МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу

«ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»

ЗАДАНИЕ №2

Вариант 1.1.3, 1.2.17, 2.5

ОТЧЕТ

о выполненном задании

студента 204 учебной группы факультета ВМК МГУ
Васильева Руслана Леонидовича

Москва, 2019

Содержание

1 Подвриант 1

1.1 Постановка задачи

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

$$\frac{dy}{dx} = f(x, y), \;\; x > x_0, \tag{1}$$

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

$$y(x_0) = y_0. \tag{2}$$

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

В 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):

$$\begin{cases} \frac{dy_1}{dx} = f_1(x, y_1, y_2), \\ \frac{dy_2}{dx} = f_2(x, y_1, y_2), \end{cases} \qquad x > x_0 \tag{3}$$

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

$$y_1(x_0) = y_1^{(0)}, \;\; y_2(x_0) = y_2^{(0)}. \tag{4}$$

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.

1.2 Цели и задачи практической работы

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);

2. Найти численное решение задачи и построить его график;

3. Найденное численное решение сравнить с точным решением дифференциального уравнения.

1.3 Методы решения

Численно решим задачу Коши (1)—(2) и (3)—(4) с помощью метода Рунге—Кутта. Для этого построим равномерную сетку на $[x_0,\ x_0 + l]$ с шагом $h = l/n$:

$$x_i = x_0 + ih, \qquad i \in \{0,\ \dots\ , n\}$$

Рекуррентное соотношение второго порядка точности имеет вид:

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

В программе использована схема $\alpha = 1$, и рекуррентная формула принимает вид:

$$y_{i+1} = y_i + hf\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}hf(x_i,\ y_i)\right) \tag{5}$$

Схема расчета заключается в следующем. Сначала делается половинный шаг $h/2$: по схеме Эйлера вычисляется величина

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

Затем находится значение функции f в точке $x_{i+\frac{1}{2}}, y_{i+\frac{1}{2}}$ и подставляется в (5).

Также реализуется 4-й порядок точности следующего вида:

$$y_{n+1} = y_n + \tfrac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \tag{6}$$

где

$$\begin{aligned}
k_1 &= f(t_n, y_n),\\
k_2 &= f\left(x_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right),\\
k_3 &= f\left(x_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right),\\
k_4 &= f\left(x_n + h, y_n + hk_3\right).
\end{aligned}$$

Чтобы унифицировать программу, задачи (1)–(2) и (3)–(4) представляются как частные случаи задачи Коши для системы из дифференциальных уравнений первого порядка с соответствующим числом начальных условий. Идейно формы (5) и (6) остаются такими же: в алгоритме все y, f и k представляются векторами.

1.4 Реализация

1.4.1 Основные функции

```
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.style.use('seaborn-bright')
plt.rc('font', size=14)
```

Функции, реализующие метод Рунге—Кутта 2-го и 4-го порядка, имеют одинаковую структуру:

```
def rk2(f, x0, y0, l, n):
    m = len(y0)
    x = np.linspace(x0, l, n + 1)
    h = (l - x0) / n
    y = np.empty(shape=(n + 1, m))
    y[0] = y0.copy()
    for i in range(n):
        k = np.empty(shape=(2, m))
        for j in range(m):
            k[0][j] = f[j](x[i], y[i])
        for j in range(m):
            k[1][j] = f[j](x[i] + h / 2, y[i] + k[0] * h / 2)
        y[i + 1] = y[i] + k[1] * h
    return x, y.T
```

```
def rk4(f, x0, y0, l, n):
    m = len(y0)
    x = np.linspace(x0, l, n + 1)
    h = (l - x0) / n
    y = np.empty(shape=(n + 1, m))
    y[0] = y0.copy()
    for i in range(n):
        k = np.empty(shape=(4, m))
        for j in range(m):
            k[0][j] = f[j](x[i], y[i])
        for j in range(m):
```

```
            k[1][j] = f[j](x[i] + h / 2, y[i] + k[0] * h / 2)
        for j in range(m):
            k[2][j] = f[j](x[i] + h / 2, y[i] + k[1] * h / 2)
        for j in range(m):
            k[3][j] = f[j](x[i] + h, y[i] + k[2] * h)
        y[i + 1] = y[i] + (k[0] + 2 * k[1] + 2 * k[2] + k[3]) * h / 6
    return x, y.T
```

Они принимают в качестве параметров:

- f — вектор-функции, соответствующие производным
- x_0, y_0 — начальные условия (число и вектор)
- l — длина промежутка
- n — число шагов

И возвращают

- x — узлы приближенного решения
- $y.T$ — значения сеточных функций на этих узлах

Для вывода таблиц использован инструмент из Pandas:

```
def make_table(xs, ys_rk2, ys_rk4, real=None):
    data = {'x_i': xs, 'P.K. II': ys_rk2, 'P.K. IV': ys_rk4}
    if real is not None:
        data['y(x_i)'] = real
    return pd.DataFrame.from_dict(data)
```

1.4.2 Тестирование

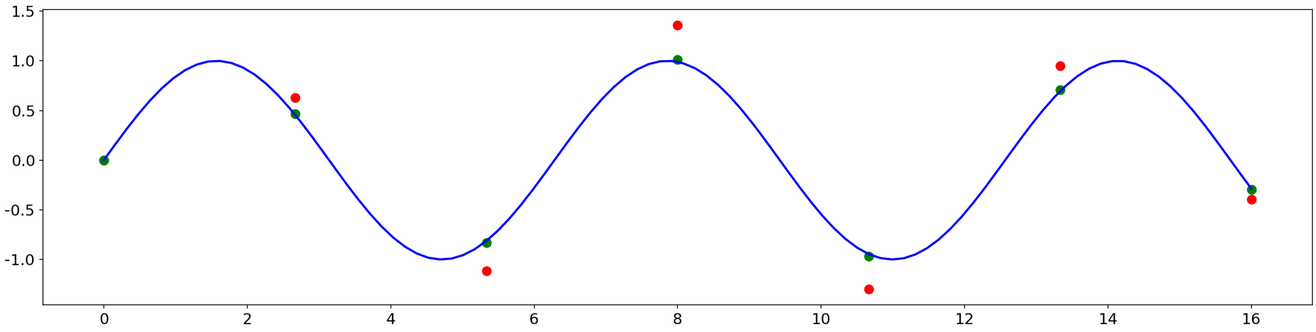
Начнем исследование с наглядной задачи:

$$f(x, y) = cos(x), \ y(0) = 0$$

Точное решение: $y = sin(x)$

```
x0 = 0
l = 16
plt.figure(figsize=(21,5))
base = np.linspace(x0, x0 + l, 100)
plt.plot(base, np.sin(base), lw=2, c='blue')
y0 = np.array([0])
def f0(x, y):
    return np.cos(x)
f = [f0]
n = 6
xs, ys1 = rk2(f, x0, y0, l, n)
plt.scatter(xs, ys1[0], c='red', lw=3)
xs, ys2 = rk4(f, x0, y0, l, n)
plt.scatter(xs, ys2[0], c='green', lw=3)
print(make_table(xs, ys1[0], ys2[0], np.sin(xs)))
```

```
         x_i    Р.К. II    Р.К. IV     y(x_i)
0   0.000000   0.000000   0.000000   0.000000
1   2.666667   0.627300   0.467388   0.457273
2   5.333333  -1.115749  -0.831322  -0.813329
3   8.000000   1.357231   1.011245   0.989358
4  10.666667  -1.298294  -0.967332  -0.946396
5  13.333333   0.951983   0.709303   0.693952
6  16.000000  -0.394954  -0.294272  -0.287903
```



На этом примере явно проиллюстрировано преимущество 4-го порядка точности. Здесь и далее оттенками голубого будут построены кривые, являющиеся графиками точного решения, а красным и зеленым – графики сеточных функций, полученных с помощью метода Рунге—Кутта 2-го и 4-го порядка соответственно.

Рассмотрим задачу из варианта:

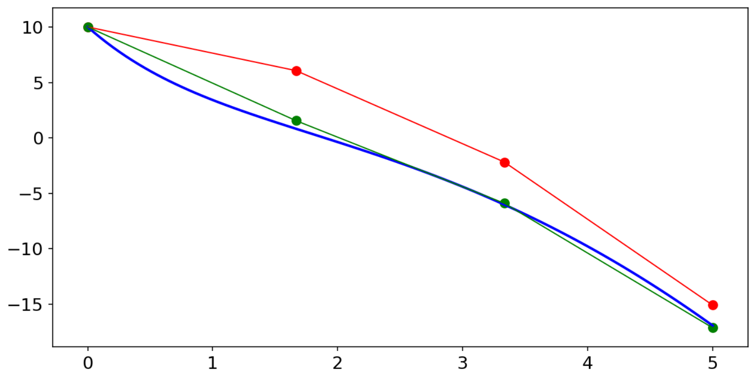
$$f(x, y) = -y - x^2,\ (x_0,\ y_0) = (0,\ 10)$$

Аналитическое решение:

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

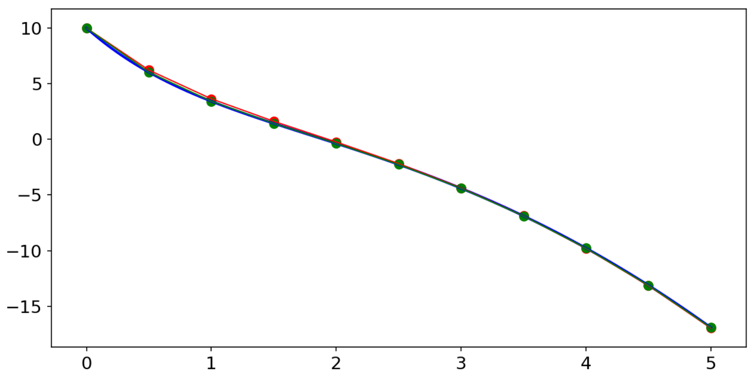
```
x0 = 0
y0 = np.array([10])
l = 5
n = 3
def y(x):
    return - x ** 2 + 2 * x + 12 * np.exp(-x) - 2
def f0(x, y):
    return -y[0] - x ** 2
f = [f0]
base = np.linspace(x0, x0 + l, 100)
plt.figure(figsize=(10, 5))
plt.plot(base, y(base), lw=2, c='blue')
xs, ys1 = rk2(f, x0, y0, l, n)
plt.plot(xs, ys1[0], c='red', lw=1)
plt.scatter(xs, ys1[0], c='red', lw=2)
xs, ys2 = rk4(f, x0, y0, l, n)
plt.plot(xs, ys2[0], c='green', lw=1)
plt.scatter(xs, ys2[0], c='green', lw=2)
print(make_table(xs, ys1[0], ys2[0], y(xs)))
plt.savefig('task.png', quality=95, dpi=150)
```

```
        x_i      Р.К. II      Р.К. IV      y(x_i)
0  0.000000  10.000000  10.000000  10.000000
1  1.666667   6.064815   1.553069   0.822063
2  3.333333  -2.178498  -5.896681  -6.016357
3  5.000000 -15.076446 -17.118861 -16.919145
```



То же самое, но с большим числом точек:

```
     x_i      Р.К. II      Р.К. IV       y(x_i)
0    0.0   10.000000   10.000000   10.000000
1    0.5    6.218750    6.030599    6.028368
2    1.0    3.636719    3.417004    3.414553
3    1.5    1.616699    1.429458    1.427562
4    2.0   -0.239563   -0.374834   -0.375977
5    2.5   -2.180977   -2.264548   -2.264980
6    3.0   -4.363111   -4.402707   -4.402555
7    3.5   -6.883194   -6.888231   -6.887631
8    4.0   -9.801996   -9.781140   -9.780212
9    4.5  -13.157498  -13.117853  -13.116692
10   5.0  -16.973436  -16.920468  -16.919145
```



Покажем важность плотности сетки при, например, частых экстремумах и перегибах:

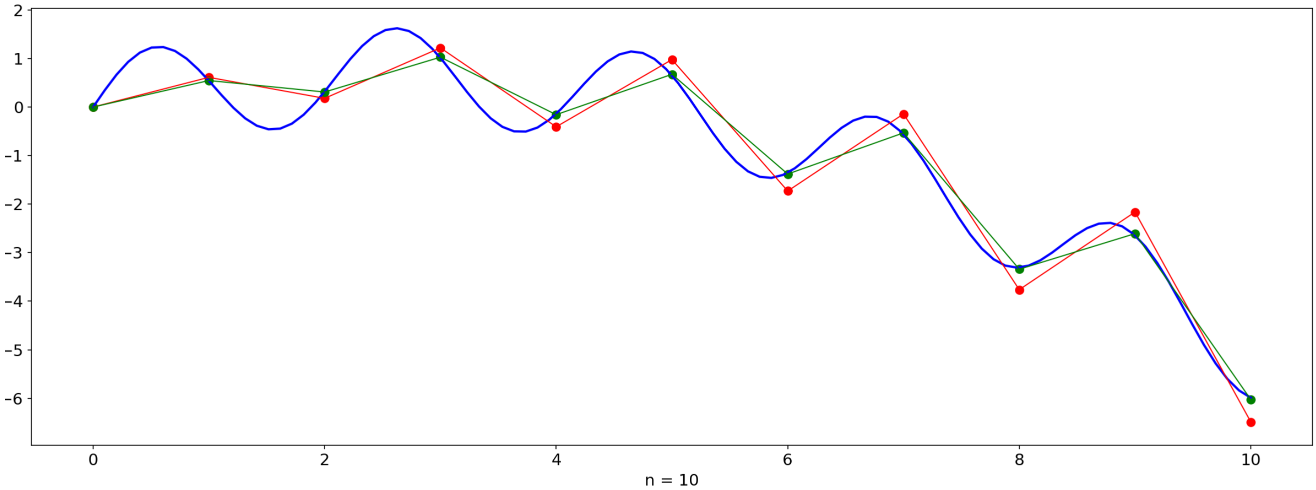
$$f(x, y) = -0.2x + 3cos(3x) + 0.5, \ \ y(0) = 0$$

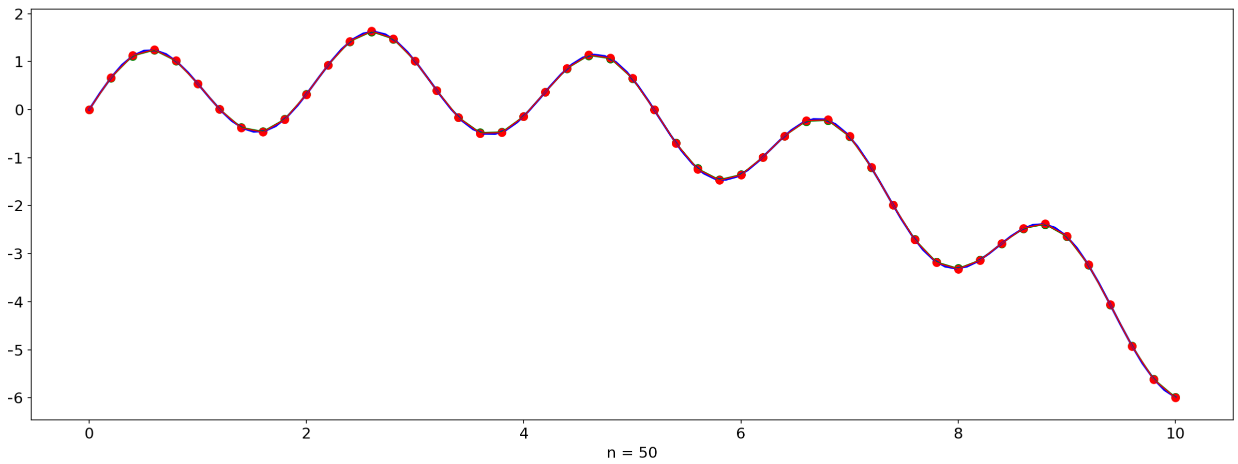
$$y = 0.5x + \sin{(3x)} - 0.1x^2$$

```
x0 = 0
y0 = np.array([0])
l = 10
n = 10
def y(x):
```

```
    return 0.5 * x + np.sin(3*x) - 0.1 * x * x
def f0(x, y):
    return -0.2 * x + 3 * np.cos(3*x) + 0.5
f = [f0]
base = np.linspace(x0, x0 + l, 100)
plt.figure(figsize=(20, 7))
plt.plot(base, y(base), lw=2, c='blue')
plt.xlabel(f'n = {n}')
xs, ys1 = rk2(f, x0, y0, l, n)
plt.plot(xs, ys1[0], c='red', lw=1)
plt.scatter(xs, ys1[0], c='red', lw=2)
xs, ys2 = rk4(f, x0, y0, l, n)
plt.plot(xs, ys2[0], c='green', lw=1)
plt.scatter(xs, ys2[0], c='green', lw=2)
print(make_table(xs, ys1[0], ys2[0], y(xs)))
plt.savefig('strange.png', quality=95, dpi=150)
```

```
     x_i    Р.К. II    Р.К. IV     y(x_i)
0    0.0   0.000000   0.000000   0.000000
1    1.0   0.612212   0.546478   0.541120
2    2.0   0.179824   0.309975   0.320585
3    3.0   1.219730   1.027766   1.012118
4    4.0  -0.406881  -0.156946  -0.136573
5    5.0   0.977881   0.674978   0.650288
6    6.0  -1.729310  -1.379501  -1.350987
7    7.0  -0.141865  -0.531578  -0.563344
8    8.0  -3.761779  -3.339962  -3.305578
9    9.0  -2.161833  -2.607312  -2.643624
10  10.0  -6.485769  -6.025546  -5.988032
```





Перейдем к системам. В варианте предложена следующая:

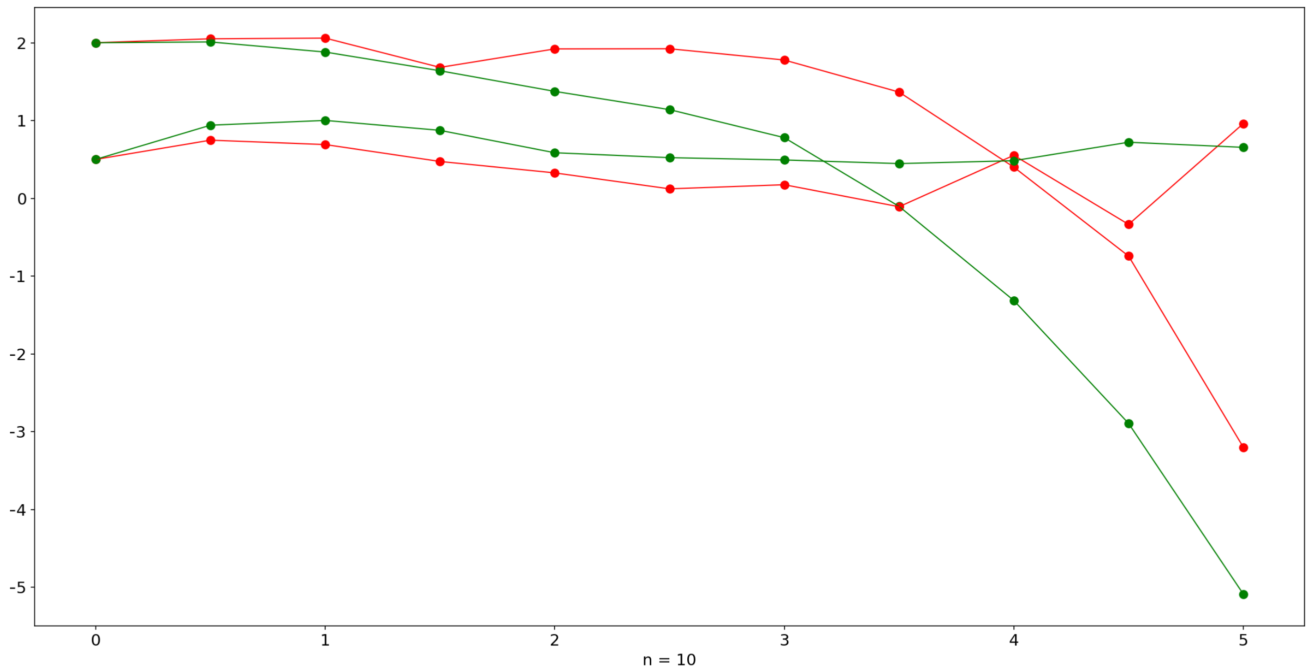
$$f_1 = sin(1.4 \cdot y_1^2) - x + y_2$$
$$f_2 = x + y_1 - 2.2y_2^2 + 1$$
$$y_1(0) = 1$$
$$y_2(0) = 0.5$$

```
def f1(x, y):
    return np.sin(1.4 * y[0] * y[0]) - x + y[1]
def f2(x, y):
    return x + y[0] - 2.2 * y[1] * y[1] + 1
x0 = 0
y0 = np.array([2, 0.5])
f = [f1, f2]
l = 5
n = 10
xs, ys1 = rk2(f, x0, y0, l, n)
xs, ys2 = rk4(f, x0, y0, l, n)
plt.figure(figsize=(20, 10))
plt.xlabel(f'n = {n}')
for i in range(2):
    plt.plot(xs, ys1[i], c='red', lw=1)
    plt.scatter(xs, ys1[i], c='red', lw=2)
    plt.plot(xs, ys2[i], c='green', lw=1)
    plt.scatter(xs, ys2[i], c='green', lw=2)
print(make_table(xs, ys1[0], ys2[0]))
print()
print(make_table(xs, ys1[1], ys2[1]))
plt.savefig('sy.png', quality=95, dpi=150)
```

```
    x_i    Р.К. II    Р.К. IV
```

```
0   0.0  2.000000  2.000000
1   0.5  2.050556  2.010047
2   1.0  2.059456  1.880006
3   1.5  1.682805  1.639569
4   2.0  1.920696  1.374532
5   2.5  1.922635  1.139920
6   3.0  1.777539  0.779472
7   3.5  1.365552 -0.103979
8   4.0  0.400436 -1.313744
9   4.5 -0.742960 -2.894210
10  5.0 -3.201198 -5.088728

    x_i    Р.К. II    Р.К. IV
0   0.0  0.500000  0.500000
1   0.5  0.747170  0.939620
2   1.0  0.690817  1.001336
3   1.5  0.473463  0.873455
4   2.0  0.326597  0.585482
5   2.5  0.121590  0.522526
6   3.0  0.174616  0.492719
7   3.5 -0.108466  0.446294
8   4.0  0.552485  0.483240
9   4.5 -0.335560  0.720810
10  5.0  0.961192  0.654732
```



