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 6

Выполнил:
Якушенко Антон Андреевич
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:
Ассистент департамента цифровых, робототехнических систем и электроники Хацукова А.И

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г.

ТЕМА: ОСНОВЫ РАБОТЫ С БИБЛЕОТЕКОЙ NUMPY

Цель работы: исследовать базовые возможности библиотеки NumPy языка программирования Python.

Ссылка на репозиторий: https://github.com/Yakush766/LB2M.git

Порядок выполнения работы:

Задание 1: Создание и изменение массивов

[1]:
```
import numpy as np

# Создание массива NumPy 3x3 с числами от 1 до 9
arr = np.arange(1, 10).reshape((3, 3))

# Умножение всех элементов массива на 2
arr = arr * 2

# Замена всех элементов больше 10 на 0
arr[arr > 10] = 0

# Вывод итогового массива
print(arr)
```

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

Рисунок 1. Выполнение задания 1

Задание 2: Работа с булевыми масками

[1]:
```
import numpy as np

# 1. Создаем массив NumPy из 20 случайных целых чисел от 1 до 100
arr = np.random.randint(1, 101, 20)
print("Исходный массив:", arr)

# 2. Находим элементы, которые делятся на 5 без остатка
divisible_by_5 = arr[arr % 5 == 0]
print("Элементы, делящиеся на 5:", divisible_by_5)

# 3. Заменяем элементы, делящиеся на 5, на -1
arr[arr % 5 == 0] = -1
print("Обновленный массив:", arr)
```

```
Исходный массив: [79 73 68 75 90 69 39 58 80 84 86 85 14 73 83 62 61 18 77 97]
Элементы, делящиеся на 5: [75 90 80 85]
Обновленный массив: [79 73 68 -1 -1 69 39 58 -1 84 86 -1 14 73 83 62 61 18 77 97]
```

Рисунок 2. Выполнение задания 2

Задание 3: Объединение и разбиение массивов

```
[1]: import numpy as np

# 1. Создаем два массива NumPy размером 1x5, заполненные случайными числами от 0 до 50
arr1 = np.random.randint(0, 51, size=(1, 5))
arr2 = np.random.randint(0, 51, size=(1, 5))

print("Первый массив:")
print(arr1)
print("Второй массив:")
print(arr2)

# 2. Объединяем эти массивы в один двумерный массив (по строкам)
combined_arr = np.concatenate((arr1, arr2), axis=0)

print("Объединенный массив:")
print(combined_arr)

# 3. Разделяем полученный массив на два массива, каждый из которых содержит 5 элементов
arr3 = combined_arr[0]
arr4 = combined_arr[1]
# Alternatively, using split:
# arr3, arr4 = np.split(combined_arr, 2)
# arr3 = arr3[0]
# arr4 = arr4[0]


print("Первый подмассив (после разделения):")
print(arr3)
print("Второй подмассив (после разделения):")
print(arr4)
```

```
Первый массив:
[[35  7 20 45 22]]
Второй массив:
[[24  6  7 24 47]]
Объединенный массив:
[[35  7 20 45 22]
 [24  6  7 24 47]]
Первый подмассив (после разделения):
[35  7 20 45 22]
Второй подмассив (после разделения):
[24  6  7 24 47]
```

Рисунок 3. Выполнение задания 3

Задание 4: Генерация и работа с линейными последовательностями

[1]:
```
import numpy as np

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

# 2. Вычисляем сумму всех элементов
total_sum = np.sum(arr)
print("Сумма всех элементов:", total_sum)

# 3. Вычисляем сумму положительных элементов
positive_elements = arr[arr > 0]
positive_sum = np.sum(positive_elements)
print("Сумма положительных элементов:", positive_sum)

# 4. Вычисляем сумму отрицательных элементов
negative_elements = arr[arr < 0]
negative_sum = np.sum(negative_elements)
print("Сумма отрицательных элементов:", negative_sum)
```

```
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.        ]
Сумма всех элементов: 7.105427357601002e-15
Сумма положительных элементов: 127.55102040816328
Сумма отрицательных элементов: -127.55102040816327
```

