numpy

Хашин С.И.
http://math.ivanovo.ac.ru/dalgebra/Khashin/index.html

Ивановский университет

.

numpy

.

Иваново-2023

План

Типы данных

bool	True или False
int8	Байт (от — 128 до 127)
int16	Целое число (от -32768 до 32767)
int, int32	Целое число (от -2147483648 до 2147483647)
int64	Целое от $-9 \cdot 10^{18}$ до $+9 \cdot 10^{18}$
uint8	Целое без знака (от 0 до 255)
uint16	Целое без знака (от 0 до 65535)
uint32	Целое без знака (от 0 до 4294967295)
uint64	Целое без знака (от 0 до $18 \cdot 10^{18}$
float32	4-байтовое плавающее число (до 10^{29})
float64	8-байтовое плавающее число (до 10^{309})
float, double	синонимы float64

байты (int8, uint8)

binary	uint8	int8
0000 0000	0	0
0000 0001	1	1
...		
0111 1111	127	127
1000 0000	128	-128
1000 0001	129	-127
...		
1111 1110	254	-2
1111 1111	255	-1
0000 0000	0	0

Основы

```
import numpy as np
np.set_printoptions(precision=4, linewidth = 120,
                    suppress=True)
a = np.array([1, 2, 3])                    # вектор
b = np.array([[1.5, 2, 3], [4, 5, 6]]) # матрица 2*3
ed = np.eye(8)          # единичная матрица 8*8
Z = np.zeros(10)        # одномерный массив из нулей
Z = np.ones(10)         # одномерный массив из единиц
Z = np.full(10, 2.5)  # вектор длины 10, заполненный числом
Z = np.arange(10,50)  # вектор  со значениями от 10 до 49

# матрица 3x4 со значениями от 0 до 11:
Z = np.arange(12).reshape(3,-1)
```

Основы

```
X = np.array((7,1,3,4,3,4))
print(X, '=X')              # [7 1 3 4 3 4] =X
print(2*X-3, '=2*X-3')      # [11 -1  3  5  3  5] =2*X-3
print(X*X, '=X*X')          # [49  1  9 16  9 16] =X*X
print(np.sum(X*X), '=sum(X*X)') # 100 =sum(X*X)
```

Задание. Построить вектор x длины 30 такой, что $x_k = 3k^2 - 2k - 7$.

Вырезки

```
A = np.arange(12)  # 0,1,..., 11
Z = A[2:7]         # 2,3,...,6
Z = A[:7]          # 0,1,...,6
Z = A[-4:]         # 8,9,10,11
Z = A[-4:-1]       # 8,9,10
Z = A[1::3]        # 1,4,7,10
Z = A[1:10:3]      # 1,4,7
Z = np.ones((10,10))
Z[1:-1,1:-1] = 0   # заполнить внутренность нулями

Z = np.zeros((8,8), dtype=int) # нулевая матрица 8*8
Z[1::2,::2] = 1    # 0,1 в
Z[::2,1::2] = 1    # шахматном порядке

# Развернуть вектор (первый становится последним:)
Z = Z[::-1]
```

Задание

1. Написать функцию, находящую сумму «внутренних» элементов матрицы, то есть всех, кроме находящихся на одном из четырех краёв матрицы.
2. Написать функцию f(A) находящую для матрицы A размера 8*8 сумму чисел, стоящих на белых и на чёрных клетках. То есть функция должна возвращать пару чисел.
3. Написать функцию f(A) находящую для матрицы A номер строки, сумма квадратов элементов которой наибольшая.
4. Найдите все симметричные матрицы размера 2*2 с натуральными коэффициентами меньшими 30 и определителем 1.

