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 :

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

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:

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

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
- $x_i = a + h * i, y_i = y(x_i), 0 \le i \le n$ - 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:

- `Matrix.hpp/Matrix.cpp`

  Вычисление определителя:

```

int triangle(double **a, int size) //приведение матрицы к верхнетругольной форме
{
  int j = 0;
  int sign = 1;
  for (int i = 0; i < size; i++){
    // Поиск первого ненулевого элемента в i столбце начиная с i+1 столбца
    for (j = i; j < size; j++){
      if (a[j][i]){
        break;
      }
    }
    if (j == size || fabs(a[j][i]) < EPS){
      fprintf(stderr,"det 0\n");
      exit(1);
    }
    if (i != j ){
      sign *= change_str(a, i, j); // фя- меняет местами строки и возвращает -1
    }
    for (int k = i + 1; k < size; k++){
      double k1 = a[i][i];
      double k2 = a[k][i];
      if (fabs(k1) < EPS){
        exit(1);
      }
      if (fabs(k2) < EPS){
        continue;
      }
      for (int l = 0; l < size ; l++){
        a[k][l] -= a[i][l] * k2 / k1;
      }
    }
  }
  return sign;
}

double det(double **a1, int size)
{
  double **a = copy_matrix(a1, size);
  int sign = triangle(a, size);
  double ans = 1;
  for (int i = 0; i < size; i++){
    ans *= a[i][i];
  }
  return ans * sign;
}
```

Вычисление обратной матрицы:

```
double **inverse(double **a1, int size)
{
  double **a = copy_matrix(a1, size);
  double **res = calloc(size, sizeof(*res));
  for (int i = 0; i < size; i++){
    res[i] = calloc(size, sizeof(*res[i]));
  }
  for (int i = 0; i < size; i++){
    res[i][i] = 1;
  }
  int j = 0;
  int sign = 1;
  for (int i = 0; i < size; i++){
    // Поиск первого ненулевого элемента в i столбце начиная с i+1 столбца
    for (j = i; j < size; j++){
      if (a[j][i]){
        break;
      }
    }
    if (j == size || fabs(a[j][i]) < EPS){
      fprintf(stderr,"det 0\n");
      exit(1);
    }
    if (i != j){
      sign *= change_str(a, i, j);
      change_str(res, i, j);
    }
    for (int k = i + 1; k < size; k++){
      double k1 = a[i][i];
      double k2 = a[k][i];
      if (fabs(k2) < EPS){
        continue;
      }
      for (int l = 0; l < size ; l++){
        a[k][l] -= a[i][l] * k2 / k1;
        res[k][l] -= res[i][l] * k2 / k1;
      }
    }
  }
  for (int i = size - 1; i > 0; i--){
    for (int j = i - 1; j >= 0; j--){
      if (fabs(a[i][i]) < EPS){
        fprintf(stderr,"det 0\n");
        exit(1);
      }
      double k1 = a[j][i] / a[i][i];
      for (int k = size - 1; k >=0; k--){
        a[j][k] -= k1 * a[i][k];
        res[j][k] -= k1 * res[i][k];
      }
    }
  }
  for (int i = 0; i < size; i++){
    for (int j = 0; j < size; j++){
      if (fabs(a[i][i]) < EPS){
        fprintf(stderr,"det 0\n");
        exit(1);
      }
      res[i][j] /= a[i][i];
    }
  }
  return res;
}
```

