**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Лабораторная работа №2

Дубовик Ксения, 3 курс, 9 группа

Преподаватель: Полещук Максим Александрович

Минск 2017

1. **Задание.**

**Задание 1.** Методом Ритца при  найти приближенное решение первой краевой задачи для ОДУ второго порядка:



с точным решением . Выбрать параметр  равный номеру в списке академической группы.

Выполнить преобразование уравнения второго порядка к системе ОДУ первого порядка где  — вектор-функция . Весовые функции  задать следующим образом: .

Вычислить и представить в отчете:

1. Коэффициенты матрицы (значения соответствующих определенных интегралов) , вектора правой части  и вектора приближенного решения системы линейных уравнений .
2. Глобальная погрешность приближенного  решения , где величина  — число узлов сетки.
3. Графики приближенного  и точного  решений на одном рисунке.

**Задание 2.** Методом Галеркина при  найти приближенное решение краевой задачи из задания 1.

Вычислить и представить в отчете величины, указанные в пунктах 1–3 задания 1.

**Задание 3.** Методом стрельбы с использованием метода Рунге-Кутты четвертого порядка точности с адаптивным выбором длины шага (использовать реализацию из лабораторной работы 1) и методом Ньютона уточнения корней нелинейных уравнений найти приближенное решение краевой задачи из задания 1. Ограничить абсолютные погрешности методов Рунге-Кутты и Ньютона величиной .

Вычислить и представить в отчете следующие величины:

1. Число потребовавшихся итераций метода Ньютона.
2. Абсолютная погрешность правого краевого условия после последней итераций метода Ньютона.

Пункт 2 задания 1.

1. **Входные данные.**

 с точным решением , .

1. **Краткая теория**.  
   *Метод Ритца*

Общий вид задачи:

.

Сначала необходимо выбрать некоторую систему функций, которую мы назовем базисной: ( дана нам в условии). Она должна удовлетворять краевым условиям . Ищем приближённое решение в виде:

Чтобы найти минимум функционала, воспользуемся необходимым условием экстремума:

В матричной форме эта система запишется в виде ,

где ,

*Метод Галёркина*

Общий вид задачи:

Необходимо выбрать некоторую систему базисных функций ( как в методе Ритца). Ищем приближённое решение в виде:

В матричной форме задача принимает вид , где

,

*Метод стрельбы*

*,.*

Приближённое решение ищем в виде

1. **Листинг программы.**

import numpy as np

from matplotlib import pyplot as plt

from sympy import \*

s = 4.0

n\_count = 20

a = 1

b = 2

err\_step = float(b - a) / n\_count

x\_plot = np.arange(a, b, 0.01)

def phi0(x):

return x \*\* 2 - 2 \* x + 1

def phi1(x):

return (1 - x) \*\* 2 \* (2 - x)

def phi2(x):

return (1 - x) \*\* 3 \* (2 - x)

def get\_global\_err(fun):

global\_err = 0

for i in np.arange(1, n\_count):

curr\_x = a + i \* err\_step

sol = solution(curr\_x)

approx\_sol = fun(curr\_x)

curr\_err = abs(sol - approx\_sol)

global\_err = max(global\_err, curr\_err)

return global\_err

def get\_plot(fun, name):

sol\_plot = plt.plot(x\_plot, solution(x\_plot), label='solution')

approx\_plot = plt.plot(x\_plot, fun(x\_plot), label=name)

plt.setp(sol\_plot, color='b')

plt.setp(approx\_plot, color='g')

plt.legend()

plt.show()

pass

def solution(x):

return (x \*\* 2 - 1.0) \* (3.0 \* s \* (x \*\* 2 - 4.0) + 8.0) / 24.0

h = 1.0 / (n\_count)

def ritz():

def p(x):

return 1.0 / x

def q(x):

return 0

def f(x):

return -s \* x

x\_sym = Symbol("x\_sym")

integr\_limits = (x\_sym, a, b)

a\_matr = np.zeros((2, 2))

a\_matr[0, 0] = Integral(

p(x\_sym) \* diff(phi1(x\_sym), x\_sym) \* diff(phi1(x\_sym), x\_sym) + q(x\_sym) \* phi1(x\_sym) \* phi1(x\_sym),