Равномерное заполнение

```
Z=np.linspace(0, 2, 9)
print(Z)
# 9 чисел от 0 до 2 включительно:
# [0., .25, .5, .75, 1., 1.25, 1.5, 1.75, 2.]

Z = np.linspace(0,1,12)[1:-1]
# вектор размера 10 со значениями от 0 до 1,
# не включая ни то, ни другое
# 0.090909 0.181818 0.272727 ... 0.818182 0.909091]
```

Вырезки

В NumPy также реализована возможность доступа ко множеству элементов массива через булев индексный массив. Индексный массив должен совпадать по форме с индексируемым.

```
A = np.array([[1, 2, 3], [4, 5, 6]])
I = np.array([[False, False, True],
              [ True, False, True]])
print(A[I])
> [3, 4, 6]
```

Такая конструкция возвращает плоский массив! Но:

```
A[I] = 0
print(A)
>[[1 2 0]
> [0 5 0]]
```

Основы

```
A = np.array([
    [1,2,3, 4],
    [1,1,6, 1],
    [5,4,3,77]])
#поменять знак у элементов, значения которых между 3 и 8
A[(3 < A) & (A <= 8)] *= -1; print(A)
> [[ 1  2  3 -4]
>  [ 1  1 -6  1]
>  [-5 -4  3 77]]
print(A.min(), A.max())   # минимум и максимум
> -6 77
print(A.argmin(), A.argmax()) # ЛИНЕЙНЫЕ позиции
> 6 11
print(np.unravel_index(A.argmax(), A.shape))
> (2, 3)
```

Без nan

```
a = np.random.randint(0,10, size=(4,2))*1.
a[1,1] = np.nan
print(a, '=a\n')
b1 = np.nanmax(a, axis=1)
b2 = np.nanmin(a, axis=1)
b3 = np.nanmean(a, axis=1)
```

Минимум/максимум

```
A = np.arange(12); print(A)
print(f'min={A.min():5.2f}, max={A.max():5.2f}')
> min= 0.00, max=11.00
print(f'mean={A.mean():5.2f}, std={A.std():5.2f}')
> mean= 5.50, std= 3.45
A = np.arange(12).reshape((3,-1)); print(A)
print('axis=0:', A.min(axis=0)) # axis=0: [0 1 2 3]
print('axis=1:', A.min(axis=1)) # axis=1: [0 4 8]
```

Задание. Пусть A — 3*4-матрица со случайными значениями $0, \ldots, 9$.

а) Найти минимум и максимум элементов матрицы

б) Найти позиция, в которой стоит минимум.

в) Найти минимум в каждой строке матрицы

г) Найти минимум в каждом столбце матрицы

Случайные числа

```
np.random.seed(777)
Z = np.random.random(10) # вектор со случайными значениями
Z = np.random.random((3,3,3)) # массив 3x3x3 со случайными
# массив 3*4 с целыми случайными значениями [10..19]:
Z = np.random.randint(10, 20, size = (3,4))
# массив 3*4 с нормальными СВ,МО=10, sigma=2:
Z = np.random.normal(10, 2, size = (3,4))
```

Задание. а) Пусть A — вектор длины N (10,100,1000) со случайными значениями [0,1]. Найти минимум и максимум A.
б) Пусть A — вектор длины N (10,100,1000) со значениями нормальной СВ, МО=0, $\sigma = 1$. Найти минимум и максимум A.
в) Аналогично, но МО=100, $\sigma = 10$.

Вставить в матрицу столбец из единиц

```
A = np.arange(12).reshape((3,-1)).astype(float); print(A)
>  [[ 0. 1.  2.  3.]
>   [ 4. 5.  6.  7.]
>   [ 8. 9. 10. 11.]]

# Вставить в начало столбец из единиц:
my = len(A)   # = A.shape[0]
A = np.column_stack((np.ones(my).reshape((-1,1)),A))
print(A)
> [[ 1.  0.  1.  2.  3.]
>  [ 1.  4.  5.  6.  7.]
>  [ 1.  8.  9. 10. 11.]]
```

Задание. Добавить в матрицу верхнюю строку из единиц.

