在矩阵代数与随机向量中
使用numpy所需的知识点

一、文档

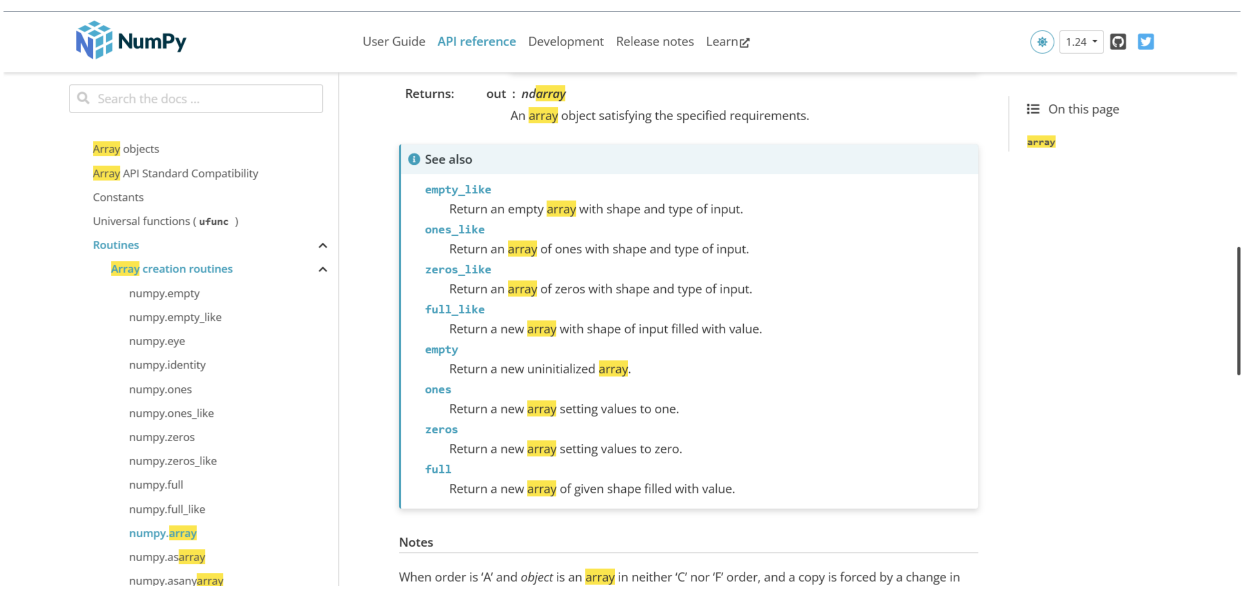
学习一门语言或者库最好的学习资料必须是它的官方文档。

> Numpy的官方文档：[Numpy的官方文档(英文)]()
>
> Numpy的中文文档：[Numpy的中文文档]()

以中文官方文档为里，当我们在新接触或者不熟悉某个库的用法时，可以在其官方文档中，找到***文章***、***Numpy 介绍***、***快速入门教程***以及***Numpy 基础知识***，这四个地方可以让我们快速入门Numpy。

其中的参考手册则能让我们查阅到某些函数的具体参数及含义。

当你在查找某些函数时，页面会出现实现功能相近的函数，我们可以利用这点来实现在不知道某些函数名，但是知道实现的结果，此时我们就可以查找与我们想要功能相近的函数。



二、入门操作

1. 常用的模块及导入

> 首先我们需要导入 ***Numpy*** 库
>
> 其次是在前两章的学习中我们最常用到的一个模块，线性代数模块：***linalg***

```
import numpy as np
import numpy.linalg as LA
```

> 在使用的过程中可以 ***help()*** 查找模块和函数的具体方法

```
help(numpy.array)   # 在已有导入包时
help('numpy.array') # 在没有导入包时
```

> 或者使用 ***dir()*** 来查看模块和函数的方法有哪些

```
dir(numpy.array)    # 在已有导入包时
dir('numpy.array')  # 在没有导入包时
```

2. 创建数组

2.1 普通数组

> ***numpy.array***的初始化为：

```
array(object, dtype=None, *, copy=True, order='K', subok=False, ndmin=0, like=None)
```

