Лабораторная работа №4

Содержание

Лабораторная работа 4.1 - Решение задачи Коши для ОДУ

Лабораторная работа 4.2 - Решение краевых задач

Лабораторная работа 4.1

Решение задачи Коши для ОДУ

Задача: Реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка в виде программ, задавая в качестве входных данных шаг сетки h. С использованием разработанного программного обеспечения решить задачу Коши для ОДУ 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

Условия:

Задача Коши:

$$xy'' - (x+1)y' + y = 0$$
$$y(1) = 2 + e$$
$$y'(1) = 1 + e$$
$$x \in [1, 2]$$

Точное решение:

$$y = x + 1 + e^x$$

Решение:

Неявный метод Эйлера. Если на правой границе интервала использовать точное значение производной от решения (т.е. тангенса угла наклона касательной), то получается неявный метод Эйлера первого порядка точности.
Метод Рунге-Кутты четвертого порядка является одним из самых широко используемых методов решения задачи Коши:

$$y_{k+1} = y_k + \Delta y_k$$
$$\Delta y_k = \frac{1}{6}(K_1^k + 2K_2^k + 2K_3^k + K_4^k)$$

$$K_1^k = hf(x_k, y_k)$$
$$K_2^k = hf(x_k + \frac{1}{2}h, y_k + \frac{1}{2}K_1^k)$$
$$K_3^k = hf(x_k + \frac{1}{2}h, y_k + \frac{1}{2}K_2^k)$$
$$K_4^k = hf(x_k + h, y_k + K_3^k)$$

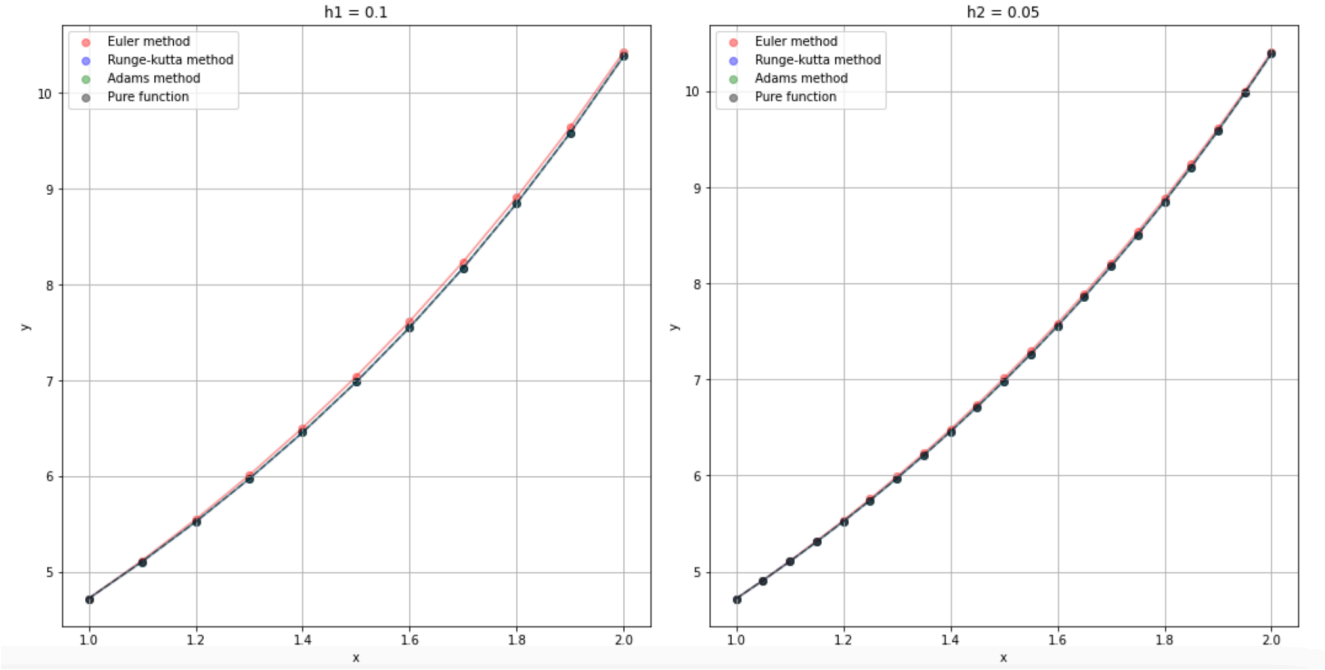
Метод Адамса. При использовании интерполяционного многочлена 3-ей степени построенного по значениям подынтегральной функции в последних четырех узлах получим метод Адамса четвертого порядка точности:

$$y_{k+1} = y_k + \frac{h}{24}(55f_k - 59f_{k-1} + 37f_{k-2} - 9f_{k-3}),$$

где f_k значение подынтегральной функции в узле x_k.