Рисунок 4. Выполнение задания 4

Задание 5: Работа с диагональными и единичными матрицами

[1]:
```python
import numpy as np

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

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

# 3. Находим сумму всех элементов каждой из этих матриц
identity_sum = np.sum(identity_matrix)
diagonal_sum = np.sum(diagonal_matrix)

print("Сумма элементов единичной матрицы:", identity_sum)
print("Сумма элементов диагональной матрицы:", diagonal_sum)

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

```
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]]
Сумма элементов единичной матрицы: 4.0
Сумма элементов диагональной матрицы: 50
Сумма элементов диагональной матрицы больше.
```

Рисунок 5. Выполнение задания 5

Задание 6: Создание и базовые операции с матрицами

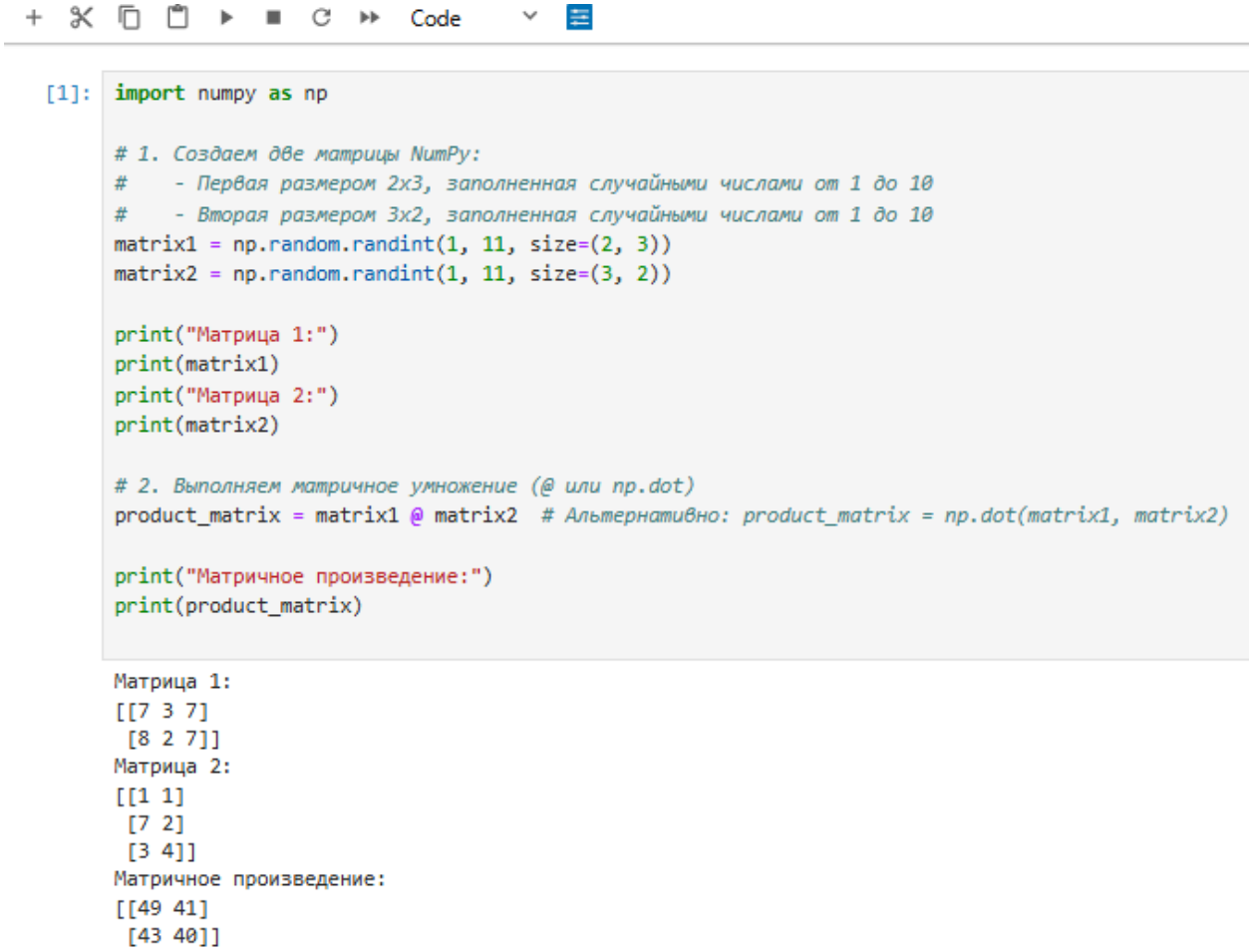
Code

[1]:
```
import numpy as np

