

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-Кутты 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-Кутты 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-Кутты**:

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

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-Кутты второго порядка):

$$\begin{aligned} y_{1,i+1} &= y_{1,i} + \frac{1}{2}\left(f_1(x_i, y_{1,i}, y_{2,i}) + f_1(x_i, \tilde{y}_{1,i}, y_{2,i})\right), \\ y_{2,i+1} &= y_{2,i} + \frac{1}{2}\left(f_2(x_i, y_{1,i}, y_{2,i}) + f_2(x_i, y_{1,i}, \tilde{y}_{2,i})\right), \\ \tilde{y}_{k,i} &= y_{k,i} + f_k(x_i, y_{k,i}, y_{k,i})h, k = 1, 2. \end{aligned}$$

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 $y_1(x), \dots, y_n(x)$ 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-Кутты второго и четвёртого порядков точности, а также вычисление численного решения системы двух дифференциальных уравнений методом Рунге-Кутты 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.

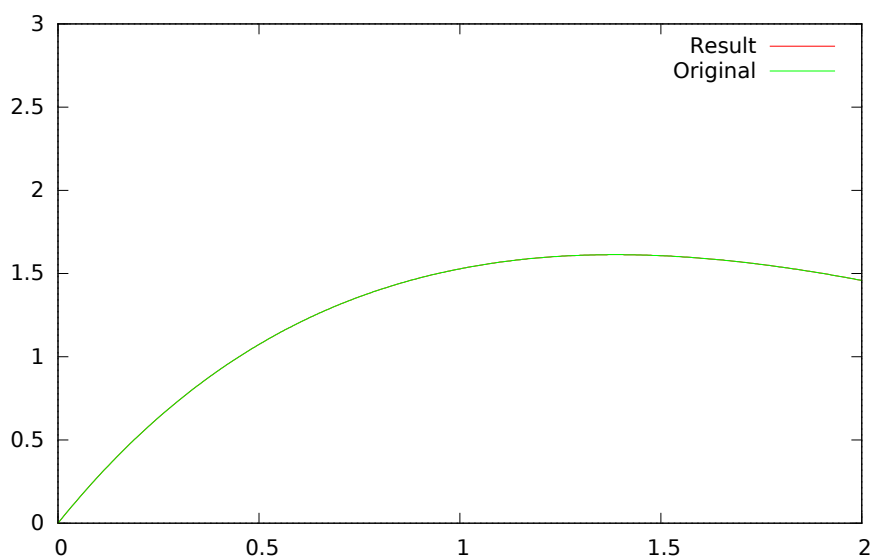
Дано дифференциальное уравнение первого порядка и начальные условия

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

Аналитическое решение задачи Коши:

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

Результат работы метода Рунге-Кутты 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-Кутты 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.

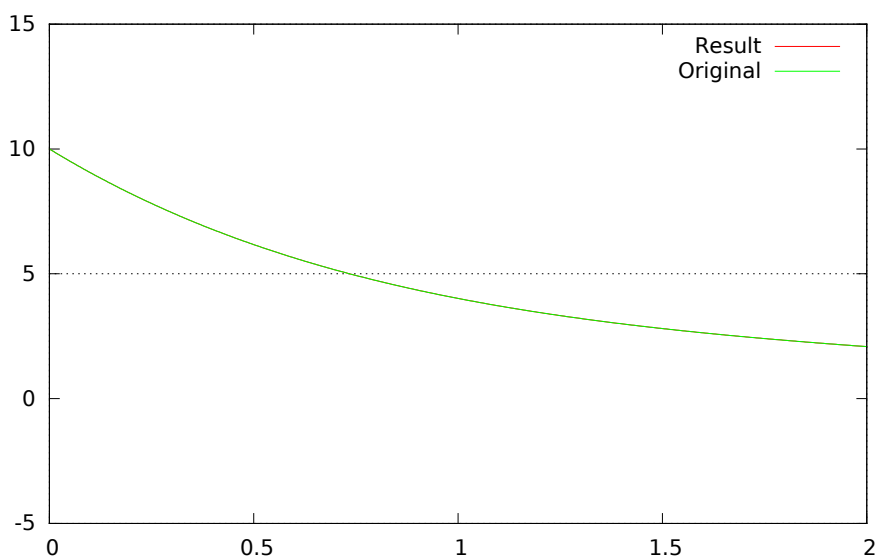
Дано дифференциальное уравнение первого порядка с начальными условиями

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

Предложенное аналитическое решение задачи Коши:

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

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

Тест 3

Номера тестов в программе 5, 6.

