искомое выражение для x_{n+1} имеет вид:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Достаточным условием сходимости метода на отрезке $[a, b]$ является выполнение следующих условий:

1. Функция имеет разные знаки на концах отрезка;

2. Первая и вторая производные функции на отрезке не меняют свой знак.

2.2 Вычисление определённого интеграла: Формула трапеций

Метод трапеций — метод численного интегрирования функции одной переменной, заключающийся в замене на каждом элементарном отрезке подынтегральной функции на многочлен первой степени, то есть линейную функцию. Площадь под графиком функции аппроксимируется прямоугольными трапециями.

Если отрезок $[a, b]$ является элементарным и не подвергается дальнейшему разбиению, значение интеграла можно найти по формуле

$$\int_a^b f(x)\,dx = \frac{f(a)+f(b)}{2}(b-a) + E(f), \qquad E(f) = -\frac{f''(\xi)}{12}\,(b-a)^3$$

Если отрезок $[a, b]$ разбивается узлами интегрирования и на каждом из элементарных отрезков применяется формула трапеций, то суммирование даст составную формулу трапеций

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} \frac{f(x_i)+f(x_{i+1})}{2}(x_{i+1}-x_i) = \frac{f(a)}{2}(x_1-a) + \sum_{i=1}^{n-1} f(x_i)\frac{x_{i+1}-x_{i-1}}{2} + \frac{f(b)}{2}(b-x_{n-1})$$

И, наконец, в случае равномерной сетки:

$$\int_a^b f(x)\,dx = h\left(\frac{f_0+f_n}{2} + \sum_{i=1}^{n-1} f_i\right) + E_n(f), \qquad E_n(f) = -\frac{f''(\xi)}{12}(b-a)h^2$$

Глава 3

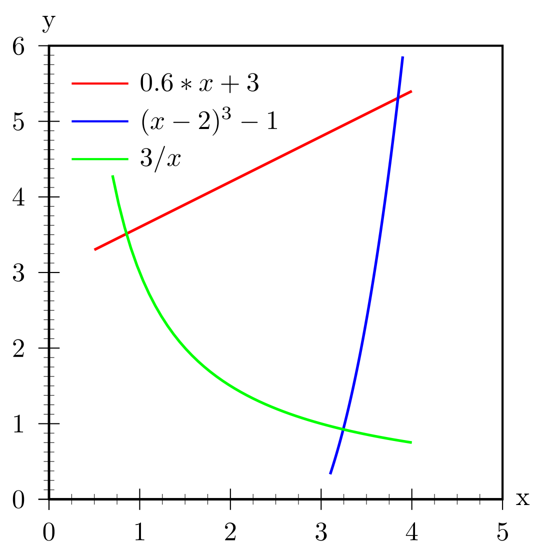
Аналитическая часть

3.1 Приближённый поиск корней

$$\begin{cases} y = 0.6 * x + 3 \\ y = (x-2)^3 - 1 \end{cases} \rightarrow \quad y \in (3, 4)$$

$$\begin{cases} y = 0.6 * x + 3 \\ y = 3/x \end{cases} \rightarrow \quad y \in (1/2, 1)$$

$$\begin{cases} y = (x-2)^3 - 1 \\ y = 3/x \end{cases} \rightarrow \quad y \in (3, 4)$$



3.2 Обоснование применимости метода поиска корней

Для функций $f_{12} = f_1 - f_2$, $f_{13} = f_1 - f_3$ и $f_{23} = f_2 - f_3$ покажем, что на концах выбранных ранее отрезков функции имеет разные знаки, а первые и вторые производные на этих отрезках не меняют свой знак.

$$f_{12}(3) \approx 4.8 > 0 > f_{12}(4) \approx -1.6$$

$$f'_{12} = (0.6) - (3(x-2)^3) = -3x^2 + 12x - 11.4 < 0 \qquad \forall x \in (3,4)$$

$$f''_{12} = (0) - (6x - 12) = 12 - 6x < 0 \qquad \forall x \in (3,4)$$

$$f_{13}(^1\!/_2) \approx -2.7 < 0 < f_{13}(1) \approx 0.6$$

$$f'_{13} = (0.6) - (-\frac{3}{x^2}) = 0.6 + \frac{3}{x^2} > 0 \qquad \forall x \in (^1\!/_2, 1)$$

$$f''_{13} = (0) - (\frac{6}{x^3}) = -\frac{6}{x^3} < 0 \qquad \forall x \in (^1\!/_2, 1)$$

$$f_{23}(3) \approx -1 < 0 < f_{23}(4) \approx 6.25$$

$$f'_{23} = (3(x-2)^2) - (-\frac{3}{x^2}) = 3x^2 + \frac{3}{x^2} - 12x + 12 > 0 \qquad \forall x \in (3,4)$$

$$f''_{23} = (6x - 12) - (\frac{6}{x_3}) = -\frac{6}{x^3} + 6x - 12 > 0 \qquad \forall x \in (3,4)$$

