

**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Имени М.В. Ломоносова**

**Факультет вычислительной математики и кибернетики**

**Компьютерный практикум по курсу**

**«ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»**

**ЗАДАНИЕ № 2**

**Численные методы решения дифференциальных уравнений**

**ОТЧЁТ**

**о выполненном задании**

**студента 205 учебной группы факультета ВМК МГУ**

**Жилина Антона Сергеевича**

Москва, 2014

# Практическая работа № 2 (1)

Подвариант № 1

Решение задачи Коши для дифференциального уравнения первого порядка или системы дифференциальных уравнений первого порядка

## ЦЕЛЬ РАБОТЫ

Освоить методы Рунге-Кутты второго и четвертого порядка точности, применяемые для численного решения задачи Коши для дифференциального уравнения (или системы дифференциальных уравнений) первого порядка.

## ПОСТАНОВКА ЗАДАЧИ

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

$$\frac{dy}{dx} = f(x, y), \quad x_0 < x \quad (1)$$

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

$$y(x_0) = y_0 \quad (2)$$

Предполагается, что правая часть уравнения (1) функция  $f(x, y)$  такова, что гарантирует существование и единственность решения задачи Коши (1)-(2).

В том случае, если рассматривается не одно дифференциальное уравнение вида (1), а система обыкновенных дифференциальных уравнений первого порядка, разрешенных относительно производных неизвестных функций, то соответствующая задача Коши имеет вид (на примере двух дифференциальных уравнений):

$$\begin{cases} \frac{dy}{dx} = f_1(x, y, z) \\ \frac{dz}{dx} = f_2(x, y, z) \end{cases}, \quad x_0 < x \quad (3)$$

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

$$y(x_0) = y_0, \quad z(x_0) = z_0 \quad (4)$$

Также предполагается, что правые части уравнений из (3) заданы так, что это гарантирует существование и единственность решения задачи Коши (3)-(4), но уже для системы обыкновенных дифференциальных уравнений первого порядка в форме, разрешенной относительно производных неизвестных функций. Заметим, что к подобным задачам сводятся многие важные задачи, возникающие в механике (уравнения движения материальной точки), небесной механике, химической кинетике, гидродинамике и т.п.

## ЦЕЛИ И ЗАДАЧИ

- 1) Решить задачу Коши (1)-(2) (или (3)-(4)) наиболее известными и широко используемыми на практике методами Рунге-Кутты второго и четвертого порядка точности, аппроксимировав дифференциальную задачу соответствующей разностной схемой (на равномерной сетке);

- полученное конечно-разностное уравнение (или уравнения в случае системы), представляющее фактически некоторую рекуррентную формулу, просчитать численно;
- 2) Найти численное решение задачи и построить его график;
  - 3) Найденное численное решение сравнить с точным решением дифференциального уравнения (подобрать специальные тесты, где аналитические решения находятся в классе элементарных функций, при проверке можно использовать ресурсы on-line системы <http://www.wolframalpha.com> или пакета Maple и т.п.).

## ИСПОЛЬЗОВАННЫЕ МЕТОДЫ

### Метод Рунге-Кутты второго порядка точности

Даны функция  $f, x_0, y_0$ . Требуется найти аппроксимацию решения задачи Коши (1)(2). Выберем число отрезков разбиения  $n$  (чем больше, тем выше точность). Пусть требуется найти решение на отрезке  $[a, b], a = x_0$ . Тогда длина одного отрезка  $h = (b - a)/n$ . Аппроксимация состоит из приближённых значений  $y(x_i), i = \overline{0, n}$ . Они находятся с помощью рекуррентной формулы:

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f(x_{i+1}, y_i + hk_1) \\ y_{i+1} &= y_i + \frac{h}{2}(k_1 + k_2) \end{aligned}$$

Для системы (3)(4):

$$\begin{aligned} k_1 &= f_1(x_i, y_i, z_i) & l_1 &= f_2(x_i, y_i, z_i) \\ k_2 &= f_1(x_{i+1}, y_i + hk_1, z_i + hl_1) & l_2 &= f_2(x_{i+1}, y_i + hk_1, z_i + hl_1) \\ y_{i+1} &= y_i + \frac{h}{2}(k_1 + k_2) & z_{i+1} &= z_i + \frac{h}{2}(l_1 + l_2) \end{aligned}$$

Можно доказать («Вводные лекции по численным методам», стр. 162), что погрешность решения зависит от  $h$  как  $O(h^2)$  при условии, что  $f(f_1, f_2)$  имеет непрерывные вторые частные производные.

