Южный федеральный университет
Институт математики, механики и компьютерных наук им. И. И. Воровича
направление подготовки «Прикладная математика и информатика»

Математические библиотеки Python. Библиотека NumPy[1].

Библиотека NumPy обеспечивает работу с большими массивами и матрицами, состоящими из элементов одного типа (численного или логического). Основным пакетом, содержащим математические функции для работы с матрицами и векторами, является пакет linalg модуля Numpy.
Существует несколько способов определить массив в NumPy. Можно определять массив элементами в явном виде, задавать его пустым, случайным, нулевым и т. п.

```
import numpy as np
a = np.array([])                      # определение пустого массива
b = np.array([0, -2, 10, 15])         # задание элементов массива в явном виде
B = np.array([[1,2,3], [3,4,5]])      # задание элементов матрицы в явном виде
c = np.zeros(5)                       # создание массива из нулей
C = np.zeros((3,3))                   # создание матрицы из нулей
D = np.array(np.mat("0 15; 10 2"))    # создание матрицы
e = np.ones(5)                        # создание массива из единиц
d = np.empty(3, dtype = np.float32)   # создание массива из "случайных" чисел
f = np.full(10, 4.6)                  # создание массива из одинаковых элементов,
                              # равных 4.6, в количестве 10 штук
E = np.eye(4)                         # создание единичной матрицы ("1" на диагонали)
E1 =np.eye(4,-1)                      # создание единичной матрицы
                              # ("1" на параллельной диагонали на 1 шаг вниз от главной)
D1 = np.diag([0, 1, -3])      # создание диагональной матрицы
```

Для заполнения массива случайными числами в NumPy есть собственный пакет random. Его использование отличается от стандартного модуля random.

```
a = np.random.sample(5)      # массив пяти случайных чисел из промежутка [0,1)
A = np.random.sample((5,5)) # матрица 5 на 5 случайных чисел из промежутка [0,1)
b = np.random.random(4)      # массив четырёх случайных чисел из промежутка [0,1)
```

Заполнять элементы многомерного массива можно согласно определенному правилу относительно его индексов, используя функцию fromfunction(func, size). В качестве параметра func можно указать подпрограмму или лямбда-функцию.

```
def fun(i,j):
    return i*i + j*j
g = np.fromfunction(fun, (4,4), dtype = int)

f = np.fromfunction(lambda i, j: i + j, (3, 3), dtype = int)
```

Тип элементов массива можно определить во время создания массива через свойство dtype . По умолчанию тип чисел - float64. Это 1 бит знака, 11 бит экспоненты, 52 бита мантисы (числа размером 8 байт). Другие возможные типы элементов: bool, int8, int16, int32, int64, float16, float32, complex64, complex128.
Используя свойство dtype, можно изменять тип данных массива по ходу работы программы.

```
print(np.array(b, dtype = np.float32))  # [ 0. -2. 10. 15.]
print(np.array(b, dtype = bool))        # [False  True  True  True]
print(np.array(b, dtype = complex))     # [ 0.+0.j -2.+0.j 10.+0.j 15.+0.j]
```

[1]Разработано А.М. Филимоновой (кафедра ВМиМФ мехмата ЮФУ)

Если нужно заполнить массив элементами одной последовательности (с постоянным шагом), то следует воспользоваться функциями arange() или linspace(). **Важно!** Верхний предел в arange(), как и в range(), не учитывается. В linspace() это устанавливается через свойство endpoint.

```
p1 = np.arange(0, 10, 1)     # [0 1 2 3 4 5 6 7 8 9]
p2 = np.arange(-1, 5, 0.52) # [-1. -0.48 0.04 0.56 1.08 1.6 2.12 2.64 3.16 3.68 4.2 4.72]
p3 = np.linspace(0, 100, 11, dtype = int) # [0 10 20 30 40 50 60 70 80 90 100]
p4 = np.linspace(-1, 3, 6)                # [-1. -0.2 0.6 1.4 2.2 3.]


C = np.random.randint(10, size = (3,6)) # матрица случайных чисел из промежутка [0,10)
print(C.shape)              # получение размеров матрицы C
C.shape = (2, 9)            # изменение размеров матрицы C
C1 = C.reshape(6,3)         # изменение размеров матрицы C
D = np.random.randint(2, size = C.shape) - 10
DT = D.transpose()          # транспонирование матрицы D

print(C+D)                  # поэлементное сложение элементов матриц
print(C-D)                  # поэлементное вычитание элементов матриц
print(C*D)                  # поэлементное умножение элементов матриц
print(C/D)                  # поэлементное деление элементов матриц
print(-10+D)                # поэлементное прибавление числа к элементам матрицы
print(C*(-1))               # поэлементное умножение элементов матрицы на (-1)
```

Основные математические функции работы с массивами модуля NumPy

np.allclose(a,c)	проверка, что массивы a и c состоят из одних и тех же элементов
np.max(a)	нахождение максимального элемента в массиве a
np.min(a)	нахождение минимального элемента в массиве a
np.mean(a)	нахождение среднего арифметического всех элементов массива a
np.var(a)	нахождение дисперсии элементов массива a[2]
np.std(a)	нахождение стандартного (средне-квадратичного) отклонения[3]
B.T или B.transpose()	транспонирование матрицы B
B.diagonal()	получение диагонали матрицы B
B.trace()	след матрицы B
np.dot(a,c)	скалярное произведение векторов a и c
np.cross(a,c)	векторное произведение векторов a и c
np.add(B,D) или B + D	поэлементное сложение двух матриц
np.subtract(B,D) или B - D	поэлементное вычитание двух матриц
np.dot(B,D) или B @ D	произведение двух матриц
np.linalg.norm(a)	норма вектора a
np.linalg.matrix_rank(B)	ранг матрицы B
np.linalg.matrix_det(B) или np.linalg.det(B)	определитель матрицы B
np.linalg.inv(B)	обратная матрица B^{-1}
np.linalg.solve(A,b)	решение СЛАУ вида $Ax = b$

К массивам и матрицам также, как и к обычным спискам и кортежам, можно обращаться напрямую по индексу p1[3] и брать срезы. Методы же добавления и изменения количества элементов отличаются.

```
# Объединение массивов в один вдоль горизонтальной оси (добавление "снизу")
print(np.vstack((C,D)))
print(np.row_stack((C,D)))
```

[2] *Дисперсия* – отклонение величины каждого x элемента массива a относительно среднего значения. $var = mean(abs(x - xmean)^2)$

[3] *Стандартное (средне-квадратичное отклонение)* – $std = sqrt(mean(abs(x - xmean)^2))$

```
# Объединение массивов в один вдоль вертикальной оси (добавление "сбоку")
print(np.hstack((C,D)))
print(np.column_stack((C,D)))

# Разделение одного массива на несколько
x = np.arange(8.0)
x1 = np.split(x, 4)           # разделение массива на четыре массива
x2 = np.split(x, [3, 5, 10])  # разделение массива на следующие массивы:
#[array([0., 1., 2.]), array([3., 4.]), array([5., 6., 7.]), array([], dtype=float64)]

X = np.arange(16.0).reshape(4, 4)
X1 = np.hsplit(X, 2)          # разделение матрицы на две вдоль горизонтальной оси
X2 = np.vsplit(X, 4)          # разделение матрицы на четыре вдоль вертикальной оси
```