Московский Государственный Университет
имени М. В. Ломоносова

Компьютерный практикум по учебному курсу «ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ» ЗАДАНИЕ № 2

ОТЧЁТ
о выполненном задании

студента 206 учебной группы факультета ВМК МГУ
Маркова Игоря Валерьевича

г. Москва, 2019 год

ПРАКТИЧЕСКАЯ РАБОТА № 2

Подвариант № 1

Решение задачи Коши для дифференциального уравнения первого порядка и 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:

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

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

$$y(x_0) = y_0$$

Предполагается, что правая часть уравнения такова, что гарантирует существование и единственность решения задачи Коши. В том случае, если рассматривается не одно дифференциальное уравнение, а система обыкновенных дифференциальных уравнений первого порядка, разрешенных относительно производных неизвестных функций, то соответствующая задача Коши имеет вид (на примере двух дифференциальных уравнений):

$$\frac{dy_1}{dx} = f_1(x, y_1, y_2),\ x > x_0$$

$$\frac{dy_2}{dx} = f_2(x, y_1, y_2),\ x > x_0$$

Начальные условия задаются в точке $x = x_0$:

$$y_1(x_0) = y_{01},\ y_2(x_0) = y_{02}$$

Предполагается, что правые части уравнений заданы так, что это гарантирует существование и единственность решения задачи Коши для системы обыкновенных дифференциальных уравнений первого порядка в форме,разрешенной относительно производных неизвестных функций.

Задачи практической работы

1. Решить задачу Коши 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), просчитать численно;
2. Найти численное решение задачи и построить его график;
3. Найденное численное решение сравнить с точным решением дифференциального уравнения.

Метод Рунге-Кутта

Предположим, что решение уравнение имеет производные достаточно высокого порядка, разложим решение дифференциального уравнения по формуле Тейлора и получим разностное уравнение:

$$\frac{y_{i+1}-y_i}{h} = f(x_i, y_i) + \frac{h}{2}\left[\frac{\partial f(x_i,y_i)}{\partial x} + \frac{\partial f(x_i,y_i)}{\partial y} f(x_i, y_i)\right]$$

Метод Рунге-Кутта заключается в том, чтобы приближенно заменить правую часть на сумму значений функции f в двух разных точках с точностью до членов

порядка h^2 . Таким образом, получим разностные схемы Рунге-Кутта:

Наиболее удобной для вычисления разностной схемой этого семейства соответствует значение параметра $\alpha = \frac{1}{2}$. Тогда формула принимает реккурентный вид:

$$\frac{y_{i+1} - y_i}{h} = y_i + \frac{h}{2}[f(x_i, y_i) + f(x_i + h, y_i + hf(x_i), y_i)]$$

В случае решения задачи Коши для системы дифференциальных уравнений, эта

формула также справедлива для любого y_i из вектора решений. Гораздо более точный результат дает схема Рунге-Кутта четвертого порядка точности следующего вида:

$$\frac{y_{i+1} - y_i}{h} = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \text{где}$$

Описание программы

В программе реализовано 4 функции:

- общие параметры функций:
  - x0, y0(i) - параметры начального условия $y(x_0) = y_0$
  - func - функция(-ии) $f(x, y)$ - данного ОДУ
  - x_length - длина области, на которой решается ОДУ
  - a - коэффициент α - в алгоритме Рунге-Кутта 2-го порядка точности
  - n - количество отрезков, на которое разбивается искомый отрезок
- diff2 (x0, y0, func, x_length = 20, a = 1, n = 100) - функция решения задачи Коши методом Рунге-Кутта 2-го порядка точности.
- diff2_sys(x0, y00, y01, func, x_length=20, n=100, a=1) - функция решения задачи Коши системы ОДУ методом Рунге-Кутта 2-го порядка точности.
- diff4 (x0, y0, func, x_length=15, n=100) - функция решения задачи Коши методом Рунге-Кутта 4-го порядка точности.
- diff4_sys (x0, y00, y01, func1, func2, x_length=15, n=100) - функция решения задачи Коши системы ОДУ методом Рунге-Кутта 4-го порядка точности.

