**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ**

**УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**



ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

**Лабораторная работа №2**

по дисциплине: Вычислительная математика

тема: «**Алгебра матриц. Быстрое умножение матриц. Вычисление обратной матрицы. Нахождение собственных чисел и собственных векторов матрицы**»

Выполнил: ст. группы ВТ-231

Масленников Даниил

Проверили:

Островский Алексей Мичеславович

Белгород 2025 г.

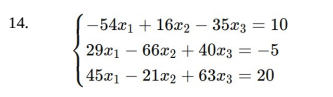
Л а б о р а т о р н а я р а б о т а № 2

### Цель работы:

Изучить алгебраические операции над матрицами, особенности алгоритмизации быстрых матричных алгоритмов (на примере умножения матриц), вычисления обратной матрицы, нахождения собственных чисел и собственных векторов матрицы.

### Задачи:

1. **Теоретическая часть**:
   * Изучить основные операции над матрицами: сложение, умножение, нахождение определителя, обратной матрицы, собственных чисел и векторов.
   * Рассмотреть алгоритм быстрого умножения матриц (алгоритм Штрассена) и его временную сложность.
2. **Практическая часть**:
   * Реализовать алгоритм Штрассена для умножения матриц на Python.
   * Сравнить результаты умножения матриц с использованием алгоритма Штрассена и стандартного метода.
   * Оценить временную сложность функции dot из библиотеки NumPy для умножения матриц.
   * Найти обратную матрицу для заданной матрицы A вручную и с использованием NumPy.
   * Найти собственные числа и собственные векторы матрицы A с использованием NumPy.
   * Решить систему линейных уравнений (СЛАУ) методом обратной матрицы и сравнить с результатами, полученными с использованием LU-разложения.
3. **Индивидуальное задание**:
   * Используя матрицу A из предыдущей лабораторной работы, выполнить следующие действия:
     + Найти обратную матрицу вручную.
     + Решить СЛАУ методом обратной матрицы.
     + Написать программу на Python для нахождения обратной матрицы и собственных чисел/векторов.
     + Сравнить результаты ручных вычислений с результатами, полученными с использованием NumPy.
4. **Анализ и отчет**:
   * Отразить в отчете все полученные результаты.
   * Сделать выводы о корректности работы алгоритмов и точности вычислений.
   * Приложить ручные расчеты и исходный код программ.

**Вариант 14**  
  
  
  


**Программное решение  
  
import** **numpy** **as** **np**

**from** **scipy.linalg** **import** lu\_factor, lu\_solve

**import** **time**

**import** **matplotlib.pyplot** **as** **plt**

# Функция для дополнения матрицы до ближайшей степени двойки

**def** **pad\_matrix**(matrix):

n = matrix.shape[**0**]

next\_power\_of\_two = **2** \*\* int(np.ceil(np.log2(n)))

padded\_matrix = np.zeros((next\_power\_of\_two, next\_power\_of\_two))

padded\_matrix[:n, :n] = matrix

**return** padded\_matrix

# Функция для удаления дополнения из матрицы

**def** **unpad\_matrix**(matrix, original\_size):

**return** matrix[:original\_size, :original\_size]

# Функция для умножения матриц по алгоритму Штрассена

**def** **strassen\_mat\_mul**(mat\_left, mat\_right):

size = mat\_left.shape[**0**]

**if** size == **1**:

**return** mat\_left \* mat\_right

**else**:

mid = size // **2**

left\_top = mat\_left[:mid, :mid]

right\_top = mat\_left[:mid, mid:]

left\_bottom = mat\_left[mid:, :mid]

right\_bottom = mat\_left[mid:, mid:]

top\_left = mat\_right[:mid, :mid]

top\_right = mat\_right[:mid, mid:]

bottom\_left = mat\_right[mid:, :mid]

bottom\_right = mat\_right[mid:, mid:]

m1 = strassen\_mat\_mul(left\_top + right\_bottom, top\_left + bottom\_right)

m2 = strassen\_mat\_mul(left\_bottom + right\_bottom, top\_left)

m3 = strassen\_mat\_mul(left\_top, top\_right - bottom\_right)

m4 = strassen\_mat\_mul(right\_bottom, bottom\_left - top\_left)

m5 = strassen\_mat\_mul(left\_top + right\_top, bottom\_right)

m6 = strassen\_mat\_mul(left\_bottom - left\_top, top\_left + top\_right)

m7 = strassen\_mat\_mul(right\_top - right\_bottom, bottom\_left + bottom\_right)

result\_top\_left = m1 + m4 - m5 + m7

result\_top\_right = m3 + m5

result\_bottom\_left = m2 + m4

result\_bottom\_right = m1 - m2 + m3 + m6

result\_matrix = np.vstack((np.hstack((result\_top\_left, result\_top\_right)),

np.hstack((result\_bottom\_left, result\_bottom\_right))))

