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: студент группы ПВ-233
Мороз Роман Алексеевич

Проверили:

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.

Цель работы обуславливает постановку и решение следующих задач:
1) Рассмотреть теоретические основы алгебры матриц и основные операции над матрицами.
2) Изучить особенности алгоритмизации быстрых матричных алгоритмов (на примере алгоритма умножения Штрассена). Эмпирически оценить временную сложность функции dot для умножения матриц из библиотеки NumPy (Python). На основании этой оценки сделать выводы в отношении используемых здесь вычислительных алгоритмов.
3) Познакомиться с алгоритмами вычисления обратной матрицы с использованием библиотеки NumPy.
4) Рассмотреть особенности программной реализации алгоритмов для нахождения собственных чисел и собственных векторов матрицы с использованием библиотеки NumPy.
5) Изучить возможности библиотеки matplotlib (Python) для подготовки графиков, сопоставляющих вычислительные затраты программ.
6) Выполнить индивидуальное задание, закрепляющее на практике полученные знания и практические навыки (номер задания соответствует номеру студента по журналу; если этот номер больше, чем максимальное число заданий, тогда вариант задания вычисляется по формуле: номер по журналу % максимальный номер задания, где % — остаток от деления).
Внимание, исходную матрицу для выполнения индивидуального задания следует взять из предыдущей лабораторной работы №1. Исходной матрицей выступит матрица размерами 3x3, составленная из коэффициентов перед переменными СЛАУ.

Первая часть данного задания предполагает нахождение обратной матрицы вручную по алгоритму, спецификация которого приводится в приложении к данной лабораторной работе. Для ручной проверки корректности полученного результата предлагается выполнить решение СЛАУ из лабораторной работы №1 методом обратной матрицы.

Вторая часть задания предполагает написание и выполнение в интерактивном блокноте Jupyter коротких программ на языке Python для:

а) нахождения обратной матрицы (использовать данные своего индивидуального задания);

б) нахождения собственных чисел и собственных векторов матрицы (использовать те же данные). Полученную в ходе численного решения обратную матрицу необходимо сравнить с результатами собственных вычислений вручную.

7) Отразить в отчете все полученные результаты. Сделать выводы. К отчету прикрепить расчёты, выполненные вручную.

Ход выполнения практической части лабораторной работы

1) Программы, демонстрирующей алгоритмическую реализацию умножения Штрассена для матриц на языке Python (Цель данного вычислительного эксперимента состоит в том, чтобы убедиться в корректности работы быстрого алгоритма, опираясь на результаты «прямого» перемножения матриц. Подсчитать количество операций умножения над вещественными числами для «прямого» перемножения двух матриц размерами 256x256 и для алгоритма Штрассена).

```python
def strassen_multiply(A, B, threshold=32):
    def split(mat):
        n = mat.shape[0]
        return mat[:n//2, :n//2], mat[:n//2, n//2:], mat[n//2:, :n//2], 
mat[n//2:, n//2:]

    n = A.shape[0]
    if n <= threshold:
        return A @ B, n**3

    A11, A12, A21, A22 = split(A)
    B11, B12, B21, B22 = split(B)

    # Промежуточные матрицы
    M1, c1 = strassen_multiply(A11 + A22, B11 + B22)
    M2, c2 = strassen_multiply(A21 + A22, B11)
    M3, c3 = strassen_multiply(A11, B12 - B22)
    M4, c4 = strassen_multiply(A22, B21 - B11)
    M5, c5 = strassen_multiply(A11 + A12, B22)
    M6, c6 = strassen_multiply(A21 - A11, B11 + B12)
    M7, c7 = strassen_multiply(A12 - A22, B21 + B22)

    # Формирование результата
    C11 = M1 + M4 - M5 + M7
    C12 = M3 + M5
    C21 = M2 + M4
    C22 = M1 - M2 + M3 + M6
```

```python
    # Сборка матрицы
    C = np.vstack((np.hstack((C11, C12)), np.hstack((C21, C22))))

    return C, c1 + c2 + c3 + c4 + c5 + c6 + c7

np.random.seed(42)
A = np.random.rand(256, 256)
B = np.random.rand(256, 256)

direct_ops = 256**3

result_strassen, strassen_ops = strassen_multiply(A, B)

print(f"Прямое умножение операций: {direct_ops:,}")
print(f"Штрассен операций: {strassen_ops:,}")
print("Проверка корректности:", np.allclose(A @ B, result_strassen))
```

