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»

студентки 3 курса 341 группы

направления02.04.03 Математическое обеспечение и администрирование информационных систем

факультета компьютерных наук и информационных технологий

Суховой Екатерины Владимировны

Проверил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

Саратов 2022

**СОДЕРЖАНИЕ**

[**РАЗДЕЛ 1 Приближение функции одного аргумента** 3](file:///C:\Users\Ekats\Documents\Методы%20вичислений%20отчет.doc#_Toc121427417)

[1.1 Интерполяционный многочлен в общем виде 3](file:///C:\Users\Ekats\Documents\Методы%20вичислений%20отчет.doc#_Toc121427418)

[1.2 Интерполяционный многочлен в форме Лагранжа 5](file:///C:\Users\Ekats\Documents\Методы%20вичислений%20отчет.doc#_Toc121427419)

[1.3 Интерполяционный многочлен в форме Ньютона 6](file:///C:\Users\Ekats\Documents\Методы%20вичислений%20отчет.doc#_Toc121427420)

[1.4 Интерполяция кубическими сплайнами 7](file:///C:\Users\Ekats\Documents\Методы%20вичислений%20отчет.doc#_Toc121427421)

[**РАЗДЕЛ 2 Численные методы решения СЛАУ** 10](file:///C:\Users\Ekats\Documents\Методы%20вичислений%20отчет.doc#_Toc121427422)

[2.1 Метод Гаусса решения СЛАУ 10](file:///C:\Users\Ekats\Documents\Методы%20вичислений%20отчет.doc#_Toc121427423)

[2.2 Приложения метода Гаусса 13](file:///C:\Users\Ekats\Documents\Методы%20вичислений%20отчет.doc#_Toc121427424)

[2.3 Решение СЛАУ методом прогонки 16](file:///C:\Users\Ekats\Documents\Методы%20вичислений%20отчет.doc#_Toc121427425)

[**РАЗДЕЛ 3 Численные методы решения дифференциальных уравнений** 18](file:///C:\Users\Ekats\Documents\Методы%20вичислений%20отчет.doc#_Toc121427426)

[3.1 Метод Эйлера решения задачи Коши для ОДУ первого порядка 18](file:///C:\Users\Ekats\Documents\Методы%20вичислений%20отчет.doc#_Toc121427427)

[3.2 Разностный метод решения краевой задачи для ОДУ второго порядка 21](file:///C:\Users\Ekats\Documents\Методы%20вичислений%20отчет.doc#_Toc121427428)

[3.3 Метод неопределенных коэффициентов решения краевой задачи для ОДУ второго порядка 23](file:///C:\Users\Ekats\Documents\Методы%20вичислений%20отчет.doc#_Toc121427429)

[**РАЗДЕЛ 4 Численные методы решения интегральных уравнений** 25](file:///C:\Users\Ekats\Documents\Методы%20вичислений%20отчет.doc#_Toc121427430)

[4.1 Решение интегрального уравнения Фредгольма в случае вырожденного ядра 25](file:///C:\Users\Ekats\Documents\Методы%20вичислений%20отчет.doc#_Toc121427431)

[4.2 Квадратурный метод решения интегрального уравнения Фредгольма 27](file:///C:\Users\Ekats\Documents\Методы%20вичислений%20отчет.doc#_Toc121427432)

РАЗДЕЛ 1 Приближение функции одного аргумента

* 1. Интерполяционный многочлен в общем виде

*Формулировка решаемой задачи.*

Построить интерполяционный многочлен в общем виде для функции заданной таблицей. Вычислить значения в точках 0.5, 1.5, 2.5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | 0 | 1 | 2 | 3 |
| f(x) | 7 | 8 | 11 | 15 |

Многочлен в программе был получен в ходе решения следующей СЛАУ:

*Код*

X = [0, 1, 2, 3]

Y = [7, 8, 11, 15]

def y(x):

return (-1. / 6) \* (x \* x \* x) + 1.5 \* (x \* x) + (-1. / 3) \* x + 7

print("\nCommon view\n")

k = 0.5

for i in range(3):

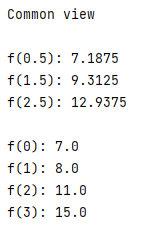
print(f"f({k + i}): {y(k + i)}")

print()

for i in range(4):

print(f"f({i}): {y(i)}")

*Результат работы программы:*



* 1. Интерполяционный многочлен в форме Лагранжа

*Формулировка решаемой задачи.*

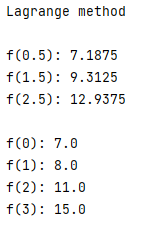
Построить интерполяционный многочлен в форме Лагранжа для функции заданной таблицей. Вычислить значения в точках 0.5, 1.5, 2.5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | 0 | 1 | 2 | 3 |
| f(x) | 7 | 8 | 11 | 15 |

*Код*

X = [0, 1, 2, 3]  
Y = [7, 8, 11, 15]  
  
  
def L(x, n):  
 ans = 0  
 for i in range(n + 1):  
 res = 1  
 for j in range(n + 1):  
 if i != j:  
 res \*= (x - X[j]) / (X[i] - X[j])  
 ans += res\*Y[i]  
 return ans  
  
  
print("\nLagrange method\n")  
  
k = 0.5  
for i in range(3):  
 print(f"f({k + i}): {L(k + i, 3)}")  
  
print()  
  
for i in range(4):  
 print(f"f({i}): {L(i, 3)}")

