Московский государственный университет им. М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Отчёт о выполнении практической работы № 2

Численные методы решения дифференциальных уравнений

Выполнил

Маслов Н.С.
группа 205

Москва, 2014

Подвариант №1. Решение задачи Коши для дифференциального уравнения первого порядка или системы дифференциальных уравнений первого порядка

Постановка задачи

Рассматривается обыкновенное дифференциальное уравнение первого порядка, разрешённое относительно производной и имеющее вид:

$$\frac{dy}{dx} = f(x, y), x_0 < x,$$

с дополнительным начальным условием, заданным в точке $x = x_0$:

$$y(x_0) = y_0.$$

Предполагается, что правая часть уравнения - функция $f = f(x, y)$ такова, что гарантирует существование и единственность решения задачи Коши.

В том случае, если рассматривается не одно дифференциальное уравнение, а система ОДУ первого порядка, разрешённых относительно производных неизвестных функций, то соответствующая задача Коши имеет следующий вид:

$$\begin{cases} \frac{dy_1}{dx} = f_1(x, y_1, y_2), \\ \frac{dy_2}{dx} = f_2(x, y_1, y_2), x_0 < x \end{cases}$$

Дополнительные (начальные) условия задаются в точке $x = x_0$:

$$y_1(x_0) = y_1^{(0)}, y_2(x_0) = y_2^{(0)}$$

Также предполагается, что правые части уравнений заданы так, что это гарантирует существование и единственность решения задачи Коши, но уже для системы обыкновенных дифференциальных уравнений первого порядка в форме, разрешённой относительно производных неизвестных функций.

Метод решения

Для решения применяем метод Рунге-Кутта решения разностного уравнения.

Рассмотрим правую часть разностного уравнения:

$$\frac{y_{i+1} - y_i}{h} = f(x_i, y_i) + \frac{1}{2}\left\{\frac{\partial f}{\partial x}(x_i, y_i) + \frac{\partial f}{\partial y}(x_i, y_i) f(x_i, y_i)\right\} h$$

Главная идея метода Рунге-Кутта состоит в том, чтобы приближённо заменить её на сумму значений функции f в двух разных точках с точностью до членов порядка h^2. Если положить

$$f(x_i, y_i) + \frac{1}{2}\left\{\frac{\partial f}{\partial x}(x_i, y_i) + \frac{\partial f}{\partial y}(x_i, y_i) f(x_i, y_i)\right\} h = \beta f(x_i, y_i) + \alpha f(x_i + \gamma h, y_i + \delta h) + o(h^2),$$

где $\alpha, \beta, \gamma, \delta$ - свободные параметры, которые необходимо подобрать; разложить функцию $f(x_i + \gamma h, y_i + \delta h)$ по степеням h и таким образом выразить параметры β, γ, δ через α, то получим уравнение для **однопараметрического семейства разностных схем Рунге-Кутта**:

$$\frac{y_{i+1}-y_i}{h} = (1-\alpha)f(x_i,y_i) + \alpha f\Big(x_i + \frac{h}{2\alpha}, y_i + \frac{h}{2\alpha}f(x_i,y_i)\Big).$$

В частности, наиболее удобные разностные схемы получаются при значениях параметра α: $\alpha = 1/2$ и $\alpha = 1$. При решении задач будем использовать значение $\alpha = 1/2$, задающее схему вычислений "предиктор - корректор".

Для решения систем дифференциальных уравнений первого порядка вычисления незначительно усложняются (для метода Рунге-Кутта второго порядка):

$$y_{1,i+1} = y_{1,i} + \frac{1}{2}\Big(f_1(x_i, y_{1,i}, y_{2,i}) + f_1(x_i, \tilde{y}_{1,i}, y_{2,i})\Big),$$

$$y_{2,i+1} = y_{2,i} + \frac{1}{2}\Big(f_2(x_i, y_{1,i}, y_{2,i}) + f_2(x_i, y_{1,i}, \tilde{y}_{2,i})\Big),$$

$$\tilde{y}_{k,i} = y_{k,i} + f_k(x_i, y_{k,i}, y_{k,i})h, k = 1, 2.$$

Описание программы

Программа после сборки запускает выбранный аргументом командной строки тест, на стандартный поток вывода при этом выводится таблица значений функций.

./diffeq test_num

В первой колонке вывода записываются значения x. В следующих $2n$ колонках ($n = 1, 2$) выводятся значения полученных в результате вычисления сеточных функций $y_1(x), ..., y_n(x)$ последовательно со значениями аналитически полученными решениями.

В последней строке выводится значение получившейся ошибки вычисления, выбранной как максимум модуля расхождения аналитического и численного решения.

Программа реализует вычисление численного решения дифференциального уравнения методом Рунге-Кутта второго и четвёртого порядков точности, а также вычисление численного решения системы двух дифференциальных уравнений методом Рунге-Кутта второго порядка.

Тесты

В качестве результатов тестирования предлагаются графики функций (одновременно построенные для полученного решения и заранее вычисленного аналитического решения).

Во всех тестах взят отрезок $[0, 2]$, шаг сетки 0.01.

Тест 1

Номера тестов в программе 1, 2.

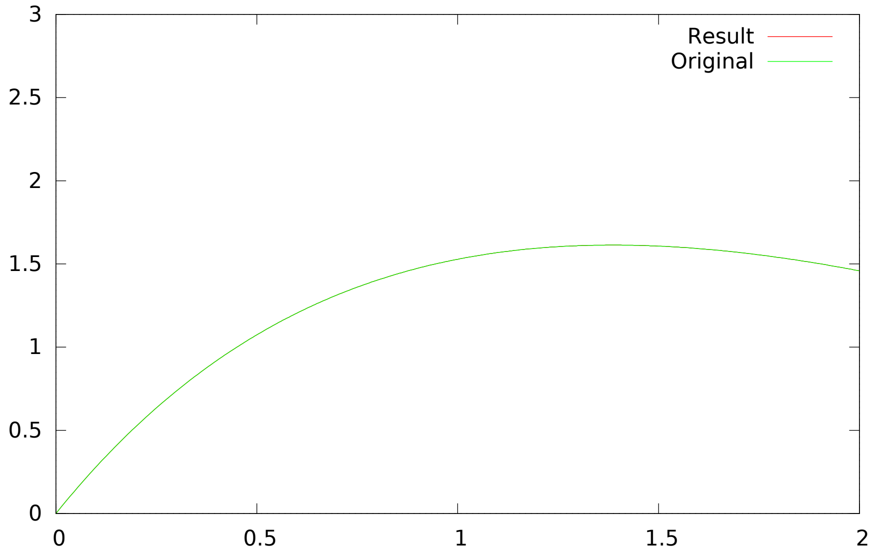
Дано дифференциальное уравнение первого порядка и начальные условия

$$y' = 3 - y - x, y(0) = 0$$

Аналитическое решение задачи Коши:

$$y = 4 - x - e^{-x}$$

Результат работы метода Рунге-Кутта второго порядка точности, максимальная ошибка составила $6.15438 \cdot 10^{-6}$



Из значения максимальной ошибки вычисления ясно, что на этом уравнении метод должен дать очень близкое решение.

