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»

Выполнил:
Жоелилала Максим Милахарисонович
2 курс, группа ИТС-б-о-23-1,
11.03.02«Инфокоммуникационные технологии и системы связи», очная форма обучения

_____
(подпись)

Проверил:
Доцент департамента цифровых, робототехнических 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

Порядок выполнения работы:

1. Ознакомились с теоретическим материалом

2. Создание репозитория

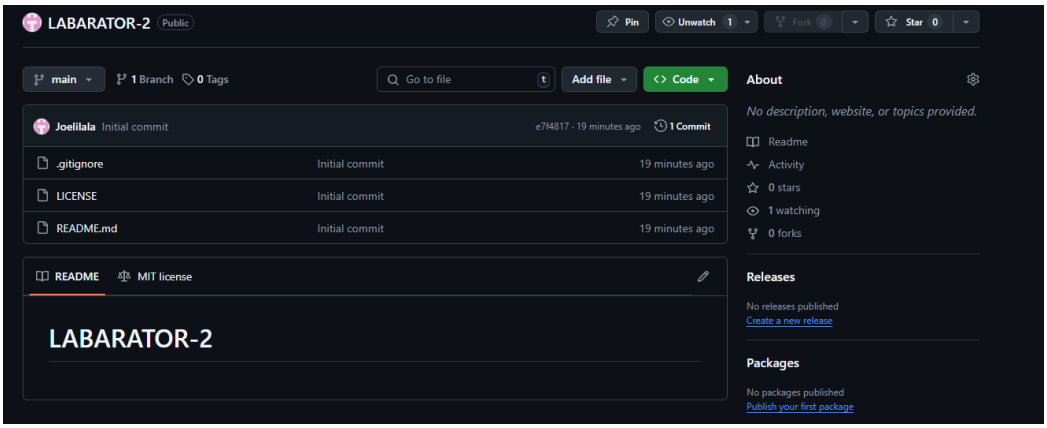

Рисунок 1. Репозиторий

3. Выполнено клонирование репозитория

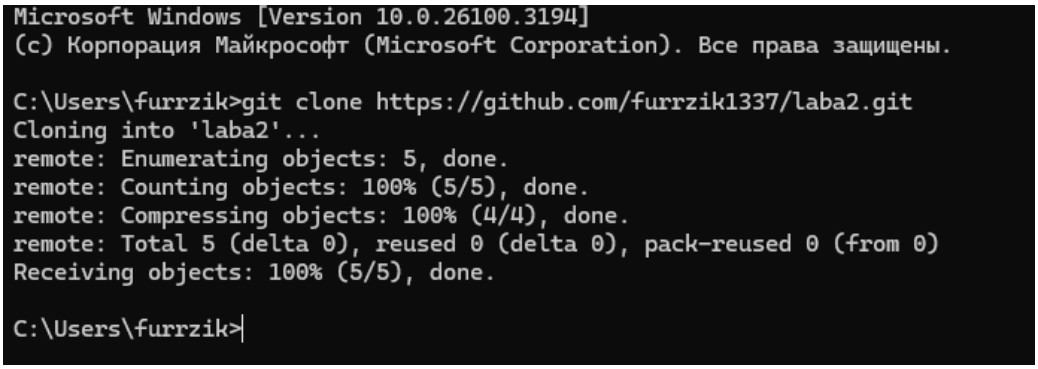

Рисунок 2. Клонирование

4. Выполнены практические задания

```
[1]: import numpy as np

arr = np.arange(1, 10).reshape(3, 3)
arr = arr * 2
arr[arr > 10] = 0

print(arr)
```

```
[[ 2  4  6]
 [ 8 10  0]
 [ 0  0  0]]
```

Рисунок 3. Практическое задание 1

```
[1]: import numpy as np

arr = np.random.randint(1, 101, 20)

print("Исходный массив:")
print(arr)

elements_divisible_by_5 = arr[arr % 5 == 0]
print("\nЭлементы, делящиеся на 5:")
print(elements_divisible_by_5)

arr[arr % 5 == 0] = -1

print("\nОбновленный массив (элементы, делящиеся на 5, заменены на -1):")
print(arr)
```

```
Исходный массив:
[ 7 25 39 32 26 57 93 92 44 86 65 20 66 86 66 25 57  2  8 60]

Элементы, делящиеся на 5:
[25 65 20 25 60]

Обновленный массив (элементы, делящиеся на 5, заменены на -1):
[ 7 -1 39 32 26 57 93 92 44 86 -1 -1 66 86 66 -1 57  2  8 -1]
```