*Результат работы программы:*



* 1. Интерполяционный многочлен в форме Ньютона

*Формулировка решаемой задачи.*

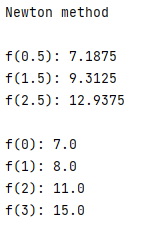
Построить интерполяционный многочлен в форме Ньютона для функции заданной таблицей. Вычислить значения в точках 0.5, 1.5, 2.5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | 0 | 1 | 2 | 3 |
| f(x) | 7 | 8 | 11 | 15 |

*Код*

X = [0, 1, 2, 3]  
Y = [7, 8, 11, 15]  
  
def f(i, j):if (j - i == 1):  
 return (Y[j] - Y[i])/(X[j] - X[i])  
 return (f(i + 1, j) - f(i, j - 1)) / (X[j] - X[i])  
  
  
def N(x, n):  
 ans = Y[0]  
 for i in range(1, n+1):  
 res = f(0, i)  
 for j in range(0, i):  
 res \*= x - X[j]  
 ans += res  
 return ans  
  
  
print("\nNewton method\n")  
k = 0.5  
for i in range(3):  
 print(f"f({k + i}): {N(k + i, 3)}")  
  
print()  
  
for i in range(4):  
 print(f"f({i}): {N(i, 3)}")

*Результат работы программы:*



* 1. Интерполяция кубическими сплайнами

*Формулировка решаемой задачи.*

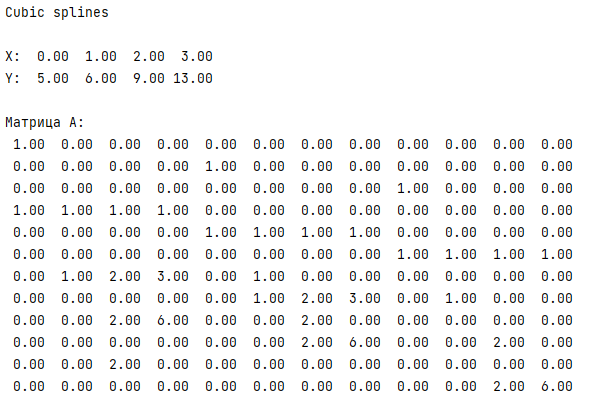
По данным интерполяции построить кубические сплайны. Вычислить значения в точках 0.5, 1.5, 2.5.

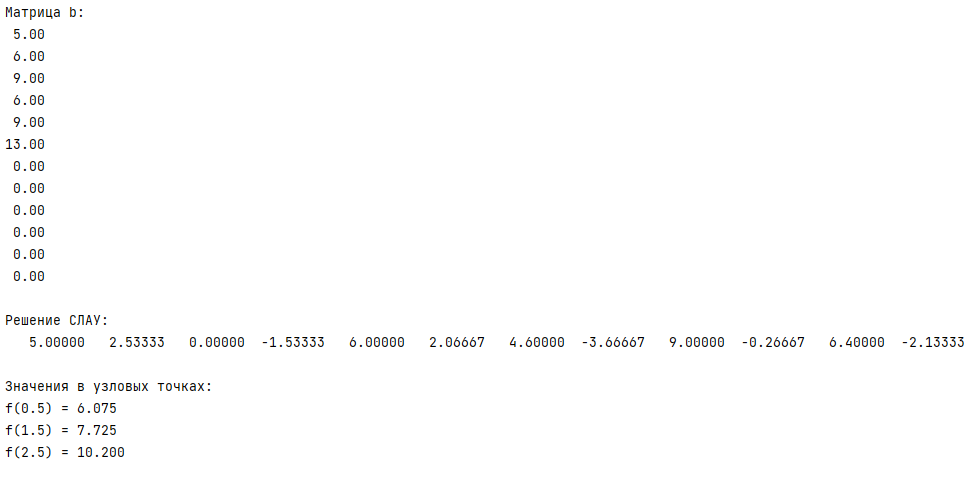
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | 0 | 1 | 2 | 3 |
| f(x) | 5 | 6 | 9 | 13 |

*Код*

import numpy as np  
import gauss  
  
X = [0, 1, 2, 3]  
Y = [5, 6, 9, 13]  
h = 1  
n = 3  
  
  
def create\_matrix():  
 view\_2 = [[1 if i \* 4 == j else 0 for j in range(4 \* n)] for i in range(0, n)]  
 view\_3 = [[1 if (i - 1) \* 4 <= j < i \* 4 else 0 for j in range(4 \* n)] for i in range(1, n + 1)]  
  
 view\_6 = [[0, 1, 2, 3, 0, 1] + [0 for i in range(6)]] + \  
 [[0 for i in range(4)] + [0, 1, 2, 3, 0, 1] + [0, 0]]  
  
 view\_7 = [[0, 0, 2, 6, 0, 0, 2, 0] + [0 for i in range(4)]] + \  
 [[0 for i in range(4)] + [0, 0, 2, 6, 0, 0, 2, 0]]  
  
 view\_8 = [[0, 0, 2] + [0 for i in range(9)]]  
  
 view\_9 = [[0 for i in range(10)] + [2, 6]]  
  
 b = [5, 6, 9, 6, 9, 13, 0, 0, 0, 0, 0, 0]  
 A = view\_2 + view\_3 + view\_6 + view\_7 + view\_8 + view\_9  
  
 return (A, b)  
  
  
print("\nCubic splines\n")  
  
print("X:", end=" ")  
for x in X:  
 print(f'{x: >5.2f}', end=" ")  
print()  
print("Y:", end=" ")  
for y in X:  
 print(f'{y: >5.2f}', end=" ")  
print("\n")  
  