Для метода Рунге-Кутта четвёртого порядка точности графики также практически совпадают, максимальная ошибка составила $7.69496 \cdot 10^{-12}$.

Тест 2 (предложенный вариант)

Номера тестов в программе 3, 4.

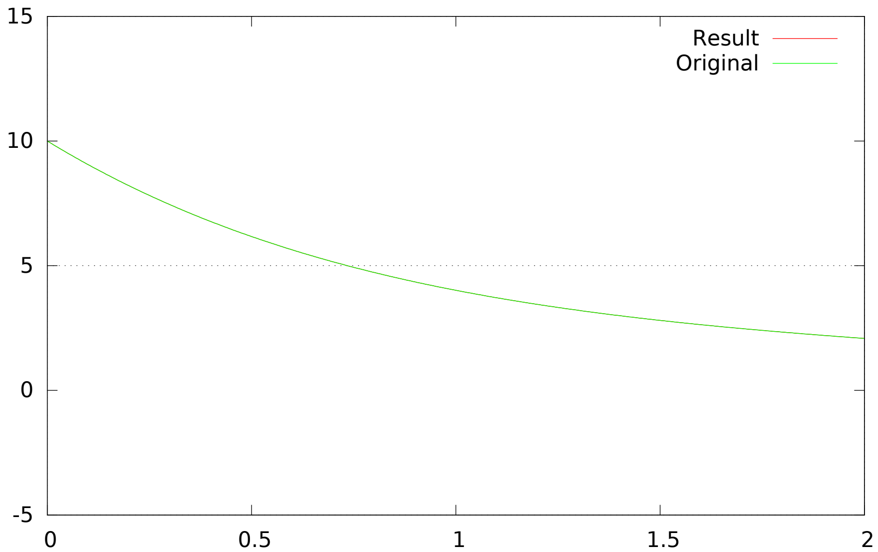
Дано дифференциальное уравнение первого порядка с начальными условиями

$$y' = \sin x - y, y(0) = 10$$

Предложенное аналитическое решение задачи Коши:

$$y = -0.5 \cos x + 0.5 \sin x + \frac{21}{2} e^{-x}$$

Результат работы метода Рунге-Кутта второго порядка точности, максимальная ошибка $1.59266 \cdot 10^{-5}$



Из значения максимальной ошибки вычисления ясно, что на этом уравнении метод должен дать очень близкое решение.

Для метода Рунге-Кутта четвёртого порядка точности графики также практически совпадают, максимальная ошибка составила $1.99263 \cdot 10^{-11}$.

Тест 3

Номера тестов в программе 5, 6.

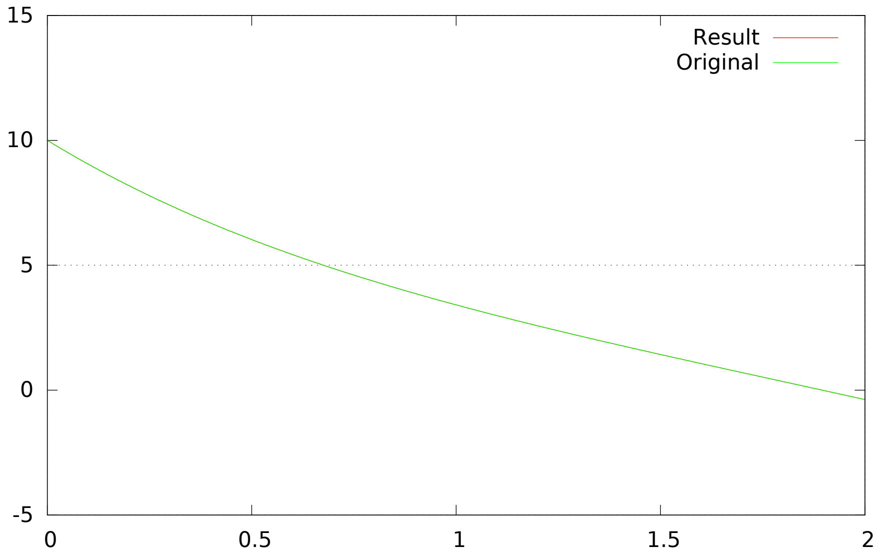
Дано дифференциальное уравнение первого порядка с начальными условиями

$$y' = -y - x^2, y(0) = 10$$

Предложенное аналитическое решение задачи Коши:

$$y = -x^2 + 2x - 2 + 12e^{-x}$$

Результат работы метода Рунге-Кутта второго порядка точности, максимальная ошибка $1.11684 \cdot 10^{-5}$



Из значения максимальной ошибки вычисления ясно, что на этом уравнении метод должен дать очень близкое решение.

Для метода Рунге-Кутта четвёртого порядка точности графики также практически совпадают, максимальная ошибка составила $1.53788 \cdot 10^{-11}$.

Тест 4, система уравнений

Номер теста в программе: 7.