Работа программы:

```
Euler error: [0.0, 0.010969439112582968, 0.021592608379268796, 0.03149510815625689, 0.040249392168547615, 0.047366574
09505174, 0.05228739358762624, 0.054372192677730524, 0.0528897500690757, 0.047004812240608373, 0.035764147762641585]
Runge error: [0.0, 1.5857183655043627e-07, 3.5049795066299794e-07, 5.810401901840123e-07, 8.561982625110431e-07, 1.18
28067307817491e-06, 1.5686442624840424e-06, 2.022556612857329e-06, 2.5545950457228628e-06, 3.1761720542533567e-06, 3.
900236505316457e-06]
Adams error: [0.0, 1.5857183655043627e-07, 4.5664178927040666e-07, 9.415343322771719e-07, 9.009293043149569e-06, 1.96
92507272850435e-05, 3.182356341557835e-05, 4.6354770587342387e-05, 6.370631471597221e-05, 8.417825741346974e-05, 0.00
01082392013938005]
```



Листинг программы:

```
import numpy as np
import matplotlib.pyplot as plt

def function_of_xyz(x, y, z):
    return -y/x + (x+1)*z/x

def absolute_solve(x):
    return np.exp(x) + x + 1

# z = y'
def g(x, y, k):
```

```
    return k

# сумма квадратов отклонений
def sse(f, y):
    return round(sum([(f_i - y_i) ** 2 for f_i, y_i
in zip(f, y)]), 5)

# аналитическое решение
def analytical(f, a, b, h):
    x = [i for i in np.arange(a, b + h, h)]
    y = [f(i) for i in x]
    return x, y

def euler(f, a, b, h, y0, z):
    n = int((b - a) / h)
    x = [i for i in np.arange(a, b + h, h)]
    y = [y0]
    k = z
    for i in range(n):
        k += h * f(x[i], y[i], k)
        y.append(y[i] + h * g(x[i], y[i], k))

    return x, y

def runge_kutta(f, a, b, h, y0, z):
    n = int((b - a) / h)
    x = [i for i in np.arange(a, b + h, h)]
    y = [y0]
    k = [z]
    for i in range(n):
        k1 = h * g(x[i], y[i], k[i])
        l1 = h * f(x[i], y[i], k[i])
        k2 = h * g(x[i] + 0.5 * h, y[i] + 0.5 * k1,
k[i] + 0.5 * l1)
        l2 = h * f(x[i] + 0.5 * h, y[i] + 0.5 * k1,
k[i] + 0.5 * l1)
        k3 = h * g(x[i] + 0.5 * h, y[i] + 0.5 * k2,
k[i] + 0.5 * l2)
        l3 = h * f(x[i] + 0.5 * h, y[i] + 0.5 * k2,
k[i] + 0.5 * l2)
```

```
        k4 = h * g(x[i] + h, y[i] + k3, k[i] + l3)
        l4 = h * f(x[i] + h, y[i] + k3, k[i] + l3)
        y.append(y[i] + (k1 + 2 * k2 + 2 * k3 + k4) /
6)
        k.append(k[i] + (l1 + 2 * l2 + 2 * l3 + l4) /
6)
    return x, y, k

def adams(f, x, y, k, h):
    n = len(x)
    x = x[:4]
    y = y[:4]
    k = k[:4]
    for i in range(3, n - 1):
        k.append(k[i] + h * (55 * f(x[i], y[i], k[i])
-
                             59 * f(x[i - 1], y[i -
1], k[i - 1]) +
                             37 * f(x[i - 2], y[i -
2], k[i - 2]) -
                              9 * f(x[i - 3], y[i -
3], k[i - 3])) / 24)

        y.append(y[i] + h * (55 * g(x[i], y[i], k[i])
-
                             59 * g(x[i - 1], y[i -
1], k[i - 1]) +
                             37 * g(x[i - 2], y[i -
2], k[i - 2]) -
                              9 * g(x[i - 3], y[i -