A, b = create\_matrix()  
  
print("Матрица A:")  
for row in A:  
 for c in row:  
 print(f'{c: >5.2f}', end=" ")  
 print()  
print()  
  
print("Матрица b:")  
for c in b:  
 print(f'{c: >5.2f}')  
  
print()  
  
solution = gauss.method\_Gauss(A, b) *# решаем систему Ax = b*print("Решение СЛАУ:")  
  
for c in solution:  
 print(f'{c: >10.5f}', end="")  
  
print("\n")  
  
print('Значения в узловых точках:')  
  
x\_new = [0.5, 1.5, 2.5]  
for i in range(len(x\_new)):  
 h\_i = x\_new[i] - X[i]  
 i\_for\_func = len(X) \* i  
 func = solution[len(X) \* i] + solution[len(X) \* i + 1] \* h\_i + solution[len(X) \* i + 2] \* h\_i \*\* 2 + \  
 solution[len(X) \* i + 3] \* h\_i \*\* 3  
 print(f'f({x\_new[i]:.1f}) = {func:.3f}')  
  
print()

*Результат работы программы:*





РАЗДЕЛ 2 Численные методы решения СЛАУ

2.1 Метод Гаусса решения СЛАУ

*Формулировка решаемой задачи.*

Здесь и далее V обозначает номер варианта. Номер варианта V = 7.

Методом Гаусса решить СЛАУ вида Ax=b, где A и b строятся следующим образом:

A=

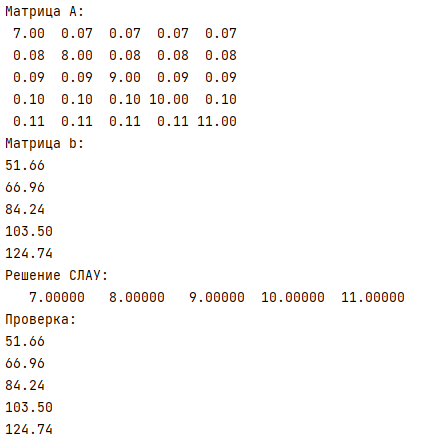
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| V | V \* 10-2 | V \* 10-2 | V \* 10-2 | V \* 10-2 |
| (V + 1) \* 10-2 | V + 1 | (V + 1) \* 10-2 | (V + 1) \* 10-2 | (V + 1) \* 10-2 |
| (V + 2) \* 10-2 | (V + 2) \* 10-2 | V + 2 | (V + 2) \* 10-2 | (V + 2) \* 10-2 |
| (V + 3) \* 10-2 | (V + 3) \* 10-2 | (V + 3) \* 10-2 | V + 3 | (V + 3) \* 10-2 |
| (V + 4) \* 10-2 | (V + 4) \* 10-2 | (V + 4) \* 10-2 | (V + 4) \* 10-2 | V + 4 |

b = A \* (V, V + 1, V + 2, V + 3, V + 4) T

*Код*

def method\_Gauss(A, b):  
 n = len(A)  
 res = []  
 for i in range(n):  
 cur\_lst = [A[i][j] for j in range(n)]  
 cur\_lst.append(b[i])  
 res.append(cur\_lst)  
  
 *# прямой ход* for i in range(n):  
 *# запоминаем первый ненулевой элемент строки i* elem = res[i][i]  
  
 *# меняем строки, если диагональный элемент == 0* if elem == 0:  
 max\_d = elem  
 m = i  
 for j in range(i + 1, n):  
 if abs(res[j][i]) > max\_d:  
 max\_d = abs(res[j][i])  
 m = j  
 res[i], res[m] = res[m], res[i]  
 elem = res[i][i]  
  
 *# делим текущую (i-ю) строку на её первый ненулевой элемент* for j in range(i, n + 1):  
 res[i][j] /= elem  
  
 *# цикл по оставшимся строкам матрицы* for k in range(i + 1, n):  
 elem = res[k][i]  
 for j in range(i, n + 1):  
 res[k][j] -= res[i][j] \* elem  
  
 *# обратный ход* x = []  
 for i in range(n):  
 x\_cur = res[n - i - 1][n]  
 for j in range(i):  
 x\_cur -= res[n - i - 1][n - j - 1] \* x[j]  
 x.append(x\_cur)  
  
 *# переворачиваем список решений, так как при добавлении шли с конца* x.reverse()  
 return x  
  
  
*# создаем список элементов главной диагонали*def make\_main\_diag(v, n):  
 return [v + i for i in range(n)]  
  
  
*# создаем матрицу А*def generate\_A(main\_diag, n):  
 return [[(main\_diag[i] if j == i else main\_diag[i] / 100) for j in range(n)] for i in range(n)]  
  