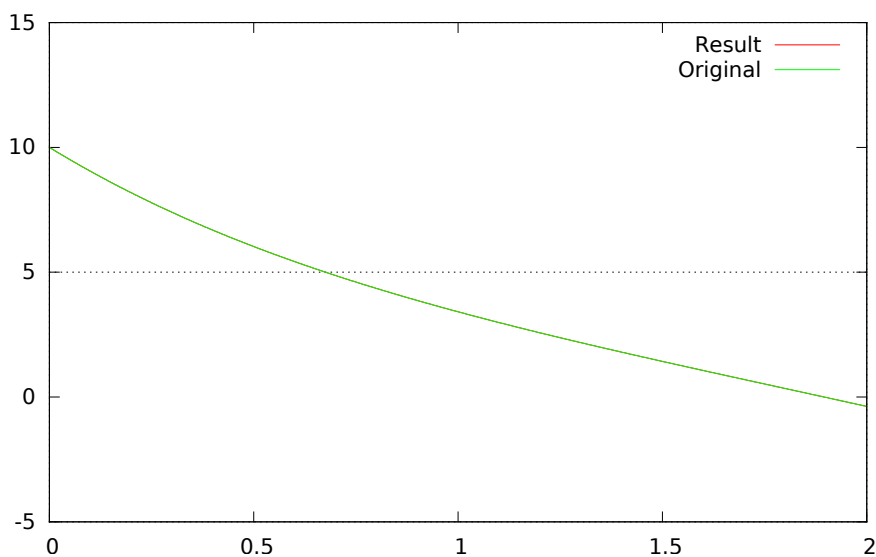
Дано дифференциальное уравнение первого порядка с начальными условиями

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

Предложенное аналитическое решение задачи Коши:

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

Результат работы метода Рунге-Кутты 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-Кутты 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-Кутты 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 $x(t)$:

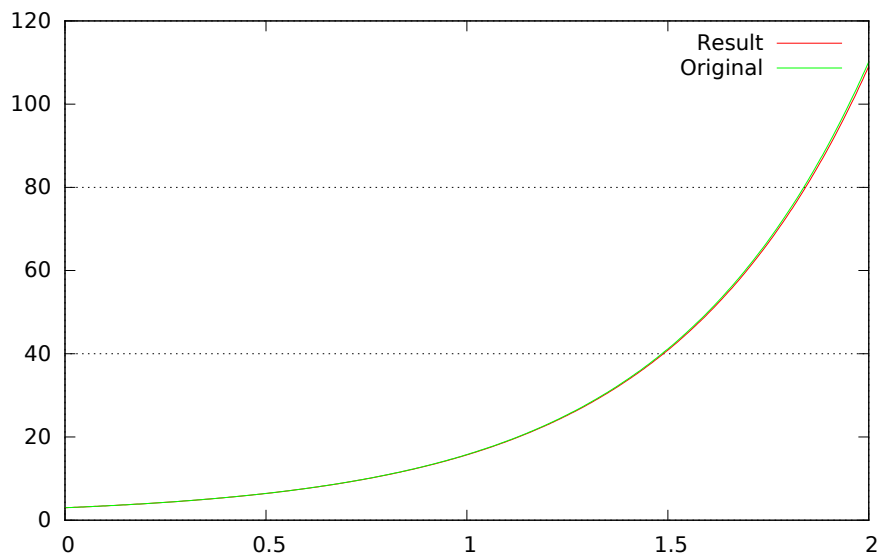
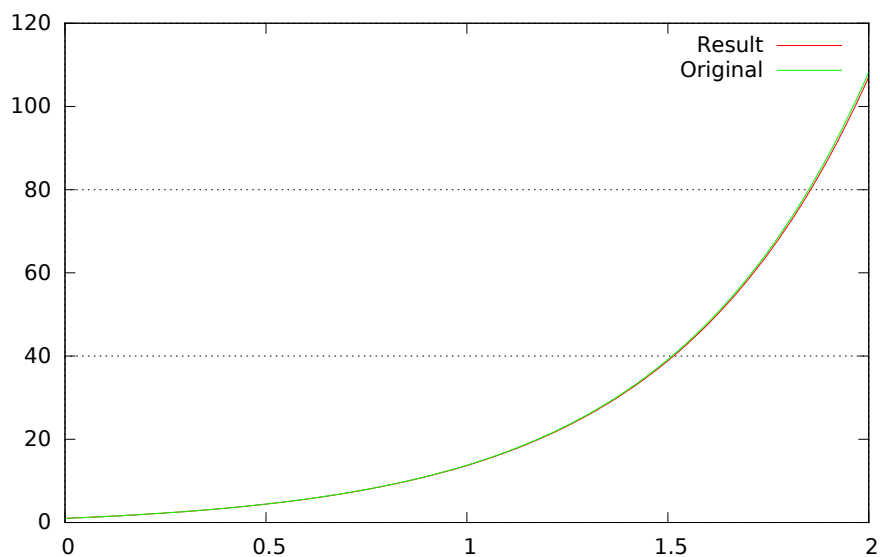


График функции $y(t)$:



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-Кутты второго порядка точности для решения системы дифференциальных уравнений:

График функции $x(t)$:

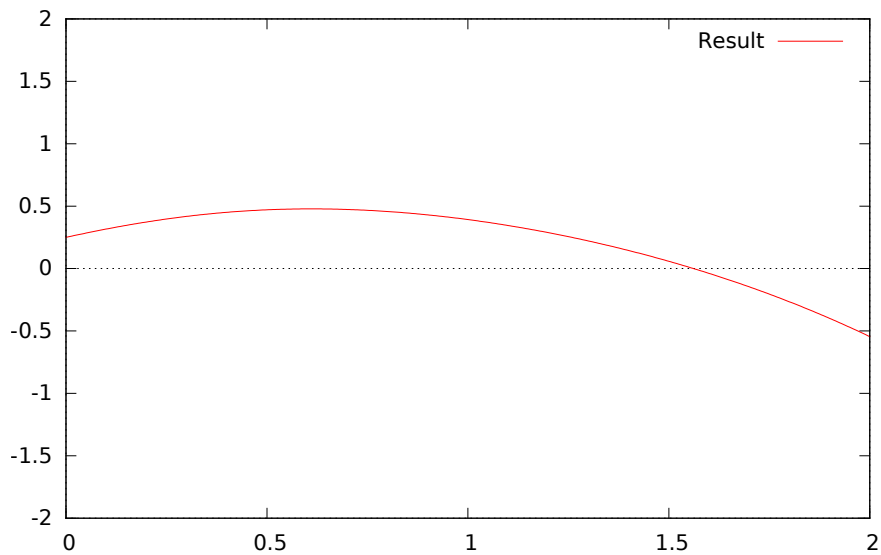
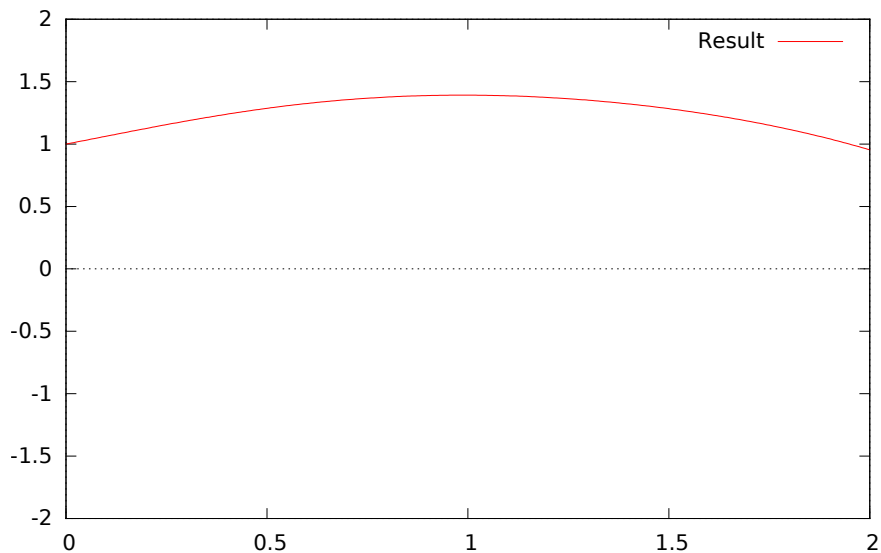


График функции $y(t)$:



Исходный код

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

```

1  #include "runge-kutta.h"
2
3  #include <assert.h>
4
5  #include "grid.h"
6
7  /* Решение дифференциального уравнения первого порядка методом РунгеКутта- второго
   порядка точности */
8  void runge_kutta_2(struct grid_function *grid, double (*f)(struct point), double start
   )
9  {
10     double h = (grid->end - grid->start) / grid->num_pieces;
11
12     for (int i = 0; i <= grid->num_pieces; i++) {
13         if (i == 0) {
14             /* Задание начального значения сеточной функции */
15             grid->values[0] = start;

```

```

16         } else {
17             /* Подсчёт точек предиктора и корректора */
18             struct point p_predict = grid_getPoint(grid, i - 1);
19             struct point p_correct = p_predict;
20
21             double correct = grid->values[i - 1] + f(p_predict) * h;
22             p_correct.y = correct;
23             p_correct.x += h;
24
25             /* Вычисление текущего значения сеточной функции */
26             /* Формула для вычисления следующего значения:
27             *  $y[i+1] = y[i] + ( f(x[i], y[i]) + f(x[i], p[i]) ) * h /$ 
28             * где  $p[i] = y[i] + f(x[i], y[i]) * h$ 
29             */
30             grid->values[i] = grid->values[i - 1] + (f(p_predict) + f(
31             p_correct)) * h / 2;
32         }
33     }
34
35     /* Решение дифференциального уравнения первого порядка методом РунгеКутта- четвёртого
36     порядка точности */
37     void runge_kutta_4(struct grid_function *grid, double (*f)(struct point), double start
38     )
39     {
40         double h = (grid->end - grid->start) / grid->num_pieces;
41
42         for (int i = 0; i <= grid->num_pieces; i++) {
43             if (i == 0) {
44                 /* Задание начального значения сеточной функции */
45                 grid->values[0] = start;
46             } else {
47                 struct point p_predict = grid_getPoint(grid, i - 1);
48                 double k1 = f(p_predict);
49                 double k2 = f((struct point) { .x = p_predict.x + h / 2, .y =
50                 p_predict.y + k1 * h / 2 });
51                 double k3 = f((struct point) { .x = p_predict.x + h / 2, .y =
52                 p_predict.y + k2 * h / 2 });
53                 double k4 = f((struct point) { .x = p_predict.x + h, .y =
54                 p_predict.y + k3 * h });
55
56                 /* Формула для вычисления следующего значения сеточной
57                 функции представлена ниже */
58                 grid->values[i] = grid->values[i - 1] + (k1 + 2 * k2 + 2 * k3
59                 + k4) * h / 6;
60             }
61         }
62     }
63
64     /* Решение системы дифференциальных уравнений первого порядка методом РунгеКутта-
65     второго порядка точности */
66     void runge_kutta_system2(struct grid_function *grid1, struct grid_function *grid2,
67     double (*f1)(double, double, double), double (*f2)(double, double, double), double
68     start1, double start2)
69     {
70         /* Проверка совместимости двух сеточных функций */
71         assert(grid1->num_pieces == grid2->num_pieces);
72         assert(grid1->start == grid2->start);
73         assert(grid2->end == grid2->end);

