

15

Basic Computations in NumPy

NumPy 常见运算

使用 NumPy 完成算术、代数、统计运算



生活只有两件好事：发现数学和教数学。

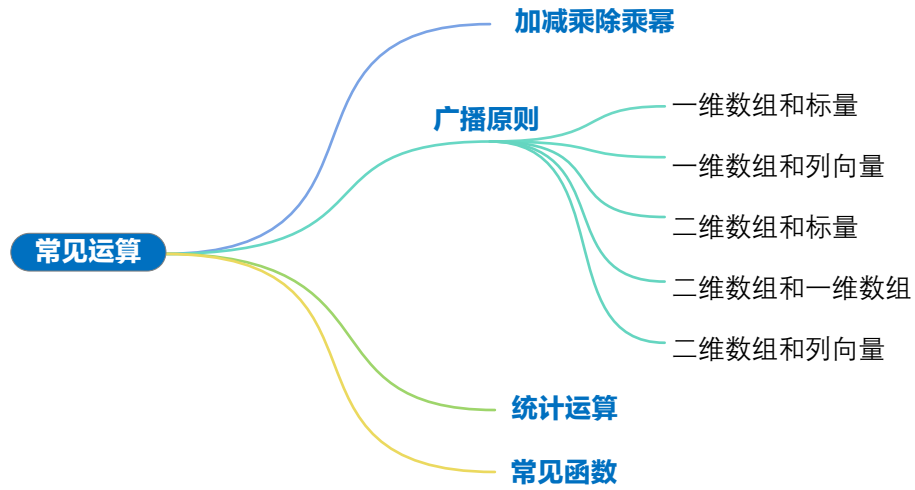
Life is good for only two things: discovering mathematics and teaching mathematics.

—— 西梅翁·德尼·泊松 (Siméon Denis Poisson) | 法国数学家 | 1781 ~ 1840



- ◀ `numpy.abs()` 计算绝对值、复数模
- ◀ `numpy.add()` 加法运算
- ◀ `numpy.argmax()` 返回数组中最大元素的索引
- ◀ `numpy.argmin()` 返回数组中最小元素的索引
- ◀ `numpy.array()` 创建 array 数据类型
- ◀ `numpy.average()` 计算数组元素的加权平均值
- ◀ `numpy.broadcast_to()` 用于将数组广播到指定的形状
- ◀ `numpy.corrcoef()` 计算数组中元素的协方差矩阵，自由度 `ddof` 没有影响
- ◀ `numpy.cos()` 计算余弦值
- ◀ `numpy.cov()` 计算数组中元素的协方差矩阵，默认自由度 `ddof` 为 0
- ◀ `numpy.divide()` 除法运算
- ◀ `numpy.exp()` 对数组中的每个元素进行指数运算
- ◀ `numpy.maximum()` 逐元素地比较两个数组，并返回元素级别上的较大值组成的新数组
- ◀ `numpy.multiply()` 乘法运算
- ◀ `numpy.power()` 乘幂运算
- ◀ `numpy.random.multivariate_normal()` 用于生成多元正态分布的随机样本
- ◀ `numpy.random.randint()` 在指定范围内产生随机整数
- ◀ `numpy.random.uniform()` 产生满足连续均匀分布的随机数
- ◀ `numpy.reshape()` 用于将数组重新调整为指定的形状
- ◀ `numpy.sin()` 计算正弦值
- ◀ `numpy.std()` 计算数组中元素的标准差，默认自由度 `ddof` 为 0
- ◀ `numpy.subtract()` 减法运算
- ◀ `numpy.var()` 计算数组中元素的方差，默认自由度 `ddof` 为 0
- ◀ `sklearn.datasets.load_iris` 导入鸢尾花数据





15.1 加减乘除乘幂

在 NumPy 中，基本的加减乘除、乘幂运算如下：

- ▶ 加法：使用 `+` 运算符或 `numpy.add()` 函数实现。
- ▶ 减法：使用 `-` 运算符或 `numpy.subtract()` 函数实现。
- ▶ 乘法：使用 `*` 运算符或 `numpy.multiply()` 函数实现。
- ▶ 除法：使用 `/` 运算符或 `numpy.divide()` 函数实现。
- ▶ 乘幂：使用 `**` 运算符或 `numpy.power()` 函数实现。