Сортировка матрицы

Отсортировать матрицу по n-ому столбцу

```
Z = np.random.randint(0,10,(4,4))
n = 1    # Нумерация с нуля
print(Z)
print(Z[:,n].argsort())
print(Z[Z[:,n].argsort()])
```

Задание. Пусть `A = np.arange(12).reshape(3,4)`.
а) Отсортировать строки матрицы по последнему столбцу.
б) Отсортировать столбцы матрицы по последней строке.

Текстовые файлы

```
Z = np.loadtxt('01.txt', skiprows=1, delimiter=',')
print(Z)

Записать np-массив в текстовый файл:
b = np.array([[1.5, 235, 3], [4.1245, 5, 6]])
np.savetxt("b.txt", b, fmt="%3.0f", delimiter=" ",
   header=" 2 3 a, b, c", comments=" ")

Форматы: %7d, %8.4f, %10.2e
```

Сжатые файлы

Запись нескольких массивов в сжатый файл. Внешние имена могут отличаться от внутренних:

```
aT = np.ones((10,10)); bT = np.zeros((3,7))
cT = np.ones((9,1));
np.savez_compressed("fname", a=aT, b=bT, c=cT)

# Чтение из сжатого файла:
Tmp = np.load("fbane.npz")
print(Tmp.files) # ['a', 'b', 'c']
aT, bT, cT = Tmp["a"], Tmp["b"], Tmp["c"]
print("Readed:aT=", aT, "\nbT=", bT, "\ncT=", cT)
```

Задание

Пусть:

```
A = np.sin(np.arange(100).reshape((-1,5)))
```

Задание. а) Сохранить массив A в файл A.csv со строкой-заголовком (придумать, каким), с разделителем ';' и с 5-ю знаками после десятичной точки.
б) Прочитать файл A.csv в матрицу B и вывести её на экран.
в) Вывести на экран разницу между A и B и наибольшее по модулю отклонение.

Перестановка строк

Переставим первые две строки. Вот так нельзя:

```
a = np.arange(20).reshape(5,4)
print(a, '=a\n')
a[0],a[1] = a[1], a[0]
print(a, '=a2\n')
```

Надо так:

```
a[[0,1]] = a[[1,0]]
print(a, '=a2\n')
```

А как переставить два столбца?

```
a[:, [0,1]] = a[:, [1,0]]
print(a, '=a2\n')
```

Список различных элементов

Как построить список различных элементов массива?

```
a = np.random.randint(0,10, size=(4,2))
print(a, '=a\n')
b = np.unique(a)
print(b, '=b\n')
```

А где они лежат?

```
b, b_idx = np.unique(a, return_index=True)
print(b, '=b\n')
print(b_idx, '=b_idx\n')
```

А как получить двумерные индексы?

```
idx_x, idx_y= np.unravel_index(b_idx, a.shape)
print(idx_x)
print(idx_y)
```

Сколько раз встречается

```
a = np.random.randint(0,10, size=(5,6))
print(a, '=a\n')
values, counts = np.unique(a, return_counts=True)
print(values, '=values\n')
print(counts, '=counts\n')
```

nonzero, where

```
A = [1,2,0,0,4,0.0] # array of double!
nz =np.nonzero(A)
print(nz)        # (array([0, 1, 4], dtype=int64),)
print(nz[0])     # [0 1 4]
idx = np.where(A>1)
print(idx)       # (array([1, 4], dtype=int64),)
idx = np.where(A>1)[0]
print(idx)       # [1 4]
```

Количество отрицательных элементов:

```
a =np.array([1,2,0,-3.0,4,-1])
print( (a<0).sum())       # 2
```

Задание. Найдите количество элементов в векторе лежащих в интервале от [-2..2].