Рисунок 4. Практическое задание 2

[1]:
```
import numpy as np

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

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

combined_arr = np.concatenate((arr1, arr2), axis=0)

print("\nОбъединенный массив:")
print(combined_arr)

arr_split1, arr_split2 = np.split(combined_arr, 2, axis=0)
print("\nПервый разделенный массив:")
print(arr_split1)
print("\nВторой разделенный массив:")
print(arr_split2)
```

```
Первый массив:
[[ 8 39 31  4  9]]

Второй массив:
[[ 8 48  6 15  6]]

Объединенный массив:
[[ 8 39 31  4  9]
 [ 8 48  6 15  6]]

Первый разделенный массив:
[[ 8 39 31  4  9]]

Второй разделенный массив:
[[ 8 48  6 15  6]]
```

Рисунок 5. Практическое задание 3

[5]:
```
import numpy as np

# Создаем массив из 50 чисел, равномерно распределённых от -10 до 10
array = np.random.uniform(-10, 10, size=50)
print("Сгенерированный массив:")
print(array)

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

# Выводим результаты
print("\nСумма всех элементов:", total_sum)
print("Сумма положительных элементов:", positive_sum)
print("Сумма отрицательных элементов:", negative_sum)
```

```
Сгенерированный массив:
[-3.7196128   5.17478839 -4.50708116  9.59744689 -0.23540313 -8.3044673
 -8.34306972  1.98150252 -6.30717104 -6.93901182  3.15408784  8.87205215
 -9.20803632 -2.6914661  -0.94685875 -0.63898046 -4.75859433  7.32907049
 -6.6189678   4.60594295 -9.54579863 -0.65291571  9.02101802  0.0150602
  0.44438072  1.00911838 -3.99178478 -7.582662   -5.6933584   5.42074516
 -4.82093895  4.45259546  4.39603654  2.98756443  1.93043794  3.56916917
 -1.109959   -8.08778917  5.55118202 -3.02308553 -4.36113712 -9.54319579
  9.18086036 -4.91890322  4.15584698  6.8852648   7.6710484  -0.90695362
 -6.21179087  1.96635626]

Сумма всех элементов: -24.297417460886443
Сумма положительных элементов: 109.37157606069268
Сумма отрицательных элементов: -133.66899352157913
```

[ ]:

Рисунок 6. Практическое задание 4

[5]:
```python
import numpy as np

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

# Создаем диагональную матрицу размером 4x4 с диагональными элементами [5, 10, 15, 20]
diagonal_elements = np.array([5, 10, 15, 20])
diagonal_matrix = np.diag(diagonal_elements)
print("\nДиагональная матрица 4x4:")
print(diagonal_matrix)

# Вычисляем суммы всех элементов в каждой матрице
identity_sum = np.sum(identity_matrix)
diagonal_sum = np.sum(diagonal_matrix)

# Выводим результаты
print("\nСумма всех элементов единичной матрицы:", identity_sum)
print("Сумма всех элементов диагональной матрицы:", diagonal_sum)

# Сравниваем результаты
if identity_sum > diagonal_sum:
    print("Сумма единичной матрицы больше суммы диагональной матрицы.")
elif identity_sum < diagonal_sum:
    print("Сумма диагональной матрицы больше суммы единичной матрицы.")
else:
    print("Суммы обеих матриц равны.")
```

```
Единичная матрица 4x4:
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]

Диагональная матрица 4x4:
[[ 5  0  0  0]
 [ 0 10  0  0]
 [ 0  0 15  0]
 [ 0  0  0 20]]

Сумма всех элементов единичной матрицы: 4.0
Сумма всех элементов диагональной матрицы: 50
Сумма диагональной матрицы больше суммы единичной матрицы.
```