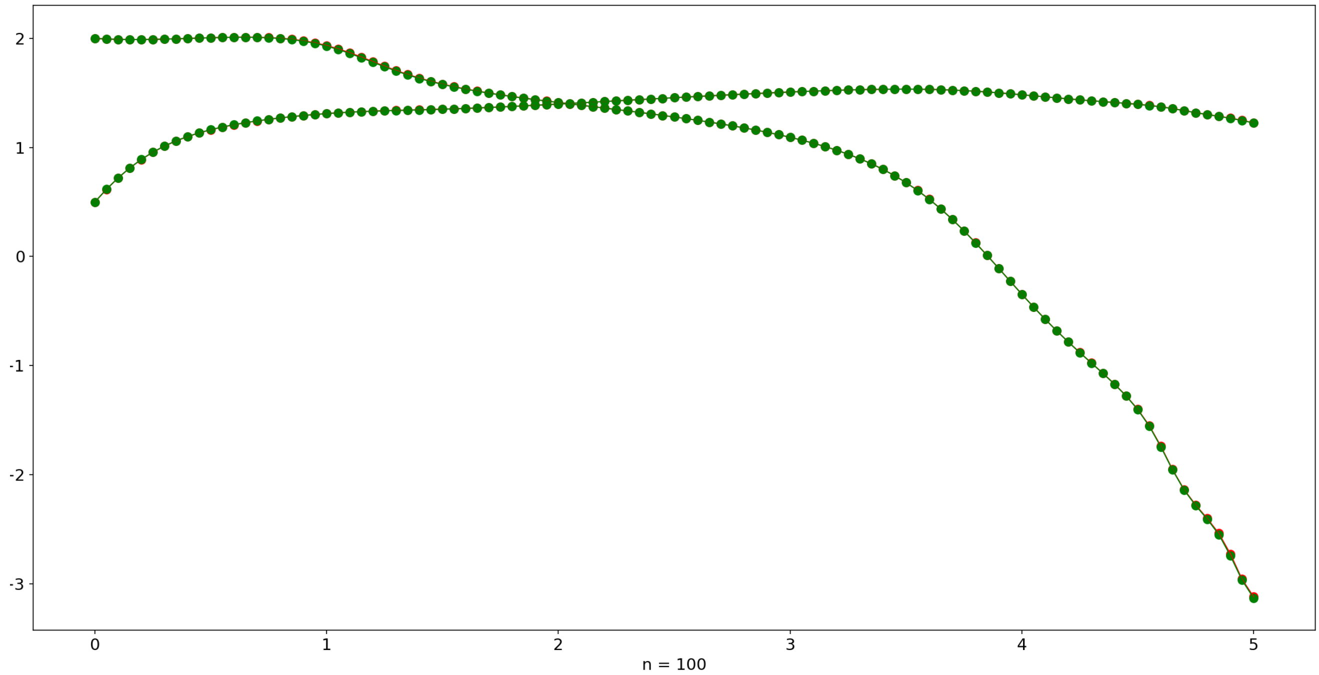
Видим, что функции сложные, а такая разреженная сетка почти не дает о них информации. Увеличим число точек:

```
       x_i    P.K. II    P.K. IV
0     0.00  2.000000  2.000000
1     0.05  1.994542  1.994524
2     0.10  1.991180  1.991129
3     0.15  1.989691  1.989599
4     0.20  1.989788  1.989653
..     ...       ...       ...
96    4.80 -2.402215 -2.408870
97    4.85 -2.539122 -2.549724
98    4.90 -2.727669 -2.745141
99    4.95 -2.958823 -2.967645
100   5.00 -3.120238 -3.132522

[101 rows x 3 columns]

       x_i    P.K. II    P.K. IV
0     0.00  0.500000  0.500000
1     0.05  0.616436  0.616569
2     0.10  0.720180  0.720528
3     0.15  0.810890  0.811486
4     0.20  0.889029  0.889862
..     ...       ...       ...
96    4.80  1.301965  1.301654
97    4.85  1.285854  1.285082
98    4.90  1.269104  1.267491
99    4.95  1.248798  1.246855
100   5.00  1.226061  1.224863

[101 rows x 3 columns]
```



Wolfram не дает аналитического решения данной системы, так что проверим также на другом тесте.

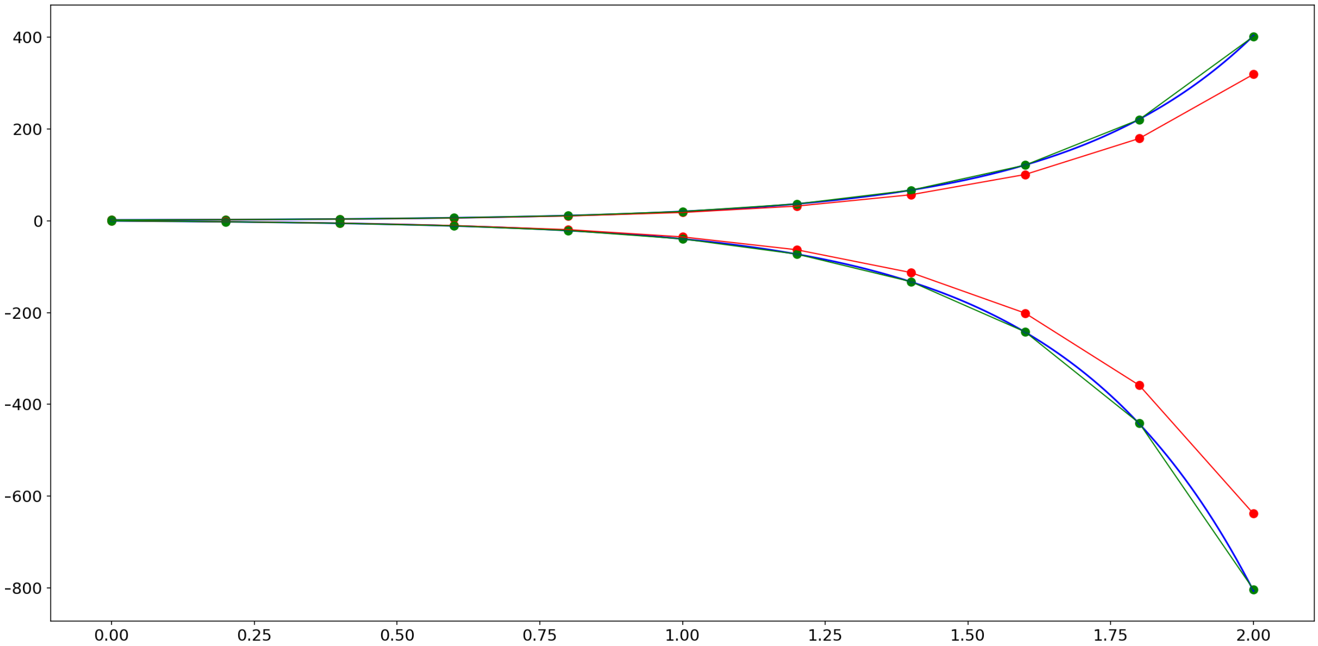
$$f_1 = y_1 - y_2$$
$$f_2 = y_2 - 4y_1$$
$$y_1(0) = 2, y_2(0) = 0$$
$$y_1 = e^{-x} + e^{3x}, y_2 = 2e^{-x} - 2e^{3x}$$

```
def f1(x, y):
    return y[0] - y[1]
def f2(x, y):
    return y[1] - 4 * y[0]
def y1(x):
    return np.exp(-x) + np.exp(3 * x)
def y2(x):
    return 2 * np.exp(-x) - 2 * np.exp(3 * x)
x0 = 0
y0 = np.array([2, 0])
f = [f1, f2]
l = 2
n = 10
xs, ys1 = rk2(f, x0, y0, l, n)
xs, ys2 = rk4(f, x0, y0, l, n)
base = np.linspace(x0, x0 + l, 200)
plt.figure(figsize=(20, 10))
plt.plot(base, y1(base), c='blue')
plt.plot(base, y2(base), c='blue')
```

```python
for i in range(2):
    plt.plot(xs, ys1[i], c='red', lw=1)
    plt.scatter(xs, ys1[i], c='red', lw=2)
    plt.plot(xs, ys2[i], c='green', lw=1)
    plt.scatter(xs, ys2[i], c='green', lw=2)
print(make_table(xs, ys1[0], ys2[0], y1(xs)))
print()
print(make_table(xs, ys1[1], ys2[1], y2(xs)))
plt.savefig('last.png', quality=95, dpi=150)
```

```
    x_i       P.K. II      P.K. IV       y(x_i)
0   0.0     2.000000     2.000000     2.000000
1   0.2     2.600000     2.640133     2.640850
2   0.4     3.840800     3.987822     3.990437
3   0.6     6.191120     6.591308     6.598459
4   0.8    10.490880    11.455127    11.472505
5   1.0    18.239730    20.413836    20.453416
6   1.2    32.110809    36.812895    36.899429
7   1.4    56.865394    66.749003    66.932928
8   1.6   100.981088   121.329375   121.712314
9   1.8   179.550098   220.786883   221.571715
10  2.0   319.438260   401.975485   403.564129

    x_i       P.K. II      P.K. IV       y(x_i)
0   0.0     0.000000     0.000000     0.000000
1   0.2    -1.920000    -2.005333    -2.006776
2   0.4    -4.992000    -5.294347    -5.299594
3   0.6   -10.176768   -10.987348   -11.001672
4   0.8   -19.173274   -21.112916   -21.147695
5   1.0   -34.996501   -39.356131   -39.435315
6   1.2   -63.005592   -72.420990   -72.594080
7   1.4  -112.733646  -132.511597  -132.879468
8   1.6  -201.144519  -241.851143  -242.617042
9   1.8  -358.429718  -440.912552  -442.482235
10  2.0  -638.326728  -803.409612  -806.586916
```