**return** result\_matrix

# Функция для умножения матриц с использованием алгоритма Штрассена

**def** **strassen\_multiply**(mat\_left, mat\_right):

original\_size = mat\_left.shape[**0**]

padded\_left = pad\_matrix(mat\_left)

padded\_right = pad\_matrix(mat\_right)

result\_padded = strassen\_mat\_mul(padded\_left, padded\_right)

**return** unpad\_matrix(result\_padded, original\_size)

# Функция для прямого умножения матриц

**def** **multiply\_matrices**(mat\_left, mat\_right):

rows\_left, common\_dim = mat\_left.shape

\_, cols\_right = mat\_right.shape

result\_matrix = np.zeros((rows\_left, cols\_right))

**for** row\_index **in** range(rows\_left):

**for** col\_index **in** range(cols\_right):

**for** common\_index **in** range(common\_dim):

result\_matrix[row\_index, col\_index] += mat\_left[row\_index, common\_index] \* mat\_right[common\_index, col\_index]

**return** result\_matrix

# Функция для генерации случайной квадратной матрицы

**def** **generate\_square\_matrix**(n, min\_val=**0**, max\_val=**10**):

matrix = np.random.randint(min\_val, max\_val+**1**, size=(n, n))

**return** matrix.astype(np.float32)

# Функция для проверки корректности результатов умножения

**def** **check\_results**(result\_strassen, result\_regular, tolerance=**1**e-**5**):

**return** np.allclose(result\_strassen, result\_regular, atol=tolerance)

# Функция для измерения времени выполнения функции dot

**def** **measure\_time\_dot**(n\_values):

times = []

**for** n **in** n\_values:

mat\_left = generate\_square\_matrix(n)

mat\_right = generate\_square\_matrix(n)

start\_time = time.time()

np.dot(mat\_left, mat\_right)

end\_time = time.time()

times.append(end\_time - start\_time)

**return** times

# Функция для нахождения обратной матрицы

**def** **find\_inverse\_matrix**(matrix):

**return** np.linalg.inv(matrix)

# Функция для нахождения собственных чисел и векторов

**def** **find\_eigenvalues\_and\_eigenvectors**(matrix):

eigenvalues, eigenvectors = np.linalg.eig(matrix)

**return** eigenvalues, eigenvectors

# Основной код

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

# Исходные данные

A = np.array([[-**54**, **16**, -**35**],

[**29**, -**66**, **40**],

[**45**, -**21**, **63**]], dtype=float)

b = np.array([**10**, -**5**, **20.00000001**], dtype=float)

# Умножение матриц по алгоритму Штрассена

random\_matrix = generate\_square\_matrix(**3**)

result\_strassen = strassen\_multiply(A, random\_matrix)

result\_regular = multiply\_matrices(A, random\_matrix)

**if** check\_results(result\_strassen, result\_regular):

print("Результаты умножения матриц - ОК")

**else**:

print("Результаты умножения матриц НЕ совпадают")

# Оценка временной сложности функции dot

n\_values = [**4**, **16**, **32**, **64**, **128**, **256**, **512**] # Используем степени двойки

times = measure\_time\_dot(n\_values)

plt.plot(n\_values, times, marker='o')

plt.xlabel('Размер матрицы (n x n)')

plt.ylabel('Время выполнения (секунды)')

plt.title('Временная сложность функции dot из NumPy')

plt.grid(**True**)

plt.show()

# Нахождение обратной матрицы

inverse\_A = find\_inverse\_matrix(A)

print("Обратная матрица A:**\n**", inverse\_A)

# Нахождение собственных чисел и векторов

eigenvalues, eigenvectors = find\_eigenvalues\_and\_eigenvectors(A)

print("Собственные числа:**\n**", eigenvalues)

print("Собственные векторы:**\n**", eigenvectors)

# Решение СЛАУ методом обратной матрицы

x = np.dot(inverse\_A, b)

print("Решение СЛАУ методом обратной матрицы:", x)

# Сравнение с LU-разложением

lu, piv = lu\_factor(A)

x\_lu = lu\_solve((lu, piv), b)

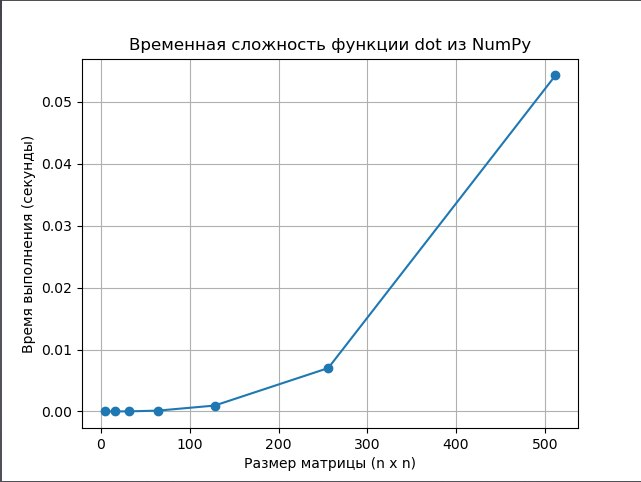
print("Решение системы с использованием LU-разложения:", x\_lu)

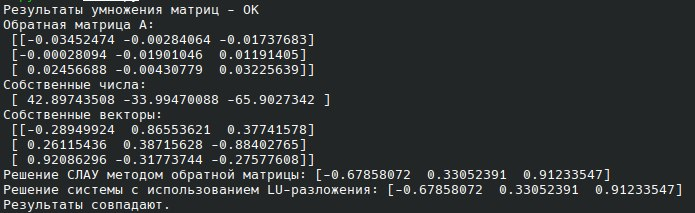
**if** np.allclose(x, x\_lu, atol=**1**e-**5**):

print("Результаты совпадают.")