Рисунок 7. Практическое задание 5

[4]:
```python
import numpy as np

# Создаем две матрицы 3x3 с случайными целыми числами от 1 до 20
matrix_a = np.random.randint(1, 21, size=(3, 3))
matrix_b = np.random.randint(1, 21, size=(3, 3))

print("Первая матрица (A):")
print(matrix_a)

print("\nВторая матрица (B):")
print(matrix_b)

# Вычисляем сумму матриц
matrix_sum = matrix_a + matrix_b

# Вычисляем разность матриц
matrix_difference = matrix_a - matrix_b

# Вычисляем поэлементное произведение матриц
elementwise_product = matrix_a * matrix_b

# Выводим результаты
print("\nСумма матриц (A + B):")
print(matrix_sum)

print("\nРазность матриц (A - B):")
print(matrix_difference)

print("\nПоэлементное произведение матриц (A * B):")
print(elementwise_product)
```

```
Первая матрица (A):
[[16  9  8]
 [ 5 17 10]
 [15  7  1]]

Вторая матрица (B):
[[20  9 16]
 [ 5  1  4]
 [ 4 11  8]]

Сумма матриц (A + B):
[[36 18 24]
 [10 18 14]
 [19 18  9]]

Разность матриц (A - B):
[[-4  0 -8]
 [ 0 16  6]
 [11 -4 -7]]

Поэлементное произведение матриц (A * B):
[[320  81 128]
 [ 25  17  40]
 [ 60  77   8]]
```

Рисунок 8. Практическое задание 6

```
[1]: import numpy as np

matrix1 = np.random.randint(1, 11, size=(2, 3))

matrix2 = np.random.randint(1, 11, size=(3, 2))
print("Первая матрица (2x3):\n", matrix1)
print("\nВторая матрица (3x2):\n", matrix2)

result_matrix = matrix1 @ matrix2

print("\nРезультат матричного умножения (2x2):\n", result_matrix)
```

```
Первая матрица (2x3):
 [[10  3  1]
 [ 1  2  5]]

Вторая матрица (3x2):
 [[4 9]
 [3 9]
 [4 2]]

Результат матричного умножения (2x2):
 [[ 53 119]
 [ 30  37]]
```

Рисунок 9. Практическое задание 7

```
1]: import numpy as np

matrix = np.random.rand(3, 3)

print("Исходная матрица:\n", matrix)

determinant = np.linalg.det(matrix)
print("\nОпределитель матрицы:", determinant)

if abs(determinant) > 1e-8:
    try:
        inverse_matrix = np.linalg.inv(matrix)
        print("\nОбратная матрица:\n", inverse_matrix)
    except np.linalg.LinAlgError:
        print("\nМатрица вырождена и не имеет обратной матрицы.")

else:
    print("\nМатрица вырождена и не имеет обратной матрицы.")
```

```
Исходная матрица:
 [[0.05554635 0.66490963 0.68912159]
 [0.69785546 0.50384752 0.747731  ]
 [0.00226231 0.38764827 0.68220259]]

Определитель матрицы: -0.12679499780252335

Обратная матрица:
 [[-0.42485469  1.47061224 -1.18270707]
 [ 3.74137159 -0.28656382 -3.46522767]
 [-2.12455231  0.15795747  3.43881013]]
```

Рисунок 10. Практическое задание 8

```
5]: import numpy as np

    # Создаем матрицу 4x4 с случайными целыми числами от 1 до 50
    matrix = np.random.randint(1, 51, size=(4, 4))

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

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

    # Вычисляем след матрицы (сумму элементов на главной диагонали)
    matrix_trace = np.trace(matrix)
    print("\nСлед матрицы (сумма элементов на главной диагонали):", matrix_trace)
```

```
Исходная матрица:
[[48  1 27 12]
 [28 34 38 12]
 [27 46 21 17]
 [12 35 47 21]]

Транспонированная матрица:
[[48 28 27 12]
 [ 1 34 46 35]
 [27 38 21 47]
 [12 12 17 21]]

След матрицы (сумма элементов на главной диагонали): 124
```