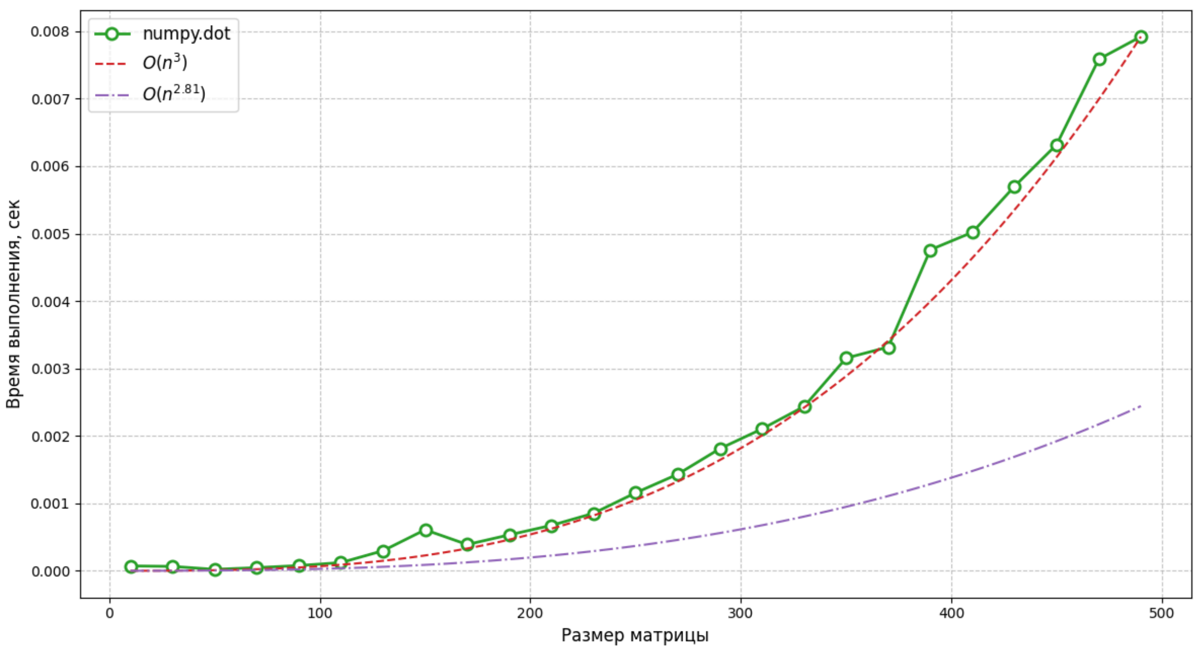
```
Прямое умножение операций: 16,777,216
Штрассен операций: 11,239,424
Проверка корректности: True
```

2) Программы эмпирической оценки временной сложности функции dot для умножения матриц из библиотеки NumPy (На основании построенного графика, сопоставляющего вычислительные затраты программ, сделать заключение о типах используемых алгоритмов в данной функции).

```python
sizes = np.arange(10, 501, 20)
times = []
repeats = 5

for n in sizes:
    A = np.random.rand(n, n)
    B = np.random.rand(n, n)

    start = time.perf_counter()
    for _ in range(repeats):
        np.dot(A, B)
    elapsed = (time.perf_counter() - start) / repeats
    times.append(elapsed)

plt.figure(figsize=(12, 7))
plt.plot(sizes, times, 'o-', color='#2ca02c', linewidth=2,
markersize=8,
```

```
        markerfacecolor='white', markeredgewidth=2, label='numpy.dot')

n_vals = np.linspace(min(sizes), max(sizes), 100)
scale = times[-1] / (sizes[-1]**3)
plt.plot(n_vals, scale*n_vals**3, '--', color='#d62728', linewidth=1.5,
label='$O(n^3)$')
plt.plot(n_vals, scale*n_vals**2.81, '-.', color='#9467bd',
linewidth=1.5, label='$O(n^{2.81})$')

plt.xlabel('Размер матрицы', fontsize=12)
plt.ylabel('Время выполнения, сек', fontsize=12)
plt.title('Временная сложность матричного умножения', pad=20,
fontsize=14)
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend(fontsize=12)
plt.tight_layout()
plt.show()
```

Временная сложность матричного умножения



3) Программы для нахождения обратной матрицы.