> 这里讲几个常用的参数（除了 ***object*** 其他都为可选参数）：
>
> 1. 其中 ***object*** 可以是一个有序数列如：列表、元组、字符串
> 2. ***dtype*** 可设置数组的数据类型
> 3. ***order*** 的参数可以简单理解为 $\begin{cases} 'K'\text{：保留原数据的存储格式} \\ 'A'\text{：原数据为列则按列存储否则按行} \\ 'C'\text{：按行存储} \\ 'F'\text{：按列存储} \end{cases}$
>
> 若感兴趣可以参考这篇文章：[深入理解numpy库中的order参数*numpy order*我是个烧饼啊的博客-CSDN博客](#))
>
> 5. ***ndim*** 的参数可以设置 ***ndarray*** 对象的维度
>
> **注意**：一维向量既为行向量又为列向量

Example

```
>>> np.array([1, 2, 3])
array([1, 2, 3])
```

```
>>> np.array([1, 2, 3.0])
array([ 1.,  2.,  3.])
```

```
>>> np.array([[1, 2], [3, 4]])
array([[1, 2],
       [3, 4]])
```

```
>>> np.array([1, 2, 3], ndmin=2)
array([[1, 2, 3]])
```

```
>>> np.array([1, 2, 3], dtype=complex)
array([ 1.+0.j,  2.+0.j,  3.+0.j])
```

```
>>> x = np.array([(1,2),(3,4)],dtype=[('a','<i4'),('b','<i4')])
>>> x['a']
array([1, 3])
```

```
>>> np.array(np.mat('1 2; 3 4'))
array([[1, 2],
       [3, 4]])
```

2.2 特殊数组

2.2.1 对角矩阵

> ***numpy.diag***的初始化为：

```
numpy.diag(v, k=0)
```

> 其中 ***k*** 为可选参数，设置对角线的位置 ***>0*** 为主对角线上方， ***<0*** 为主对角线下方， ***v*** 为N维的数组，也可为单位矩阵的阶数（则生成N阶对角矩阵）
>
> 描述：提取对角线或构造对角线数组

> **注意**：
>
> 当 ***array***是一个**一维**数组时，结果形成一个以一维数组为对角线元素的矩阵
>
> 当 ***array***是一个**二维**矩阵时，结果输出矩阵的对角线元素

Example

```
>>> x = np.arange(9).reshape((3,3))
>>> x
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

```
>>> np.diag(x)
array([0, 4, 8])
>>> np.diag(x, k=1)
array([1, 5])
>>> np.diag(x, k=-1)
array([3, 7])
```

```
>>>np.diag(np.diag(x))
array([[0, 0, 0],
       [0, 4, 0],
       [0, 0, 8]])
```

2.2.2 单位矩阵

实现单位矩阵有两种方式，一种是 **2.2.1** 中的 ***numpy.diag*** 还有一种则是以下这种方式。

> ***numpy.eye***的初始化为:

```
numpy.eye(N, M=None, k=0, dtype=<class 'float'>, order='C', *, like=None)
```

> 其中除了 ***N***，其他全为可选参数，经常使用 ***N***，作为单位矩阵的长度

Example

```
>>> np.eye(2, dtype=int)
array([[1, 0],
       [0, 1]])
>>> np.eye(3, k=1)
array([[0.,  1.,  0.],
       [0.,  0.,  1.],
       [0.,  0.,  0.]])
```

2.2.2 全零数组

> ***numpy.zeros***的初始化为:

```
numpy.zeros(shape, dtype=float, order='C', *, like=None)
```

> 其中除了 ***shape*** 其他都为可选参数
>
> 1. ***shape*** 为数组的各个维度上的大小，例如***(2, 3)*** 或者 ***2***
> 2. ***dtype*** 为设置数组中元素的数据类型
> 3. ***order***， 默认为 'C' $\begin{cases}'C'：按行存储\\'F'：按列存储\end{cases}$

Example