# 1. Создаем две квадратные матрицы NumPy размером 3x3, заполненные случайными целыми числами от 1 до 20
matrix1 = np.random.randint(1, 21, size=(3, 3))
matrix2 = np.random.randint(1, 21, size=(3, 3))

print("Матрица 1:")
print(matrix1)
print("Матрица 2:")
print(matrix2)

# 2. Вычисляем их сумму
sum_matrix = matrix1 + matrix2
print("Сумма матриц:")
print(sum_matrix)

# 3. Вычисляем их разность
diff_matrix = matrix1 - matrix2
print("Разность матриц:")
print(diff_matrix)

# 4. Вычисляем их поэлементное произведение
elementwise_product = matrix1 * matrix2  # Или np.multiply(matrix1, matrix2)
print("Поэлементное произведение матриц:")
print(elementwise_product)
```

```
Матрица 1:
[[ 5 11  2]
 [13  1 16]
 [ 5 16 12]]
Матрица 2:
[[ 9  1 20]
 [16  3  7]
 [15 19 13]]
Сумма матриц:
[[14 12 22]
 [29  4 23]
 [20 35 25]]
Разность матриц:
[[ -4  10 -18]
 [ -3  -2   9]
 [-10  -3  -1]]
Поэлементное произведение матриц:
[[ 45  11  40]
 [208   3 112]
 [ 75 304 156]]
```

Рисунок 6. Выполнение задания 6

Задание 7: Умножение матриц

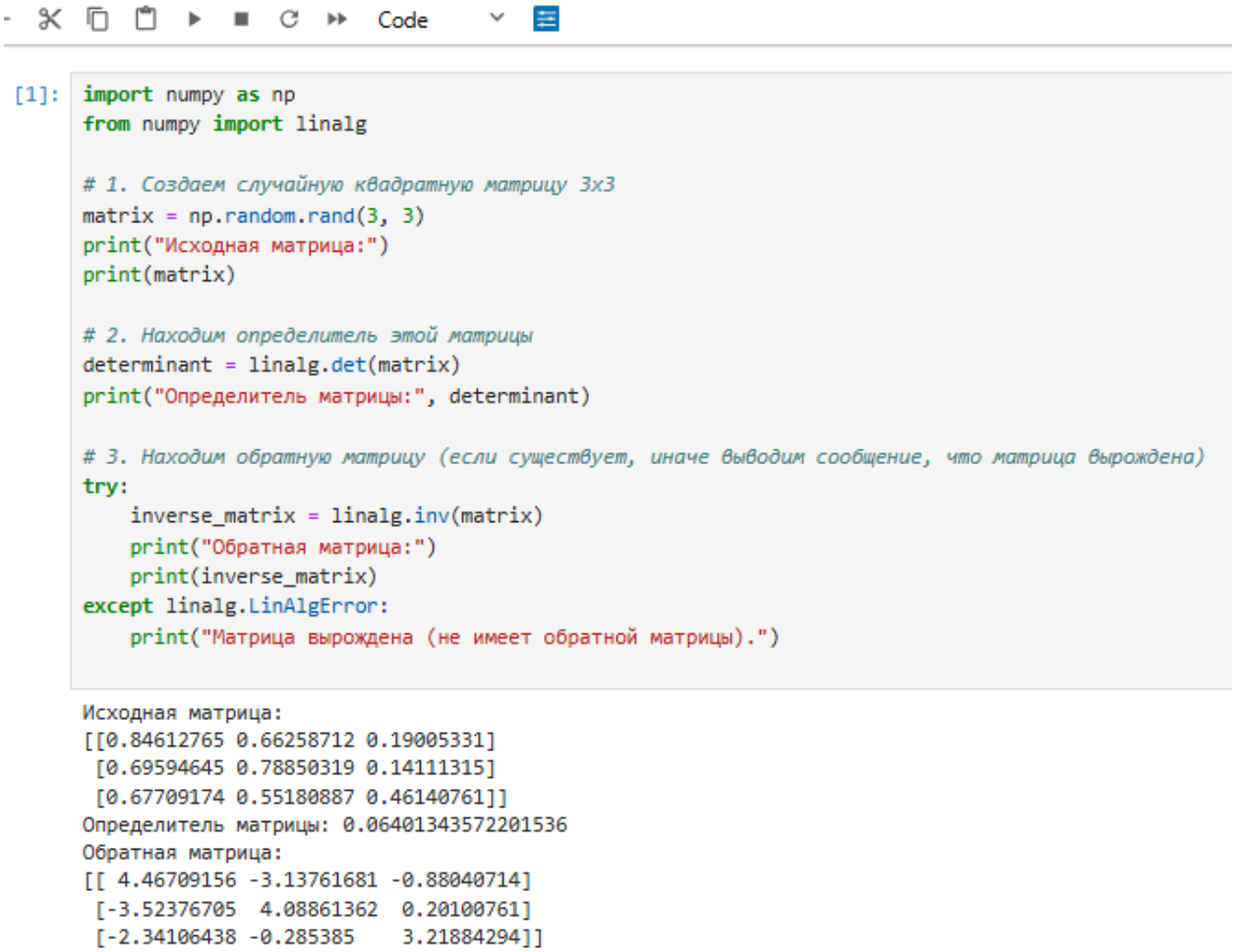