Рисунок 11. Практическое задание 9

5. Выполнено индивидуальное задание 7 вариант

```
import numpy as np

# Матрица коэффициентов
A = np.array([[1, -2/3, 0],
              [-1, -1, 1],
              [1, 1, 1]])

# Вектор констант
B = np.array([0, 100, 1000])

# Метод Крамера
def cramer(A, B):
  det_A = np.linalg.det(A)
  if det_A == 0:
    return "Система не имеет решений или имеет бесконечно много решений"

  A1 = A.copy()
  A1[:, 0] = B
  det_A1 = np.linalg.det(A1)
  x = det_A1 / det_A

  A2 = A.copy()
  A2[:, 1] = B
  det_A2 = np.linalg.det(A2)
  y = det_A2 / det_A

  A3 = A.copy()
  A3[:, 2] = B
  det_A3 = np.linalg.det(A3)
  z = det_A3 / det_A
  return x, y, z


# Матричный метод (обратная матрица)
def matrix_method(A, B):
  try:
    A_inv = np.linalg.inv(A)
    x = np.dot(A_inv, B)
    return x
  except np.linalg.LinAlgError:
      return "Система не имеет решений или имеет бесконечно много решений"


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

# Решение методом Крамера
solution_cramer = cramer(A, B)
```

```
print("Решение с помощью np.linalg.solve:", solution_linalg)
print("Решение методом Крамера:", solution_cramer)
print("Решение матричным методом (обратная матрица):", solution_matrix)
```

```
Решение с помощью np.linalg.solve: [180. 270. 550.]
Решение методом Крамера: (180.00000000000003, 270.00000000000006, 549.9999999999998)
Решение матричным методом (обратная матрица): [180. 270. 550.]
```

Рисунок 12. Индивидуальное задание

6. Добавил задание в репозиторий

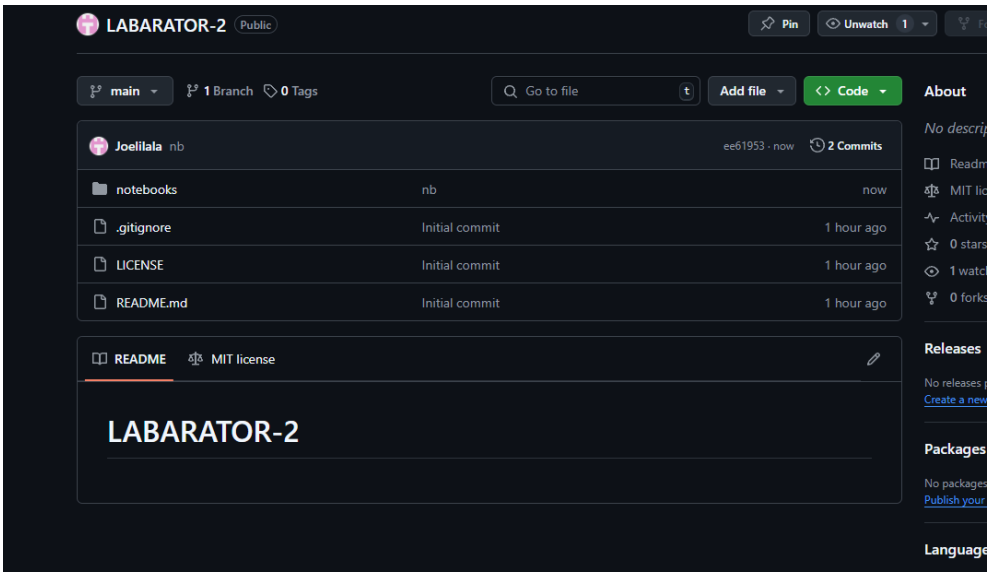

Рисунок 13. Добавил задания

Ответы на контрольные вопросы:

1. Каково назначение библиотеки NumPy?

Библиотека NumPy (Numerical Python) — это фундаментальная библиотека для научных вычислений в Python. Ее основное назначение заключается в предоставлении мощных инструментов для работы с многомерными массивами и матрицами, а также для выполнения математических операций над этими структурами данных.

2. Что такое массивы ndarray?

ndarray (n-dimensional array) — это основной объект в библиотеке NumPy, представляющий собой многомерный массив однотипных элементов.

