Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2
дисциплины
«Искусственный интеллект и машинное обучение»
Вариант 12

Выполнил:
Ляш Денис Александрович
2 курс, группа ИТС-б-о-23-1,
11.03.02 «Инфокоммуникационные технологии и системы связи»,
направленность (профиль) «Инфокоммуникационные системы и сети», очная форма обучения

_____
(подпись)

Проверил:
Доцент департамента цифровых, роботехнических систем и электроники Воронкин Р. А.

_____
(подпись)

Отчет защищен с оценкой_____ Дата защиты_____

Ставрополь, 2025 г.

Тема: Основы работы с библиотекой NumPy

Цель: исследовать базовые возможности библиотеки NumPy языка программирования Python.

Ссылка на репозиторий: https://github.com/Zloybanan4ik/ii_lr2.git

Порядок выполнения работы:

1. В процессе выполнения лабораторной работы была изучены примеры для закрепления теоретических указаний к работе

2. Выполнение практической работы «Создание и изменение массивов»

1. Создание и изменение массивов

Создайте массив NumPy размером 3x3, содержащий числа от 1 до 9. Умножьте все элементы массива на 2, а затем замените все элементы больше 10 на 0. Выведите итоговый массив.

```
import numpy as np

# Создаем массив 3x3 с числами от 1 до 9
arr = np.arange(1, 10).reshape(3, 3)
print("Исходный массив:")
print(arr)

# Умножаем все элементы на 2
arr = arr * 2
print("\nМассив после умножения на 2:")
print(arr)

# Заменяем элементы > 10 на 0
arr[arr > 10] = 0
print("\nИтоговый массив:")
print(arr)
```

```
Исходный массив:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Массив после умножения на 2:
[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]

Итоговый массив:
[[ 2  4  6]
 [ 8 10  0]
 [ 0  0  0]]
```

Рисунок 1. Практическая работа «Создание и изменение массивов»

3. Выполнение практической работы «Работа с булевыми масками»

2. Работа с булевыми масками

Создайте массив NumPy из 20 случайных целых чисел от 1 до 100. Найдите и выведите все элементы, которые делятся на 5 без остатка. Затем замените их на -1 и выведите обновленный массив.

```
arr = np.random.randint(1, 101, 20)
print("Исходный массив:")
print(arr)

# Находим элементы, кратные 5
mask = arr % 5 == 0
multiples_of_5 = arr[mask]
print("\nЭлементы, кратные 5:")
print(multiples_of_5)

# Заменяем их на -1
arr[mask] = -1
print("\nОбновленный массив:")
print(arr)
```

```
Исходный массив:
[30 45 15 70  2 96 77 14 94 79 15 77 33 50 76 34 58 76 73 56]

Элементы, кратные 5:
[30 45 15 70 15 50]

Обновленный массив:
[-1 -1 -1 -1  2 96 77 14 94 79 -1 77 33 -1 76 34 58 76 73 56]
```

Рисунок 2. Практическая работа «Работа с булевыми масками»

4. Выполнение практической работы «Объединение и разбиение массивов»

3. Объединение и разбиение массивов

Создайте два массива NumPy размером 1×5, заполненные случайными числами от 0 до 50.

- Объедините эти массивы в один двумерный массив (по строкам).
- Разделите полученный массив на два массива, каждый из которых содержит 5 элементов.

Выведите все промежуточные и итоговые результаты.

```
arr1 = np.random.randint(0, 51, 5).reshape(1, 5)
arr2 = np.random.randint(0, 51, 5).reshape(1, 5)

print("Первый массив:")
print(arr1)
print("\nВторой массив:")
print(arr2)

# Объединяем массивы по строкам
combined = np.vstack((arr1, arr2))
print("\nОбъединенный массив:")
print(combined)

# Разделяем массив на два массива по 5 элементов
split_arr = np.split(combined, 2)
print("\nПервый разделенный массив:")
print(split_arr[0].flatten())
print("\nВторой разделенный массив:")
print(split_arr[1].flatten())
```

```
Первый массив:
[[ 3 18 40 18 11]]

Второй массив:
[[ 8 46 15 38 33]]

Объединенный массив:
[[ 3 18 40 18 11]
 [ 8 46 15 38 33]]

Первый разделенный массив:
[ 3 18 40 18 11]

Второй разделенный массив:
[ 8 46 15 38 33]
```