  
*# создаем матрицу b*def count\_b(A, main\_diag):  
 return [sum([A[i][j] \* main\_diag[j] for j in range(n)]) for i in range(n)]  
  
  
def check(A, x):  
 return [sum([A[i][j] \* x[j] for j in range(n)]) for i in range(n)]  
  
  
v = 7 *# номер варианта*n = 5 *# размерность матрицы*main\_diag = make\_main\_diag(v, n) *# создаем список элементов главной диагонали*A = generate\_A(main\_diag, n) *# создаем матрицу А*print("Матрица A:")  
for row in A:  
 for c in row:  
 print(f'{c: >5.2f}', end=" ")  
 print()  
  
b = count\_b(A, main\_diag) *# создаем матрицу b*print("Матрица b:")  
for c in b:  
 print(f'{c: >5.2f}') *# , end = " ")*x = method\_Gauss(A, b) *# решаем систему Ax = b*print("Решение СЛАУ:")  
  
for c in x:  
 print(f'{c: >10.5f}', end="")  
  
print()  
print("Проверка: ")  
new\_b = check(A, x)  
  
for c in new\_b:  
 print(f'{c: >5.2f}')

*Результат работы программы:*



2.2 Приложения метода Гаусса

*Формулировка решаемой задачи.*

Здесь и далее V обозначает номер варианта. Номер варианта V = 7

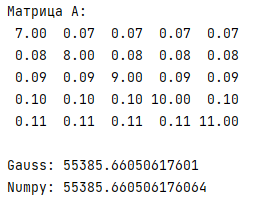
Найти определитель матрицы A и обратную к ней, A строится следующим образом:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| V | V \* 10-2 | V \* 10-2 | V \* 10-2 | V \* 10-2 |
| (V + 1) \* 10-2 | V + 1 | (V + 1) \* 10-2 | (V + 1) \* 10-2 | (V + 1) \* 10-2 |
| (V + 2) \* 10-2 | (V + 2) \* 10-2 | V + 2 | (V + 2) \* 10-2 | (V + 2) \* 10-2 |
| (V + 3) \* 10-2 | (V + 3) \* 10-2 | (V + 3) \* 10-2 | V + 3 | (V + 3) \* 10-2 |
| (V + 4) \* 10-2 | (V + 4) \* 10-2 | (V + 4) \* 10-2 | (V + 4) \* 10-2 | V + 4 |

*Код (нахождение определителя)*

import gauss  
import numpy as np  
from copy import deepcopy  
  
  
def determinant(M):  
 n = len(M)  
 res = deepcopy(M)  
  
 mas = []  
  
 for i in range(n):  
 elem = res[i][i]  
 mas.append(elem)  
 for j in range(n):  
 res[i][j] /= elem  
 for k in range(i + 1, n):  
 elem = res[k][i]  
 for j in range(n):  
 res[k][j] -= res[i][j] \* elem  
  
 det = 1  
 for i in range(len(mas)):  
 det \*= mas[i]  
  
 print(f"Gauss: {det}")  
  
  
def determiner\_np(M):  
 A = np.array(M, float)  
 print(f"Numpy: {np.linalg.det(A)}")  
  
  
v = 7 *# номер варианта*n = 5 *# размерность матрицы*main\_diag = gauss.make\_main\_diag(v, n) *# создаем список элементов главной диагонали*A = gauss.generate\_A(main\_diag, n) *# создаем матрицу А*print("Матрица A:")  
for row in A:  
 for c in row:  
 print(f'{c: >5.2f}', end=" ")  
 print()  
print()  
  
  
determinant(A)  
determiner\_np(A)

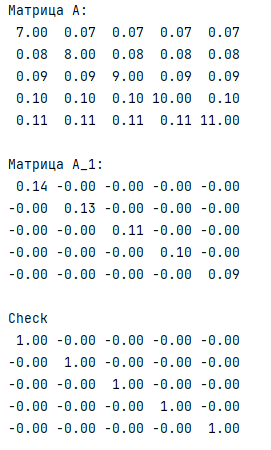
*Результат работы программы:*

**

*Код (нахождение обратной матрицы)*

import gauss  
import numpy as np  
  
  
def inverse(A):  
 result = []  
 for i in range(0, n):  
 b\_k = [0 for \_ in range(0, i)] + [1] + [0 for \_ in range(i + 1, n)]  
 result.append(gauss.method\_Gauss(A, b\_k))  
 return result  
  
  
def check(M1, M2):  
 A = np.array(M1)  
 B = np.array(M2)  
 return np.multiply(A,B)  
  
  
v = 7 *# номер варианта*n = 5 *# размерность матрицы*main\_diag = gauss.make\_main\_diag(v, n) *# создаем список элементов главной диагонали*A = gauss.generate\_A(main\_diag, n) *# создаем матрицу А*print("Матрица A:")  
for row in A:  
 for c in row:  
 print(f'{c: >5.2f}', end=" ")  
 print()  
print()  
  
A\_1 = inverse(A)  
print("Матрица A\_1:")  
for row in A\_1:  
 for c in row:  
 print(f'{c: >5.2f}', end=" ")  
 print()  
print()  
  
  
E = check(A, A\_1)  
  
print("Check")  
for row in E:  
 for c in row:  
 print(f'{c: >5.2f}', end=" ")  
 print()

*Результат работы программы:*

**

2.3 Решение СЛАУ методом прогонки

*Формулировка решаемой задачи.*

Здесь и далее V обозначает номер варианта. Номер варианта V = 7.