Код программы

Функции решения дифференциального уравнения и системы ОДУ второго порядка точности

```
def diff2(x0, y0, func, x_length = 20, a = 1, n = 100):
    h = float(x_length) / n
    x = np.arange(x0, x0 + x_length + h, h)
    y = np.array([0.0]*len(x))
    y[0] = y0
    for i in range(len(x)-1):
        y[i+1] = y[i] +((1 - a)*func(x[i], y[i])
                    + a*func(x[i] + float(h) / (2*a), y[i]
                    + (float(h) / (2*a)) * func(x[i], y[i])))*h
    return x, y
```

```
def diff2_sys(x0, y00, y01, func, x_length=20, n=100, a=1):
    h = float(x_length) / n
    x = np.arange(x0, x0 + x_length + h, h)
    y = np.array([[0.0]*len(x)]*2)
    y[0][0] = y00
    y[1][0] = y01
    coef = float(h) / (2*a)
    for i in range(len(x)-1):
        for j in range(2):
            func_ij = func[j](x[i], y[0][i], y[1][i])
            y[j][i+1] = y[j][i] + ((1 - a)*func_ij
                        + a*func[j](x[i] + coef,
                           y[0][i] + coef * func_ij,
                           y[1][i] + coef * func_ij))*h
    return x, y
```

Функции решения дифференциального уравнения и системы ОДУ четвертого порядка точности

```
def diff4(x0, y0, func, x_length=15, n=100):
    h = float(x_length) / n
    x = np.arange(x0, x0 + x_length + h, h)
    y = np.array([0.0]*len(x))
    y[0] = y0
    for i in range(len(x)-1):
        k1 = func(x[i], y[i])
        k2 = func(x[i] + h / 2, y[i] + (h / 2)*k1)
        k3 = func(x[i] + (h / 2), y[i] + (h / 2)*k2)
        k4 = func(x[i] + h, y[i] + h*k3)
        y[i+1] = y[i] + (h / 6) * (k1 + 2*k2 + 2*k3 + k4)
    return x, y
```