Рисунок 3. Практическая работа «Объединение и разбиение массивов»

5. Выполнение практической работы «Генерация и работа с линейными последовательностями»

4. Генерация и работа с линейными последовательностями

Создайте массив из 50 чисел, равномерно распределенных от -10 до 10. Вычислите сумму всех элементов, сумму положительных элементов и сумму отрицательных элементов. Выведите результаты.

```
import numpy as np

# Создаем массив из 50 чисел от -10 до 10
arr = np.linspace(-10, 10, 50)
print("Сгенерированный массив:")
print(arr)

# Вычисляем суммы
total_sum = np.sum(arr)
positive_sum = np.sum(arr[arr > 0])
negative_sum = np.sum(arr[arr < 0])

print("\nРезультаты:")
print(f"Сумма всех элементов: {total_sum:.2f}")
print(f"Сумма положительных элементов: {positive_sum:.2f}")
print(f"Сумма отрицательных элементов: {negative_sum:.2f}")
```

```
Сгенерированный массив:
[-10.         -9.59183673  -9.18367347  -8.7755102   -8.36734694
  -7.95918367  -7.55102041  -7.14285714  -6.73469388  -6.32653061
  -5.91836735  -5.51020408  -5.10204082  -4.69387755  -4.28571429
  -3.87755102  -3.46938776  -3.06122449  -2.65306122  -2.24489796
  -1.83673469  -1.42857143  -1.02040816  -0.6122449   -0.20408163
   0.20408163   0.6122449    1.02040816   1.42857143   1.83673469
   2.24489796   2.65306122   3.06122449   3.46938776   3.87755102
   4.28571429   4.69387755   5.10204082   5.51020408   5.91836735
   6.32653061   6.73469388   7.14285714   7.55102041   7.95918367
   8.36734694   8.7755102    9.18367347   9.59183673  10.        ]

Результаты:
Сумма всех элементов: 0.00
Сумма положительных элементов: 127.55
Сумма отрицательных элементов: -127.55
```

Рисунок 4. Практическая работа «Генерация и работа с линейными последовательностями»

6. Выполнение практической работы «Работа с диагональными и единичными матрицами»

5. Работа с диагональными и единичными матрицами

Создайте:

- Единичную матрицу размером 4x4.
- Диагональную матрицу размером 4x4 с диагональными элементами[5,10,15,20](не использовать циклы).

Найдите сумму всех элементов каждой из этих матриц и сравните результаты.

```
# Создаем единичную матрицу 4x4
identity_matrix = np.eye(4)
print("Единичная матрица:")
print(identity_matrix)

# Создаем диагональную матрицу с заданными элементами
diagonal_elements = [5, 10, 15, 20]
diagonal_matrix = np.diag(diagonal_elements)
print("\nДиагональная матрица:")
print(diagonal_matrix)

# Вычисляем суммы элементов
sum_identity = np.sum(identity_matrix)
sum_diagonal = np.sum(diagonal_matrix)

print("\nСравнение сумм:")
print(f"Сумма элементов единичной матрицы: {sum_identity}")
print(f"Сумма элементов диагональной матрицы: {sum_diagonal}")

if sum_identity > sum_diagonal:
    print("Единичная матрица имеет большую сумму элементов")
elif sum_identity < sum_diagonal:
    print("Диагональная матрица имеет большую сумму элементов")
else:
    print("Матрицы имеют одинаковую сумму элементов")
```

```
Единичная матрица:
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]

Диагональная матрица:
[[ 5  0  0  0]
 [ 0 10  0  0]
 [ 0  0 15  0]
 [ 0  0  0 20]]

Сравнение сумм:
Сумма элементов единичной матрицы: 4.0
Сумма элементов диагональной матрицы: 50
Диагональная матрица имеет большую сумму элементов
```

Рисунок 5. Практическая работа «Работа с диагональными и единичными матрицами»

7. Выполнение практической работы «Создание и базовые операции с матрицами»

6. Создание и базовые операции с матрицами

Создайте две квадратные матрицы NumPy размером 3x3, заполненные случайными целыми числами от 1 до 20. Вычислите и выведите:

- Их сумму
- Их разность
- Их поэлементное произведение

```
matrix1 = np.random.randint(1, 21, size=(3, 3))
matrix2 = np.random.randint(1, 21, size=(3, 3))

print("Первая матрица:")
print(matrix1)
print("\nВторая матрица:")
print(matrix2)

# Вычисляем операции
matrix_sum = matrix1 + matrix2
matrix_diff = matrix1 - matrix2
matrix_elementwise = matrix1 * matrix2

print("\nРезультаты операций:")
print("\nСумма матриц:")
print(matrix_sum)
print("\nРазность матриц:")
print(matrix_diff)
print("\nПоэлементное произведение:")
print(matrix_elementwise)
```

```
Первая матрица:
[[16  6 10]
 [ 1  7  3]
 [15  6 10]]

Вторая матрица:
[[16  5  5]
 [17  8 11]
 [14 14 12]]

Результаты операций:

Сумма матриц:
[[32 11 15]
 [18 15 14]
 [29 20 22]]

Разность матриц:
[[  0   1   5]
 [-16  -1  -8]
 [  1  -8  -2]]

Поэлементное произведение:
[[256  30  50]
 [ 17  56  33]
 [210  84 120]]
```

Рисунок 6. Практическая работа «Создание и базовые операции с матрицами»

8. Выполнение практической работы «Умножение матриц»

7. Умножение матриц

Создайте два матрицы NumPy:

- Первую размером 2x3, заполненную случайными числами от 1 до 10.
- Вторую размером 3x2, заполненую случайными числами от 1 до 10.

Выполните матричное умножение и выведите результат.

```
matrix_a = np.random.randint(1, 11, size=(2, 3))
print("Матрица A (2x3):")
print(matrix_a)

# Создаем матрицу 3x2
matrix_b = np.random.randint(1, 11, size=(3, 2))
print("\nМатрица B (3x2):")
print(matrix_b)

# Выполняем матричное умножение
result = np.dot(matrix_a, matrix_b)  # или np.dot(matrix_a, matrix_b)
print("\nРезультат матричного умножения:")
print(result)
```

```
Матрица A (2x3):
[[5 9 7]
 [8 1 8]]

Матрица B (3x2):
[[2 6]
 [9 8]
 [9 5]]

Результат матричного умножения:
[[154 137]
 [ 97  96]]
```

Рисунок 7. Практическая работа «Умножение матриц»

9. Выполнение практической работы «Определитель и обратная матрица»

8. Определитель и обратная матрица

Создайте случайную квадратную матрицу 3x3. Найдите и выведите:

- Определитель этой матрицы
- Обратную матрицу(если существует, иначе выведите сообщению, что матрица вырождена)

Используйте функции np.linalg.det и np.linalg.inv.

```
matrix = np.random.randint(1, 10, size=(3, 3))
print("Исходная матрица:")
print(matrix)

# Вычисляем определитель
det = np.linalg.det(matrix)
print(f"\nОпределитель матрицы: {det:.2f}")

# Проверяем обратимость матрицы и вычисляем обратную
if not np.isclose(det, 0):
    inv_matrix = np.linalg.inv(matrix)
    print("\nОбратная матрица:")
    print(inv_matrix)
else:
    print("\nМатрица вырождена (определитель равен 0), обратной матрицы не существует")
```

```
Исходная матрица:
[[8 7 7]
 [8 8 4]
 [5 3 2]]

Определитель матрицы: -52.00

Обратная матрица:
[[-0.07692308 -0.13461538  0.53846154]
 [-0.07692308  0.36538462 -0.46153846]
 [ 0.30769231 -0.21153846 -0.15384615]]
```

Рисунок 8. Практическая работа «Определитель и обратная матрица»

10. Выполнение практической работы «Транспонирование и след матрицы»

9. Транспонирование и след матрицы

Создайте матрицу NumPy размером 4x4, содержащую случайные целые числа от 1 до 50. Выведите:

- Исходную матрицу
- Траспонированную матрицу
- След матрицы(сумму всех элементов на главной диагонали)

Используйте np.trace для нахождение следа.

```
matrix = np.random.randint(1, 51, size=(4, 4))
print("Исходная матрица:")
print(matrix)

# Транспонируем матрицу
transposed = matrix.T
print("\nТранспонированная матрица:")
print(transposed)

# Вычисляем след матрицы
trace = np.trace(matrix)
print(f"\nСлед матрицы (сумма диагональных элементов): {trace}")
```

```
Исходная матрица:
[[11 19 37 49]
 [27 37 14  6]
 [17  3 35 49]
 [32 11 31 33]]

Транспонированная матрица:
[[11 27 17 32]
 [19 37  3 11]
 [37 14 35 31]
 [49  6 49 33]]

След матрицы (сумма диагональных элементов): 116
```

