МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Кафедра инфокоммуникаций

Отчет

по лабораторной работе №10

«Исследование методов работы с матрицами и векторами с помощью библиотеки NumPy»

по дисциплине:

«Введение в системы искусственного интеллекта»

Вариант 8

Выполнил: студент группы ИВТ-б-о-18-1 (2)
Михайличенко Руслан Михайлович
_____(подпись)
Проверил:
Воронкин Роман Александрович
_____(подпись)

Ставрополь, 2022 г.

Цель работы: исследовать методы работы с матрицами и векторами с помощью библиотеки NumPy языка программирования Python.

Ход работы:

```
#Импорт библиотеки
import numpy as np

#Создание вектора-строки
v_hor_np = np.array([1, 2])
print("Создание вектора-строки")
print(v_hor_np)

#Создание нулеового вектора
print("Создание нулеового вектора")
print(np.zeros((5,)))

#Создание единичного вектора
print("Создание единичного вектора")
print(np.ones((5,)))

#Создание вектора-столбца
print("Создание вектора-столбца")
v_vert_np = np.array([[1], [2]])
print(v_vert_np)

#Создание нулевого вектора-столбца
print("Создание нулевого вектора-столбца")
v_vert_zeros = np.zeros((5, 1))
print(v_vert_zeros)

#Создание нулевого вектора-столбца
print("Создание единичного вектора-столбца")
v_vert_ones = np.ones((5, 1))
print(v_vert_ones)
```

Рисунок 1 – Пример

```
Создание вектора-строки
[1 2]
Создание нулеового вектора
[0. 0. 0. 0. 0.]
Создание единичного вектора
[1. 1. 1. 1. 1.]
Создание вектора-столбца
[[1]
 [2]]
Создание нулевого вектора-столбца
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]]
Создание единичного вектора-столбца
[[1.]
 [1.]
 [1.]
 [1.]
 [1.]]
```

Рисунок 2 – Результат работы кода

```python
#Квадратная матрица
#Array
print("Методом массива")
m_sqr_arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(m_sqr_arr)

#Matrix
print("Методом Matrix")
m_sqr_mx = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(m_sqr_mx)

#Matlab
print("Методом Matlab")
m_sqr_mx = np.matrix('1 2 3; 4 5 6; 7 8 9')
print(m_sqr_mx)
```

Рисунок 3 – Пример

```
Методом массива
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Методом Matrix
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Методом Matlab
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Рисунок 4 – Результат работы

```
#Диагональная матрица
#Вручную
print("Вручную")
m_diag = [[1, 0, 0], [0, 5, 0], [0, 0, 9]]
m_diag_np = np.matrix(m_diag)
print(m_diag_np)

#По средством NumPy
print("NumPy")
m_sqr_mx = np.matrix('1 2 3; 4 5 6; 7 8 9')
diag = np.diag(m_sqr_mx)
m_diag_np = np.diag(np.diag(m_sqr_mx))
print(m_diag_np)
```

Рисунок 5 – Пример

```
Вручную
[[1 0 0]
 [0 5 0]
 [0 0 9]]
NumPy
[[1 0 0]
 [0 5 0]
 [0 0 9]]
```

Рисунок 6 – Результат работы

```
#Единичная матрица
#Вручную
print("Вручную")
m_e = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
m_e_np = np.matrix(m_e)
print(m_e_np)

#Функция eye()
print("eye()")
m_eye = np.eye(3)
print(m_eye)

#Функция identity()
print("identity()")
m_idnt = np.identity(3)
print(m_idnt)
```

Рисунок 7 – Пример

```
Вручную
[[1 0 0]
 [0 1 0]
 [0 0 1]]
eye():
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
identity():
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Рисунок 8 – Результат работы

```
#Нулевая матрица
#Функция zeros()
print("zeros()")
m_zeros = np.zeros((3, 3))
print(m_zeros)

print("2x5")
m_var = np.zeros((2, 5))
print(m_var)
```

```
zeros()
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
2x5
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

Рисунок 9 – Решение примера

```
import numpy as np

#Транспонирование матрицы
print("Транспонирование матрицы")
A = np.matrix('1 2 3; 4 5 6')
print(A)
print("transpose():")
A_t = A.transpose()
print(A_t)
print("\n")

#Сокращенный вариант
print("Сокращенный вариант:")
print(A.T)
print("\n")

#Двойное транспонирование
print("Двойное транспонирование")
print(A)
print(A.T)
print((A.T).T)
print("\n")

#Сумма транспонированных матриц
print("Сумма транспонированных матриц")
A = np.matrix('1 2 3; 4 5 6')
B = np.matrix('7 8 9; 0 7 5')
L = (A + B).T
R = A.T + B.T
print(L)
print(R)
print("\n")

#Произведение транспонированных матриц
print("Произведение транспонированных матриц")
A = np.matrix('1 2; 3 4')
B = np.matrix('5 6; 7 8')
L = (A.dot(B)).T
R = (B.T).dot(A.T)
print(L)
print(R)
print("\n")

#Транспонирование произведения матрицы на число
print("Транспонирование произведения матрицы на число")
A = np.matrix('1 2 3; 4 5 6')
k = 3
L = (k * A).T
R = k * (A.T)
print(L)
print(R)
print("\n")

#Определители исходной и транспонированной матрицы совпадают
print("Определители исходной и транспонированной матрицы совпадают")
A = np.matrix('1 2; 3 4')
A_det = np.linalg.det(A)
A_T_det = np.linalg.det(A.T)
print(format(A_det, '.9g'))
print(format(A_T_det, '.9g'))
```

Рисунок 10 – Код примера

```
Транспонирование матрицы
[[1 2 3]
 [4 5 6]]
transpose():
[[1 4]
 [2 5]
 [3 6]]


