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)**

Кафедра программного обеспечения вычислительной техники и автоматизированных 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 ПВ-233

Ситников Алексей Павлович

Проверил:

Горбов Даниил Игоревич

Белгород 2025 г.

**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.

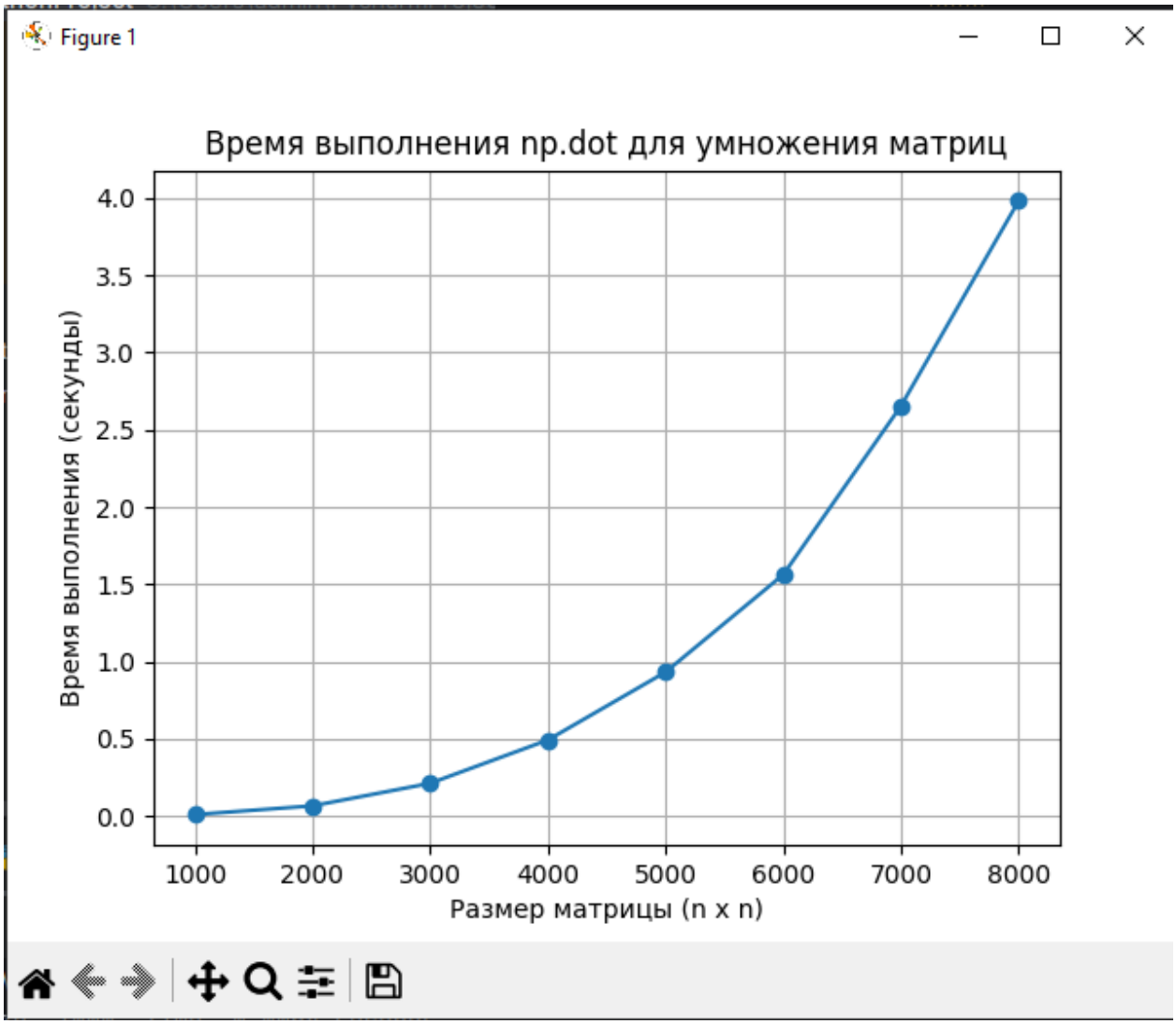
Эмпирически оценить временную сложность функции dot для умножения матриц из библиотеки NumPy (Python).

