Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОССУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОННИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №7

По дисциплине «Методы численного анализа»

По теме «Методы численного интегрирования и дифференцирования»

Выполнил:

студент гр. 653504

Куликов А.Д.

Проверил:

Пашук А. В.

Минск 2018

**Задание.**

1. С помощью численных методов найти значение интеграла
2. Найти численное решение задачи Коши

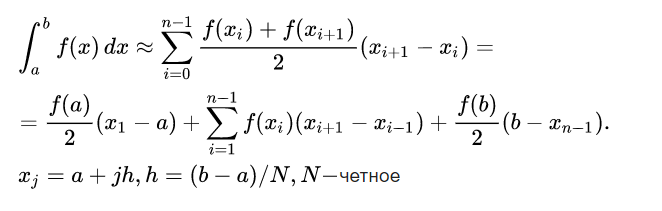
**Решение.**

1. **Вычисление шага интегрирования по формуле трапеций с точностью 0.001**

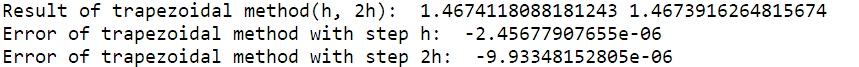
С помощью вещественного двоичного поиска найдем необходимое значение шага интегрирования, а затем пересчитаем с учетом дополнительного условия (n должно делиться нацело на 4):



1. **Вычисление интеграла по формуле трапеций с шагом h и 2h. Дать уточненную оценку погрешности.**

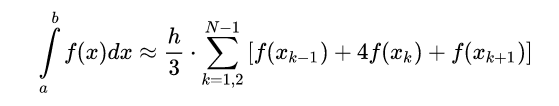
Формула трапеций: 

Оценка погрешности:

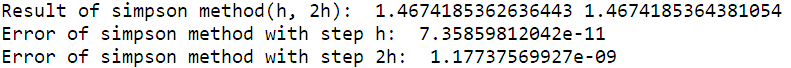


1. **Вычисление интеграла по формуле Симпсона с шагом h и 2h. Дать уточненную оценку погрешности.**

Формула Симпсона:



Оценка погрешности:



1. **Вычислить определенный интеграл по формуле Ньютона-Лейбница.**

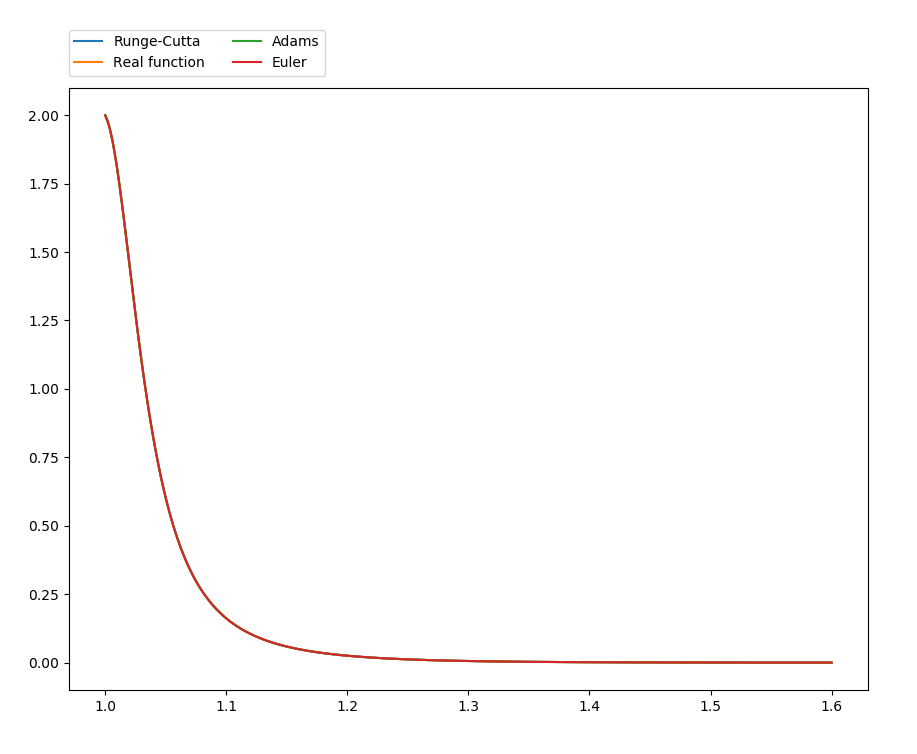
Обе формулы интегрирования дали достаточно хорошее приближение. Формула Симпсона при одинаковом количестве шагов дала большую точность, чем формула трапеций.

1. **Найти шаг интегрирования для решения задачи Коши методом Рунге-Кутта(4) с точностью 10-4** .

Выберем . Далее будем уменьшать h, пока не выполняется условие .



