Минобрнауки России

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«САРАТОВСКИЙ национальный исследовательский**

**ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра математической физики и вычислительной математики

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ ПОДГОТОВКЕ**

по дисциплине «Методы вычислений»

студента \_3\_ курса \_341\_\_ группы

|  |
| --- |
| направления 02.03.03 – Математическое обеспечение и администрирование информационных систем |
| код и наименование направления (специальности) |
| Факультета компьютерных наук и информационных технологий |
| наименование факультета, института, колледжа |
| Филиппенко Дмитрия Александровича |

фамилия, имя, отчество

проверил:   подпись, дата

Саратов 2024

СОДЕРЖАНИЕ

[Задание 1. Полином Ньютона 3](#_Toc185338814)

[1) В общем виде: 3](#_Toc185338815)

[2) В форме Лагранжа: 4](#_Toc185338816)

[3) В форме Ньютона. 5](#_Toc185338817)

[Задание 3. Метод Гаусса 6](#_Toc185338818)

[Кодовое представление: 6](#_Toc185338819)

[Задание 4. Метод прогонки. 8](#_Toc185338820)

[Кодовое представление: 8](#_Toc185338821)

[Задание 5. Решение задачи Коши методом Эйлера 10](#_Toc185338822)

[Задание 6. Метод неопределенных коэффициентов для краевой задачи ОДУ 15](#_Toc185338823)

[7 Решение интегрального уравнения Фредгольма в случае вырожденного ядра. 16](#_Toc185338824)

# Задание 1. Полином Ньютона

С помощью данной таблицы - функции вычислите приближенное значение функции в указанных точках, используя интерполяционный многочлен:

1) в общем виде;

2) в форме Лагрнажа;

3) в форме Ньютона.

Таблица - функция:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Xi | 0 | 1 | 2 | 3 |
| Y(Xi) | 11 | 12 | 19 | 38 |

Найти:

|  |  |  |  |
| --- | --- | --- | --- |
| X | 0,5 | 1,5 | 2,5 |
| Pn(X) | ? | ? | ? |

## 1) В общем виде:

import numpy

var = 12.0

M2 = numpy.array([[0., 0., 0., 1.], [1., 1., 1., 1.], [8., 4., 2., 1.], [27., 9., 3., 1.]]) # матрица

v2 = numpy.array([var, 1+var, 8+var, 27+var]) # свободные члены

abcd = numpy.linalg.solve(M2, v2)

print(abcd)

for i in range(0, 3, 1):

answer = abcd[0]\*(i+0.5)\*\*3 + abcd[1]\*(i+0.5)\*\*2 + abcd[2]\*(i+0.5) + abcd[3]

print(answer)

Вывод:  
12.125

15.375

27.625

## 2) В форме Лагранжа:

import numpy as np

x = np.array([0, 1, 2, 3])

V = 12.0

y = np.array([V, 1 + V, 8 + V, 27 + V])

def lagrange(x, y, x\_values):

n = len(x)

result = []

for x\_val in x\_values:

L\_n = 0

for i in range(n):

l\_i = 1

for j in range(n):

if i != j:

l\_i \*= (x\_val - x[j]) / (x[i] - x[j])

L\_n += l\_i \* y[i]

result.append(L\_n)

return result

x\_value = [0.5, 1.5, 2.5]

interpolated\_values = lagrange(x, y, x\_value)

for x\_val, y\_val in zip(x\_value, interpolated\_values):

print(f"x = {x\_val}: y = {y\_val}")

Вывод:  
x = 0.5: y = 12.125

x = 1.5: y = 15.375

x = 2.5: y = 27.625

## 3) В форме Ньютона.

from math import prod

def NetRec(y\_i, x\_i, xs, xe):

    if xs == xe:

        return y\_i[xs]

    else:

        a = (NetRec(y\_i,x\_i,xs+1,xe)-NetRec(y\_i,x\_i,xs,xe-1))/(xe-xs)

        print(f"{xs}, {xe}", a)

        return a

def main():

    variant, n = int(input()), int(input())

    y\_i = [variant + i\*\*3 for i in range(0,n+1)] # значения функции в углах интерполяции

    x\_i = [i for i in range(0,n)] # узлы интерполяции (переменная Х)

    for x in [0.5 + i for i in range(0, 3)]:

        answer = sum([NetRec(y\_i,x\_i,0,i)\*prod([x-i1 for i1 in range(0,i)]) for i in range(0,n)])

        print(f"P\_n({x}) = {answer}")

main()