Код:

```
import numpy as np
import time
import matplotlib.pyplot as plt


def measure_dot_time(size, repetitions=5):
    A = np.random.rand(size, size)
    B = np.random.rand(size, size)

    total_time = 0
    for _ in range(repetitions):
        start_time = time.perf_counter()
        np.dot(A, B)
        end_time = time.perf_counter()
        total_time += (end_time - start_time)

    return total_time / repetitions


sizes = [1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000]
times = []

for size in sizes:
    elapsed_time = measure_dot_time(size)
    times.append(elapsed_time)
    print(f"Size: {size}, Time: {elapsed_time:.4f} seconds")

# Построение графика
plt.plot(sizes, times, marker='o')
plt.xlabel('Размер матрицы (n x n)')
plt.ylabel('Время выполнения (секунды)')
plt.title('Время выполнения np.dot для умножения матриц')
plt.grid()
plt.show()
```

Итог:

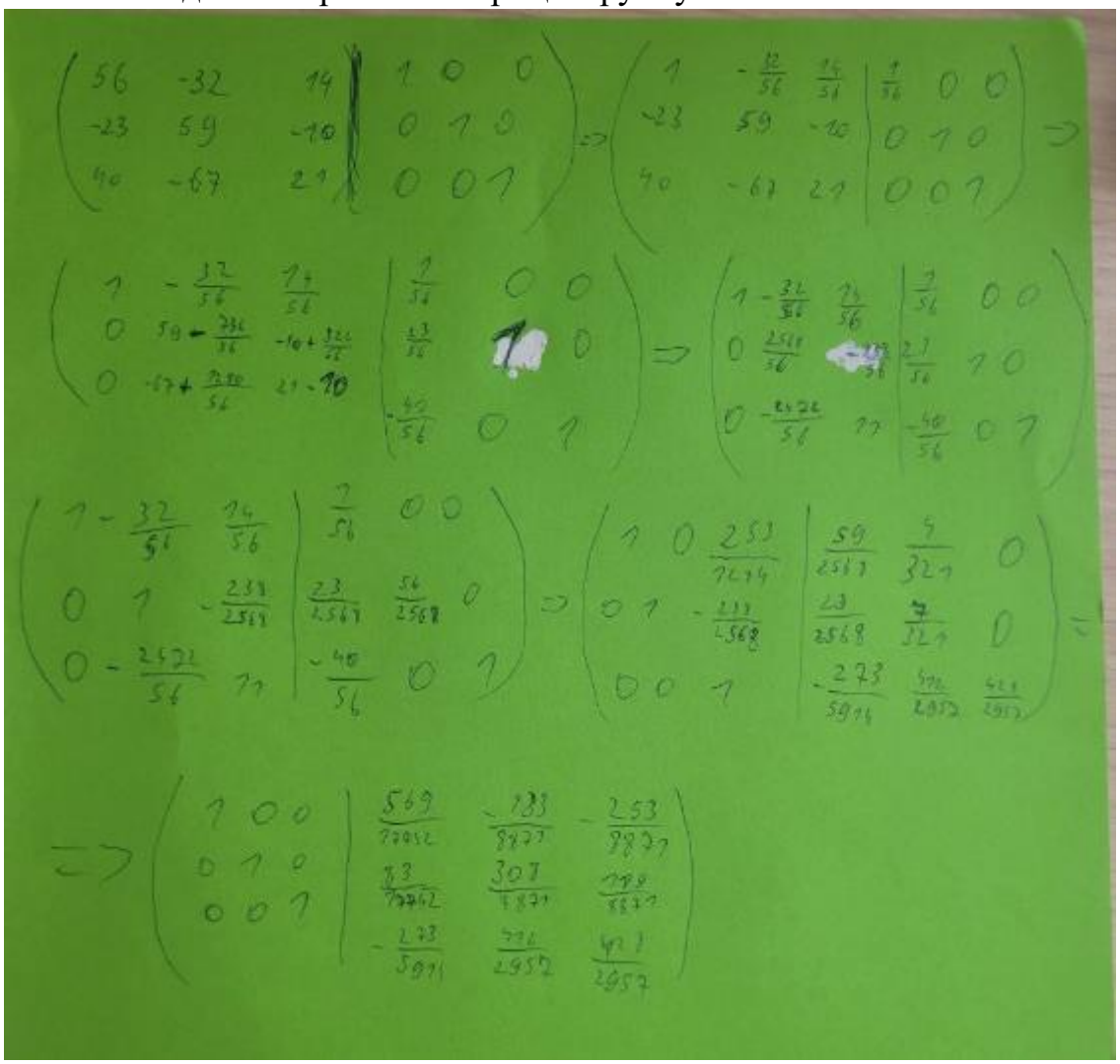
```
C:\Users\admin\PycharmProjects\pythonProject2\.venv\Scripts\python.exe C:\Users\admin\PycharmProjects\pythonProject\main.py
Size: 1000, Time: 0.0094 seconds
Size: 2000, Time: 0.0658 seconds
Size: 3000, Time: 0.2118 seconds
Size: 4000, Time: 0.4920 seconds
Size: 5000, Time: 0.9318 seconds
Size: 6000, Time: 1.5626 seconds
Size: 7000, Time: 2.6543 seconds
Size: 8000, Time: 3.9815 seconds
```




