Самарский национальный исследовательский университет

имени академика С.П. Королева

Естественнонаучный институт

Механико-математический факультет

Кафедра информатики и вычислительной математики

Отчет по лабораторной работе №3 «Численное решение систем линейных алгебраических уравнений специального вида»

Выполнил:

студент группы 4345-020303D

С.С. Ильметов

Проверил:

доцент В.П. Сироченко

Самара-2024

1. **Решение систем линейных алгебраических уравнений с симметричной матрицей методом квадратного корня**

**1.1 Постановка задачи**

Реализовать метод квадратного корня в виде компьютерной программы и проверить ее на тестовых задачах

**1.2 Краткое описание численного метода**

Решение систем линейных алгебраических уравнений (СЛАУ) является одной из самых распространенных и важных задач вычислительной математики, т.к. к ней сводятся многочисленные научные и практические задачи.

Метод квадратного корня используется для решения линейных систем следующего вида:

Здесь A = {aij}, i, j = 1, n; - симметричная матрица, т.е. aij = a ji

Поскольку А - симметричная матрица, то ее можно представить в виде произведения двух взаимно транспонированных между собой треугольных матриц.

Метод квадратного корня основан на разложении матрицы А в произведение

где S – верхняя треугольная матрица, – транспонированная к ней матрица, D –диагональная матрица, на диагонали которой находятся числа ±1:

Если матрицы S и D получены, то система уравнений Ax = b заменяется системой

Обозначим y = Sx. Тогда решение исходной системы сводится к последовательному решению двух систем уравнений с треугольными матрицами

Sx=y

Общий вид формул для нахождения элементов матриц D и S при i = 1,…,n

После нахождения элементов D и S решение исходной системы сводится к решению двух систем с треугольными матрицами:

Sx=y

Здесь – нижняя треугольная матрица, S – верхняя треугольная матрица.

* 1. **Листинг программы**

Для реализации программы была выбран язык программирования Python 3.12.2 . В качестве IDLE был выбран Visual Code.

