《Python程序设计》

Python科学计算

刘潇
机械科学与工程学院

2023年11月9日

2023秋季

本节要点

- 了解科学计算的实现过程
- 掌握Numpy模块数组对象的创建和常用方法
- 掌握Scipy和Matplotlib模块的常用方法

主要内容

1. 科学计算简介

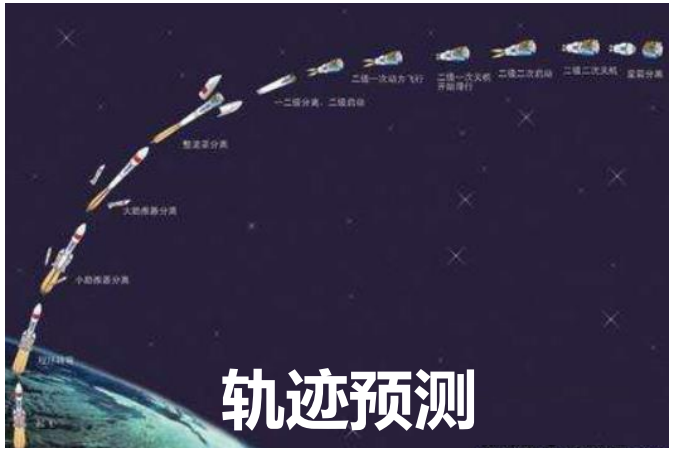
2. Numpy和Scipy

3. Matplotlib

4. 计算文件转换示例

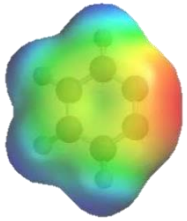
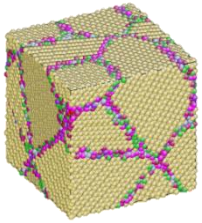
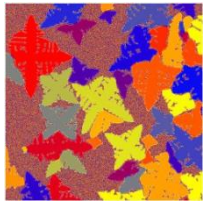
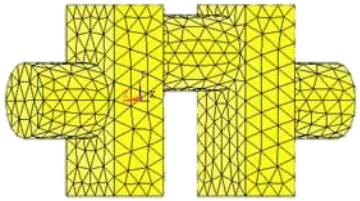
科学计算

科学计算为解决科学和工程中的数学问题利用计算机进行的数值计算，包括数学建模、计算方法求解和计算机实现三个阶段



更小尺度的计算方法

量子力学
密度泛函理论 → 蒙特卡洛方法
分子动力学 → 缺陷动力学
相变动力学
连续动力学 → 有限元
有限差分

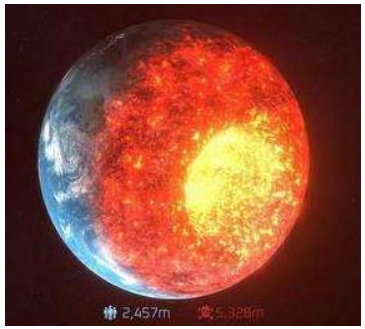
时间 / s	10^{-15}		10^{-6}	10^{-3}	10^{0}
长度 / m	10^{-10}		10^{-7}	10^{-4}	10^{0}
	纳观		微观	介观	宏观

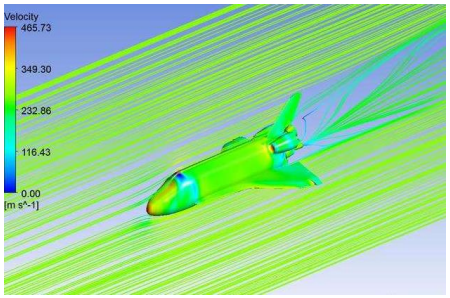
注：研究对象的层次划分并不严格，存在着交叉

计算 vs 实验

在一些特定条件下，实验方法：

1. 无法实现	在星球内核 天气预报
2. 过于危险	飞行模拟 爆炸模拟
3. 价格昂贵	高温高压模拟 风洞模拟
4. 盲区	有些过程由于发生的时间尺度和空间尺度很小，难以直接观察到