3], k[i - 3])) / 24)
        x.append(x[i] + h)
    return x, y

def runge_romberg(dict_):
    k = dict_[0]['h'] / dict_[1]['h']
    y1 = [yi for xi, yi in zip(dict_[0]['Euler']
['x'], dict_[0]['Euler']['y']) if xi in dict_[1]
['Euler']['x']]
```

```
    y2 = [yi for xi, yi in zip(dict_[1]['Euler']
['x'], dict_[1]['Euler']['y']) if xi in dict_[0]
['Euler']['x']]
    euler = [y1 + (y2 - y1) / (k ** 2 - 1) for y1, y2
in zip(y1, y2)]
    x_ex = [xi for xi in dict_[0]['Euler']['x'] if xi
in dict_[1]['Euler']['x']]
    y_ex = [absolute_solve(i) for i in x_ex]
    for i in range(len(euler)):
        euler[i] = abs(euler[i] - y_ex[i])

    y1 = [yi for xi, yi in zip(dict_[0]['Runge']
['x'], dict_[0]['Runge']['y']) if xi in dict_[1]
['Runge']['x']]
    y2 = [yi for xi, yi in zip(dict_[1]['Runge']
['x'], dict_[1]['Runge']['y']) if xi in dict_[0]
['Runge']['x']]
    runge = [y1 + (y2 - y1) / (k ** 2 - 1) for y1, y2
in zip(y1, y2)]
    x_ex = [xi for xi in dict_[0]['Runge']['x'] if xi
in dict_[1]['Runge']['x']]
    y_ex = [absolute_solve(i) for i in x_ex]
    for i in range(len(runge)):
        runge[i] = abs(runge[i] - y_ex[i])

    y1 = [yi for xi, yi in zip(dict_[0]['Adams']
['x'], dict_[0]['Adams']['y']) if xi in dict_[1]
['Adams']['x']]
    y2 = [yi for xi, yi in zip(dict_[1]['Adams']
['x'], dict_[1]['Adams']['y']) if xi in dict_[0]
['Adams']['x']]
    adams = [y1 + (y2 - y1) / (k ** 2 - 1) for y1, y2
in zip(y1, y2)]
    x_ex = [xi for xi in dict_[0]['Adams']['x'] if xi
in dict_[1]['Adams']['x']]
    y_ex = [absolute_solve(i) for i in x_ex]
    for i in range(len(adams)):
        adams[i] = abs(adams[i] - y_ex[i])
```

```
    return {'Euler': euler, 'Runge': runge, 'Adams':
adams}

def show(res, pure, h):
    n = len(res)
    plt.figure(figsize=(18,9))
    for i in range(n):
        plt.subplot(1, n, i + 1)
        plt.subplots_adjust(wspace=0.1, hspace=0.6)
        plt.scatter(res[i]["Euler"]["x"], res[i]
["Euler"]["y"], color='r', alpha=0.4, label='Euler
method')
        plt.plot(res[i]["Euler"]["x"], res[i]
["Euler"]["y"], color='r', alpha=0.4)
        plt.scatter(res[i]["Runge"]["x"], res[i]
["Runge"]["y"], color='b', alpha=0.4, label='Runge-
kutta method')
        plt.plot(res[i]["Runge"]["x"], res[i]
["Runge"]["y"], color='b', alpha=0.4)
        plt.scatter(res[i]["Adams"]["x"], res[i]
["Adams"]["y"], color='g', alpha=0.4, label='Adams
method')
        plt.plot(res[i]["Adams"]["x"], res[i]
["Adams"]["y"], color='g', alpha=0.4)
        plt.scatter(pure[i][0], pure[i][1],
color='k', alpha=0.4, label='Pure function')
        plt.plot(pure[i][0], pure[i][1],
linestyle='dashed', color='k', alpha=0.4)

        plt.legend(loc='best')
        plt.title('h{0} = '.format(i + 1) +
str(h[i]))
        plt.xlabel('x')
        plt.ylabel('y')
        plt.grid(True)
    plt.savefig('Methods.png')
    plt.show()

"""Вариант 17:
```