Рисунок 9. Практическая работа «Транспонирование и след матрицы»

11. Выполнение практической работы «Системы линейных уравнений»

10. Системы линейных уравнений

Решите систему линейных уравнений вида:

$$\begin{cases} 2x + 3y - z = 5 \\ 4x - y + 2z = 6 \\ -3x + 5y + 4z = -2 \end{cases}$$

Используйте матричное представление Ax = B, где A - это матрица коэффициентов, x - вектор неизвестных, B - вектор правой части. Решите систему с помощью np.linalg.solve и выведите результат.

```
# Матрица коэффициентов
A = np.array([
    [2, 3, -1],
    [4, -1, 2],
    [-3, 5, 4]
])

# Вектор правой части
B = np.array([5, 6, -2])

# Решение системы
try:
    solution = np.linalg.solve(A, B)
    print("Решение системы:")
    print(f"x = {solution[0]:.2f}")
    print(f"y = {solution[1]:.2f}")
    print(f"z = {solution[2]:.2f}")
except np.linalg.LinAlgError:
    print("Система не имеет единственного решения (матрица вырождена)")
```

```
Решение системы:
x = 1.64
y = 0.58
z = 0.01
```

Рисунок 10. Практическая работа «Системы линейных уравнений»

12. Выполнение индивидуального задания

11. Индивидуальное задание

Решите индивидуальное задание согласно варианта. Каждое задание предусматривает построение системы линейных уравнений. Решите полученную систему уравнений с использованием библиотеки NumPy. Для решения системы используйте метод Крамера и матричный метод. Сравните полученные результаты, с результатами, полученными с помощью np.linalg.solve.

12. Продажа билетов. На концерт продано 500 билетов трех категорий: стандартные, премиум и VIP. Билеты премиум стоят в 1.5 раза дороже стандартных, а VIP — в 2 раза дороже премиум. Общий доход от продажи составил 600 000 рублей. Сколько билетов каждого типа было продано, если известно, что стандартных билетов было на 100 больше, чем премиум?

Рисунок 11. Индивидуальное задание

```python
import numpy as np

# Дано:
total_tickets = 500
total_income = 600000
premium_to_standard_ratio = 1.5
vip_to_premium_ratio = 2
standard_more_than_premium = 100

# Система уравнений:
# 1) x + y + z = 500
# 2) x + 1.5y + 3z = 600 (предполагаем p=1000)
# 3) x = y + 100

# Матрица коэффициентов и вектор свободных членов
A = np.array([
    [1, 1, 1],
    [1, 1.5, 3],
    [1, -1, 0]
])
B = np.array([500, 600, 100])

# Решение с помощью np.linalg.solve
solution_linalg = np.linalg.solve(A, B)
x_linalg, y_linalg, z_linalg = solution_linalg

# Метод Крамера
def cramer_solve(A, B):
    det_A = np.linalg.det(A)
    solutions = []
    for i in range(len(B)):
        Ai = A.copy()
        Ai[:, i] = B
        det_Ai = np.linalg.det(Ai)
        solutions.append(det_Ai / det_A)
    return solutions

solution_cramer = cramer_solve(A, B)
x_cramer, y_cramer, z_cramer = solution_cramer

# Матричный метод
A_inv = np.linalg.inv(A)
solution_matrix = A_inv.dot(B)
x_matrix, y_matrix, z_matrix = solution_matrix

# Проверка доходов
p_standard = 1000  # предположение
p_premium = p_standard * premium_to_standard_ratio
p_vip = p_premium * vip_to_premium_ratio
```

```
income_linalg = (x_linalg * p_standard + y_linalg * p_premium + z_linalg * p_vip)
income_cramer = (x_cramer * p_standard + y_cramer * p_premium + z_cramer * p_vip)
income_matrix = (x_matrix * p_standard + y_matrix * p_premium + z_matrix * p_vip)

# Вывод результатов
print("Решение системы линейных уравнений:")
print(f"Метод np.linalg.solve: x = {x_linalg:.0f}, y = {y_linalg:.0f}, z = {z_linalg:.0f}")
print(f"Метод Крамера:         x = {x_cramer:.0f}, y = {y_cramer:.0f}, z = {z_cramer:.0f}")
print(f"Матричный метод:       x = {x_matrix:.0f}, y = {y_matrix:.0f}, z = {z_matrix:.0f}")

print("\nПроверка дохода:")
print(f"np.linalg.solve: {income_linalg:.0f} руб.")
print(f"Метод Крамера:    {income_cramer:.0f} руб.")
print(f"Матричный метод:  {income_matrix:.0f} руб.")

print("\nИтоговый ответ:")
print(f"Стандартных билетов: {int(round(x_linalg))}")
print(f"Премиум билетов:     {int(round(y_linalg))}")
print(f"VIP билетов:        {int(round(z_linalg))}")
```