Сокращенный вариант:
[[1 4]
 [2 5]
 [3 6]]


Двойное транспонирование
[[1 2 3]
 [4 5 6]]
[[1 4]
 [2 5]
 [3 6]]
[[1 2 3]
 [4 5 6]]


Сумма транспонированных матриц
[[ 8  4]
 [10 12]
 [12 11]]
[[ 8  4]
 [10 12]
 [12 11]]


Произведение транспонированных матриц
[[19 43]
 [22 50]]
[[19 43]
 [22 50]]


Транспонирование произведения матрицы на число
[[ 3 12]
 [ 6 15]
 [ 9 18]]
[[ 3 12]
 [ 6 15]
 [ 9 18]]


Определители исходной и транспонированной матрицы совпадают
-2
-2
```

Рисунок 11 – Результат работы

```
#Умножение матрицы на число
print("Умножение матрицы на число")
A = np.matrix('1 2 3; 4 5 6')
C = 3 * A
print(C)
print("\n")

#Произведение единицы и матрицы
print("Произведение единицы и матрицы")
A = np.matrix('1 2; 3 4')
L = 1 * A
R = A
print(L)
print(R)
print("\n")

#Произведение нуля и матрицы
print("Произведение нуля и матрицы")
A = np.matrix('1 2; 3 4')
Z = np.matrix('0 0; 0 0')
L = 0 * A
R = Z
print(L)
print(R)
print("\n")

#Произведение матрицы на сумму чисел
print("Произведение матрицы на сумму чисел")
A = np.matrix('1 2; 3 4')
p = 2
q = 3
L = (p + q) * A
R = p * A + q * A
print(L)
print(R)
print("\n")

#Произведение матрицы на произведение двух чисел
print("Произведение матрицы на произведение двух чисел")
A = np.matrix('1 2; 3 4')
p = 2
q = 3
L = (p * q) * A
R = p * (q * A)
print(L)
print(R)
print("\n")

#Произведение суммы матриц на число
print("Произведение суммы матриц на число")
A = np.matrix('1 2; 3 4')
B = np.matrix('5 6; 7 8')
k = 3
L = k * (A + B)
R = k * A + k * B
print(L)
print(R)
print("\n")
```

Рисунок 12 – Код примера

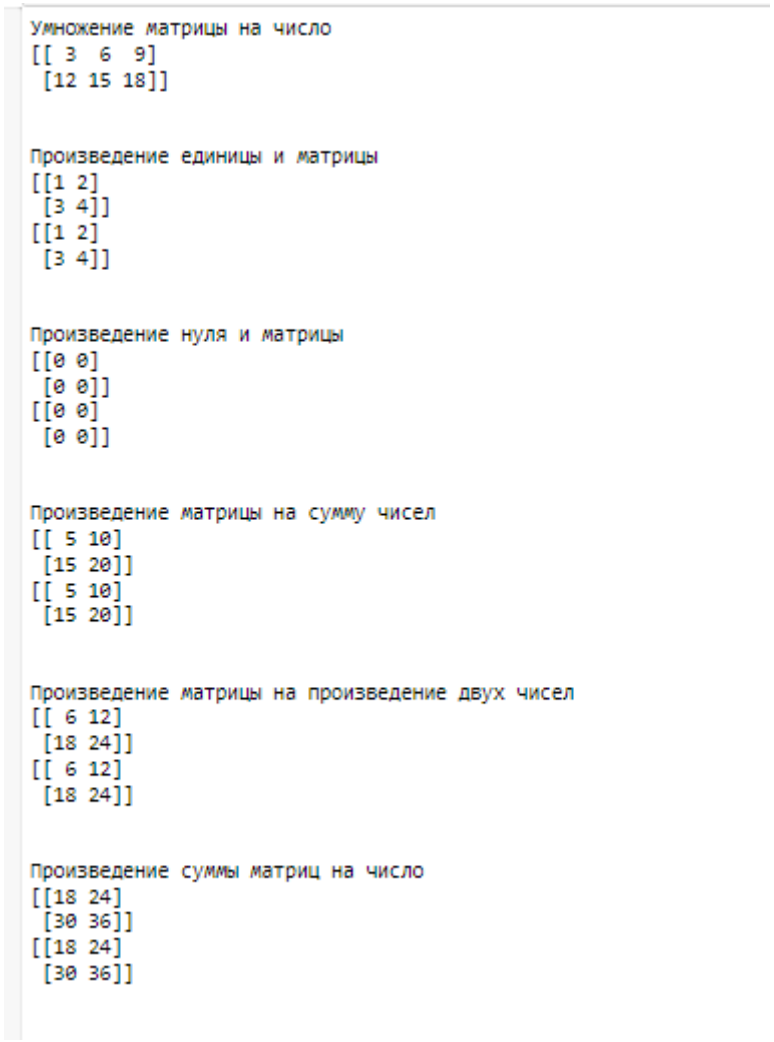

Рисунок 13 – Результат работы

```
#Сложение матриц
print("Сложение матриц")
A = np.matrix('1 6 3; 8 2 7')
B = np.matrix('8 1 5; 6 9 12')
C = A + B
print(C)
print("\n")

#Коммутативность сложения
print("Коммутативность сложения")
A = np.matrix('1 2; 3 4')
B = np.matrix('5 6; 7 8')
L = A + B
R = B + A
print(L)
print(R)
print("\n")

#Ассоциативность сложения
print("Ассоциативность сложения")
A = np.matrix('1 2; 3 4')
B = np.matrix('5 6; 7 8')
C = np.matrix('1 7; 9 3')
L = A + (B + C)
R = (A + B) + C
print(L)
print(R)
print("\n")