```



```

65
66     double h = (grid1->end - grid1->start) / grid1->num_pieces;
67
68     for (int i = 0; i <= grid1->num_pieces; i++) {
69         if (i == 0) {
70             /* Задание начальных значений сеточных функций */
71             grid1->values[0] = start1;
72             grid2->values[0] = start2;
73         } else {
74             struct point p_predict = grid_getPoint(grid1, i - 1);
75             struct point p_correct = p_predict;
76
77             double correct = grid1->values[i - 1] + f1(p_predict.x,
p_predict.y, grid2->values[i - 1]) * h;
78             p_correct.y = correct;
79
80             /* Формулы для вычисления значений сеточных функций приведены
в отчёте */
81             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;
82
83             p_predict = grid_getPoint(grid2, i - 1);
84             p_correct = p_predict;
85
86             correct = grid2->values[i - 1] + f2(p_predict.x, grid1->values
[i - 1], p_predict.y) * h;
87             p_correct.y = correct;
88
89             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;
90         }
91     }
92 }

```

Листинг 2: runge-kutta.h

```

1  #ifndef INCLUDE_RUNGE_KUTTA_H
2  #define INCLUDE_RUNGE_KUTTA_H
3
4  #include "grid.h"
5
6  void runge_kutta_2(struct grid_function *grid, double (*f)(struct point), double start
);
7  void runge_kutta_4(struct grid_function *grid, double (*f)(struct point), double start
);
8  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);
9
10 #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

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