Задача Коши:
$$
xy''-(x+1)y'+y=0\\
y(1)=2+e\\
y'(1)=1+e\\
x\in{[1,2]}\\
$$
Точное решение:
$$
y = x+1+e^x
$$
"""

```
a = 1
b = 2
y0 = 2 + np.exp(1)
z = 1 + np.exp(1)
h = 0.1

res = []
pure = []
steps = [h, h/2]
for h in steps:
    x_eul, y_eul = euler(function_of_xyz, a, b, h,
y0, z)
    x_rung, y_rung, k_rung =
runge_kutta(function_of_xyz, a, b, h, y0, z)
    x_ad, y_ad = adams(function_of_xyz, x_rung,
y_rung, k_rung, h)
    x_anal, y_anal = analytical(absolute_solve, a, b,
h)

    pure.append((x_anal, y_anal))
    res.append({
                "h": h,
                "Euler": {'x': x_eul, 'y': y_eul},
```

```
                    "Runge": {'x': x_rung, 'y': y_rung},
                    "Adams": {'x': x_ad, 'y': y_ad},
                    })

err = runge_romberg(res)
print("Euler error: {0}".format(err['Euler']))
print("Runge error: {0}".format(err['Runge']))
print("Adams error: {0}".format(err['Adams']))

show(res, pure, steps)
```

Лабораторная работа 4.2

Решение краевых задач

Задача: Реализовать метод стрельбы и конечно-разностный метод решения краевой задачи для ОДУ в виде программ. С использованием разработанного программного обеспечения решить краевую задачу для обыкновенного дифференциального уравнения 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

Условия:

Краевая задача:

$$(x^2 - 1)y'' + (x - 3)y' - y = 0$$
$$y'(0) = 0$$
$$y'(1) + y(1) = -0.75$$
$$x \in [0, 1]$$

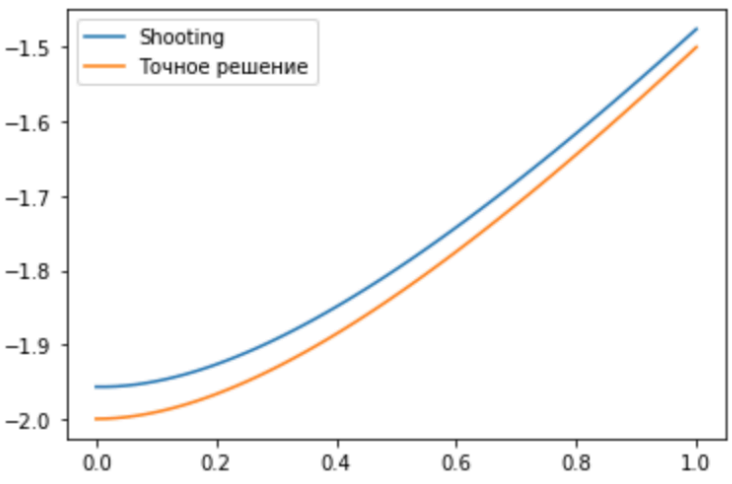
Точное решение:

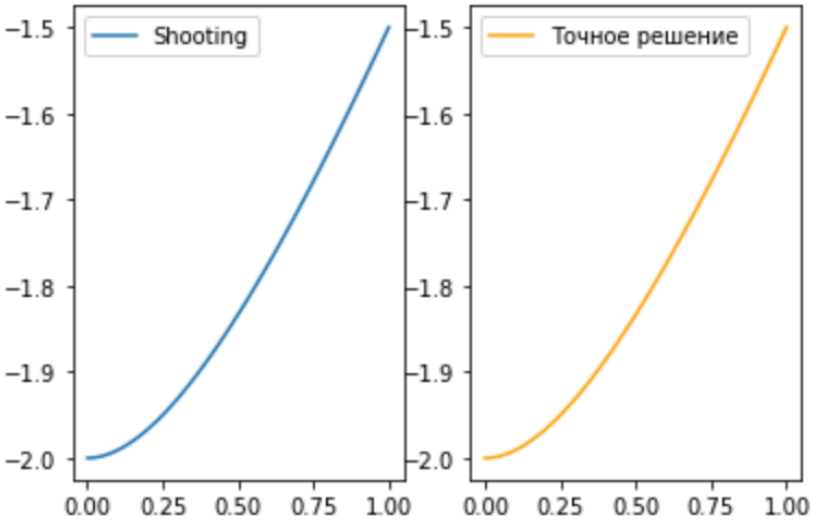
$$y = x - 3 + \frac{1}{x + 1}$$

Решение:

Метод стрельбы. Суть метода заключена в многократном решении задачи Коши для приближенного нахождения решения краевой задачи. Пусть надо решить краевую задачу на отрезке [a,b]. Вместо исходной задачи формулируется задача Коши с уравнением и с начальными условиями $y(a) = y_0$, $y'(b) = \eta$, где η - некоторое значение тангенса угла наклона касательной к решению в точке $x = a$. Положим сначала некоторое начальное значение параметру η=η0, после чего решим каким либо методом задачу Коши (4.28),(4.32). Пусть y = y0 (x, y0 ,η0) решение этой задачи на интервале [a,b], тогда сравнивая значение функции y0 (b, y0 ,η0) со значением y1 в правом конце отрезка можно получить информацию для корректировки угла наклона касательной к решению в левом конце отрезка. Решая задачу Коши для нового значения η=η1, получим другое решение со значением y1(b,y0,η1)на правом конце. Таким образом, значение решения на правом конце y(b, y0 ,η) будет являться функцией одной переменной η. Задачу можно сформулировать таким образом: требуется найти такое значение переменной η * , чтобы решение y(b, y ,η *) в правом конце отрезка совпало 0 со значением y1.

Работы программы:





All errors
Shooting method runge error: [0.0,
2.0774217901475822e-09, 4.074732329328867e-09,
5.993554541561252e-09, 7.835620596807757e-09,
9.602738382241682e-09, 1.1296768853696904e-08,
1.2919601388716728e-08, 1.447313824343155e-08,
1.5959278343302685e-08, 1.737990773520437e-08,
1.8736890261550343e-08, 2.003206112100031e-08,
2.1267217986675746e-08, 2.244412189433831e-08,
2.356449058105 17e-08, 2.4629996486780215e-08,
2.5642266754388743e-08, 2.6602880343062907e-08,
2.751336891648748e-08, 2.837521617671257e-08,
2.918985719801981e-08, 2.9958680647368396e-08,
3.0683027452127476e-08, 3.136419257643297e-08,
3.2003425687321396e-08, 3.260193293108671e-08,
3.316087471283424e-08, 3.3681369693283614e-08,
3.4164496121036336e-08, 3.461128916804057e-08,
3.502274470434940 7e-08, 3.539981929812086e-08,
3.574342977152867e-08, 3.6054454755074516e-08,
3.6333734465543444e-08, 3.658206870760239e-08,
3.6800219760380 06e-08, 3.698890838066404e-08,
3.7148813802900804e-08, 3.7280570408526614e-08,
3.7384763507120056e-08, 3.7461924895509924e-08,
3.751252442008024e-08, 3.7536960650896845e-08,
3.7535542229960583e-08, 3.7508466999014445e-08,
3.745578602831756e-08, 3.737735121411845e-08,
3.7272732900106575e-08, 3.7141083542024944e-08,
3.6980909001727014e-08, 3.6789661095326665e-08,

```
3.6562957994590306e-08, 3.6292988170316676e-08,
3.5964878630778685e-08, 3.554725003240833e-08,
3.496255196466791e-08, 3.3962650025642915e-08,
3.125755942967601e-08, 1.1102230246251565e-15]
Finite difference method runge error:
[0.03608370200272559, 0.03585570198888721,
0.03562890168366217, 0.0354029025866176,
0.03517735236740416, 0.03495193968146792 5,
0.03472638960521501, 0.03450045961047077,
0.03427393600929074, 0.034046630809750456,
0.03381837893144035, 0.03358903573632288,
0.03335847483652432, 0.0331265861456893 1,
0.03289327414485776 5, 0.0326584563375951 2,
0.03242206187227636 4, 0.0321840303122364 9,
0.03194431053687752, 0.0317028597589161 8,
0.03145964264474865, 0.0312146305264804 36,
0.03096780069553584, 0.0307191357689500 23,
0.03046862312048137 5, 0.0302162543695980 72,
0.02996202492218125 2, 0.0297059335574947 35,
0.02944798205656784 5, 0.0291881748676960 93,
0.02892651880522767 4, 0.0286630227782249 67,
0.02839769754595855 6, 0.0281305554975219 7,
0.02786161045313395 6, 0.0275908774849713 55,
0.02731837275557302 8, 0.0270441133720822 35,
0.02676811725475891 6, 0.0264904030183565 9,
0.02621098986509218 2, 0.0259298974880735 06,
0.02564714598414608, 0.0253627557752178 5,
0.02507674753721711 4, 0.0247891421358967 8,
0.02449996056876036, 0.0242092239124387 38,
0.02391695327487575 2, 0.0236231697516926 52,
0.02332789438610372, 0.0230311481317038 16,
0.02273295181735957, 0.0224333261132518 84,
0.02213229149677209 4, 0.0218298682163129 42,
0.02152607624970071, 0.0212209352511820 7,
0.02091446447415479 2, 0.0206066826377544 48,
0.02029760763534716]
```

