Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Численные методы»

Студент:	П. С. Ветренко
Преподаватель:	И. Э. Иванов
Группа:	М8О-306Б
Вариант:	2
Дата:	
Оценка:	
Подпись:	

Москва, 2020

Лабораторная работа №4.1

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.

Задача Коши:

$y'' + y - 2\cos(x) = 0,$
$y(0) = 1,$
$y'(0) = 0,$
$x \in [0, 1], \quad h = 0.1$

Точное решение:

$y = x\sin(x) + \cos(x)$

1 Описание метода решения

Метод Эйлера
Один из методов решения ОДУ, достаточно простой, но имеет свой недостаток: при применении на практике имеет невысокую точность. Итерационная формула метода Эйлера:

$$y_{k+1} = y_k + hf(x_k, y_k).$$

Метод Рунге-Кутты
Запишем совокупность формул для семейства методов Рунге-Кутты:

$$y_{k+1} = y_k + \Delta y_k,$$

$$\Delta y_k = \sum_{i=1}^{p} c_i K_i^k,$$

$$K_i^k = hf(x_k + a_i h, \quad y_k + h\sum_{j=1}^{i-1} b_{ij} K_j^k), \quad i = 2, 3, \dots, p$$

Вычислять значения вспомогательных величин будем следующим образом:

$$K_1 = h * g(x[i], y[i], z[i])$$

$$L_1 = h * f(x[i], y[i], z[i])$$

$$K_2 = h * g(x[i] + 0.5 * h, y[i] + 0.5 * K1, z[i] + 0.5 * L1)$$

$$L_2 = h * f(x[i] + 0.5 * h, y[i] + 0.5 * K1, z[i] + 0.5 * L1)$$

$$K_3 = h * g(x[i] + 0.5 * h, y[i] + 0.5 * K2, z[i] + 0.5 * L2)$$

$$L_3 = h * f(x[i] + 0.5 * h, y[i] + 0.5 * K2, z[i] + 0.5 * L2)$$

$$K_4 = h * g(x[i] + h, y[i] + K3, z[i] + L3)$$

$$L_4 = h * f(x[i] + h, y[i] + K3, z[i] + L3)$$

Тогда можем найти $y[i+1]$ и $z[i+1]$:

$$y[i+1] = y[i] + \frac{(K1 + 2 * K2 + 2 * K3 + K4)}{6}$$

$$z[i+1] = z[i] + \frac{(L1 + 2 * L2 + 2 * L3 + L4)}{6}$$

Метод Адамса
Решение дифференциального уравнения удовлетворяет интегральному соотношению:

$$y_{k+1} = y_k + \int_{x_k}^{x_{k+1}} f(x, y(x))dx.$$

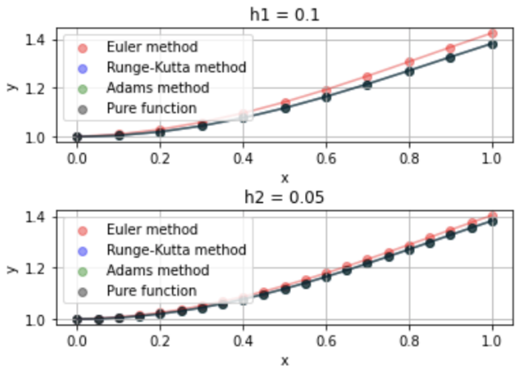
Если аппроксимировать подынтегральную функцию многочленом какой-либо степени, а затем вычислить интеграл, то можно получить формулу Адамса. При использовании интерполяционного многочлена 3 степени получим формулу Адамса 4 порядка:

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

2 Входные и выходные данные

```
Лабораторная работа 4.1
Вариант 2:

y(0) = 1
y'(0) = 0
x = [0, 1]
h = 0.1
Точное решение: x*sin(x)+cos(x)
Задача Коши: y'' + y - 2*cos(x) = 0

Текущий шаг: 0.1

Метод Эйлера:
x = 0.0, y = 1
x = 0.1, y = 1.01
x = 0.2, y = 1.0298000833055605
x = 0.3, y = 1.0589034973348903
x = 0.4, y = 1.0965246061733833
x = 0.5, y = 1.1416016888302
x = 0.6, y = 1.1928144058365224
x = 0.7, y = 1.248605691082673
x = 0.8, y = 1.3072077631636867
x = 0.9, y = 1.3666718918000067
x = 1.0, y = 1.42490150088374


Метод Рунге-Кутта:
x = 0.0, y = 1
x = 0.1, y = 1.0049875017359664
x = 0.2, y = 1.0198004024992113
x = 0.3, y = 1.0439924431189946
x = 0.4, y = 1.076828128023344
x = 0.5, y = 1.1172950077157338
x = 0.6, y = 1.164120632787752
x = 0.7, y = 1.215793941746397
x = 0.8, y = 1.2705907822889453
x = 0.9, y = 1.326603206879753
x = 1.0, y = 1.3817721293338139

Метод Адамса:
x = 0.0, y = 1
x = 0.1, y = 1.0049875017359664
x = 0.2, y = 1.0198004024992113
x = 0.3, y = 1.0439924431189946
x = 0.4, y = 1.076824857465474
x = 0.5, y = 1.1172829515822182
x = 0.6, y = 1.1640988057708541
x = 0.7, y = 1.2157589785000067
x = 0.8, y = 1.2705401412068373
x = 0.9, y = 1.3265350469093606
x = 1.0, y = 1.3816848544271656


Аналитический метод:
x = 0.0, y = 1.0
x = 0.1, y = 1.0049875069427086
x = 0.2, y = 1.019800444000254
x = 0.3, y = 1.0439925511240078
x = 0.4, y = 1.0768283309263453
x = 0.5, y = 1.1172953311924743
x = 0.6, y = 1.1641210989466995
x = 0.7, y = 1.2157945683508722
x = 0.8, y = 1.2705915820667837
x = 0.9, y = 1.3266041869353995
x = 1.0, y = 1.381773290676O363

Текущий шаг: 0.05

Метод Эйлера:
x = 0.0, y = 1
x = 0.05, y = 1.0025
x = 0.1, y = 1.0074875013019748
x = 0.15, y = 1.014931304677085
x = 0.2, y = 1.0247816351801824
x = 0.25, y = 1.0369703444845357
x = 0.3, y = 1.0514111900362308
x = 0.35, y = 1.0680001900584635
x = 0.4, y = 1.0866160531697868
x = 0.45, y = 1.1071206811182002
x = 0.5, y = 1.1293597428755813
x = 0.55, y = 1.1531633180852254
x = 0.6, y = 1.178346607609954
x = 0.65, y = 1.204710708690206
x = 0.7, y = 1.2320434519914778
x = 0.75, y = 1.2601202975991934
x = 0.8, y = 1.2887052868072801
x = 0.85, y = 1.3175520463450845
x = 0.9, y = 1.3464048414964511
x = 0.95, y = 1.3749996743854298
x = 1.0, y = 1.4030654235357645

Метод Рунге-Кутта:
x = 0.0, y = 1
x = 0.05, y = 1.0012492187771262
x = 0.1, y = 1.0049875062923144
x = 0.15, y = 1.0111867961034626
x = 0.2, y = 1.0198004407663686
x = 0.25, y = 1.0307634062939368
x = 0.3, y = 1.0439925434455168
x = 0.35, y = 1.0593869348964966
x = 0.4, y = 1.0768283170721344
x = 0.45, y = 1.0961815751673691
x = 0.5, y = 1.1172953096168645
x = 0.55, y = 1.140002472027643
x = 0.6, y = 1.1641210683411622
x = 0.65, y = 1.189454926753374
x = 0.7, y = 1.215794527690952
x = 0.75, y = 1.242917892920238
x = 0.8, y = 1.2705915306532372
x = 0.85, y = 1.2985714333129046
x = 0.9, y = 1.326604124428651
x = 0.95, y = 1.3544277509530933
x = 1.0, y = 1.3817732171231707


Метод Адамса:
x = 0.0, y = 1
x = 0.05, y = 1.0012492187771262
x = 0.1, y = 1.0049875062923144
x = 0.15, y = 1.0111867961034626
x = 0.2, y = 1.0198003892541205
x = 0.25, y = 1.0307632149117598
x = 0.3, y = 1.0439921916586006
x = 0.35, y = 1.0593863614874168
x = 0.4, y = 1.0768274717651924
x = 0.45, y = 1.0961804105595792
x = 0.5, y = 1.117293779451764
x = 0.55, y = 1.140000532885337
x = 0.6, y = 1.1641186792812677
x = 0.65, y = 1.1894520497463112
x = 0.7, y = 1.2157911278632094
x = 0.75, y = 1.2429139387743757
x = 0.8, y = 1.2705869942988037
x = 0.85, y = 1.2985662906693989
x = 0.9, y = 1.3265983554144996
x = 0.95, y = 1.3544213396563758
x = 1.0, y = 1.3817661519532642
```

```
Аналитический метод:
x = 0.0, y = 1.0
x = 0.05, y = 1.0012492188585003
x = 0.1, y = 1.0049875069427086
x = 0.15, y = 1.011186797807082
x = 0.2, y = 1.019800444000254
x = 0.25, y = 1.0307634115242754
x = 0.3, y = 1.0439925511240078
x = 0.35, y = 1.0593869454567868
x = 0.4, y = 1.0768283309263453
x = 0.45, y = 1.0961815927027305
x = 0.5, y = 1.1172953311924743
x = 0.55, y = 1.1400024979713683
x = 0.6, y = 1.1641210989466995
x = 0.65, y = 1.1894549622774817
x = 0.7, y = 1.2157945683508722
x = 0.75, y = 1.2429179388913214
x = 0.8, y = 1.2705915820667837
x = 0.85, y = 1.298571490254231
x = 0.9, y = 1.3266041869353995
x = 0.95, y = 1.3544278190137886
x = 1.0, y = 1.3817732906760363

Погрешность метода Эйлера: [0.0, 0.004174993491283141, 0.00832682326351386, 0.012413510444662768, 0.01639342424583
9228, 0.02022570898618614, 0.023870707480966757, 0.02729037636806919, 0.030448688978100824, 0.0333120214300886, 0.
035849517758378635]
Погрешность метода Рунге-Кутты: [0.0, 3.687959493348103e-09, 2.8745323454870686e-08, 7.45628390141917e-08, 1.39886
73774889548e-07, 2.2284303025088548e-07, 3.209744774412826e-07, 4.3128962357030787e-07, 5.503230744441368e-07, 6.7
42060136843975e-07, 7.987457701918999e-07]
Погрешность метода Адамса: [0.0, 3.687959493348103e-09, 4.591607294379685e-08, 1.566867237423608e-05]
```



