

# NumPy基础操作练习

2024



01

概述

02

数组对象基础

03

数组的索引和切片

04

针对数组的操作

05

运算和通用函数

06

简单统计应用

07

矩阵，线性代数，傅里叶变换

08

随机数的产生（选做）

09

文件操作（选做）

01

# 概述

——关于NumPy 的简介



- NumPy , 是Numerical Python 的简称, 是目前Python数值计算中最重要的基础包。
- NumPy 前身是一款名为Numeric的库, 有Jim Hugunin 等人开发。2005年, Travis Oliphant 在其基础上结合了另一个同性质库Numarray的特点, 并加入其他扩展开发了NumPy。
- NumPy 本身并不提供建模和科学函数, 理解NumPy的数组以及基于数组的计算将帮助你更高效的使用基于数组的工具, 如Pandas。



# 1.概述

- NumPy 节约内存和CPU计算时间
- 与 list 区别

```
In [3]: import numpy as np
```

```
In [4]: my_arr = np.arange(10000000)
```

```
In [5]: my_list = list(range(10000000))
```

```
In [6]: %time for _ in range(10): my_arr2 = my_arr * 2
```

```
Wall time: 255 ms
```

```
In [7]: %time for _ in range(10): my_list2 = [x * 2 for x in my_list]
```

```
Wall time: 9.12 s
```



02

# 数组对象基础

——初识  
——创建



- numPy 导入

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

In [2]: np.__version__
Out[2]: '1.15.1'
```

- ndarray对象

```
In [3]: data = np.array([1, 2, 3, 4, 5])

In [4]: data
Out[4]: array([1, 2, 3, 4, 5])

In [5]: type(data)
Out[5]: numpy.ndarray
```



### ● dir

- dir 可以做很多的事情，列出一个函数，一个包内部包含的命令或者函数，
- 比如 dir (np)

```
In [6]: dir(data)
Out[6]:
['_T',
 '_abs_',
 '_add_',
 '_and_',
 '_array_',
 '_array_finalize_',
 '_array_interface_',
 '_array_prepare_',
 '_array_priority_',
 '_array_struct_',
 '_array_ufunc_',
 '_array_wrap_',
 '_bool_',
 '_class_',
 '_complex_',
 '_contains_',
 '_copy_',
 '_deepcopy_',
 '_delattr_']
```

- ? , ipython的命令，可以理解为对当前对象的“询问”，系统给出对象的一些特征，比如数据类型，数据等。

```
In [16]: data?
Type: ndarray
String form: [1 2 3 4 5]
Length: 5
File: c:\programdata\anaconda
Docstring: <no docstring>
Class docstring:
ndarray(shape, dtype=float, buffer=None,
         strides=None, order=None)

An array object represents a multidimensional
array of fixed-size items. An associated data
format of each element in the array (its
occupies in memory, whether it is an integer
or something else, etc.)
```





### ● 数组元素的类型

```
In [17]: data.dtype  
Out[17]: dtype('int32')
```

### ● 数组元素类型改变?

```
In [26]: new_data = data.astype(np.float)  
  
In [27]: new_data.dtype, new_data  
Out[27]: (dtype('float64'), array([1., 2., 3., 4., 5.]))
```

类型	说明
Int: int8, int16, int32, int64	有符号整数型。
uint: uint8, uint16, uint32, uint64	无符号整数型。
bool	布尔型
float: float16, float32, float64, float128 (python 只有float64? )	浮点型
complex: complex64, complex128, complex256	复数型
string_	字符串类型
Unicode	Unicode类型



- 数组的常用属性

```
In [31]: data.shape
Out[31]: (5, )

In [32]: data.size
Out[32]: 5

In [33]: data.ndim
Out[33]: 1
```

属性	说明
dtype	数组中的元素类型
shape	返回整数元组，对应每个轴的元素个数
size	数组中元素的个数
ndim	维度，轴
dbytes	返回保存数据的字节数



- `np.array( )`是最基本的创建方法

```
In [34]: data1 = np.array([1, 2, 3, 4], dtype = float)
```

```
In [35]: data1
```

```
Out[35]: array([1., 2., 3., 4.])
```

```
In [36]: data1.dtype
```

```
Out[36]: dtype('float64')
```

```
In [40]: data2 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

```
In [41]: data2
```

```
Out[41]:  
array([[1, 2, 3],  
       [4, 5, 6],  
       [7, 8, 9]])
```



- 元素个数不一样的时候，类型不相同的时候：

```
In [42]: data3 = np.array([[1, 3], [4, 5, 6], [7, 8, 9]])
```

```
In [43]: data3
```

```
Out[43]: array([list([1, 3]), list([4, 5, 6]), list([7, 8, 9])], dtype=object)
```



- 用函数创建数组——针对有规律的数生成的数组

```
In [3]: np.zeros((2, 10))
```

```
Out[3]:
```

```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],  
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