Листинг программы:

```python
import math
import numpy as np
import matplotlib.pyplot as plt
```

```
def func(x, y, y_der):
    return - y_der*(x-3)/(x**2-1) + y/(x**2-1)

def g(x, y, k):
    return k

def p(x):
    return (x-3)/(x**2-1)

def q(x):
    return - 1/(x**2-1)

def absolute_solve(x):
    return x-3 + 1/(x+1)

def f(x):
    return 0

def stop(y, y1, eps):
    if abs(y[-1] - y1) > eps:
        return True
    else:
        return False

def runge_kutta(f, a, b, h, y0, z):
    n = int((b - a) / h)
    x = [i for i in np.arange(a, b + h, h)]
    y = [y0]
    k = [z]
    for i in range(n):
        k1 = h * g(x[i], y[i], k[i])
        l1 = h * f(x[i], y[i], k[i])
        k2 = h * g(x[i] + 0.5 * h, y[i] + 0.5 * k1,
k[i] + 0.5 * l1)
        l2 = h * f(x[i] + 0.5 * h, y[i] + 0.5 * k1,
k[i] + 0.5 * l1)
        k3 = h * g(x[i] + 0.5 * h, y[i] + 0.5 * k2,
k[i] + 0.5 * l2)
```

```
        l3 = h * f(x[i] + 0.5 * h, y[i] + 0.5 * k2,
k[i] + 0.5 * l2)
        k4 = h * g(x[i] + h, y[i] + k3, k[i] + l3)
        l4 = h * f(x[i] + h, y[i] + k3, k[i] + l3)
        y.append(y[i] + (k1 + 2 * k2 + 2 * k3 + k4) /
6)
        k.append(k[i] + (l1 + 2 * l2 + 2 * l3 + l4) /
6)
    return x, y, k

def newN(n_last, n, ans_last, ans, y1):
    x, y = ans_last[0], ans_last[1]
    phi_last = y[-1] - y1
    x, y = ans[0], ans[1]
    phi = y[-1] - y1
    return n - (n - n_last) / (phi - phi_last) * phi

def shooting_method(a, b, y0, y1, h, eps):
    n_last = 1
    n = 0.8
    y_der = n_last
    ans_last = runge_kutta(func, a, b, h, n_last,
y_der)[:2]
    y_der = n
    ans = runge_kutta(func, a, b, h, n, y_der)[:2]

    while stop(ans[1], y1, eps):
        n, n_last = newN(n_last, n, ans_last, ans,
y1), n
        ans_last = ans
        y_der = n
        ans = runge_kutta(func, a, b, h, y0, y_der)
[:2]

    return ans

def tma(a, b, c, d, shape):
    p = [-c[0] / b[0]]
    q = [d[0] / b[0]]
    x = [0] * (shape + 1)
```

```
    for i in range(1, shape):
        p.append(-c[i] / (b[i] + a[i] * p[i - 1]))
        q.append((d[i] - a[i] * q[i - 1]) / (b[i] +
a[i] * p[i - 1]))
    for i in reversed(range(shape)):
        x[i] = p[i] * x[i + 1] + q[i]
    return x[:-1]

def finite_difference(a, b, alpha, beta, delta,
gamma, y0, y1, h):
    n = int((b - a) / h)
    x = [i for i in np.arange(a, b + h, h)]
    A = [0] + [1 - p(x[i]) * h / 2 for i in range(0,
n - 1)] + [-gamma]
    B = [alpha * h - beta] + [q(x[i]) * h ** 2 - 2
for i in range(0, n - 1)] + [delta * h + gamma]

    C = [beta] + [1 + p(x[i]) * h / 2 for i in
range(0, n - 1)] + [0]
    D = [y0 * h] + [f(x[i]) * h ** 2 for i in
range(0, n - 1)] + [y1 * h]

    y = tma(A, B, C, D, len(A))
    return x, y

def runge_romberg(ans, exact):
    k = ans[0]['h'] / ans[1]['h']
    y1 = [yi for xi, yi in zip(ans[0]['Shooting']