### Метод Рунге-Кутты четвёртого порядка точности

Итерационная формула:

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1\right) \\ k_3 &= f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2\right) \\ k_4 &= f(x_i + h, y_i + hk_3) \\ y_{i+1} &= y_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned}$$

Для системы (3)(4):

$$\begin{aligned} k_1 &= f_1(x_i, y_i, z_i) & l_1 &= f_2(x_i, y_i, z_i) \\ k_2 &= f_1\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1, z_i + \frac{h}{2}l_1\right) & l_2 &= f_2\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1, z_i + \frac{h}{2}l_1\right) \\ k_3 &= f_1\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2, z_i + \frac{h}{2}l_2\right) & l_3 &= f_2\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2, z_i + \frac{h}{2}l_2\right) \\ k_4 &= f_1(x_i + h, y_i + hk_3, z_i + hl_3) & l_4 &= f_2(x_i + h, y_i + hk_3, z_i + hl_3) \\ y_{i+1} &= y_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) & z_{i+1} &= z_i + \frac{h}{6}(l_1 + 2l_2 + 2l_3 + l_4) \end{aligned}$$

## ТЕСТИРОВАНИЕ

### Запуск программы

1. Поместите проект на машину под управлением OS Linux. На машине должен быть установлен компилятор GCC.
2. Сделайте `ls` в папку с проектом.
3. Скомпилируйте проект: `make`
4. Перейдите в папку `./bin`
5. Запустите программы для тестирования методов Рунге-Кутты для ур-й и систем:  
`./runge_kutta`  
`./runge_kutta_system`
6. Программы выведут погрешности для тестов, приведённые в таблицах ниже.
7. В той же папке (`./bin`) появятся файлы вида с префиксами `runge-kutta-2`, `runge-kutta-4`, `runge-kutta2-system`, `runge-kutta4-system`. Они описывают решения задач Коши (тестов 1-5) в виде набора точек графика.
8. При желании можно построить визуальные графики с помощью утилиты `gnuplot`. Команды `gnuplot` для ур-й и систем приведены ниже.

```
set style line 1 lc rgb '#0060ad' lt 1 lw 2 pt 7 ps 1.5
```

```
plot 'имя файла' with linespoints ls 1
```

```
set style line 1 lc rgb '#0060ad' lt 1 lw 2 pt 7 ps 1.5
```

```
set style line 2 lc rgb '#dd181f' lt 1 lw 2 pt 5 ps 1.5
```

```
plot 'plotting-data3.dat' index 0 with linespoints ls 1, \  
    '' index 1 with linespoints ls 2
```

### Одиночные ОДУ первого порядка

$f(x, y)$	$[a, b]$	$y_0$	$n$	Точное решение, $y$	Погрешность (1)	Погрешность (2)
$3 - y - x$	$[0, 10]$	0	10	$4 - x - 4e^{-x}$	0.52848224	0.02848224
$3 - y - x$	$[0, 10]$	0	100	$4 - x - 4e^{-x}$	0.00264617	0.00000133
$3 - y - x$	$[0, 10]$	0	1000	$4 - x - 4e^{-x}$	0.00002471	0.00000000
$\sin x - y$	$[0, 10]$	10	100	$0.5 (\sin x - \cos x + 21e^{-x})$	0.00686607	0.00000346
$-y - x^2$	$[0, 10]$	10	100	$x(2 - x) - 2 + 12e^{-x}$	0.00525304	0.00000270

### Системы ОДУ первого порядка

$f_1(x, y, z)$	$f_2(x, y, z)$	$[a, b]$	$y_0$	$z_0$	$n$	Погрешность (1)	Погрешность (2)
$z - \cos x$	$y + \sin x$	$[0, 10]$	0	0	100	8.41272709	0.00269470
$z - \cos x$	$y + \sin x$	$[0, 10]$	0	0	1000	0.09108614	0.00000030
$z - \cos x$	$y + \sin x$	$[0, 10]$	0	0	3000	0.01017260	0.00000000
$2.1 \cdot z - y^2$	$e^{-y} + x + 2.1 \cdot z$	$[0, 10]$	1	0.25	100	N/A	N/A
$(y - z)/x$	$(y + z)/x$	$[1, 10]$	1	1	100	N/A	N/A

В таблицах (1) – метод Рунге-Кутты второго, (2) - четвёртого порядка точности.