```
[1]: import numpy as np

# 1. Создаем две матрицы NumPy:
#    - Первая размером 2x3, заполненная случайными числами от 1 до 10
#    - Вторая размером 3x2, заполненная случайными числами от 1 до 10
matrix1 = np.random.randint(1, 11, size=(2, 3))
matrix2 = np.random.randint(1, 11, size=(3, 2))

print("Матрица 1:")
print(matrix1)
print("Матрица 2:")
print(matrix2)

# 2. Выполняем матричное умножение (@ или np.dot)
product_matrix = matrix1 @ matrix2  # Альтернативно: product_matrix = np.dot(matrix1, matrix2)

print("Матричное произведение:")
print(product_matrix)
```

```
Матрица 1:
[[7 3 7]
 [8 2 7]]
Матрица 2:
[[1 1]
 [7 2]
 [3 4]]
Матричное произведение:
[[49 41]
 [43 40]]
```

Рисунок 7. Выполнение задания 7

Задание 8: Определитель и обратная матрица



```
[1]: import numpy as np
     from numpy import linalg

     # 1. Создаем случайную квадратную матрицу 3x3
     matrix = np.random.rand(3, 3)
     print("Исходная матрица:")
     print(matrix)

     # 2. Находим определитель этой матрицы
     determinant = linalg.det(matrix)
     print("Определитель матрицы:", determinant)

     # 3. Находим обратную матрицу (если существует, иначе выводим сообщение, что матрица вырождена)
     try:
         inverse_matrix = linalg.inv(matrix)
         print("Обратная матрица:")
         print(inverse_matrix)
     except linalg.LinAlgError:
         print("Матрица вырождена (не имеет обратной матрицы).")
```

```
Исходная матрица:
[[0.84612765 0.66258712 0.19005331]
 [0.69594645 0.78850319 0.14111315]
 [0.67709174 0.55180887 0.46140761]]
Определитель матрицы: 0.06401343572201536
Обратная матрица:
[[ 4.46709156 -3.13761681 -0.88040714]
 [-3.52376705  4.08861362  0.20100761]
 [-2.34106438 -0.285385    3.21884294]]
```

Рисунок 8. Выполнение задания 8

Задание 9: Транспонирование и след матрицы

[1]:
```python
import numpy as np