Дана система дифференциальных уравнений первого порядка с начальными условиями

$$\begin{cases} x' = x + y, \\ y' = x + y \end{cases}, x(0) = 3, y(0) = 1$$

Аналитическое решение системы

$$\begin{cases} x = 1 + 2e^{2t}, \\ y = -1 + 2e^{2t} \end{cases}$$

Результат работы метода Рунге-Кутта второго порядка точности для решения системы дифференциальных уравнений, максимальная ошибка составила 1.08293 (это связано с тем, что функции-решения достаточно быстро растут)

График функции x(t):

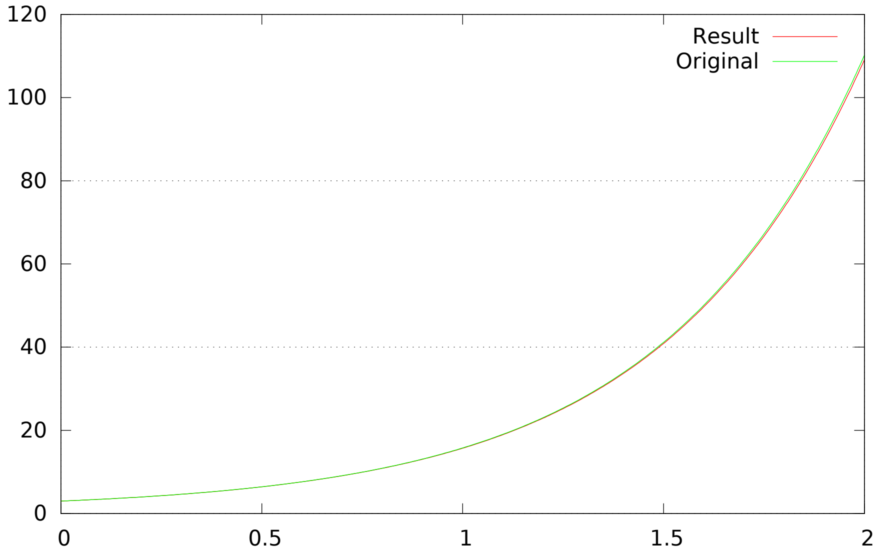
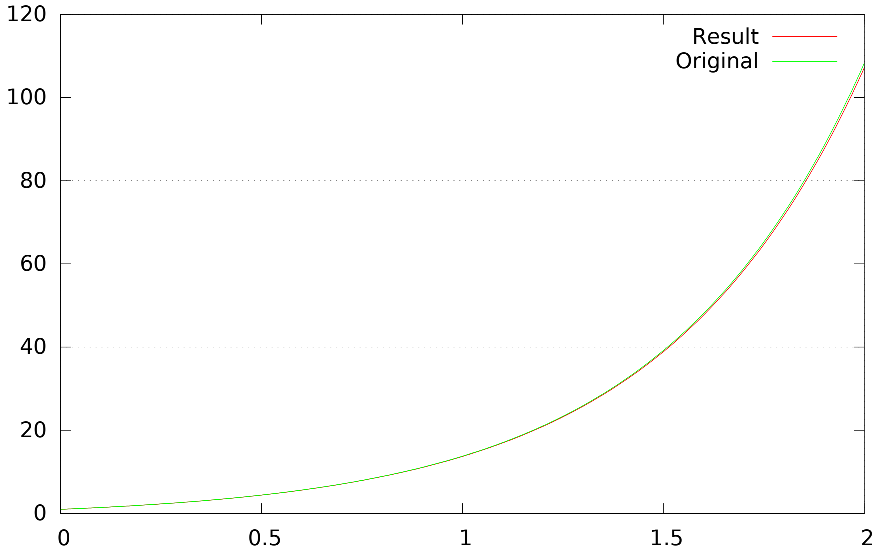


График функции y(t):



Тест 5, система уравнений (предложенный вариант)

Номер теста в программе: 8.

Дана система дифференциальных уравнений первого порядка с начальными условиями

$$\begin{cases} x' = \cos(t + 1.1y) + x, \\ y' = -y^2 + 2.1x + 1.1 \end{cases}, x(0) = 0.25, y(0) = 1$$

Аналитическое решение системы получить не удалось.

Результат работы метода Рунге-Кутта второго порядка точности для решения системы дифференциальных уравнений:

График функции x(t):

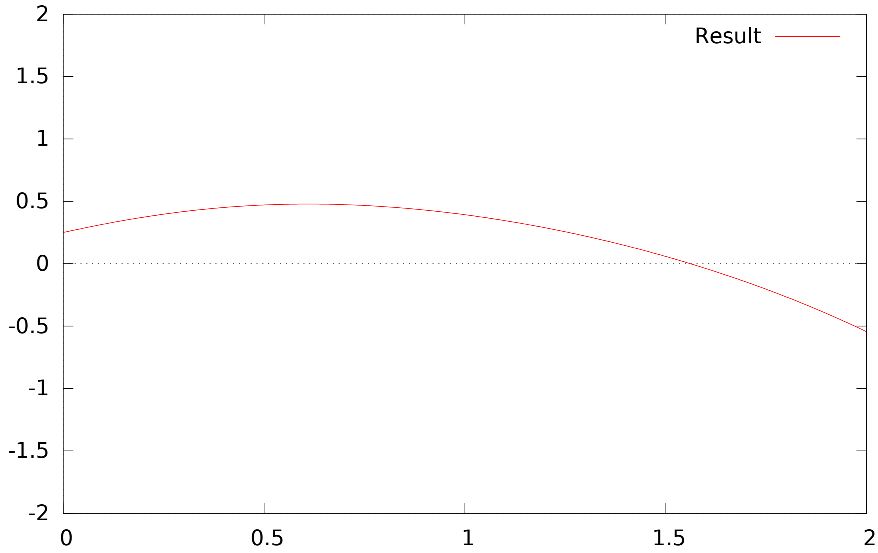
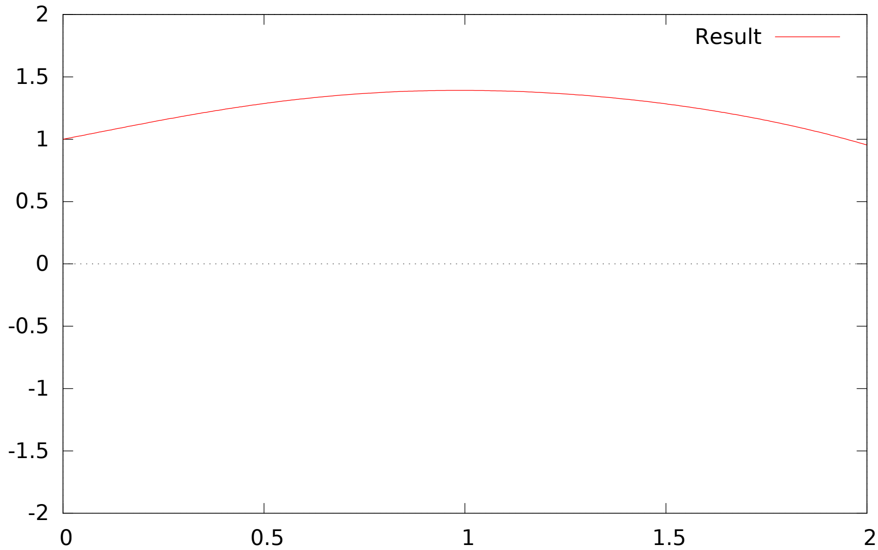