下面，我们先聊一聊相同形状的数组之间的加减乘除乘幂运算。



本章配套的 Jupyter Notebook 文件主要是 Bk1_Ch15_01.ipynb。请大家一边阅读本章，一边在 JupyterLab 中实践。

一维数组

图 1 所示为两个等长度一维数组之间的加、减、乘、除、乘幂运算。这一组运算都是逐项完成，也就是对应位置完成运算。

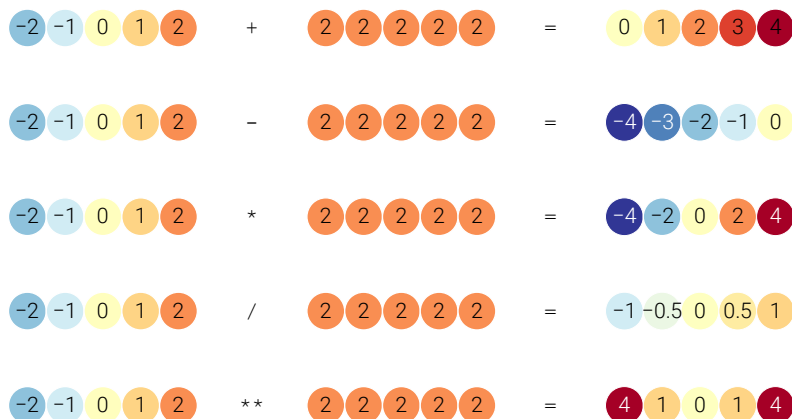


图 1. 一维数组加、减、乘、除、乘幂

二维数组

图 2 所示为二维数组之间的加、减、乘、除、乘幂运算。类似运算也可以用在三维、多维数组上。

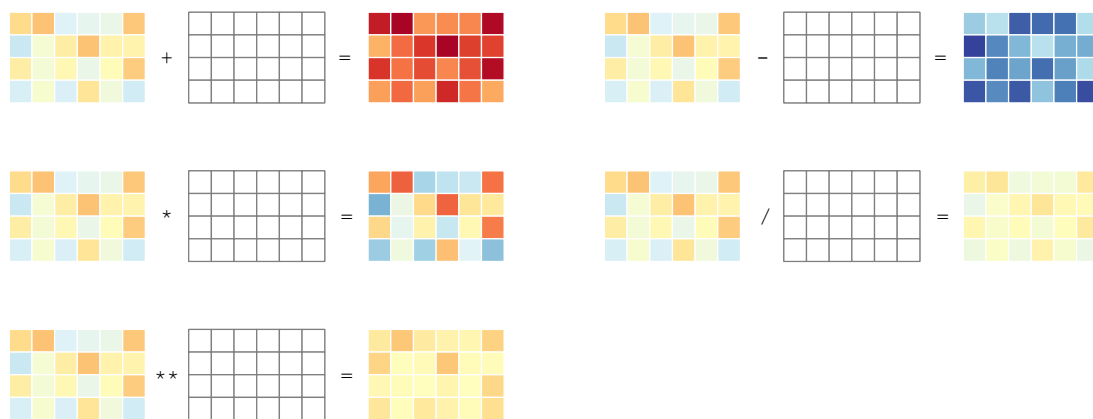


图 2. 二维数组加、减、乘、除、乘幂，空白网格代表矩阵的每个元素均为 2

15.2 广播原则

简单来说，NumPy 的广播原则（broadcasting）指定了不同形状的数组之间的算术运算规则，将形状较小的数组扩展为与形状较大的数组相同，再进行运算，以提高效率。

下面，我们首先以一维数组为例介绍什么是广播原则。

一维数组和标量

图 3 所示一维数组和标量之间完成加、减、乘、除、乘幂运算，大家可以发现图 3 可以替代



图 3. 一维数组和标量加、减、乘、除、乘幂，广播原则

一维数组和列向量

图 4 和图 5 所示为将广播原则用在一维数组和列向量的加法和乘法上。广播过程相当于把一维数组 (5,) 展成 (3, 5) 二维数组，把列向量 (3, 1) 也展成 (3, 5) 二维数组。运算结果也是二维数组。