# 1. Создаем матрицу NumPy размером 4x4, содержащую случайные целые числа от 1 до 50
matrix = np.random.randint(1, 51, size=(4, 4))

# 2. Выводим исходную матрицу
print("Исходная матрица:")
print(matrix)

# 3. Выводим транспонированную матрицу
transposed_matrix = matrix.T  # Альтернативно: transposed_matrix = np.transpose(matrix)
print("Транспонированная матрица:")
print(transposed_matrix)

# 4. Выводим след матрицы (сумму элементов на главной диагонали)
matrix_trace = np.trace(matrix)
print("След матрицы:", matrix_trace)
```

```
Исходная матрица:
[[45 35 48 44]
 [17 12 19 19]
 [35 17 35 39]
 [35  9 41 31]]
Транспонированная матрица:
[[45 17 35 35]
 [35 12 17  9]
 [48 19 35 41]
 [44 19 39 31]]
След матрицы: 123
```

Рисунок 9. Выполнение задания 9

Задание 10: Системы линейных уравнений

```
[1]: import numpy as np
     from numpy import linalg

     # Коэффициенты системы уравнений (матрица A)
     A = np.array([[2, 3, -1],
                   [4, -1, 2],
                   [3, 5, 4]])

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

     # Решение системы уравнений Ax = b с помощью linalg.solve
     x = linalg.solve(A, b)

     # Выводим результат
     print("Решение системы:")
     print("x =", x[0])
     print("y =", x[1])
     print("z =", x[2])
```

```
Решение системы:
x = 2.0
y = 0.0
z = -1.0
```

Рисунок 10. Выполнение задания 10

Выполнение индивидуального задания:

[1]:
```python
import numpy as np
from numpy import linalg

# Условия задачи:
# - x: Количество воды в первом резервуаре
# - y: Количество воды во втором резервуаре
# - z: Количество воды в третьем резервуаре

# Уравнения:
# 1. x = 2y  (В первый резервуар поступает в два раза больше воды, чем во второй)
# 2. z = y + 50 (В третий резервуар поступает на 50 литров больше, чем во второй)
# 3. x + y + z = 500 (Всего в систему поступает 500 литров воды)

# Приводим уравнения к виду Ax = b:
# 1. x - 2y + 0z = 0
# 2. 0x - y + z = 50
# 3. x + y + z = 500

# 1. Матричный метод (linalg.solve)
A = np.array([[1, -2, 0],
              [0, -1, 1],
              [1, 1, 1]])
b = np.array([0, 50, 500])

x_linalg = linalg.solve(A, b)
print("Решение матричным методом (linalg.solve):")
print("Резервуар 1:", x_linalg[0], "литров")
print("Резервуар 2:", x_linalg[1], "литров")
print("Резервуар 3:", x_linalg[2], "литров")


# 2. Метод Крамера
def kramer(A, b):
  """Решает систему линейных уравнений методом Крамера.

  Args:
    A: Матрица коэффициентов (NumPy array).
    b: Вектор правой части (NumPy array).

  Returns:
    Вектор решения (NumPy array), если система имеет единственное решение.
    None, если определитель матрицы равен нулю (система не имеет решения или имеет бесконечно много решений).
  """
  det_A = linalg.det(A)
  if np.isclose(det_A, 0):  # Проверка, что определитель не равен нулю (с учетом погрешности вычислений)
    return None  # Матрица вырождена

  n = A.shape[0]  # Размерность матрицы
  x = np.zeros(n)  # Инициализируем вектор решения
```

Рисунок 11.1. Выполнение индивидуального задания

```python
  for i in range(n):
    # Создаем матрицу A_i, заменяя i-й столбец матрицы A на вектор b
    A_i = A.copy()  # Создаем копию матрицы A, чтобы не изменять исходную матрицу
    A_i[:, i] = b  # Заменяем i-й столбец
    x[i] = linalg.det(A_i) / det_A  # Вычисляем x_i

  return x


x_kramer = kramer(A, b)
print("\nРешение методом Крамера:")

if x_kramer is not None:
  print("Резервуар 1:", x_kramer[0], "литров")
  print("Резервуар 2:", x_kramer[1], "литров")
  print("Резервуар 3:", x_kramer[2], "литров")
else:
  print("Система не имеет единственного решения (матрица вырождена).")