Вывод:  
x = 0.5: y = 12.125

x = 1.5: y = 15.375

x = 2.5: y = 27.625

# Задание 3. Метод Гаусса

Решить СЛАУ методом Гаусса. V = 12

A \* x = b

A =

b = A \*

## Кодовое представление:

import numpy as np

def gauss\_method(matrix, rhs):

    """

    :param matrix: матрица коэффициентов системы (размер n x n)

    :param rhs: вектор правых частей (размер n)

    :return: решение системы в виде вектора (размер n)

    """

    n = len(matrix)

    aug\_matrix = np.hstack([matrix, rhs.reshape(-1, 1)])

    # Прямой ход, делаем матрицу треугольного вида

    for i in range(n):

        # меняем местами в текущем столбце строку с макс элементом

        max\_row = i + np.argmax(np.abs(aug\_matrix[i:, i]))

        aug\_matrix[[i, max\_row]] = aug\_matrix[[max\_row, i]]

        # приведение текущей строки к удобному виду, чтобы на главной позиции был 1

        aug\_matrix[i] = aug\_matrix[i] / aug\_matrix[i, i]

        # ниже главного жлемента обнуляем, чтоб получилась треугольная матрица

        for j in range(i + 1, n):

            aug\_matrix[j] -= aug\_matrix[i] \* aug\_matrix[j, i]

    # Обратный ход

    solution = np.zeros(n)

    for i in range(n - 1, -1, -1): # проходимся снизу вверх

        solution[i] = aug\_matrix[i, -1] - np.sum(aug\_matrix[i, i + 1:n] \* solution[i + 1:n])

    return solution

# матрица коэффициентов

A = np.array([[0, 1, -1],

              [-3, -1, 2],

              [-2, 1, 2]], dtype=float)

b = np.array([8, -11, -3], dtype=float) # векторы правых частей

solution = gauss\_method(A, b)

print("Решение системы:", solution)

Вывод:  
Решение системы: [-0.28571429 4.14285714 -3.85714286]

# Задание 4. Метод прогонки.

Решить СЛАУ методом прогонки. V = 12

A \* x = b

A =

b = A \*

## Кодовое представление:

import numpy as np

n = 5

V = 4

a = np.array([(V + i) / 100 for i in range(1, n)])  # поддиагонал

b\_diagonal = np.array([V + i for i in range(0, n)])  # главная диагональ

c = np.array([(V + i) / 100 for i in range(2, n + 1)])  # наддиагональ

A = np.zeros((n, n))

np.fill\_diagonal(A, b\_diagonal)

np.fill\_diagonal(A[1:], a)

np.fill\_diagonal(A[:, 1:], c)

b = A @ b\_diagonal  # умножаем матрицу A на столбец главной диагонали b\_diagonal

# метод прогонки

def progonka(a, b, c, d):

    n = len(d)

    P = np.zeros(n-1)

    Q = np.zeros(n)

    # Прямой ход

    P[0] = c[0] / b[0]

    Q[0] = d[0] / b[0]

    for i in range(1, n-1):

        denominator = b[i] - a[i-1] \* P[i-1] #  это знаменатель в формулах для P и Q

        # Он учитывает предыдущие коэффициенты, что позволяет нам исключить неизвестные из предыдущих уравнений

        P[i] = c[i] / denominator # исп для того, чтобы "исключить" одно из неизвестных при переходе от одного уравнения к следующему

        Q[i] = (d[i] - a[i-1] \* Q[i-1]) / denominator # показывает, что происходит с текущим уравнением после учета влияния всех предыдущих уравнений

    # находим последний элемент

    Q[-1] = (d[-1] - a[-2] \* Q[-2]) / (b[-1] - a[-2] \* P[-2])

    # обратный ход, начинаем с конца системы

    x = np.zeros(n)

    x[-1] = Q[-1]

    for i in range(n-2, -1, -1):

        x[i] = Q[i] - P[i] \* x[i+1]

    return x, P, Q

x, P, Q = progonka(a, b\_diagonal, c, b)

print("Матрица A:")

print(A)

print("\nВектор правой части b = A \* (столбец главной диагонали):")

print(b)

print("\nПрогоночные коэффициенты P\_i:")

print(P)

print("\nПрогоночные коэффициенты Q\_i:")

print(Q)

print("\nРешение системы (вектор x):")

print(x)