**from math import \***

**def sign(a):**

**if a > 0.0:**

**return 1**

**if a < 0.0:**

**return -1**

**return 0**

**with open("matrix2.txt", "r") as fin:**

**# Читаем размер матрицы**

**n = int(fin.readline())**

**m = n + 1 # Расширенная матрица**

**# Создаем матрицу**

**A = [[0 for j in range(m + 1)] for i in range(m)]**

**# Читаем элементы матрицы**

**for i in range(1, m):**

**row = fin.readline().split()**

**for j in range(1, m + 1):**

**A[i][j] = float(row[j - 1])**

**S = [[0 for j in range(m)] for i in range(m)]**

**D = [[0 for j in range(m)] for i in range(m)]**

**S[1][1] = sqrt(abs(A[1][1]))**

**D[1][1] = sign(A[1][1])**

**for j in range(2, n + 1):**

**S[1][j] = A[1][j] / (S[1][1] \* D[1][1])**

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

**summa = 0**

**for l in range(1, i):**

**summa += S[l][i] \* S[l][i] \* D[l][l]**

**S[i][i] = sqrt(abs(A[i][i] - summa))**

**D[i][i] = sign(A[i][i] - summa)**

**for j in range(i + 1, n + 1):**

**summa = 0**

**for k in range(1, i):**

**summa += S[k][i] \* S[k][j] \* D[k][k]**

**S[i][j] = (A[i][j] - summa) / (S[i][i] \* D[i][i])**

**y = [0] \* m**

**y[1] = A[1][m] / (S[1][1] \* D[1][1])**

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

**summa = 0**

**for k in range(1, i):**

**summa += S[k][i] \* y[k] \* D[k][k]**

**y[i] = (A[i][m] - summa) / (S[i][i] \* D[i][i])**

**x = [0] \* m**

**x[n] = y[n] / S[n][n]**

**for i in range(n-1, 0, -1):**

**summa = 0**

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

**summa += S[i][k] \* x[k]**

**x[i] = (y[i] - summa) / S[i][i]**

**for i in range(1, n + 1):**

**print(x[i])**

* 1. **Результаты расчетов**

|  |  |
| --- | --- |
| **Входные данные** | **Выходные данные** |
| **3**  **1 2 1 1**  **2 -2 2 1**  **1 2 -1 2** | **1.1666666666666665**  **0.1666666666666667**  **-0.49999999999999994** |
| **4**  **5 -5 6 7 8**  **-1 1 3 -5 -10**  **1 2 2 3 4**  **2 -2 -3 -3 3** | **0.702914798206278**  **-0.48514573991031396**  **-0.6216367713004485**  **0.82707399103139** |

Входные данные записаны в файл matrix.txt:

**Система уравнений с матрицей Гильберта**

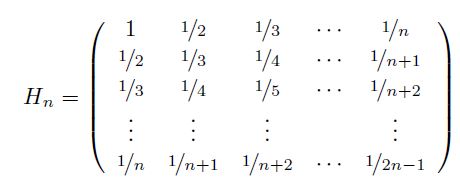
1. **Постановка задачи**

Решить систему с матрицей Гилберта H и вектором b методом квадратного корня

1. **Краткое описание численного метода**

Матрица Гильберта H имеет элементы:

Матрица Гильберта H имеет вид:



Матрица Гильберта симметрична и очень плохо обусловлена даже для небольших значений n.

Зададим точное решение системы вектор x = (T формулой i, где  
i = 1, …, n   
N – номер студента в списке группы и равен 10, а вектор правых частей b = (b1, b2, …, bn)T определим по заданному решению: b = Hx

**3**. **Листинг программы**

Для реализации программы была выбран язык программирования Python 3.12.2 . В качестве IDLE был выбран Visual Code.

from math import \*

def sign(a):

if a == 0.0:

return 0

if a < 0.0:

return -1

return 1

n = int(input("Введите размер матрицы: "))

m = n + 1 # Расширенная матрица

A = [[0 for j in range(m + 1)] for i in range(n + 1)]

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

for j in range(1, m + 1):

A[i][j] = 1.0 / (i + j - 1)

print("Точное решение системы: ")

X = [0] \* m

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

X[i] = 10 \* i

print(X[i], end=" ")

print()

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

summa = 0

for j in range(1, m):

summa += A[i][j] \* X[j]

A[i][m] = summa

S = [[0 for j in range(m)] for i in range(m)]

D = [[0 for j in range(m)] for i in range(m)]

S[1][1] = sqrt(abs(A[1][1]))

D[1][1] = sign(A[1][1])

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

S[1][j] = A[1][j] / (S[1][1] \* D[1][1])

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

summa = 0

for l in range(1, i):

summa += S[l][i] \* S[l][i] \* D[l][l]

S[i][i] = sqrt(abs(A[i][i] - summa))

D[i][i] = sign(A[i][i] - summa)

for j in range(i + 1, n + 1):

summa = 0

for k in range(1, i):

summa += S[k][i] \* S[k][j] \* D[k][k]

S[i][j] = (A[i][j] - summa) / (S[i][i] \* D[i][i])

y = [0] \* m

y[1] = A[1][m] / (S[1][1] \* D[1][1])

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

summa = 0

for k in range(1, i):

summa += S[k][i] \* y[k] \* D[k][k]

y[i] = (A[i][m] - summa) / (S[i][i] \* D[i][i])

x = [0] \* m

x[n] = y[n] / S[n][n]

for i in range(n-1, 0, -1):

summa = 0

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

summa += S[i][k] \* x[k]

x[i] = (y[i] - summa) / S[i][i]

print("Расчитанное решение системы: ")