```
In [4]: np.zeros?
```

```
Docstring:
```

```
zeros(shape, dtype=float, order='C')
```

Return a new array of given shape and type, filled with zeros.

Parameters

-----

shape : int or tuple of ints

Shape of the new array, e.g., ``(2, 3)`` or ``(2)``



### ● 常用创建数组的函数

函数	说明
asarray	输入参数为列表、元组，或者由他们组成的嵌套对象或数组，返回一个数组。如果参数是数组则返回数组本身
arange	根据开始值、结束值、步长来创建一个数组
ones, ones_like	创建元素值为1的数组
zeros, zeros_like	创建元素值为0的数组
eye, identity	创建对角元素为1，其余元素为0的数组
empty, empty_like	创建空数组，没有元素，只分配存储空间
diag	创建对角线元素是指定数值、其余元素为0的二维数组，可调整对角线位置
linspace	根据开始值、结束值和元素量创建元素是等差数列的数组
logspace	根据开始值、结束值和元素量和对数底创建元素是等比数列的数组





- 举例

```
In [42]: np.eye(4)
Out[42]:
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

```
In [43]: np.eye(4, 5, dtype = int, k = 2)
Out[43]:
array([[0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0],
       [0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0]])
```

```
In [28]: da = np.arange(1, 13)
```

```
In [29]: da.shape = 3, 4
```

```
In [30]: da
```

```
Out[30]:
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

```
In [31]: da1 = np.ones(da.shape)
```

```
In [32]: da1
```

```
Out[32]:
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

```
In [33]: da2 = np.ones_like(da)
```

```
In [34]: da2
```

```
Out[34]:
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])
```



- 举例
- 如何建立一个全部元素都是517.712? (np.full, np.full\_like)

```
In [44]: np.diag([1, 2, 3, 4], k = 1)
Out[44]:
array([[0, 1, 0, 0, 0],
       [0, 0, 2, 0, 0],
       [0, 0, 0, 3, 0],
       [0, 0, 0, 0, 4],
       [0, 0, 0, 0, 0]])
```

```
In [50]: de = np.arange(16).reshape(4, 4)

In [51]: de
Out[51]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

```
In [52]: np.diag(de)
Out[52]: array([ 0,  5, 10, 15])
```



- 举例：元素是等差的数列`np.arange()` , `np.linspace()`

```
In [54]: np.arange(1, 100, 3)
```

```
Out[54]:
```

```
array([ 1,  4,  7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46,
        49,
        52, 55, 58, 61, 64, 67, 70, 73, 76, 79, 82, 85, 88, 91, 94, 97])
```

```
In [58]: np.linspace(1, 100, 20)
```

```
Out[58]:
```

```
array([ 1.          ,  6.21052632, 11.42105263, 16.63157895,
        21.84210526, 27.05263158, 32.26315789, 37.47368421,
        42.68421053, 47.89473684, 53.10526316, 58.31578947,
        63.52631579, 68.73684211, 73.94736842, 79.15789474,
        84.36842105, 89.57894737, 94.78947368, 100.          ])
```

# 03

## 数组的索引和切片

- 数组的轴
- 索引和切片



### ● 数组的轴

```
In [71]: a = np.arange(24).reshape((2, 3, 4))
```

```
In [72]: a
```

```
Out[72]:  
array([[ [ 0,  1,  2,  3],  
         [ 4,  5,  6,  7],  
         [ 8,  9, 10, 11]],  
       [[12, 13, 14, 15],  
        [16, 17, 18, 19],  
        [20, 21, 22, 23]]])
```

```
In [73]: a.ndim, np.ndim(a)
```

```
Out[73]: (3, 3)
```

```
In [74]: a.shape
```

```
Out[74]: (2, 3, 4)
```

```
In [75]: a[1]
```

```
Out[75]:  
array([[12, 13, 14, 15],  
       [16, 17, 18, 19],  
       [20, 21, 22, 23]])
```

```
In [76]: a[1][1]
```

```
Out[76]: array([16, 17, 18, 19])
```

```
In [77]: a[1][1][2]
```

```
Out[77]: 18
```

```
In [79]: a[1, 1, 2]
```

```
Out[79]: 18
```

```
In [80]: a[(1, 1, 2)]
```

```
Out[80]: 18
```



- 获取元素——下标 索引  
下标可以是整数, 列表, 数组

- `c = np`

```
array([ 0.          ,  5.26315789, 10.52631579, 15.78947368,  
       21.05263158, 26.31578947, 31.57894737, 36.84210526,  
       42.10526316, 47.36842105, 52.63157895, 57.89473684,  
       63.15789474, 68.42105263, 73.68421053, 78.94736842,  
       84.21052632, 89.47368421, 94.73684211, 100.          ])
```

```
In [100]: c[[0, 1, 2, 4, 5]]
```

```
Out[100]: array([ 0.          ,  5.26315789, 10.52631579, 21.05263158, 26.  
                .31578947])
```





- 切片 ——:

```
In [147]: f
Out[147]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
In [148]: f[0:2][:, 0:2]
Out[148]:
array([[0, 1],
       [4, 5]])
```

```
In [160]: e
Out[160]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [161]: e[:6:2]
Out[161]: array([0, 2, 4])
```

```
In [162]: e[::-1]
Out[162]: array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```