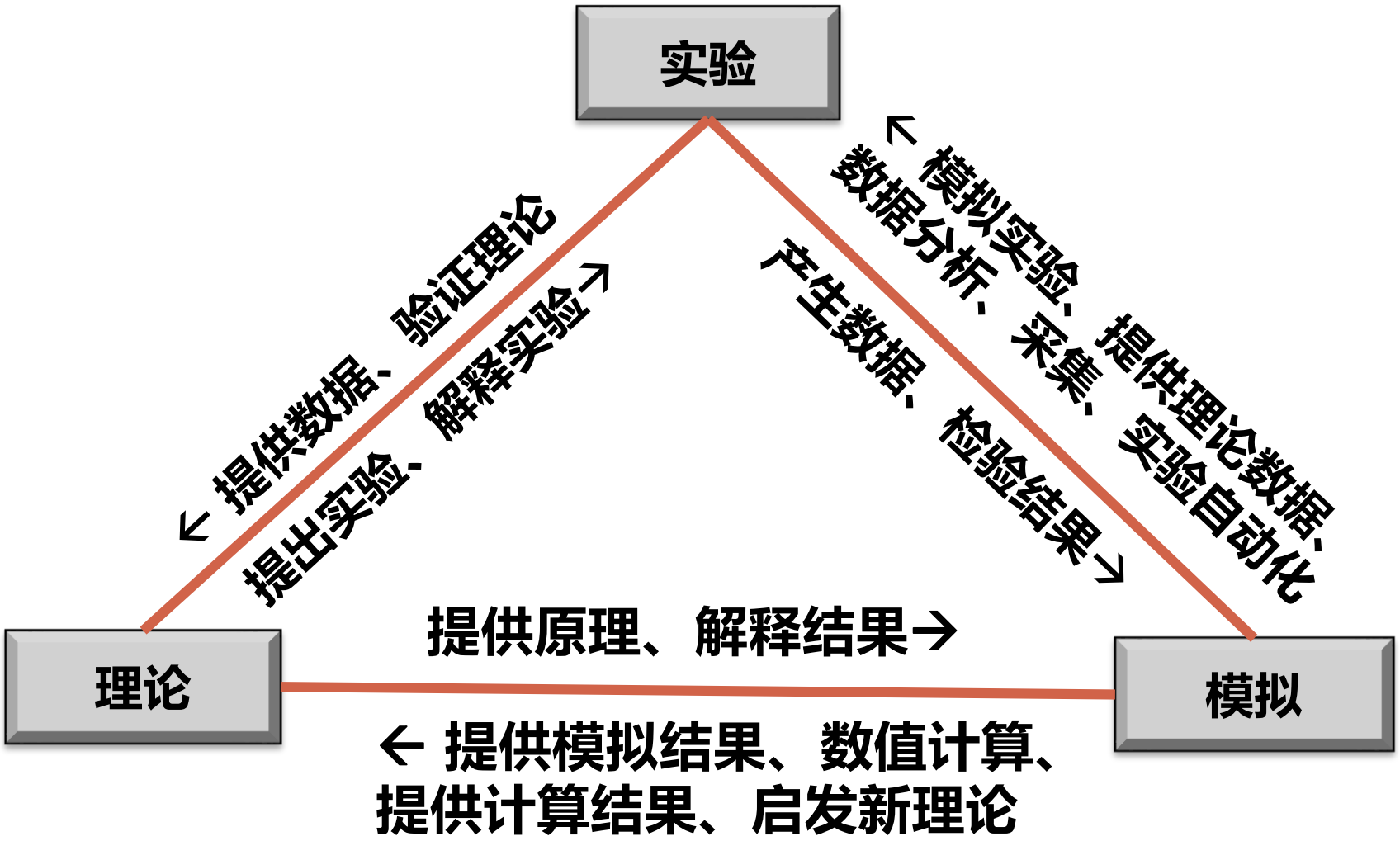




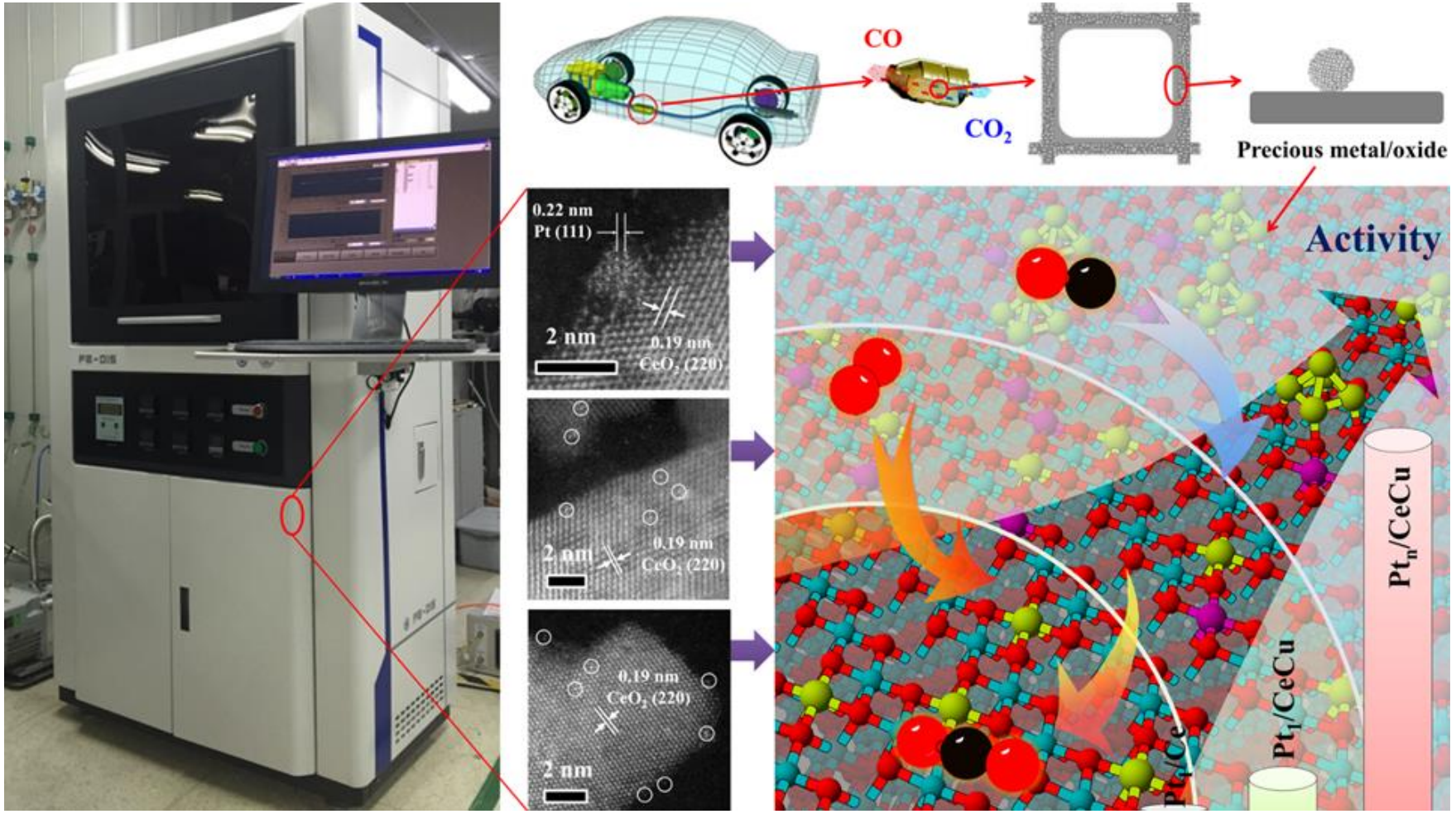
而计算材料方法则可以：

- 替代实验
- 启发实验
- 解释实验
- 帮助发展理论

计算 vs 实验

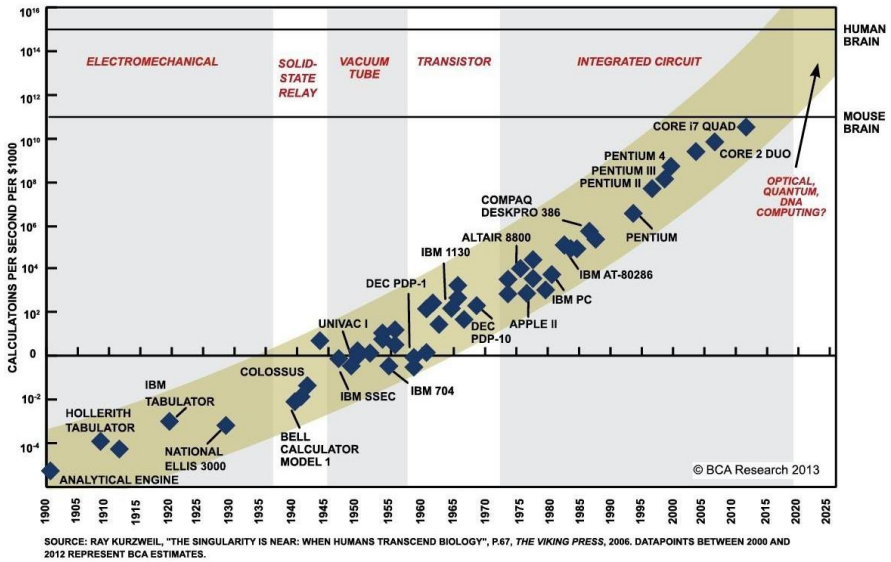

实验
理论
模拟
← 提供数据、验证理论
提出实验、解释实验→
← 模拟实验、提供理论数据、
数据分析、采集、实验自动化
产生数据、检验结果→
提供原理、解释结果→
