Московский государственный университет
имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу

«Введение в численные методы»
Задание 2

Числнные методы решения дифференцальных уравнений

ОТЧЕТ

о выполненном задании

студента 203 учебной группы факультета ВМК МГУ
Травниковой Арины Сергеевны

гор. Москва
2018 г.

Содержание

Постановка задачи

Часть 1

Рассматривается ОДУ первого порядка, разрешённое относительно производной и имеющее вид, с дополнительным начальным условием в точка a:

$$\begin{cases} \frac{dy}{dx} = f(x,y) \quad a \leq x \leq b \\ y(a) = y_0 \end{cases}$$

Необходимо найти решение данной задачи Коши в предположении, что правая часть уравнения $f = f(x,y)$ таковы, что гарантирует существование и единственность решения задачи Коши.

Часть 2

Рассматривается система линейных ОДУ первого порядка, разрешённых относительно производной, с дополнительными условиями в точке a :

$$\begin{cases} \frac{dy_1}{dx} = f_1(x,y_1,y_2) \\ \frac{dy_2}{dx} = f_2(x,y_1,y_2) \quad a \leq x \leq b \\ y_1(a) = y_1^0 \\ y_2(a) = y_2^0 \end{cases}$$

Необходимо найти решение данной задачи Коши в предположении, что правые части уравнений таковы, что гарантируют существование и единственность решения задачи Коши для системы.

Часть 3

Рассматривается краевая задача для дифференциального уравнения второго порядка с дополнительными условиями в граничных точках:

$$\begin{cases} y" + p(x)y' + q(x)y = f(x) \\ \sigma_1 y(a) + \gamma_1 y'(a) = \delta_1 \qquad\qquad a \leq x \leq b \\ \sigma_2 y(b) + \gamma_2 y'(b) = \delta_2 \end{cases}$$

Необходимо найти решение данной краевой задачи.

Цели

Часть 1

Изучить методы Рунге-Кутта второго и четвертого порядка точности, применяемые для численного решения задач Коши для дифференциального уравнения (или системы) первого порядка:

- Решить заданные задачи Коши для методоми Рунге-Кутта второго и четвертого порядка точности, апроксимировав дифференциальну задачу соответствующей разностной схемой на равномерной сетке; полученное конечно-разностное уравнение просчитать численно
- Найти численное решение и построить его график
- Сравнить численное решение с точным на различных тестах, используя wolframalpha.com

Часть 2

Изучить метод прогонки решения краевой задачи для дифференциального уравнения второго порядка:

- Решить краевую заданную задачу методом конечных разностей, аппроксимировав ее разностной схеой второго порядка точности на равномерной сетке; полученную систему конечно-разностных уравнений решить методом прогонки
- Найти численное решение задачи и построить его график
- Сравнить численное решение и с точным на различных тестах, используя wolframalpha.com

Описание алгоритмов

Часть 1

Будем использовать следующие формулы для численного решения задачи Коши, приближающие точное решение с четвёртым порядком точности относительно диаметра разбиения отрезка, на котором решается поставленная задача.
Положим:

- n - число точек разбиения отрезка
- $h = \frac{a-b}{n}$ - диаметр разбиения отрезка
- $x_i = a + h * i, y_i = y(x_i), 0 \leq i \leq n$ - сетка и сеточная функция

Метод Рунге-Кутты 2 порядка точности для рекуррентного вычисления сеточной функции примет следующий вид:

$$y_{i+1} = y_i + \frac{h}{2}(f(x_i + f(x_i + h, y_i + h * f(x_i))))$$

Метод Рунге-Кутты 4 порядка точности для рекуррентного вычисления сеточной функции примет следующий вид:

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

Часть 2

Метод Рунге-Кутты 2 порядка для рекуррентного вычисления сеточной функции примет следующий вид:

$$\begin{cases} k_1 = f(x_i, y_1^i, y_2^i) \\ k_2 = f(x_i + \frac{h}{2}, y_1^i + \frac{h}{2}k_1, y_2^i + \frac{h}{2}k_{21}) \\ k_{21} = f(x_i, y_1^i, y_2^i) \\ k_{22} = f(x_i + \frac{h}{2}, y_1^i + \frac{h}{2}k_1, y_2^i + \frac{h}{2}k_{21}) \\ y_1^{i+1} = y_1^i + hk_2 \\ y_2^{i+1} = y_2^i + hk_{22}) \end{cases}$$

Метод Рунге-Кутты 4 порядка для рекуррентного вычисления сеточной функции примет следующий вид:

$$\begin{cases} k_1 = f(x_i, y_1^i, y_2^i) \\ k_2 = f(x_i + \frac{h}{2}, y_1^i + \frac{h}{2}k_1, y_2^i + \frac{h}{2}k_{21}) \\ k_3 = f(x_i + \frac{h}{2}, y_1^i + \frac{h}{2}k_2, y_2^i + \frac{h}{2}k_{22}) \\ k_4 = f(x_i + h, y_1^i + hk_3, y_2^i + hk_{23}) \\ k_{21} = f(x_i, y_1^i, y_2^i) \\ k_{22} = f(x_i + \frac{h}{2}, y_1^i + \frac{h}{2}k_1, y_2^i + \frac{h}{2}k_{21}) \\ k_{23} = f(x_i + \frac{h}{2}, y_1^i + \frac{h}{2}k_2, y_2^i + \frac{h}{2}k_{22}) \\ k_{24} = f(x_i + h, y_1^i + hk_3, y_2^i + hk_{23}) \\ y_1^{i+1} = y_1^i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ y_2^{i+1} = y_2^i + \frac{h}{6}(k_{21} + 2k_{22} + 2k_{23} + k_{24}) \end{cases}$$