```
>>> np.zeros(5)
array([ 0.,  0.,  0.,  0.,  0.])
```

```
>>> np.zeros(5)
array([ 0.,  0.,  0.,  0.,  0.])
```

```
>>> np.zeros((5,), dtype=int)
array([0, 0, 0, 0, 0])
```

```
>>> np.zeros((2, 1))
array([[ 0.],
       [ 0.]])
```

```
>>> s = (2,2)
>>> np.zeros(s)
array([[ 0.,  0.],
       [ 0.,  0.]])
```

2.2.3 全1数组

> ***numpy.ones*** 的初始化为：

```
numpy.ones(shape, dtype=None, order='C', *, like=None)
```

> 其中除了***shape*** 其他都为可选参数
>
> 1. ***shape***为数组的各个维度上的大小，例如***(2, 3)*** 或者 ***2***
> 2. ***dtype***为设置数组中元素的数据类型
> 3. ***order***，默认为 'C' $\begin{cases}'C'：按行存储\\'F'：按列存储\end{cases}$

Example

```
>>> np.ones(5)
array([1., 1., 1., 1., 1.])
```

```
>>> np.ones((5,), dtype=int)
array([1, 1, 1, 1, 1])
```

```
>>> np.ones((2, 1))
array([[1.],
       [1.]])
```

```
>>> s = (2,2)
>>> np.ones(s)
array([[1.,  1.],
       [1.,  1.]])
```

3. 查看数组的属性

3.1 长度

> 当 ***array*** 为一维数组时，返回值为该数组的**长度**
>
> 当 ***array*** 为N维数组时，返回值为该数组的**维度**

```
len(array)
```

3.2 维度

3.2.1 array.ndim

> 使用 ***array.ndim*** 返回的是 ***array*** 数组的维度（一个数）

```
arrray.ndim
```

Example

```
>>> x = np.array([1, 2, 3])
>>> x.ndim
1
```

```
>>> y = np.zeros((2, 3, 4))
>>> y.ndim
3
```

3.2.2 array.shape

> 使用 ***array.shape*** 返回的是 ***array*** 数组各个维度的维数，用一个元组来表示
>
> 也可以跟 **4.2.1** 中的 ***array.reshape*** 一样更改 ***array*** 对象各个维度上的大小，但是当各个维度上的大小相乘**不等于**原来的大小，则会更改失败

```
array.shape
```

Example

```
>>> x = np.array([1, 2, 3, 4])
>>> x.shape
(4,)

>>> y = np.zeros((2, 3, 4))
>>> y.shape
(2, 3, 4)

>>> y.shape = (3, 8)
>>> y
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]])

>>> y.shape = (3, 6)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: total size of new array must be unchanged
np.zeros((4,2))[::2].shape = (-1,)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: Incompatible shape for in-place modification. Use
`.reshape()` to make a copy with the desired shape.
```

3.3 大小

> 返回值为 ***array*** 数组各个维度上的维数相乘

```
ndarray.size
```

Example

```
>>> x = np.zeros((3, 5, 2), dtype=np.complex128)
>>> x.size
30
```

3.4 类型

> ***ndarray***对象中元素的数据类型

```
ndarray.dtype
```

> ***ndarray***对象的数据类型

```
type(array)
```

Example

```
>>> x
array([[0, 1],
       [2, 3]])
>>> x.dtype
dtype('int32')
>>> type(x.dtype)
<type 'numpy.dtype'>
```

4. 设置数组属性

4.1 维度

4.1.1 reshape

> ***numpy.reshape***的初始化为:

```
numpy.reshape(a, newshape, order='C')
```

> 其中***reshape*** 的参数有:
>
> 1. ***a*** 为***array*** 数组
> 2. ***newshape*** 为新***array*** 的大小，用一个元组表示
> 3. ***order*** (可选)的参数有 $\begin{cases} 'A'\text{：原数据为列则按列存储否则按行} \\ 'C'\text{：按行存储} \\ 'F'\text{：按列存储} \end{cases}$
>
> **注意：** 当各个维度上的大小相乘**不等于**原来的大小，则会更改失败

> 有返回值，所谓有返回值，即不对原始多维数组进行修改

Example

