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. Интерполяционный многочлен

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)	12	13	20	39

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 X)
    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

$$A = \begin{pmatrix} \boldsymbol{V} & \cdots & \frac{\boldsymbol{V}}{\boldsymbol{100}} \\ \frac{\boldsymbol{V+1}}{\boldsymbol{100}} & \boldsymbol{V+1} & \frac{\boldsymbol{V+1}}{\boldsymbol{100}} \\ \vdots & \boldsymbol{V+2} & \vdots \\ \frac{\boldsymbol{V+3}}{\boldsymbol{100}} & \boldsymbol{V+3} & \frac{\boldsymbol{V+3}}{\boldsymbol{100}} \\ \frac{\boldsymbol{V+4}}{\boldsymbol{100}} & \cdots & \boldsymbol{V+4} \end{pmatrix}$$

$$b = A * \begin{pmatrix} V \\ V+1 \\ \vdots \\ V+4 \end{pmatrix}$$

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

$$A = \begin{pmatrix} \boldsymbol{V} & \frac{\boldsymbol{V}}{\boldsymbol{100}} & \boldsymbol{0} & \dots & \boldsymbol{0} \\ \frac{\boldsymbol{V+1}}{\boldsymbol{100}} & \boldsymbol{V+1} & \frac{\boldsymbol{V+1}}{\boldsymbol{100}} & & \vdots \\ \boldsymbol{0} & \frac{\boldsymbol{V+2}}{\boldsymbol{100}} & \boldsymbol{V+2} & \frac{\boldsymbol{V+2}}{\boldsymbol{100}} & \boldsymbol{0} \\ \vdots & & \frac{\boldsymbol{V+3}}{\boldsymbol{100}} & \boldsymbol{V+3} & \frac{\boldsymbol{V+3}}{\boldsymbol{100}} \\ \boldsymbol{0} & \dots & \boldsymbol{0} & \frac{\boldsymbol{V+4}}{\boldsymbol{100}} & \boldsymbol{V+4} \end{pmatrix}$$

$$b = A * \begin{pmatrix} V \\ V+1 \\ \vdots \\ V+4 \end{pmatrix}$$

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.

$y_т = x^3 (x - 11)$

$\frac{dy}{dx} = 4x^3 - 33x^2$

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_e
xact:.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_e
xact:.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

$$\begin{cases} y'' + x^2 y' + xy = 4x^4 - 3Vx^3 + 6x - 2V, \\ y(0) = y(V) = 0 \end{cases}$$

Предполагаемое решение:

$$y_{\text{толч}} = x^2(x - V), \quad x \in [0; V]$$

Требуется:

1. Решить задачу с использованием **разностного метода**.
2. Найти **неопределённые коэффициенты**.

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_приближ\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")
```

Вывод:

```
x         y_приближ     y_точное     разница
0.00000   0.00000  -0.00000     4.95682e-14
0.12000   -0.08113     -0.17107     8.99459e-02
0.24000   -0.49780     -0.67738     1.79578e-01
0.36000   -1.24058     -1.50854     2.67963e-01
0.48000   -2.30063     -2.65421     3.53578e-01
0.60000   -3.66960     -4.10400     4.34401e-01
0.72000   -5.33949     -5.84755     5.08059e-01
0.84000   -7.30244     -7.87450     5.72061e-01
0.96000   -9.55038     -10.17446    6.24084e-01
…
10.44000 -170.00586   -170.03002   2.41607e-02
10.56000 -160.55742   -160.57958   2.21662e-02
10.68000 -150.54217   -150.56237   2.01971e-02
10.80000 -139.94975   -139.96800   1.82527e-02
10.92000 -128.76978   -128.78611   1.63321e-02
11.04000 -116.99190   -117.00634   1.44345e-02
11.16000 -104.60574   -104.61830   1.25592e-02
11.28000 -91.60094    -91.61165    1.07055e-02
11.40000 -77.96713    -77.97600    8.87273e-03
11.52000 -63.69393    -63.70099    7.06015e-03
11.64000 -48.77099    -48.77626    5.26718e-03
11.76000 -33.18793    -33.19142    3.49321e-03
11.88000 -16.93439    -16.93613    1.73767e-03
12.00000  0.00000  0.00000  0.00000e+00
```

Задание 7. Решение интегрального уравнения Фредгольма в случае вырожденного ядра.

Решить следующее интегральное уравнение Фредгольма.

$$y(x) + 1\int_0^1 (xt + x^2t^2 + x^3t^3)y(t)dt = v(\frac{4}{3}x + \frac{1}{4}x^2 + \frac{1}{5}x^3),$$

$y_{точн} = vx.$

Вывести:

x_0	x_1	…	x_n
$y_{мет}(x_0)$	$y_{мет}(x_1)$	…	$y_{мет}(x_n)$
$y_{точн}(x_0)$	$y_{точн}(x_1)$	…	$y_{точн}(x_n)$
e_0	e_1	…	e_n

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()
```