Стандартный метод Гаусса

```c
double *gauss(double **a1, double *f1, int size)
{
  double **a = copy_matrix(a1, size);
  double *f = calloc(size, sizeof(*f));
  for (int i = 0; i < size; i++){
    f[i] =f1[i];
  }
  //прямой ход
  int j = 0;
  int sign = 1;
  for (int i = 0; i < size; i++){
    // Поиск первого ненулевого элемента в i столбце начиная с i+1 столбца
    for (j = i; j < size; j++){
      if (a[j][i]){
        break;
      }
    }
    if (j == size || fabs(a[j][i]) < EPS){
      fprintf(stderr,"det 0\n");
      exit(1);
    }
    if (i != j){
      sign *= change_str(a, i, j);
      double tmp = f[i];
      f[i] = f[j];
      f[j] = tmp;
    }
    for (int k = i + 1; k < size; k++){
      double k1 = a[i][i];
      double k2 = a[k][i];
      if (fabs(k2) < EPS){
        continue;
      }
      f[k] -= f[i] * k2 / k1;
      for (int l = 0; l < size ; l++){
        a[k][l] -= a[i][l] * k2 / k1;
      }
    }

  }
  //обратный ход
  for (int i = size - 1; i > 0; i--){
    for (int j = i - 1; j >= 0; j--){
      if (fabs(a[i][i]) < EPS){
        fprintf(stderr,"det 0\n");
        exit(1);
      }
      double k1 = a[j][i] / a[i][i];
      f[j] -= k1 * f[i];
      for (int k = size - 1; k >=0; k--){
        a[j][k] -= k1 * a[i][k];
      }
    }
  }
  for (int j = 0; j < size; j++){
    if (fabs(a[j][j]) < EPS){
      fprintf(stderr,"det 0\n");
      exit(1);
    }
    f[j] /= a[j][j];
  }
  return f;
}
```

Метод Гаусса с выбором ведущего элемента.

```c
double *gauss_main(double **a1, double *f1, int size)
{
  double **a = copy_matrix(a1, size);
  double *f = calloc(size, sizeof(*f));
  //хранит перестановки столбцов
  int *vec = calloc(size, sizeof(*vec));

  for (int i = 0; i < size; i++){
    f[i] =f1[i];
  }
  //прямой ход
  int j = 0;
  for (int i = 0; i < size; i++){
    vec[i] = i;
  }
  for (int i = 0; i < size; i++){
    // Поиск первого ненулевого элемента в i столбце начиная с i+1 столбца
    double max = 0;
    int idx = 0;
    for (j = i; j < size; j++){
      if (fabs(a[i][j]) > max){
        max = fabs(a[i][j]);
        idx = j;
      }
    }
    double tmp = vec[i];
    vec[i] = vec[idx];
    vec[idx] = tmp;
    for (int l = 0; l < size; l++){
      double tmp = a[l][i];
      a[l][i] = a[l][idx];
      a[l][idx] = tmp;
    }

    for (int k = i + 1; k < size; k++){
      double k1 = a[i][i];
      double k2 = a[k][i];
      f[k] -= f[i] * k2 / k1;
      for (int l = i; l < size ; l++){
        a[k][l] -= a[i][l] * k2 / k1;
      }
    }

  }
  //обратный ход
  for (int i = size - 1; i > 0; i--){
    for (int j = i - 1; j >= 0; j--){
      double k1 = a[j][i] / a[i][i];
      f[j] -= k1 * f[i];
      for (int k = size - 1; k >=0; k--){
        a[j][k] -= k1 * a[i][k];
      }
    }
  }
  for (int j = 0; j < size; j++){
    f[j] /= a[j][j];
  }
  double *ans = calloc(size, sizeof(*ans));
  for (int i = 0; i < size; i++){
    ans[vec[i]] = f[i];
  }
  return ans;
}
```

Метод верхней релаксации

```c
double *relax(double **a, double *f, double w, int size, int max_iter, double solution_eps)
{
  //критерий остановки - максимальное число итераций
  double *x = calloc(size, sizeof(*x));
  double *tmp = calloc(size, sizeof(*tmp));
  for (int k = 0; k < max_iter; k++){
    for (int i = 0; i < size; i++){
      double sub1 = 0, sub2 = 0;
      for (int j = 0; j < i; j++){
        sub1 += a[i][j] * tmp[j];
      }
      for (int j = i; j < size; j++){
        sub2 += a[i][j] * x[j];
      }
      tmp[i] = x[i] + w / a[i][i] * (f[i] - sub1 - sub2);

      //Проверяем 2 критерий остановки алгоритма - расстояние между векторами решениями
      на 2 последовательных шагах становится меньше заданного значения - solution_eps
    }
    double dist = 0;
    for (int i = 0; i < size; i++){
      dist += (x[i] -tmp[i]) * (x[i] -tmp[i]);
    }
    if (sqrt(dist) < solution_eps){
      return tmp;
    }
    for (int i = 0; i < size; i++){
      x[i] = tmp[i];
    }
  }
  return x;
}
```