- 修改元素

```
In [88]: c = np.arange(12).reshape(3, 4)
```

```
In [89]: c
```

```
Out[89]:
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
In [90]: c[0, 0] = 100
```

```
In [91]: c
```

```
Out[91]:
```

```
array([[100,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
In [92]: c[1] = 114
```

```
In [93]: c
```

```
Out[93]:
```

```
array([[100,  1,  2,  3],
       [114, 114, 114, 114],
       [ 8,  9, 10, 11]])
```

# 04

## 针对数组的操作

——变形  
——组合与分割



### ● 变形

np.reshape() 或 .reshape()  
np.flatten() .flatten()  
np.ravel() .ravel() (扁平化数组)

```
In [175]: a = np.arange(12)
```

```
In [176]: b = np.reshape(a, (3, 4))
```

```
In [177]: b
```

```
Out[177]:
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
In [178]: b = a.reshape(3, 4)
```

```
In [179]: b
```

```
Out[179]:
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
In [188]: a.shape = (-1, 1)
```

```
In [189]: a
```

```
Out[189]:
```

```
array([[ 0],
       [ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [10],
       [11]])
```

-1 代表让  
numpy自  
己判断0轴  
数据个数



- 组合与分割

np.stack()

np.hstack() 水平组合

np.vstack() 垂直组合

np.concatenate()

np.hsplite()

np.vsplite()

np.split()

```
In [219]: a
Out[219]:
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

```
In [220]: b
Out[220]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
In [221]: c
Out[221]:
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

```
In [222]: np.hstack((a, b, c))
Out[222]:
array([[ 0,  1,  2,  0,  1,  2,  3,  0,  1,  2,  3,  4],
       [ 3,  4,  5,  4,  5,  6,  7,  5,  6,  7,  8,  9],
       [ 6,  7,  8,  8,  9, 10, 11, 10, 11, 12, 13, 14]])
```

# 05

## 运算和通用函数

- 算术运算
- 比较和逻辑运算
- 通用函数





- 算数运算

广播——自动补齐（其中一个轴上元素个数相等才可以补齐）

```
In [253]: a = np.arange(10).reshape(2, 5)
```

```
In [254]: a
```

```
Out[254]:  
array([[0, 1, 2, 3, 4],  
       [5, 6, 7, 8, 9]])
```

```
In [255]: b=np.array([2, 4, 6, 8, 9])
```

```
In [256]: b
```

```
Out[256]: array([2, 4, 6, 8, 9])
```

```
In [257]: a+b
```

```
Out[257]:  
array([[ 2,  5,  8, 11, 13],  
       [ 7, 10, 13, 16, 18]])
```

```
In [258]: b=np.array([2, 4, 6, 8])
```

```
In [259]: a+b
```



- 比较和逻辑运算

```
In [260]: np.array([3, 4, 5]) > np.array([1, 4, 9])  
Out[260]: array([ True, False, False])
```



### ● 通用函数

NumPy 中有一类函数被称为“通用函数” (Universal Function —— Ufunc) , 能对数组中每一个元素进行操作, 即**元素级函数**, 并且这些函数都是在C语言级别实现的, 因此它们的计算速度非常快。

```
In [2]: alpha = np.linspace(-1, 1, 11)
```

```
In [3]: y = np.sin(np.pi*alpha)
```

```
In [4]: y
```

```
Out[4]:
```

```
array([-1.22464680e-16, -5.87785252e-01, -9.51056516e-01, -9.51056516e-01,  
       -5.87785252e-01,  0.00000000e+00,  5.87785252e-01,  9.51056516e-01,  
        9.51056516e-01,  5.87785252e-01,  1.22464680e-16])
```



### ● 常用通用函数

函数	说明
np.sin , np.cos, np.tan	三角函数
np.arcsin, np.arccos, np.atctan	反三角函数
np.sinh, np.cosh, np.tanh	双曲三角函数
np.arcsinh, np.arccosh, np.arctanh	双曲反三角函数
np.sqrt, np.exp,	求平凡根, 求自然指数
np.log, np.log2, np.log10	计算对数 (e , 2 , 10 为底)
np.add, np.subtract, np.multiply, np.divide	+ - */
np.equal, np.not_equal, np.less_equal, np.greater, np.greater_equal	比较运算符
np.power, np.remainder, np.reciprocal	指数运算, 取余, 倒数
np.real, np.imag, np.conj	返回复数 实部, 虚部, 完整的复数
np.sign, np.abs	返回符号, 绝对值
np.floor, np.ceil, np rint	取整
np, round	四舍五入

06

## 简单统计函数



- 常用简单统计函数

函数	说明
np.mean, np.average	计算平均值，加权平均值
np.var	计算方差
np.std	计算标准差
np.min, np.max	计算最小值，最大值
np.argmin,np.argmax	返回最小值，最大值的索引
np.ptp	计算全距，即最大值最小值的差
np.percentile	计算百分位在统计对象中的值
np.median	计算统计对象的中值
np.sum	计算统计对象的和





- 举例：统计成绩（选做）

以下几行代码产生随机数，表示高斯分布的物理，数学成绩

```
In [96]: g = np.random.normal(0, 0.5, 1000)
```

```
In [97]: g = (g-g.min())/10
```

```
In [98]: phy = np.round(100*g, 1)+60
```

```
In [99]: g1 = np.random.normal(0, 0.5, 1000)
```

```
In [100]: g1 = (g1-g1.min())/10
```

```
In [101]: mat = np.round(100*g1, 1)+60
```