1.5 Вывод

Таким образом, мы изучили методы Рунге—Кутта для решения задачи Коши, реализовав и 2-й, и 4-й порядок точности. Они были сопоставлены на простых и не очень функциях и системах, последний, разумеется, выдавал более точный результат. Также была продемонстрирована значимость выбора числа узлов.

2 Подвриант 2

2.1 Постановка задачи

Рассматривается линейное дифференциальное уравнение второго порядка вида:

$$y''(x) + p(x) \cdot y'(x) + q(x) \cdot y = f(x), \qquad a < x < b, \tag{7}$$

с дополнительными условиями в граничных точках

$$\begin{cases} \sigma_1 y(a) + \gamma_1 y'(a) = \delta_1, \\ \sigma_2 y(b) + \gamma_2 y'(b) = \delta_2. \end{cases} \tag{8}$$

2.2 Цели практической работы

1. Решить краевую задачу методом конечных разностей, аппроксимировав ее разностной схемой второго порядка точности (на равномерной сетке)
2. Полученную систему конечно-разностных уравнений решить методом прогонки;
3. Найти разностное решение задачи и построить его график
4. Найденное разностное решение сравнить с точным решением дифференциального уравнения

2.3 Метод и алгоритм решения

Построим равномерную сетку на $[a,\ b]$ с шагом $h = (b-a)/n$:

$$x_i = a + ih, \qquad i \in \{0,\ \dots,n\}$$

Заменим уравнение (7) его разностным аналогом, аппроксимировав первые и вторые производные со вторым порядком точности:

$$\begin{aligned} y'(x_i) &\approx \frac{y_{i+1} - y_{i-1}}{2h} \\ y''(x_i) &\approx \frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} \qquad\qquad i \in \{1,\ \dots,n-1\} \end{aligned}$$

На границах приблизим соответственно правой и левой разностной производной:

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

Подставляя и группируя, сводим дифференциальную задачу к разностной:

$$(\sigma_1 h - \gamma_1) y_0 + \gamma_1 y_1 = \delta_1 h \tag{9}$$

$$(1 - \frac{p_i h}{2}) y_{i-1} + (q_i h^2 - 2) y_i + (1 + \frac{p_i h}{2}) y_{i+1} = f_i h^2 \qquad i \in \{1,\ \dots,n-1\} \tag{10}$$

$$-\gamma_2 y_{n-1} + (\sigma_2 h + \gamma_2) y_n = \delta_2 h \tag{11}$$

Получили СЛАУ с трехдиагональной матрицей. Достаточным условием существования и единственности решения является, например, диагональное преобладание. В общем, будем считать, что система не вырождена. Тогда к ней можно применить **метод прогонки**.

Рассматривается система вида

$$A_i y_{i-1} + C_i y_i + B_i y_{i+1} = F_i$$

Ее неизвестные связаны следующим образом:

$$y_i = \alpha_{i+1} y_{i+1} + \beta_{i+1},$$

где

$$\alpha_{i+1} = \frac{-B_i}{A_i \alpha_i + C_i} \qquad\qquad \beta_{i+1} = \frac{F_i - A_i \beta_i}{A_i \alpha_i + C_i}$$

Чтобы упростить работу с перегонкой, формально считаем, что

$$A_0 = B_n = y_{-1} = y_{n+1} = 0$$

Тогда, чтобы соотношения не были противоречивы, положим также

$$\alpha_0 = \beta_0 = 0$$

2.4 Реализация

2.4.1 Основные функции

Метод перегоднки (для системы с трехдиагональной матрицей, учитывая вышепоставленные условия):

```
def trid(A, B, C, F):
    n = len(F)
    alpha = np.zeros(n + 1)
    beta = np.zeros(n + 1)
    for i in range(0, n):
        alpha[i + 1] = -B[i] / (A[i] * alpha[i] + C[i])
        beta[i + 1] = (F[i] - A[i] * beta[i]) / (A[i] * alpha[i] + C[i])
    z = np.zeros(n + 1)
    for i in reversed(range(n)):
        z[i] = alpha[i + 1] * z[i + 1] + beta[i + 1]
        print(z[i])
    return z[:-1]
```