nonzero, where

```
A = [1,2,0,0,4,0.0] # array of double!

A[2]=A[5]=np.nan
print(A)          # [ 1.  2. nan  0.  4. nan]
print(np.where( np.isnan(A))[0]) # [2 5]
print(np.where(~np.isnan(A))[0]) # [0 1 3 4]
A[np.isnan(A)] = 11
print(A)          # [ 1.  2. 11.  0.  4. 11.]
```

Задание

1. Написать функцию f(A), умножающую отрицательные элементы матрицы на 3.
2. Написать функцию f(A), которая отрицательные элементы матрицы заменяет на 0, а элементы, большие 255 — на 255.
3. Написать функцию f(A), которая каждый элемент матрицы x заменяет на $1/x$. Как избежать worning'a?
4. Написать функцию f(A), которая каждый элемент матрицы x заменяет на $\sqrt{x}$. Проверить для матрицы имеющей отрицательые коэффициенты. Как избежать worning'a?

nonzero, where

Выбрать элементы с данным значением в другом массиве:

```
a = np.array([1,1,1,2,2,2,2,2])
b = np.array([3,4,5,6,7,8,9,0])
print(b[a==2])  # [6 7 8 9 0]
```

Нормализация столбцов матрицы

```
A = np.arange(12).reshape((3,-1)).astype(float); print(A)
#Из каждого столбца вычесть его среднее значение:
A -= A.mean(axis=0); print(A) # mean=[4 5 6 7]
> [[-4. -4. -4. -4.]                  # [[ 0 1  2  3]
>  [ 0.  0.  0.  0.]                  #  [ 4 5  6  7]
>  [ 4.  4.  4.  4.]]                 #  [ 8 9 10 11]]
print('A.std(axis=0):',A.std(axis=0))
> A.std(axis=0): [3.266 3.266 3.266 3.266]

# Каждый столбец поделить на его ср.кв.отклонение:
A /= A.std(axis=0); print(A)
> [[-1.2247 -1.2247 -1.2247 -1.2247]
>  [ 0.      0.      0.      0.    ]
>  [ 1.2247  1.2247  1.2247  1.2247]]
```

Задание

Пусть A — матрица 8*8, со значением в k-й клетке 2^k.

а) Нормализовать столбцы матрицы.

б) Нормализовать строки матрицы.

Бинаризация

Бинаризация признака со значениями (0,1,2,3):

```
a = np.array([[2,3,0,0,3,3]])
print(np.eye(4)[a])
[[[0. 0. 1. 0.]     ! 2
  [0. 0. 0. 1.]     ! 3
  [1. 0. 0. 0.]     ! 0
  [1. 0. 0. 0.]     ! 0
  [0. 0. 0. 1.]     ! 3
  [0. 0. 0. 1.]]]   ! 3
```

Бинаризация столбца матрицы

```
def binarize(X, k):
    ''' Бинаризация k-го столбца матрицы X
    :return Новая матрица
    '''
    X0 = X[:, :k]                    # левая часть
    X1 = X[:, k].astype(int)
    maxX1 = X1.max()
    X1bin = np.eye(maxX1+1)[X1] # бинаризация
    X2 = X[:, k:]                    # правая часть
    return np.hstack((X0, X1bin, X2))
```

Удаление нулевых столбцов

Исходная матрица:

```
a = np.array([2,3,0,0,3]); b = np.eye(4)[a]
print(b, '=b\n')
> [[0. 0. 1. 0.]
>  [0. 0. 0. 1.]
>  [1. 0. 0. 0.]
>  [1. 0. 0. 0.]
>  [0. 0. 0. 1.]] =b
c =b[:,~(b == 0).all(0)] # удаление нулевых столбцов
print(c, '=c\n')
> [[0. 1. 0.]
>  [0. 0. 1.]
>  [1. 0. 0.]
>  [1. 0. 0.]
>  [0. 0. 1.]] =c
```