## 6.简单统计函数

```
In [102]: np.mean(phy)
```

```
Out[102]: 76.6707
```

```
In [103]: np.mean(mat)
```

```
Out[103]: 76.59209999999999
```

```
In [104]: np.std(phy)
```

```
Out[104]: 4.913743126171737
```

```
In [105]: np.max(phy)
```

```
Out[105]: 94.1
```

```
In [106]: np.max(mat)
```

```
Out[106]: 94.0
```

```
In [108]: marks = np.vstack((phy,mat))
```

```
In [109]: marks
```

```
Out[109]:  
array([[77.9, 75.3, 77.4, ..., 75. , 71.1, 81.3],  
       [75. , 88.2, 86.3, ..., 68.9, 71.8, 75.5]])
```

```
In [112]: np.mean(marks)
```

```
Out[112]: 76.6314
```

```
In [113]: np.mean(marks, 1)
```

```
Out[113]: array([76.6707, 76.5921])
```

```
In [114]: np.mean(phy)
```

```
Out[114]: 76.6707
```

```
In [115]: np.mean(marks, 0)
```

```
Out[115]:  
array([76.45, 81.75, 81.85, 74.6 , 75.   , 81.9 , 75.8 , 74.8 , 74.05,  
       76.5 , 75.35, 79.55, 71.7 , 80.35, 73.7 , 76.25, 77.   , 77.35,  
       81.15, 77.85, 76.4 , 73.2 , 76.4 , 77.15, 79.7 , 79.15, 79.15,  
       70.3 , 77.1 , 75.15, 80.05, 75.7 , 74.5 , 72.95, 81.7 , 82.75,  
       78.4 , 80.2 , 73.85, 75.9 , 75.7 , 75.3 , 79.45, 75.95, 77.6 ,  
       77.9 , 79.05, 80.5 , 75.55, 81.65, 80.25, 71.4 , 80.1 , 77.85])
```



```
In [117]: phy[phy>np.mean(phy)]
```

```
Out[117]:
```

```
array([77.9, 77.4, 84.3, 88.1, 80.4, 79.6, 77.2, 79.3, 85.7, 81.9, 82.2,  
       77.9, 84.5, 77.8, 78.5, 81.4, 85.1, 82.3, 84.9, 82.3, 79.5, 82.7,  
       83.8, 82.2, 79.1, 82.7, 80.2, 79.1, 79.5, 85.4, 84.8, 84.4, 78.3,  
       78. , 81.8, 79.7, 77.2, 81.7, 81.2, 87.5, 80.2, 78.1, 78.7, 78.4,  
       84. , 79. , 78.6, 78.5, 78. , 80.2, 80. , 81.8, 78.8, 84.1, 77. ,  
       80.7, 84.9, 79.2, 80.5, 78.4, 83.6, 85.9, 76.7, 79.5, 80.2, 84.1,  
       83.7, 77.2, 80.1, 79.8, 80.2, 83.3, 80.7, 81.5, 90. , 86. , 88. ,  
       89.1, 81.7, 78.4, 77. , 84.3, 82.4, 83.7, 77.4, 77.7, 78.8, 77.7,  
       78.9, 77.9, 79.8, 77.6, 79.2, 76.9, 79.5, 82.2, 83. , 83.6, 84.2,
```

```
In [118]: np.where(phy > phy.mean(), 1, 0)
```

```
Out[118]:
```

```
array([1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0,  
       0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0,  
       0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0,  
       1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1,  
       0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,  
       1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1,  
       0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1,  
       0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1
```



```
In [119]: np.where(phy > phy.mean())
```

```
Out[119]:
```

```
(array([ 0,  2,  5,  8, 11, 13, 15, 17, 18, 24, 25, 26, 28,
        29, 31, 34, 35, 37, 42, 45, 46, 47, 49, 50, 52, 57,
        58, 59, 60, 61, 62, 63, 64, 66, 67, 68, 69, 74, 75,
        80, 84, 85, 87, 92, 93, 94, 96, 98, 99, 103, 105, 110,
```

```
In [122]: np.select([phy >= grand_a, np.logical_and(phy>=grand_b, phy<grand_a)], ['A',
...: , 'B'], default = 'C')
```

```
Out[122]:
```

```
array(['B', 'C', 'B', 'C', 'C', 'B', 'C', 'C', 'B', 'C', 'C', 'B', 'C',
       'B', 'C', 'B', 'C', 'B', 'B', 'C', 'C', 'C', 'C', 'C', 'B', 'B',
       'B', 'C', 'B', 'B', 'C', 'B', 'C', 'C', 'B', 'B', 'C', 'B', 'C',
       'C', 'C', 'C', 'B', 'C', 'C', 'B', 'B', 'B', 'C', 'B', 'B', 'C',
```

07

## 矩阵（选做）

- 创建：np.matrix() np.mat()
- 矩阵操作
- 综合应用：多项式、线性方程组

- 创建: `np.matrix()` `np.mat()`

```
In [123]: np.matrix?  
Init signature: np.matrix(data, dtype=None, copy=True)  
Docstring:  
matrix(data, dtype=None, copy=True)
```

```
In [124]: np.mat?  
Signature: np.mat(data, dtype=None)  
Docstring:  
Interpret the input as a matrix.
```

Unlike ``matrix``, ``asmatrix`` does not make a copy if the input is already a matrix or an ndarray. Equivalent to ``matrix(data, copy=False)``.