Решение краевой задачи методом конечных разностей:

```
def fdm_new(p, q, f, a, b, sigma, gamma, delta, n):
    h = (b - a) / n
    x = np.linspace(a, b, n + 1)
    F = np.empty(n + 1)
    F[0] = delta[0] * h
    F[1:n] = f(x[1:n])
    F[n] = delta[1] * h
    A = np.zeros(n + 1)
    B = np.zeros(n + 1)
    C = np.zeros(n + 1)
    C[0] = sigma[0] * h - gamma[0]
    B[0] = gamma[0]
    A[n] = -gamma[1]
    C[n] = sigma[1] * h + gamma[1]
    A[1:n] = 1 - p(x[1:n]) * h / 2
    C[1:n] = q(x[1:n]) * h * h - 2
    B[1:n] = 1 + p(x[1:n]) * h / 2
    y = trid(A, B, C, F)
    return x, y
```

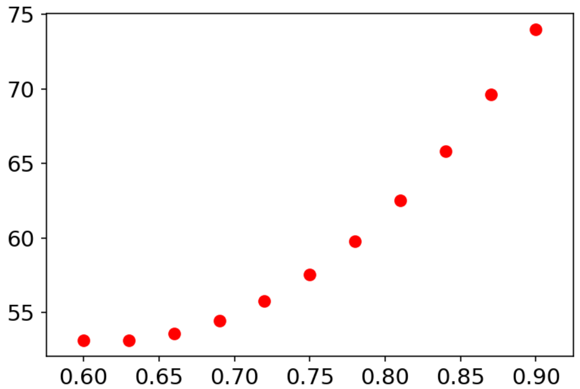
Имена всех параметров стандартны и совпадают с использовавшимися ранее.

2.4.2 Тестирование

В варианте предлагается решить задачу:

$$y'' + 2y' - xy = x^2;$$
$$y'(0.6) = 0.7$$
$$y(0.9) - 0.5y'(0.9) = 1$$

```
def p(x):
    return 2
def q(x):
    return -x
def f(x):
    return x ** 2
a, b = 0.6, 0.9
sigma = [0, 1]
gamma = [1, -0.5]
delta = [0.7, 1]
xs, ys = fdm(p, q, f, a, b, sigma, gamma, delta, 10)
plt.scatter(xs, ys, c='red', lw=2)
plt.savefig('hm1.png', quality=95, dpi=150)
```



Для предложенной задачи Wolfram не предлагает точного решения.

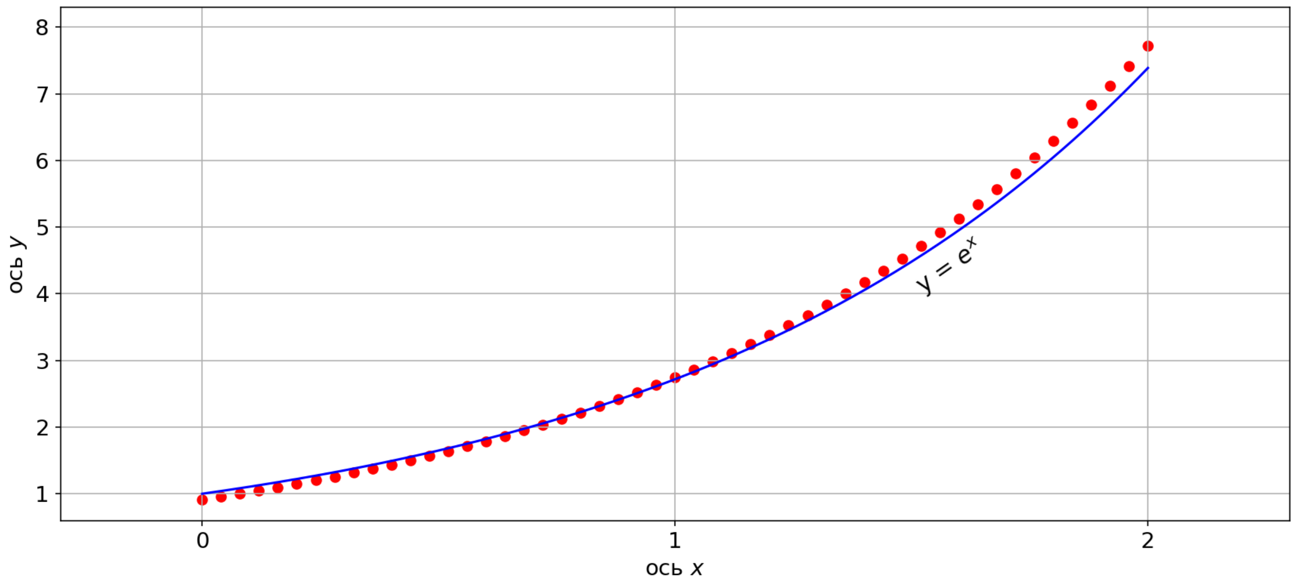
Попробуем протестировать алгоритм также на других условиях, например:

$$y' = y''$$
$$y(0) = y'(0)$$
$$y(2) - y'(2) = 0$$

Аналитическое решение - экспонента.

```
def p(x):
    return -1
```

```
def q(x):
    return 0
def f(x):
    return 0
a, b = 0, 2
sigma = [1, 1]
gamma = [1, -1]
delta = [2, 0]
xs, ys = fdm(p, q, f, a, b, sigma, gamma, delta, 50)
base = np.linspace(a, b, 200)
plt.figure(figsize=(14, 6))
plt.plot(base, np.exp(base), c='blue')
plt.scatter(xs, ys, c='red', lw=1)
plt.text(1.5, 4, 'y = $e^{x}$', fontsize=15, rotation=35)
plt.xlabel('ось $x$')
plt.ylabel('ось $y$')
plt.xticks(np.linspace(-1, 3, 5))
plt.yticks(np.linspace(0, 9, 10))
plt.xlim([-0.3, 2.3]) # лимиты по оси x
plt.ylim([0.6, 8.3])
plt.grid()
plt.savefig('exp.png', quality=95, dpi=150)
```