Часть 3

Для решения данной задачи запишем заданное дифференциальное уравнение в узлах сетки и краевые условия:

$$\begin{cases} y_i'' + p_i y_i' + q_i y_i = f_x, x_i = a + i\frac{b-a}{n} \quad 0 \le i \le n \\ \sigma_1 y_0 + \gamma_1 y_0' = \delta_1 \\ \sigma_2 y_n + \gamma_2 y_n' = \delta_2 \end{cases}$$

Для $1 \le i \le n-1$ существует следующее разностное приближение для первой и второй производной и самой сеточной функции:

$$\begin{cases} y_i'' = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} \\ y_i' = \frac{y_{i+1} - y_{i-1}}{2h} \end{cases}$$

В результате подстановки этих разностных отношений в начальное уравнение в виде сеточной функции получим линейную систему из n+1 уравнений с n+1 неизвестными $y_0, y_1, ..., y_n$:

$$\begin{cases} y_i'' = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + p_i\frac{y_{i+1} - y_{i-1}}{2h} + q_i y_i = f_i \quad 1 \le i \le n-1 \\ \sigma_1 y_0 + \gamma_1\frac{y_0 - y_0}{h} = \delta_1 \\ \sigma_2 y_1 + \gamma_2\frac{y_n - y_{n-1}}{h} = \delta_2 \end{cases}$$

Явно выписав коэффициенты перед $y_0, y_1, ..., y_n$, получим систему с трехдиагональной матрицей:

$$A = \begin{bmatrix} \sigma_1 - \gamma_1/h & \gamma_1/h & 0 & 0 & ... & \delta_1 \\ 1 - h/2p_1 & q_1 * h^2 - 2 & 1 + h/2 * p_1 & 0 & ... & f_1 h^2 \\ 0 & 1 - h/2p_2 & q_2 * h^2 - 2 & 1 + h/2 * p_2 & ... & f_2 h^2 \\ 0 & 0 & 0 & ... & ... & f_k h^2 \\ 0 & 0 & 0 & ... & ... & f_l h^2 \\ 0 & ... & 1 - h/2p_{n-1} & q_{n-1} * h^2 - 2 & 1 + h/2 * p_{n-1} & f_{n-1} h^2 \\ 0 & ... & 0 & -\gamma_2/h & \sigma_2 + \gamma_2/h & \delta_2 \end{bmatrix}$$

Для решения полученной системы используется метод прогонки. Этот метод существенно упрощает решение системы с трехдиагональной матрицей и имеет сложность O(n):

Переобозначая коэффициенты перед y получим:

$$\begin{cases} C_0 y_0 + B_0 y_1 = F_0 \\ A_i y_{i-1} + C_i y_i + B_i y_{i+1} = F_i \quad 1 \le i \le n-1 \\ A_n y_{n-1} + C_n y_n = F_n \end{cases}$$

$$\begin{cases} \alpha_0 = -\frac{B_0}{C_0} \\ \beta_0 = \frac{F_0}{C_0} \\ \alpha_i = -\frac{B_i}{C_i + A_i \alpha_{i-1}} & \\ & 1 \le i \le n-1 \\ \beta_i = \frac{F_i - A_i \beta_{i-1}}{C_i + A_i \alpha_{i-1}} & \\ \\ y_n = \frac{F_n - A_n \beta_{n-1}}{C_n + A_n \alpha_{n-1}} \\ y_i = \beta_i + \alpha_i * y_{i+1} & 0 \le i \le n-1 \end{cases}$$

Описание программы

Рассмотрим ключевые функции программы:

Метод Рунге-Кутта для численного решения задачи Коши:

```
void double_runge_kutt(FILE *out, double a, double b,double init,
            double(*f)(double x, double y))
{
  double y = init;
  for (double x = a; x <= b; x += h){
    fprintf(out, "%f ", y);
    y = y + h/2 * (f(x, y) + f(x + h, y + f(x, y) * h));
  }
}

void square_runge_kutt(FILE *out, double a, double b, double init,
            double(*f)(double x, double y))
{
  double y = init;
  for (double x = a; x <= b; x += h){
    fprintf(out, "%f ", y);
    double k1 = f(x, y);
    double k2 = f(x + h/2, y + k1 * h /2);
    double k3 = f(x + h/2, y + k2 * h/ 2);
    double k4 = f(x + h, y + h * k3);
    y = y + h * (k1 + 2 * k2 + 2 * k3 + k4) / 6;
  }
}
```