График функции y(t):



Исходный код

Листинг 1: Решение дифференциального уравнения (системы ДУ) методом Рунге-Кутта (файл runge-kutta.c)

```
#include "runge-kutta.h"

#include <assert.h>

#include "grid.h"

/* Решение дифференциального уравнения первого порядка методом РунгеКутта- второго
    порядка точности */
void runge_kutta_2(struct grid_function *grid, double (*f)(struct point), double start
    )
{
        double h = (grid->end - grid->start) / grid->num_pieces;

        for (int i = 0; i <= grid->num_pieces; i++) {
                if (i == 0) {
                        /* Задание начального значения сеточной функции */
                        grid->values[0] = start;
```

```
                } else {
                        /* Подсчёт точек предиктора и корректора */
                        struct point p_predict = grid_getPoint(grid, i - 1);
                        struct point p_correct = p_predict;

                        double correct = grid->values[i - 1] + f(p_predict) * h;
                        p_correct.y = correct;
                        p_correct.x += h;

                        /* Вычисление текущего значения сеточной функции */
                        /* Формула для вычисления следующего значения:
                         * y[i+1] = y[i] + ( f (x[i], y[i]) + f (x[i], p[i]) ) * h / 2,
                         * где p[i] = y[i] + f (x[i], y[i]) * h
                         */
                        grid->values[i] = grid->values[i - 1] + (f(p_predict) + f(p_correct)) * h / 2;
                }
        }
}

/* Решение дифференциального уравнения первого порядка методом РунгеКутта- четвёртого порядка точности */
void runge_kutta_4(struct grid_function *grid, double (*f)(struct point), double start)
{
        double h = (grid->end - grid->start) / grid->num_pieces;

        for (int i = 0; i <= grid->num_pieces; i++) {
                if (i == 0) {
                        /* Задание начального значения сеточной функции */
                        grid->values[0] = start;
                } else {

                        struct point p_predict = grid_getPoint(grid, i - 1);
                        double k1 = f(p_predict);
                        double k2 = f((struct point) { .x = p_predict.x + h / 2, .y = p_predict.y + k1 * h / 2 });
                        double k3 = f((struct point) { .x = p_predict.x + h / 2, .y = p_predict.y + k2 * h / 2 });
                        double k4 = f((struct point) { .x = p_predict.x + h, .y = p_predict.y + k3 * h});

                        /* Формула для вычисления следующего значения сеточной функции представлена ниже */
                        grid->values[i] = grid->values[i - 1] + (k1 + 2 * k2 + 2 * k3 + k4) * h / 6;
                }
        }
}

/* Решение системы дифференциальных уравнений первого порядка методом РунгеКутта- второго порядка точности */
void runge_kutta_system2(struct grid_function *grid1, struct grid_function *grid2, double (*f1)(double, double, double), double (*f2)(double, double, double), double start1, double start2)
{
        /* Проверка совместимости двух сеточных функций */
        assert(grid1->num_pieces == grid2->num_pieces);
        assert(grid1->start == grid2->start);
        assert(grid2->end == grid2->end);
```

```

	double h = (grid1->end - grid1->start) / grid1->num_pieces;

	for (int i = 0; i <= grid1->num_pieces; i++) {
		if (i == 0) {
			/* Задание начальных значений сеточных функций */
			grid1->values[0] = start1;
			grid2->values[0] = start2;
		} else {
			struct point p_predict = grid_getPoint(grid1, i - 1);
			struct point p_correct = p_predict;

			double correct = grid1->values[i - 1] + f1(p_predict.x, p_predict.y, grid2->values[i - 1]) * h;
			p_correct.y = correct;

			/* Формулы для вычисления значений сеточных функций приведены в отчёте */
			grid1->values[i] = grid1->values[i - 1] + (f1(p_predict.x, p_predict.y, grid2->values[i - 1]) + f1(p_correct.x, p_correct.y, grid2->values[i - 1])) * h / 2;

			p_predict = grid_getPoint(grid2, i - 1);
			p_correct = p_predict;

			correct = grid2->values[i - 1] + f2(p_predict.x, grid1->values[i - 1], p_predict.y) * h;
			p_correct.y = correct;

			grid2->values[i] = grid2->values[i - 1] + (f2(p_predict.x, grid1->values[i - 1], p_predict.y) + f2(p_correct.x, grid1->values[i - 1], p_correct.y)) * h / 2;
		}
	}
}
```

Листинг 2: runge-kutta.h

```
#ifndef INCLUDE_RUNGE_KUTTA_H
#define INCLUDE_RUNGE_KUTTA_H

#include "grid.h"

void runge_kutta_2(struct grid_function *grid, double (*f)(struct point), double start);
void runge_kutta_4(struct grid_function *grid, double (*f)(struct point), double start);
void runge_kutta_system2(struct grid_function *grid1, struct grid_function *grid2, double (*f1)(double, double, double), double (*f2)(double, double, double), double start1, double start2);

#endif
```

Подвариант 2. Решение краевой задачи для обыкновенного дифференциального уравнения второго порядка, разрешённого относительно старшей производной

Постановка задачи

Рассматривается линейное дифференциальное уравнение второго порядка вида

$$y'' + p(x) \cdot y' + q(x) \cdot y = -f(x), 1 < x < 0$$

с дополнительными условиями в граничных точках

$$\begin{cases} \sigma_1 y(0) + \gamma_1 y'(0) = \delta_1, \\ \sigma_2 y(1) + \gamma_2 y'(1) = \delta_2 \end{cases}$$

Требуется решить краевую задачу методом конечных разностей, аппроксимировав её разностной схемой второго порядка точности (на равномерной сетке); полученную систему конечно-разностных уравнений решить методом прогонки.