Тестирование программы

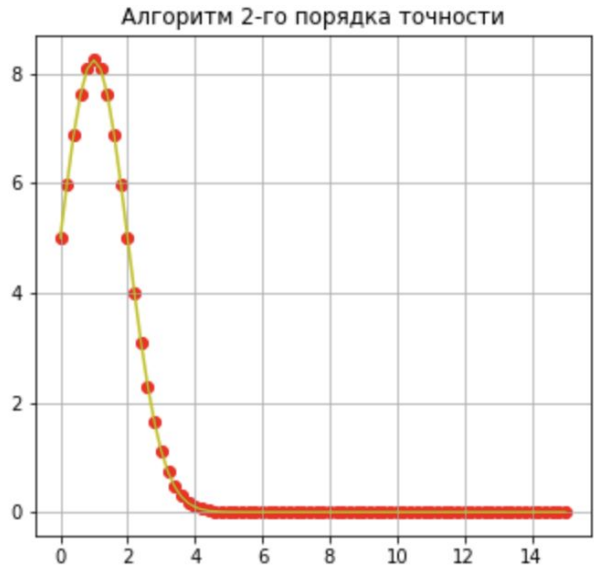
Таблица 1- задача 4

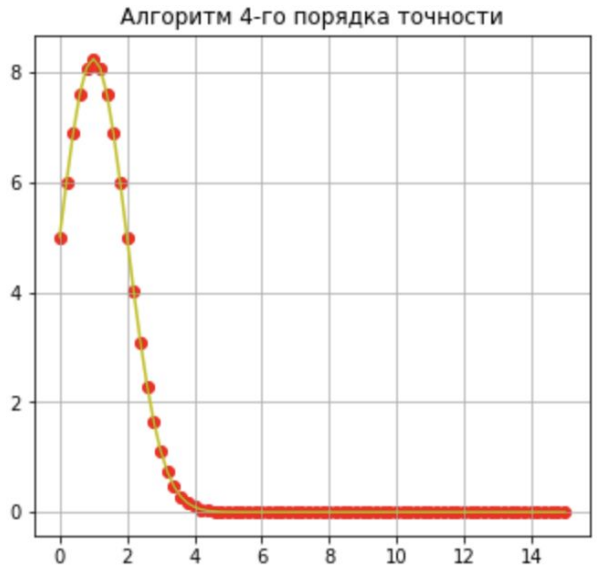
$y' = y - y * x,\ x(0) = 5$, решение - $5e^{-\frac{1}{2}x(x-2)}$

```
def norm(y, y_pred): # функция для вычсления нормы max|y_i - y(x_i)| i=0...n
    norm = 0
    for i in range(len(y)):
        if np.fabs(y_pred[i] - y[i]) > norm:
            norm = np.fabs(y_pred[i] - y[i])
    return norm
```

```
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
x, y = diff2(0.0, 5.0, func1, x_length=15, n = 75, a = 1)
axes[0].scatter(x, y, color="r")
axes[0].grid()
y_ = ans1(x)
axes[0].set_title('Алгоритм 2-го порядка точности')
axes[0].plot(x, y_, color="y")

print("Норма для алгоритма 2-го порядка точности: ", norm(y, y_))

x, y = diff4(0.0, 5.0, func1, x_length=15, n = 75)
axes[1].scatter(x, y, color="r")
axes[1].grid()
y_ = ans1(x)
axes[1].set_title('Алгоритм 4-го порядка точности')
axes[1].plot(x, y_, color="y")
print("Норма для алгоритма 4-го порядка точности: ", norm(y, y_))
```

```
Норма для алгоритма 2-го порядка точности:  0.024218011443865706
Норма для алгоритма 4-го порядка точности:  0.00026909639525019013
```



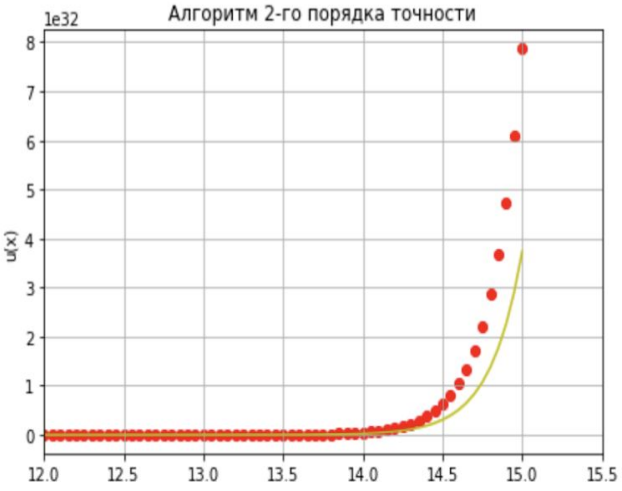


Дополнительный тест

$y_1' = 2y_1 + y_2;\ y_2' = 3y_1 + 4y_2;\ y_1(0) = 2;\ y_2(1) = 2$

решение - $y_1 = e^x + e^{5x};\ y_2 = -e^x + 3e^{5x}$

```
Норма для алгоритма 2-го порядка точности (коэффициент):  1.9257046456068e+33
Норма для алгоритма 4-го порядка точности (коэффициент):  1.4918172747090327e+33
```