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

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

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:

$$\begin{aligned} \alpha_{i+1} &= \frac{c_i}{b_i - a_i \alpha_i} \\ \beta_{i+1} &= \frac{\beta_i a_i - d_i}{b_i - a_i \alpha_i} \end{aligned}$$

Начальные значения берём из краевых условий:

$$\begin{aligned} \alpha_1 &= \frac{-\gamma_1}{\sigma_1 h - \gamma_1}, \\ \beta_1 &= \frac{\delta_1 h}{\sigma_1 h - \gamma_1} \end{aligned}$$

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 $[0, 1]$:

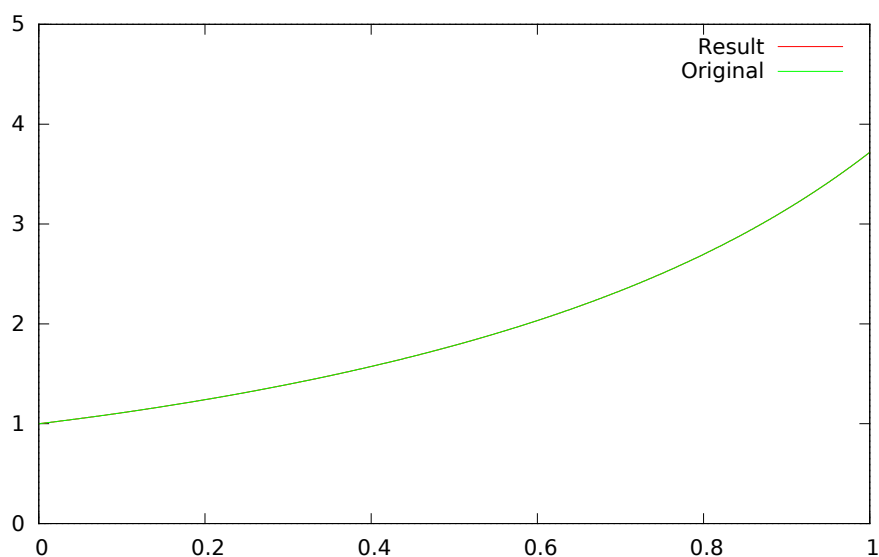
$$y'' - 2xy' - 2y = -4x,$$

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

Аналитическое решение задачи:

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

Результат работы программы, максимальная ошибка составила 0.000281828:



Тест 2

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

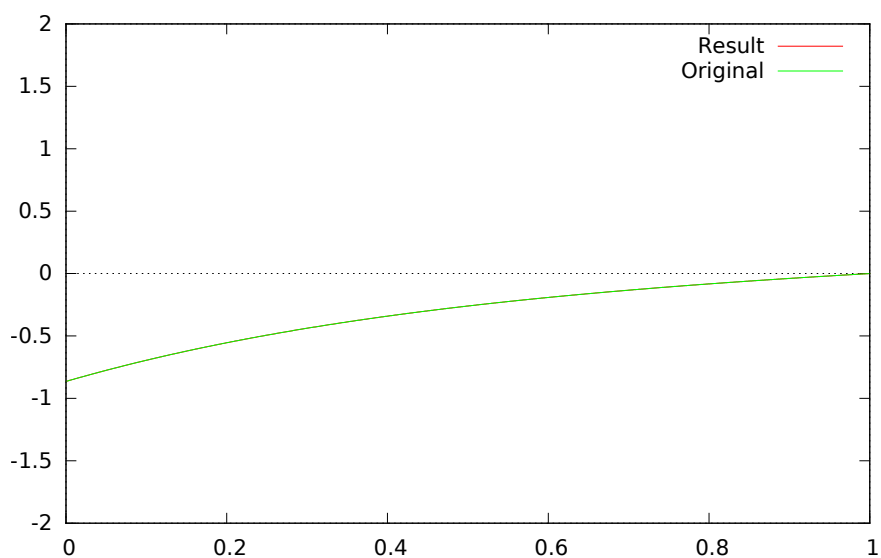
$$y'' + y' - 2y = 0,$$

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

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 $[0, 1]$:

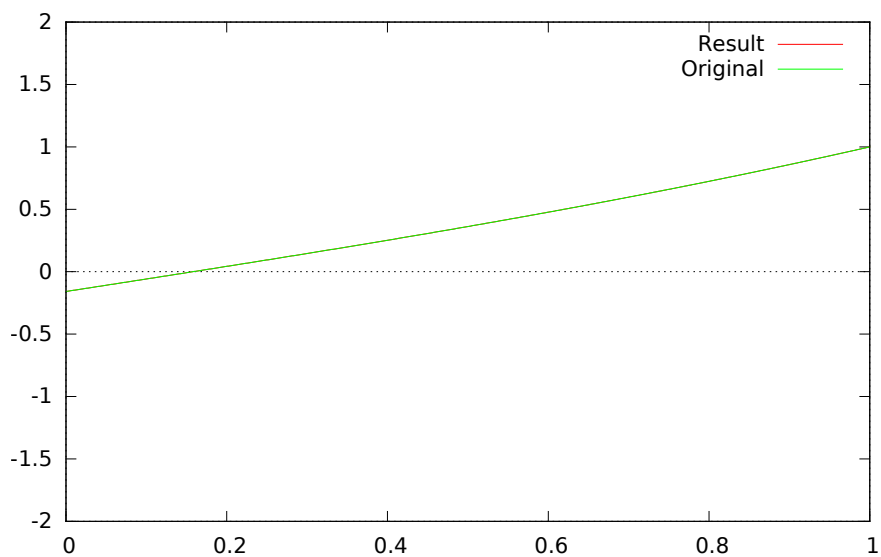
$$y'' = \sin x$$

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

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)

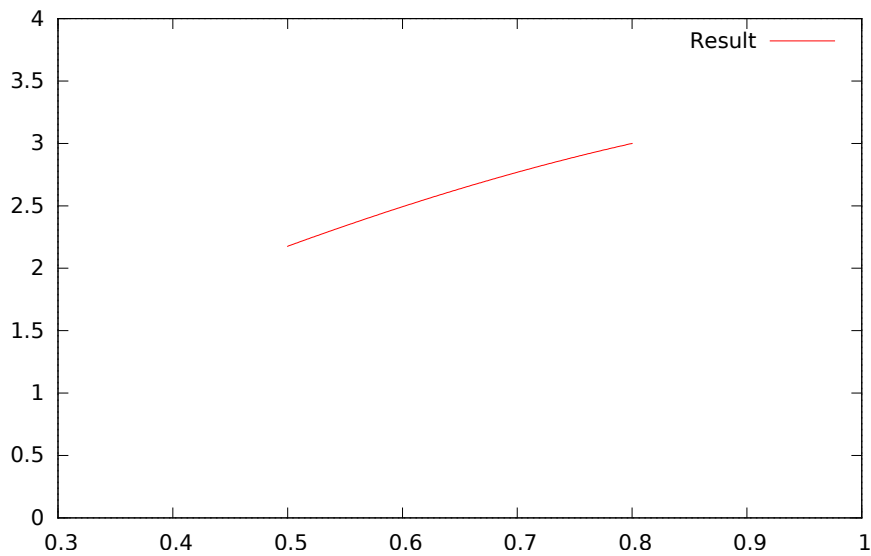
Дана следующая краевая задача на отрезке $[0.5, 0.8]$:

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

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

Аналитическое решение получить не удалось.

Результат работы программы:



Исходный код

Листинг 3: Функция решения краевой задачи