Глава 4

Тестирование

```
[shhdup@shhdup-think lab01]$ ./lab01 -t
lab01 v0.1.0, Mingalev Oleg 2014

Use lab01 -h to see help page

Variant: 6-3-2

Functions:
 0.6x+3
 (x-2)^3-1
 3/x

Root approximation method: Newton's method
Definite integral approximation method: Trapezoidal rule
========================================================

=== Self-testing ===
== Testing root() ==
1. [x^3-2x] x = -1.414214 f(x) = -0.000000 [OK]
2. [1-1/x] x = 1.000000 f(x) = -0.000000 [OK]
3. [-2x-2^(-x)] x = -1.000000 f(x) = -0.000000 [OK]
4. [x^3-x] x = 1.000001 f(x) = 0.000002 [OK]
== Testing integral() ==
1. [x^2] [0.000000;9.000000] I=243.000000 true=243.000000 [OK]
2. [1/x] [1.000000;2.718281] I=1.000000 true=1.000000 [OK]
3. [sqrt(1-x^2)] [-1.000000;1.000000] I=1.570796 true=1.570796 [OK]
=!= All tests are OK =!=
```

Глава 5

Результаты счёта

```
[shhdup@shhdup-think lab01]$ ./lab01 -r
lab01 v0.1.0, Mingalev Oleg 2014

Use lab01 -h to see help page

Variant: 6-3-2

Functions:
 0.6x+3
 (x-2)^3-1
 3/x

Root approximation method: Newton's method
Definite integral approximation method: Trapezoidal rule
========================================================

(1, 2): x = 3.847760, f1(3.847760) = 5.308656, f2(3.847760) = 5.308656
(1, 3): x = 0.854102, f1(0.854102) = 3.512461, f3(0.854102) = 3.512462
(2, 3): x = 3.243929, f2(3.243929) = 0.924807, f3(3.243929) = 0.924804
Area: 7.488425
```

Приложение A

Исходный код

Makefile

```
.SUFFIXES: .c .o

lab01: lab01.o variant.o test.o
   gcc -lm --std=gnu99 -o lab01 variant.o lab01.o test.o

.c.o:
   gcc -lm --std=gnu99 -c -o $@ $<

lab01.o: lab01.c variant.h test.h
test.o: test.c test.h lab01.h
variant.o: variant.c variant.h

clean:
   rm -f *.o
   rm -f *~
   rm -f *.bak
```

lab01.h

```
#ifndef LAB01_H
#define LAB01_H

```

```c
double root(
        double (*f)(double),  // Function f
        double (*f1)(double), // Derivative f
        double (*g)(double),  // Function g
        double (*g1)(double), // Derivative g
        double a, double b,   // Segment with root
        double eps            // Precision
);

double integral(
    // Calc definite integral
        double (*f)(double), // Function
        double a, double b,  // Segment
        double eps           // Precision
);

#endif
```

lab01.c

```c
#include<stdio.h>
#include<math.h>
#include<unistd.h>
#include<stdlib.h>

#include "variant.h"
#include "test.h"


double sign(double d) {
    return (1e-6 < d) - (d < -1e-6);
}

double root(
    // Solve f(x) = g(x)
        double (*f)(double),  // Function f
```

```
        double (*f1)(double), // Derivative f
        double (*g)(double),  // Function g
        double (*g1)(double), // Derivative g
        double a, double b,   // Segment with root
        double eps            // Precision
){
    int right = (f(a)-g(a) < 0) ^
        ((f((a+b)/2) - g((a+b)/2)) * 2 > (f(a)-g(a) + f(b)-g(b)));
    double x;
    if (right) {
        x = b;
        eps *= -1;
    } else {
        x = a;
    }
    while (sign(f(x)-g(x))*sign(f(x+eps)-g(x+eps)) > 0) {
        x = x - (f(x)-g(x)) / (f1(x)-g1(x));
    }
    return x;
}

double integral(
    // Calc definite integral
        double (*f)(double), // Function
        double a, double b,  // Segment
        double eps           // Precision
){
    int n = 32;
    double h = (b-a)/n;
    double I = .5*f(a) + .5*f(b);
    for (int i = 1; i < n; ++i) {
        I += f(a+i*h);
    }

    double I0, h0, n0;