3 Исходный код программы

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


target = '''
y(0) = 1
y'(0) = 0
x = [0,1]
h = 0.1
Точное решение: x*sin(x)+cos(x)
Задача Коши: y''+ y -2*cos(x) = 0
'''

def cauchyProblem(x,y,y_der):
return 2*math.cos(x)-y

def pureFunction(x):
return x*math.sin(x)+math.cos(x)

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

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

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,y_der):
n = int((b -a) / h)
x = [i for i in np.arange(a,b + h,h)]
y = [y0]
k = y_der
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 rungeKutt(f,a,b,h,y0,y_der):
n = int((b -a) / h)
x = [i for i in np.arange(a,b + h,h)]
y = [y0]
k = [y_der]
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 rungeRomberg(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 = [pureFunction(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 = [pureFunction(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 = [pureFunction(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)
for i in range(n):
plt.subplot(n,1,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],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()

if __name__ == '__main__':
print('\nЛабораторная работа 4.1\nВариант 2:')
a = 0
b = 1
h = 0.1
y0 = 1
y_der = 0
print(target)
res = []
pure = []
steps = [h,h/2]
for h in steps:
print(f"Текущий шаг: {h}")
print("\nМетод Эйлера:")
x_eul,y_eul = euler(cauchyProblem,a,b,h,y0,y_der)
for x,y in zip(x_eul,y_eul):
print(f'x = {round(x,4)},y = {y}')
print()

print("\nМетод Рунге-Кутта:")
x_rung,y_rung,k_rung = rungeKutt(cauchyProblem,a,b,h,y0,y_der)
for x,y in zip(x_rung,y_rung):
print(f'x = {round(x,4)},y = {y}')
print()
```