Число обусловленности

```c
double cond_num(double **a, int n)
{
  double res1 = 0;
  for (int i = 0; i < n; i++){
    for (int j = 0; j < n; j++){
      if(fabs(a[i][j]) > res1){
        res1 = fabs(a[i][j]);
      }
    }
  }
  double res2 = 0;
  double **b = inverse(a, n);
  for (int i = 0; i < n; i++){
    for (int j = 0; j < n; j++){
      if(fabs(b[i][j]) > res2){
        res2 = fabs(b[i][j]);
      }
    }
  }
  return res1 * res2;
}
```

В данном разделе содержится реализация дополнительных функций, использующихся в программе.
В частности, реализованы классы:

– Вывод на стандратный поток матрицы

```
void print_matrix(double **a, int size);
```

– Ввод матрицы со стандратного потока

```
void scan_matrix(double **a, int size);
```

– Вывод на стандратный поток матрицы-столбца

```
void print_vect(double *a, int size);
```

– Ввод матрицы-столбца со стандратного потока

```
void scan_vect(double *a, int size);
```

Тестирование

- Часть 1

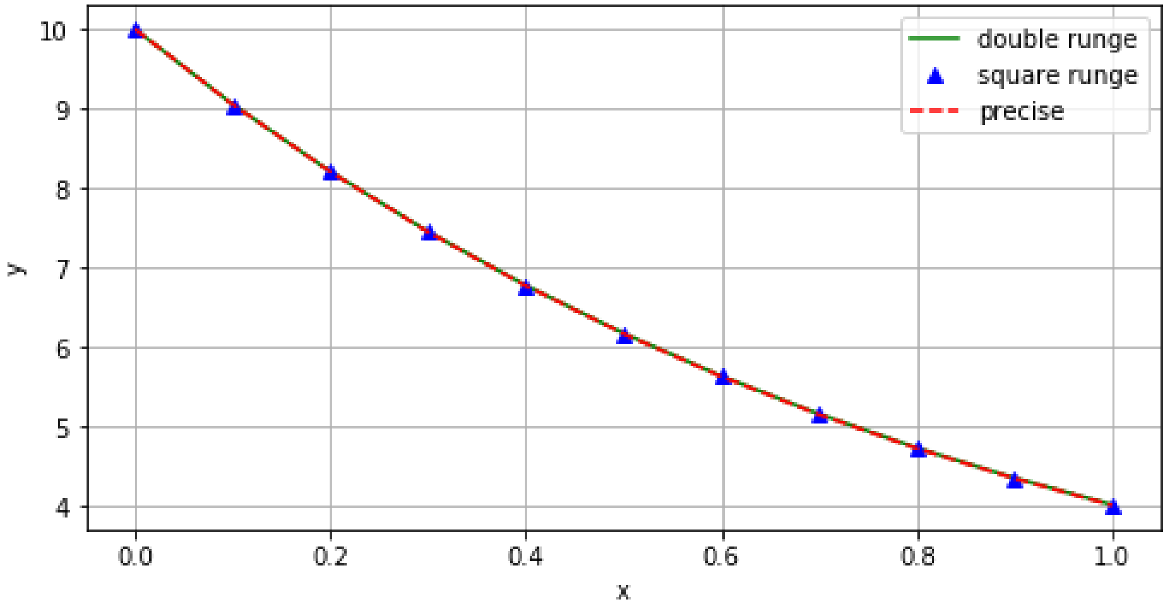
Будем проводить тестирование на задачах Коши из таблицы 1.

