**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-Кутта 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:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

с дополнительным начальным условием, заданным в точке :

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

Предполагается, что правая часть уравнения (1) функция 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):

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

Дополнительные начальные условия задаются в точке :

|  |  |  |
| --- | --- | --- |
|  |  | (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-Кутта 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-Iine системы http://www.wolframalpha.com или пакета Марiе и т.п.).

# Использованные методы

## Метод Рунге-Кутты второго порядка точности

Даны функция . Требуется найти аппроксимацию решения задачи Коши (1)(2). Выберем число отрезков разбиения n (чем больше, тем выше точность). Пусть требуется найти решение на отрезке . Тогда длина одного отрезка . Аппроксимация состоит из приближённых значений . Они находятся с помощью рекуррентной формулы:

Для системы (3)(4):

|  |  |
| --- | --- |
|  |  |

Можно доказать («Вводные лекции по численным методам», стр. 162), что погрешность решения зависит от как при условии, что () имеет непрерывные вторые частные производные.

## Метод Рунге-Кутты четвёртого порядка точности

Итерационная формула:

Для системы (3)(4):

|  |  |
| --- | --- |
|  |  |

# 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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | Точное решение, | Погрешность (1) | Погрешность (2) |
|  |  | 0 | 10 |  | 0.52848224 | 0.02848224 |
|  |  | 0 | 100 |  | 0.00264617 | 0.00000133 |
|  |  | 0 | 1000 |  | 0.00002471 | 0.00000000 |
|  |  | 10 | 100 |  | 0.00686607 | 0.00000346 |
|  |  | 10 | 100 |  | 0.00525304 | 0.00000270 |

## Системы ОДУ первого порядка

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | Погрешность (1) | Погрешность (2) |
|  |  |  | 0 | 0 | 100 | 8.41272709 | 0.00269470 |
|  |  |  | 0 | 0 | 1000 | 0.09108614 | 0.00000030 |
|  |  |  | 0 | 0 | 3000 | 0.01017260 | 0.00000000 |
|  |  |  | 1 | 0.25 | 100 | N/A | N/A |
|  |  |  | 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

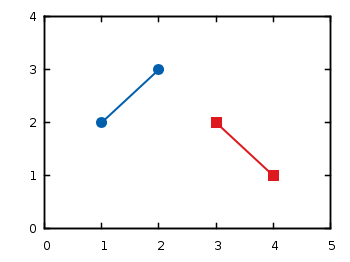
set style line 1 lc rgb '#0060ad' lt 1 lw 2 pt 5 ps 1.5

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 fabsl(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 = fabsl(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\*expl(-x);

}

real f\_test2(real x, real y) {

return sinl(x) - y;

}

real u\_test2(real x) {

return 0.5 \* (sin(x) - cos(x) + 21\*expl(-x));

}

real f\_test3(real x, real y) {

return -y - x\*x;

}

real u\_test3(real x) {

return (2-x)\*x - 2 + 12\*expl(-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;

}