```

```c
    int COUNT = 1;

    do {
        COUNT += 1;
        I0 = I; h0 = h; n0 = n;
        n = n0*2;
        h = (b-a)/n;
        for (int i = 1; i < n; i+=2) {
            I += f(a+i*h);
        }
    } while (fabs(I*h-I0*h0) >= eps/3);
    return I*h;
}


double roots[3][3];
int print_roots = 0;

int find_roots(void) {
    for (int i = 0; i < 3; ++i) {
        for (int j = i+1; j < 3; ++j) {
            if (print_roots) printf("(%d, %d): ", i+1, j+1);
            double x = root(funcs[i][0], funcs[i][1],
                            funcs[j][0], funcs[j][1],
                            segs[i][j][0], segs[i][j][1],
                            0.00001
                        );
            roots[i][j] = x;
            roots[j][i] = x;
            if (print_roots) printf("x = %f, f%d(%f) = %f, f%d(%f) = %f\n",
                    x, i+1, x, (*funcs[i][0])(x),
                    j+1, x, (*funcs[j][0])(x)
            );
        }
    }
}
```

```
double calculate_area(void) {
    double ans = 0;
    ans += integral(funcs[0][0], roots[0][1], roots[0][2], 1e-5);
    ans += integral(funcs[1][0], roots[1][2], roots[1][0], 1e-5);
    ans += integral(funcs[2][0], roots[2][0], roots[2][1], 1e-5);
    return fabs(ans);
}

void help(void) {
    printf("lab01␣v0.1.0,␣Mingalev␣Oleg␣2014\n");
    printf("Usage:␣lab01␣[-h]␣[-t]␣[-r]␣[-v]\n\n");
    printf("Options:\n");
    printf("␣-h:␣Show␣this␣help\n");
    printf("␣-t:␣Run␣tests\n");
    printf("␣-r:␣Print␣roots\n");
    //printf(" -v: Variant info\n");
    exit(0);
}

void variant_info(void){
    printf("lab01␣v0.1.0,␣Mingalev␣Oleg␣2014\n\n");
    printf("Use␣lab01␣-h␣to␣see␣help␣page\n\n");
    printf("Variant:␣6-3-2\n\n");
    printf("Functions:\n");
    printf("␣0.6x+3\n");
    printf("␣(x-2)^3-1\n");
    printf("␣3/x\n\n");
    printf("Root␣approximation␣method:␣Newton's␣method\n");
    printf("Definite␣integral␣approximation␣method:␣Trapezoidal␣rule\n");
    printf("========================================================\n\n");
}

int main(int argc, char *argv[]) {
    int cur = 0;
    variant_info();
```

```c
    while ((cur = getopt(argc, argv, "hrtv")) != -1) {
        switch (cur) {
            case 'h': case '?': help(); break;
            //case 'v': variant_info(); break;
            case 't': exit(run_tests()); break;
            case 'r': print_roots = 1; break;
        }
    }
    find_roots();
    double ans = calculate_area();
    printf("Area: %f\n", ans);
    return 0;
}
```

variant.h

```c
#ifndef VARIANT_H
#define VARIANT_H

double f1(double x);
double f1_d(double x);
double f2(double x);
double f2_d(double x);
double f3(double x);
double f3_d(double x);
extern double (*funcs[3][2])(double);
extern double segs[3][3][2];

#endif
```

variant.c

```c
#include "variant.h"

/* ===== Functions to research  ===== */

```

```
double f1(double x) {
    return .6*x+3;
}
double f1_d(double x) {
    return .6;
}

double f2(double x) {
    return (x-2)*(x-2)*(x-2) - 1;
}
double f2_d(double x) {
    return 3*(x-2)*(x-2);
}

double f3(double x) {
    return 3 / x;
}
double f3_d(double x) {
    return -3 / (x*x);
}

double (*funcs[3][2])(double) = {
    {f1, f1_d},
    {f2, f2_d},
    {f3, f3_d}
};

double segs[3][3][2] = {
    { {0 , 0}, {3 , 4}, {.5, 1} },
    { {3 , 4}, {0 , 0}, {3 , 4} },
    { {.5, 1}, {3 , 4}, {0 , 0} }
};

/* ================================ */
```