这两幅图中，大家还会看到，行向量、列向量之间的运算也可以获得同样的结果，请大家在 JupyterLab 中自己完成。

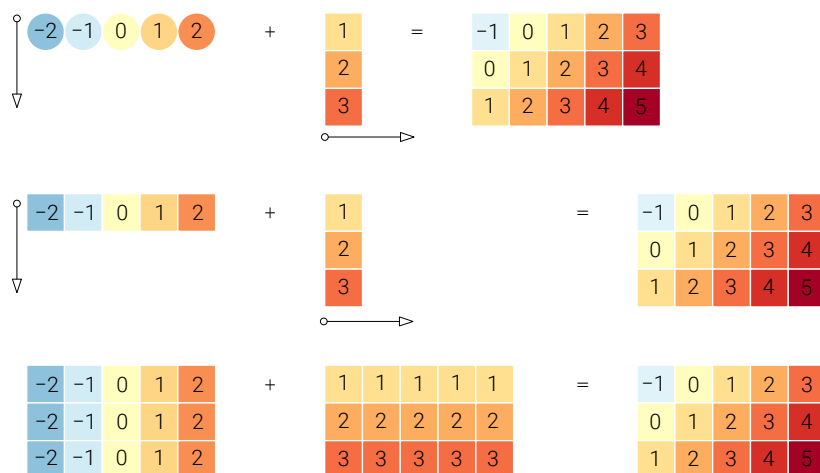


图 4. 一维数组和列向量加法，广播原则

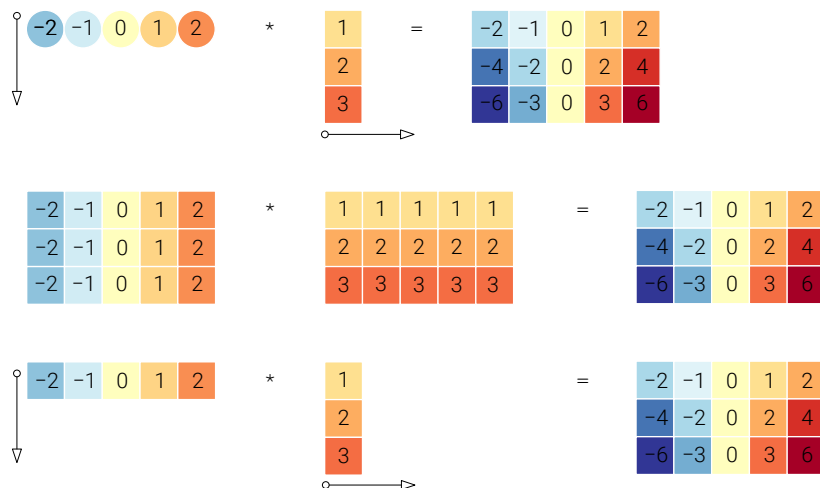


图 5. 一维数组和列向量乘法，广播原则

二维数组和标量

图 6 所示二维数组和标量的运算相当于图 2。

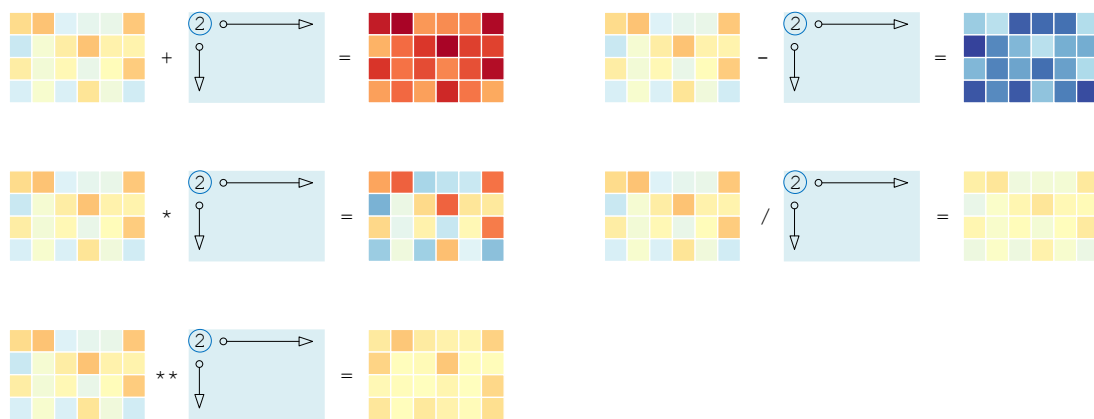


图 6. 二维数组和标量加、减、乘、除、乘幂，广播原则

二维数组和一维数组

图 7 所示为二维数组和一维数组之间的广播原则运算。二维数组的形状为 $(4, 6)$ ，一维数组的形状为 $(6,)$ 。

图 7 等价于图 8。图 8 中，行向量是二维数组，形状为 $(1, 6)$ 。

注意，当前 NumPy 不支持 $(4, 6)$ 和 $(4,)$ 之间的广播运算，会报错。这种情况，要用 $(4, 6)$ 和 $(4, 1)$ 之间的广播原则。