Рисунок 12. Решение индивидуального задания

```
Решение системы линейных уравнений:
Метод np.linalg.solve: x = 300, y = 200, z = 0
Метод Крамера:         x = 300, y = 200, z = 0
Матричный метод:       x = 300, y = 200, z = 0

Проверка дохода:
np.linalg.solve: 600000 руб.
Метод Крамера:    600000 руб.
Матричный метод:  600000 руб.

Итоговый ответ:
Стандартных билетов: 300
Премиум билетов:     200
VIP билетов:        0
```

Рисунок 13. Вывод индивидуального задания

Ответы на контрольные вопросы:

1. **Каково назначение библиотеки NumPy?**

NumPy - это библиотека Python для работы с многомерными массивами и выполнения быстрых математических операций над ними. Она обеспечивает высокую производительность для научных вычислений, анализа данных и машинного обучения.

2. **Что такое массивы ndarray?**

ndarray - это основной тип данных в NumPy: многомерный, однородный массив. Все элементы имеют один тип. Характеризуется размерностью (shape) и типом данных (dtype). Эффективнее списков Python по памяти и скорости, поддерживает векторизованные операции.

3. **Как осуществляется доступ к частям многомерного массива?**

Доступ к частям ndarray осуществляется индексацией и срезами.

- arr[index] - доступ к элементу (одномерный).
- arr[row, col] - доступ к элементу (многомерный).
- arr[start:stop:step] - срез (slicing) по каждой оси.

Другие способы: индексация массивом индексов и булева индексация. Срезы возвращают view (представление), а не копию (кроме случаев с массивами индексов/булевыми масками).

4. **Как осуществляется расчет статистик по данным?**

NumPy предоставляет функции для расчета статистик: np.mean(), np.median(), np.std(), np.sum(), np.min(), np.max(), np.percentile(). Аргумент axis указывает ось вычисления (None - все элементы, 0 - столбцы, 1 - строки). Есть функции для работы с NaN (например, np.nanmean()). Также доступны np.argmin(), np.argmax(), np.corrcoef(), np.histogram().

5. **Как выполняется выборка данных из массивов ndarray?**

Выборка данных из массивов ndarray выполняется с использованием индексации и срезов.

Индексация позволяет получить доступ к отдельным элементам массива, указывая их положение (индексы) вдоль каждой оси.

Срезы позволяют выбрать подмножество элементов, определяя начальный и конечный индексы (и опционально шаг) для каждой оси.

Комбинируя индексацию и срезы, можно извлекать сложные фрагменты данных из массива. Кроме того, возможно использование булевых массивов (масок) для выбора элементов, удовлетворяющих определенным условиям.

6. Приведите основные виды матриц и векторов. Опишите способы их создания в языке Python.

Основные виды матриц: квадратная (количество строк равно количеству столбцов), диагональная (ненулевые элементы только на главной диагонали), единичная (диагональная матрица с единицами на главной диагонали), нулевая (все элементы равны нулю), симметричная (элементы симметричны относительно главной диагонали), треугольная (все элементы выше или ниже главной диагонали равны нулю).

Основные виды векторов: вектор-строка (матрица 1 x n), вектор-столбец (матрица n x 1), нулевой вектор (все элементы равны нулю), единичный вектор (вектор, длина которого равна 1).

Можно создать матрицу с помощью numpy.array() передав в качестве аргумента список списков (каждый подсписок - строка матрицы). Можно использовать функции numpy.zeros(), numpy.ones(), numpy.eye() (для единичной матрицы), numpy.diag() (для диагональной матрицы) и numpy.full() (для заполнения матрицы одинаковым значением).