#Для любой матрицы существует противоположная ей
print("Для любой матрицы существует противоположная ей")
A = np.matrix('1 2; 3 4')
Z = np.matrix('0 0; 0 0')
L = A + (-1) * A
print(L)
print(Z)
print("\n")
```

Рисунок 14 – Код примера

```
Сложение матриц
[[ 9  7  8]
 [14 11 19]]


Коммутативность сложения
[[ 6  8]
 [10 12]]
[[ 6  8]
 [10 12]]


Ассоциативность сложения
[[ 7 15]
 [19 15]]
[[ 7 15]
 [19 15]]


Для любой матрицы существует противоположная ей
[[0 0]
 [0 0]]
[[0 0]
 [0 0]]
```

Рисунок 15 – Результат работы

```
#Умножение матриц
#Ассоциативность умножения
print("Ассоциативность умножения")
A = np.matrix('1 2; 3 4')
B = np.matrix('5 6; 7 8')
C = np.matrix('2 4; 7 8')
L = A.dot(B.dot(C))
R = (A.dot(B)).dot(C)
print(L)
print(R)
print("\n")

#Дистрибутивность умножения
print("Дистрибутивность умножения")
A = np.matrix('1 2; 3 4')
B = np.matrix('5 6; 7 8')
C = np.matrix('2 4; 7 8')
L = A.dot(B + C)
R = A.dot(B) + A.dot(C)
print(L)
print(R)
print("\n")

#Умножение матриц в общем виде не коммутативно
print("Умножение матриц в общем виде не коммутативно")
A = np.matrix('1 2; 3 4')
B = np.matrix('5 6; 7 8')
L = A.dot(B)
R = B.dot(A)
print(L)
print(R)
print("\n")

#Произведение заданной матрицы на единичную
print("Произведение заданной матрицы на единичную")
A = np.matrix('1 2; 3 4')
E = np.matrix('1 0; 0 1')
L = E.dot(A)
R = A.dot(E)
print(L)
print(R)
print(A)
print("\n")

#Произведение заданной матрицы на нулевую матрицу
print("Произведение заданной матрицы на нулевую матрицу")
A = np.matrix('1 2; 3 4')
Z = np.matrix('0 0; 0 0')
L = Z.dot(A)
R = A.dot(Z)
print(L)
print(R)
print(Z)
print("\n")
```

Рисунок 16 – Код примера

```
Ассоциативность умножения
[[192 252]
 [436 572]]
[[192 252]
 [436 572]]


Дистрибутивность умножения
[[35 42]
 [77 94]]
[[35 42]
 [77 94]]


Умножение матриц в общем виде не коммутативно
[[19 22]
 [43 50]]
[[23 34]
 [31 46]]


Произведение заданной матрицы на единичную
[[1 2]
 [3 4]]
[[1 2]
 [3 4]]
[[1 2]
 [3 4]]


Произведение заданной матрицы на нулевую матрицу
[[0 0]
 [0 0]]
[[0 0]
 [0 0]]
[[0 0]
 [0 0]]
```

Рисунок 17 – Результат работы

```
import numpy as np
#Определитель матрицы
print("Определитель матрицы")
A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
print(A)
print(np.linalg.det(A))
print("\n")

#Определитель матрицы остается неизменным при ее транспонировании
print("Определитель матрицы остается неизменным при ее транспонировании")
A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
print(A)
print(A.T)
det_A = round(np.linalg.det(A), 3)
det_A_t = round(np.linalg.det(A.T), 3)
print(det_A)
print(det_A_t)
print("\n")

#Если у матрицы есть строка или столбец, состоящие из нулей,
#то определитель такой матрицы равен нулю
print("Если у матрицы есть строка или столбец, состоящие из нулей,")
print("то определитель такой матрицы равен нулю")
A = np.matrix('-4 -1 2; 0 0 0; 8 3 1')
print(A)
print(np.linalg.det(A))
print("\n")

#При перестановке строк матрицы знак ее определителя
#меняется на противоположный
print("При перестановке строк матрицы знак ее определителя")
print("меняется на противоположный")
A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
print(A)
B = np.matrix('10 4 -1; -4 -1 2; 8 3 1')
print(B)
print(np.linalg.det(A))
print(np.linalg.det(B))
print("\n")

#Если у матрицы есть две одинаковые строки, то ее определитель равен нулю
print("Если у матрицы есть две одинаковые строки, то ее определитель равен нулю")
A = np.matrix('-4 -1 2; -4 -1 2; 8 3 1')
print(A)
print(np.linalg.det(A))
print("\n")

#Если все элементы строки или столбца матрицы умножить на
#какое-то число, то и определитель будет умножен на это число
print("Если все элементы строки или столбца матрицы умножить на")
print("какое-то число, то и определитель будет умножен на это число")
A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
print(A)
k = 2
B = A.copy()
B[2, :] = k * B[2, :]
print(B)
det_A = round(np.linalg.det(A), 3)
det_B = round(np.linalg.det(B), 3)
print(det_A * k)
print(det_B)
print("\n")
```

Рисунок 18 – Код примера

```
Определитель матрицы
[[-4 -1  2]
 [10  4 -1]
 [ 8  3  1]]
-14.000000000000009


Определитель матрицы остается неизменным при ее транспонировании
[[-4 -1  2]
 [10  4 -1]
 [ 8  3  1]]
[[-4 10  8]
 [-1  4  3]
 [ 2 -1  1]]
-14.0
-14.0


Если у матрицы есть строка или столбец, состоящие из нулей,
то определитель такой матрицы равен нулю
[[-4 -1  2]
 [ 0  0  0]
 [ 8  3  1]]
0.0