```
1  #include "boundary.h"
2
3  #include <stdlib.h>
4
5  void boundary_solve(struct grid_function *grid, double (*p)(double), double (*q)(
    double), double (*f)(double), struct boundary_start s)
6  {
7      /* Allocate memory for tridiagonal solver */
8      double *alpha = (double *) malloc((grid->num_pieces + 1) * sizeof (double));
9      double *beta = (double *) malloc((grid->num_pieces + 1) * sizeof (double));
10
11     /* Compute tridiagonal solver coefficients */
12     double h = (grid->end - grid->start) / grid->num_pieces;
13     double x = grid->start;
14
15     /* Calculate start values */
16     alpha[0] = -s.gamma[0] / (s.sigma[0] * h - s.gamma[0]);
17     beta[0] = s.delta[0] * h / (s.sigma[0] * h - s.gamma[0]);
18
19     /* Calculate tridiagonal coefficient values */
20     for (int i = 1; i <= grid->num_pieces; i++) {
21         /* Calculate all required coefficients */
22         double a = 1 / h / h - p(x) / 2 / h;
23         double b = 2 / h / h - q(x);
24         double c = 1 / h / h + p(x) / 2 / h;
25         double d = -f(x);
26
27         /* Calculate current alpha and beta */
28         alpha[i] = c / (b - a * alpha[i - 1]);
29         beta[i] = (beta[i - 1] * a - d) / (b - a * alpha[i - 1]);
30
31         x += h;
32     }
33
34     /* Calculate function values downward, start from the last point */
```

```

35
36      /* Start point */
37      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);
38
39      /* Step process */
40      for (int i = grid->num_pieces - 1; i >= 0; i--) {
41          grid->values[i] = grid->values[i + 1] * alpha[i + 1] + beta[i + 1];
42      }
43
44      /* Free memory */
45      free(alpha);
46      free(beta);
47  }

```

Листинг 4: boundary.h

```

1  #ifndef INCLUDE_BOUNDARY_H
2  #define INCLUDE_BOUNDARY_H
3
4  #include "grid.h"
5
6  /* Описание краевых условий уравнения по формуле из отчёта */
7  struct boundary_start {
8      double sigma[2];
9      double gamma[2];
10     double delta[2];
11 };
12
13 void boundary_solve(struct grid_function *grid, double (*p)(double), double (*q)(
double), double (*f)(double), struct boundary_start start);
14
15 #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)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <math.h>
5
6 #include "grid.h"
7 #include "test.h"
8 #include "runge-kutta.h"
9 #include "functions.h"
10 #include "systems.h"
11
12 void test1_rk2(void)
13 {
14     printf("# Testing equation 1, Runge-Kutta, double precision\n");
15     test_eq(0, 2, 400, runge_kutta_2, f0, orig0, 0);
16 }
17
18 void test1_rk4(void)
19 {
20     printf("# Testing equation 1, Runge-Kutta, quad precision\n");
21     test_eq(0, 2, 400, runge_kutta_4, f0, orig0, 0);
22 }
23
24 void test2_rk2(void)
25 {
26     printf("# Testing equation 2, Runge-Kutta, double precision\n");
27     test_eq(0, 2, 400, runge_kutta_2, f1, orig1, 10);
28 }
29
30 void test2_rk4(void)
31 {
32     printf("# Testing equation 2, Runge-Kutta, quad precision\n");
33     test_eq(0, 2, 400, runge_kutta_4, f1, orig1, 10);
34 }
35
36 void test3_rk2(void)
37 {
38     printf("# Testing equation 3, Runge-Kutta, double precision\n");
39     test_eq(0, 2, 400, runge_kutta_2, f2, orig2, 10);
40 }
41
42 void test3_rk4(void)
43 {
44     printf("# Testing equation 3, Runge-Kutta, quad precision\n");
45     test_eq(0, 2, 400, runge_kutta_4, f2, orig2, 10);
46 }
47
48 void test4(void)
49 {
50     printf("# Testing system of equations 1, Runge-Kutta, double precision\n");
51     test_sys(0, 2, 400, runge_kutta_system2, fs0_1, fs0_2, x0_orig, y0_orig, 3, 1)
52     ;
53 }
```



```

54 void test5(void)
55 {
56     printf("# Testing system of equations 2, Runge-Kutta, double precision\n");
57     test_sys(0, 2, 400, runge_kutta_system2, f1_1, f1_2, x0_orig, y0_orig, 0.25,
58     1);
59 }
60 void test6(void)
61 {
62     printf("# Testing boundary problem 1\n");
63     test_bound(0, 1, 1000, bp0, bq0, bf0, bstart0, borig0);
64 }
65
66 void test7(void)
67 {
68     printf("# Testing boundary problem 2\n");
69     test_bound(0, 1, 1000, bp1, bq1, bf1, bstart1, borig1);
70 }
71
72 void test8(void)
73 {
74     printf("# Testing boundary problem 3\n");
75     test_bound(0, 1, 1000, bp2, bq2, bf2, bstart2, borig2);
76 }
77
78 void test9(void)
79 {
80     printf("# Testing boundary problem 4\n");
81     test_bound(0.5, 0.8, 300, bp3, bq3, bf3, bstart3, NULL);
82 }
83
84 int main(int argc, char *argv[])
85 {
86     if (argc < 2) {
87         fprintf(stderr, "Usage: %s test_id\n", argv[0]);
88         exit(-1);
89     }
90
91     int test_id = atoi(argv[1]);
92
93     switch (test_id) {
94         case 1:
95             test1_rk2();
96             break;
97         case 2:
98             test1_rk4();
99             break;
100        case 3:
101            test2_rk2();
102            break;
103        case 4:
104            test2_rk4();
105            break;
106        case 5:
107            test3_rk2();
108            break;
109        case 6:
110            test3_rk4();
111            break;
112        case 7:
113            test4();

```