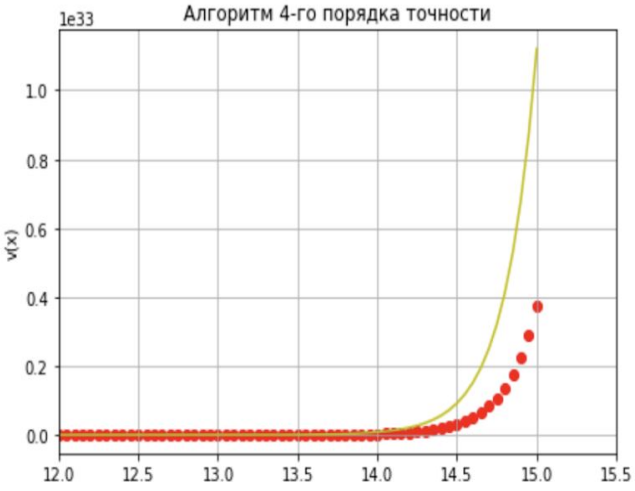
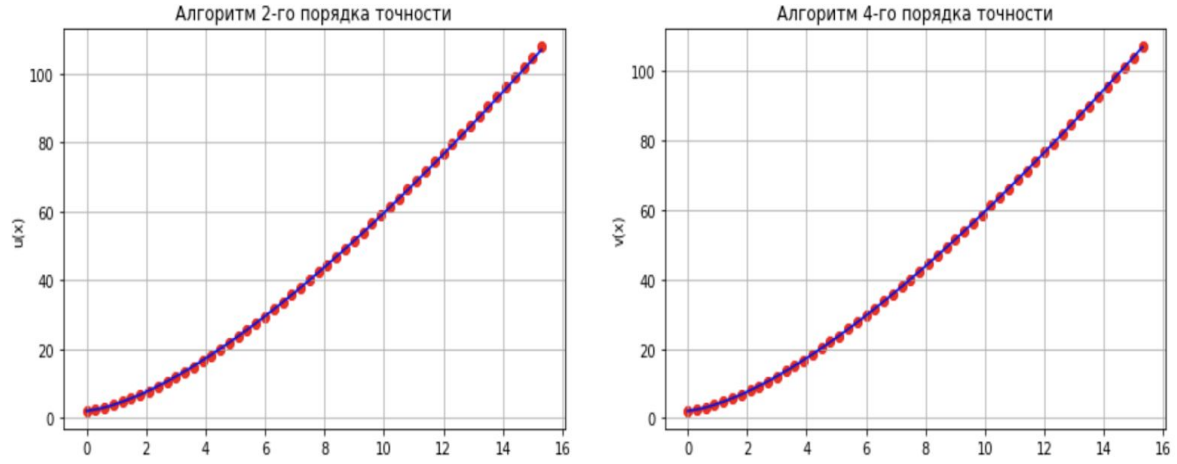
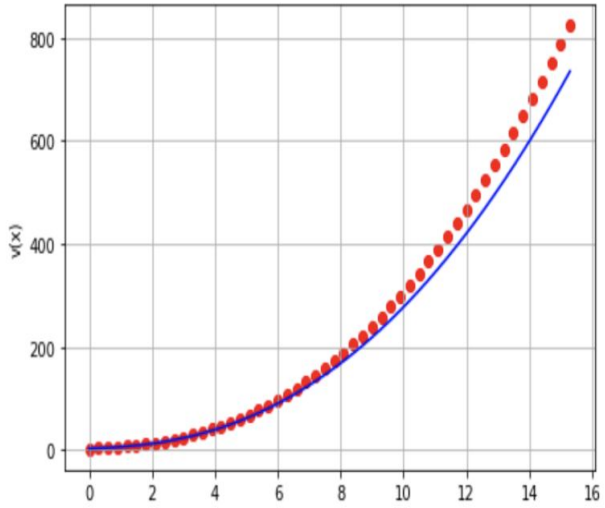
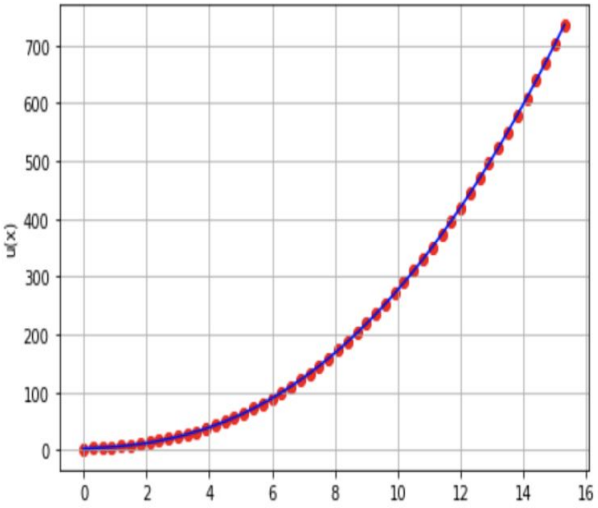