Вывод:

Вектор правой части b = A \* (столбец главной диагонали):

[16.3 25.62 36.86 50.14 64.56]

Прогоночные коэффициенты P\_i:

[0.015 0.0140021 0.0133352 0.01285886]

Прогоночные коэффициенты Q\_i:

[4.075 5.0840126 6.0933464 7.10287086 8.00878437]

Решение системы (вектор x):

[4. 4.99999998 6.00000151 6.99988704 8.00878437]

# Задание 5. Решение задачи Коши методом Эйлера

Решить задачу методом Эйлера и усовершенствованным методом Эйлера.

y(11) = 0

Найти приближение к решению на [11;16] с шагами h = 0.1; h=0.05  
Кодовое представление:

import numpy as np

V = 12.0

def f(x, y):

    return 4 \* (x\*\*3) - 3 \* (x\*\*2) \* V

def exact\_solution(x):

    return (x\*\*3) \* (x -V)

# Метод Эйлера

def euler\_method(f, x0, y0, h, x\_end):

    x\_values = [x0]

    y\_values = [y0]

    x = x0

    y = y0

    while x < x\_end:

        y += h \* f(x, y)

        x += h

        x\_values.append(x)

        y\_values.append(y)

    return np.array(x\_values), np.array(y\_values)

# Улучшенный метод Эйлера

def improved\_euler\_method(f, x0, y0, h, x\_end):

    x\_values = [x0]

    y\_values = [y0]

    x = x0

    y = y0

    while x < x\_end:

        y\_predict = y + h \* f(x, y)

        y += (h / 2) \* (f(x, y) + f(x + h, y\_predict))

        x += h

        x\_values.append(x)

        y\_values.append(y)

    return np.array(x\_values), np.array(y\_values)

x0 = V

y0 = 0

x\_end = V + 5

steps = [1, 0.1, 0.05]

for h in steps:

    print(f"\nШаг h = {h}")

    # Метод Эйлера

    x\_euler, y\_euler = euler\_method(f, x0, y0, h, x\_end)

    print("Метод Эйлера:")

    print("x\t\tПриближенное y\tТочное y\t\tРазница")

    for x\_val, y\_val in zip(x\_euler, y\_euler):

        y\_exact = exact\_solution(x\_val)

        diff = y\_val - y\_exact

        print(f"{x\_val:.5f}\t{y\_val:.5f}\t{y\_exact:.5f}\t{diff:.5f}")

    # Улучшенный метод Эйлера

    x\_improved, y\_improved = improved\_euler\_method(f, x0, y0, h, x\_end)

    print("\nУлучшенный метод Эйлера:")

    print("x\t\tПриближенное y\tТочное y\t\tРазница")

    for x\_val, y\_val in zip(x\_improved, y\_improved):

        y\_exact = exact\_solution(x\_val)

        diff = y\_val - y\_exact

        print(f"{x\_val:.5f}\t{y\_val:.5f}\t{y\_exact:.5f}\t{diff:.5f}")

# Открываем файл для записи

with open("results\_4\_1.txt", "w", encoding="utf-8") as file:

    for h in steps:

        file.write(f"\nШаг h = {h}\n")

        print(f"\nШаг h = {h}")

        # Метод Эйлера

        x\_euler, y\_euler = euler\_method(f, x0, y0, h, x\_end)

        file.write("Метод Эйлера:\n")

        file.write("x\t\tПриближенное y\tТочное y\t\tРазница\n")

        print("Метод Эйлера:")

        print("x\t\tПриближенное y\tТочное y\t\tРазница")

        for x\_val, y\_val in zip(x\_euler, y\_euler):

            y\_exact = exact\_solution(x\_val)

            diff = y\_val - y\_exact

            file.write(f"{x\_val:.5f}\t{y\_val:.5f}\t{y\_exact:.5f}\t{diff:.5f}\n")

            print(f"{x\_val:.5f}\t{y\_val:.5f}\t{y\_exact:.5f}\t{diff:.5f}")