Погрешность не подсчитывалась для трёх последних тестов с системами ОДУ, т.к. для них решения в классе элементарных функций не существует.

## ВЫВОДЫ

В ходе практической работы были реализованы метод Рунге-Кутты второго и четвёртого порядков точности, применительно как к «простым» ОДУ первого порядка, разрешённым относительно производной, так и к соответствующим системам.

Тестирование показало, что метод Рунге-Кутты четвёртого порядка точности действительно намного более точный, чем метод второго порядка точности. В то время, как второму хватает 100 итераций для получения приемлемого результата, первому требуется порядка 10000 итераций.

В применении к системам из двух ОДУ первого порядка, метод Рунге-Кутты четвёртого порядка показывает ещё большее преимущество.

## ПРИЛОЖЕНИЕ 1. ПОСТРОЕНИЕ ГРАФИКОВ

Вывод программы – набор точек. Чтобы обозначить их на графике, сформировав графическое изображение, потребуется утилита `gnuplot`. Установить её в ОС Debian и Ubuntu можно командой:

```
sudo apt-get install gnuplot
```

Потребуется ввести пароль администратора и подтвердить установку программы.

Программа запускается из командной строки в виде `gnuplot имя_файла_с_командами`

Сохраните данный фрагмент команд `gnuplot` в файле `./bin/command.gnu`

```
set terminal wxt size 640,480 enhanced font 'Verdana,10' persist
```

```
set style line 1 lc rgb '#0060ad' lt 1 lw 2 pt 5 ps 1.5
```

```
plot 'имя_файла_с_точками' with linespoints ls 1
```

При этом замените `имя_файла_с_точками` на имя одного из файлов с результатами. Чтобы построить два графика для решений системы из двух уравнений, для соответствующего выходного файла надо применить следующий скрипт `gnuplot`:

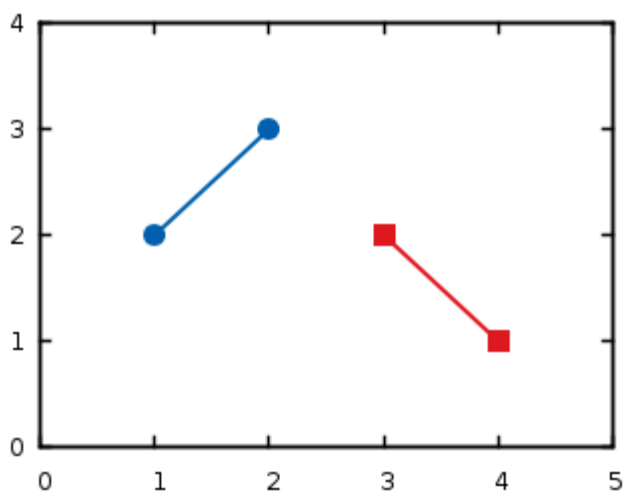
```
set terminal wxt size 640,480 enhanced font 'Verdana,10' persist
```

```
set style line 1 lc rgb '#0060ad' lt 1 lw 2 pt 5 ps 1.5
```

```
set style line 2 lc rgb '#dd181f' lt 1 lw 2 pt 4 ps 1.5
```

```
plot 'имя_файла_с_точками' index 0 with linespoints ls 1, \  
    '' index 1 with linespoints ls 2
```

Результат будет приблизительно похож на следующее:



## ПРИЛОЖЕНИЕ 2. КОД ПРОГРАММЫ

Замечание: код программы, вместе с данным отчётом, доступен в Интернете (адрес указан в печатной версии).