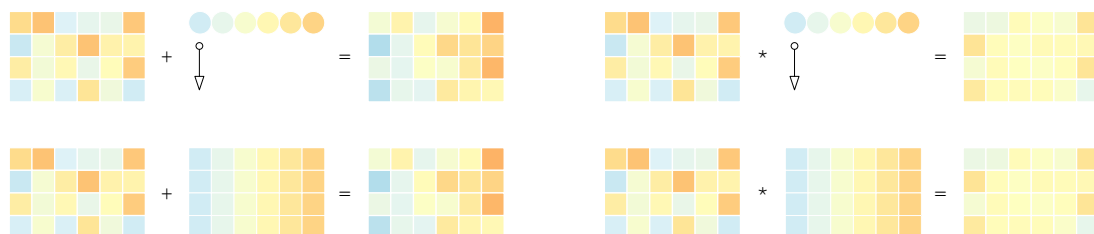


图 7. 二维数组和一维数组加、乘，广播原则

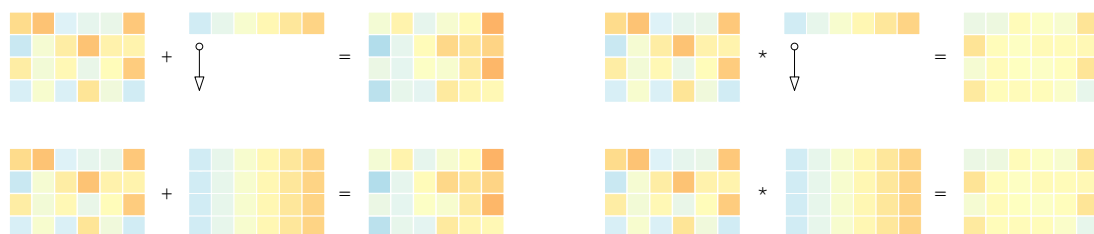


图 8. 二维数组和行向量加、乘，广播原则

二维数组和列向量

图 9 所示为二维数组和列向量之间的广播运算。二维数组的形状为 $(4, 6)$ ，列向量形状为 $(4, 1)$ 。它们在行数上匹配。

注意，这个列向量也是二维数组。

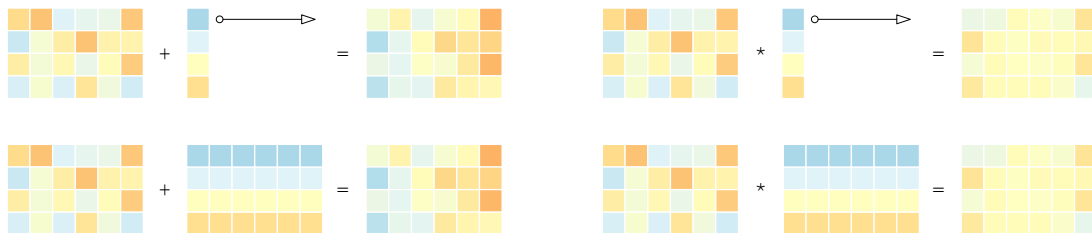


图 9. 二维数组和列向量加、乘，广播原则

15.3 统计运算

图 10 所示为求最大值的操作。给定二维数组 `A`，`A.max()` 计算整个数组中最大值。而 `A.max(axis = 0)` 在列方向计算最大值，结果为一维数组。`A.max(axis = 1)` 在行的方向上计算最大值，结果同样为一维数组。而 `A.max(axis = 1, keepdims = True)` 的结果为列向量（二维数组）。