```
print("\nМетод Адамса:")
x_ad,y_ad = adams(cauchyProblem,x_rung,y_rung,k_rung,h)
for x,y in zip(x_ad,y_ad):
print(f'x = {round(x,4)},y = {y}')
print()

print("\nАналитический метод:")
x_anal,y_anal = analytical(pureFunction,a,b,h)
for x,y in zip(x_anal,y_anal):
print(f'x = {round(x,4)},y = {y}')
print()

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 = rungeRomberg(res)
print("Погрешность метода Эйлера: {0}".format(err['Euler']))
print("Погрешность метода Рунге-Кутты: {0}".format(err['Runge']))
print("Погрешность метода Адамса: {0}".format(err['Adams']))

show(res,pure,steps)
```

Лабораторная работа №4.2

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.

Краевая задача:

$xy^{''} + 2y^{'} - xy = 0,$
$y(1) = \exp^{-1},$
$y(2) = 0.5\exp^{-2}$

Точное решение:

$y(x) = \frac{\exp^{-x}}{x}$

1 Описание методов решения

Метод стрельбы
Если исходная задача Коши имеет вид:

$$\begin{cases} y'' = f(x, y, y') \\ y(a) = y_0 \\ y(b) = y_1 \end{cases}$$

тогда сделаем замену начальных условий на:
$y(a) = y_0$
$y'(b) = \eta$
Необходимо найти η такое, чтобы решение $y(b, y_0, \eta)$ в правом конце отрезка совпало со значением y_1. Это эквивалентно нахождению корня уравнения:

$$\Phi(\eta) = y(b, y_0, \eta) - y_1 = 0.$$

Уравнение решается по итерационной формуле:

$$\eta_{j+2} = \eta_{j+1} - \frac{\eta_{j+1} - \eta_j}{\Phi(\eta_{j+1}) - \Phi(\eta_j)} \Phi(\eta_{j+1})$$

Итерационный процесс прекращается по достижению заданной точности.
Конечно-разностный метод
Если исходная краевая задача имеет вид:
$y'' + p(x)y' + q(x)y = f(x)$,
$y(a) = y_0, y(b) = y_1$,
разобьём отрезок $[a, b]$ на интервалы и аппроксимируем производные:

$$y_k' = \frac{y_{k+1} - y_{k-1}}{2h} + O(h^2);$$

$$y_k'' = \frac{y_{k+1} - 2y_k + y_{k-1}}{h^2} + O(h^2);$$

Приводя подобные и подставив аппроксимации, мы получаем систему линейных уравнений с трехдиагональной матрицей. Как известно из прошлых лабораторных работ, такую систему удобно решить методом прогонки:

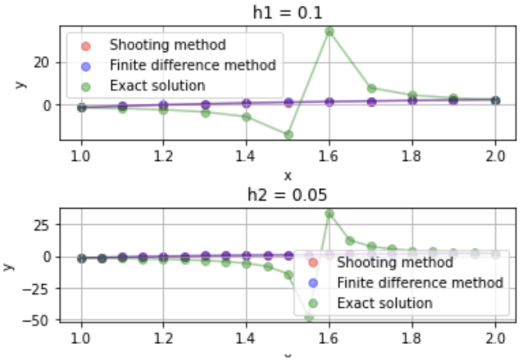
$$\begin{cases} (-2 + h^2 q(x_1))y_1 + (1 + \frac{p(x_1)h}{2})y_2 = h^2 f(x_1) - (1 - \frac{p(x_1)h}{2})y_a \\ (1 - \frac{p(x_k)h}{2})y_{k-1} + (-2 + h^2 q(x_k))y_k + (1 + \frac{p(x_k)h}{2})y_{k+1} = h^2 f(x_k), \quad k = 2, \dots, n-2 \\ (1 - \frac{p(x_{n-1})h}{2})y_{n-1} + (-2 + h^2 q(x_{n-1}))y_{n-1} = h^2 f(x_{n-1}) - (1 + \frac{p(x_{n-1})h}{2})y_b \end{cases}$$

2 Входные и выходные данные