### ● 矩阵操作

```
In [175]: c = np.arange(9).reshape(3, 3)
```

```
In [176]: C = np.mat(c)
```

```
In [177]: C.I
```

```
LinAlgError                                Traceback (most recent call last)
<ipython-input-177-83d1c4b8f7f3> in <module>()
----> 1 C.I
```

```
In [178]: c[0,0]=11
```

```
In [179]: C
```

```
Out[179]:
matrix([[11,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8]])
```

```
In [180]: C.I
```

```
Out[180]:
matrix([[ 0.09090909, -0.18181818,  0.09090909],
        [-0.18181818, -2.3030303 ,  1.48484848],
        [ 0.09090909,  2.15151515, -1.24242424]])
```

```
In [181]: np.round(C*C.I)
```

```
Out[181]:
matrix([[1.,  0.,  0.],
        [0.,  1.,  0.],
        [0.,  0.,  1.]])
```

- 综合应用——多项式

$$f(x) = a_0x^n + a_1x^{(n-1)} + a_2x^{(n-1)} + \dots + a_{n-1}x + a_n$$

eg.  $f(x) = x^2 - 2x + 1$

```
In [186]: a = np.array([1,-2,1])
In [187]: p = np.poly1d(a)
In [188]: p
Out[188]: poly1d([ 1, -2,  1])
In [189]: type(p)
Out[189]: numpy.lib.polynomial.poly1d
In [190]: p(5)
Out[190]: 16
```

```
In [191]: print(p)
      2
1 x - 2 x + 1
```

```
In [196]: p = np.poly1d(a, variable = 'm')
In [197]: print(p)
      2
1 m - 2 m + 1
```

`np.poly1d()` : numpy提供的创建多项式的函数。可用 `np.poly1d?` 进行详细查看





`r=True` 矩阵`a` 为多项式的根

$$(x - 1)(x + 2)(x - 1) = x^3 - x + 2$$

```
In [198]: p2 = np.poly1d(a, r = True)
```

```
In [199]: p2
```

```
Out[199]: poly1d([ 1.,  0., -3.,  2.])
```

```
In [200]: print(p2)
```

```
      3
1 x - 3 x + 2
```



### 多项式运算

```
In [204]: p+p2  
Out[204]: poly1d([ 1.,  1., -5.,  3.])
```



### 拟合多项式np.polyfit()

```
In [225]: f3 = np.polyfit(x, y, 3)
```

```
In [226]: pf3 = np.poly1d(f3)
```

```
In [227]: pf3(x)
```

```
Out[227]:
```

```
array([-0.203109 , -0.2019105 , -0.20071271, ...,  0.20071271,  
       0.2019105 ,  0.203109  ])
```

```
In [228]: np.polyval(f3, x)
```

```
Out[228]:
```

```
array([-0.203109 , -0.2019105 , -0.20071271, ...,  0.20071271,  
       0.2019105 ,  0.203109  ])
```

```
In [229]: space3 = np.abs(np.polyval(f3, x)-y)
```

```
In [230]: space5 = np.abs(np.polyval(f5, x)-y)
```

```
In [231]: space8 = np.abs(np.polyval(f8, x)-y)
```

```
In [232]: np.max(space3), np.max(space5), np.max(space8)
```

```
Out[232]: (0.20310899877773028, 0.015943766615308455, 0.0006625828771720828)
```



- 综合应用——解线性方程组

$$\begin{aligned}a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\&\dots\dots\dots \\a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m\end{aligned}$$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

$$\begin{cases} 3x_0 + x_1 = 9 \\ x_0 + 2x_1 = 8 \end{cases}$$

```
In [233]: a = np.array([[3, 1], [1, 2]])
In [234]: b = np.array([9, 8])
In [235]: x = np.linalg.solve(a, b)
In [236]: x
Out[236]: array([2., 3.]
```