Метод решения

Перейдём от первоначального уравнения к конечно-разностному:

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + p(x_i)\frac{y_{i+1} - y_{i-1}}{2h} + q(x_i)y_i = -f(x_i)$$

Приведём это уравнение к трёхдиагональному виду (выразим y):

$$a_i y_{i-1} - b_i y_i + c_i y_{i+1} = d_i$$

,

$$a_i = \frac{1}{h^2} - \frac{p(x_i)}{2h}, b_i = \frac{2}{h^2} - q(x_i), c_i = \frac{1}{h^2} + \frac{p(x_i)}{2h}, d_i = -f(x_i)$$

Получившуюся линейную систему можно решать обычным способом, но гораздо более короткий путь - использовать метод прогонки. В этом случае решение ищется в виде

$$y_{j-1} = y_j \alpha_j + \beta_j, j = n, (n-1), ..., 2,$$

где α_j, β_j - прогоночные коэффициенты, которые требуется предварительно вычислить.

Соответственно, решение производится в два этапа. На первом этапе (прямой ход) от левого до правого края интервала вычисляются прогоночные коэффициенты. На втором этапе (обратный ход) находится решение уравнения.

Рекуррентные формулы для вычисления прогоночных коэффициентов:

$$\alpha_{i+1} = \frac{c_i}{b_i - a_i \alpha i}$$

$$\beta_{i+1} = \frac{\beta_i a_i - d_i}{b_i - a_i \alpha_i}$$

Начальные значения берём из краевых условий:

$$\alpha_1 = \frac{-\gamma_1}{\sigma_1 h - \gamma_1},$$

$$\beta_1 = \frac{\delta_1 h}{\sigma_1 h - \gamma_1}$$

Затем вычисляем y_i в обратном порядке, начиная от y_n. Начальное значение также считается из краевого условия:

$$y_n = \frac{\gamma_2 \beta_n + \delta_2 h}{\gamma_2(1 - \alpha_n) + \sigma_2 h}$$

Описание программы

Тестирование производится в той же среде с тем же форматом вывода, что и в первом подварианте.

Для хранения уравнения используется три функции: $p(x), q(x)$ и $f(x)$, а также структура, содержащая коэффициенты краевого условия.

Тесты

Во всех тестах шаг сетки взят равным 0.001.

Тест 1

Дана следующая краевая задача на отрезке $[0, 1]$:

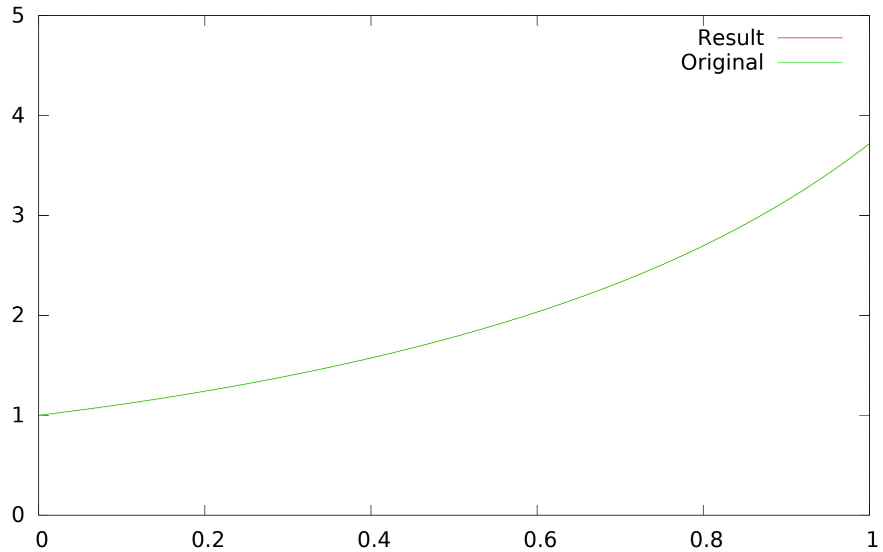
$$y'' - 2xy' - 2y = -4x,$$

$$\begin{cases} y(0) - y'(0) = 0, \\ y(1) = 1 + e \end{cases}$$

Аналитическое решение задачи:

$$y = x + e^{x^2}$$

Результат работы программы, максимальная ошибка составила 0.000281828:



Тест 2

Дана следующая краевая задача на отрезке $[0, 1]$:

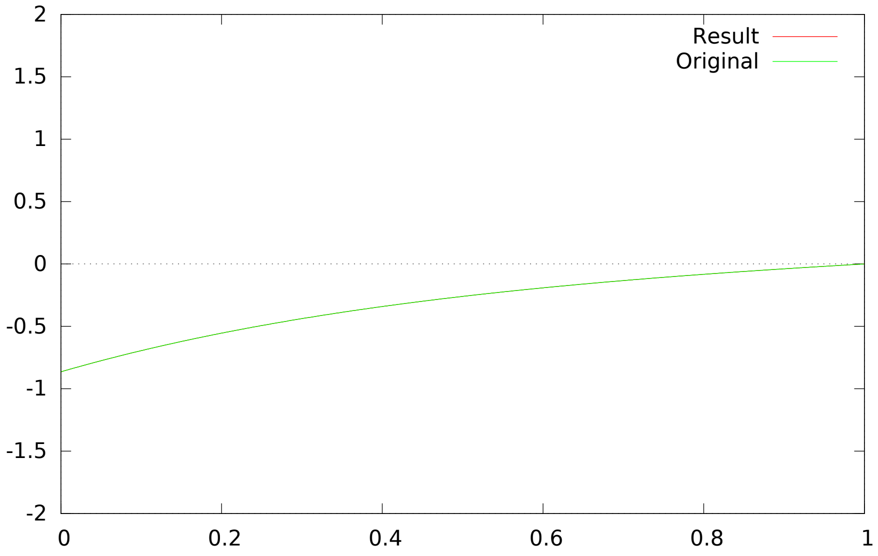
$$y'' + y' - 2y = 0,$$

$$\begin{cases} y(0) + y'(0) = 1, \\ y(1) = 0 \end{cases}$$

Аналитическое решение задачи:

$$y = -\frac{e^{3-2x} - e^{x}}{2 + e^{3}}$$

Результат работы программы, максимальная ошибка составила $5.91875 \cdot 10^{-5}$:



Тест 3

Дана следующая краевая задача на отрезке $[0, 1]$:

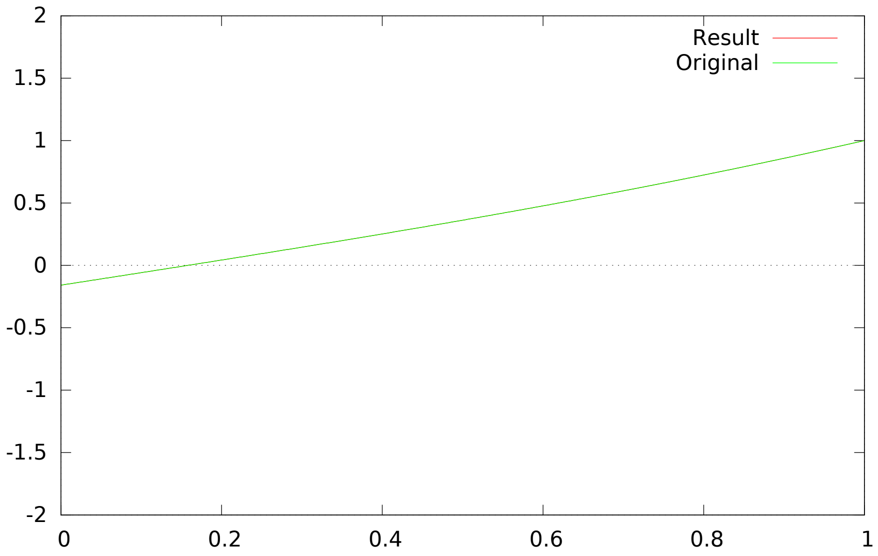
$$y'' = \sin x$$

$$\begin{cases} y'(0) = 1, y(1) = 1 \end{cases}$$

Аналитическое решение задачи:

$$y = 2x - \sin x - 1 + \sin 1$$

Результат работы программы, максимальная ошибка составила $1.53456 \cdot 10^{-7}$:



Тест 4 (предложенный вариант)

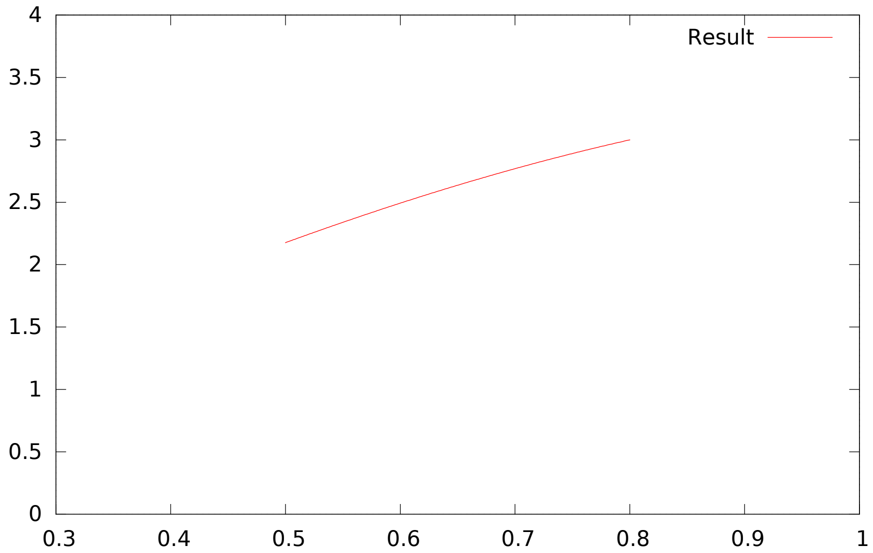
Дана следующая краевая задача на отрезке $[0.5, 0.8]$:

$$y'' + 2x^2 y' + y = x,$$

$$\begin{cases} 2y(0.5) - y'(0.5) = 1, \\ y(0.8) = 3 \end{cases}$$

Аналитическое решение получить не удалось.

Результат работы программы:



Исходный код

Листинг 3: Функция решения краевой задачи

```
#include "boundary.h"

#include <stdlib.h>

void boundary_solve(struct grid_function *grid, double (*p)(double), double (*q)(
    double), double (*f)(double), struct boundary_start s)
{
        /* Allocate memory for tridiagonal solver */
        double *alpha = (double *) malloc((grid->num_pieces + 1) * sizeof (double));
        double *beta = (double *) malloc((grid->num_pieces + 1) * sizeof (double));

        /* Compute tridiagonal solver coeffitients */
        double h = (grid->end - grid->start) / grid->num_pieces;
        double x = grid->start;

        /* Calculate start values */
        alpha[0] = -s.gamma[0] / (s.sigma[0] * h - s.gamma[0]);
        beta[0] = s.delta[0] * h / (s.sigma[0] * h - s.gamma[0]);

        /* Calculate tridiagonal coeffitient values */
        for (int i = 1; i <= grid->num_pieces; i++) {
                /* Calculate all required coeffitients */
                double a = 1 / h / h - p(x) / 2 / h;
                double b = 2 / h / h - q(x);
                double c = 1 / h / h + p(x) / 2 / h;
                double d = -f(x);

                /* Calculate current alpha and beta */
                alpha[i] = c / (b - a * alpha[i - 1]);
                beta[i] = (beta[i - 1] * a - d) / (b - a * alpha[i - 1]);

                x += h;
        }

        /* Calculate function values downward, start from the last point */
```

```

	/* Start point */
	grid->values[grid->num_pieces] = (s.gamma[1] * beta[grid->num_pieces] + s.delta[1] * h) / (s.gamma[1] * (1 - alpha[grid->num_pieces]) + s.sigma[1] * h);

	/* Step process */
	for (int i = grid->num_pieces - 1; i >= 0; i--) {
		grid->values[i] = grid->values[i + 1] * alpha[i + 1] + beta[i + 1];
	}

	/* Free memory */
	free(alpha);
	free(beta);
}
```

Листинг 4: boundary.h

```
#ifndef INCLUDE_BOUNDARY_H
#define INCLUDE_BOUNDARY_H

#include "grid.h"

struct boundary_start {
	double sigma[2];
	double gamma[2];
	double delta[2];
};

void boundary_solve(struct grid_function *grid, double (*p)(double), double (*q)(double), double (*f)(double), struct boundary_start start);

#endif
```

Выводы

Численные методы решения дифференциальных уравнений и систем дифференциальных уравнений позволяют с достаточно высокой точностью получать решения уравнений в условиях, когда получить аналитическое решение не представляется возможным.

Методы не очень сложны для вычислений, рассмотренные выше имеют алгоритмическую сложность $O(n)$, что позволяет проводить требуемые вычисления практически на любых вычислителях с приемлемой скоростью.

При необходимости увеличения точности вычисления, как правило, достаточно уменьшить длину отрезка сеточной функции, что приведёт к увеличению времени работы.