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

print(x[i])

summa = 0

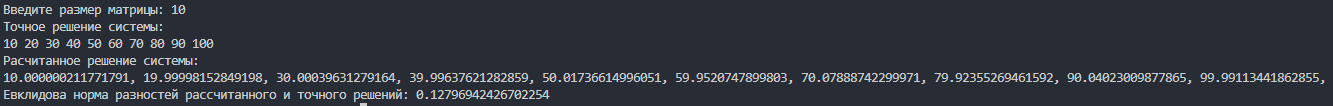
for i in range(1, n + 1):

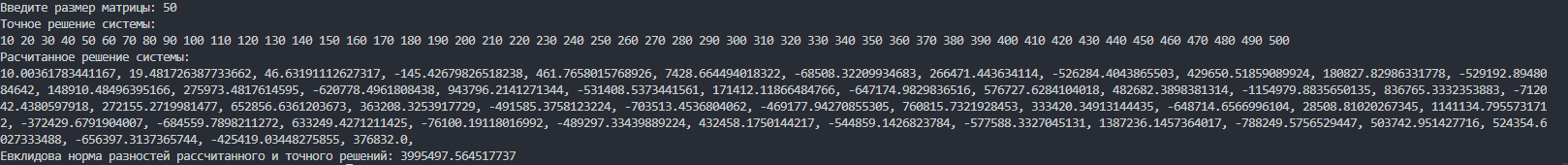
summa += (abs(x[i] - X[i]) \* (abs(x[i] - X[i])))

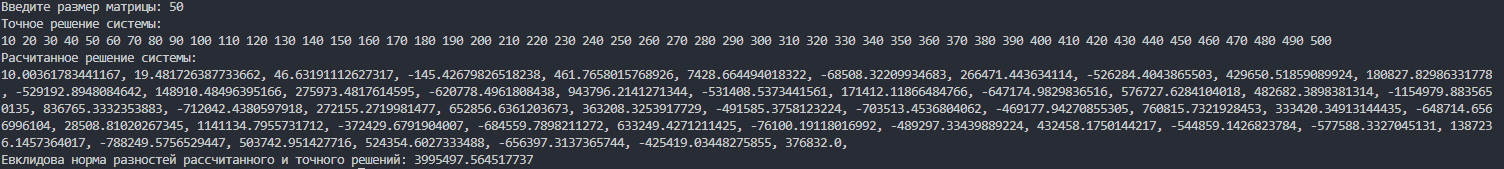
norma = sqrt(summa)

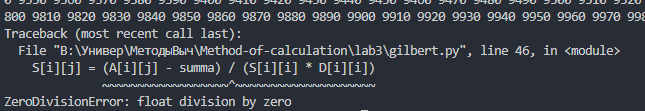
print("Евклидова норма разностей рассчитанного и точного решений:", norma)

1. **Результаты расчетов**

Решена система с матрицей Гилберта методом квадратного корня для n = 10;50:100;1000 Рисунок 1 – Результат выполнения программы при N = 10

Рисунок 2 – Результат выполнения программы при N = 50

Рисунок 3 – Результат выполнения программы при N=100

Рисунок 4 – Результат выполнения программы при N = 1000

**РЕШЕНИЕ СИСТЕМЫ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ С ТРЕХДИАГОНАЛЬНОЙ МАТРИЦЕЙ МЕТОДОМ ПРОГОНКИ**

**2.1 Постановка задачи**

Реализовать метод прогонки в виде компьютерной программы и решить систему этим методом:

Число уравнений n = 101; 301. Параметр d = 0; 0,5; 1.

**2.2 Краткое описание численного метода**

Метод прогонки является частным случаем метода Гаусса и используется для решения систем линейных уравнений вида Ax = f, где A – трёхдиагональная матрица.

Трёхдиагональной матрицей называется матрица такого вида, где во всех остальных местах, кроме главной диагонали и двух соседних с ней, стоят нули.

В общем виде система Ax = f имеет следующую матрицу A:

– векторы-столбцы неизвестных

– свободные члены

Система выглядит так:

Для нахождения решения установим связь неизвестных компонент xi:

,

,

Метод прогонки состоит из двух этапов: прямой ход и обратной ход. На первом этапе определяются прогоночные коэффициенты , а на втором – находят неизвестные x.