Методом прогонки решить СЛАУ вида Ax=b, где A и b строятся следующим образом:

A=

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| V | V \* 10-2 | 0 | 0 | 0 |
| (V + 1) \* 10-2 | V + 1 | (V + 1) \* 10-2 | 0 | 0 |
| 0 | (V + 2) \* 10-2 | V + 2 | (V + 2) \* 10-2 | 0 |
| 0 | 0 | (V + 3) \* 10-2 | V + 3 | (V + 3) \* 10-2 |
| 0 | 0 | 0 | (V + 4) \* 10-2 | V + 4 |

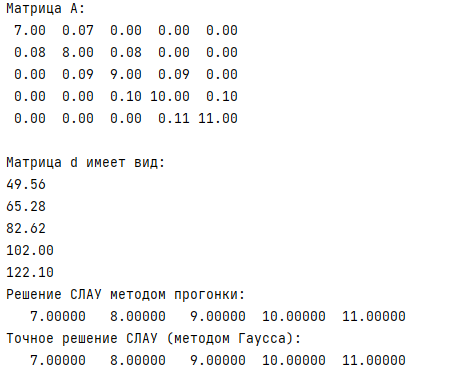
b = A \* (V, V + 1, V + 2, V + 3, V + 4) T

*Код*

import gauss  
  
  
def run\_through(A, d):  
 n = len(A)  
 a = [0] + [A[i][i - 1] for i in range(1, n - 1)] + [A[n - 1][n - 2]]  
 b = [-A[i][i] for i in range(0, n - 1)] + [-A[n - 1][n - 1]]  
 c = [A[i][i + 1] for i in range(0, n - 1)] + [0]  
  
 p = [c[0] / b[0]]  
 q = [-d[0] / b[0]]  
  
 for i in range(n - 1):  
 p.append(c[i + 1] / (b[i + 1] - a[i + 1] \* p[i]))  
 q.append((a[i + 1] \* q[i] - d[i + 1]) / (b[i + 1] - a[i + 1] \* p[i]))  
  
 x = [0 for \_ in range(n)]  
 x[n - 1] = q[n - 1]  
 for i in range(n - 1, 0, -1):  
 x[i - 1] = p[i - 1] \* x[i] + q[i - 1]  
  
 return x  
  
  
def generate\_A\_for\_run(main\_diag, n):  
 A = []  
 for i in range(n):  
 tmp = []  
 for j in range(n):  
 if i == j:  
 tmp.append(main\_diag[i])  
 elif i == j - 1 or i == j + 1:  
 tmp.append(main\_diag[i] / 100)  
 else:  
 tmp.append(0)  
 A.append(tmp)  
 return A  
  
  
def count\_d(A, list\_for\_variant):  
 return [sum([A[i][j] \* list\_for\_variant[j] for j in range(n)]) for i in range(n)]  
  
  
v = 7 *# номер варианта*n = 5 *# размерность матрицы*main\_diag = gauss.make\_main\_diag(v, n) *# создаем список элементов главной диагонали*A = generate\_A\_for\_run(main\_diag, n) *# создаем матрицу А*print("Матрица A:")  
for row in A:  
 for c in row:  
 print(f'{c: >5.2f}', end=" ")  
 print()  
print()  
  
d = count\_d(A, main\_diag)  
print("Матрица d имеет вид:")  
for c in d:  
 print(f'{c: >5.2f}')  
  
x = run\_through(A, d) *# решаем систему Ax = b*print("Решение СЛАУ методом прогонки:")  
for c in x:  
 print(f'{c: >10.5f}', end="")  
print()  
  
print("Точное решение СЛАУ (методом Гаусса):")  
for c in gauss.method\_Gauss(A, d):

print(f'{c: >10.5f}', end="")

*Результат работы программы:*

**

РАЗДЕЛ 3 Численные методы решения дифференциальных уравнений

* 1. Метод Эйлера решения задачи Коши для ОДУ первого порядка

Формулировка решаемой задачи.

Здесь и далее V обозначает номер варианта. Номер варианта V = 7.

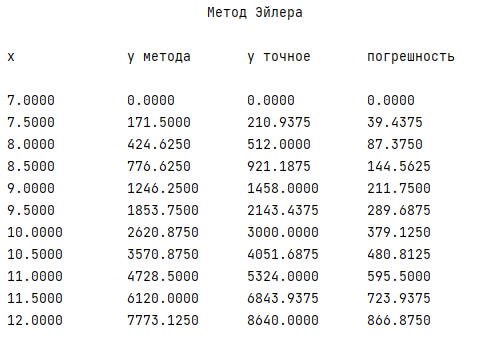
Решить задачу Коши

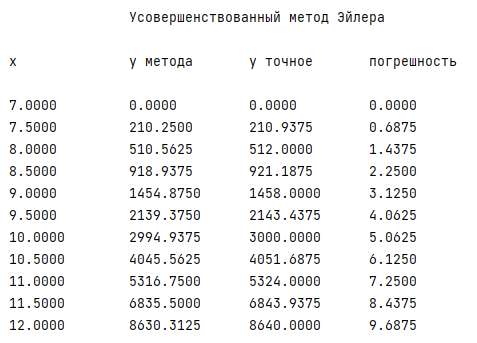
методом Эйлера, усовершенствованным методом Эйлера, методом предиктора-корректора.