### ● 傅里叶变换 (选做)

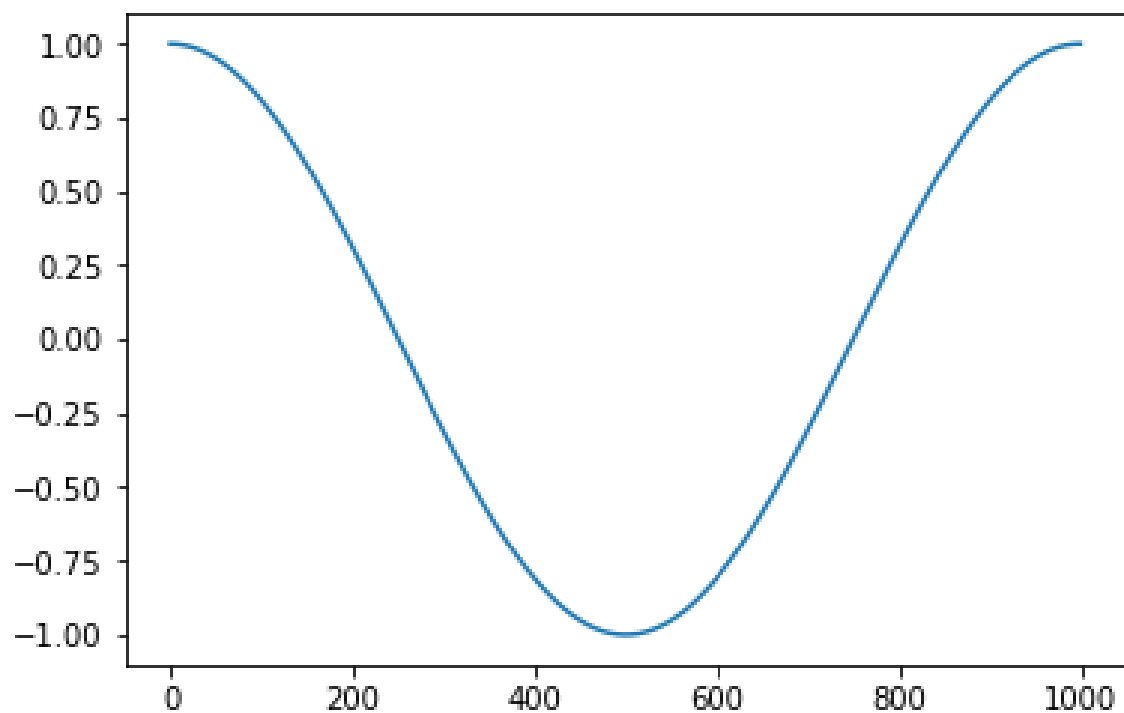
NumPy中, fft模块提供了快速傅里叶变换的功能。在这个模块中, 许多函数都是成对存在的, 也就是说许多函数存在对应的逆操作函数。例如, fft和ifft函数就是其中的一对。

```
In [3]: import numpy as np
from matplotlib.pyplot import plot, show
x = np.linspace(0, 2 * np.pi, 1000) #创建一个包含1000个点的余弦波信号
wave = np.cos(x)
plot (wave)
transformed = np.fft.fft(wave) #使用fft函数对余弦波信号进行傅里叶变换。
print (np.all(np.abs(np.fft.ifft(transformed) - wave) < 10 ** -9)) #对变换后的结果应用ifft函数, 应该可以近似地还原初始信号。
plot(transformed) #使用Matplotlib绘制变换后的信号。
show()
```