← 提供模拟结果、数值计算、
提供计算结果、启发新理论

计算 vs 实验



Nature Communications, 2020, 11:4240

摩尔定律



SOURCE: RAY KURZWEIL, "THE SINGULARITY IS NEAR: WHEN HUMANS TRANSCEND BIOLOGY", P.67, *THE VIKING PRESS*, 2006. DATAPOINTS BETWEEN 2000 AND 2012 REPRESENT BCA ESTIMATES.

摩尔定律：计算机CPU的速度每1.5年增加一倍

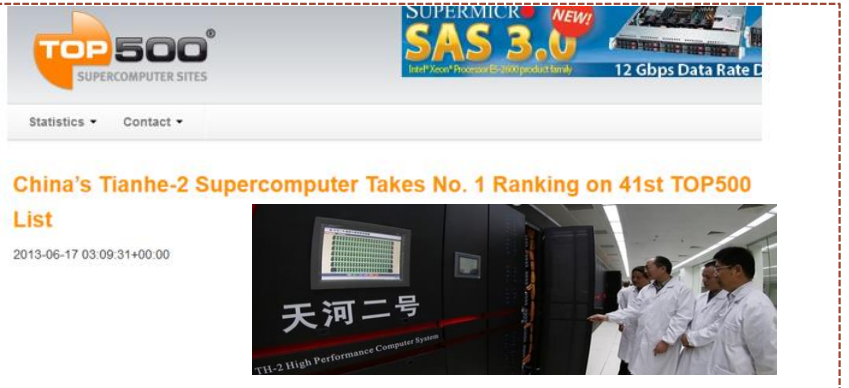
1946~1957 真空管，第一代
1958~1963 晶体管，第二代
1966~1970 集成电路，第三代
1971~超大规模集成电路，第四代

单个计算机计算能力发展

	1944	~1960	~1970	~1990	~2000	~2010	~2015
CPU (flop/s)	3 flop/s	~1 M	~10 M	~100 M	~2 G	~40 G	~3T
RAM				32MB	1GB	32 GB	128 GB