```
>>> a = np.array([[1,2,3], [4,5,6]])
>>> np.reshape(a, 6)
array([1, 2, 3, 4, 5, 6])
```

```
>>> np.reshape(a, 6, order='F')
array([1, 4, 2, 5, 3, 6])
```

> ***ndarray.reshape*** 的初始化为：

```
ndarray.reshape(shape, order='C')
```

4.1.2 resize

> ***numpy.resize*** 的初始化为：

```
numpy.resize(a, new_shape)
```

> 其中 ***a*** 为数组对象
>
> ***new_shape*** 为更改后的数组各个维度上的大小
>
> **注意：** 无返回值，所谓无返回值，即会对原始多维数组进行修改

Example

```
>>> a=np.array([[0,1],[2,3]])
>>> np.resize(a,(2,3))
array([[0, 1, 2],
       [3, 0, 1]])

>>> np.resize(a,(1,4))
array([[0, 1, 2, 3]])

>>> np.resize(a,(2,4))
array([[0, 1, 2, 3],
       [0, 1, 2, 3]])
```

4.2 方向

> *ndarray.T*

> 对 *ndarray* 数组进行转置
>
> **原理**：将第一行变为第一列，将第二行变为第二列，以此类推

Example

```
>>> a = np.array([[1, 2], [3, 4]])
>>> a
array([[1, 2],
       [3, 4]])
>>> a.T
array([[1, 3],
       [2, 4]])
```

```
>>> a = np.array([1, 2, 3, 4])
>>> a
array([1, 2, 3, 4])
>>> a.T
array([1, 2, 3, 4])
```

4.3 数据类型

> *ndarray.astype* 的初始化为：

```
ndarray.astype(dtype, order='K', casting='unsafe', subok=True, copy=True)
```

> 其中除了 *dtype* 其他都为可选参数
>
> 1. *dtype* 参数为需要转换成的数据类型
> 2. *oreder* 参数为$\begin{cases} 'K'\text{：保留原数据的存储格式} \\ 'A'\text{：原数据为列则按列存储否则按行} \\ 'C'\text{：按行存储} \\ 'F'\text{：按列存储} \end{cases}$
> 3. *casting* 参数为
> $$\begin{cases} 'no'\text{表示根本不应强制转换数据类型。} \\ 'equiv'\text{表示只允许字节顺序更改。} \\ 'safe'\text{表示只允许保留值的强制转换。} \\ 'same_kind'\text{表示仅安全施放或同类内的施法，像}float64\text{到}float32\text{，是允许的。} \\ 'unsafe'\text{是指可以进行任何数据转换} \end{cases}$$

Example

```
>>> x = np.array([1, 2, 2.5])
>>> x
array([1. ,  2. ,  2.5])
```

```
>>> x.astype(int)
array([1, 2, 2])
```

5. 矩阵内的元素级运算

5.1 对轴的理解

> 对简单的二维矩阵，沿着竖轴方向的为0轴，沿着横轴方向的为1轴。

5.2 沿轴求和，求均值

5.2.1 sum

> ***numpy.sum*** 的初始化为：

```
numpy.sum(a, axis=None, dtype=None, out=None, keepdims=<no value>, initial=<no value>, where=<no value>)
```

> 其中除了 ***a*** 其他都为可选参数，这里讲一个常用的可选参数
>
> 1. ***a*** 为需要求和的数组
> 2. ***axis*** 为轴，需要沿哪条轴进行计算求和，若不加这个参数，则求和变为求该数组的内部累加和

Example

```
>>> np.sum([0.5, 1.5])
2.0
>>> np.sum([0.5, 0.7, 0.2, 1.5], dtype=np.int32)
1
>>> np.sum([[0, 1], [0, 5]])
6
>>> np.sum([[0, 1], [0, 5]], axis=0)
array([0, 6])
>>> np.sum([[0, 1], [0, 5]], axis=1)
array([1, 5])
>>> np.sum([[0, 1], [np.nan, 5]], where=[False, True], axis=1)
array([1., 5.])
```

> ***ndarray.sum*** 的初始化为：