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

Точное решение:

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

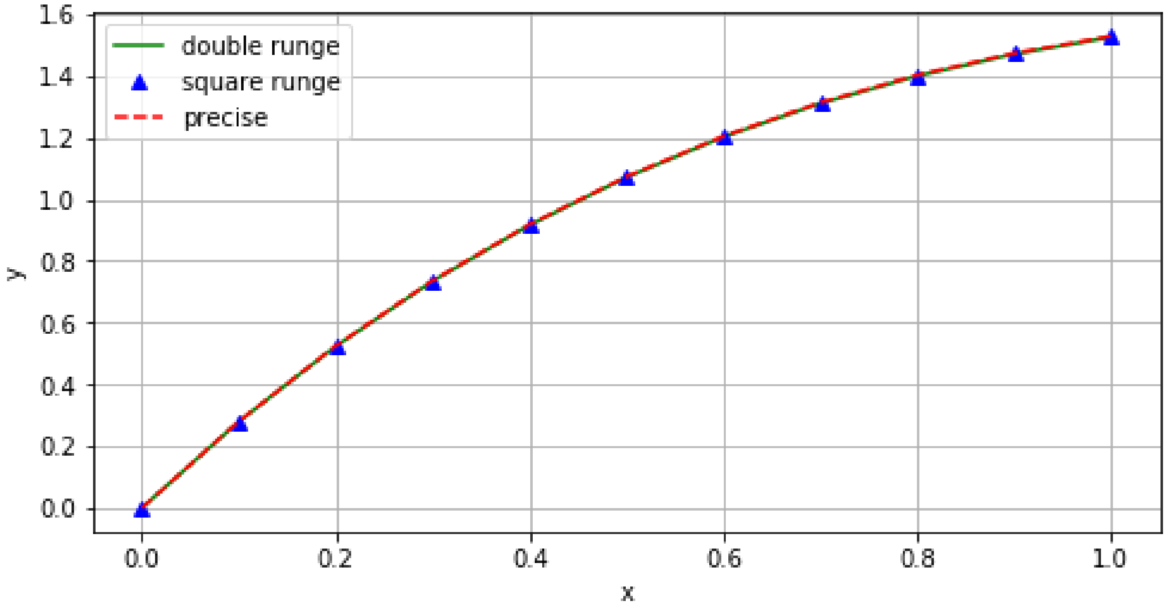
В этом примере хорошая (невидимая на графике) точность достигается уже при 10 шагах сетки:



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

Точное решение:

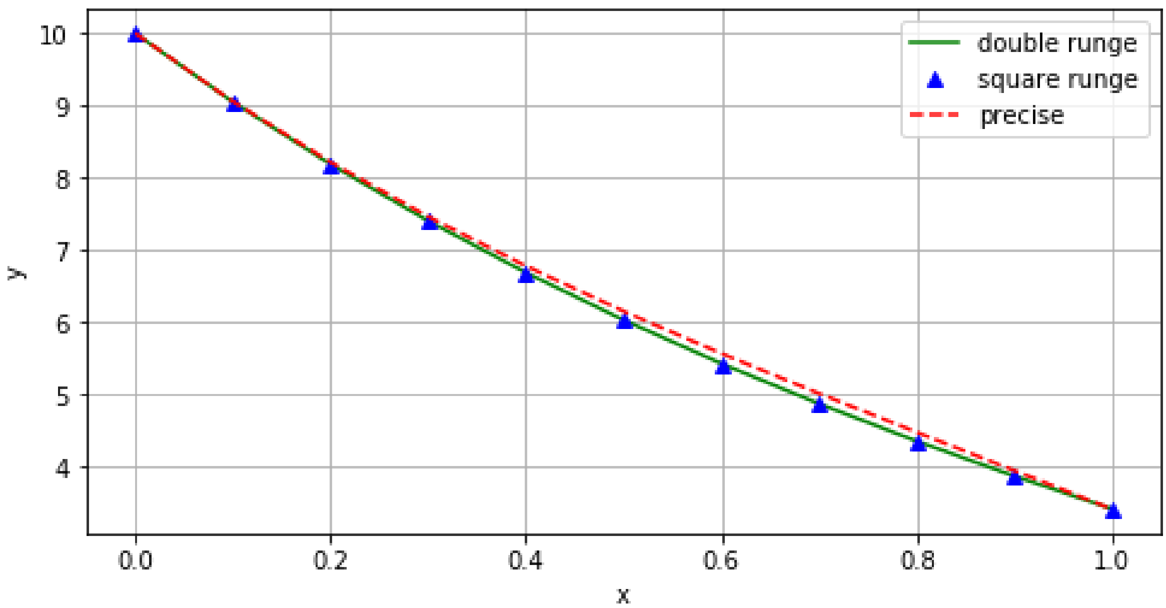
$$y = 4 - x - 4e^{-x}$$



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

Точное решение:

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



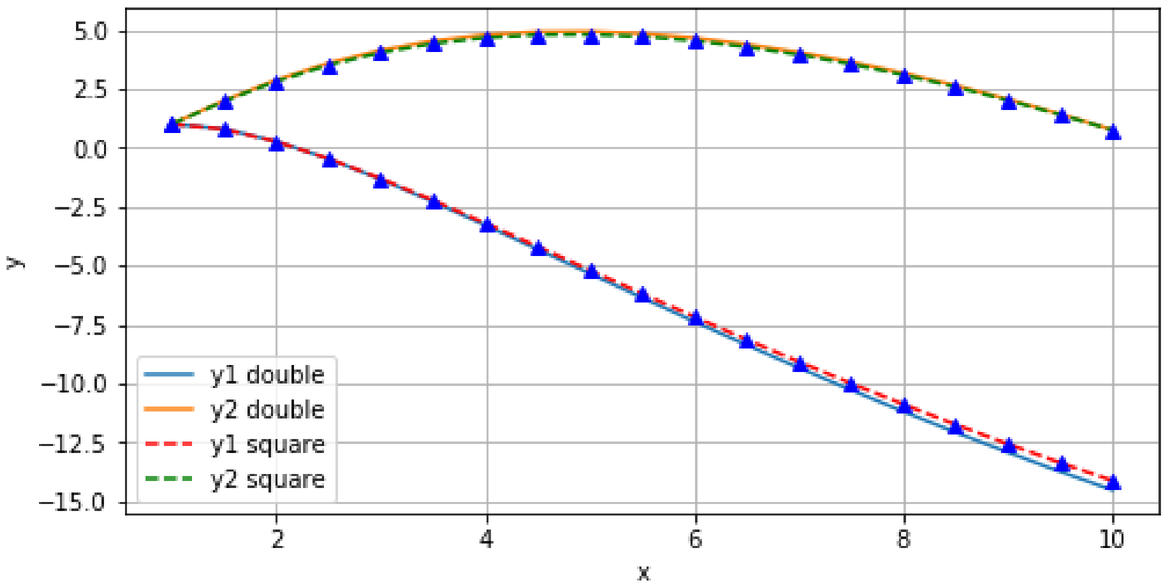
- Часть 2

Будем проводить тестирование на задачах Коши из таблицы 2.

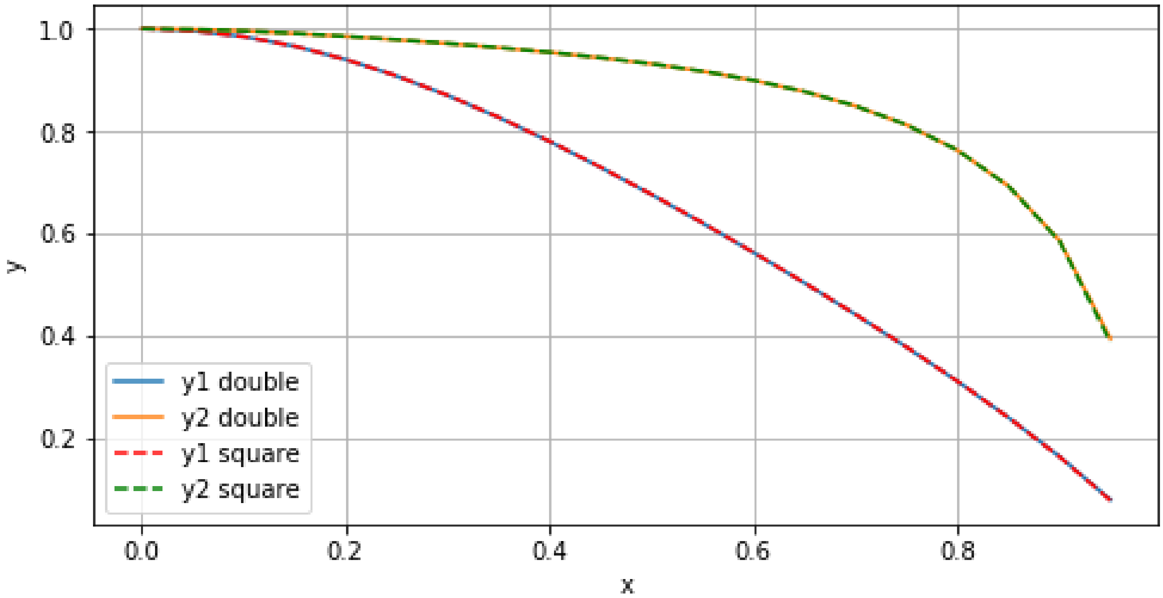
$$\begin{cases} u' = \frac{u-v}{x} \\ v' = \frac{u+v}{x} \\ x_0 = 1 \\ u_0 = 1 \\ v_0 = 1 \end{cases}$$