```

114             break;
115         case 8:
116             test5();
117             break;
118         case 9:
119             test6();
120             break;
121         case 10:
122             test7();
123             break;
124         case 11:
125             test8();
126             break;
127         case 12:
128             test9();
129             break;
130     }
131
132     return 0;
133 }

```

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

```

1  #include "grid.h"
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <assert.h>
6
7  /* Создание сеточной функции, подготовка структуры, описывающей сеточную функцию */
8  struct grid_function *grid_create(double start, double end, int num_pieces)
9  {
10     struct grid_function *ret = (struct grid_function *) malloc(sizeof(struct
        grid_function));
11     ret->start = start;
12     ret->end = end;
13     ret->num_pieces = num_pieces;
14
15     ret->values = (double *) malloc((num_pieces + 1) * sizeof (double));
16
17     return ret;
18 }
19
20 /* Запись значения сеточной функции по номеру значения */
21 void grid_set(struct grid_function *f, int point, double value)
22 {
23     assert(point >= 0 || point <= f->num_pieces);
24     f->values[point] = value;
25 }
26
27 /* Получение значения сеточной функции по номеру значения */
28 double grid_get(struct grid_function *f, int point)
29 {
30     assert(point < 0 || point >= f->num_pieces);
31     return f->values[point];
32 }
33
34 /* Получение точки сеточной функции по номеру координаты( x, y) */
35 struct point grid_getPoint(struct grid_function *f, int point)
36 {
37     assert(point >= 0 && point <= f->num_pieces);

```

```

38
39     struct point ret = {
40         .x = f->start + (f->end - f->start) * point / f->num_pieces,
41         .y = f->values[point]
42     };
43
44     return ret;
45 }
46
47 /* Удаление структуры сеточной функции из памяти */
48 void grid_free(struct grid_function *f)
49 {
50     free(f->values);
51     free(f);
52 }

```

Листинг 7: grid.h

```

1  #ifndef INCLUDE_GRID_H
2  #define INCLUDE_GRID_H
3
4  struct grid_function {
5      double start;           /**< start of segment */
6      double end;             /**< end of segment */
7      int num_pieces;         /**< number of points */
8      double *values;         /**< values */
9  };
10
11 struct point {
12     double x;
13     double y;
14 };
15
16 struct grid_function *grid_create(double start, double end, int num_pieces);
17 void grid_set(struct grid_function *f, int point, double value);
18 double grid_get(struct grid_function *f, int point);
19 struct point grid_getPoint(struct grid_function *f, int elem);
20 void grid_free(struct grid_function *f);
21
22 #endif

```

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

```

1  #include "test.h"
2
3  #include <stdio.h>
4  #include <math.h>
5
6  /* Тестирование решателя дифференциального уравнения первого порядка */
7  void test_eq(double start, double end, int num_pieces, eq_solver solver, func2 f, func
    orig, double x0)
8  {
9      struct grid_function *grid = grid_create(start, end, num_pieces);
10     solver(grid, f, x0);
11
12     double error = 0;
13
14     for (int i = 0; i <= grid->num_pieces; i++) {
15         struct point p = grid_getPoint(grid, i);
16         double y_orig = orig(p.x);
17

```

```

18         printf("%.4g\t%.4g\t%.4g\n", p.x, p.y, y_orig);
19
20         double c_error = fabs(p.y - y_orig);
21         if (c_error > error)
22             error = c_error;
23     }
24
25     printf("# Error: %.6g\n", error);
26     grid_free(grid);
27 }
28
29 /* Тестирование решателя системы дифференциальных уравнений первого порядка */
30 void test_sys(double start, double end, int num_pieces, sys_solver solver, func3 f1,
31     func3 f2, func orig1, func orig2, double x0, double y0)
32 {
33     struct grid_function *grid1 = grid_create(start, end, num_pieces);
34     struct grid_function *grid2 = grid_create(start, end, num_pieces);
35
36     solver(grid1, grid2, f1, f2, x0, y0);
37
38     double error = 0;
39
40     for (int i = 0; i <= grid1->num_pieces; i++) {
41         struct point p1 = grid_getPoint(grid1, i);
42         struct point p2 = grid_getPoint(grid2, i);
43
44         double y1_o = orig1(p1.x);
45         double y2_o = orig2(p2.x);
46
47         printf("%.4g\t%.4g\t%.4g\t%.4g\n", p1.x, p1.y, y1_o, p2.y,
48             y2_o);
49
50         double c_error1 = fabs(p1.y - y1_o);
51         double c_error2 = fabs(p2.y - y2_o);
52
53         if (c_error2 > c_error1)
54             c_error1 = c_error2;
55
56         if (c_error1 > error)
57             error = c_error1;
58     }
59
60     printf("# Error: %.6g\n", error);
61
62     grid_free(grid1);
63     grid_free(grid2);
64 }
65
66 /* Тестирование решателя дифференциального уравнения второго порядка */
67 void test_bound(double start, double end, int num_pieces, func p, func q, func f,
68     struct boundary_start s, func orig)
69 {
70     struct grid_function *grid = grid_create(start, end, num_pieces);
71     boundary_solve(grid, p, q, f, s);
72
73     double error = 0;
74
75     for (int i = 0; i <= grid->num_pieces; i++) {
76         struct point p = grid_getPoint(grid, i);
77         double y_orig = 0;
78         if (orig != NULL)
79             y_orig = orig(p.x);

```