### Общие файлы (не специфичные для задачи)

```
./Makefile
# Флаги gcc

CC_FLAGS      := -O3 -masm=intel -Werror -Wall -Wextra -Wno-unused-result -Wno-
unused-parameter -std=gnu99 -c

LD_FLAGS      :=

# Списки требуемых объектных файлов
C_SOURCES     := $(shell find src -type f -iname '*.c')
C_TARGETS     := $(shell find src/main -type f -iname '*.c')

OBJECTS       := $(C_SOURCES:.c=.o)
O_TARGETS     := $(C_TARGETS:.c=.o)
O_SOURCES     := $(filter-out $(O_TARGETS), $(OBJECTS))
TARGETS       := bin/runge_kutta bin/runge_kutta_system

MAIN_FILES.bin/runge_kutta      := src/main/runge_kutta.o
MAIN_FILES.bin/runge_kutta_system := src/main/runge_kutta_system.o

# Не нужно вызывать `make clean; make` при компиляции заново
all: clean_bin $(TARGETS) clean_obj

# Сборка целей
$(TARGETS): $(OBJECTS)
    gcc $(LD_FLAGS) $(O_SOURCES) $(MAIN_FILES.$@) -o $@ -lm

# Компиляция
%.o: %.c
    gcc $(CC_FLAGS) -o $@ $<

# Очистка
.PHONY: clean_bin clean_obj clean
```

```

clean_bin:
    rm -f $(TARGETS)

clean_obj:
    rm -f $(OBJECTS)

clean:
    rm -f $(TARGETS) $(OBJECTS)

./src/base/common.h
#ifndef COMMON_H
#define COMMON_H

#include <stdbool.h>

typedef long double real;

typedef real (*one_arg_func)(real);
typedef real (*two_arg_func)(real, real);
typedef real (*three_arg_func)(real, real, real);

#define swap(a, b) do { typeof(a) temp = a; \
                        a = b; \
                        b = temp; \
                    } while (0)

bool is_zero(real value);

extern const int OUTPUT_WIDTH;

#endif

```



```

./src/base/common.c
#include <math.h>
#include "common.h"

const int OUTPUT_WIDTH = 18;
const real ZERO_EPSILON = 1e-50;

bool is_zero(real value) {
    return fabs1(value) < ZERO_EPSILON;
}

./src/base/vectors.h
#ifndef VECTORS_H
#define VECTORS_H

#include <stdlib.h>
#include "common.h"

typedef struct vector_m {
    real *storage;
    size_t dimension;
} *vector;

vector new_vector(size_t n);
void delete_vector(vector vec);

real vector_distance(vector v1, vector v2);

#define vidx(vec, idx) vec->storage[idx]

#endif

```

```

./src/base/vectors.c
#include <math.h>
#include "vectors.h"

vector new_vector(size_t n) {
    vector result = malloc(sizeof(struct vector_m));
    result->storage = malloc(n * sizeof(real));
    result->dimension = n;
    return result;
}

void delete_vector(vector vec) {
    free(vec->storage);
    free(vec);
}

real vector_distance(vector v1, vector v2) {
    size_t n = v1->dimension;
    real max = 0;

    for (size_t i = 0; i < n; ++i) {
        real next = fabs1(vidx(v1, i) - vidx(v2, i));
        if (max < next) { max = next; }
    }

    return max;
}

```

## Файлы с реализацией численных методов

```
./src/methods/runge_kutta.h
#include "../base/common.h"
#include "../base/vectors.h"

typedef struct cauchy_problem {
    two_arg_func f;
    real a, b;
    real y0;
    size_t n;
} cauchy_problem;

typedef struct cauchy_solution {
    vector x;
    vector y;
} cauchy_solution;

void print_cauchy_solution(cauchy_solution solution, const char *fname);
real cauchy_solution_error(cauchy_solution solution, one_arg_func u);

cauchy_solution runge_kutta2_solve(cauchy_problem p);
cauchy_solution runge_kutta4_solve(cauchy_problem p);
```

```

./src/methods/runge_kutta.c
#include <stdio.h>

#include "runge_kutta.h"

void print_cauchy_solution(cauchy_solution s, const char *fname) {
    FILE *output = fopen(fname, "w");
    if (output == NULL) { exit(1); }

    size_t n = s.x->dimension;

    for (size_t i = 0; i < n; ++i) {
        fprintf(output, "%-11.8Lf %-11.8Lf\n", vidx(s.x, i), vidx(s.y, i));
    }

    fclose(output);
}

real cauchy_solution_error(cauchy_solution s, one_arg_func u) {
    size_t n = s.x->dimension;
    vector actual_y = new_vector(n);

    for (size_t i = 0; i < n; ++i) {
        vidx(actual_y, i) = u(vidx(s.x, i));
    }

    real result = vector_distance(s.y, actual_y);
    delete_vector(actual_y);
    return result;
}

cauchy_solution runge_kutta2_solve(cauchy_problem p) {
    real h = (p.b - p.a) / p.n;

    vector x = new_vector(p.n + 1);