Точное решение:

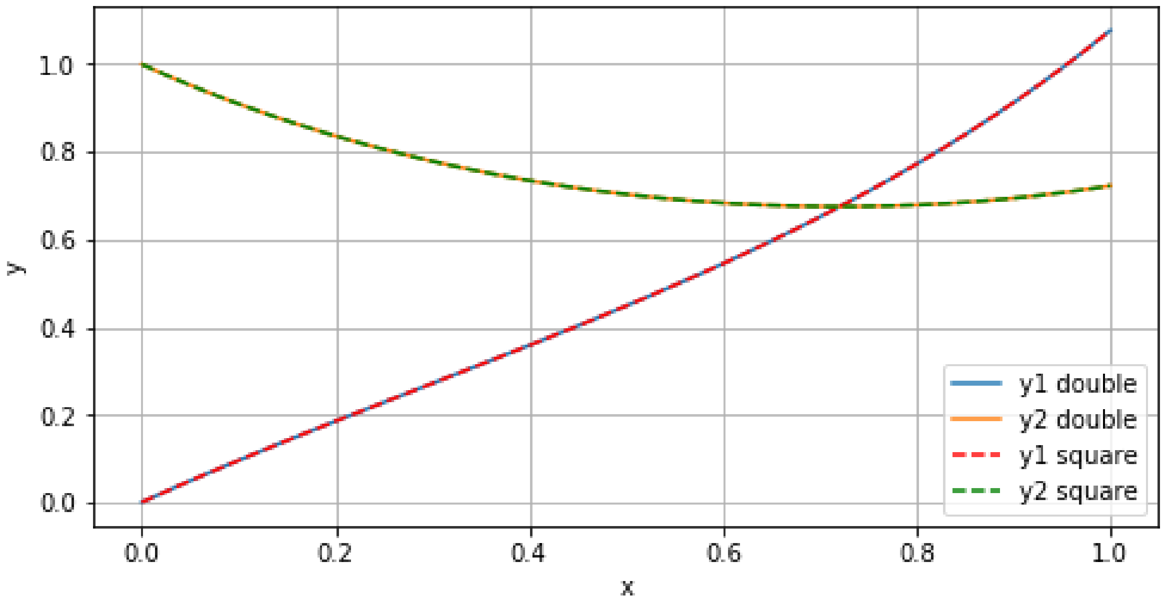
$$u_{precise}(x) = x(cos(ln(x)) - sin(ln(x)))v_{precise}(x) = x(cos(ln(x)) + sin(ln(x)))$$