По графику видно, что сложность между $O(n^2)$ и $O(n^3)$.

Вариант 13

Нахождение обратной матрицы вручную:



Нахождение обратной матрицы методом *Ньютона-Шульца,* и с использованием np.linalg.inv.

Код:

```
import numpy as np

def compute_inverse_newton_schulz(matrix, num_iterations=5):

    assert matrix.shape[0] == matrix.shape[1], "Матрица не квадратная"

    initial_approximation = matrix.T / (np.linalg.norm(matrix) ** 2)
```

```
    # Единичная матрица того же размера, что и A
    identity_matrix = np.eye(matrix.shape[0])


    for _ in range(num_iterations):
        initial_approximation = initial_approximation @ (2 * identity_matrix
- matrix @ initial_approximation)

    return initial_approximation



matrix = np.array([[56, -32, 14],
                   [-23, 59, -10],
                   [40, -67, 21]], dtype=np.float64)

print("Исходная матрица:")
print(matrix)


inverse_matrix_ns = compute_inverse_newton_schulz(matrix, num_iterations=10)
print("\nОбратная матрица, полученная методом Ньютона-Шульца:")
print(inverse_matrix_ns)


inverse_matrix_np = np.linalg.inv(matrix)
print("\nОбратная матрица, полученная с помощью np.linalg.inv:")
print(inverse_matrix_np)
```

Вывод:

```
Исходная матрица:
[[ 56. -32.  14.]
 [-23.  59. -10.]
 [ 40. -67.  21.]]

Обратная матрица, полученная методом Ньютона-Шульца:
[[ 0.03013844 -0.00943891 -0.02279297]
 [ 0.00631018  0.0300293   0.00857767]
 [-0.03556629  0.10887828  0.11333969]]

Обратная матрица, полученная с помощью np.linalg.inv:
[[ 0.03207079 -0.01499267 -0.0285199 ]
 [ 0.00467816  0.03471987  0.0134145 ]
 [-0.04616165  0.1393304   0.14474129]]
```

Решение совпало с вычисленным вручную.

Нахождение собственных чисел и векторов:

```
import numpy as np

A = np.array([[56, -32, 14],
              [-23, 59, -10],
```

```
                  [40, -67, 21]], dtype=np.float64)


print("Исходная матрица:\n", A)
eigenvalues, eigenvectors = np.linalg.eig(A)
print("Собственные числа матрицы:\n", eigenvalues)
print("Собственные вектора матрицы:\n", eigenvectors)
```

Вывод:

```
Исходная матрица:
 [[ 56. -32.  14.]
 [-23.  59. -10.]
 [ 40. -67.  21.]]
Собственные числа матрицы:
 [100.36923408  29.67373678   5.95702915]
Собственные вектора матрицы:
 [[-0.55724078  0.7957939  -0.21424563]
 [ 0.47452133  0.51610946  0.09045821]
 [-0.68140459 -0.31676979  0.97258219]]
```

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