# **Лабораторная работа №25 (2 часа)**

**Тема работы: «Разработка, отладка и испытание программ**

**управления массивами с помощью универсальных функций»**

**1 Цель работы**

Закрепить навык создания и работы с массивами больших данных на базе модуля NumPy.

**2 Задание**

Заполните массив случайными числами, перемешайте массив, возвести массив в степень.

**3 Оснащение работы**

Задание по варианту, ЭВМ, среда разработки **Python 3.7, IDLE**.

**4 Основные теоретические сведения**

**numpy.random**

Для создания массивов со случайными элементами служит модуль numpy.random.

>>>

>>> import numpy as np # Импортировать numpy и писать np.random

>>> np.random

<module 'numpy.random' from '/usr/local/lib/python3.4/dist-packages/numpy/random/\_\_init\_\_.py'>

>>> import numpy.random as rand # Можно и присвоить отдельное имя. Вопрос вкуса

>>> rand

<module 'numpy.random' from '/usr/local/lib/python3.4/dist-packages/numpy/random/\_\_init\_\_.py'>

**Создание массивов**

Самый простой способ задать массив со случайными элементами – использовать функцию sample (или random, или random\_sample, или ranf – это всё одна и та же функция).

>>>

>>> np.random.sample()

0.6336371838734877

>>> np.random.sample(3)

array([ 0.53478558, 0.1441317 , 0.15711313])

>>> np.random.sample((2, 3))

array([[ 0.12915769, 0.09448946, 0.58778985],

[ 0.45488207, 0.19335243, 0.22129977]])

Без аргументов возвращает просто число в промежутке [0, 1), с одним целым числом – одномерный массив, с кортежем – массив с размерами, указанными в кортеже (все числа – из промежутка [0, 1)).

С помощью функции randint или random\_integers можно создать массив из целых чисел. Аргументы: low, high, size: от какого, до какого числа (randint не включает в себя это число, а random\_integers включает), и size – размеры массива.

>>>

>>> np.random.randint(0, 3, 10)

array([0, 2, 0, 1, 1, 0, 2, 2, 2, 0])

>>> np.random.random\_integers(0, 3, 10)

array([2, 2, 3, 3, 1, 1, 0, 2, 3, 2])

>>> np.random.randint(0, 3, (2, 10))

array([[0, 1, 2, 0, 0, 0, 1, 1, 1, 2],

[0, 0, 2, 2, 2, 0, 1, 2, 2, 1]])

Также можно генерировать числа согласно различным [распределениям](https://docs.scipy.org/doc/numpy/reference/routines.random.html%23distributions) (Гаусса, Парето и другие). Чаще всего нужно равномерное распределение, которое можно получить с помощь функции uniform.

>>>

>>> np.random.uniform(2, 8, (2, 10))

array([[ 3.1517914 , 3.10313483, 2.84007134, 3.21556436, 4.64531786,

2.99232714, 7.03064897, 4.38691765, 5.27488548, 2.63472454],

[ 6.39470358, 5.63084131, 4.69996748, 7.07260546, 7.44340813,

4.10722203, 7.52956646, 4.8596943 , 3.97923973, 5.64505363]])

**Выбор и перемешивание**

Перемешать NumPy массив можно с помощью функции shuffle:

>>>

>>> a = np.arange(10)

>>> a

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> np.random.shuffle(a)

>>> a

array([2, 8, 7, 3, 5, 0, 4, 9, 1, 6])

Также можно перемешать массив с помощью функции permutation (она, в отличие от shuffle, возвращает перемешанный массив). Также она, вызванная с одним аргументом (целым числом), возвращает перемешанную последовательность от 0 до N.

>>>

>>> np.random.permutation(10)

array([1, 2, 3, 8, 7, 9, 4, 6, 5, 0])

Сделать случайную выборку из массива можно с помощью функции choice. Про неё стоит рассказать подробнее.

numpy.random.choice(a, size=None, replace=True, p=None)

a: одномерный массив или число. Если массив, будет производиться выборка из него. Если число, то выборка будет производиться из np.arange(a).

size: размерности массива. Если None, возвращается одно значение.

replace: если True, то одно значение может выбираться более одного раза.

p: вероятности. Это означает, что элементы можно выбирать с неравными вероятностями. Если не заданы, используется равномерное распределение.