```
Лабораторная работа 4.2
Вариант 2:

Краевая задача: x*y'' + 2*y' -x*y = 0
y(1) = exp^(-1)
y(2) = exp^(-2)/2
Точное решение: y(x) = e^(-x)/x

Интервал: [1, 2]
y0 = -1.557407724654902, y1 = 2.185039863261519

Текущий шаг: 0.1

Метод стрельбы:
x: 1.0, y: -1.55741
x: 1.1, y: -0.8981
x: 1.2, y: -0.35691
x: 1.3, y: 0.09773
x: 1.4, y: 0.48834
x: 1.5, y: 0.83142
x: 1.6, y: 1.13943
x: 1.7, y: 1.42193
x: 1.8, y: 1.68648
x: 1.9, y: 1.93918
x: 2.0, y: 2.18504


Конечно-разностный метод:
x: 1.0, y: -1.55741
x: 1.1, y: -0.8201
x: 1.2, y: -0.22431
x: 1.3, y: 0.26092
x: 1.4, y: 0.66077
x: 1.5, y: 0.99573
x: 1.6, y: 1.28246
x: 1.7, y: 1.53454
x: 1.8, y: 1.76312
x: 1.9, y: 1.97738
x: 2.0, y: 2.18504
```

```
Текущий шаг: 0.05

Метод стрельбы:
x: 1.0, y: -1.55741
x: 1.05, y: -1.21053
x: 1.1, y: -0.89808
x: 1.15, y: -0.61495
x: 1.2, y: -0.35688
x: 1.25, y: -0.12032
x: 1.3, y: 0.09776
x: 1.35, y: 0.29992
x: 1.4, y: 0.48836
x: 1.45, y: 0.66498
x: 1.5, y: 0.83144
x: 1.55, y: 0.98917
x: 1.6, y: 1.13944
x: 1.65, y: 1.28337
x: 1.7, y: 1.42194
x: 1.75, y: 1.55605
x: 1.8, y: 1.68649
x: 1.85, y: 1.81398
x: 1.9, y: 1.93918
x: 1.95, y: 2.06268
x: 2.0, y: 2.18504


Конечно-разностный метод:
x: 1.0, y: -1.55741
x: 1.05, y: -1.17013
x: 1.1, y: -0.82252
x: 1.15, y: -0.51007
x: 1.2, y: -0.22871
x: 1.25, y: 0.02522
x: 1.3, y: 0.25504
x: 1.35, y: 0.46374
x: 1.4, y: 0.65399
x: 1.45, y: 0.82822
x: 1.5, y: 0.98863
x: 1.55, y: 1.13717
x: 1.6, y: 1.27562
x: 1.65, y: 1.4056
x: 1.7, y: 1.52855
x: 1.75, y: 1.64579
x: 1.8, y: 1.75854
x: 1.85, y: 1.86787
x: 1.9, y: 1.97479
x: 1.95, y: 2.08023
x: 2.0, y: 2.18504
```

```
Погрешности
Погрешности метода стрельбы: [0.0, 1.0666656062445827, 2.215252544905908, 3.6998439571811135, 6.286227649796147, 1
4.932848993523999, 33.09310220728563, 6.274672028759597, 2.599779893002403, 0.9879208136261444, 2.220446049250313e
-15]
Погрешности конечно-разностного метода: [2.220446049250313e-16, 1.1438534501033089, 2.346377144894384, 3.861063688
925025, 6.456393123266517, 15.09478110224306, 32.95235394270327, 6.164057515383285, 2.524672623285572, 0.950581804
8090597, 4.884981308350689e-15]
```



3 Исходный код программы