```

```

vidx(x, 0) = p.a;

vector y = new_vector(p.n + 1);
vidx(y, 0) = p.y0;

real x_i = p.a;
real y_i = p.y0;

for (size_t i = 0; i < p.n; ++i) {
    real x_ip1 = ((p.n-(i+1)) * p.a + (i+1) * p.b) / p.n;

    real k1 = p.f(x_i, y_i);
    real k2 = p.f(x_ip1, y_i + h*k1);

    real y_ip1 = y_i + (h/2) * (k1 + k2);

    vidx(x, i+1) = x_i = x_ip1;
    vidx(y, i+1) = y_i = y_ip1;
}

cauchy_solution solution = { x, y };
return solution;
}

cauchy_solution runge_kutta4_solve(cauchy_problem p) {
    real h = (p.b - p.a) / p.n;

    vector x = new_vector(p.n + 1);
    vidx(x, 0) = p.a;

    vector y = new_vector(p.n + 1);
    vidx(y, 0) = p.y0;

    real x_i = p.a;

```

```

real y_i = p.y0;

for (size_t i = 0; i < p.n; ++i) {
    real x_ip1 = ((p.n-(i+1)) * p.a + (i+1) * p.b) / p.n;
    real x_half = x_i + h/2;

    real k1 = p.f(x_i, y_i);
    real k2 = p.f(x_half, y_i + (h/2)*k1);
    real k3 = p.f(x_half, y_i + (h/2)*k2);
    real k4 = p.f(x_ip1, y_i + h*k3);

    real y_ip1 = y_i + (h/6) * (k1 + 2*k2 + 2*k3 + k4);

    vidx(x, i+1) = x_i = x_ip1;
    vidx(y, i+1) = y_i = y_ip1;
}

cauchy_solution solution = { x, y };
return solution;
}

```

```

./src/methods/runge_kutta_system.h
#include "../base/common.h"
#include "../base/vectors.h"

typedef struct cauchy_problem_system {
    three_arg_func f1;
    three_arg_func f2;
    real a, b;
    real y0;
    real z0;
    size_t n;
} cauchy_problem_system;

typedef struct cauchy_solution_system {
    vector x;
    vector y;
    vector z;
} cauchy_solution_system;

void print_cauchy_solution_system(cauchy_solution_system solution, const char
*fname);

real cauchy_solution_error_u(cauchy_solution_system s, one_arg_func u);
real cauchy_solution_error_v(cauchy_solution_system s, one_arg_func v);

cauchy_solution_system runge_kutta2_solve_system(cauchy_problem_system p);
cauchy_solution_system runge_kutta4_solve_system(cauchy_problem_system p);

```

```

./src/methods/runge_kutta_system.c
#include <stdio.h>

#include "runge_kutta_system.h"

void print_cauchy_solution_system(cauchy_solution_system s, const char *fname)
{
    FILE *output = fopen(fname, "w");
    if (output == NULL) { exit(1); }

    size_t n = s.x->dimension;

    for (size_t i = 0; i < n; ++i) {
        fprintf(output, "%-11.8Lf %-11.8Lf\n", vidx(s.x, i), vidx(s.y, i));
    }

    fprintf(output, "\n");

    for (size_t i = 0; i < n; ++i) {
        fprintf(output, "%-11.8Lf %-11.8Lf\n", vidx(s.x, i), vidx(s.z, i));
    }

    fclose(output);
}

real cauchy_solution_error_u(cauchy_solution_system s, one_arg_func u) {
    size_t n = s.x->dimension;
    vector actual_y = new_vector(n);

    for (size_t i = 0; i < n; ++i) {
        vidx(actual_y, i) = u(vidx(s.x, i));
    }

    real result = vector_distance(s.y, actual_y);
    delete_vector(actual_y);
}