        # Улучшенный метод Эйлера

        x\_improved, y\_improved = improved\_euler\_method(f, x0, y0, h, x\_end)

        file.write("\nУлучшенный метод Эйлера:\n")

        file.write("x\t\tПриближенное y\tТочное y\t\tРазница\n")

        print("\nУлучшенный метод Эйлера:")

        print("x\t\tПриближенное y\tТочное y\t\tРазница")

        for x\_val, y\_val in zip(x\_improved, y\_improved):

            y\_exact = exact\_solution(x\_val)

            diff = y\_val - y\_exact

            file.write(f"{x\_val:.5f}\t{y\_val:.5f}\t{y\_exact:.5f}\t{diff:.5f}\n")

            print(f"{x\_val:.5f}\t{y\_val:.5f}\t{y\_exact:.5f}\t{diff:.5f}")

**Вывод:**

**Шаг h = 1**

**Метод Эйлера:**

x Приближенное y Точное y Разница

12.00000 0.00000 0.00000 0.00000

13.00000 1728.00000 2197.00000 -469.00000

14.00000 4432.00000 5488.00000 -1056.00000

15.00000 8352.00000 10125.00000 -1773.00000

16.00000 13752.00000 16384.00000 -2632.00000

17.00000 20920.00000 24565.00000 -3645.00000

**Улучшенный метод Эйлера:**

x Приближенное y Точное y Разница

12.00000 0.00000 0.00000 0.00000

13.00000 2216.00000 2197.00000 19.00000

14.00000 5528.00000 5488.00000 40.00000

15.00000 10188.00000 10125.00000 63.00000

16.00000 16472.00000 16384.00000 88.00000

17.00000 24680.00000 24565.00000 115.00000

**Шаг h = 0.1**

**Метод Эйлера:**

x Приближенное y Точное y Разница

12.00000 0.00000 0.00000 0.00000

12.10000 172.80000 177.15610 -4.35610

12.20000 354.34840 363.16960 -8.82120

12.30000 544.86360 558.26010 -13.39650

12.40000 744.56640 762.64960 -18.08320

12.50000 953.68000 976.56250 -22.88250

12.60000 1172.43000 1200.22560 -27.79560

…

**Улучшенный метод Эйлера:**

x Приближенное y Точное y Разница

12.00000 0.00000 0.00000 0.00000

12.10000 177.17420 177.15610 0.01810

12.20000 363.20600 363.16960 0.03640

12.30000 558.31500 558.26010 0.05490

12.40000 762.72320 762.64960 0.07360

12.50000 976.65500 976.56250 0.09250

12.60000 1200.33720 1200.22560 0.11160

12.70000 1433.99900 1433.86810 0.13090

12.80000 1677.87200 1677.72160 0.15040

12.90000 1932.19020 1932.02010 0.17010

…

**Шаг h = 0.05**

**Метод Эйлера**:

x Приближенное y Точное y Разница

12.00000 0.00000 0.00000 0.00000

12.05000 86.40000 87.48451 -1.08451

12.10000 174.97353 177.15610 -2.18258

12.15000 265.74773 269.04201 -3.29428

12.20000 358.74990 363.16960 -4.41970

12.25000 454.00750 459.56641 -5.55891

12.30000 551.54813 558.26010 -6.71198

12.35000 651.39953 659.27851 -7.87898

12.40000 753.58960 762.64960 -9.06000

…

**Улучшенный метод Эйлера**:

x Приближенное y Точное y Разница

12.00000 0.00000 0.00000 0.00000

12.05000 87.48676 87.48451 0.00226

12.10000 177.16063 177.15610 0.00452

12.15000 269.04881 269.04201 0.00681

12.20000 363.17870 363.16960 0.00910

12.25000 459.57781 459.56641 0.01141

12.30000 558.27383 558.26010 0.01372

12.35000 659.29456 659.27851 0.01606

12.40000 762.66800 762.64960 0.01840

12.45000 868.42226 868.40151 0.02076

12.50000 976.58563 976.56250 0.02312

12.55000 1087.18651 1087.16101 0.02551

12.60000 1200.25350 1200.22560 0.02790

…

# Задание 6. Метод неопределенных коэффициентов для краевой задачи ОДУ

Метод неопределенных коэффициентов для краевого задачи ОДУ

import numpy as np

V = 12  #  правая граница

n = 100  # кол-во узлов

h = V / n  # шаг

def p(x): return x\*\*2

def q(x): return x

def f(x): return 4 \* x\*\*4 - 3 \* V \* x\*\*3 + 6 \* x - 2 \*

def y\_exact(x): return x\*\*2 \* (x - V)