Вектор, по сути, это одномерный массив NumPy. Его можно создать также с помощью numpy.array() передав в качестве аргумента список значений. Для создания последовательности значений можно использовать numpy.arange() или numpy.linspace(). Вектор-строку и вектор-столбец можно создать, соответственно, изменив форму (reshape) одномерного массива или создав двумерный массив с одной строкой или одним столбцом.

7. Как выполняется транспонирование матриц?

В Python с использованием NumPy транспонирование можно выполнить с помощью метода .T или функции numpy.transpose().

8. Приведите свойства операции транспонирования матриц.

– Транспонирование транспонированной матрицы возвращает исходную матрицу

– Транспонирование суммы матриц равно сумме транспонированных матриц

– Транспонирование произведения матриц равно произведению транспонированных матриц в обратном порядке

– Транспонирование произведения матрицы на скаляр равно произведению скаляра на транспонированную матрицу

9. Какие имеются средства в библиотеке NumPy для выполнения транспонирования матриц?

Метод .T: Это самый простой и часто используемый способ. Он вызывается у объекта ndarray (матрицы) и возвращает транспонированную версию.

Функция numpy.transpose(): Эта функция принимает массив ndarray в качестве аргумента и возвращает его транспонированную версию

10. Какие существуют основные действия над матрицами?

– Сложение: Можно складывать матрицы одинаковой размерности. Сумма получается путем сложения соответствующих элементов.

– Вычитание: Аналогично сложению, можно вычитать матрицы одинаковой размерности. Разность получается путем вычитания соответствующих элементов.

– Умножение на скаляр: Каждый элемент матрицы умножается на заданное число (скаляр).

– Умножение матриц: Умножение матрицы A размерности m x n на матрицу B размерности n x p возможно. Результатом будет

матрица C размерности m x p. Элемент C_{ij} вычисляется как скалярное произведение i-й строки матрицы A на j-й столбец матрицы B.

– Транспонирование: Обмен местами строк и столбцов матрицы.

– Вычисление определителя: Определитель можно вычислить только для квадратных матриц. Это скалярная величина, которая характеризует свойства матрицы.

– Нахождение обратной матрицы: Обратная матрица существует только для квадратных невырожденных матриц (определитель не равен нулю). Произведение матрицы на ее обратную равно единичной матрице.

– Вычисление ранга матрицы: Ранг матрицы - это максимальное число линейно независимых строк (или столбцов) матрицы.

– Поиск собственных значений и собственных векторов: Эти величины связаны с линейными преобразованиями, описываемыми матрицами.

11. Как осуществляется умножение матрицы на число?

Умножение матрицы на число (скаляр) осуществляется путем умножения каждого элемента матрицы на это число. То есть, если у нас есть матрица A и число k, то результатом умножения k * A будет новая матрица, в которой каждый элемент a_{ij} заменен на k * a_{ij}

12. Какие свойства операции умножения матрицы на число?

– Коммутативность: k × A = A × k, где k - скаляр, A - матрица. Порядок умножения не важен.

– Ассоциативность: (k × l) × A = k × (l × A), где k и l - скаляры, A - матрица.

– Дистрибутивность относительно сложения матриц: k × (A + B) = k × A + k × B, где k - скаляр, A и B - матрицы одинаковой размерности.

– Дистрибутивность относительно сложения скаляров: (k + l) × A = k × A + l × A, где k и l - скаляры, A - матрица.

– Умножение на единицу: 1 × A = A. Умножение на единицу не меняет матрицу.

– Умножение на ноль: 0 × A = O, где O - нулевая матрица той же размерности, что и A. Умножение на ноль дает нулевую матрицу.

13. Как осуществляется операции сложения и вычитания матриц?

Сложение: Если у нас есть матрицы A и B одинаковой размерности m x n, то их сумма C = A + B будет матрицей той же размерности m x n, где каждый элемент c_{ij} равен сумме соответствующих элементов a_{ij} и b_{ij} >: $c_{ij} = a_{ij} + b_{ij}$.

Вычитание: Аналогично, если у нас есть матрицы A и B одинаковой размерности m x n, то их разность D = A - B будет матрицей той же размерности m x n, где каждый элемент d_{ij} равен разности соответствующих элементов a_{ij} и b_{ij}: $d_{ij} = a_{ij} – b_{ij}$.

14. Каковы свойства операций сложения и вычитания матриц?

Сложение:

Коммутативность: A + B = B + A (порядок слагаемых не важен).