integr\_limits)

a\_matr[0, 1] = a\_matr[1, 0] = Integral(

p(x\_sym) \* diff(phi1(x\_sym), x\_sym) \* diff(phi2(x\_sym), x\_sym) + q(x\_sym) \* phi1(x\_sym) \* phi2(x\_sym),

integr\_limits)

a\_matr[1, 1] = Integral(

p(x\_sym) \* diff(phi2(x\_sym), x\_sym) \* diff(phi2(x\_sym), x\_sym) + q(x\_sym) \* phi2(x\_sym) \* phi2(x\_sym),

integr\_limits)

b\_vec = np.zeros((2, 1))

b\_vec[0, 0] = Integral(

f(x\_sym) \* phi1(x\_sym) - p(x\_sym) \* diff(phi1(x\_sym), x\_sym) \* diff(phi0(x\_sym), x\_sym) - q(x\_sym) \* phi1(

x\_sym) \* phi0(x\_sym),

integr\_limits)

b\_vec[1, 0] = Integral(

f(x\_sym) \* phi2(x\_sym) - p(x\_sym) \* diff(phi2(x\_sym), x\_sym) \* diff(phi0(x\_sym), x\_sym) - q(x\_sym) \* phi2(

x\_sym) \* phi0(x\_sym),

integr\_limits)

c\_vec = np.linalg.solve(a\_matr, b\_vec)

c1, c2 = c\_vec[0][0], c\_vec[1][0]

def ritz\_solution(x):

return phi0(x) + c1 \* phi1(x) + c2 \* phi2(x)

global\_err = get\_global\_err(ritz\_solution)

print('A = ', a\_matr)

print ('b = ', b\_vec)

print('c = ', c\_vec)

print('ritz global err = ', global\_err)

get\_plot(ritz\_solution, 'ritz')

pass

def galerckin():

def p(x):

return -1.0 / x

def q(x):

return 0

def f(x):

return -s \* x \*\* 2

x = Symbol('x')

A\_matr = np.zeros((2, 2))

global b

A\_matr[0, 0] = Integral((diff(phi1(x), x, b) + p(x) \* diff(phi1(x), x) + q(x) \* phi1(x)) \* phi1(x), (x, a, b))

A\_matr[0, 1] = Integral((diff(phi2(x), x, b) + p(x) \* diff(phi2(x), x) + q(x) \* phi2(x)) \* phi1(x), (x, a, b))

A\_matr[1, 0] = Integral((diff(phi1(x), x, b) + p(x) \* diff(phi1(x), x) + q(x) \* phi1(x)) \* phi2(x), (x, a, b))

A\_matr[1, 1] = Integral((diff(phi2(x), x, b) + p(x) \* diff(phi2(x), x) + q(x) \* phi2(x)) \* phi2(x), (x, a, b))

b\_vec = np.zeros((2, 1))

b\_vec[0, 0] = -Integral(((diff(phi0(x), x, b) + p(x) \* diff(phi0(x), x) + q(x) \* phi0(x) + f(x)) \* phi1(x)),

(x, a, b))

b\_vec[1, 0] = -Integral(((diff(phi0(x), x, b) + p(x) \* diff(phi0(x), x) + q(x) \* phi0(x) + f(x)) \* phi2(x)),

(x, a, b))

c = np.linalg.solve(A\_matr, b\_vec)

c1, c2 = c[0][0], c[1][0]

def galerckin\_solution(x):

return phi0(x) + c1 \* phi1(x) + c2 \* phi2(x)

global\_err = get\_global\_err(galerckin\_solution)

print('galerckin global err = ', global\_err)

get\_plot(galerckin\_solution, 'galerckin')

pass

def shooting\_method():

def deriv(x, u):

return 1.0 / x \* u + s \* x \*\* 2

def mrk(N, h, start, x):

corr = np.zeros(2 \* N + 1)

corr[0] = start

for i in range(0, 2 \* N):

x0 = x[i]

y0 = corr[i]

k1 = deriv(x0, y0)

k2 = deriv(x0 + h / 2.0, y0 + h \* k1 / 2.0)