# Сетка

x = np.linspace(0, V, n+1)

# Матрица и вектор

A = np.zeros((n+1, n+1))

b = np.zeros(n+1)

# узлы

for i in range(1, n):

    A[i, i-1] = 1 / h\*\*2 - p(x[i]) / (2 \* h)

    A[i, i] = -2 / h\*\*2 + q(x[i])

    A[i, i+1] = 1 / h\*\*2 + p(x[i]) / (2 \* h)

    b[i] = f(   x[i])

A[0, 0] = 1

A[n, n] = 1

b[0] = 0

b[n] = 0

y\_approx = np.linalg.solve(A, b)

y\_exact\_values = y\_exact(x)

difference = y\_approx - y\_exact\_values

print("x\t\t y\_approx\t y\_exact\t difference")

for xi, yi\_approx, yi\_exact, di in zip(x, y\_approx, y\_exact\_values, difference):

    print(f"{xi:.5f}\t {yi\_approx:.5f}\t {yi\_exact:.5f}\t {di:.5e}")

output\_file = "results\_5\_1.txt"

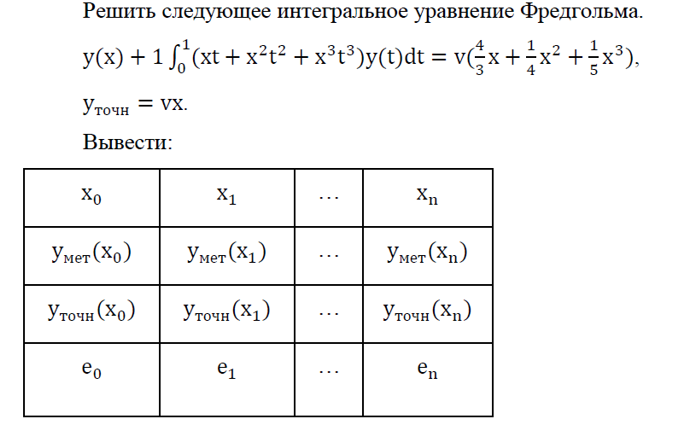
with open(output\_file, "w", encoding="utf-8") as file:

    file.write("x\t\t y\_приблж\t y\_точное\t разница\n")

    for xi, yi\_approx, yi\_exact, di in zip(x, y\_approx, y\_exact\_values, difference):

        file.write(f"{xi:.5f}\t {yi\_approx:.5f}\t {yi\_exact:.5f}\t {di:.5e}\n")

# 7 Решение интегрального уравнения Фредгольма в случае вырожденного ядра.



Кодовое представление:

import numpy as np

import matplotlib.pyplot as plt

def g(x):

    return 1 + x

def h(t):

    return 1 - t

def f(x):

    return x \* np.exp(-x)

# методом трапеции ищем площадь под кривой

def numerical\_integral(func, a, b, n):

    x = np.linspace(a, b, n)

    dx = (b - a) / (n - 1)

    return np.trapz(func(x), x)  # Метод трапеций

# найдя все x можно подставить в уравнение

def solve\_integral\_equation(a, b, n):

    # Дискретизация интервала [a, b]

    x\_values = np.linspace(a, b, n)

    # Строим правую часть уравнения

    integral\_value = numerical\_integral(h, a, b, n)  # Интеграл по t для фиксированного x

    f\_values = f(x\_values)

    y\_values = np.array([f\_values[i] / (g(x\_values[i]) \* integral\_value) for i in range(n)])

    return x\_values, y\_values

a = 0  # Левая граница

b = 1  # Правая граница

n = 100  # Количество точек дискретизации

x\_values, y\_values = solve\_integral\_equation(a, b, n)

plt.plot(x\_values, y\_values, label="Решение y(x)")

plt.xlabel("x")

plt.ylabel("y(x)")

plt.title("Решение интегрального уравнения Фредгольма с вырожденным ядром")

plt.legend()

plt.grid(True)

plt.show()