Таблица 2 - 13 (Решение получено при помощи функции ordeint библиотеки scipy)

```
Норма для алгоритма 2-го порядка точности:  90.98226572653365
Норма для алгоритма 4-го порядка точности:  0.0034453213223741 84
```







Выводы

В ходе работы рассмотрен метод Рунге-Кутта второго и четвертого порядков для решения задачи Коши дифференциального уравнения первого порядка и системы дифференциальных уравнений. Было показано на конкретных примерах, что метод Рунге-Кутта четвертого порядка точнее, чем второго, но реализуется сложнее (в схеме второго порядка точности на каждом шаге функцию f(x,y) приходилось вычислять два раза, здесь - четыре раза).

Подвариант №2

Решение краевой задачи для дифференциального уравнения второго порядка,разрешенного относительно старшеий производной

Цель работы

Освоить метод прогонки решения краевой задачи для дифференциального уравнения второго порядка.

Постановка задачи

Рассматривается линейное дифференциальное уравнение второго порядка вида:

$y'' + p(x)y' + q(x)y = f(x)$ с дополнительными условиями в граничных точках:

$$\begin{cases} \sigma_1 y(0) + \gamma_1 y_1'(0) = \delta_1 \\ \sigma_2 y(1) + \gamma_2 y_1'(1) = \delta_2 \end{cases}$$

Цели и задачи практической работы:

1. Решить краевую задачу (1)-(2) методом конечных разностей,аппроксимировав ее разностнои схемой второго порядка точности (на равномерной сетке); полученную систему конечно-разностных уравнений решить методом прогонки;

2. Найти разностное решение задачи и построить его график;
3. Найденное разностное решение сравнить с точным решением дифференциального уравнения.

Описание метода

Рассмотрим на $[a;b]$ равномерную сетку из $n+1$ узлов с шагом $h = \frac{b-a}{n}$ и узлами $x_i = a + ih,\ i = 0, 1, ..., n$. Обозначим

$$y_i = y(x_i),\ p_i = p(x_i),\ q_i = q(x_i),\ f_i = f(x_i)$$

Аппроксимировав производные разностными функциями второго порядка точности, получим:

Перегруппировав коэффициенты при соответствующих y_i, получим:

$$A_i y_{i-1} + B_i y_i + C_i y_i + 1 = D_i,\ i = 1, ..., n-1$$

Для получения еще двух уравнений, аппроксимируем производные в граничных точках:

$$\begin{cases} \sigma_1 \frac{y_0 + y_1}{2} + \gamma_1 \frac{y_1 - y_0}{h} = \delta_1 \\ \sigma_2 \frac{y_n + y_{n+1}}{2} + \gamma_2 \frac{y_{n+1} - y_n}{h} = \delta_2 \end{cases}$$

Перегруппируем слагаемые, получим:

Имеем СЛАУ с трехдиагональной матрицей, с $n+1$ неизвестными. Ее можно решить методом прогонки. Метод основан на предположении, что искомые неизвестные связаны рекуррентным соотношением:

$x_i = \alpha_{i+1} x_{i+1} + \beta_{i+1},\ 0 \le i \le n-1$

С помощью этой формулы выразим x_{i-1} через x_{i+1} и подставим выраженные x_{i-1} и x_i в исходное уравнения. Получим

$(A_i\alpha_i\alpha_{i+1} + C_i\alpha_{i+1} + B_i)x_{i+1} + A_i\alpha_i\beta_{i+1} + A_i\beta_i + C_i\beta_{i+1} - D_i = 0,\ i = 1...n+1$

Это равенство будет выполняться независимо от решения, если потребовать, чтобы:

$$\begin{cases} A_i\alpha_i\alpha_{i+1} + C_i\alpha_{i+1} + B_i = 0 \\ A_i\alpha_i\beta_{i+1} + A_i\beta_i + C_i\beta_{i+1} - D_i = 0 \end{cases}$$

Получаем рекуррентные соотношения для прогоночных коэффициентов:

$$\alpha_{i+1} = \frac{-B_i}{A_i\alpha_i + C_i}, \beta_{i+1} = \frac{D_i - A_i\beta_i}{A_i\alpha_i + C_i}, \mathrm{i} = 1,\ldots,\mathrm{n+1}$$

Находим из первого уравнения α_1, β_1 , а по рекуррентным соотношениям $\alpha_2, ..., \alpha_n$ $\beta_2, ..., \beta_n$. Далее из последнего уравнения находим $y_{n+2} = \beta_{n+2}$. Отсюда находятся остальные неизвестные $y_{n+1}, ..., y_1$ в процессе обратной прогонки с помощью реккурентной формулы.