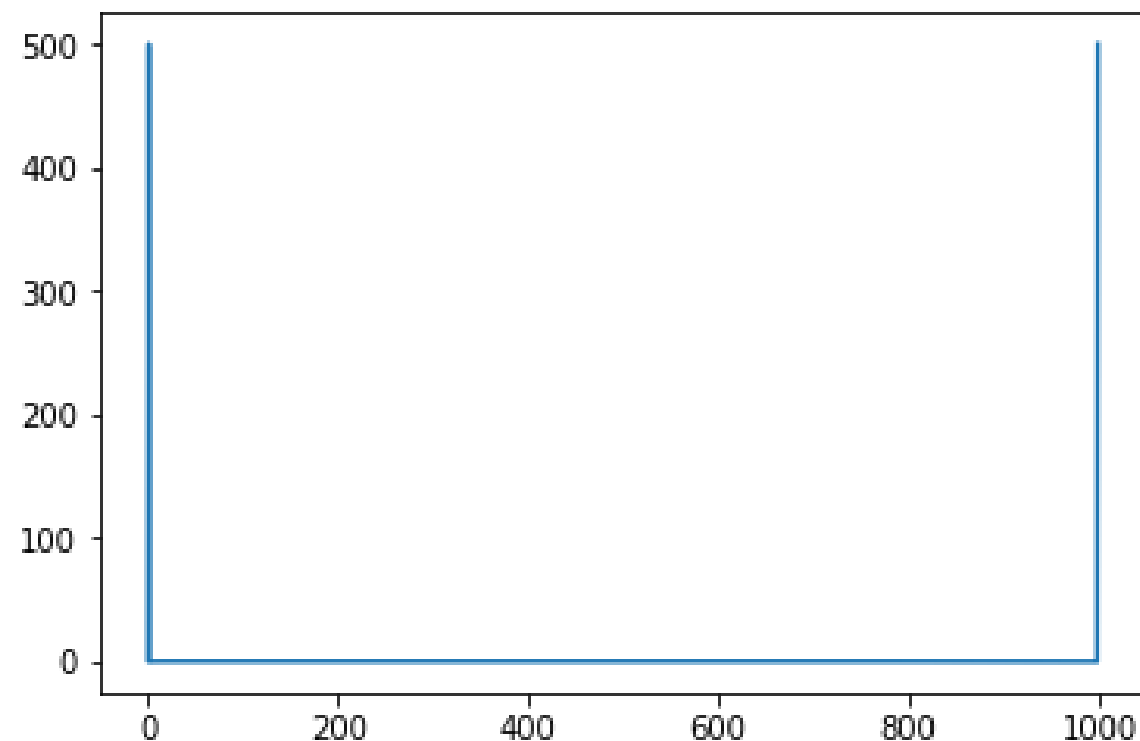

True



- 傅里叶变换



fft



08

## 随机数的产生 (选做)



Numpy中的random模块包含了很多方法可以用来产生随机数

### 1、numpy.random.rand(d0, d1, ..., dn)

**作用：**产生一个给定形状的数组（其实应该是ndarray对象或者是一个单值），数组中的值服从[0, 1)之间的均匀分布。

**参数：**d0, d1, ..., dn : int, 可选。如果没有参数则返回一个float型的随机数，该随机数服从[0, 1)之间的均匀分布。

**返回值：**ndarray对象或者一个float型的值

```
In [12]: # [0, 1)之间均匀分布的随机数, 3行2列
a = np.random.rand(3, 2)
print(a)
# 不提供形状
b = np.random.rand()
print(b)
```

```
[[0.58371013 0.91576208]
 [0.55376148 0.68510616]
 [0.22174249 0.11074821]]
0.5674480103600796
```





### 2、`numpy.random.uniform(low=0.0, high=1.0, size=None)`

**作用：**返回一个在区间`[low, high)`中**均匀分布**的数组，`size`指定形状。

**参数：**`low, high`: `float`型或者`float`型的类数组对象。指定抽样区间为`[low, high)`，`low`的默认值为0.0，`high`的默认值为1.0

`size`: `int`型或`int`型元组。指定形状，如果不提供`size`，则返回一个服从该分布的随机数。

```
In [13]: # 在[1, 10)之间均匀抽样，数组形状为3行2列
a = np.random.uniform(1, 10, (3, 2))
print(a)
# 不提供size
b = np.random.uniform(1, 10)
print(b)

[[4.99149493  7.8876633 ]
 [3.20706012  3.2019976 ]
 [7.75868064  5.55468498]]
1.8794664237029526
```



### 3、numpy.random.randn(d0, d1, ..., dn)

**作用：**返回一个指定形状的数组，数组中的值服从**标准正态分布**（均值为0，方差为1）。

**参数：**d0, d1, ..., dn : int, 可选。如果没有参数，则返回一个服从标准正态分布的float型随机数。

**返回值：**ndarray对象或者float

```
In [14]: # 3行2列
a = np.random.randn(3, 2)
print(a)
# 不提供形状
b = np.random.randn()
print(b)

[[ 0.6307828 -2.1005185 ]
 [-1.0407031 -0.06673689]
 [ 0.24937363 -0.98680756]]
-1.5373043167734712
```



### 4、numpy.random.normal(loc=0.0, scale=1.0, size=None)

**作用：**返回一个由size指定形状的数组，数组中的值服从  $\mu=loc, \sigma=scale$  的正态分布。

**参数：**loc : float型或者float型的类数组对象，指定均值  $\mu$

scale : float型或者float型的类数组对象，指定标准差  $\sigma$

size : int型或者int型的元组，指定了数组的形状。如果不提供size，且loc和scale为标量（不是类数组对象），则返回一个服从该分布的随机数。

```
In [15]: # 标准正态分布, 3行2列
a = np.random.normal(0, 1, (3, 2))
print(a)
# 均值为1, 标准差为3
b = np.random.normal(1, 3)
print(b)
```

```
[[-0.27482116  1.00299395]
 [ 0.8415794  -2.56769844]
 [-0.6862089  -0.29478727]]
1.768583625530653
```



### 5、numpy.random.randint(low, high=None, size=None, dtype='i')

**作用：**返回一个在区间[low, high)中离散均匀抽样的数组，size指定形状，dtype指定数据类型。

**参数：**low, high: int型，指定抽样区间[low, high)

size: int型或int型的元组，指定形状

dtype: 可选参数，指定数据类型，比如int,int64等，默认是np.int

**返回值：**如果指定了size，则返回一个int型的ndarray对象，否则返回一个服从该分布的int型随机数。

```
In [16]: # 在[1, 10)之间离散均匀抽样，数组形状为3行2列
a = np.random.randint(1, 10, (3, 2))
print(a)
# 不提供size
b = np.random.randint(1, 10)
print(b)
# 指定dtype
c = np.random.randint(1, 10, dtype=np.int64)
print(c)
type(c)

[[6 4]
 [6 5]
 [4 8]]
6
8
```

Out[16]: numpy.int64



### 5、numpy.random.randint(low, high=None, size=None, dtype='i')

**作用：**返回一个在区间[low, high)中离散均匀抽样的数组，size指定形状，dtype指定数据类型。

**参数：**low, high: int型，指定抽样区间[low, high)

size: int型或int型的元组，指定形状

dtype: 可选参数，指定数据类型，比如int,int64等，默认是np.int

**返回值：**如果指定了size，则返回一个int型的ndarray对象，否则返回一个服从该分布的int型随机数。

```
In [16]: # 在[1, 10)之间离散均匀抽样，数组形状为3行2列
a = np.random.randint(1, 10, (3, 2))
print(a)
# 不提供size
b = np.random.randint(1, 10)
print(b)
# 指定dtype
c = np.random.randint(1, 10, dtype=np.int64)
print(c)
type(c)

[[6 4]
 [6 5]
 [4 8]]
6
8
```