Ассоциативность: (A + B) + C = A + (B + C).

Существование нейтрального элемента: A + O = A, где O - нулевая матрица той же размерности, что и A. Нулевая матрица является нейтральным элементом относительно сложения.

Существование противоположного элемента: Для каждой матрицы A существует матрица -A (полученная умножением A на -1), такая, что A + (-A) = O.

Вычитание:

Вычитание, по сути, является сложением с противоположным элементом: A - B = A + (-B). Поэтому большинство свойств вычитания выводятся из свойств сложения.

Вычитание не коммутативно: A - B ≠ B - A (в общем случае).

Вычитание не ассоциативно: (A - B) - C ≠ A - (B - C) (в общем случае).

15. Какие имеются средства в библиотеке NumPy для выполнения операций сложения и вычитания матриц?

В библиотеке NumPy для выполнения операций сложения и вычитания матриц используются стандартные операторы + и -, соответственно.

16. Как осуществляется операция умножения матриц?

Элемент c_{ij} матрицы C вычисляется как скалярное произведение i-й строки матрицы A на j-й столбец матрицы B:

То есть, для каждого элемента c_{ij} мы берем i-ю строку матрицы A и j-й столбец матрицы B, умножаем соответствующие элементы этих строки и столбца, и суммируем полученные произведения.

17. Каковы свойства операции умножения матриц?

Ассоциативность: (A * B) * C = A * (B * C). Если размеры матриц позволяют выполнять умножение в указанном порядке.

Дистрибутивность относительно сложения: A * (B + C) = A * B + A * C и (A + B) * C = A * C + B * C. Опять же, при условии, что размеры матриц согласованы.

Умножение на единичную матрицу: A * I = A и I * A = A, где I - единичная матрица подходящей размерности. Единичная матрица является нейтральным элементом относительно умножения матриц.

Умножение на нулевую матрицу: A * O = O и O * A = O, где O - нулевая матрица подходящей размерности. Результатом будет нулевая матрица.

Некоммутативность (в общем случае): A * B ≠ B * A. В отличие от умножения чисел, порядок умножения матриц важен.(kA) * B = k * (A * B) = A * (kB), где k - скаляр.

Транспонирование произведения матриц равно произведению транспонированных матриц в обратном порядке.

18. Какие имеются средства в библиотеке NumPy для выполнения операции умножения матриц?

Оператор @ (матричное умножение): Это предпочтительный и самый современный способ. Например: C = A @ B

Функция numpy.matmul(): Эта функция выполняет матричное умножение. Например: C = numpy.matmul(A, B).

Функция numpy.dot(): Эта функция может выполнять как скалярное произведение векторов, так и матричное умножение. В случае матриц она ведет себя как numpy.matmul(). Например: C = numpy.dot(A, B).

19. Что такое определитель матрицы? Каковы свойства определителя матрицы?

Определитель матрицы - это скалярная величина, которая может быть вычислена только для квадратных матриц. Он предоставляет информацию о свойствах матрицы, таких как её обратимость и линейная независимость её строк (или столбцов).

Свойства определителя матрицы:

1. Определитель единичной матрицы равен 1: det(I) = 1.
2. Определитель транспонированной матрицы равен определителю исходной матрицы: $\det(A^T) = \det(A)$.
3. Если матрица имеет нулевую строку (или столбец), то её определитель равен нулю.
4. Если матрица имеет две одинаковые строки (или столбца), то её определитель равен нулю.
5. Если поменять местами две строки (или столбца) матрицы, то определитель изменит знак на противоположный.
6. Если умножить строку (или столбец) матрицы на скаляр k, то определитель умножится на k. $\det(kA) = k_n\det(A)$, где n - размерность матрицы A (n x n).
7. Определитель произведения матриц равен произведению определителей: det(A * B) = det(A) * det(B).
8. Матрица обратима (имеет обратную матрицу) тогда и только тогда, когда её определитель не равен нулю. Если det(A) != 0, то существует A_1.

9. Определитель диагональной и треугольной матриц равен произведению элементов на главной диагонали.

10. Если к строке (или столбцу) матрицы прибавить другую строку (или столбец), умноженную на скаляр, то определитель не изменится.

20. Какие имеются средства в библиотеке NumPy для нахождения значения определителя матрицы?