```
ndarray.sum(axis=None, dtype=None, out=None, keepdims=False, initial=0, where=True)
```

5.2.2 mean

> ***numpy.mean*** 的初始化为：

```
numpy.mean(a, axis=None, dtype=None, out=None, keepdims=<no value>, *, where=<no value>)[sourc
```

> 其中除了 ***a*** 其他都为可选参数，这里讲一个常用的可选参数
>
> 1. ***a*** 为需要求均值的数组
> 2. ***axis*** 为轴，需要沿哪条轴进行计算求均值，若不加这个参数，则求和变为求该数组的内部总的均值

Example

```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.mean(a)
2.5
>>> np.mean(a, axis=0)
array([2., 3.])
>>> np.mean(a, axis=1)
array([1.5, 3.5])
```

5.3 通用函数

5.3.1 广播机制broadcast

> 官方文档的描述为：受某些限制， 较小的阵列在较大的阵列上"广播"，以便它们 具有兼容的形状。
>
> 我的理解是从低维的角度对高维进行操作

官方文档关于这一部分的说明在[广播 — NumPy v1.24 手册]

如果看不懂官方文档的可以参考《利用Python进行数据分析》这本书的P435-P442，或者[numpy的广播机制]这篇文章

5.3.2 sqrt

> ***numpy.sqrt*** 的初始化为：

```
numpy.sqrt(x, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj]) = <ufunc 'sqrt'>
```

> 虽然看上去很复杂但是实际上目前经常用到的也只有***x***

> *np.sqrt(ndarray)* 即是对 *ndarray* 中的每一个元素进行求开平方

Example

```
>>> np.sqrt([1,4,9])
array([ 1.,  2.,  3.])
>>> np.sqrt([4, -1, -3+4J])
array([ 2.+0.j,  0.+1.j,  1.+2.j])
>>> np.sqrt([4, -1, np.inf])
array([ 2., nan, inf])
```

5.3.3 multiply

> *numpy.multiply* 的初始化为：

```
numpy.multiply(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K',
dtype=None, subok=True[, signature, extobj]) = <ufunc 'multiply'>
```

> 我们只需用到 *x1* 与 *x2*，这两个参数为两个需要相乘的数组， 其用法与 * 相同，计算的是元素级的运算

Example

```
>>> np.multiply(2.0, 4.0)
8.0
```

```
>>> x1 = np.arange(9.0).reshape((3, 3))
>>> x2 = np.arange(3.0)
>>> np.multiply(x1, x2)
array([[  0.,   1.,   4.],
       [  0.,   4.,  10.],
       [  0.,   7.,  16.]])
```

```
>>> x1 = np.arange(9.0).reshape((3, 3))
>>> x2 = np.arange(3.0)
>>> x1 * x2
array([[  0.,   1.,   4.],
       [  0.,   4.,  10.],
       [  0.,   7.,  16.]])
```

6. 矩阵间的运算

6.1 矩阵间的加减

> 这一部分的普遍用法是使用一个函数或者一个符号进行运算

Example

```
>>> np.add(1.0, 4.0)
5.0
>>> x1 = np.arange(9.0).reshape((3, 3))
>>> x2 = np.arange(3.0)
>>> np.add(x1, x2)
array([[  0.,   2.,   4.],
       [  3.,   5.,   7.],
       [  6.,   8.,  10.]])
```

```
>>> x1 = np.arange(9.0).reshape((3, 3))
>>> x2 = np.arange(3.0)
>>>x1 + x2
array([[ 0.,  2.,  4.],
       [ 3.,  5.,  7.],
       [ 6.,  8., 10.]])
```

> 从上面的例子可以看出，不管是使用函数 ***numpy.add*** 还是使用 **操作符+**， 这两种计算的结果都是相同的。

> 减法则如以下这些例子

Example

```
>>> np.subtract(1.0, 4.0)
-3.0
```

```
>>> x1 = np.arange(9.0).reshape((3, 3))
>>> x2 = np.arange(3.0)
>>> np.subtract(x1, x2)
array([[ 0.,  0.,  0.],
       [ 3.,  3.,  3.],
       [ 6.,  6.,  6.]])
```