Описание программы

- реализована функция diff (f, p, q, a, b, a0, b0, a1, b1, A, B, N = 100), где:
  - f, p, q - соответственно функции $f(x),\ p(x),\ q(x)$
  - a, b - соответственно левая и правая краевые точки
  - a0, a1 и b0, b1 - коэффициенты 1-го и 2-го дополнительного условия
  - A, B - правые части 1-го и 2-го дополнительного условия
  - N - количество итераций

Код программы

Функция решения дифференциального уравнения

```
def norm(y, y_pred): # функция для вычсления нормы max|y_i - y(x_i)| i=0...n
    norm = 0
    for i in range(len(y)):
        if np.fabs(y_pred[i] - y[i]) > norm:
            norm = np.fabs(y_pred[i] - y[i])
    return norm
```

```
def diff(f, p, q, a, b, a0, b0, a1, b1, A, B, N = 100):
    h = (b - a) / N
    y = np.array([0.0]*(N + 1))
    f1 = np.array([0.0]*(N + 1))
    f2 = np.array([0.0]*(N + 1))
    x = np.array([a + h * i for i in range(N+1)])
    f1[1] = -a1 / (a0 * h - a1)
    f2[1] = A * h / (a0 * h - a1)
    for i in range(1, N):
        k = 1.0 / (h * h) - p(x[i]) / (2.0 * h)
        l = 2.0 / (h * h) - q(x[i])
        m = 1.0 / (h * h) + p(x[i]) / (2.0 * h)
        r = f(x[i])
        f1[i + 1] = m / (l - k * f1[i])
        f2[i + 1] = (f2[i] * k - r) / (l - k * f1[i])

    y[N] = (b1 * f2[N] + B * h) / (b1 * (1.0 - f1[N]) + b0 * h)

    for i in range(N):
        j = N - i
        y[j - 1] = y[j] * f1[j] + f2[j]
    return x, y
```

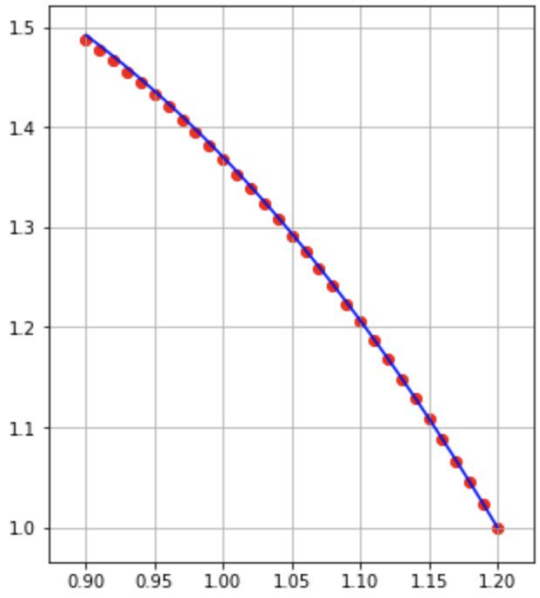
Тестирование программы

Тест №1

$$\begin{cases} y'' - xy' + 2y = x - 1 \\ y(0.9) - 0.5y'(0.9) = 2 \\ y(1.2) = 1 \end{cases}$$