- неизвестные коэффициенты, которые нужно найти.

После того как прогоночные коэффициенты найдены, решение системы Ax = f находится по формуле , начиная с 𝑥𝑛−1 и вплоть до 𝑥1. Но для начала счета по этой формуле требуется знать 𝑥𝑛, которое определяется как решение системы двух уравнений: уравнения при 𝑖 = 𝑛 − 1 и последнего уравнения системы Ax = f

**2.3 Листинг программы**

Для реализации программы была выбран язык программирования Python 3.12.2 . В качестве IDLE был выбран Visual Code.

n = int(input("Введите размер матрицы: "))

d = float(input("Введите параметр: "))

A = [0] \* (n + 1)

B = [0] \* (n + 1)

C = [0] \* (n + 1)

F = [0] \* (n + 1)

C[1] = 1

B[1] = -d

F[1] = 0

for i in range(2, n):

A[i] = 1

C[i] = -2

B[i] = 1

F[i] = 10 \* ((i-1) / (n-1))\*\*2

A[n] = 1

C[n] = -1

F[n] = 10

print("A:")

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

print(A[i], end=" ")

print()

print("C:")

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

print(A[i], end=" ")

print()

print("B:")

for i in range(1, n):

print(A[i], end=" ")

print()

print("F:")

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

print(A[i], end=" ")

print()

x = [0] \* (n + 1)

alpha = [0] \* (n + 1)

beta = [0] \* (n + 1)

alpha[1] = -B[1] / C[1]

beta[1] = F[1] / C[1]

for i in range(2, n):

tmp = (C[i] + A[i] \* alpha[i - 1])

alpha[i] = -B[i] / tmp

beta[i] = (F[i] - A[i] \* beta[i - 1]) / tmp

x[n] = (F[n] - A[n] \* beta[n - 1]) / (C[n] + A[n] \* alpha[n - 1])

for i in range(n - 1, 0, -1):

x[i] = alpha[i] \* x[i + 1] + beta[i]

print("Решение:")

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

print(x[i])

**2.4 Результаты расчетов**

Система, которую необходимо решить выглядит следующим образом:

Число уравнений n = 101; 301. Параметр d = 0; 0,5; 1.

Также необходимо получить задачу при n=3, затем решить ее и сравнить найденное решение с результатом расчета с помощью компьютерной программы.

Решим тестовую задачу. Возьмем n=3, N=10, d=0. Найдем коэффициенты трехдиагональной матрицы из данной системы:

Получаем:

Матрица имеет вид:

Коэффициенты правой части:

Из первого уравнения видно, что

Найдем , сложив вторую и третью строки   
 -12.5

Тогда -22.5

Из данного уравнения следует, что = 0, = -12.5, = -.5.

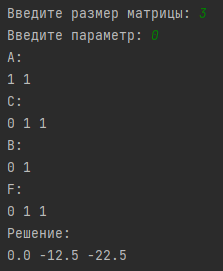
******

Рисунок 12 – Результат выполнения программы

Были проверены разные случаи, когда число уравнений n = 101; 301, а параметр d = 0; 0,5; 1.