```
>>> x1 = np.arange(9.0).reshape((3, 3))
>>> x2 = np.arange(3.0)
>>> x1 - x2
array([[0., 0., 0.],
       [3., 3., 3.],
       [6., 6., 6.]])
```

6.2 矩阵间的乘法

> ***numpy.dot*** 计算的是矩阵的内积即是对两个矩阵进行线性代数的计算，其与使用 **@** 结果是相同的

Example

```
>>> np.dot(3, 4)
12
```

```
>>> np.dot([2j, 3j], [2j, 3j])
(-13+0j)
```

```
>>> a = [[1, 0], [0, 1]]
>>> b = [[4, 1], [2, 2]]
>>> np.dot(a, b)
array([[4, 1],
       [2, 2]])
```

> 也可以使用其方法 ***ndarray.dot()***

7. 线性代数模块的使用

7.1 矩阵的逆

> ***numpy.linalg.inv*** 的初始化为：

```
linalg.inv(a)
```

Example

```
>>> a = np.array([[[1., 2.], [3., 4.]], [[1, 3], [3, 5]]])
>>> inv(a)
array([[[-2.  ,  1.  ],
        [ 1.5 , -0.5 ]],
       [[-1.25,  0.75],
        [ 0.75, -0.25]]])
```

7.2 求矩阵的行列式

> ***numpy.linalg.det*** 的初始化为：

```
linalg.det(a)
```

Example

```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.linalg.det(a)
-2.0
```

```
>>> a = np.array([ [[1, 2], [3, 4]], [[1, 2], [2, 1]], [[1, 3], [3, 1]] ])
>>> a.shape
(3, 2, 2)
>>> np.linalg.det(a)
array([-2., -3., -8.])
```

7.3 求线性矩阵方程的解

> ***numpy.linalg.solve*** 的初始化为：

```
linalg.solve(a, b)
```

> 其中 ***a*** 为系数矩阵，***b*** 为结果矩阵，返回值为解矩阵

```
>>> a = np.array([[1, 2], [3, 5]])
>>> b = np.array([1, 2])
>>> x = np.linalg.solve(a, b)
>>> x
array([-1.,  1.])
```

7.3 求矩阵的特征值和特征向量及验证

> ***numpy.linalg.eig*** 的初始化为：

```
linalg.eig(a)
```

Example

```
>>> import numpy.linalg as LA
>>> w, v = LA.eig(np.diag((1, 2, 3)))
>>> w; v
array([1., 2., 3.])
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

> 其中第一个数组为**特征值**，第二个数组，每列都是一个与第一个数组对应的**特征向量**，如***1***的特征向量为***[1, 0, 0]***

验证：$\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$

> 其中$\boldsymbol{A}$ 为p 阶方阵，λ 为$\boldsymbol{A}$ 的特征值，称$\boldsymbol{x}$ 为$\boldsymbol{A}$ 的属于特征值λ 的一个特征向量
>
> **注意：** 因为在矩阵计算时，计算机会因为精度问题，导致小数点后很多位会出现误差，所以我们在比较时就使用 ***numpy.allclose***

7.4 求矩阵的奇异值分解

> ***numpy.linalg.svd*** 的初始化为：

```
linalg.svd(a, full_matrices=True, compute_uv=True, hermitian=False)
```

> 其中，我们只需要将需要奇异值分解的矩阵当作 ***a*** 参数放入函数中即能运行分解

Example

```
>>> import numpy.linalg as LA
>>> A = np.array([1, 1, 2, -2, 2, 2]).reshape((2,3), order="F")
>>> A
array([[ 1,  2,  2],
       [ 1, -2,  2]])

>>> svd(A)
(array([[-0.70710678, -0.70710678],
        [-0.70710678,  0.70710678]]),
 array([3.16227766, 2.82842712]),
 array([[-4.47213595e-01, -4.15130595e-16, -8.94427191e-01],
        [ 7.32343785e-17, -1.00000000e+00,  3.39211021e-16],
        [-8.94427191e-01,  6.18327757e-17,  4.47213595e-01]]))
```