k3 = deriv(x0 + h / 2.0, y0 + h \* k2 / 2.0)

k4 = deriv(x0 + h, y0 + h \* k3)

corr[i + 1] = (y0 + (k1 + 2.0 \* k2 + 2.0 \* k3 + k4) \* h / 6.0)

dercorr = np.zeros(N + 1)

dercorr[0] = 0

i = 0

for j in range(0, N):

y0 = dercorr[j]

k1 = corr[i]

k2 = corr[i + 1]

k3 = corr[i + 1]

k4 = corr[i + 2]

dercorr[j + 1] = (y0 + (k1 + 2.0 \* k2 + 2.0 \* k3 + k4) \* 2 \* h / 6.0)

i = i + 2

return dercorr

N = 21

a = 1

b = 2

alpha = -2

delta = -2

h = (b - a) / (1 + 2 \* N)

x = np.arange(a, b, h)

shot1 = mrk(N, h, alpha, x)

shot2 = mrk(N, h, alpha + delta, x)

shift\_alpha = (1 - shot1[N]) / (shot2[N] - shot1[N]) \* delta

shot3 = mrk(N, h, alpha + shift\_alpha, x)

x = np.arange(a, b, h \* 2)

plt.plot(x, shot1, label='shoot1')

plt.plot(x, shot2, label='shoot2')

plt.plot(x, shot3, label='shoot3')

plt.plot(x, solution(x), label='solution')

plt.legend()

plt.show()

print('shooting global error = ', (max(abs(solution(x) - shot3))))

pass

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

# ritz()

# galerckin()

shooting\_method()

1. **Результаты.**

*Результаты задания 1:*

s = 4.