```
import math
import numpy as np
import matplotlib.pyplot as plt
from NM_4_1 import rungeKutt


target = '''
Краевая задача: x*y''+ 2*y'-x*y = 0
y(1) = exp^(-1)
y(2) = exp^(-2)/2
Точное решение: y(x) = e^(-x)/x
'''

def func(x,y,y_der):
return (x*y-2*y_der)/x

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

def p(x):
return 2

def q(x):
return -x

def pureFunction(x):
return -math.tan(x)

def f(x):
return 0

def first(x,y,x0):
i = 0
while i <len(x) -1 and x[i + 1] <x0:
i += 1
return (y[i + 1] -y[i]) / (x[i + 1] -x[i])

def stop(y,y1,eps):
if abs(y[-1] -y1) >eps:
```

```
return True
else:
return False

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

while stop(ans[1],y1,eps):
n,n_last = newN(n_last,n,ans_last,ans,b,y1),n
ans_last = ans
y_der = n
ans = rungeKutt(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 finiteDifferenceMethod(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 show(ans,exact,h):
n = len(ans)
for i in range(n):
plt.subplot(n,1,i + 1)
plt.subplots_adjust(wspace=0.1,hspace=0.6)
plt.scatter(ans[i]["Shooting"]["x"],ans[i]["Shooting"]["y"],color='r',alpha=0.4,label=
method')
plt.plot(ans[i]["Shooting"]["x"],ans[i]["Shooting"]["y"],color='r',alpha=0.4)
plt.scatter(ans[i]["FD"]["x"],ans[i]["FD"]["y"],color='b',alpha=0.4,label='Finite
difference method')
plt.plot(ans[i]["FD"]["x"],ans[i]["FD"]["y"],color='b',alpha=0.4)
plt.scatter(exact[i][0],exact[i][1],color='g',alpha=0.4,label='Exact solution')
plt.plot(exact[i][0],exact[i][1],color='g',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('ShootAndFD.png')
plt.show()

def rungeRomberg(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 = [pureFunction(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'][
Y2 = [yi for xi,yi in zip(ans[1]['FD']['x'],ans[1]['FD']['y']) if xi in ans[0]['FD'][
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 = [pureFunction(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)

if __name__ == '__main__':
print('\nЛабораторная работа 4.2\nВариант 2:')
print(target)

a = 1
b = 2
alpha = 1
delta = 1
gamma = 0
beta = 0
y0 = pureFunction(a)
y1 = pureFunction(b)
step = 0.1
eps = 1e-5

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

res = []
res2 = []
ans = []
steps = [step,step / 2]
i = 0
```

```
for h in steps:
print(f'Текущий шаг: {h}')
print('\nМетод стрельбы:')
res.append(shootingMethod(a,b,y0,y1,h,eps))
for x,y in zip(res[i][0],res[i][1]):
print(f'x: {round(x,5)},y: {round(y,5)}')
print()

print('\nКонечно-разностный метод:')
res2.append(finiteDifferenceMethod(a,b,alpha,beta,delta,gamma,y0,y1,h))
for x,y in zip(res2[i][0],res2[i][1]):
print(f'x: {round(x,5)},y: {round(y,5)}')
print()

ans.append({
"h": h,
"Shooting": {'x': res[i][0], 'y': res[i][1]},
"FD":       {'x': res2[i][0],'y': res2[i][1]}
})

i += 1

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

err = rungeRomberg(ans,exact)
print("Погрешности")
print('Погрешности метода стрельбы: {}'.format(err['Shooting']))
print('Погрешности конечно-разностного метода: {}'.format(err['FD']))
show(ans,exact,steps)
```

4 Выводы

Для решения задачи данной лабораторной работы необходимо было вспомнить курс дифференциальных уравнений. Затем, изучив новые методы решения ОДУ, я реализовала программы, которые находят решения этих уравнений.
Также потребовались знания из предыдущих лабораторных работ, что показывает взаимосвязь между разделами курса численных методов.