```

76
77         printf("%.4g\t%.4g\t%.4g\n", p.x, p.y, y_orig);
78
79         double c_error = fabs(p.y - y_orig);
80         if (c_error > error)
81             error = c_error;
82     }
83
84     if (orig != NULL)
85         printf("# Error: %.6g\n", error);
86
87     grid_free(grid);
88 }

```

Листинг 9: test.h

```

1  #ifndef INCLUDE_TEST_H
2  #define INCLUDE_TEST_H
3
4  #include "grid.h"
5  #include "boundary.h"
6
7  typedef double (*func)(double);
8  typedef double (*func2)(struct point);
9  typedef double (*func3)(double, double, double);
10 typedef void (*eq_solver)(struct grid_function *, func2, double);
11 typedef void (*sys_solver)(struct grid_function *, struct grid_function *, func3,
    func3, double, double);
12
13 void test_eq(double start, double end, int num_pieces, eq_solver solver, func2 f, func
    answer, double x0);
14 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);
15 void test_bound(double start, double end, int num_pieces, func p, func q, func f,
    struct boundary_start s, func orig);
16
17 #endif

```

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

```

1  #include "functions.h"
2
3  #include <math.h>
4
5  // start point 0, 10
6
7  double f1(struct point p)
8  {
9      return (double) sin(p.x) - p.y;
10 }
11
12 double orig1(double x)
13 {
14     return (-cos(x) + sin(x) + 21 * exp(-x)) / 2;
15 }
16
17 // start point 0, 0
18
19 double f0(struct point p)
20 {

```

```

21         return (double) (3 - p.y - p.x);
22     }
23
24     double orig0(double x)
25     {
26         return (double) (4 - x - 4 * exp(-x));
27     }
28
29     // start point 0, 10
30
31     double f2(struct point p)
32     {
33         return (double) -p.y - p.x * p.x;
34     }
35
36     double orig2(double x)
37     {
38         return (double) - x * x + 2 * x - 2 + 12 * exp(-x);
39     }

```

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

```

1  #include "systems.h"
2
3  #include <math.h>
4
5  // start point 0, 0.25, 1
6
7  double f1_1(double t, double x, double y)
8  {
9      return cos(t + 1.1 * y) + x;
10 }
11
12 double f1_2(double t, double x, double y)
13 {
14     return -y * y + 2.1 * x + 1.1;
15 }
16
17 // start point 0, 3, 1
18
19 double fs0_1(double t, double x, double y)
20 {
21     return x + y;
22 }
23
24 double fs0_2(double t, double x, double y)
25 {
26     return x + y;
27 }
28
29 double x0_orig(double t)
30 {
31     return 1 + 2 * exp(2 * t);
32 }
33
34 double y0_orig(double t)
35 {
36     return -1 + 2 * exp(2 * t);
37 }

```

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

```
1 #include "functions.h"
2
3 #include "boundary.h"
4
5 #include <math.h>
6
7 // test 0, start point 0, end point 1
8 struct boundary_start bstart0 = {
9     .sigma = { 1, 1 },
10    .gamma = { -1, 0 },
11    .delta = { 0, 3.718 }
12 };
13
14 double bp0(double x)
15 {
16     return -2 * x;
17 }
18
19 double bq0(double x)
20 {
21     return -2;
22 }
23
24 double bf0(double x)
25 {
26     return 4 * x;
27 }
28
29 double borig0(double x)
30 {
31     return x + exp(x * x);
32 }
33
34 // test 1, start point 0, end point 1
35 struct boundary_start bstart1 = {
36     .sigma = { 1, 1 },
37     .gamma = { 1, 0 },
38     .delta = { 1, 0 }
39 };
40
41 double bp1(double x)
42 {
43     return 1;
44 }
45
46 double bq1(double x)
47 {
48     return -2;
49 }
50
51 double bf1(double x)
52 {
53     return 0;
54 }
55
56 double borig1(double x)
57 {
58     return -(exp(3 - 2 * x) - exp(x)) / (2 + exp(3));
```

```

59 }
60
61 // test 3, start point 0, end point 1
62 struct boundary_start bstart2 = {
63     .sigma = { 0, 1 },
64     .gamma = { 1, 0 },
65     .delta = { 1, 1 }
66 };
67
68 double bp2(double x)
69 {
70     return 0;
71 }
72
73 double bq2(double x)
74 {
75     return 0;
76 }
77
78 double bf2(double x)
79 {
80     return -sin(x);
81 }
82
83 double borig2(double x)
84 {
85     return 2 * x - sin(x) - 1 + sin(1);
86 }
87
88 // test #14, start point 0.5, end point 0.8
89
90 struct boundary_start bstart3 = {
91     .sigma = { 2, 1 },
92     .gamma = { -1, 0 },
93     .delta = { 1, 3 }
94 };
95
96 double bp3(double x)
97 {
98     return 2 * x * x;
99 }
100
101 double bq3(double x)
102 {
103     return 1;
104 }
105
106 double bf3(double x)
107 {
108     return -x;
109 }

```