>>>

>>> a = np.arange(10)

>>> a

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> np.random.choice(a, 10, p=[0.5, 0.25, 0.25, 0, 0, 0, 0, 0, 0, 0])

array([0, 0, 0, 0, 1, 2, 0, 0, 1, 1])

**Инициализация генератора случайных чисел**

seed(число) – инициализация генератора.

>>>

>>> np.random.seed(1000)

>>> np.random.random(10)

array([ 0.65358959, 0.11500694, 0.95028286, 0.4821914 , 0.87247454,

0.21233268, 0.04070962, 0.39719446, 0.2331322 , 0.84174072])

>>> np.random.seed(1000)

>>> np.random.random(10)

array([ 0.65358959, 0.11500694, 0.95028286, 0.4821914 , 0.87247454,

0.21233268, 0.04070962, 0.39719446, 0.2331322 , 0.84174072])

get\_state и set\_state – возвращают и устанавливают состояние генератора.

>>>

>>> np.random.seed(1000)

>>> state = np.random.get\_state()

>>> np.random.random(10)

array([ 0.65358959, 0.11500694, 0.95028286, 0.4821914 , 0.87247454,

0.21233268, 0.04070962, 0.39719446, 0.2331322 , 0.84174072])

>>> np.random.set\_state(state)

>>> np.random.random(10)

array([ 0.65358959, 0.11500694, 0.95028286, 0.4821914 , 0.87247454,

0.21233268, 0.04070962, 0.39719446, 0.2331322 , 0

**Возведение в степень**

linalg.matrix\_power(M, n) - возводит матрицу в степень n.

**Разложения**

linalg.cholesky(a) - разложение Холецкого.

linalg.qr(a[, mode]) - QR разложение.

linalg.svd(a[, full\_matrices, compute\_uv]) - сингулярное разложение.

**Некоторые характеристики матриц**

linalg.eig(a) - собственные значения и собственные векторы.

linalg.norm(x[, ord, axis]) - норма вектора или оператора.

linalg.cond(x[, p]) - число обусловленности.

linalg.det(a) - определитель.

linalg.slogdet(a) - знак и логарифм определителя (для избежания переполнения, если сам определитель очень маленький).

**Системы уравнений**

linalg.solve(a, b) - решает систему линейных уравнений Ax = b.

linalg.tensorsolve(a, b[, axes]) - решает тензорную систему линейных уравнений Ax = b.

linalg.lstsq(a, b[, rcond]) - метод наименьших квадратов.

linalg.inv(a) - обратная матрица.

**Замечания:**

linalg.LinAlgError – исключение, вызываемое данными функциями в случае неудачи (например, при попытке взять обратную матрицу от вырожденной).

Подробная документация, как всегда, на английском: <https://docs.scipy.org/doc/numpy/reference/routines.linalg.html>

Массивы большей размерности в большинстве функций linalg интерпретируются как набор из нескольких массивов нужной размерности. Таким образом, можно одним вызовом функции проделывать операции над несколькими объектами.

>>>

>>> a = np.arange(18).reshape((2,3,3))

>>> a

array([[[ 0, 1, 2],

[ 3, 4, 5],

[ 6, 7, 8]],

[[ 9, 10, 11],

[12, 13, 14],

[15, 16, 17]]])

>>> np.linalg.det(a)

array([ 0., 0.])

**5 Порядок выполнения работы**

1. Выделить ключевые моменты задачи.

2. Построить алгоритм решения задачи.

3. Запрограммировать полученный алгоритм.

4. Провести тестирование полученной программы.

**6 Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7 Контрольные вопросы и задания**

1. Как произвести сортировку массива средствами модуля NymPy?
2. Как генерировать случайные числа?
3. Для чего делают разложение данных в массиве?
4. Опишите характеристики матриц.
5. Опишите какие функции можно использовать для решения системы уравнений.

**8 Рекомендуемая литература**

**Плас, Дж. В.** Python для сложных задач. Наука о данных и машинное обучение / Дж.В. Плас. – СПб: Питер, 2018.

**Прохоренок, Н.А.** Python 3. Самое необходимое / Н.А Прохоренок, В.А. Дронов – СПб.: БВХ-Петербург, 2016.