test.h

```c
#ifndef TEST_H
#define TEST_H

int run_tests(void);

#endif
```

test.c

```c
#include <math.h>
#include <stdio.h>

#include "test.h"
#include "lab01.h"

double test_root_left_1(double x) {
    return x*x*x - 2*x;
}
double test_root_left_1_d(double x) {
    return 3*x*x - 2;
}

double test_root_left_2(double x) {
    return -1/x + 1;
}
double test_root_left_2_d(double x) {
    return 1 / (x*x);
}

double test_root_right_1(double x) {
    return -pow(2, -x) - 2*x;
}
double test_root_right_1_d(double x) {
    return pow(2, -x) * log(2) - 2;
```

```
}

double test_root_right_2(double x) {
    return x*x*x - x;
}
double test_root_right_2_d(double x) {
    return 3*x*x - 1;
}

double zero(double x) {
    return 0;
}

const int ROOT_TEST_COUNT = 4;
double (*root_tests_funcs[][2])(double) = {
    {test_root_left_1 , test_root_left_1_d},
    {test_root_left_2 , test_root_left_2_d},
    {test_root_right_1 , test_root_right_1_d},
    {test_root_right_2 , test_root_right_2_d}
};
double root_tests_segs[][2] = {
    {-1.5, -1},
    {.5  , 2 },
    {-1.5, 0 },
    {0.8 , 2 }
};
char* root_tests_str[] = {
    "x^3-2x",
    "1-1/x",
    "-2x-2^(-x)",
    "x^3-x",
};

double test_integral_1(double x) {
    return x*x;
}
```

```c

double test_integral_2(double x) {
    return 1/x;
}

double test_integral_3(double x) {
    return sqrt(1 - x*x);
}

int INTEGRAL_TEST_COUNT = 3;

char* integral_tests_str[] = {
    "x^2",
    "1/x",
    "sqrt(1-x^2)"
};

double (*integral_tests_funcs[])(double) = {
    test_integral_1,
    test_integral_2,
    test_integral_3
};

double integral_tests_segs[][2] = {
    {0 , 9},
    {1 , 2.718281},
    {-1, 1}
};

double integral_tests_ans[] = {
   243,
   1,
   1.570796,
};


```

```c
int run_tests(void) {
    printf("===␣Self-testing␣===\n");
    printf("==␣Testing␣root()␣==\n");
    double eps_root = 1e-5;
    int failed = 0;
    for (int test = 0; test < ROOT_TEST_COUNT; ++test) {
        double r = root(
                root_tests_funcs[test][0], root_tests_funcs[test][1],
                zero, zero,
                root_tests_segs[test][0], root_tests_segs[test][1],
                1e-5
        );
        double v = (*root_tests_funcs[test][0])(r);
        printf("%d.␣[%s]␣x␣=␣%f␣f(x)␣=␣%f␣",
                test+1, root_tests_str[test], r, v
        );
        if (fabs(v) < eps_root) {
            printf("[OK]\n");
        } else {
            printf("[FAIL]\n");
            failed = 1;
        }
    }

    double eps_int = 1e-5;
    printf("==␣Testing␣integral()␣==\n");
    for (int test = 0; test < INTEGRAL_TEST_COUNT; ++test) {
        float r = integral(
            integral_tests_funcs[test],
            integral_tests_segs[test][0], integral_tests_segs[test][1],
            1e-5
        );
        printf("%d.␣[%s]␣[%f;%f]␣I=%f␣true=%f␣",
                test+1, integral_tests_str[test],
                integral_tests_segs[test][0], integral_tests_segs[test][1],
                r, integral_tests_ans[test]
```

```c
        );
        if (fabs(r-integral_tests_ans[test]) < eps_int) {
            printf("[OK]\n");
        } else {
            printf("[FAIL]\n");
            failed = 1;
        }
    }

    if (!failed) {
        printf("=!= All tests are OK =!=\n");
        return 0;
    } else {
        printf("!!! Some tests failed !!!\n");
        return 1;
    }
}
```