['x'], ans[0]['Shooting']['y']) if xi in ans[1]
['Shooting']['x']]
    y2 = [yi for xi, yi in zip(ans[1]['Shooting']
['x'], ans[1]['Shooting']['y']) if xi in ans[0]
['Shooting']['x']]
    shoot_err = [y1 + (y2 - y1) / (k ** 2 - 1) for
y1, y2 in zip(y1, y2)]
    x_ex = [xi for xi in ans[0]['Shooting']['x'] if
xi in ans[1]['Shooting']['x']]
    y_ex = [absolute_solve(i) for i in x_ex]
    for i in range(len(shoot_err)):
        shoot_err[i] = abs(shoot_err[i] - y_ex[i])
```

```
    y1 = [yi for xi, yi in zip(ans[0]['FD']['x'],
ans[0]['FD']['y']) if xi in ans[1]['FD']['x']]
    y2 = [yi for xi, yi in zip(ans[1]['FD']['x'],
ans[1]['FD']['y']) if xi in ans[0]['FD']['x']]
    fd_err = [y1 + (y2 - y1) / (k ** 2 - 1) for y1,
y2 in zip(y1, y2)]
    x_ex = [xi for xi in ans[0]['FD']['x'] if xi in
ans[1]['FD']['x']]
    y_ex = [absolute_solve(i) for i in x_ex]
    for i in range(len(fd_err)):
        fd_err[i] = abs(fd_err[i] - y_ex[i])

    return {'Shooting': shoot_err, 'FD': fd_err}

def sse(f, y):
    return round(sum([(f_i - y_i) ** 2 for f_i, y_i
in zip(f, y)]), 5)

"""Вариант 17:


---


Краевая задача:
$$
(x^2-1)y''+(x-3)y'-y=0\\
y'(0)=0\\
y'(1)+y(1)=-0.75\\
x\in{[0,1]}\\
$$
Точное решение:
$$
y = x-3+\frac{1}{x+1}
$$
"""

# вариант 17
a = 0
```

```
b = 1
alpha = 0
delta = 1
gamma = 1
beta = 1
y0 = 0.0
y1 = -0.75

# для метода стрельбы создадим краевые условия
первого рода
y0_ = absolute_solve(0)
y1_ = absolute_solve(1)
step = 1 / 60
eps = 1e-5

print(f'Interval: [{a}, {b}]')
print(f'y0 = {y0}, y1 = {y1}')
print()

res = []
res2 = []
ans = []
steps = [step, step / 2]
i = 0
for h in steps:
    res.append(shooting_method(a, b, y0_, y1_, h,
eps))
    res2.append(finite_difference(a, b, alpha, beta,
delta, gamma, y0, y1, h))
    ans.append({
        "h": h,
        "Shooting": {'x': res[i][0],  'y': res[i]
[1]},
        "FD":       {'x': res2[i][0], 'y': res2[i]
[1]}
    })

    i += 1

abs_solve = []
```

```
for x_ in ans[0]['FD']['x']:
  abs_solve.append(absolute_solve(x_))
plt.plot(ans[0]['FD']['x'],ans[0]['FD']['y'])
plt.plot(ans[0]['FD']['x'],abs_solve)
plt.legend(['Shooting','Точное решение'])

abs_solve = []
for x_ in ans[1]['Shooting']['x']:
  abs_solve.append(absolute_solve(x_))

figure, ax = plt.subplots(1,2)
ax[0].plot(ans[1]['Shooting']['x'],ans[1]['Shooting']
['y'])
ax[1].plot(ans[1]['Shooting']
['x'],abs_solve,c="orange")
ax[0].legend(['Shooting'])
ax[1].legend(['Точное решение'])

exact = []
for h in steps:
    x_ex = [i for i in np.arange(a, b + h, h)]
    y_ex = [absolute_solve(i) for i in x_ex]
    exact.append((x_ex, y_ex))

err = runge_romberg(ans, exact)
print("All errors")
print('Shooting method runge error:
{}'.format(err['Shooting']))
print('Finite difference method runge error:
{}'.format(err['FD']))
```