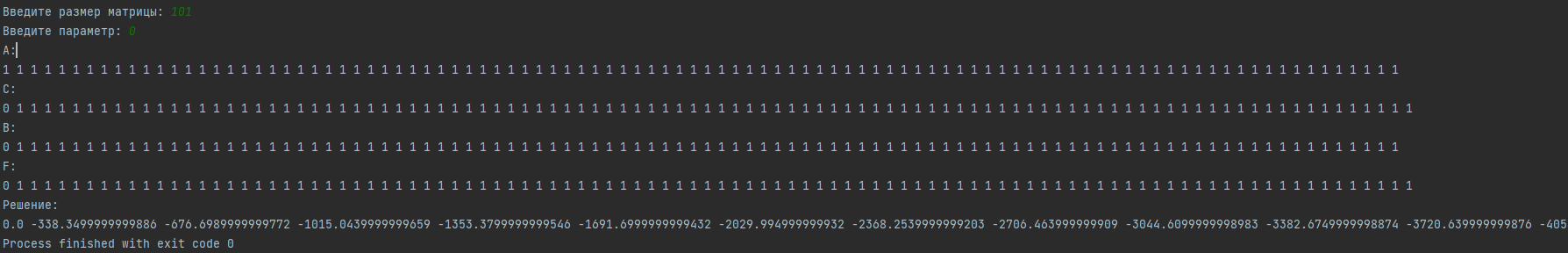
******

Рисунок 13 – Результат выполнения программы

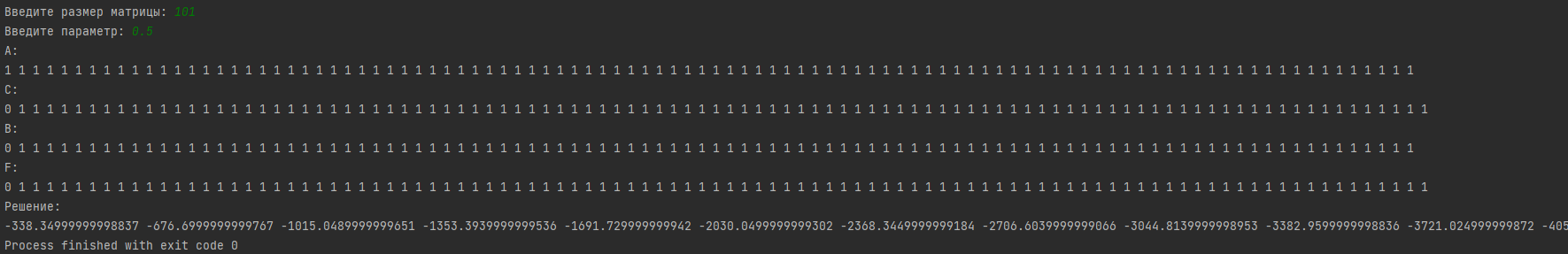


Рисунок 14 – Результат выполнения программы

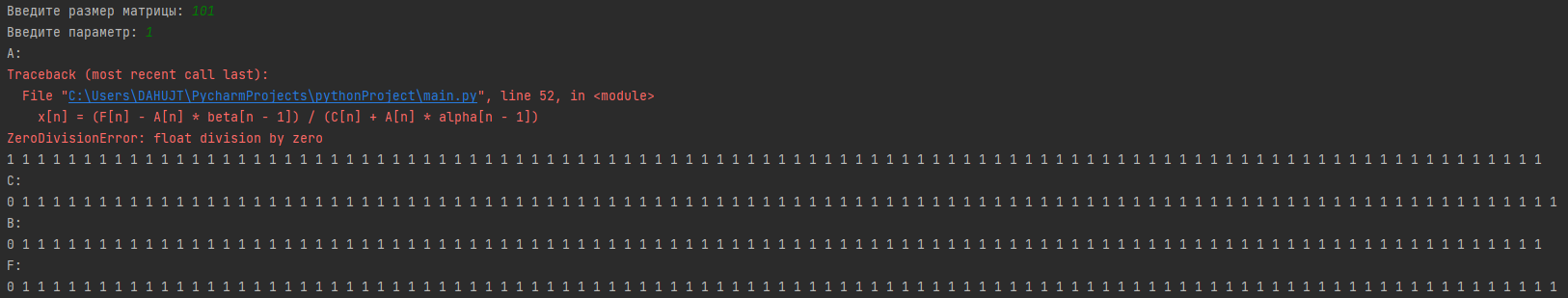


Рисунок 15 – Результат выполнения программы