此外，计算最小值、求和、均值、方差、标准差等统计运算遵循相同的规则，请大家参考本章 Jupyter Notebook。

⚠ 注意，计算方差、标准差时，NumPy 默认分母为 n （样本数量），而不是 $n - 1$ ；为了计算样本方差或标准差，需要设定 `ddof = 1`。

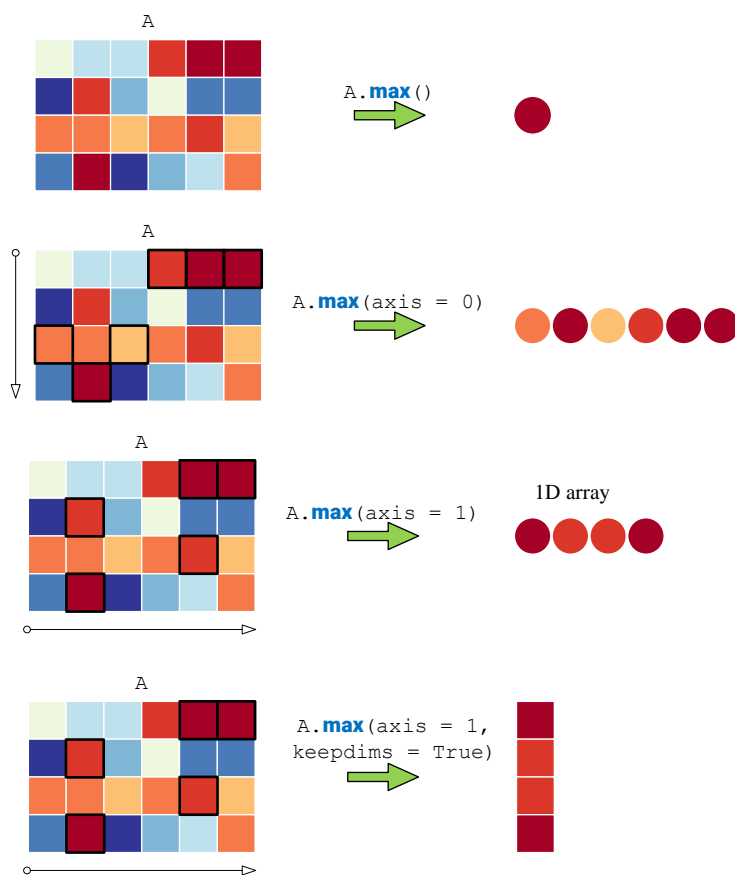


图 10. 沿不同轴求最大值



什么是方差？

方差是统计学中衡量数据分散程度的一种指标，用于衡量一组数据与其平均值之间的偏离程度。方差的计算是将每个数据点与平均值的差的平方求和，并除以数据点的个数 n 减 1，即 $n - 1$ 。方差越大，数据点相对于平均值的离散程度就越高，反之亦然。方差常用于数据分析、建模和实验设计等领域。方差开平方结果为标准差。

NumPy 还提供计算协方差矩阵、相关性系数矩阵的函数。图 11 (a) 所示为鸢尾花数据协方差矩阵，图 11 (b) 为相关性系数矩阵。

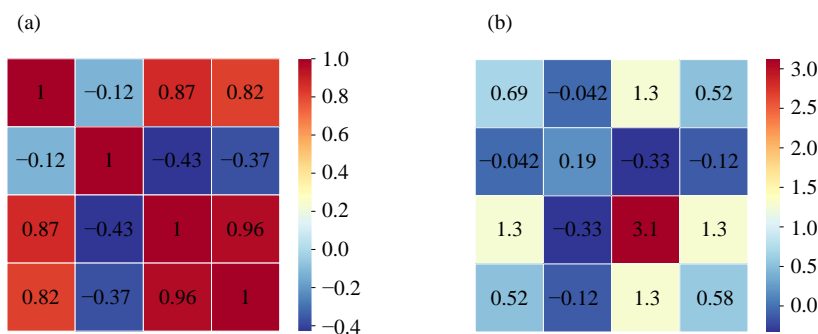


图 11. 鸢尾花数据协方差矩阵、相关性系数矩阵

图 12 完成很多有关鸢尾花数据统计计算，并绘制图 11，请大家自行学习并逐行注释。

值得一提的是，ⁱ 和 ^k 在计算协方差矩阵、相关性系数矩阵时，输入的数组形状。
iris_data_array 的每一列代表一个特征，而转置之后 iris_data_array.T 每行代表一个特征。