Out[16]: numpy.int64



### 6、numpy.random.random(size=None)

**作用：**返回从[0, 1)之间**均匀抽样**的数组，size指定形状。

**参数：**size: int型或int型的元组，如果不提供则返回一个服从该分布的随机数

**返回值：**float型或者float型的ndarray对象

```
In [17]: # [0, 1)之间的均匀抽样, 3行2列
a = np.random.random((3, 2))
print(a)
# 不指定size
b = np.random.random()
print(b)
```

```
[[0.04220128 0.21376208]
 [0.98318818 0.43036825]
 [0.09173696 0.31713883]]
0.3378327714198529
```

09

# 读写文件（参考）

——写入CSV  
——读入CSV



### 将数据写入csv

CSV文件是一种常见的文件格式，用来存储批量数据（一维二维）

将数据写入CSV文件的方法：

```
np.savetxt(fname,array,fmt='%.18e',delimiter=None)
```

fname：文件、字符串或产生器,可以是.gz 或.bz2的压缩文件

array：存入文件的数组

fmt：写入文件的格式,例如：%d %.2f %.18e

delimiter：分割字符串,默认是任何空格。

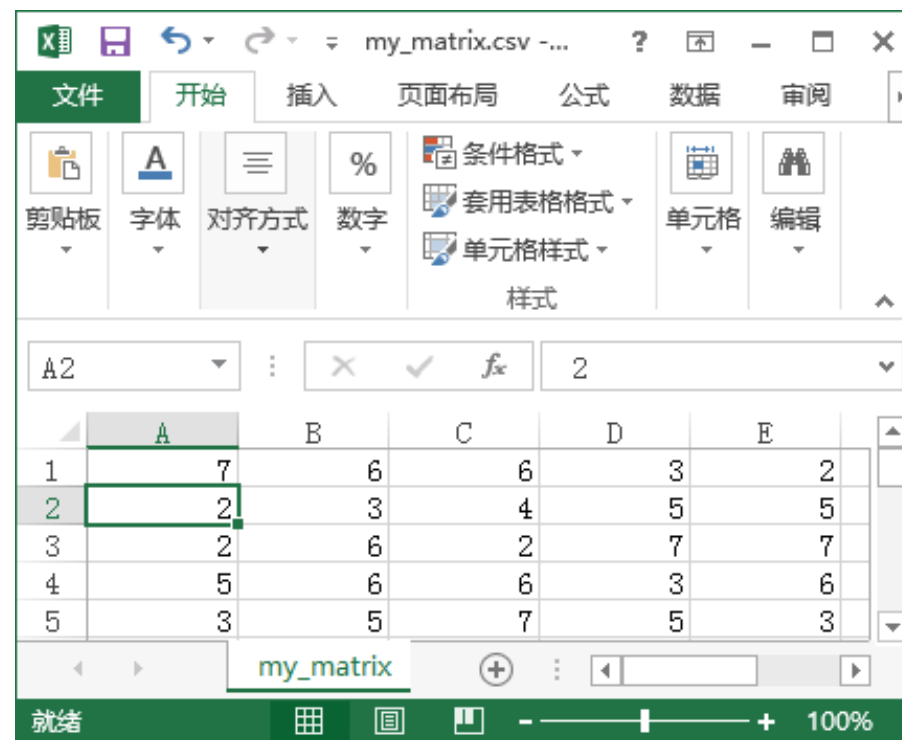




### 将数据写入csv

```
In [7]: import numpy as np
my_matrix = np.mat(np.random.randint(2, 8, size=(5, 5)))
print (my_matrix)
np.savetxt(r"C:\Users\Administrator\Desktop\my_matrix.csv", my_matrix, fmt = "%d", delimiter=",")
```

```
[[7 6 6 3 2]
 [2 3 4 5 5]
 [2 6 2 7 7]
 [5 6 6 3 6]
 [3 5 7 5 3]]
```



	A	B	C	D	E
1	7	6	6	3	2
2	2	3	4	5	5
3	2	6	2	7	7
4	5	6	6	3	6
5	3	5	7	5	3



### 从csv读取数据

使用 loadtxt函数读取csv文件

`np.loadtxt(filepath, dtype, delimiter, usecols, unpack, skiprows):`

`filepath`:加载文件路径

`dtype`:读入的数据类型

`delimiter`:加载文件分隔符

`usecols`:加载数据文件中列索引

`unpack`:当加载多列数据时是否需要将数据列进行解耦赋值给不同的变量

`skiprows`:可以跳过前几行数据，常用来跳过表头信息



### 从csv读取数据

```
In [8]: import numpy as np
#以整型的格式读入第2列和第4列的数据, 跳过第一行, 分隔符为, 号, 分别赋值给my_matrix4, my_matrix5
my_matrix4, my_matrix5 = np.loadtxt(r'C:\Users\Administrator\Desktop\my_matrix.csv',
                                     delimiter=',', usecols=(2, 4), unpack=True, skiprows=1)

print (my_matrix4)
print (my_matrix5)

[4.  2.  6.  7.]
[5.  7.  6.  3.]
```