A11 = 0.07867951

A12 = -0.05651535

A21 = -0.05651535

A22 = 0.04754518

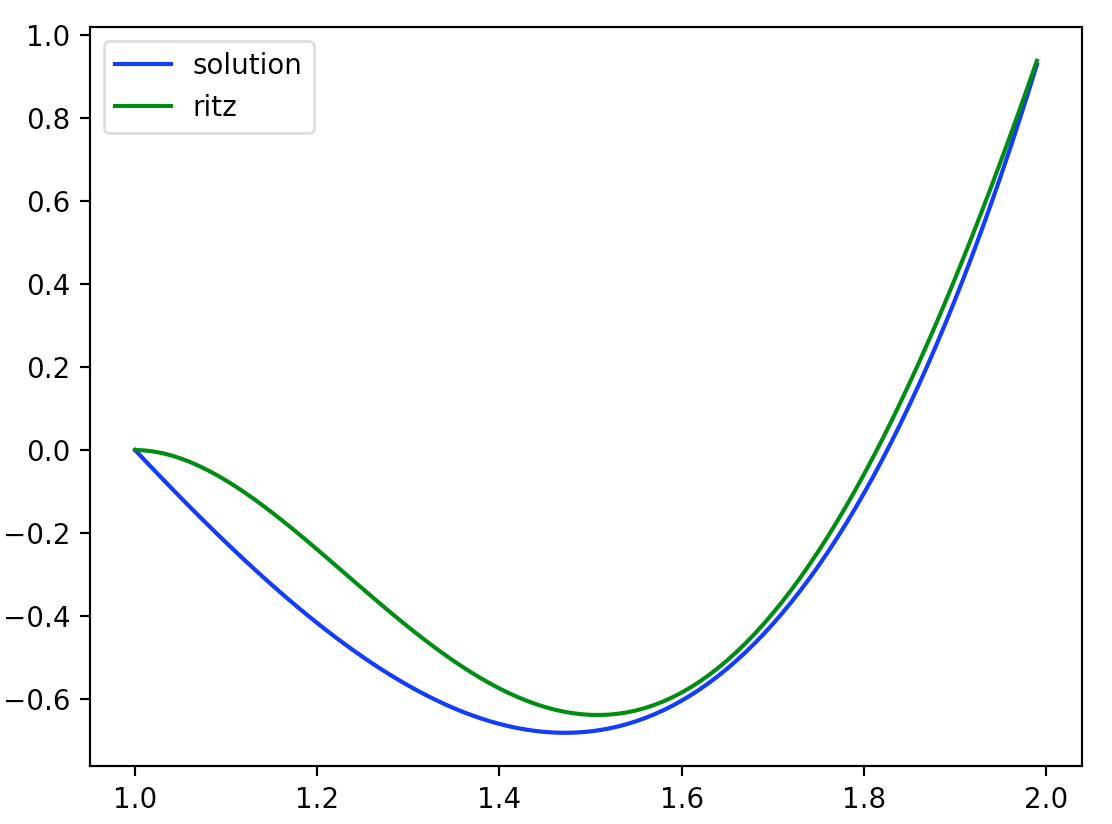
B1 = -0.46480514

B2 = 0.29593947

C1 = -9.82746509

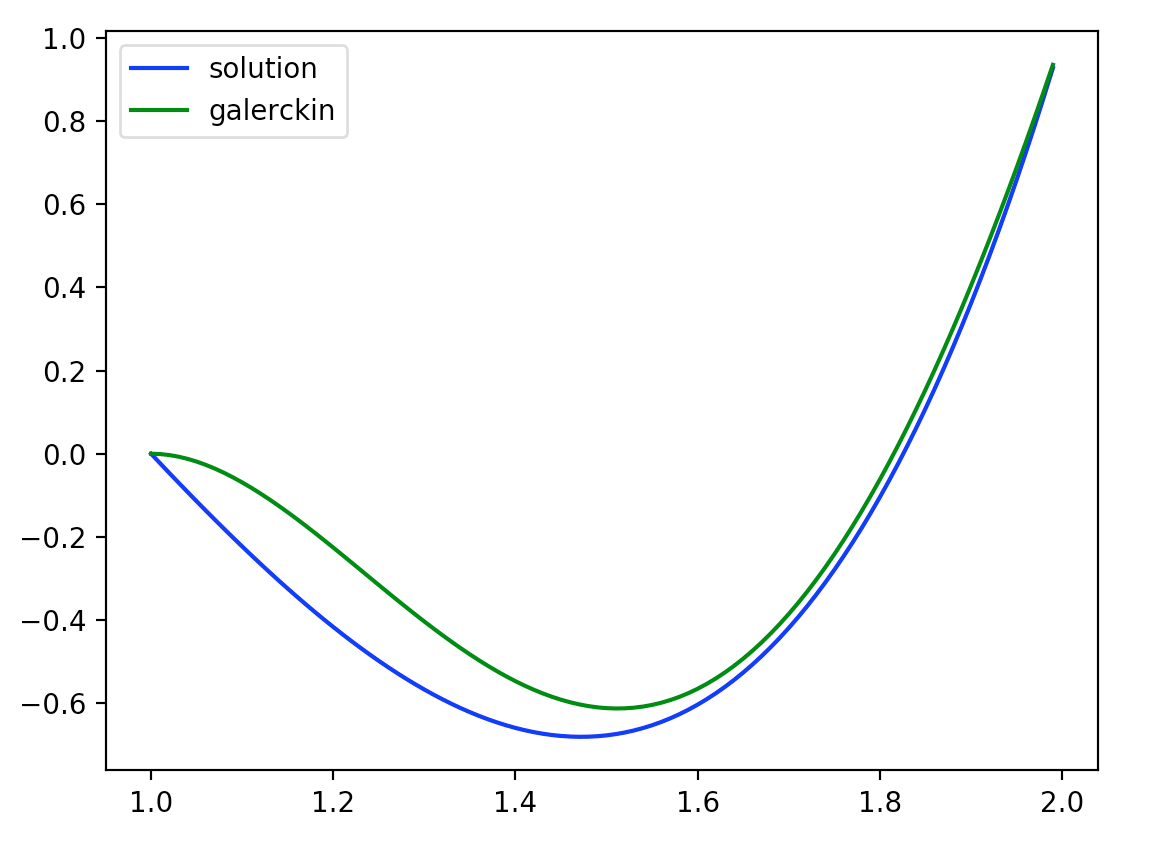
C2 = -5.45719091

= 0.176980472306 (глобальная погрешность приближенного решения)