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris

# 导入鸢尾花数据
iris = load_iris()
iris_data_array = iris.data

a print(iris_data_array.max()) # 整个矩阵的最大值
b print(iris_data_array.max(axis = 0)) # 每列最大值
c print(np.argmax(iris_data_array, axis=0)) # 每列最大值位置
d print(iris_data_array.max(axis = 1)) # 每行最大值位置
e print(np.average(iris_data_array, axis = 0)) # 每列均值

# 计算每一列方差
f print(np.var(iris_data_array, axis = 0))
# 注意，NumPy中默认分母为n

g print(np.var(iris_data_array, axis = 0, ddof = 1))
# 将分母设为n - 1


# 计算每一列标准差
h print(np.std(iris_data_array, axis = 0))

# 计算协方差矩阵；注意转置
i SIGMA = np.cov(iris_data_array.T, ddof = 1)
print(SIGMA)

# 可视化协方差矩阵
fig, ax = plt.subplots(figsize = (5,5))
j sns.heatmap(SIGMA, cmap = 'RdYlBu_r', annot = True,
               ax = ax, fmt = ".2f", square = True,
               xticklabels = [], yticklabels = [], cbar = True)

# 计算协方差矩阵；注意转置
k CORR = np.corrcoef(iris_data_array.T)
print(CORR)

fig, ax = plt.subplots(figsize = (5,5))
l sns.heatmap(CORR, cmap = 'RdYlBu_r', annot = True,
               ax = ax, fmt = ".2f", square = True,
               xticklabels = [], yticklabels = [], cbar = True)
```

图 12. NumPy 中的统计运算； Bk1_Ch15_01.ipynb



什么是协方差矩阵？

协方差矩阵是一个方阵，其中的元素代表了数据中各个维度之间的协方差。协方差是用来衡量两个随机变量之间的关系的统计量，它描述的是两个变量的变化趋势是否相似，以及它们之间的相关性强度。协方差矩阵可以用于多变量分析和线性代数中的特征值分解、奇异值分解等计算。在机器学习领域，协方差矩阵常用于数据降维、主成分分析、特征提取等方面。



什么是相关系数矩阵？

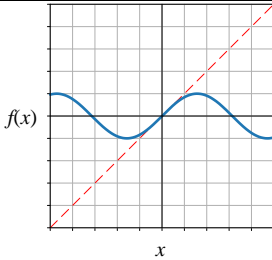
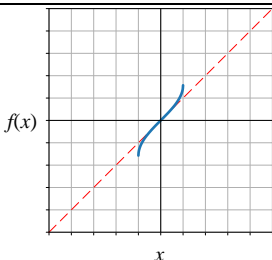
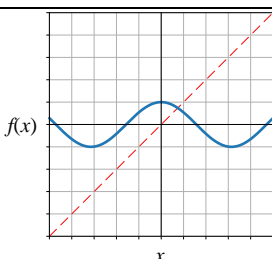
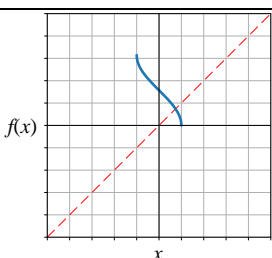
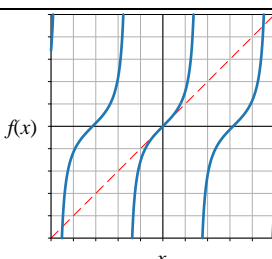
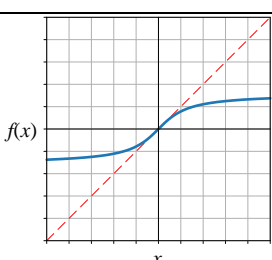
相关性系数矩阵是一个方阵，其中的元素代表了数据中各个维度之间的相关性系数。相关性系数是用来衡量两个变量之间线性关系的程度，它取值范围在-1 到 1 之间，数值越接近于 1 或-1，说明两个变量之间的线性关系越强；数值越接近于 0，说明两个变量之间的线性关系越弱或不存在。相关性系数矩阵可以用于多变量分析、线性回归等领域，通常与协方差矩阵一起使用。在机器学习领域，相关性系数矩阵常用于特征选择和数据可视化等方面。