При перестановке строк матрицы знак ее определителя
меняется на противоположный
[[-4 -1  2]
 [10  4 -1]
 [ 8  3  1]]
[[10  4 -1]
 [-4 -1  2]
 [ 8  3  1]]
-14.000000000000009
14.000000000000009


Если у матрицы есть две одинаковые строки, то ее определитель равен нулю
[[-4 -1  2]
 [-4 -1  2]
 [ 8  3  1]]
0.0


Если все элементы строки или столбца матрицы умножить на
какое-то число, то и определитель будет умножен на это число
[[-4 -1  2]
 [10  4 -1]
 [ 8  3  1]]
[[-4 -1  2]
 [10  4 -1]
 [16  6  2]]
-28.0
-28.0
```

Рисунок 19 – Результат работы

```
#Обратная матрица
import numpy as np
#inv()
print("inv()")
A = np.matrix('1 -3; 2 5')
A_inv = np.linalg.inv(A)
print(A_inv)
print("\n")

#Обратная матрица обратной матрицы есть исходная матрица:
print("Обратная матрица обратной матрицы есть исходная матрица:")
A = np.matrix('1. -3.; 2. 5.')
A_inv = np.linalg.inv(A)
A_inv_inv = np.linalg.inv(A_inv)
print(A)
print(A_inv_inv)
print("\n")

#Обратная матрица транспонированной матрицы равна транспонированной
#матрице от обратной матрицы:
print("Обратная матрица транспонированной матрицы равна транспонированной")
print("матрице от обратной матрицы:")
A = np.matrix('1. -3.; 2. 5.')
L = np.linalg.inv(A.T)
R = (np.linalg.inv(A)).T
print(L)
print(R)
print("\n")

#Обратная матрица произведения матриц равна произведению обратных матриц:
print("Обратная матрица произведения матриц равна произведению обратных матриц:")
A = np.matrix('1. -3.; 2. 5.')
B = np.matrix('7. 6.; 1. 8.')
L = np.linalg.inv(A.dot(B))
R = np.linalg.inv(B).dot(np.linalg.inv(A))
print(L)
print(R)
print("\n")

#Ранг матрицы
#matrix_rank():
print("matrix_rank()")
m_eye = np.eye(4)
print(m_eye)
rank = np.linalg.matrix_rank(m_eye)
print(rank)
print("\n")
m_eye[3][3] = 0
print(m_eye)
rank = np.linalg.matrix_rank(m_eye)
print(rank)
print("\n")
```

Рисунок 20 – Код примера

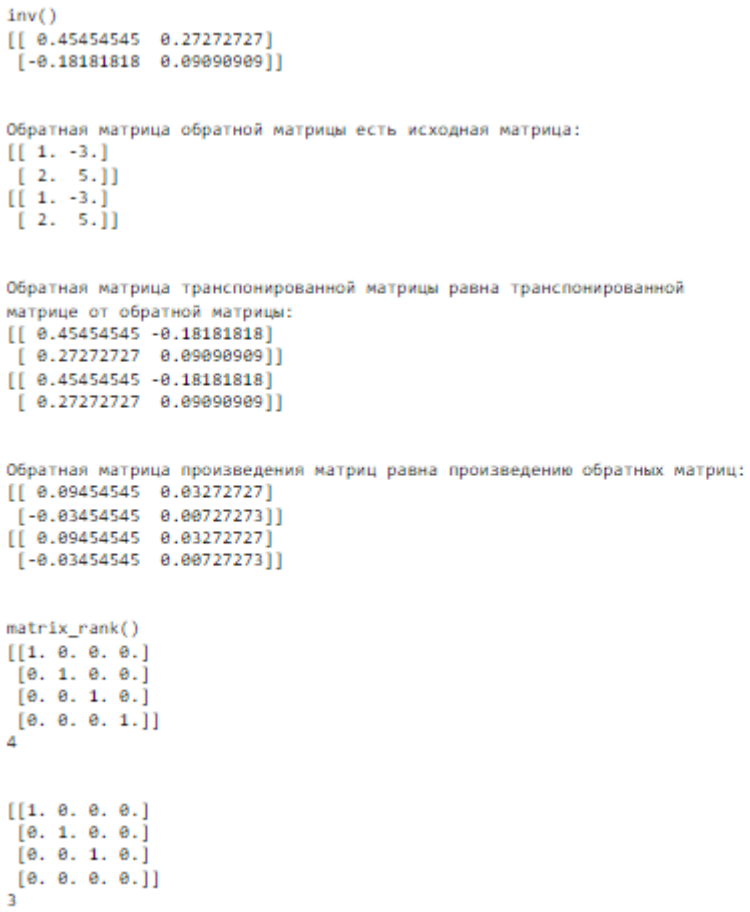

Рисунок 21 – Результат работы

Вывод: исследовал методы работы с матрицами и векторами с помощью библиотеки NumPy языка программирования Python.

Ответы на вопросы

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

Матрицы

Матрицей в математике называют объект, записываемый в виде прямоугольной таблицы, элементами которой являются числа (могут быть как действительные, так и комплексные).

Представленная выше матрица состоит из i-строк и j-столбцов. Каждый ее элемент имеет соответствующее позиционное обозначение, определяемое

номером строки и столбца на пересечении которых он расположен: aij- находится на i-ой строке и j-м столбце.

Важным элементом матрицы является главная диагональ, ее составляют элементы, у которых совпадают номера строк и столбцов.

Виды матриц и способы их создания в Python

Матрица в Python – это двумерный массив, поэтому задание матриц того или иного вида предполагает создание соответствующего массива. Для работы с массивами в Python используется тип данных список (англ. list). Но с точки зрения представления матриц и проведения вычислений с ними списки – не очень удобный инструмент, для этих целей хорошо подходит библиотека Numpy, ее мы и будем использовать в дальнейшей работе.Напомним, для того, чтобы использовать библиотеку Numpy ее нужно предварительно установить, после этого можно импортировать в свой проект. По установке Numpy можно подробно прочитать в разделе “Установка библиотеки Numpy” из введения. Для того чтобы импортировать данный модуль, добавьте в самое начало программы следующую строку

import numpy as np

Если после импорта не было сообщений об ошибке, то значит все прошло удачно и можно начинать работу. Numpy содержит большое количество функций для работы с матрицами, которые мы будем активно использовать. Обязательно убедитесь в том, что библиотека установлена и импортируется в проект без ошибок.

Вектором называется матрица, у которой есть только один столбец или одна строка.

Вектор-строка

Вектор-столбец

Квадратная матрица

Довольно часто, на практике, приходится работать с квадратными матрицами. Квадратной называется матрица, у которой количество столбцов и строк совпадает. В общем виде они выглядят так.

Диагональная матрица Особым видом квадратной матрицы является диагональная – это такая матрица, у которой все элементы, кроме тех, что расположены на главной диагонали, равны нулю.

Единичная матрица

Единичной матрицей называют такую квадратную матрицу, у которой элементы главной диагонали равны единицы, а все остальные нулю.

Нулевая матрица

У нулевой матрицы все элементы равны нулю.

2. Как выполняется транспонирование матриц?

Транспонирование матрицы – это процесс замены строк матрицы на ее столбцы, а столбцов соответственно на строки. Полученная в результате матрица называется транспонированной. Символ операции транспонирования – буква T.

Численный пример Для исходной матрицы:

Пример на Python Решим задачу транспонирования матрицы на Python.

Создадим матрицу A:

```
>>> A = np.matrix('1 2 3; 4 5 6')
>>> print(A)
[[1 2 3]
[4 5 6]]
```

Транспонируем матрицу с помощью метода transpose():

```
>>> A_t = A.transpose()
>>> print(A_t)
[[1 4]
[2 5]
[3 6]]
```

Существует сокращенный вариант получения транспонированной матрицы, он очень удобен в практическом применении:

```
>>> print(A.T)
[[1 4]
```

[2 5]

[3 6]]

3. Приведите свойства операции транспонирования матриц.

```
>>> A = np.matrix('1 2 3; 4 5 6')
>>> print(A)
[[1 2 3]
[4 5 6]]
>>> R = (A.T).T
>>> print(R)
[[1 2 3]
[4 5 6]]
```

Свойство 2. Транспонирование суммы матриц равно сумме транспонированных матриц:

```
>>> A = np.matrix('1 2 3; 4 5 6')
>>> B = np.matrix('7 8 9; 0 7 5')
>>> L = (A + B).T
>>> R = A.T + B.T
>>> print(L)
[[ 8 4]
[10 12]
[12 11]]
>>> print(R)
[[ 8 4]
[10 12]
[12 11]]
```

Свойство 3. Транспонирование произведения матриц равно произведению транспонированных матриц расставленных в обратном порядке:

```
>>> A = np.matrix('1 2; 3 4')
>>> B = np.matrix('5 6; 7 8')
```

```
>>> L = (A.dot(B)).T
>>> R = (B.T).dot(A.T)
>>> print(L)
[[19 43]
[22 50]]
>>> print(R)
[[19 43]
[22 50]]
```

Свойство 4. Транспонирование произведения матрицы на число равно произведению этого числа на транспонированную матрицу:

```
 >>> A = np.matrix('1 2 3; 4 5 6')
>>> k = 3
>>> L = (k * A).T
>>> R = k * (A.T)
>>> print(L)
[[ 3 12]
[ 6 15]
[ 9 18]]
\>>> print(R)
[[ 3 12]
[ 6 15]
[ 9 18]]
```

Свойство 5. Определители исходной и транспонированной матрицы совпадают:

```
 >>> A = np.matrix('1 2; 3 4')
>>> A_det = np.linalg.det(A)
>>> A_T_det = np.linalg.det(A.T)
>>> print(format(A_det, '.9g'))
-2
>>> print(format(A_T_det, '.9g'))
```

-2

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

Транспонирование матрицы с помощью NumPy transpose()

Первый метод, который мы разберем, — это использование библиотеки NumPy. NumPy в основном работает с массивами в Python, а для транспонирования мы можем вызвать метод transpose()

Метод 2. Использование метода numpy.transpose()

Мы также можем транспонировать матрицу в Python с помощью numpy.transpose(). При этом мы передаем матрицу в метод transpose() в качестве аргумента.

Метод 3. Транспонирование матрицы с использованием библиотеки SymPy

Применение библиотеки SymPy – это еще один подход к транспонированию матрицы. Эта библиотека использует символьную математику для решения алгебраических задач.

Метод 4. Транспонирование матрицы с использованием вложенного цикла

В Python матрицу можно транспонировать и без применения каких-либо библиотек. Для этого нам придется использовать вложенные циклы.

Мы создаем одну матрицу, а затем вторую (того же размера, что и первая) — для сохранения результатов после транспонирования. При этом важно отметить, что мы далеко не всегда знаем размерность исходной матрицы. Поэтому матрицу для результата мы создаем не напрямую, а используя размер исходной.

Метод 5. Использование генератора списка

Следующий метод, который мы разберем, — это использование генератора списка. Этот метод похож на предыдущий с использованием вложенных циклов, но он более «питонический». Можно сказать, что это

более продвинутый способ транспонирования матрицы в одной строке кода без использования библиотек.

Метод 6. Транспонирование матрицы с помощью pymatrix

Pymatrix – ещё одна облегченная библиотека для матричных операций в Python. Мы можем выполнить транспонирование и с её помощью

Метод 7. Использование метода zip

Zip – еще один метод транспонирования матрицы.

5. Какие существуют основные действия над матрицами?

Умножение матрицы на число

При умножении матрицы на число, все элементы матрицы умножаются на это число:

Сложение матриц

Складывать можно только матрицы одинаковой размерности — то есть матрицы, у которых совпадает количество столбцов и строк.

Умножение матриц

Умножение матриц это уже более сложная операция, по сравнению с рассмотренными выше.

Умножать можно только матрицы, отвечающие следующему требованию: количество столбцов первой матрицы должно быть равно числу строк второй матрицы.

6. Как осуществляется умножение матрицы на число?

При умножении матрицы на число, все элементы матрицы умножаются на это число:

```
>>> A = np.matrix('1 2 3; 4 5 6')
>>> C = 3 * A
>>> print(C)
[[ 3 6 9]
[12 15 18]]
```

7. Какие свойства операции умножения матрицы на число?

Свойство 1. Произведение единицы и любой заданной матрицы равно заданной матрице:

```
>>> A = np.matrix('1 2; 3 4')
>>> L = 1 * A
>>> R = A
>>> print(L)
[[1 2]
[3 4]]
>>> print(R)
[[1 2]
[3 4]]
```

Свойство 2. Произведение нуля и любой матрицы равно нулевой матрице, размерность которой равна исходной матрицы:

```
>>> A = np.matrix('1 2; 3 4')
>>> Z = np.matrix('0 0; 0 0')
>>> L = 0 * A
>>> R = Z
>>> print(L)
[[0 0]
[0 0]]
>>> print(R)
[[0 0]
[0 0]]
```

Свойство 3. Произведение матрицы на сумму чисел равно сумме произведений матрицы на каждое из этих чисел:

```
>>> A = np.matrix('1 2; 3 4')
>>> p = 2
>>> q = 3
>>> L = (p + q) * A
>>> R = p * A + q * A
```

```
>>> print(L
[[ 5 10]
[15 20]]
>>> print(R)
[[ 5 10]
[15 20]]
```

Свойство 4. Произведение матрицы на произведение двух чисел равно произведению второго числа и заданной матрицы, умноженному на первое число:

```
 >>> A = np.matrix('1 2; 3 4')
>>> p = 2
>>> q = 3
>>> L = (p * q) * A
>>> R = p * (q * A)
>>> print(L)
[[ 6 12]
[18 24]]
>>> print(R)
[[ 6 12]
[18 24]]
```

Свойство 5. Произведение суммы матриц на число равно сумме произведений этих матриц на заданное число:

```
 >>> A = np.matrix('1 2; 3 4')
>>> B = np.matrix('5 6; 7 8')
>>> k = 3
>>> L = k * (A + B)
>>> R = k * A + k * B
>>> print(L)
[[18 24]
[30 36]]
```

>>> print(R)

[[18 24]

[30 36]]

8. Как осуществляется операции сложения и вычитания матриц?

Сложение происходит поэлеметно

[[6.0 8.0]

[10.0 12.0]]

print(x + y)

print()

print(np.add(x, y))

print('С числом')

print(x + 1)

print('С массивом другой размерности')

print(x + arr)

[[6. 8.]

 [10. 12.]]

[[6. 8.]

 [10. 12.]]

С числом

[[2. 3.]

 [4. 5.]]

С массивом другой размерности

[[2. 4.]

 [4. 6.]]

 Вычитание

print(x - y)

print(np.subtract(x, y))

[[-4. -4.]

 [-4. -4.]]

[[-4. -4.]

[-4. -4.]]

9. Каковы свойства операций сложения и вычитания матриц?

Свойство 1. Коммутативность сложения. От перестановки матриц их сумма не изменяется:

```
>>> A = np.matrix('1 2; 3 4')
>>> B = np.matrix('5 6; 7 8')
>>> L = A + B
>>> R = B + A
>>> print(L)
[[ 6 8]
[10 12]]
>>> print(R)
[[ 6 8]
[10 12]]
```

Свойство 2. Ассоциативность сложения. Результат сложения трех и более матриц не зависит от порядка, в котором эта операция будет выполняться:

```
>>> A = np.matrix('1 2; 3 4')
>>> B = np.matrix('5 6; 7 8')
>>> C = np.matrix('1 7; 9 3')
>>> L = A + (B + C)
>>> R = (A + B) + C
>>> print(L)
[[ 7 15]
[19 15]]
>>> print(R)
[[ 7 15]
[19 15]]
```

Свойство 3. Для любой матрицы существует противоположная ей , такая, что их сумма является нулевой матрицей :

```
>>> A = np.matrix('1 2; 3 4')
>>> Z = np.matrix('0 0; 0 0')
>>> L = A + (-1)*A
>>> print(L)
[[0 0]
[0 0]]
>>> print(Z)
[[0 0]
[0 0]]
```

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

Рассмотрим сложение и вычитание массивов. Вначале выполним поэлементное сложение.

```
a + b
array([[ 6,  8, 10],
       [12, 14, 16]])
```

Мы также можем выполнить сложение двух элементов внешнего измерения одного массива.

```
# для этого возьмем элементы по индексу
a[0] + a[1]
array([3, 5, 7])
```

Аналогично выполняется вычитание массивов.