Способ решения с использованием сеточных функций позволяет также применять данные методы к дискретным функциям (например, значениям, полученным с сенсоров или датчиков).

Приложение 1. Исходный код проекта

Исходные коды проекта доступны на Github: https://github.com/webconn/cmc_DiffEquations

Ниже приведены исходные коды модулей, не описанные выше.

Листинг 5: Исходный код интерфейса программы (файл main.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "grid.h"
#include "test.h"
#include "runge-kutta.h"
#include "functions.h"
#include "systems.h"

void test1_rk2(void)
{
        printf("# Testing equation 1, Runge-Kutta, double precision\n");
        test_eq(0, 2, 400, runge_kutta_2, f0, orig0, 0);
}

void test1_rk4(void)
{
        printf("# Testing equation 1, Runge-Kutta, quad precision\n");
        test_eq(0, 2, 400, runge_kutta_4, f0, orig0, 0);
}

void test2_rk2(void)
{
        printf("# Testing equation 2, Runge-Kutta, double precision\n");
        test_eq(0, 2, 400, runge_kutta_2, f1, orig1, 10);
}

void test2_rk4(void)
{
        printf("# Testing equation 2, Runge-Kutta, quad precision\n");
        test_eq(0, 2, 400, runge_kutta_4, f1, orig1, 10);
}

void test3_rk2(void)
{
        printf("# Testing equation 3, Runge-Kutta, double precision\n");
        test_eq(0, 2, 400, runge_kutta_2, f2, orig2, 10);
}

void test3_rk4(void)
{
        printf("# Testing equation 3, Runge-Kutta, quad precision\n");
        test_eq(0, 2, 400, runge_kutta_4, f2, orig2, 10);
}

void test4(void)
{
        printf("# Testing system of equations 1, Runge-Kutta, double precision\n");
        test_sys(0, 2, 400, runge_kutta_system2, fs0_1, fs0_2, x0_orig, y0_orig, 3, 1)
    ;
}

```

```
void test5(void)
{
        printf("# Testing system of equations 2, Runge-Kutta, double precision\n");
        test_sys(0, 2, 400, runge_kutta_system2, f1_1, f1_2, x0_orig, y0_orig, 0.25, 1);
}

void test6(void)
{
        printf("# Testing boundary problem 1\n");
        test_bound(0, 1, 1000, bp0, bq0, bf0, bstart0, borig0);
}

void test7(void)
{
        printf("# Testing boundary problem 2\n");
        test_bound(0, 1, 1000, bp1, bq1, bf1, bstart1, borig1);
}

void test8(void)
{
        printf("# Testing boundary problem 3\n");
        test_bound(0, 1, 1000, bp2, bq2, bf2, bstart2, borig2);
}

void test9(void)
{
        printf("# Testing boundary problem 4\n");
        test_bound(0.5, 0.8, 300, bp3, bq3, bf3, bstart3, NULL);
}

int main(int argc, char *argv[])
{
        if (argc < 2) {
                fprintf(stderr, "Usage: %s test_id\n", argv[0]);
                exit(-1);
        }

        int test_id = atoi(argv[1]);

        switch (test_id) {
                case 1:
                        test1_rk2();
                        break;
                case 2:
                        test1_rk4();
                        break;
                case 3:
                        test2_rk2();
                        break;
                case 4:
                        test2_rk4();
                        break;
                case 5:
                        test3_rk2();
                        break;
                case 6:
                        test3_rk4();
                        break;
                case 7:
                        test4();
```

```
                        break;
                case 8:
                        test5();
                        break;
                case 9:
                        test6();
                        break;
                case 10:
                        test7();
                        break;
                case 11:
                        test8();
                        break;
                case 12:
                        test9();
                        break;
        }

        return 0;
}
```

Листинг 6: Библиотека для работы с сеточными фунциям (файл grid.c)

```
#include "grid.h"

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

/* Создание сеточной функции, подготовка структуры, описывающей сеточную функцию */
struct grid_function *grid_create(double start, double end, int num_pieces)
{
        struct grid_function *ret = (struct grid_function *) malloc(sizeof(struct
    grid_function));
        ret->start = start;
        ret->end = end;
        ret->num_pieces = num_pieces;

        ret->values = (double *) malloc((num_pieces + 1) * sizeof (double));

        return ret;
}

/* Запись значения сеточной функции по номеру значения */
void grid_set(struct grid_function *f, int point, double value)
{
        assert(point >= 0 || point <= f->num_pieces);
        f->values[point] = value;
}

/* Получение значения сеточной функции по номеру значения */
double grid_get(struct grid_function *f, int point)
{
        assert(point < 0 || point >= f->num_pieces);
        return f->values[point];
}

/* Получение точки сеточной функции по номеру координаты( x, y) */
struct point grid_getPoint(struct grid_function *f, int point)
{
        assert(point >= 0 && point <= f->num_pieces);
```

```c

        struct point ret = {
                .x = f->start + (f->end - f->start) * point / f->num_pieces,
                .y = f->values[point]
        };

        return ret;
}

/* Удаление структуры сеточной функции из памяти */
void grid_free(struct grid_function *f)
{
        free(f->values);
        free(f);
}
```

Листинг 7: grid.h

```c
#ifndef INCLUDE_GRID_H
#define INCLUDE_GRID_H

struct grid_function {
        double start;           /**< start of segment */
        double end;             /**< end of segment */
        int num_pieces;         /**< number of points */
        double *values;         /**< values */
};

struct point {
        double x;
        double y;
};

struct grid_function *grid_create(double start, double end, int num_pieces);
void grid_set(struct grid_function *f, int point, double value);
double grid_get(struct grid_function *f, int point);
struct point grid_getPoint(struct grid_function *f, int elem);
void grid_free(struct grid_function *f);

#endif
```

Листинг 8: Тестирующие функции для решателей (файл test.c)

```c
#include "test.h"

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

/* Тестирование решателя дифференциального уравнения первого порядка */
void test_eq(double start, double end, int num_pieces, eq_solver solver, func2 f, func
     orig, double x0)
{
        struct grid_function *grid = grid_create(start, end, num_pieces);
        solver(grid, f, x0);

        double error = 0;

        for (int i = 0; i <= grid->num_pieces; i++) {
                struct point p = grid_getPoint(grid, i);
                double y_orig = orig(p.x);

```