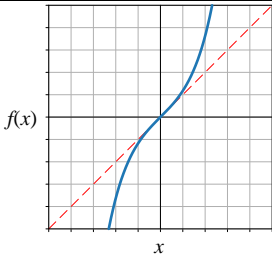
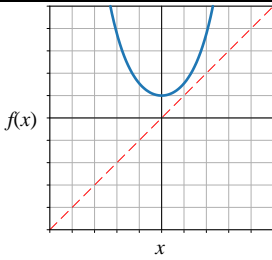
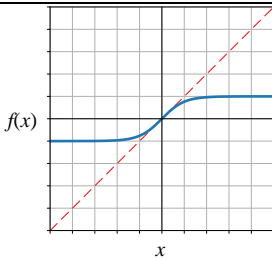
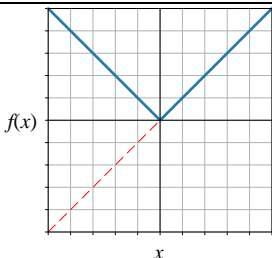
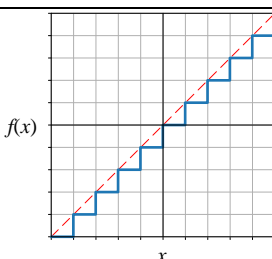
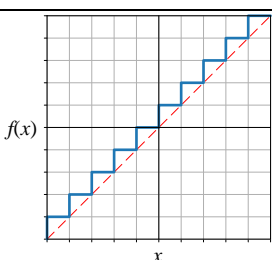
15.4 常见函数

NumPy 还提供大量常用函数。

NumPy 中还给出很多常用常数，比如 `numpy.pi`（圆周率）、`numpy.e`（欧拉数、自然底数）、`numpy.Inf`（正无穷）、`numpy.NAN`（非数）等等。

函数	NumPy 函数	图像
$f(x) = x^p$ 幂函数 (power function)	<code>numpy.power(x, 2)</code>	
	<code>numpy.power(x, 3)</code>	

$f(x) = \sin(x)$ 正弦函数 (sine function)	<code>numpy.sin()</code>	
$f(x) = \arcsin(x)$ 反正弦函数 (inverse sine function)	<code>numpy.arcsin()</code>	
$f(x) = \cos(x)$ 余弦函数 (sine function)	<code>numpy.cos()</code>	
$f(x) = \arccos(x)$ 反余弦函数 (inverse cosine function)	<code>numpy.arccos()</code>	
$f(x) = \tan(x)$ 正切函数 (tangent function)	<code>numpy.tan()</code>	
$f(x) = \arctan(x)$ 反正切函数 (inverse tangent function)	<code>numpy.arctan()</code>	

$f(x) = \sinh(x)$ 双曲正弦函数 (hyperbolic sine function)	<code>numpy.sinh()</code>	
$f(x) = \cosh(x)$ 双曲余弦函数 (hyperbolic cosine function)	<code>numpy.cosh()</code>	
$f(x) = \tanh(x)$ 双曲正切函数 (hyperbolic tangent function)	<code>numpy.tanh()</code>	
$f(x) = x $ 绝对值函数 (absolute function)	<code>numpy.abs()</code>	
$f(x) = \lfloor x \rfloor$ 向下取整函数 (floor function)	<code>numpy.floor()</code>	
$f(x) = \lceil x \rceil$ 向上取整函数 (ceil function)	<code>numpy.ceil()</code>	

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：jiang.visualize.ml@gmail.com

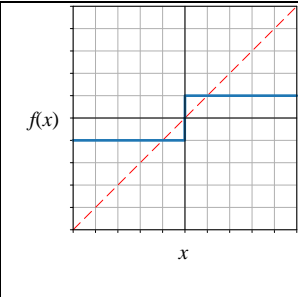
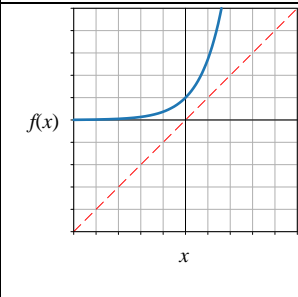
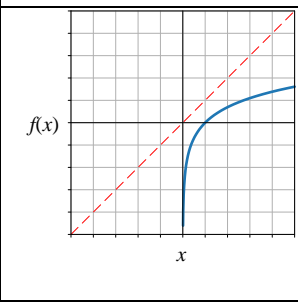
$f(x) = \text{sgn}(x)$ 符号函数 (sign function)	<code>numpy.sign()</code>	
$f(x) = \exp(x) = e^x$ 指数函数 (exponential function)	<code>numpy.exp()</code>	
$f(x) = \ln(x)$ 对数函数 (logarithmic function)	<code>numpy.log()</code>	

图 13 代码为自定义函数可视化一元函数。