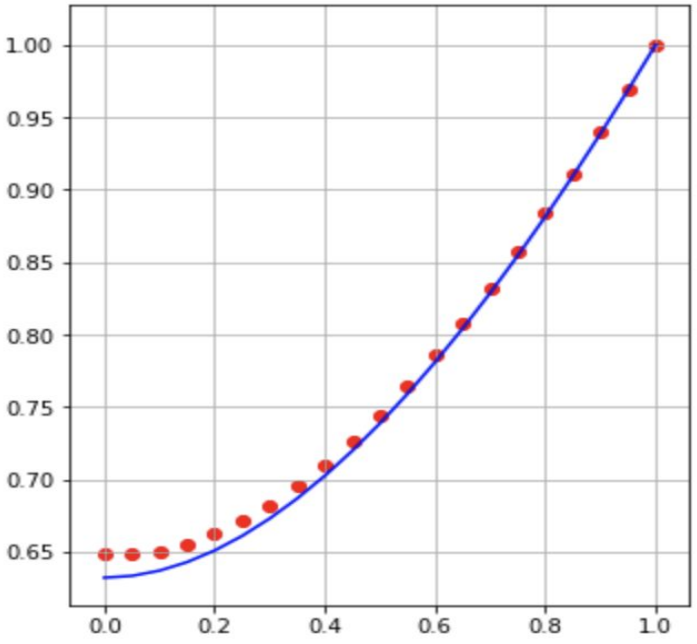
```
# y"-xy'+2y=x-1
# y(0.9) - 0.5y'(0.9) =2
# y(1.2) = 1
def ans1(x):
    return ((0.661683 - 0.661683*(x**2))*erfi(x/np.sqrt(2.0))
            - 1.45132 * (x**2)
            + (0.527947 * np.exp((x**2)/2.0) + 1)*x
            + 0.951319)

def f1(x):
    return x - 1

def p1(x):
    return -x

def q1(x):
    return 2

x, y = diff(f1, p1, q1, a=0.9, b=1.2, a0=1.0, b0=1.0,
            a1=-0.5, b1=0.0, A=2, B=1, N=30)

plt.figure(figsize=(5, 6))
plt.grid()
plt.scatter(x, y, color="r")
plt.plot(x, ans1(x), color="b")
print('Норма: ', norm(y, ans1(x)))
```

```
Норма:  0.004280103041253147
```



Тест №2

```
Норма:  0.01588159284331314
```

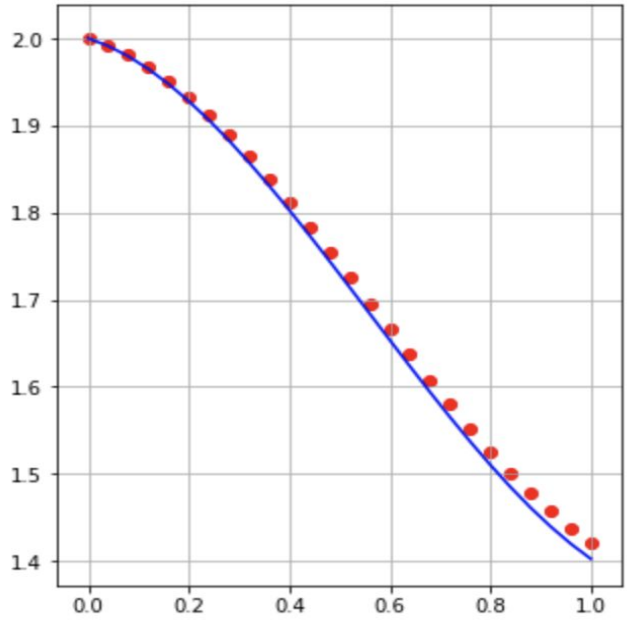


Тест №3

y'' + y = 3x; y(0) = 2; y(1) + y'(1)=1
Решение при помощи WolframAlpha:

$$\frac{3\,x\sin(1)+2\sin(1-x)-5\sin(x)+3\,x\cos(1)+2\cos(1-x)}{\sin(1)+\cos(1)}$$

```
Норма:  0.018663852669889014
```



Выводы

В ходе практической работы был рассмотрен метод прогонки, применяемый для разностного решения краевой задачи для ОДУ второго порядка, разрешенного относительно старшей производной. При тестировании было показано, что при увеличении числа разбиений, точность решения задачи увеличивается. Данный метод оказался простым в реализации и точным для вычислении.