**Графики интегральных кривых**



**8. Точное решение задачи Коши.**

Имеем дифференциальное уравнение Бернулли. Воспользуемся подстановкой Бернулли для нахождения решения дифференциального уравнения:

Подставив начальные условия, получим

Сводные таблицы можно найти в файлах Excel.

**Приложение. Исходный код программы**

import numpy as np

import sympy as sp

from matplotlib import pyplot as plt

import xlsxwriter

def f(x):

return x\*\*2 / (x + 1)\*\*2

def g(x, y):

return 4 \* y\*\*2 \* np.exp(4\*x) \* (1 - x\*\*3) - 4 \* x\*\*3 \* y

def d2f(x):

return 2 / (x + 1)\*\*2 - 8 \* x / (x + 1) \*\* 3 + 6 \* x\*\*2 / (x + 1)\*\*4

def trapezoidal\_rule(a, b, f, h=0.001):

res = f(a) + f(b)

for i in range(1, int((b - a) / h)):

res += 2 \* f(a + h \* i)

return res \* h / 2

def find\_h\_trap(a, b, f, e=0.001):

h\_good = 0.0001

good\_res = trapezoidal\_rule(a, b, f, h\_good)

h = 1

res = trapezoidal\_rule(a, b, f, h)

while abs(res - good\_res) > e:

h = h / 2

res = trapezoidal\_rule(a, b, f, h)

if abs(res - good\_res) < e:

break

if res < good\_res:

h += h / 2

else:

h -= h / 2

return h

def trapezoidal\_error(a, b, d2f, h):

m2 = np.max([d2f(x) for x in np.arange(a, b, h)])

return h\*\*2 \* (b - a) / 12 \* m2

def simpson(a, b, f, h):

res = 0

n = int((b - a) / h)

for i in range(1, n, 2):

res += f(a + h \* (i - 1)) + 4 \* f(a + h \* i) + f(a + h \* (i + 1))

res \*= h / 3

return res

def recalculate\_h(a, b, h):

n = int((b - a) / h)

while n % 4:

n += 1

return (b - a) / n

def simpson\_error(a, b, d4f, h):

m4 = np.max([abs(d4f(x)) for x in np.arange(a, b, h)])

return (b - a) \* h\*\*4 / 2880 \* m4

def runge\_kutta4(a, b, g, h, start\_point):

res = [start\_point]

n = int((b - a - h / 2) / h) + 1

for k in range(1, n + 1):

xk = res[k - 1][0]

yk = res[k - 1][1]

f1 = g(xk, yk)

f2 = g(xk + h / 2, yk + h / 2 \* f1)

f3 = g(xk + h / 2, yk + h / 2 \* f2)

f4 = g(xk + h, yk + h \* f3)

res.append((xk + h, yk + h / 6 \* (f1 + 2 \* f2 + 2 \* f3 + f4)))

return res

def euler(a, b, g, h, start\_point):

res = [start\_point]

n = int((b - a - h / 2) / h) + 1

for k in range(1, n + 1):

xk = res[k - 1][0]

yk = res[k - 1][1]

res.append((xk + h, yk + h / 2 \* (g(xk, yk) + g(xk + h, yk + h \* g(xk, yk)))))

return res

def right(x):

return 2 \* np.exp(1) / (np.exp(x\*\*4) + 2 \* np.exp(x\*\*4+4) - 2 \* np.exp(4\*x+1))

def find\_diff\_h(a, b, g, start\_point, e=0.0001, method='runge'):

h0 = np.power(e, 1 / 4)

n = int((b - a) / h0)

if n % 2:

n += 1

h0 = (b - a) / n

y2 = 1

y1 = 0

while 1 / 15 \* abs(y2 - y1) >= e:

if method == 'runge':

res1 = runge\_kutta4(a, a + 2 \* h0, g, h0, start\_point)

res2 = runge\_kutta4(a, a + 2 \* h0, g, 2 \* h0, start\_point)

elif method == 'adams':

res1 = adams(a, a + 2 \* h0, g, h0, start\_point)

res2 = adams(a, a + 2 \* h0, g, 2 \* h0, start\_point)

y1 = res1[-1][1]

y2 = res2[-1][1]

h0 /= 2

return h0

def adams(a, b, g, h, start\_point):

n = int((b - a - h / 2) / h) + 1

x1 = a + h

y1 = runge\_kutta4(a, x1, g, h, start\_point)[-1][1]

res = [start\_point, (x1, y1)]

for k in range(2, n + 1):

xk = res[k - 1][0]

yk = res[k - 1][1]

xkk = res[k - 2][0]

ykk = res[k - 2][1]

res.append((xk + h, yk + h / 2 \* (3 \* g(xk, yk) - g(xkk, ykk))))

return res

def save\_data(file\_name, data):

workbook = xlsxwriter.Workbook(file\_name)

worksheet = workbook.add\_worksheet()

row = 0

for col, data in enumerate(data):

worksheet.write\_column(row, col, data)