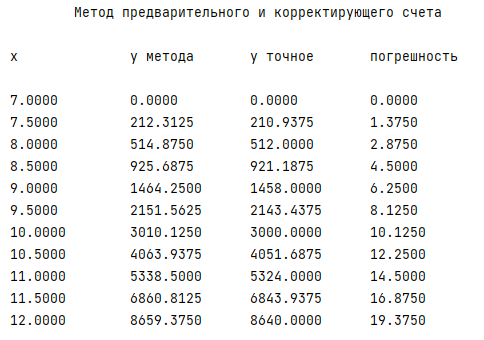
*Код*

V = 7  
x\_0 = V  
y\_0 = 0  
h = 0.5  
lft = V  
rgt = V + 5  
eps = 1e-9  
  
  
def y(x):  
 return x \*\* 4 - V \* x \*\* 3  
  
  
def f(x, y):  
 return 4 \* x \*\* 3 - 3 \* V \* x \*\* 2  
  
  
def y\_k\_euler(x\_k, y):  
 return y + h \* f(x\_k, y)  
  
  
def y\_k\_euler\_improved(x\_k, y\_k):  
 return y\_k + h \* f(x\_k + h / 2, y\_k + h / 2 \* f(x\_k, y\_k))  
  
  
def y\_k\_correct(x\_k, x\_k\_1, y\_k):  
 return y\_k + (h / 2) \* (f(x\_k, y\_k) + f(x\_k\_1, y\_k + h \* f(x\_k, y\_k)))  
  
  
def print\_table(lst1, lst2, lst3, lst4):  
 s = 45  
 print(f'{"x": <15}{"y метода": <15}{"y точное": <15}{"погрешность": <15}\n')  
  
 for i in range(len(lst1)):  
 print(f'{lst1[i]: <15.4f}{lst2[i]: <15.4f}{lst3[i]: <15.4f}{lst4[i]: <15.4f}')  
  
  
x\_h = [x\_0]  
y\_x = [y\_0]  
  
*# метод Эйлера*y\_euler = [y\_0]  
eps\_eul\_met = [abs(y\_x[0] - y\_euler[0])]  
*# Усовершенствованный метод Эйлера*y\_euler\_improved = [y\_0]  
eps\_eul\_improved = [abs(y\_x[0] - y\_euler\_improved[0])]  
  
*# Метод предварительного и корректирующего счета*y\_cor = [y\_0]  
eps\_cor = [abs(y\_x[0] - y\_cor[0])]  
  
x = lft  
k = 1  
while x < rgt - eps:  
 x\_h.append(x + h)  
 y\_x.append(y(x + h))  
  
 *# метод Эйлера* y\_euler.append(y\_k\_euler(x\_h[k - 1], y\_euler[k - 1]))  
 eps\_eul\_met.append(abs(y\_x[k] - y\_euler[k]))  
  
 *# Усовершенствованный метод Эйлера* y\_euler\_improved.append(y\_k\_euler\_improved(x\_h[k - 1], y\_euler\_improved[k - 1]))  
 eps\_eul\_improved.append(abs(y\_x[k] - y\_euler\_improved[k]))  
  
 *# Метод предварительного и корректирующего счета* y\_cor.append(y\_k\_correct(x\_h[k - 1], x\_h[k], y\_cor[k - 1]))  
 eps\_cor.append(abs(y\_x[k] - y\_cor[k]))  
  
 x += h  
 k += 1  
  
print(f'{"Метод Эйлера": ^63}\n')  
print\_table(x\_h, y\_euler, y\_x, eps\_eul\_met)  
print()  
  
print(f'{"Усовершенствованный метод Эйлера": ^63}\n')  
print\_table(x\_h, y\_euler\_improved, y\_x, eps\_eul\_improved)  
print()  
  
print(f'{"Метод предварительного и корректирующего счета": ^63}\n')  
print\_table(x\_h, y\_cor, y\_x, eps\_cor)

Результат работы программы:







3.2 Разностный метод решения краевой задачи для ОДУ второго порядка

*Формулировка решаемой задачи.*

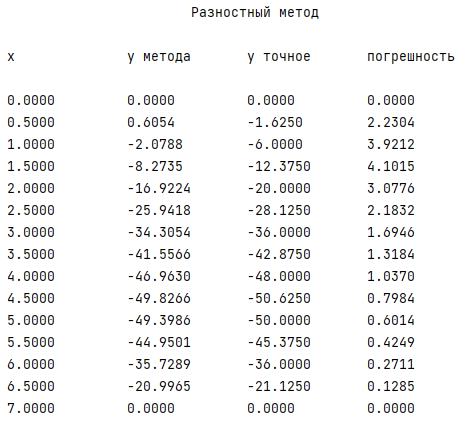
Здесь и далее T = V обозначает номер варианта. Номер варианта T =V = 7.