```



```

    return result;
}

real cauchy_solution_error_v(cauchy_solution_system s, one_arg_func v) {
    size_t n = s.x->dimension;
    vector actual_z = new_vector(n);

    for (size_t i = 0; i < n; ++i) {
        vidx(actual_z, i) = v(vidx(s.x, i));
    }

    real result = vector_distance(s.z, actual_z);
    delete_vector(actual_z);
    return result;
}

cauchy_solution_system runge_kutta2_solve_system(cauchy_problem_system p) {
    real h = (p.b - p.a) / p.n;

    vector x = new_vector(p.n + 1);
    vidx(x, 0) = p.a;

    vector y = new_vector(p.n + 1);
    vidx(y, 0) = p.y0;

    vector z = new_vector(p.n + 1);
    vidx(z, 0) = p.z0;

    real x_i = p.a;
    real y_i = p.y0;
    real z_i = p.z0;

    for (size_t i = 0; i < p.n; ++i) {
        real x_ip1 = ((p.n-(i+1)) * p.a + (i+1) * p.b) / p.n;

```

```

    real k1 = p.f1(x_i, y_i, z_i);
    real l1 = p.f2(x_i, y_i, z_i);

    real k2 = p.f1(x_ip1, y_i + h*k1, z_i + h*l1);
    real l2 = p.f2(x_ip1, y_i + h*k1, z_i + h*l1);

    real y_ip1 = y_i + (h/2) * (k1 + k2);
    real z_ip1 = z_i + (h/2) * (l1 + l2);

    vidx(x, i+1) = x_i = x_ip1;
    vidx(y, i+1) = y_i = y_ip1;
    vidx(z, i+1) = z_i = z_ip1;
}

cauchy_solution_system solution = { x, y, z };
return solution;
}

cauchy_solution_system runge_kutta4_solve_system(cauchy_problem_system p) {
    real h = (p.b - p.a) / p.n;

    vector x = new_vector(p.n + 1);
    vidx(x, 0) = p.a;

    vector y = new_vector(p.n + 1);
    vidx(y, 0) = p.y0;

    vector z = new_vector(p.n + 1);
    vidx(z, 0) = p.z0;

    real x_i = p.a;
    real y_i = p.y0;
    real z_i = p.z0;

```

```

for (size_t i = 0; i < p.n; ++i) {
    real x_ip1 = ((p.n-(i+1)) * p.a + (i+1) * p.b) / p.n;
    real x_half = x_i + h/2;

    real k1 = p.f1(x_i, y_i, z_i);
    real l1 = p.f2(x_i, y_i, z_i);

    real k2 = p.f1(x_half, y_i + (h/2)*k1, z_i + (h/2)*l1);
    real l2 = p.f2(x_half, y_i + (h/2)*k1, z_i + (h/2)*l1);

    real k3 = p.f1(x_half, y_i + (h/2)*k2, z_i + (h/2)*l2);
    real l3 = p.f2(x_half, y_i + (h/2)*k2, z_i + (h/2)*l2);

    real k4 = p.f1(x_ip1, y_i + h*k3, z_i + h*l3);
    real l4 = p.f2(x_ip1, y_i + h*k3, z_i + h*l3);

    real y_ip1 = y_i + (h/6) * (k1 + 2*k2 + 2*k3 + k4);
    real z_ip1 = z_i + (h/6) * (l1 + 2*l2 + 2*l3 + l4);

    vidx(x, i+1) = x_i = x_ip1;
    vidx(y, i+1) = y_i = y_ip1;
    vidx(z, i+1) = z_i = z_ip1;
}

cauchy_solution_system solution = { x, y, z };
return solution;
}