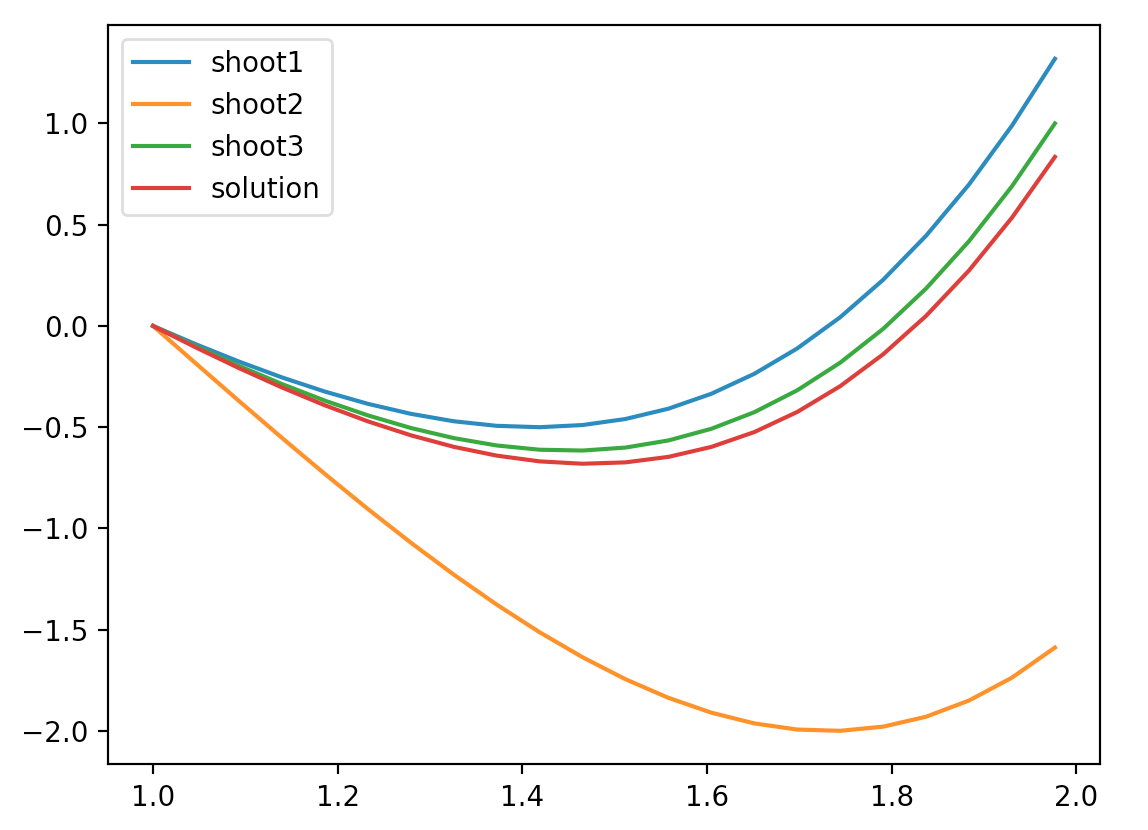


*Результаты задания 2:*

= 0.190881751362 (глобальная погрешность приближенного решения)

****

*Результаты задания 3:*



Egl = 0.165274609432

1. **Анализ результатов:**

 Основным преимуществом метода Галёркина является то, что основой для него служит исходное дифференциальное уравнение. Поэтому метод Галеркина с успехом применяется при решении задач, из которых не удается подобрать функционал для минимизации. Недостаток метода— в трудностях машинной реализации соответствующего алгоритма для проблем, содержащих в качестве неизвестных вектор-функции или дифференциальные операторы порядка выше второго.

Основным недостатком метода Ритца является то, что он применим только для уравнений с симметричными положительно определенными операторами. От этого недостатка свободен метод Галеркина.

Метод стрельбы часто неустойчив к вычислительной погрешности, но он прост, применим как к линейным, так и к нелинейным задачам и позволяет использовать при численном интегрировании схемы Рунге - Кутта ( или другие) высокого порядка точности. К большинству задач он применяется успешно. Основным недостатком данного метода является следующее обстоятельство: если исходное уравнение содержит быстро растущие решения (например экспоненциального вида), то вычисления происходят с большой потерей точности.