Решить краевую задачу , c точным решением разностным методом

*Код*

import numpy as np  
  
import gauss  
  
V = 7  
x\_0 = 0  
x\_T = V  
y\_0 = 0  
y\_T = 0  
h = 0.5  
eps = 1e-9  
  
  
def y(x):  
 return x \*\* 3 - V \* x \*\* 2  
  
  
def f(x):  
 return 4 \* x \*\* 4 - 3 \* V \* x \*\* 3 + 6 \* x - 2 \* V  
  
  
def p(x):  
 return x \*\* 2  
  
  
def q(x):  
 return x  
  
  
def coef\_y\_k\_min\_1(x):  
 return 1 / (h \*\* 2) - p(x) / (2 \* h)  
  
  
def coef\_y\_k(x):  
 return -2 / (h \*\* 2) + q(x)  
  
  
def coef\_y\_k\_plus\_1(x):  
 return 1 / (h \*\* 2) + p(x) / (2 \* h)  
  
  
def print\_table(lst1, lst2, lst3, lst4):  
 s = 45  
 print(f'{"x": <15}{"y метода": <15}{"y точное": <15}{"погрешность": <15}\n')  
  
 for i in range(len(lst1)):  
 print(f'{lst1[i]: <15.4f}{lst2[i]: <15.4f}{lst3[i]: <15.4f}{lst4[i]: <15.4f}')  
  
lft = 0  
rgt = V  
x\_h = [x\_0]  
y\_ex = [y\_0]  
n = int(rgt / h)  
  
x = lft  
while x < rgt - eps:  
 x\_h.append(x + h)  
 y\_ex.append(y(x + h))  
 x += h  
  
*# СЛАУ*cur\_lst = [1] + [0] \* n  
  
A = [cur\_lst] *# первая строка матрицы A – коэф1 при y0 и 0 при остальных y*b = [0] *# первое значение столбца b равно 0*for k in range(1, n):  
 cur\_lst = [0] \* (n + 1)  
 cur\_lst[k - 1] = coef\_y\_k\_min\_1(x\_h[k])  
 cur\_lst[k] = coef\_y\_k(x\_h[k])  
 cur\_lst[k + 1] = coef\_y\_k\_plus\_1(x\_h[k])  
 A.append(cur\_lst)  
 b.append(f(x\_h[k]))  
  
cur\_lst = [0] \* n + [1]  
A.append(cur\_lst) *# последняя строка матрицы A - коэф. 1 при y\_n и 0 при остальных y*b.append(0) *# последнее значение столбца b равно 0*y\_dif = gauss.method\_Gauss(A, b)  
e = [abs(y\_dif[i] - y\_ex[i]) for i in range(n + 1)]  
  
print(f'{"Разностный метод": ^63}\n')  
print\_table(x\_h, y\_dif, y\_ex, e)

*Результат работы программы:*



* 1. Метод неопределенных коэффициентов решения краевой задачи для ОДУ второго порядка

*Формулировка решаемой задачи.*

Здесь и далее T = V обозначает номер варианта. Номер варианта T =V = 7.

Решить краевую задачу , c точным решением методом неопределенных коэффициентов.

*Код*

import gauss  
import numpy  
  
V = 7  
x\_0 = 0  
x\_T = V  
y\_0 = 0  
y\_T = 0  
n = 10  
eps = 1e-9  
  
  
def y(x):  
 return x \*\* 3 - V \* x \*\* 2  
  
  
def f(x):  
 return 4 \* x \*\* 4 - 3 \* V \* x \*\* 3 + 6 \* x - 2 \* V  
  
  
def p(x):  
 return x \*\* 2  
  
  
def q(x):  
 return x  
  
  
*# фи в точке икс*def fi\_k(x, k):  
 return x \*\* k \* (x - V)  
  
  
*# фи штрих в точке икс*def fi\_k\_prime(x, k):  
 return k \* x \*\* (k - 1) \* (x - V) + x \*\* k  
  
  
*# фи два штриха в точке икс*def fi\_k\_double\_prime(x, k):  
 return k \* (k - 1) \* x \*\* (k - 2) \* (x - V) + 2 \* k \* x \*\* (k - 1)  
  
  
def print\_table(lst1, lst2, lst3, lst4):  
 s = 45  
 print(f'{"x": <20}{"y метода": <20}{"y точное": <20}{"погрешность": <30}\n')  
  
 for i in range(len(lst1)):  
 print(f'{lst1[i]: <20.5f}{lst2[i]: <20.5f}{lst3[i]: <20.5f}{lst4[i]: <30.20f}')  
  
  
lft = 0  
rgt = V  
h = (rgt - lft) / (n + 1)  
x\_h = [x\_0 + i \* h for i in range(n + 2)]  
y\_ex = [y(x) for x in x\_h]  
  
  
*# СЛАУ*A = []  
b = []  
  
for j in range(1, n + 1):  
 cur\_lst = []  
 for k in range(1, n+1):  
 cur\_sum = fi\_k\_double\_prime(x\_h[j], k) + p(x\_h[j]) \* fi\_k\_prime(x\_h[j], k) + q(x\_h[j]) \* fi\_k(x\_h[j], k)  
 cur\_lst.append(cur\_sum)  
 A.append(cur\_lst)  
 b.append(f(x\_h[j]))  
  
sol = gauss.method\_Gauss(A, b)  
  
y\_met = []  
for i in range(len(sol) + 1):  
 cur\_sum = 0  
 for k in range(1, n + 1):  
 cur\_sum += sol[k - 1] \* fi\_k(x\_h[i], k)  
 y\_met.append(cur\_sum)  
y\_met.append(0)  
e = [abs(y\_met[i] - y\_ex[i]) for i in range(len(y\_met))]  
  
print(f'{"Метод неопределенных коэффициентов": ^63}\n')  
print\_table(x\_h, y\_met, y\_ex, e)