По сути, это таблица данных (или многомерная таблица), где все элементы имеют один и тот же тип (например, целые числа, числа с плавающей точкой, булевы значения или строки).

3. Как осуществляется доступ к частям многомерного массива?

Доступ к частям многомерного массива осуществляется с помощью индексов, которые указывают на конкретные элементы или срезы массива. В зависимости от языка программирования синтаксис может немного различаться, но общая концепция остается одинаковой.

4. Как осуществляется расчет статистик по данным?

1. Сбор данных

Источники данных: Данные могут поступать из различных источников, таких как базы данных, CSV-файлы, API и т.д.

Формат данных: Убедитесь, что данные находятся в удобном формате для анализа.

2. Предварительная обработка данных

Очистка данных: Удаление или исправление пропусков, дубликатов и аномалий.

Трансформация данных: Приведение данных к единому формату, например, преобразование типов (числовые, категориальные), нормализация и стандартизация.

Фильтрация: Удаление ненужных или нерелевантных данных.

3. Исследовательский анализ данных (EDA)

Визуализация: Построение графиков и диаграмм (гистограммы, диаграммы рассеяния, боксплоты) для понимания распределения и взаимосвязей между переменными.

Статистические описания: Вычисление основных статистических показателей.

4. Расчет статистик

Вот некоторые основные статистики, которые часто рассчитываются:

Центральные тенденции:

Среднее (Mean): Сумма всех значений, деленная на количество значений.

Медиана (Median): Среднее значение, которое делит набор данных на две равные части.

Мода (Mode): Наиболее часто встречающееся значение в наборе данных.

5. Как выполняется выборка данных из массивов ndarray?

Выборка данных из ndarray выполняется через:

Индексация: Доступ к отдельным элементам по их координатам (например, arr[0, 1]).

Срезы (slicing): Извлечение подмассивов по диапазонам индексов (arr[1:3, :]). Срезы создают представления (views), а не копии.

Булева индексация: Выбор элементов на основе логических условий (arr[arr > 5]).

Fancy indexing: Выбор элементов с использованием массивов индексов.

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

Основные виды матриц и векторов:

Вектор-строка: 1 строка, n столбцов.

Вектор-столбец: n строк, 1 столбец.

Квадратная матрица: Количество строк равно количеству столбцов (n x n).

Нулевая матрица: Все элементы равны нулю.

Единичная матрица: Квадратная матрица с единицами на главной диагонали и нулями в остальных местах.

Диагональная матрица: Квадратная матрица, все элементы вне главной диагонали равны нулю.

Треугольная матрица (верхняя/нижняя): Квадратная матрица, у которой все элементы ниже/выше главной диагонали равны нулю.

7. Как выполняется транспонирование матриц?

Транспонирование матрицы - это операция, при которой строки и столбцы матрицы меняются местами. Строки становятся столбцами, а столбцы становятся строками.

Если исходная матрица имеет размерность m x n, то транспонированная матрица будет иметь размерность n x m.

В NumPy для транспонирования матрицы используется метод .T или функция np.transpose().

8. Приведите свойства операции транспонирования матриц.

1.(A.T).T == A (Транспонирование дважды возвращает исходную матрицу)

2.(A + B).T == A.T + B.T (Транспонирование суммы равно сумме транспонированных)

3.(c × A).T == c × A.T (Транспонирование произведения на скаляр равно произведению скаляра на транспонированную)

4.(A @ B).T == B.T @ A.T (Транспонирование произведения равно произведению транспонированных в обратном порядке). @ - оператор матричного умножения в NumPy (эквивалентно np.dot(A, B)).

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

Для транспонирования матриц в NumPy существует несколько удобных методов:

1.numpy.transpose() — основной метод для транспонирования матрицы. Он меняет строки и столбцы местами

2. Метод .T — альтернатива transpose(), который делает то же самое, но короче и удобнее в записи.

10. Какие существуют основные действия над матрицами?