В библиотеке NumPy для нахождения определителя матрицы используется функция numpy.linalg.det(). Она принимает массив ndarray (квадратную матрицу) в качестве аргумента и возвращает скалярное значение, представляющее собой определитель этой матрицы.

21. Что такое обратная матрица? Какой алгоритм нахождения обратной матрицы?

Обратная матрица - это такая матрица (обозначается A^{-1}, которая при умножении на исходную матрицу A дает единичную матрицу I.

Обратная матрица существует только для квадратных *невырожденных* матриц (то есть, для матриц, у которых определитель не равен нулю).

Алгоритм нахождения обратной матрицы (метод Гаусса-Жордана):

1. Создать расширенную матрицу: Справа от исходной матрицы A приписать единичную матрицу I той же размерности. Получится расширенная матрица [A | I].

2. Выполнить элементарные преобразования строк над расширенной матрицей: Цель - привести матрицу A к единичной матрице. При этом все те же преобразования применяются и к матрице I.

3. Получить обратную матрицу: После того как матрица A преобразована в единичную матрицу, на месте исходной единичной матрицы I окажется обратная матрица A^{-1}. Теперь расширенная матрица будет иметь вид [I | A^{-1}].

22. Каковы свойства обратной матрицы?

Свойства обратной матрицы:

1. $A * A^{-1} = A^{-1} * A = I$, где A - исходная матрица, A^{-1} - ее обратная, а I - единичная матрица. Это основное определение обратной матрицы.

2. $(A^{-1})^{-1} = A$. Обратная к обратной матрице - это исходная матрица.

3. $(A * B^{-1} = B^{-1} * A^{-1}$. Обратная произведения матриц равна произведению обратных матриц в обратном порядке. Это свойство применимо, если и A, и B - квадратные невырожденные матрицы.

4. $(A^T)^{-1} = (A^{-1})^T$. Обратная транспонированной матрицы равна транспонированной обратной матрице.

5. $\det(A^{-1}) = 1 / \det(A)$. Определитель обратной матрицы равен обратной величине определителя исходной матрицы.

6. Если A - диагональная матрица, то A^{-1} также является диагональной матрицей, и ее диагональные элементы являются обратными диагональным элементам матрицы A (при условии, что все диагональные элементы A ненулевые).

7. Если A - ортогональная матрица ($A^T * A = I$), то $A^{-1} = A^T$. То есть, обратная матрица ортогональной матрицы равна ее транспонированной матрице.

8. Умножение системы уравнений на обратную матрицу позволяет решить эту систему. Если $Ax = b$, то $x = A^{-1}b$.

23. Какие имеются средства в библиотеке NumPy для нахождения обратной матрицы?

В библиотеке NumPy для нахождения обратной матрицы используется функция numpy.linalg.inv(). Эта функция принимает в качестве аргумента квадратную матрицу (ndarray) и возвращает ее обратную матрицу. Если матрица необратима (то есть, ее определитель равен нулю), то будет сгенерировано исключение numpy.linalg.LinAlgError

24. Самостоятельно изучите метод Крамера для решения систем линейных уравнений. Приведите алгоритм решения системы линейных уравнений методом Крамера средствами библиотеки NumPy.

```
import numpy as np
def cramer_rule(A, b):
    det_A = np.linalg.det(A)
    if np.isclose(det_A, 0): return None
    x = [np.linalg.det(np.insert(A, i, b, axis=1)[:, 1:]) / det_A for i in range(A.shape[0])]
    return np.array(x)

# Пример
A = np.array([[2, 1, -1], [1, -1, 1], [1, 1, 1]])
b = np.array([1, 2, 6])
x = cramer_rule(A, b)
print(x)
```

25. Самостоятельно изучите матричный метод для решения систем линейных уравнений. Приведите алгоритм решения системы линейных уравнений матричным методом средствами библиотеки NumPy.

```
import numpy as np
def matrix_solve(A, b):
  try:
    A_inv = np.linalg.inv(A)
    x = A_inv @ b
    return x
  except np.linalg.LinAlgError:
    return None

# Пример:
A = np.array([[2, 1], [1, 1]])
b = np.array([1, 2])
x = matrix_solve(A, b)
print(x) # Вывод: [ -1.  3.]
```

Вывод: в ходе лабораторной работы были изучены базовые функции библиотеки NumPy для создания массивов, выполнения поэлементных и матричных операций, индексации и вычисления определителей/обратных матриц, а также решение задач с использованием, изученных технологий.