На самом деле, всё просто!

```
c =b[:,~(b == 0).all(0)] # удаление нулевых столбцов

print(b==0, 'b==0\n')
> [[ True  True False  True]
>  [ True  True  True False]
>  [False  True  True  True]
>  [False  True  True  True]
>  [ True  True  True False]] b==0

print((b == 0).all(0))
> [False  True False False]

print(~(b == 0).all(0))
> [ True False  True  True]

c =b[:,~(b == 0).all(0)]
```

Разное

```
# список нулевых строк матрицы A:
LZero = np.all(A == 0, axis=1)
# удалить нулевые строки матрицы:
Z = A[~LZero]

#В каждой строке матрицы A найти индекс
#последнего ненулевого элемента:
N = A.shape[0]
for a1 in A:
   print(N-(a1!=0)[::-1].argmax())

# Сортировка матрицы A по 3-у столбцу:
B = A[ np.argsort(A[:,2]) ]
```

Матрица Гильберта

```
import numpy as np
np.set_printoptions(precision=8, linewidth = 100)
def hilb_mat(N):
    A = np.zeros((N,N))
    for i in range(N):
        for j in range(N): A[i,j] = 1/(1+i+j)
    return A
N = 4
A = hilb_mat(N)
```

```
[[1.         0.5        0.333333 0.25    ]
 [0.5        0.333333 0.25     0.2     ]
 [0.333333 0.25     0.2      0.166667]
 [0.25     0.2      0.166667 0.142857]] =A
```

Матрица Гильберта, свойства

1. Обратная матрица `A1 = np.linalg.inv(A)` является целой.
2. `A1*A = E` — единичная матрица.

Проверим, насколько точно это выполняется.

```
N = 4
A = hilb_mat(N)
```

```
[[   16.  -120.   240.  -140.]
 [ -120.  1200. -2700.  1680.]
 [  240. -2700.  6480. -4200.]
 [ -140.  1680. -4200.  2800.]] =1/A
```

```
print(A1[-2,-2], ', ', A1[-1,-1])
> 6480.000000000418 2800.000000000184
```

Матрица Гильберта, проверка

```
N = 8
print(A1[-2,-2], ', ', A1[-1,-1])
2175421171.579993 ,  176679353.81126776

print(A2[-2:,-2:], '=A*1/A\n')
[[ 1.00000e+00 -3.72529e-09]
 [ 0.00000e+00  1.00000e+00]] =A*1/A
```

При $N = 2, 3, \ldots, 20$ найти среднеквадратичное отклонение матрицы `A.dot(np.linalg.inv(A))` от единичной матрицы.

Как решать систему линейных уравнений?

```
import numpy as np
np.set_printoptions(precision=6, linewidth = 100)
A = np.array((
    (2,0,0),
    (0,3,0),
    (0,0,4.)))
b = np.array((1.,1,1,))
print(A, '=A\n',b, '=b\n')
x = np.linalg.solve(A,b)
print(x, '=x\n')
```

```
>[[2. 0. 0.]
> [0. 3. 0.]
> [0. 0. 4.]] =A
> [1. 1. 1.] =b
>[0.5      0.333333 0.25    ] =x
```

Как решать систему линейных уравнений?

```
A = np.array((
    (2,0,0),
    (0,3,3),
    (0,1,1.)))
b = np.array((1.,1,1,))
print(A, '=A\n',b, '=b\n')
x = np.linalg.solve(A,b)
print(x, '=x\n')

> numpy.linalg.LinAlgError: Singular matrix
```

Как решать систему линейных уравнений?

```
 D = np.linalg.inv(np.array((
    (1,2,0),
    (3,3,1),
    (2,0,5.))))
 A = A.dot(D)
 A[-1,-1]+= 1e-13
 print(A, '=A\n')
[[-2.727273  1.818182 -0.363636]
 [ 5.181818 -2.454545  1.090909]
 [ 1.727273 -0.818182  0.363636]] =A

[-2.667318e+12 -2.667318e+12  6.668295e+12] =x
```