Метод Рунге-Кутта для численного решения задачи Коши для системы:

```
void double_runge_kutt_system(FILE *out1, FILE *out2, double a, double b,
               double init1, double init2,
               double(*f1)(double x, double y1, double y2),
               double(*f2)(double x, double y1, double y2))
{
  double y1 = init1;
  double y2 = init2;
  for (double x = a; x <= b; x += h){
    fprintf(out1, "%f ", y1);
    fprintf(out2, "%f ", y2);
    double k1 = f1(x, y1, y2);
    double k21 = f2(x, y1, y2);
    double k2 = f1(x + h/2, y1 + k1 * h /2, y2 + k21 * h / 2);
    double k22 = f2(x + h/2, y1 + k1 * h /2, y2 + k21 * h / 2);

    y1 = y1 + h * k2;
    y2 = y2 + h * k22;
  }
}

void square_runge_kutt_system(FILE *out1, FILE *out2, double a, double b,
               double init1, double init2,
               double(*f1)(double x, double y1, double y2),
               double(*f2)(double x, double y1, double y2))
{
  double y1 = init1;
  double y2 = init2;
  for (double x = a; x <= b; x += h){
    fprintf(out1, "%f ", y1);
    fprintf(out2, "%f ", y2);
    double k1 = f1(x, y1, y2);
    double k21 = f2(x, y1, y2);

    double k2 = f1(x + h/2, y1 + k1 * h /2, y2 + k21 * h / 2);
    double k22 = f2(x + h/2, y1 + k1 * h /2, y2 + k21 * h / 2);

    double k3 = f1(x + h/2, y1 + k2 * h /2, y2 + k22 * h / 2);
    double k23 = f2(x + h/2, y1 + k2 * h /2, y2 + k22 * h / 2);

    double k4 = f1(x + h, y1 + k3 * h , y2 + k23 * h);
    double k24 = f2(x + h, y1 + k3 * h , y2 + k23 * h);

    y1 = y1 + h * (k1 + 2 * k2 + 2 * k3 + k4) / 6;
    y2 = y2 + h * (k21 + 2 * k22 + 2 * k23 + k24) / 6;
  }
}
```

Решение краевой задачи:

```c

double *tridiag(double **m, double *f, int n) //метод прогонки для трехдиагональной матрицы
{
  double EPS = 1e-9;
  double *res = calloc(n, sizeof(*res)), *a(..), *b(...), *c(..), *alpha(..), *beta(..);
  for (int i = 0; i < n; ++i) {
    a[i] = (i > 0 ? m[i][i - 1] : 0);
    b[i] = (i < n - 1 ? m[i][i + 1] : 0);
    c[i] = m[i][i];
  }
  if (fabs(c[0]) < EPS) {
    exit(1);
  }
  alpha[0] = -b[0] / c[0];
  beta[0] = f[0] / c[0];
  for (int i = 1; i < n; ++i) {
    if (fabs(c[i] + a[i] * alpha[i - 1]) < EPS) {
      exit(1);
    }
    alpha[i] = -b[i] / (c[i] + a[i] * alpha[i - 1]);
  }

  for (int i = 0; i < n; ++i) {
    beta[i] = (f[i] - a[i] * beta[i - 1]) /(c[i] + a[i] * alpha[i - 1]);
  }
  res[n - 1] = beta[n - 1] + alpha[n - 1];
  for (int i = n - 2; i >= 0; --i) {
    res[i] = beta[i] + alpha[i] * res[i + 1];
  }
  return res;
}

//k[0]y(0) + k[1]y'(0) =k[2], k[3]y(1) + k[4]y'(1) =k[5] - краевые условия
void boundary_problem(FILE *out, double a, double b,
          double (*p)(double x), double (*q)(double x),
          double (*f)(double x), double *k)

{
  int size = (b-a) / h + 1;
  double **m = calloc(size + 1, sizeof(*m));
  for (int i = 0; i < size + 1; i++){
    m[i] = calloc(size + 1, sizeof(*m[i]));
  }
  double *right = calloc(size + 1, sizeof(*right));
 m[0][0] = k[0] - k[1] / h;
 m[0][1] = k[1] / h;
 m[size][size-1] = - k[4] / h;
 m[size][size] = k[3] + k[4] / h;
  right[size] = k[5];
  right[0] = k[2];
  int j = 1;
  for (double i = a + h; i < b; i += h, j++) {
    m[j][j-1] = 1 - 0.5 * h * p(i);
    m[j][j] = q(i) * h * h - 2;
    m[j][j+1] = 1 + 0.5 * h * p(i);
    right[j] = f(i) * h * h;
  }

  double *sol = tridiag(m, right, size + 1);
  for (int i = 0; i <= size; i++) {
    fprintf(out, "%f\n", sol[i]);
  }
}
```

В данном разделе содержатся примеры реализации функций для тестирования, использующихся в программе.

- Функция из задачи Коши для Оду и точное решение

```
double input1 (double x, double y){
  return sin(x) - y;
}
double ans1(double x){
  return -0.5*cos(x) + 0.5*sin(x) +10.5 * exp(-x);
}
```

- Запись результатов выполнения в файл для построения графика и проверки корректности метода

```
void test(FILE *o1, FILE *o2, FILE *o3, double init, //Отрезок везде [0,1]
      double (*f)(double x), double (*ans)(double x))
{
  double_runge_kutt(o1, 0, 1, init, f);
  square_runge_kutt(o2, 0, 1, init, f);
  for (double x = 0; x <= 1; x += h){
    fprintf(o3, "%f ", ans(x));
  }
}

int main()
{ ...\\
  FILE *output1 = fopen("out1.txt", "w");
  FILE *output2 = fopen("out2.txt", "w");
  FILE *output3 = fopen("out3.txt", "w");
  test(output1, output2, output3, 10, input1, ans1);
}
```