超级计算机

2010年中国“天河一号” 登顶全球500超算排行榜榜首

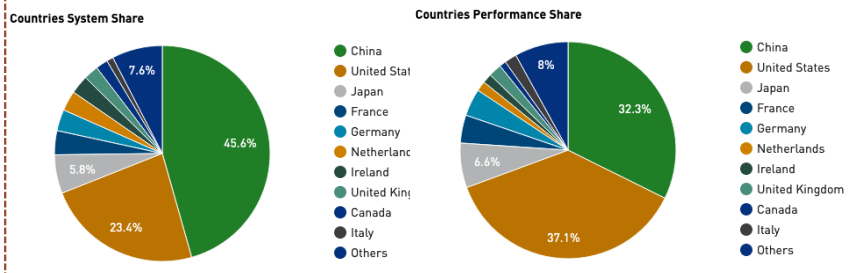


2014年国防科学技术大学研制“天河二号”以峰值计算速度每秒5.49亿亿次优异性能连续第四次获得冠军成为全球最快超级计算机

2016年6月中国自主芯片制造的“神威太湖之光”的浮点运算速度为每秒9.3亿亿次，速度比第二名“天河二号”快出近两倍，登上榜首。

2019年54届TOP500超算榜前十名

Rank	Name	Country
1	Summit	United States
2	Sierra	United States
3	Sunway TaihuLight	China
4	Tianhe-2A	China



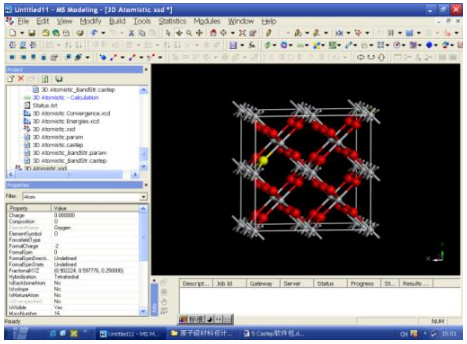
计算模拟的实现

硬件

- **中央处理器(**CPU) – 完成计算
- **内存** – 存储数据供CPU调用
- **HD** – 长期存储数据
- **系统总线** (主板) – 连接各个单元
- **网络** (可选) – 连接各个计算节点
- **输入输出设备** – 与人交互
- **可视化设备** – 分析结果

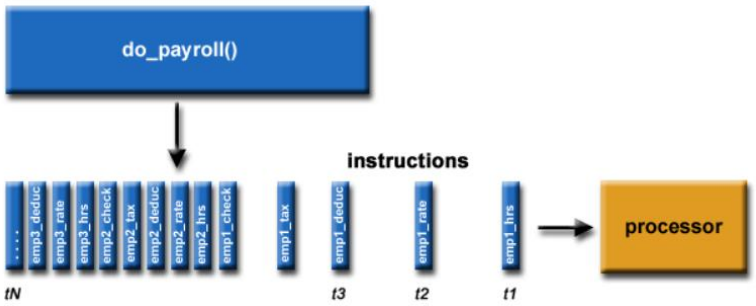
软件

- **操作系统** -- 让硬件听指挥
- **编译器** – 告诉计算机如何工作
- **计算程序** – 指挥计算机工作
- **数学库** – 避免重复劳动
- **并行环境** – 人多力量大
- **可视化软件** – 有效表达信息