9. $$\begin{cases} 168x_1 - 150x_2 + 184x_3 = -170 \\ 155x_1 - 185x_2 + 171x_3 = 150 \\ -163x_1 + 190x_2 - 179x_3 = -180 \end{cases}$$

```
A = np.array([ [168, -150, 183],
               [155, -185, 171],
               [-163, 190, -179]
```

```
             ])

try:
    A_inv = np.linalg.inv(A)
    error = np.linalg.norm(A @ A_inv - np.eye(3))
    print(f"Ошибка обращения: {error:.2e}")
    print(A_inv)
except np.linalg.LinAlgError:
    print("Матрица вырождена")
```

```
Ошибка обращения: 8.80e-14
[[-0.1298027  -1.64485981 -1.70404984]
 [ 0.02658359  0.05046729  0.07538941]
 [ 0.14641745  1.55140187  1.62616822]]
```

4) Программы для нахождения собственных чисел и собственных векторов матрицы.

```
eigenvalues, eigenvectors = np.linalg.eig(A)

idx = np.argmax(np.abs(eigenvalues))
v = eigenvectors[:, idx]
lb = eigenvalues[idx]
error = np.linalg.norm(A @ v - lb * v)

print(f"Максимальное собственное значение: {lb:.4f}")
print(f"Собственный вектор: {v}")
print(f"Ошибка проверки: {error:.2e}")
```

$$\left(\begin{array}{ccc|ccc} 168 & -150 & 184 & 1 & 0 & 0 \\ 155 & -185 & 171 & 0 & 1 & 0 \\ -163 & 190 & -179 & 0 & 0 & 1 \end{array}\right) \quad \left(\begin{array}{ccc|ccc} 1 & -\frac{25}{28} & 1\frac{2}{21} & \frac{1}{168} & 0 & 0 \\ 155 & -185 & 171 & 0 & 1 & 0 \\ -163 & 190 & -179 & 0 & 0 & 1 \end{array}\right)$$

$$\left(\begin{array}{ccc|ccc} 1 & -\frac{25}{28} & 1\frac{2}{21} & \frac{1}{168} & 0 & 0 \\ 0 & -46\frac{17}{28} & 1\frac{5}{21} & -\frac{155}{168} & 1 & 0 \\ 0 & 44\frac{13}{28} & -\frac{10}{21} & \frac{163}{168} & 0 & 1 \end{array}\right) \quad \left(\begin{array}{ccc|ccc} 1 & -\frac{25}{28} & 1\frac{2}{21} & \frac{1}{168} & 0 & 0 \\ 0 & 1 & -\frac{184}{3915} & \frac{31}{1566} & -\frac{28}{1305} & 0 \\ 0 & 44\frac{13}{28} & -\frac{10}{21} & \frac{47}{522} & \frac{83}{87} & 1 \end{array}\right)$$

$$\left(\begin{array}{ccc|ccc} 1 & 0 & 0 & -\frac{125}{1104} & -1\frac{259}{552} & -1\frac{287}{552} \\ 0 & 1 & 0 & \frac{8}{345} & \frac{1}{69} & \frac{13}{345} \\ 0 & 0 & 1 & \frac{47}{368} & 1\frac{65}{184} & 1\frac{77}{184} \end{array}\right)$$

$$\det(A-\lambda E) = \begin{vmatrix} 168-\lambda & -150 & 184 \\ 155 & -185-\lambda & 171 \\ -163 & 190 & -179-\lambda \end{vmatrix} = \lambda^3 - 196\lambda^2 + 7285\lambda - 5520 \approx$$

$$\approx (\lambda - 228{,}051)\cdot(\lambda + 0{,}774)\cdot(\lambda + 31{,}277) = 0$$

$\lambda_1 \approx 228{,}051$

$\lambda_2 \approx -0{,}774$

$\lambda_3 \approx -31{,}277$

$X_1: (A - \lambda E)x = 0$

$$\left(\begin{array}{ccc|c} 59{,}949 & -150 & 184 & 0 \\ 155 & -413{,}051 & 171 & 0 \\ -163 & 190 & -210{,}051 & 0 \end{array}\right)$$

```
Максимальное собственное значение: -228.6639
Собственный вектор: [ 0.51093471  0.59939062 -0.61617905]
Ошибка проверки: 4.02e-14
```

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