- Функции из задачи Коши для системы ОДУ и точное решение

```
double sys13(double x, double y1, double y2)
{
  return (y1-y2) / x;

}

double sys23(double x, double y1, double y2)
{
  return (y1 + y2) / x;
}

double prec1(double x)
{
  return x * (cos(log(x)) - sin(log(x)));
}

double prec2(double x)
{
  return x * (cos(log(x)) + sin(log(x)));
}
```

- Запись результатов выполнения в файл для построения графика и проверки корректности метода

```
int main()
{ ...\\
  FILE *s1 = fopen("1.txt", "w");
  FILE *s2 = fopen("2.txt", "w");
  FILE *s3 = fopen("3.txt", "w");
  FILE *s4 = fopen("4.txt", "w");
  FILE *p1 = fopen("p1.txt", "w");
  FILE *p2 = fopen("p2.txt", "w");
  double_runge_kutt_system(s1, s2, 1, 10, 1, 1, sys13, sys23);
  square_runge_kutt_system(s3, s4, 1, 10, 1, 1, sys13, sys23);
  for (x = 1; x <= 10; x += h){
    fprintf(p1, "%f ", prec1(x));
  }
  for (x = 1; x <= 10; x += h){
    fprintf(p2, "%f ", prec2(x));
  }
}
```

- Функции из краевой задачи и точное решение

```
double p(double x)
{
  return 2;
}

double q(double x)
{
  return 1;
}

double f(double x)
{
  return 1;
}

double sol(double x)
{
  return 1 - x*exp(1-x);
}
```

- Запись результатов выполнения в файл для построения графика и проверки корректности метода

```
int main()
{ ...\\
   FILE *output_b = fopen("out_b.txt", "w");
  k[0] = 1;
  k[1] = 0;
  k[2] = 1;
  k[3] = 1;
  k[4] = 1;
  k[5] = 0;
  FILE *output_b1 = fopen("out_b1.txt", "w");
  boundary_problem(output_b, 0, 1, p, q, f, k);
  for (x = 0; x <= 1; x += h){
    fprintf(output_b1, "%f ", sol(x));
  }
}
```

- Секция Python

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import math

X = np.loadtxt("x.txt")
Y_d = np.loadtxt("out1.txt")
Y_sq = np.loadtxt("out2.txt")
Y_ex = np.loadtxt("out3.txt")

fig = plt.figure(figsize = (8, 4))
ax = plt.subplot(111)