workbook.close()

def prepare\_data(pts\_h, pts\_2h):

n = len(pts\_2h)

data = [np.zeros(n), np.zeros(n), np.zeros(n), np.zeros(n)]

data[0] = [t[0] for t in pts\_2h]

data[1] = [pts\_h[i][1] for i in range(0, len(pts\_h), 2)]

data[2] = [t[1] for t in pts\_2h]

data[3] = np.abs([data[2][i] - data[1][i] for i in range(n)])

return data

def result\_table(sol, pts\_runge, pts\_adams):

n = min(len(pts\_runge), len(pts\_adams) // 2)

pts\_runge\_resized = pts\_runge[:n]

pts\_adams\_resized = pts\_adams[::2]

pts\_adams\_resized = pts\_adams\_resized[:n]

x = [t[0] for t in pts\_runge\_resized]

y\_runge = [t[1] for t in pts\_runge\_resized]

y\_adams = [t[1] for t in pts\_adams\_resized]

y\_true = [sol(t) for t in x]

data = [np.zeros(n), np.zeros(n), np.zeros(n), np.zeros(n), np.zeros(n), np.zeros(n)]

data[0] = x

data[1] = y\_true

data[2] = y\_runge

data[3] = np.abs([data[2][i] - data[1][i] for i in range(n)])

data[4] = y\_adams

data[5] = np.abs([data[4][i] - data[1][i] for i in range(n)])

return data

def main():

x = sp.Symbol('x')

y = x\*\*2 / (x + 1)\*\*2

a = 1

b = 4

h = find\_h\_trap(a, b, f)

print('Step size for trapezoidal method: ', h)

h = recalculate\_h(a, b, h)

print('Recalculated step size: ', h)

res\_trapezoidal\_h = trapezoidal\_rule(a, b, f, h)

res\_trapezoidal\_2h = trapezoidal\_rule(a, b, f, 2 \* h)

# print(res\_trapezoidal, h, trapezoidal\_rule(a, b, f, h))

trap\_error\_h = trapezoidal\_error(a, b, d2f, h)

trap\_error\_2h = trapezoidal\_error(a, b, d2f, 2\*h)

print('Result of trapezoidal method(h, 2h): ', res\_trapezoidal\_h, res\_trapezoidal\_2h)

print('Error of trapezoidal method with step h: ', trap\_error\_h)

print('Error of trapezoidal method with step 2h: ', trap\_error\_2h)

res\_simpson\_h = simpson(a, b, f, h)

res\_simpson\_2h = simpson(a, b, f, 2 \* h)

print('Result of simpson method(h, 2h): ', res\_simpson\_h, res\_simpson\_2h)

d4f = sp.lambdify(x, y.diff(x).diff(x).diff(x).diff(x), 'numpy')

simpson\_error\_h = simpson\_error(a, b, d4f, h)

simpson\_error\_2h = simpson\_error(a, b, d4f, 2 \* h)

print('Error of simpson method with step h: ', simpson\_error\_h)

print('Error of simpson method with step 2h: ', simpson\_error\_2h)

#diff calculating

a = 1

b = 1.6

start\_point = (1., 2.)

h = np.float16(find\_diff\_h(a, b, g, start\_point))

print('Step size for Runge-Cutta method: ', h)

pts\_runge\_h = runge\_kutta4(a, b, g, h, start\_point)

pts\_runge\_2h = runge\_kutta4(a, b, g, 2 \* h, start\_point)

plt.plot([t[0] for t in pts\_runge\_h], [t[1] for t in pts\_runge\_h], label='Runge-Cutta')

plt.plot([t[0] for t in pts\_runge\_h], [right(t[0]) for t in pts\_runge\_h], label='Real function')

h = find\_diff\_h(a, b, g, start\_point, method='adams')

pts\_adams\_h = adams(a, b, g, h, start\_point)

pts\_adams\_2h = adams(a, b, g, 2 \* h, start\_point)

plt.plot([t[0] for t in pts\_adams\_h], [t[1] for t in pts\_adams\_h], label='Adams')

#plt.plot([t[0] for t in pts\_adams\_2h], [t[1] for t in pts\_adams\_2h])

pts\_euler\_h = euler(a, b, g, h, start\_point)

pts\_euler\_2h = euler(a, b, g, 2 \* h, start\_point)

plt.plot([t[0] for t in pts\_euler\_h], [t[1] for t in pts\_euler\_h], label='Euler')

#plt.plot([t[0] for t in pts\_euler\_2h], [t[1] for t in pts\_euler\_2h])

plt.legend(bbox\_to\_anchor=(0., 1.02, 1., .102), loc=3,

ncol=2, borderaxespad=0.)

plt.show()

data = result\_table(right, pts\_runge\_h, pts\_adams\_h)

#save\_data('Summary.xlsx', data)

if \_\_name\_\_ == '\_\_main\_\_':

main()