```

## Тесты и файлы с `main()`

```
./src/main/runge_kutta.c
#include <stdio.h>

#include <math.h>

#include "../base/vectors.h"

#include "../methods/runge_kutta.h"

real f_test1(real x, real y) {
    return 3 - y - x;
}

real u_test1(real x) {
    return 4 - x - 4*exp1(-x);
}

real f_test2(real x, real y) {
    return sin1(x) - y;
}

real u_test2(real x) {
    return 0.5 * (sin(x) - cos(x) + 21*exp1(-x));
}

real f_test3(real x, real y) {
    return -y - x*x;
}

real u_test3(real x) {
    return (2-x)*x - 2 + 12*exp1(-x);
}

const size_t BUF_SIZE = 80;
```

```

int main(void) {
    cauchy_problem problems[] = {
        // f      a  b   y0 n
        { f_test1, 0, 10, 0, 10  },
        { f_test1, 0, 10, 0, 100 },
        { f_test1, 0, 10, 0, 1000 },
        { f_test2, 0, 10, 10, 100 },
        { f_test3, 0, 10, 10, 100 }
    };

    one_arg_func checks[] = {
        u_test1,
        u_test1,
        u_test1,
        u_test2,
        u_test3
    };

    char fname[BUF_SIZE];

    for (size_t i = 0; i < 5; ++i) {
        cauchy_solution solution = runge_kutta2_solve(problems[i]);

        snprintf(fname, BUF_SIZE, "runge-kutta2-%zu", i+1);
        print_cauchy_solution(solution, fname);

        printf("Runge-Kutta 2: test %zu, error: %11.8Lf\n", i+1,
            cauchy_solution_error(solution, checks[i]));
    }

    for (size_t i = 0; i < 5; ++i) {
        cauchy_solution solution = runge_kutta4_solve(problems[i]);

        snprintf(fname, BUF_SIZE, "runge-kutta4-%zu", i+1);
    }
}

```

```
    print_cauchy_solution(solution, fname);

    printf("Runge-Kutta 4: test %zu, error: %11.8Lf\n", i+1,
           cauchy_solution_error(solution, checks[i]));
}

return 0;
}
```

```

./src/main/runge_kutta_system.c
#include <stdio.h>

#include <math.h>

#include "../methods/runge_kutta_system.h"

real f1_test1(real x, real u, real v) {
    return v - cosl(x);
}

real f2_test1(real x, real u, real v) {
    return u + sinl(x);
}

real u_test1(real x) {
    return -sinl(x);
}

real v_test1(real x) {
    return 0;
}

real f1_test2(real x, real u, real v) {
    return (2.1L)*v - u*u;
}

real f2_test2(real x, real u, real v) {
    return expl(-u) + x + (2.1L)*v;
}

real f1_test3(real x, real u, real v) {
    return (u - v) / x;
}

real f2_test3(real x, real u, real v) {

```

```

    return (u + v) / x;
}

const size_t BUF_SIZE = 80;

int main(void) {
    cauchy_problem_system problems[] = {
        // f1      f2      a b   y0 z0    n
        { f1_test1, f2_test1, 0, 10, 0, 0, 100 },
        { f1_test1, f2_test1, 0, 10, 0, 0, 1000 },
        { f1_test1, f2_test1, 0, 10, 0, 0, 3000 },
        { f1_test2, f2_test2, 0, 10, 1, 0.25, 100 },
        { f1_test3, f2_test3, 1, 10, 1, 1, 100 },
    };

    char fname[BUF_SIZE];

    for (size_t i = 0; i < 5; ++i) {
        cauchy_solution_system          solution          =
runge_kutta2_solve_system(problems[i]);

        snprintf(fname, BUF_SIZE, "runge-kutta2-system-%zu", i+1);
        print_cauchy_solution_system(solution, fname);

        if (i < 3) {
            printf("Runge-Kutta 2: test %zu, error u: %11.8Lf, error v:
%11.8Lf\n", i+1,
                cauchy_solution_error_u(solution, u_test1),
                cauchy_solution_error_v(solution, v_test1));
        }
    }

    for (size_t i = 0; i < 5; ++i) {

```



```

        cauchy_solution_system(solution,
runge_kutta4_solve_system(problems[i]);

        snprintf(fname, BUF_SIZE, "runge-kutta4-system-%zu", i+1);
        print_cauchy_solution_system(solution, fname);

        if (i < 3) {
            printf("Runge-Kutta 4: test %zu, error u: %11.8Lf, error v:
%11.8Lf\n", i+1,
                cauchy_solution_error_u(solution, u_test1),
                cauchy_solution_error_v(solution, v_test1));
        }
    }

    return 0;
}

```