# Сравнение результатов
print("\nСравнение результатов:")
print("Матричный метод:", x_linalg)
if x_kramer is not None:
    print("Метод Крамера:  ", x_kramer)
else:
    print("Метод Крамера: не применился из-за вырожденности матрицы.")
print("Разница между решениями:", x_linalg - x_kramer if x_kramer is not None else "Метод Крамера не применился")
```

```
Решение матричным методом (linalg.solve):
Резервуар 1: 225.0 литров
Резервуар 2: 112.5 литров
Резервуар 3: 162.5 литров

Решение методом Крамера:
Резервуар 1: 225.00000000000003 литров
Резервуар 2: 112.50000000000001 литров
Резервуар 3: 162.49999999999997 литров

Сравнение результатов:
Матричный метод: [225.  112.5 162.5]
Метод Крамера:   [225.  112.5 162.5]
Разница между решениями: [-2.84217094e-14 -1.42108547e-14  2.84217094e-14]
```

Рисунок 11.2. Выполнение индивидуального задания

Вывод: исследовали базовые возможности библиотеки NumPy языка программирования Python.

Ответы на контрольные вопросы:

1. Каково назначение библиотеки NumPy?

NumPy (Numerical Python) — это библиотека Python, предназначенная для эффективной работы с многомерными массивами, выполнения математических и логических операций над ними. Она предоставляет мощные инструменты для линейной алгебры, преобразований Фурье и генерации случайных чисел. NumPy является основой для многих других научных библиотек Python, таких как SciPy, Matplotlib и Scikit-learn.

2. Что такое массивы ndarray?

ndarray (n-dimensional array) — это основной тип данных в NumPy, представляющий собой многомерный массив однотипных элементов. Он позволяет хранить и обрабатывать большие объемы данных эффективно, благодаря оптимизированным алгоритмам и структурам хранения. Массивы ndarray могут иметь любую размерность (количество осей), например, одномерные (векторы), двумерные (матрицы) и т.д.

3. Как осуществляется доступ к частям многомерного массива?

Доступ к элементам массива ndarray осуществляется с использованием индексов, заключенных в квадратные скобки []. Для многомерных массивов индексы разделяются запятыми. Например:

- arr[0] - доступ к первому элементу одномерного массива.
- arr[1, 2] - доступ к элементу, находящемуся во второй строке и третьем столбце двумерного массива.
- Можно использовать срезы (slices) для доступа к подмножествам массива: arr[1:5], arr[:, 0].

4. Как осуществляется расчет статистик по данным?

NumPy предоставляет широкий набор функций для расчета статистических показателей по массивам:

- np.mean(arr) - среднее значение
- np.median(arr) - медиана
- np.std(arr) - стандартное отклонение

– np.var(arr) - дисперсия

– np.min(arr) - минимальное значение

– np.max(arr) - максимальное значение

– np.sum(arr) - сумма всех элементов

Эти функции могут применяться как ко всему массиву, так и к отдельным осям, используя аргумент axis.

5. Как выполняется выборка данных из массивов ndarray?

Выборка данных из ndarray может осуществляться несколькими способами:

– Индексация и срезы: Как описано в вопросе 3.

– Булева индексация: Создание булева массива той же формы, что и исходный, где True соответствует элементам, которые нужно выбрать.

– Целочисленная индексация: Использование массива целых чисел для указания индексов элементов, которые нужно выбрать.

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

– Вектор: Одномерный массив.

– Матрица: Двумерный массив.

– Нулевая матрица: Матрица, заполненная нулями.

– Матрица единиц: Матрица, заполненная единицами.

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

7. Как выполняется транспонирование матриц?

Транспонирование матрицы — это операция, при которой строки и столбцы матрицы меняются местами.

8. Приведите свойства операции транспонирования матриц.

– (A^T)^T = A - Транспонирование транспонированной матрицы возвращает исходную матрицу.

– (A + B)^T = A^T + B^T - Транспонирование суммы матриц равно сумме транспонированных матриц.

– (kA)^T = k(A^T) - Транспонирование произведения матрицы на скаляр равно произведению скаляра на транспонированную матрицу.