```

import numpy as np
import matplotlib.pyplot as plt

# 自定义可视化函数
a def visualize_fx(x_array, f_array, title, step = False):

    fig, ax = plt.subplots(figsize = (5,5))
    b ax.plot([-5,5],[-5,5], c = 'r', ls = '--', lw = 0.5)

    if step:
    c     ax.step(x_array, f_array)
    d else:
        ax.plot(x_array, f_array)

    e ax.set_xlim(-5, 5)
      ax.set_ylim(-5, 5)
      ax.axvline(0, c = 'k')
      ax.axhline(0, c = 'k')
      ax.set_xticks(np.arange(-5, 5+1))
      ax.set_yticks(np.arange(-5, 5+1))
      ax.set_xlabel('x')
      ax.set_ylabel('f(x)')
      plt.grid(True)
    f ax.set_aspect('equal', adjustable='box')
      fig.savefig(title + '.svg', format='svg')

```


图 13. 自定义可视化函数;  Bk1_Ch15_02.ipynb

图 14 代码调用图 13 中自定义函数可视化几个一元函数，请大家自行学习并逐行注释。

```

# 幂函数, p = 2
x_array = np.linspace(-5, 5, 1001)
a f_array = np.power(x_array, 2)
visualize_fx(x_array, f_array, '幂函数_p=2')

# 反正弦函数
b x_array_ = np.copy(x_array)
c x_array_[(x_array_ < -1) | (x_array_ > 1)] = np.nan
f_array = np.arcsin(x_array_)
visualize_fx(x_array_, f_array, '反正弦函数')

# 正切函数
d f_array = np.tan(x_array)
e f_array[:-1][np.diff(f_array) < 0] = np.nan
visualize_fx(x_array, f_array, '正切函数')

# 向下取整函数
f_array = np.floor(x_array)
f visualize_fx(x_array, f_array, '向下取整函数', True)

# 对数函数
x_array_ = np.copy(x_array)
g x_array_[x_array_ <= 0] = np.nan
f_array = np.log(x_array_)
visualize_fx(x_array_, f_array, '对数函数')

```

图 14. 可视化几个一元函数，使用时配合前文代码；  Bk1_Ch15_02.ipynb



请大家完成下面 3 道题目，它们的目的都是利用 NumPy 计算并可视化公式。

Q1. 给定如下一元高斯函数，参数 $a = 1$, $b = 2$, $c = 1$ 。请用 NumPy 和 Matplotlib 线图可视化函数图像。

$$f(x) = a \exp\left(-\frac{(x-b)^2}{2c^2}\right)$$

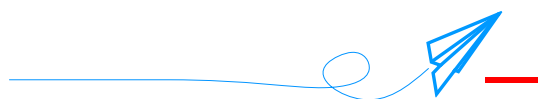
Q2. 给定如下二元高斯函数。请用 NumPy 和 Matplotlib 三维网格面可视化二元函数图像。

$$f(x_1, x_2) = \exp(-x_1^2 - x_2^2)$$

Q3. 下式为二元高斯分布的概率密度函数，请用 NumPy 和 Matplotlib 填充等高线可视化这个二元函数图像。参数具体为 $\mu_X = 0$, $\mu_Y = 0$, $\sigma_X = 1$, $\sigma_Y = 1$, $\rho_{X,Y} = 0.6$ 。

$$f(x, y) = \frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2(1-\rho^2)}\left[\left(\frac{x-\mu_X}{\sigma_X}\right)^2 - 2\rho\left(\frac{x-\mu_X}{\sigma_X}\right)\left(\frac{y-\mu_Y}{\sigma_Y}\right) + \left(\frac{y-\mu_Y}{\sigma_Y}\right)^2\right]\right)$$

* 题目答案请参考 Bk1_Ch15_03.ipynb。



本章介绍了 NumPy 中基本数学运算工具，其中包括加减乘除乘幂、广播原则、统计运算、常见函数。需要大家格外注意广播原则，它可以帮我们提高运算效率。此外，一般情况，我们更多会使用 Pandas 中的统计运算工具。

NumPy 中主力运算工具将是本书第 17、18 章要介绍的有关线性代数函数。