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



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

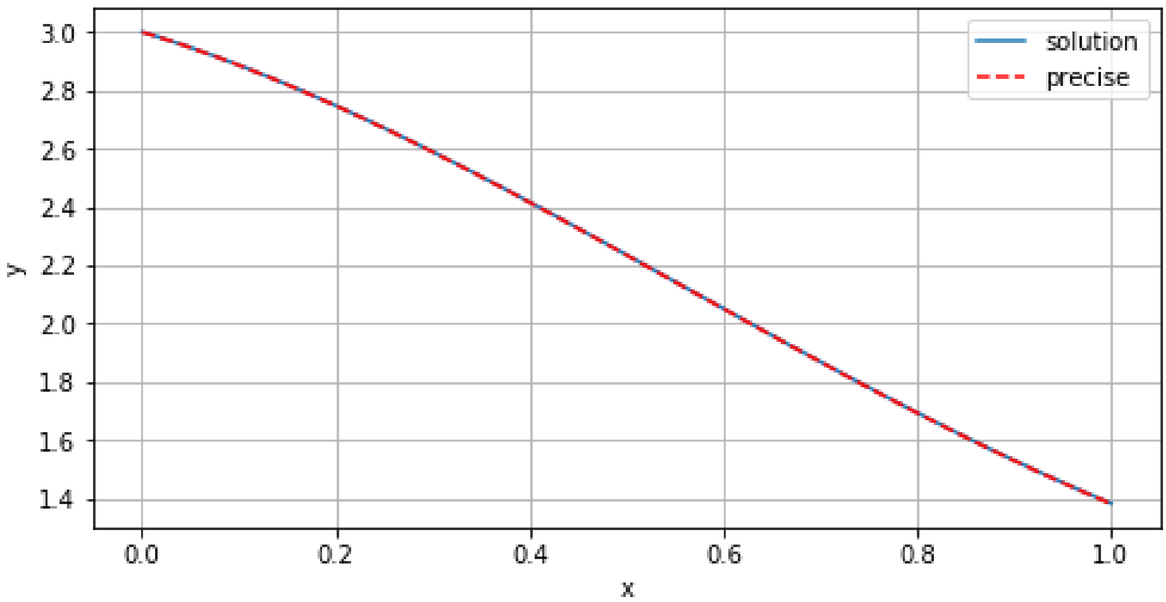


- Часть 3

$$\begin{cases} y'' + y = 4 * sin(x) \\ y_0 + y'_0 = 2 \\ y_1 + y'_1 = 0 \end{cases}$$

Точное решение:

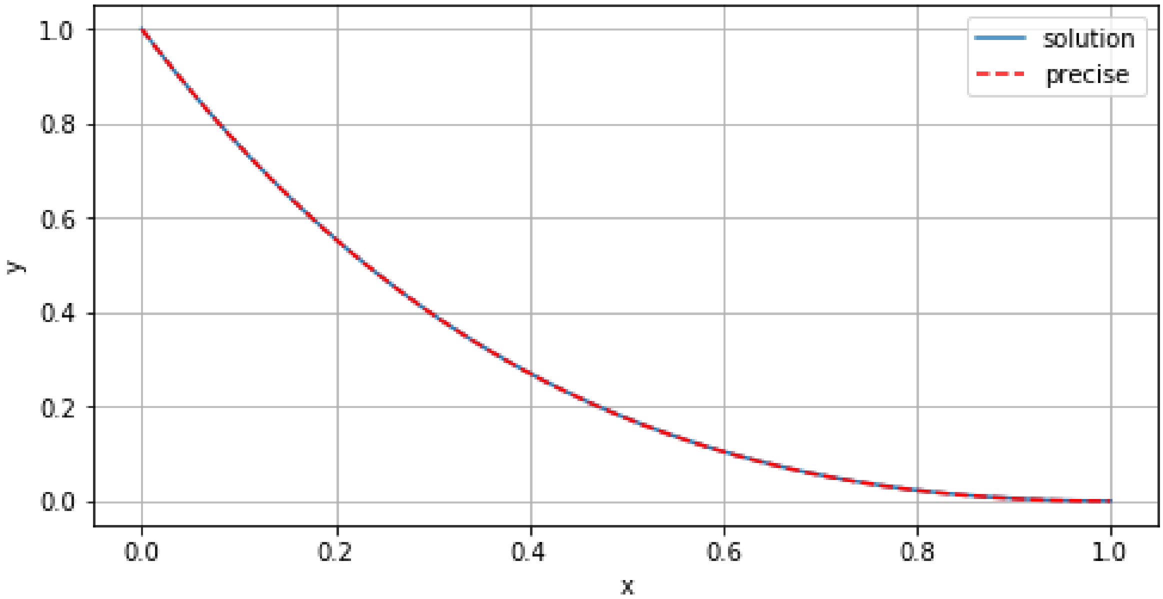
$$y = sin(x) + (3 - 2x)cos(x)$$



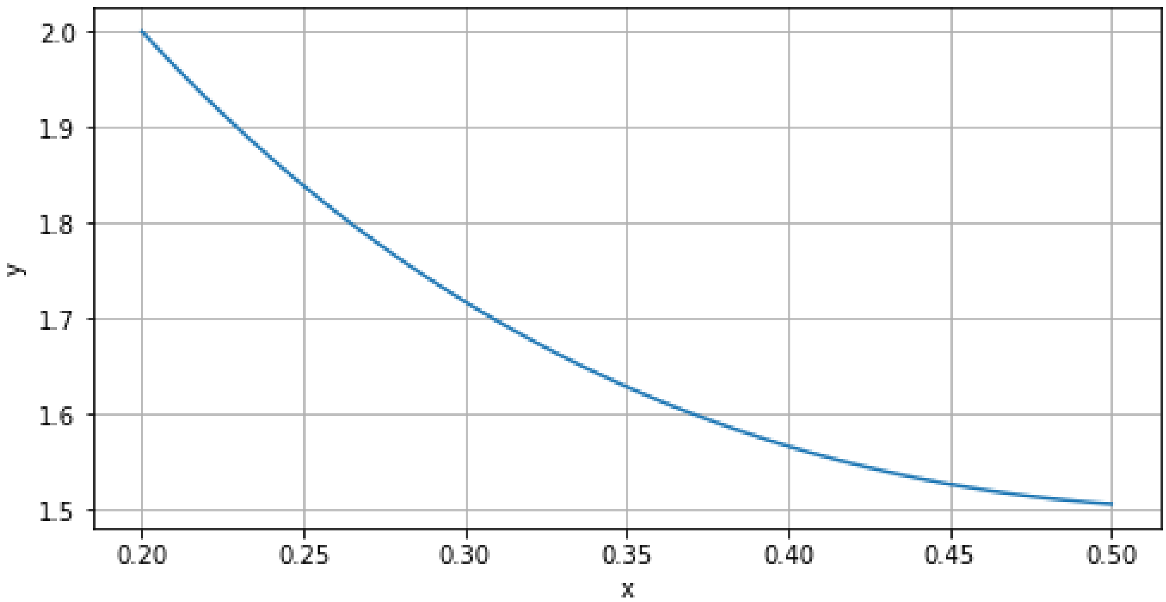
$$\begin{cases} y'' + 2y' + y = 1 \\ y_0 = 1 \\ y_1 + y_1' = 0 \end{cases}$$

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

- Часть 1

  Из теоретического материала следует, что метод Гаусса с выбором главного элемента имеет более высокую точность, однако на примерах, рассмотренных в ходе проведения тестирования, оба метода сходятся и значимое различие в точности результатов не обнаруживается. Вместе с этим методы одновременно трубуют невырожденности матрицы А.Таким образом, можем сказать, что для подобных задач (размер матрицы не велик, сами элементы не являются большими в абсолютном значении) значительной разницы в применении методов не будет.

- Часть 2

  В случае симметрической положительно определенной матрицы метод верхней релаксации будет сходится очень быстро, особенно при $\omega = 1$ (Метод Зейделя). В этом случае для получения решения точности порядка 10^{-6} метод верхней релаксации потребует порядка $10 - 30$ итераций в зависимости от ω, что показывает превосходство данного метода над методом Гаусса в плане скорости. Однако для быстрой сходимости он накладывает на матрицу и много ограничений: матрица А должна быть симетрической, положительно определённой, с небольшим числом обусловленности (близким к 1).