a = ["double runge", "square runge", "precise"] #пример для построения графика решения задачи Коши для ОДУ, остальные - аналогично
ax.plot(X, Y_d,'g', X, Y_sq, 'b^', X, Y_ex, 'r--')
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.legend(a)
ax.grid()
```

Тестирование

Для проведения тестирования построим графики полученных численных решений в программе и точных решений из wolframalpha.com при помощи программы на языке Python.

- Часть 1

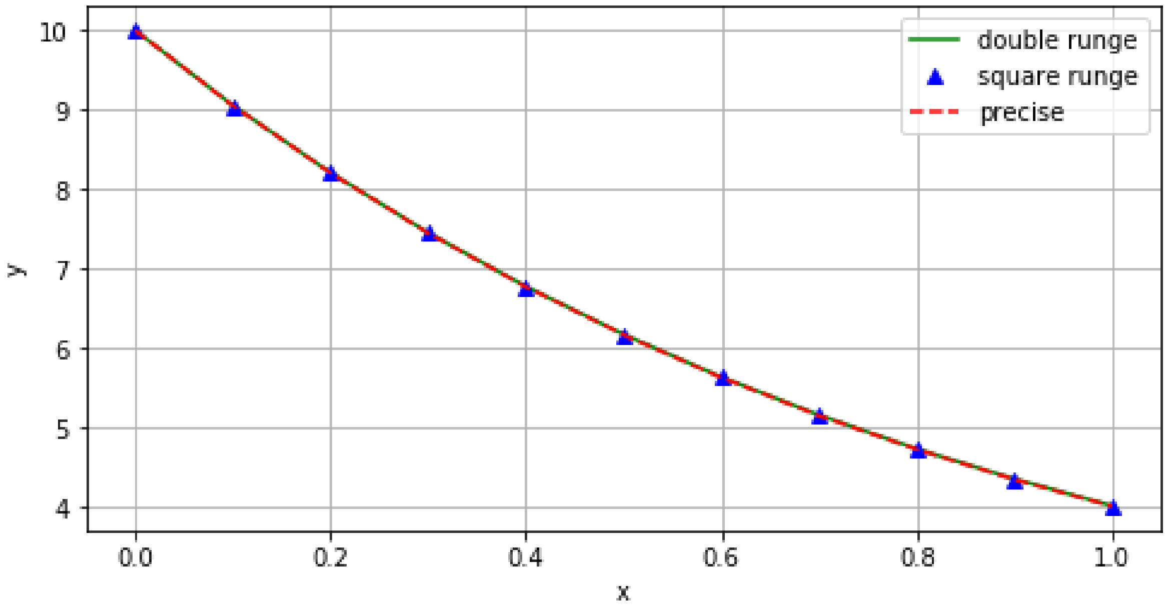
  Будем проводить тестирование на задачах Коши из таблицы 1.

$$\begin{cases} y' = sin(x) - y \\ x_0 = 0 \\ y_0 = 10 \end{cases}$$

Точное решение:

$$y = -0.5cosx + 0.5sinx + 10.5e^{-x}$$