```
                printf("%.4g\t%.4g\t%.4g\n", p.x, p.y, y_orig);

                double c_error = fabs(p.y - y_orig);
                if (c_error > error)
                        error = c_error;
        }

        printf("# Error: %.6g\n", error);
        grid_free(grid);
}

/* Тестирование решателя системы дифференциальных уравнений первого порядка */
void test_sys(double start, double end, int num_pieces, sys_solver solver, func3 f1, func3 f2, func orig1, func orig2, double x0, double y0)
{
        struct grid_function *grid1 = grid_create(start, end, num_pieces);
        struct grid_function *grid2 = grid_create(start, end, num_pieces);

        solver(grid1, grid2, f1, f2, x0, y0);

        double error = 0;

        for (int i = 0; i <= grid1->num_pieces; i++) {
                struct point p1 = grid_getPoint(grid1, i);
                struct point p2 = grid_getPoint(grid2, i);

                double y1_o = orig1(p1.x);
                double y2_o = orig2(p2.x);

                printf("%.4g\t\t%.4g\t%.4g\t\t%.4g\t%.4g\n", p1.x, p1.y, y1_o, p2.y, y2_o);

                double c_error1 = fabs(p1.y - y1_o);
                double c_error2 = fabs(p2.y - y2_o);

                if (c_error2 > c_error1)
                        c_error1 = c_error2;

                if (c_error1 > error)
                        error = c_error1;
        }

        printf("# Error: %.6g\n", error);

        grid_free(grid1);
        grid_free(grid2);
}

/* Тестирование решателя дифференциального уравнения второго порядка */
void test_bound(double start, double end, int num_pieces, func p, func q, func f, struct boundary_start s, func orig)
{
        struct grid_function *grid = grid_create(start, end, num_pieces);
        boundary_solve(grid, p, q, f, s);

        double error = 0;
        for (int i = 0; i <= grid->num_pieces; i++) {
                struct point p = grid_getPoint(grid, i);
                double y_orig = 0;
                if (orig != NULL)
                        y_orig = orig(p.x);
```

```
                printf("%.4g\t%.4g\t%.4g\n", p.x, p.y, y_orig);

                double c_error = fabs(p.y - y_orig);
                if (c_error > error)
                        error = c_error;
        }

        if (orig != NULL)
                printf("# Error: %.6g\n", error);

        grid_free(grid);
}
```

Листинг 9: test.h

```
#ifndef INCLUDE_TEST_H
#define INCLUDE_TEST_H

#include "grid.h"
#include "boundary.h"

typedef double (*func)(double);
typedef double (*func2)(struct point);
typedef double (*func3)(double, double, double);
typedef void (*eq_solver)(struct grid_function *, func2, double);
typedef void (*sys_solver)(struct grid_function *, struct grid_function *, func3,
    func3, double, double);

void test_eq(double start, double end, int num_pieces, eq_solver solver, func2 f, func
     answer, double x0);
void test_sys(double start, double end, int num_pieces, sys_solver solver, func3 f1,
    func3 f2, func answer1, func answer2, double x0, double y0);
void test_bound(double start, double end, int num_pieces, func p, func q, func f,
    struct boundary_start s, func orig);

#endif
```

Листинг 10: Описание дифференциальных уравнений первого порядка из тестов (файл func1.c)

```
#include "functions.h"

#include <math.h>

// start point 0, 10

double f1(struct point p)
{
        return (double) sin(p.x) - p.y;
}

double orig1(double x)
{
        return (-cos(x) + sin(x) + 21 * exp(-x)) / 2;
}

// start point 0, 0

double f0(struct point p)
{
```

```
        return (double) (3 - p.y - p.x);
}

double orig0(double x)
{
        return (double) (4 - x - 4 * exp(-x));
}

// start point 0, 10

double f2(struct point p)
{
        return (double) -p.y - p.x * p.x;
}

double orig2(double x)
{
        return (double) - x * x + 2 * x - 2 + 12 * exp(-x);
}
```

Листинг 11: Описание систем ДУ первого порядка (файл system1.c)

```
#include "systems.h"

#include <math.h>

// start point 0, 0.25, 1

double f1_1(double t, double x, double y)
{
        return cos(t + 1.1 * y) + x;
}

double f1_2(double t, double x, double y)
{
        return -y * y + 2.1 * x + 1.1;
}

// start point 0, 3, 1

double fs0_1(double t, double x, double y)
{
        return x + y;
}

double fs0_2(double t, double x, double y)
{
        return x + y;
}

double x0_orig(double t)
{
        return 1 + 2 * exp(2 * t);
}

double y0_orig(double t)
{
        return -1 + 2 * exp(2 * t);
}
```

Листинг 12: Описание дифференциальных уравнений второго порядка из тестов (файл func2.c)

```c
#include "functions.h"

#include "boundary.h"

#include <math.h>

// test 0, start point 0, end point 1
struct boundary_start bstart0 = {
        .sigma = { 1, 1 },
        .gamma = { -1, 0 },
        .delta = { 0, 3.718 }
};

double bp0(double x)
{
        return -2 * x;
}

double bq0(double x)
{
        return -2;
}

double bf0(double x)
{
        return 4 * x;
}

double borig0(double x)
{
        return x + exp(x * x);
}

// test 1, start point 0, end point 1
struct boundary_start bstart1 = {
        .sigma = { 1, 1 },
        .gamma = { 1, 0 },
        .delta = { 1, 0 }
};

double bp1(double x)
{
        return 1;
}

double bq1(double x)
{
        return -2;
}

double bf1(double x)
{
        return 0;
}

double borig1(double x)
{
        return -(exp(3 - 2 * x) - exp(x)) / (2 + exp(3));
```

```
}

// test 3, start point 0, end point 1
struct boundary_start bstart2 = {
        .sigma = { 0, 1 },
        .gamma = { 1, 0 },
        .delta = { 1, 1 }
};

double bp2(double x)
{
        return 0;
}

double bq2(double x)
{
        return 0;
}

double bf2(double x)
{
        return -sin(x);
}

double borig2(double x)
{
        return 2 * x - sin(x) - 1 + sin(1);
}

// test #14, start point 0.5, end point 0.8

struct boundary_start bstart3 = {
        .sigma = { 2, 1 },
        .gamma = { -1, 0 },
        .delta = { 1, 3 }
};

double bp3(double x)
{
        return 2 * x * x;
}

double bq3(double x)
{
        return 1;
}

double bf3(double x)
{
        return -x;
}
```