– (AB)^T = B^T A^T - Транспонирование произведения матриц равно произведению транспонированных матриц в обратном порядке.

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

В NumPy транспонирование можно выполнить несколькими способами:

– arr.T - атрибут, возвращающий транспонированное представление массива.

– np.transpose(arr) - функция, возвращающая транспонированную копию массива.

10. Какие существуют основные действия над матрицами?

– Сложение и вычитание.

– Умножение на число (скалярное умножение).

– Матричное умножение.

– Транспонирование.

– Вычисление определителя.

– Нахождение обратной матрицы.

11. Как осуществляется умножение матрицы на число?

Умножение матрицы на число осуществляется путем умножения каждого элемента матрицы на это число.

12. Какие свойства операции умножения матрицы на число?

– Коммутативность: kA = Ak

– Ассоциативность: (kl)A = k(lA)

– Дистрибутивность относительно сложения матриц: k(A + B) = kA + kB

– Дистрибутивность относительно сложения чисел: (k + l)A = kA + lA

13. Как осуществляется операция сложения и вычитания матриц?

Сложение и вычитание матриц осуществляется поэлементно. Матрицы должны иметь одинаковую форму.

14. Каковы свойства операций сложения и вычитания матриц?

– Коммутативность (для сложения): A + B = B + A

– Ассоциативность (для сложения): (A + B) + C = A + (B + C)

– Существование нулевой матрицы: A + 0 = A

– Дистрибутивность относительно умножения на скаляр: k(A + B) = kA + kB

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

– arr1 + arr2 - сложение

– arr1 - arr2 - вычитание

16. Как осуществляется операция умножения матриц?

Умножение матрицы A (m x n) на матрицу B (n x p) дает матрицу C (m x p). Элемент C[i, j] вычисляется как скалярное произведение i-й строки матрицы A и j-го столбца матрицы B.

17. Каковы свойства операции умножения матриц?

– Ассоциативность: (AB)C = A(BC)

– Дистрибутивность относительно сложения: A(B + C) = AB + AC и (A + B)C = AC + BC

– Умножение на единичную матрицу: AI = IA = A (где I - единичная матрица)

– Не коммутативность: В общем случае AB ≠ BA

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

– np.dot(arr1, arr2) - функция для матричного умножения.

– arr1 @ arr2 - оператор матричного умножения (начиная с Python 3.5).

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

Определитель — это скалярная величина, характеризующая квадратную матрицу.

– Свойства:

– Определитель единичной матрицы равен 1.

– Если поменять местами две строки (или столбца) матрицы, то определитель изменит знак.

– Если матрица имеет нулевую строку (или столбец), то ее определитель равен 0.

– Определитель произведения матриц равен произведению их определителей: det(AB) = det(A) * det(B).

– Определитель транспонированной матрицы равен определителю исходной матрицы: det(A^T) = det(A).

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

– np.linalg.det(arr) - функция для вычисления определителя.

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

Обратная матрица (A^{-1}) — это матрица, при умножении которой н

а исходную матрицу A получается единичная матрица ($AA^{-1} = A^{-1}A = I$). Обратная матрица существует только для невырожденных (имеющих ненулевой определитель) квадратных матриц.

Алгоритм нахождения обратной матрицы:

1. Вычислить определитель матрицы. Если он равен 0, обратная матрица не существует.

2. Найти матрицу алгебраических дополнений (кофакторов).

3. Транспонировать матрицу алгебраических дополнений (получить присоединенную матрицу).

4. Разделить каждый элемент присоединенной матрицы на определитель исходной матрицы.

22. Каковы свойства обратной матрицы?

– $AA^{-1} = A^{-1}A = I$ (определение).

– $(A^{-1})^{-1} = A$

– $(AB)^{-1} = B^{-1}A^{-1}$

– $(A^T)^{-1} = (A^{-1})^T$

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

– np.linalg.inv(arr) - функция для вычисления обратной матрицы.