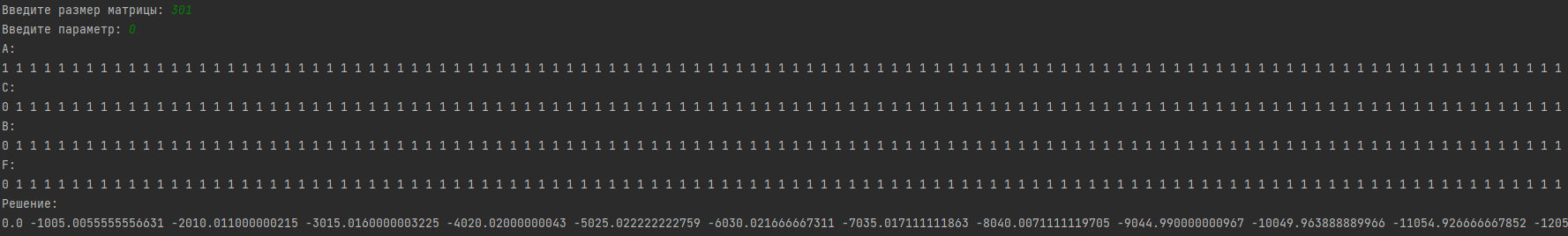


Рисунок 16 – Результат выполнения программы

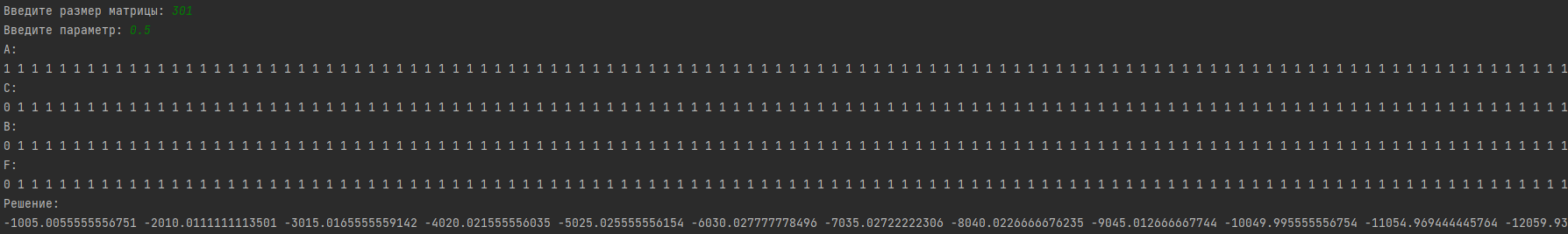


Рисунок 17 – Результат выполнения программы

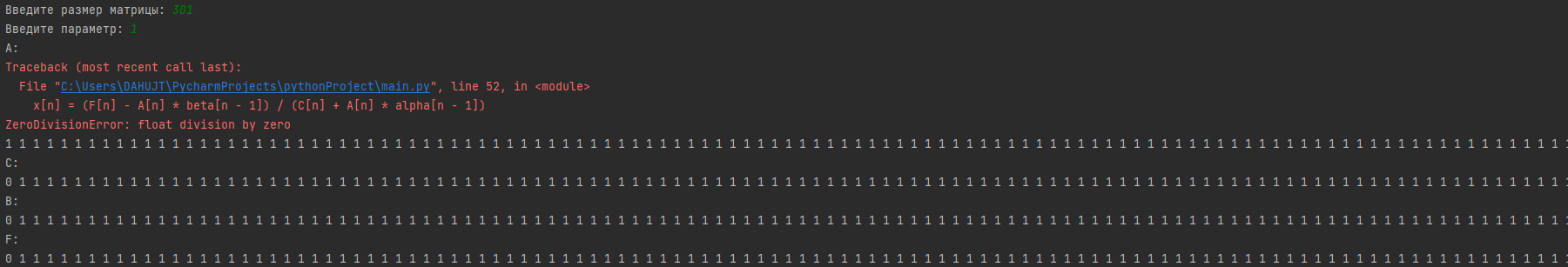


Рисунок 18 – Результат выполнения программы