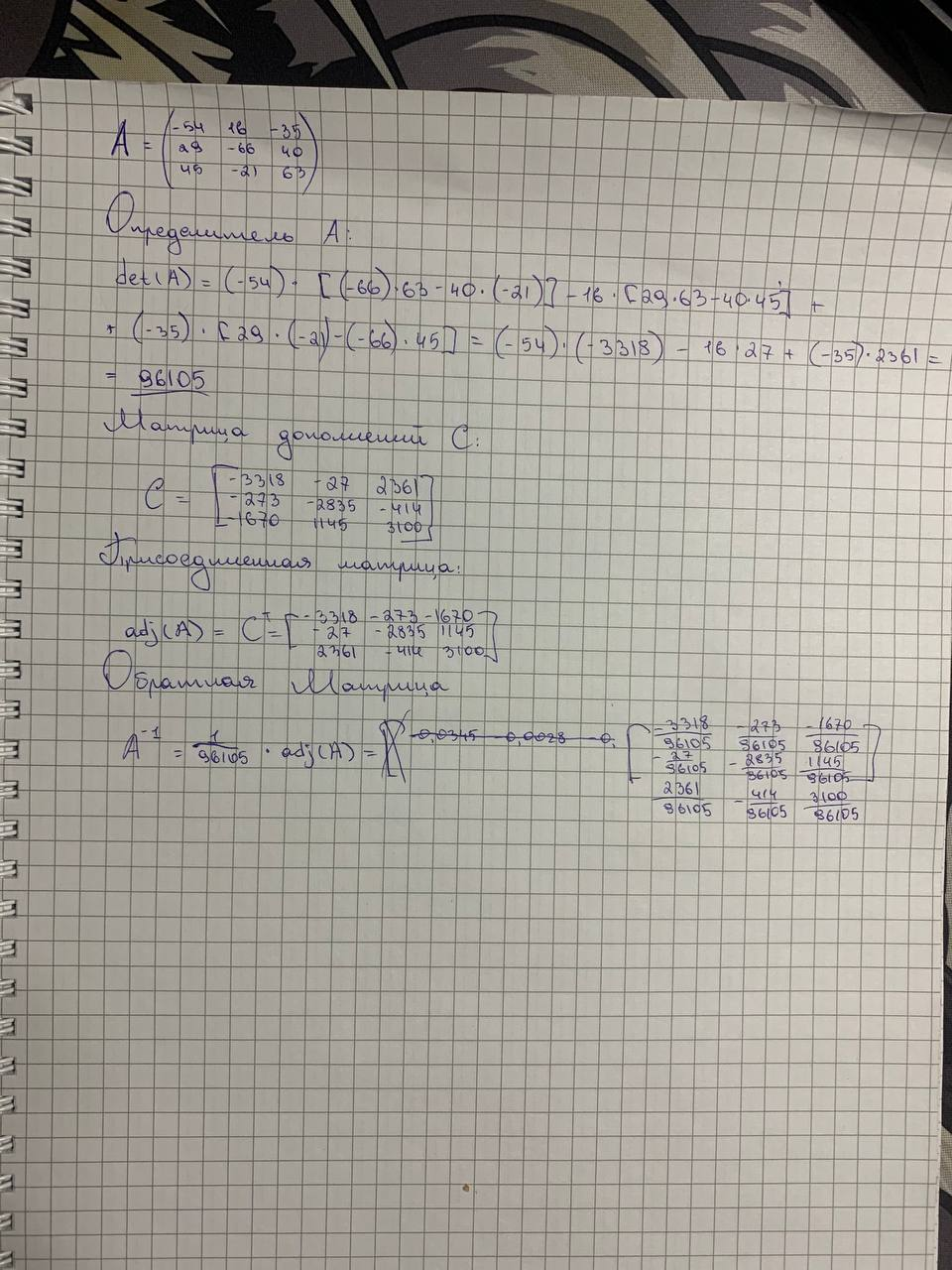
**else**:

print("Результаты НЕ совпадают.")

**Результат работы программы**

****

**Аналитическое решение**

  
  
  
**Вывод**: В ходе лабораторной работы были успешно изучены и применены основные операции над матрицами, включая умножение, нахождение обратной матрицы, собственных чисел и векторов. Результаты ручных вычислений совпали с результатами, полученными с использованием библиотеки NumPy, что подтверждает корректность работы алгоритмов и точность вычислений.