*Результат работы программы:*

**

РАЗДЕЛ 4 Численные методы решения интегральных уравнений

4.1 Решение интегрального уравнения Фредгольма в случае вырожденного ядра

*Формулировка решаемой задачи.*

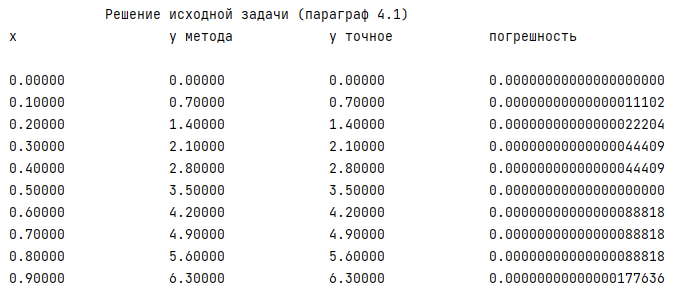
Решить следующее интегральное уравнение

где V = 7 (вариант), y(x) = Vx – точное решение

*Код*

import gauss  
import numpy  
  
V = 7  
n = 3  
h = 0.1  
  
  
def y(x):  
 return V \* x  
  
  
def f\_x(x):  
 return V \* ((4 \* x) / 3 + (x \*\* 2) / 4 + (x \*\* 3) / 5)  
  
  
def a\_i(x, i):  
 return x \*\* i  
  
  
def b\_i(t, i):  
 return t \*\* i  
  
  
def print\_table(lst1, lst2, lst3, lst4):  
 s = 45  
 print(f'{"x": <20}{"y метода": <20}{"y точное": <20}{"погрешность": <30}\n')  
  
 for i in range(len(lst1)):  
 print(f'{lst1[i]: <20.5f}{lst2[i]: <20.5f}{lst3[i]: <20.5f}{lst4[i]: <30.20f}')  
  
  
x\_h = [i / 10 for i in range(10)]  
y\_ex = [y(x) for x in x\_h]  
  
lst\_A = [[1 / (i + k + 1) for k in range(1, n + 1)] for i in range(1, n + 1)]  
  
lst\_phi = [28 / 9 + 7 / 16 + 7 / 25, 7 / 3 + 7 / 20 + 7 / 30, 28 / 15 + 7 / 24 + 1 / 5]  
  
*# СЛАУ*A = []  
b = []  
for k in range(n):  
 cur\_lst = []  
 for i in range(n):  
 if k == i:  
 cur\_lst.append(lst\_A[i][k] + 1)  
 else:  
 cur\_lst.append(lst\_A[i][k])  
 A.append(cur\_lst)  
 b.append(lst\_phi[k])  
  
lst\_q = gauss.method\_Gauss(A, b)  
  
y\_met = []  
for k in range(len(x\_h)):  
 cur\_sum = 0  
 for i in range(n):  
 cur\_sum += lst\_q[i] \* a\_i(x\_h[k], i + 1)  
 y\_met.append(f\_x(x\_h[k]) - cur\_sum)  
  
e = [abs(y\_met[i] - y\_ex[i]) for i in range(len(x\_h))]  
print(f'{"Решение исходной задачи (параграф 4.1)": ^63}')  
print\_table(x\_h, y\_met, y\_ex, e)

*Результат работы программы:*



**4.2 Квадратурный метод решения интегрального уравнения Фредгольма**

*Формулировка решаемой задачи.*

Решить следующее интегральное уравнение

где V = 7 (вариант), y(x) = Vx – точное решение

*Код*

import gauss  
import numpy  
  
V = 7  
n = 11  
h = 0.1  
  
  
def y(x):  
 return V \* x  
  
  
def f\_x(x):  
 return V \* ((4 \* x) / 3 + (x \*\* 2) / 4 + (x \*\* 3) / 5)  
  
  
def A\_k\_j(x, t):  
 return x \* t + x \*\* 2 \* t \*\* 2 + x \*\* 3 \* t \*\* 3  
  
  
def print\_table(lst1, lst2, lst3, lst4):  
 s = 45  
 print(f'{"x": <20}{"y метода": <20}{"y точное": <20}{"погрешность": <30}\n')  
  
 for i in range(len(lst1)):  
 print(f'{lst1[i]: <20.5f}{lst2[i]: <20.5f}{lst3[i]: <20.5f}{lst4[i]: <30.20f}')  
  
  
x\_h = [i / 10 for i in range(n - 1)]  
y\_ex = [y(x) for x in x\_h]  
  
lambda\_ = 1  
  
*# СЛАУ*A = []  
b = []  
for k in range(n - 1):  
 cur\_lst = []  
 for j in range(n - 1):  
 if k == j:  
 cur\_lst.append(lambda\_ \* h \* A\_k\_j(x\_h[k], x\_h[j]) + 1)  
 else:  
 cur\_lst.append(lambda\_ \* h \* A\_k\_j(x\_h[k], x\_h[j]))  
 A.append(cur\_lst)  
 b.append(f\_x(x\_h[k]))  
  
y\_met = gauss.method\_Gauss(A, b)  
  
e = [abs(y\_met[i] - y\_ex[i]) for i in range(n - 1)]  
print(f'{"Решение исходной задачи (параграф 4.2)": ^63}')  
print\_table(x\_h, y\_met, y\_ex, e)

*Результат работы программы:*