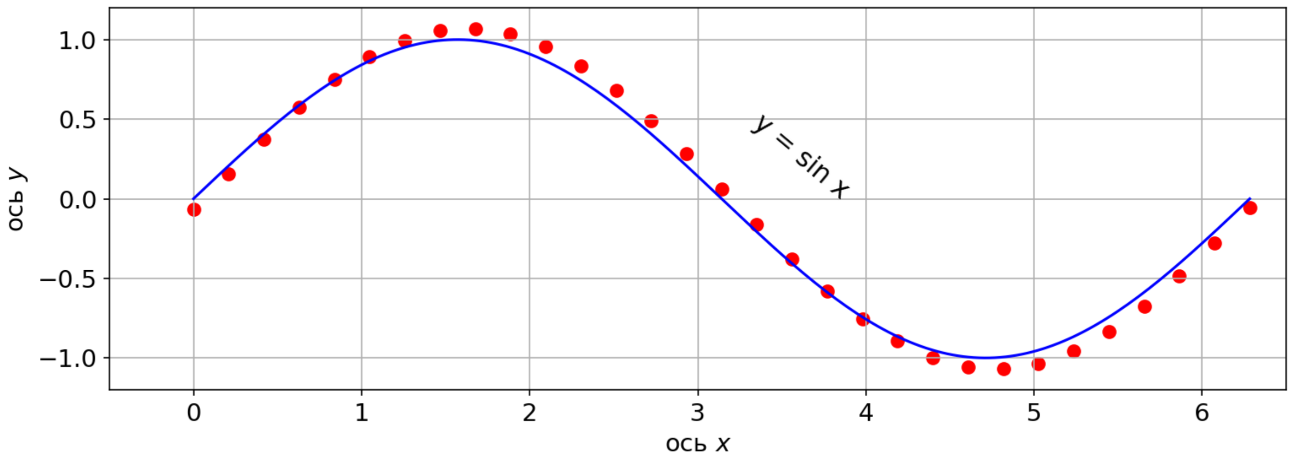


Попробуем классическое дифференциальное уравнение гармонических колебаний:

$$y'' = y$$
$$y(0) + y'(0) = 1$$
$$y(2\pi) - y'(2\pi) = 1$$

Точное решение — синус.

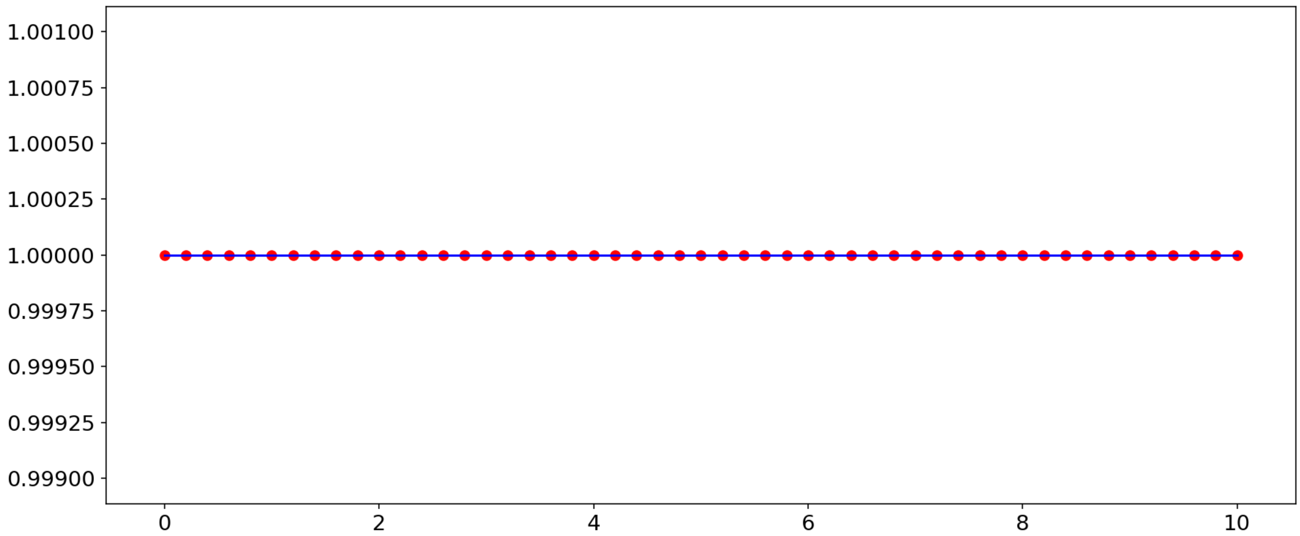
```
def p(x):
    return 0
def q(x):
    return 1
def f(x):
    return 0
a, b = 0, np.pi * 2
sigma = [1, 1]
gamma = [1, 1]
delta = [1, 1]
xs, ys = fdm(p, q, f, a, b, sigma, gamma, delta, 30)
base = np.linspace(a, b, 200)
plt.figure(figsize=(12,4))
plt.plot(base, np.sin(base), c='blue')
plt.scatter(xs, ys, c='red', lw=2)
plt.text(3.3, 0, 'y = sin x', fontsize=15, rotation=-40)
plt.xlabel('ось $x$') # подпись оси x
plt.ylabel('ось $y$') # подпись оси y
plt.xticks(np.linspace(0, 10, 11))
plt.yticks(np.linspace(-1, 1, 5))
plt.xlim([-0.5, 6.5])
plt.ylim([-1.2, 1.2])
plt.grid()
plt.savefig('sin.png', quality=95, dpi=150)
```



Наконец, рассмотрим вырожденный случай, когда решением является константа (единица):

$$y'' + y' = 0$$
$$y(0) + y'(0) = 1$$
$$y(10) + y'(10) = 1$$

```
def p(x):
    return 1
def q(x):
    return 0
def f(x):
    return 0
a, b = 0, 10
sigma = [1, 1]
gamma = [1, 1]
delta = [1, 1]
xs, ys = fdm(p, q, f, a, b, sigma, gamma, delta, 50)
base = np.linspace(a, b, 200)
plt.figure(figsize=(14, 6))
plt.plot(base, np.full(200, 1), c='blue')
plt.scatter(xs, ys, c='red', lw=1)
plt.savefig('const.png', quality=95, dpi=150)
```



2.5 Вывод

Итак, мы исследовали и реализовали метод конечных разностей, позволяющий численно находить решение обыкновенных дифференциальных уравнений второго порядка (краевая задача). Так же, как и при работе с методом Рунге — Кутта отмечена значимость плотности сетки.

Список литературы

[1] Костомаров Д. П. *Вводные лекции по численным методам: учебное пособие* / Д. П. Костомаров, А. П. Фаворский. — Москва: Логос, 2004.