```
b - a
array([[6, 6, 6],
       [6, 6, 6]])
```

11. Как осуществляется операция умножения матриц?

Пример:

Каждый элемент cij новой матрицы является суммой произведений элементов i-ой строки первой матрицы и j-го столбца второй матрицы. Математически это записывается так:

Решим задачу умножения матриц на языке Python. Для этого будем использовать функцию dot() из библиотеки Numpy:

```
>>> A = np.matrix('1 2 3; 4 5 6')
>>> B = np.matrix('7 8; 9 1; 2 3')
>>> C = A.dot(B)
>>> print(C)
[[31 19]
[85 55]]
```

12. Каковы свойства операции умножения матриц?

Свойство 1. Ассоциативность умножения. Результат умножения матриц не зависит от порядка, в котором будет выполняться эта операция:

```
>>> A = np.matrix('1 2; 3 4')
>>> B = np.matrix('5 6; 7 8')
>>> C = np.matrix('2 4; 7 8')
>>> L = A.dot(B.dot(C))
>>> R = (A.dot(B)).dot(C)
>>> print(L)
[[192 252]
[436 572]]
>>> print(R)
[[192 252]
[436 572]]
```

Свойство 2. Дистрибутивность умножения. Произведение матрицы на сумму матриц равно сумме произведений матриц:

```
>>> A = np.matrix('1 2; 3 4')
>>> B = np.matrix('5 6; 7 8')
>>> C = np.matrix('2 4; 7 8')
>>> L = A.dot(B + C)
>>> R = A.dot(B) + A.dot(C)
>>> print(L)
```

```
[[35 42]
[77 94]]
>>> print(R)
[[35 42]
[77 94]]
```

Свойство 3. Умножение матриц в общем виде не коммутативно. Это означает, что для матриц не выполняется правило независимости произведения от перестановки множителей:

```
>>> A = np.matrix('1 2; 3 4')
>>> B = np.matrix('5 6; 7 8')
>>> L = A.dot(B)
>>> R = B.dot(A)
>>> print(L)
[[19 22]
[43 50]]
>>> print(R)
[[23 34]
[31 46]]
```

Свойство 4. Произведение заданной матрицы на единичную равно исходной матрице:

```
>>> A = np.matrix('1 2; 3 4')
>>> E = np.matrix('1 0; 0 1')
>>> L = E.dot(A)
>>> R = A.dot(E)
>>> print(L)
[[1 2]
[3 4]]
>>> print(R)
[[1 2]
[3 4]]
```

```
>>> print(A)
[[1 2]
[3 4]]
```

Свойство 5. Произведение заданной матрицы на нулевую матрицу равно нулевой матрице:

```
 >>> A = np.matrix('1 2; 3 4')
>>> Z = np.matrix('0 0; 0 0')
>>> L = Z.dot(A)
>>> R = A.dot(Z)
>>> print(L)
[[0 0]
[0 0]]
>>> print(R)
[[0 0]
[0 0]]
>>> print(Z)
[[0 0]
[0 0]]
```

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

При необходимости выполнения операций по правилам линейной алгебры можно воспользоваться методом dot(A, B). В зависимости от вида операндов, функция выполнит:

- если аргументы скаляры (числа), то выполнится умножение;
- если аргументы вектор (одномерный массив) и скаляр, то выполнится умножение массива на число;
- если аргументы вектора, то выполнится скалярное умножение (сумма поэлементных произведений);
- если аргументы тензор (многомерный массив) и скаляр, то выполнится умножение вектора на число;

- если аргументы тензора, то выполнится произведение тензоров по последней оси первого аргумента и предпоследней — второго;
- если аргументы матрицы, то выполнится произведение матриц (это частный случай произведения тензоров);
- если аргументы матрица и вектор, то выполнится произведение матрицы и вектора (это тоже частный случай произведения тензоров).

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

Определитель матрицы размера (n-го порядка) является одной из ее численных характеристик. Определитель матрицы A обозначается как |A| или det(A), его также называют детерминантом.

Рассмотрим квадратную матрицу 2×2 в общем виде:

Определитель такой матрицы вычисляется следующим образом:

Минор элемента определителя – это определитель, полученный из данного, путем вычеркивания всех элементов строки и столбца, на пересечении которых стоит данный элемент. Для матрицы 3×3 следующего вида:

Минор M23 будет выглядеть так:

Введем еще одно понятие – алгебраическое дополнение элемента определителя – это минор этого элемента, взятый со знаком плюс или минус:

В общем виде вычислить определитель матрицы можно через разложение определителя по элементам строки или столбца. Суть в том, что определитель равен сумме произведений элементов любой строки или столбца на их алгебраические дополнения. Для матрицы 3×3 это правило будет выполняться следующим образом:

Это правило распространяется на матрицы любой размерности.

Свойства определителя матрицы.

Свойство 1. Определитель матрицы остается неизменным при ее транспонировании:

```
>>> A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
>>> print(A)
[[-4 -1 2]
[10 4 -1]
[ 8 3 1]]
>>> print(A.T)
[[-4 10 8]
[-1 4 3]
[ 2 -1 1]]
>>> det_A = round(np.linalg.det(A), 3)
>>> det_A_t = round(np.linalg.det(A.T), 3)
>>> print(det_A)
-14.0
>>> print(det_A_t)
-14.0
```

Свойство 2. Если у матрицы есть строка или столбец, состоящие из нулей, то определитель такой матрицы равен нулю:

```
>>> A = np.matrix('-4 -1 2; 0 0 0; 8 3 1')
>>> print(A)
[[-4 -1 2]
[ 0 0 0]
[ 8 3 1]]
>>> np.linalg.det(A)
0.0
```

Свойство 3. При перестановке строк матрицы знак ее определителя меняется на противоположный:

```
>>> A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
>>> print(A)
[[-4 -1 2]
```