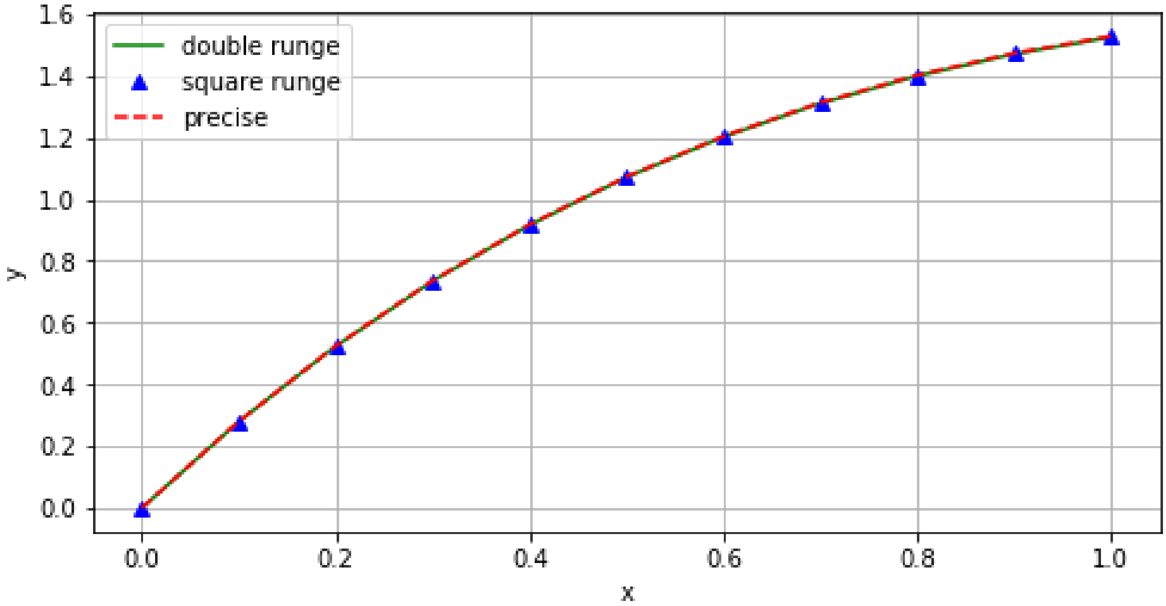
В этом примере хорошая (невидимая на графике) точность достигается уже при 10 шагах сетки:



$$\begin{cases} y' = 3 - y - x \\ x_0 = 0 \\ y_0 = 0 \end{cases}$$

Точное решение:

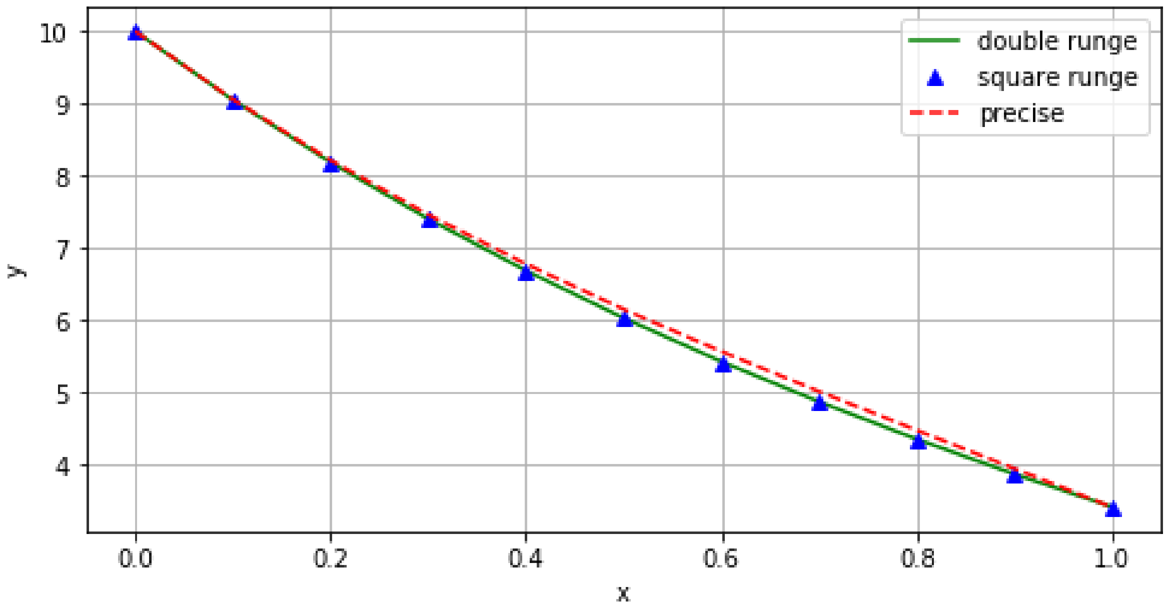
$$y = 4 - x - 4e^{-x}$$



$$\begin{cases} y' = -x^2 - y \\ x_0 = 0 \\ y_0 = 10 \end{cases}$$

Точное решение:

$$y = -x^3 + 2x - 2 + 12e^{-x}$$



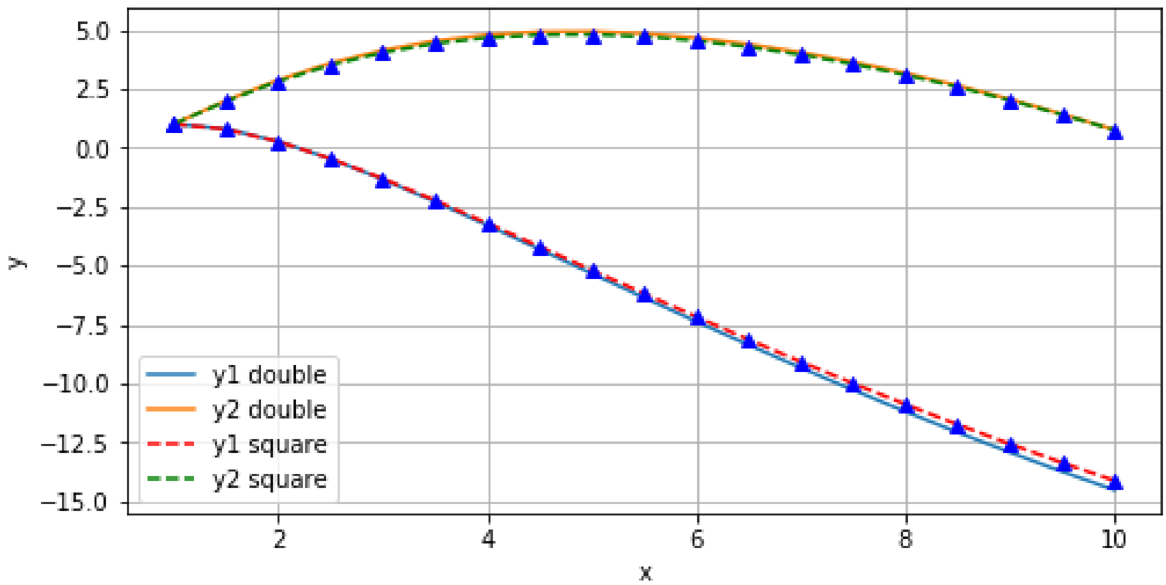
- Часть 2

Будем проводить тестирование на задачах Коши из таблицы 2.

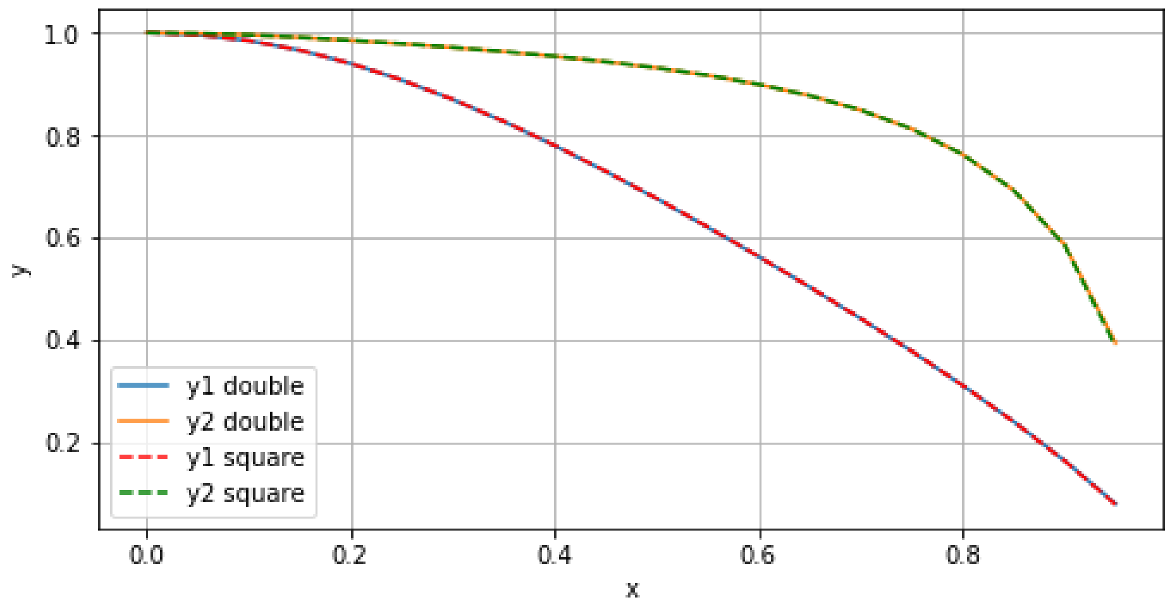
$$\begin{cases} u' = \frac{u-v}{x} \\ v' = \frac{u+v}{x} \\ x_0 = 1 \\ u_0 = 1 \\ v_0 = 1 \end{cases}$$

Точное решение:

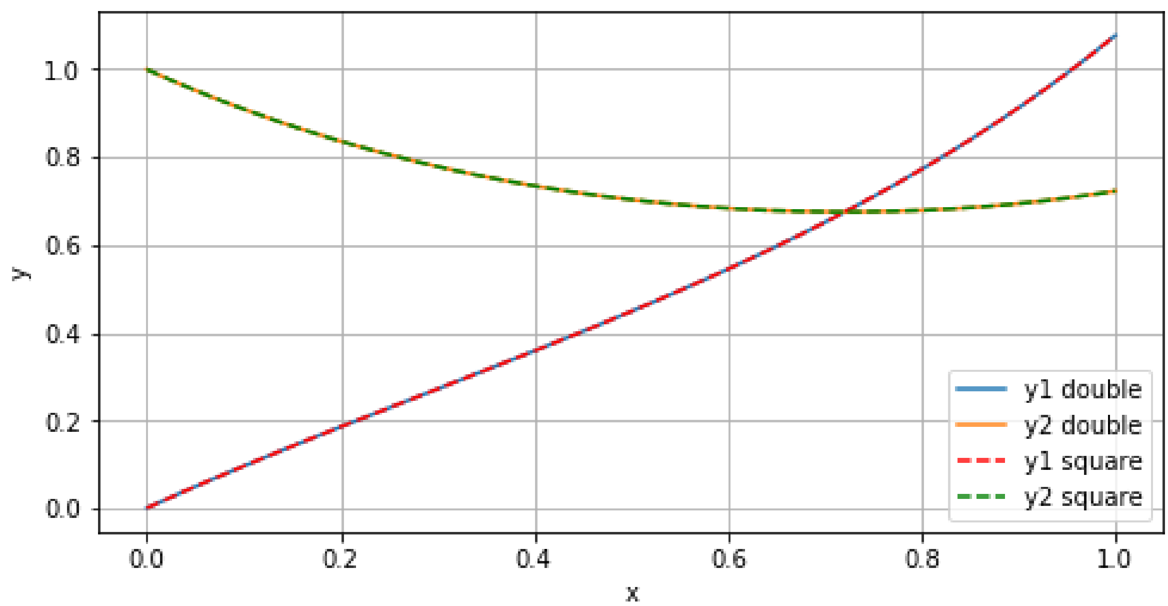
$$u_{precise}(x) = x(cos(ln(x)) - sin(ln(x)))v_{precise}(x) = x(cos(ln(x)) + sin(ln(x)))$$



$$\begin{cases} u' = -2 * x * u^2 + v^2 - x - 1 \\ v' = \frac{1}{v^2} - u - \frac{x}{u} \\ x_0 = 0 \\ u_0 = 1 \\ v_0 = 1 \end{cases}$$



$$\begin{cases} u' = x * u + v \\ v' = u - v \\ x_0 = 0 \\ u_0 = 0 \\ v_0 = 1 \end{cases}$$

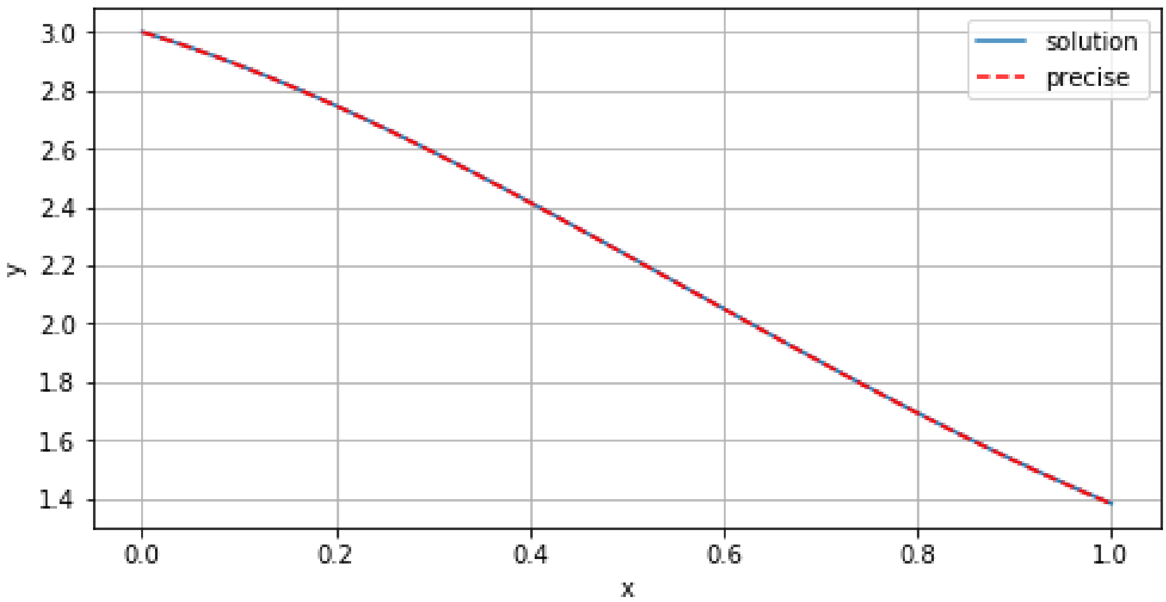


- Часть 3

$$\begin{cases} y'' + y = 4 * sin(x) \\ y_0 + y_0' = 2 \\ y_1 + y_1' = 0 \end{cases}$$

Точное решение:

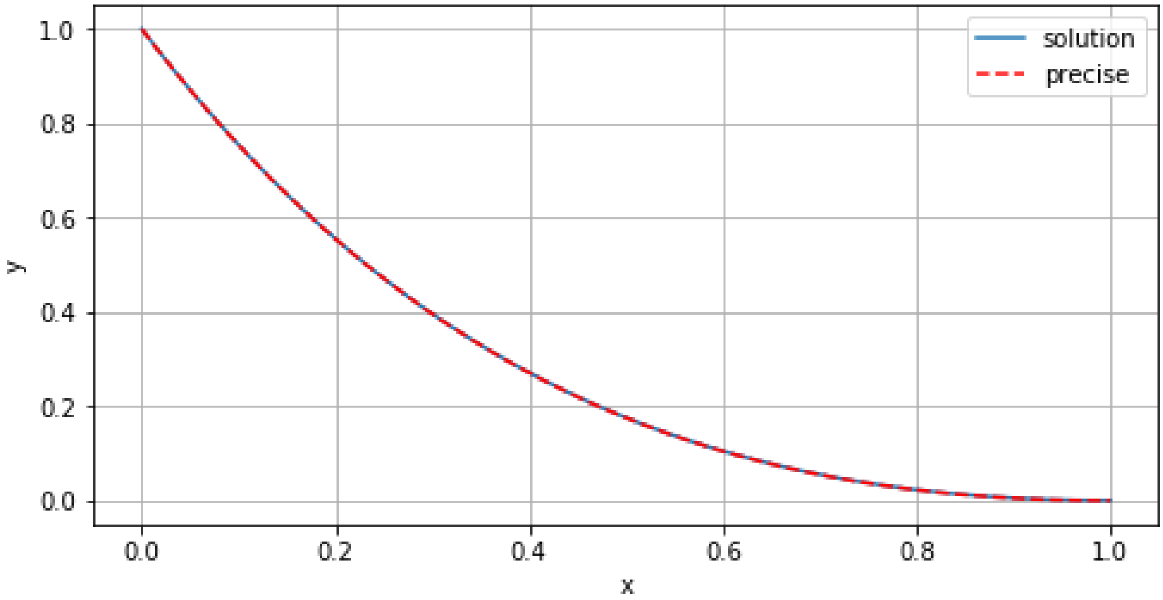
$$y = sin(x) + (3 - 2x)cos(x)$$



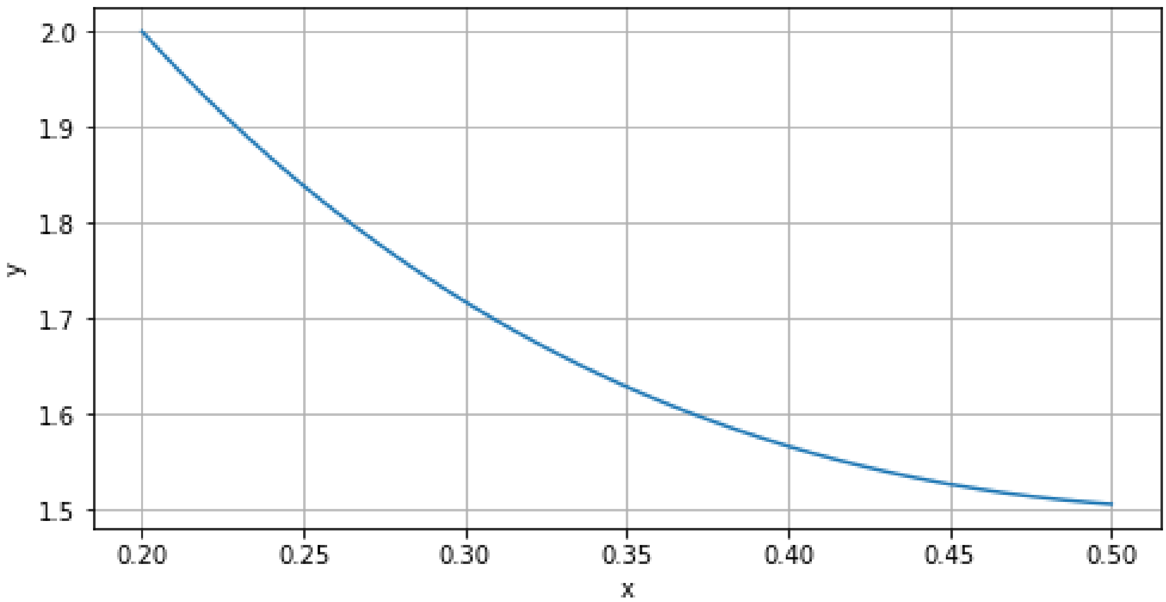
$$\begin{cases} y'' + 2y' + y = 1 \\ y_0 = 1 \\ y_1 + y_1' = 0 \end{cases}$$

Точное решение:

$$y = 1 - xexp1 - x$$



$$\begin{cases} y'' + 2y' - \frac{y}{x} = 3 \\ y_{0.2} = 2 \\ 0.5y_{0.5} - y_{0.5}' = 1 \end{cases}$$



Выводы

Тестирование показывает, что приведённые выше методы Рунге-Кутта численного решения решения обыкновенных дифференциальных уравнений (а также систем ОДУ), разрешенных относительно производной, имеют высокую точность даже при большом шаге сетки. Одновременно с этим его важным преимуществом является простота реализации.
С уменьшением шага сетки решение краевой задачи Коши с использованием метода прогонки для систем с трёхдиагональной матрицей тоже дает точный результат, мало отличимый от истинного.