Основные операции над матрицами в Python с использованием библиотеки NumPy включают:

1. Создание матрицы:
   Используются функции np.array() или np.matrix().
2. Транспонирование:
   Можно использовать функцию np.transpose() или метод .T.
3. Умножение матриц:
   Для умножения используется оператор @ или функция np.dot().
4. Сложение/вычитание матриц:
   Применяются операторы + и -.
5. Обращение матрицы:
   Используется функция np.linalg.inv()
6. Определение детерминанта:
   Функция np.linalg.det()
7. Вычисление собственных значений и векторов:
   С помощью функций np.linalg.eigvals() и np.linalg.eig().
8. Нахождение ранга матрицы:

Определяется функцией np.linalg.matrix_rank().

11. Как осуществляется умножение матрицы на число?

В Python с использованием библиотеки NumPy умножение матрицы на число осуществляется просто с помощью оператора *.

12. Какие свойства операции умножения матрицы на число?

– Ассоциативность
– Дистрибутивность относительно сложения матриц
– Дистрибутивность относительно сложения скаляров
– Умножение на единицу
– Умножение на ноль
– Коммутативность

13. Как осуществляется операции сложения и вычитания матриц?

В NumPy сложение и вычитание матриц выполняется поэлементно, с использованием операторов + и -. Матрицы должны иметь одинаковый размер

14. Каковы свойства операций сложения и вычитания матриц?

– Коммутативность
– Ассоциативность
– Дистрибутивность
– Существование нулевой матрицы
– Существование противоположной матрицы
– Размерность

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

– Сложение: + или np.add()
– Вычитание: - или np.subtract()

16. Как осуществляется операция умножения матриц?

– * или np.multiply() - поэлементное умножение.
– @ или np.matmul() - матричное умножение (стандартное умножение матриц).

– np.dot() - универсальное умножение (выбирает тип умножения в зависимости от размерности массивов).

17. Каковы свойства операции умножения матриц?

– Некоммутативность
– Ассоциативность
– Дистрибутивность
– Умножение на скаляр
– Умножение на единичную матрицу
– Транспонирование
– Размерности
– Нельзя делить

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

– C = A @ B
– numpy.matmul(A, B)
– numpy.dot(A, B)

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

Определитель — это скалярное значение, которое может быть вычисленотолько для квадратных матриц

Свойства:

– Определитель единичной матрицы
– Умножение на скаляр
– Транспонирование
– Перестановка строк/столбцов
– Умножение матриц
– Нулевая строка/столбец
– Линейная зависимость

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

Для нахождения определителя матрицы в NumPy используется функция numpy.linalg.det()

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

Обратная матрица, обозначаемая A^{-1}, — это такая матрица, которая при умножении на исходную матрицу A дает единичную матрицу

Алгоритм: NumPy использует LU-разложение.

22. Каковы свойства обратной матрицы?

Существование: Обратная матрица существует только для квадратных матриц, которые являются невырожденными (т.е. имеют ненулевое определитель).

Обозначение: Обратная матрица для матрицы A обозначается как A^(-1).

Не единственность: если матрица A имеет обратную, то её обратная матрица уникальна.

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

Для нахождения обратной матрицы используется функция numpy.linalg.inv(A)

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(A, b):
 det_A = np.linalg.det(A)
if np.isclose(det_A, 0): return None  # No solution/infinite solutions
```

```
x = [np.linalg.det(np.column_stack([b if j == i else A[:,j] for j in range(A.shape[0])])) / det_A for i in range(A.shape[0])]
  return np.array(x)

# Пример:
A = np.array([[2, 1], [1, 3]])
b = np.array([8, 10])
x = cramer(A, b)
print(x) # [3.42857143 1.14285714]
```

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):
  """Решает Ax = b матричным методом."""
  try:
    A_inv = np.linalg.inv(A)
    x = A_inv @ b
    return x
  except np.linalg.LinAlgError:
    return None  # Нет решения (матрица вырождена)

# Пример:
A = np.array([[2, 1], [1, 3]])
b = np.array([8, 10])
x = matrix_solve(A, b)
print(x) # [3.42857143 1.14285714]
```

Вывод: в ходе лабораторной работы были приобретены навыки работы с базовыми возможностями библиотеки NumPy

Ссылка на GitHub: https://github.com/Joelilala/LABARATOR-2