```
[10 4 -1]
[ 8 3 1]]
>>> B = np.matrix('10 4 -1; -4 -1 2; 8 3 1')
>>> print(B)
[[10 4 -1]
[-4 -1 2]
[ 8 3 1]]
>>> round(np.linalg.det(A), 3)
-14.0
>>> round(np.linalg.det(B), 3)
14.0
```

Свойство 4. Если у матрицы есть две одинаковые строки, то ее определитель равен нулю:

```
>>> A = np.matrix('-4 -1 2; -4 -1 2; 8 3 1')
>>> print(A)
[[-4 -1 2]
[-4 -1 2]
[ 8 3 1]]
>>> np.linalg.det(A)
0.0
```

Свойство 5. Если все элементы строки или столбца матрицы умножить на какое-то число, то и определитель будет умножен на это число:

Свойство 6. Если все элементы строки или столбца можно представить как сумму двух слагаемых, то определитель такой матрицы равен сумме определителей двух соответствующих матриц:

Свойство 7. Если к элементам одной строки прибавить элементы другой строки, умноженные на одно и тоже число, то определитель матрицы не изменится:

Свойство 8. Если строка или столбец матрицы является линейной комбинацией других строк (столбцов), то определитель такой матрицы равен нулю:

```
>>> A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
>>> print(A)
[[-4 -1 2]
[10 4 -1]
[ 8 3 1]]
>>> k = 2
>>> A[1, :] = A[0, :] + k * A[2, :]
>>> round(np.linalg.det(A), 3)
0.0
```

Свойство 9. Если матрица содержит пропорциональные строки, то ее определитель равен нулю:

```
>>> A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
>>> print(A)
[[-4 -1 2]
[10 4 -1]
[ 8 3 1]]
>>> k = 2
>>> A[1, :] = k * A[0, :]
>>> print(A)
[[-4 -1 2]
[-8 -2 4]
[ 8 3 1]]
>>> round(np.linalg.det(A), 3)
0.0
```

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

На Python определитель посчитать очень просто. Создадим матрицу A размера 3×3 из приведенного выше численного примера:

```
>>> A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
>>> print(A)
[[-4 -1 2]
[10 4 -1]
[ 8 3 1]]
```

Для вычисления определителя этой матрицы воспользуемся функцией det() из пакета linalg.

```
>>> np.linalg.det(A)
-14.000000000000009
```

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

Обратной матрицей матрицы называют матрицу, удовлетворяющую следующему равенству:

Для того, чтобы у квадратной матрицы A была обратная матрица необходимо и достаточно чтобы определитель |A| был не равен нулю. Введем понятие союзной матрицы. Союзная матрица строится на базе исходной A путем замены всех элементов матрицы A на их алгебраические дополнения.

Исходная матрица:

Союзная ей матрица A :

Транспонируя матрицу , мы получим так называемую присоединенную матрицу :

Теперь, зная как вычислять определитель и присоединенную матрицу, мы можем определить матрицу, обратную матрице:

17. Каковы свойства обратной матрицы?

Свойство 1. Обратная матрица обратной матрицы есть исходная матрица:

```
>>> A = np.matrix('1. -3.; 2. 5.')
```

```
>>> A_inv = np.linalg.inv(A)
>>> A_inv_inv = np.linalg.inv(A_inv)
>>> print(A)
[[1. -3.]
[2. 5.]]
>>> print(A_inv_inv)
[[1. -3.]
[2. 5.]]
```

Свойство 2. Обратная матрица транспонированной матрицы равна транспонированной матрице от обратной матрицы:

```
>>> A = np.matrix('1. -3.; 2. 5.')
>>> L = np.linalg.inv(A.T)
>>> R = (np.linalg.inv(A)).T
>>> print(L)
[[ 0.45454545 -0.18181818]
[ 0.27272727 0.09090909]]
>>> print(R)
[[ 0.45454545 -0.18181818]
[ 0.27272727 0.09090909]]
```

Свойство 3. Обратная матрица произведения матриц равна произведению обратных матриц:

```
>>> A = np.matrix('1. -3.; 2. 5.')
>>> B = np.matrix('7. 6.; 1. 8.')
>>> L = np.linalg.inv(A.dot(B))
>>> R = np.linalg.inv(B).dot(np.linalg.inv(A))
>>> print(L)
[[ 0.09454545 0.03272727]
[-0.03454545 0.00727273]]
>>> print(R)
[[ 0.09454545 0.03272727]
```

[-0.03454545 0.00727273]]

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

Решим задачу определения обратной матрицы на Python. Для получения обратной матрицы будем использовать функцию *inv()*:

>>> A = np.matrix('1 -3; 2 5')

>>> A_inv = np.linalg.inv(A)

>>> print(A_inv)

[[0.45454545 0.27272727]

[-0.18181818 0.09090909]]

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

Метод Крамера основан на использовании определителей в решении систем линейных уравнений. Это значительно ускоряет процесс решения.

Метод Крамера может быть использован в решении системы стольких линейных уравнений, сколько в каждом уравнении неизвестных. Если определитель системы не равен нулю, то метод Крамера может быть использован в решении, если же равен нулю, то не может. Кроме того, метод Крамера может быть использован в решении систем линейных уравнений, имеющих единственное решение.

Определение. Определитель, составленный из коэффициентов при неизвестных, называется определителем системы и обозначается (дельта).

Определители

получаются путём замены коэффициентов при соответствующих неизвестных свободными членами

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

import numpy # импортируем библиотеку

M1 = numpy.array([[2., 5.], [1., -10.]]) # Матрица (левая часть системы)

v1 = numpy.array([1., 3.]) # Вектор (правая часть системы)

#Запишем все числа с точкой, т.к. иначе в Python 2 они будут участвовать в целочисленных операциях (остатки от деления будут отбрасываться)

In

numpy.linalg.solve(M1, v1)

Out

array([1. , -0.2])