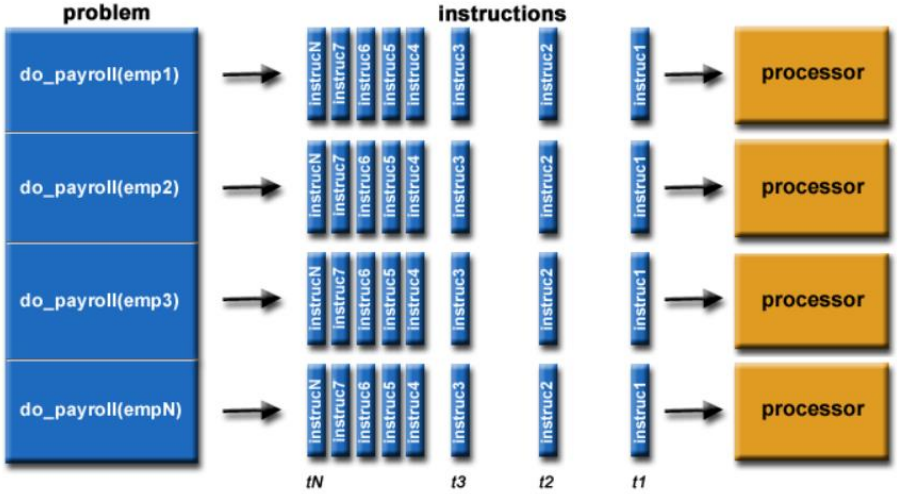
高效计算策略

串行计算



单个处理器，顺序执行计算机程序

并行计算



是由多个处理器组成，在这些处理器之间可以相互通讯和协调

优势：

1、可以大大加快运算速度，即在更短的时间内完成相同的计算量，或者解决原来根本不能计算的非常复杂的问题。

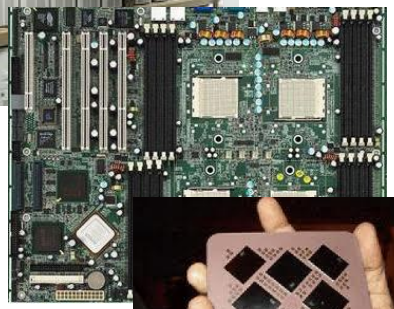
2、克服传统计算机的计算速度受到诸多限制：物理极限、量子效应、加工工艺、散热、成本等等

3、可多个处理器公用内存，并行计算较之SMP机器投入较低、灵活性强。

并行计算软硬件

并行机的发展

多节点



+SMP

+多核心

多节点+多处理器+多核心+GPU

并行机软件环境

操作系统：

- Linux
- Unix
- Windows
- …

并行环境：

- MPI
- OpenMP
- …

编程语言：

- Fortran
- C / CPP
- Matlab
- Python…

编译器：

- Ifort (Fortran)
- GCC (C/CPP)
- PGI (Fortran/C)
- VTune …

数学库：

- MKL (Intel)
- ACML (AMD)
- BLAS
- LAPACK…

科学计算软件：

- LAMMPS
- VASP
- ANSYS
- …

Windows集群?

可行但是还在初级阶段具有较多的问题

1、软件兼容性：大多的科学计算软件是在Unix、Linux系统下编写的。向Windows移植的工程量浩大、需求也不强烈。因此仅有为数不多的商业化软件完成了部分移植。

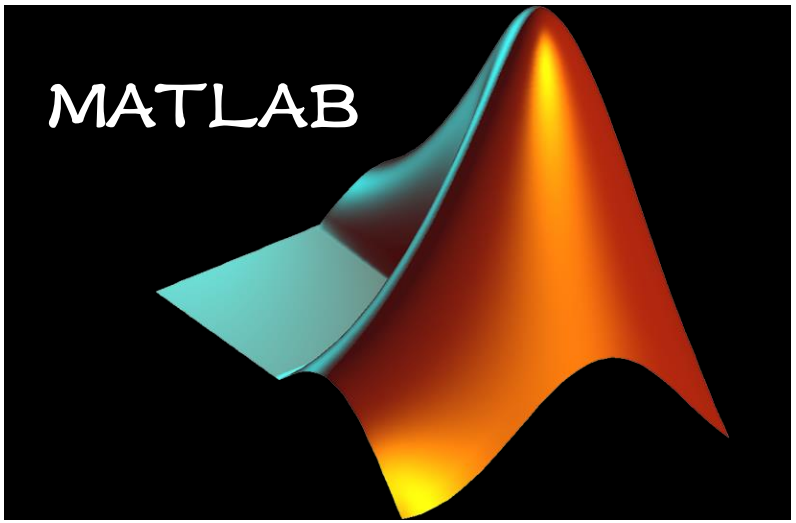
2、性能问题：由于Windows系统自身的特性（牺牲性能来提高用户体验），使得Windows集群的效率普遍不高。

3、操作系统问题：微软集群操作系统Windows Compute Cluster Server尚不完善，仍有较多问题

惨不忍睹的过去、卧薪尝胆的今天、（或许）无限美好的明天

为什么是Python?

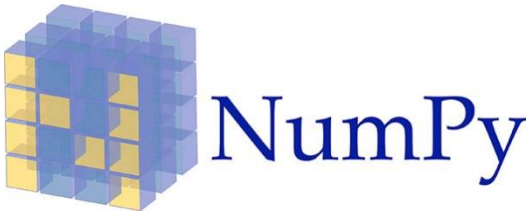
Python：简单易学、更加关注解决问题、开源、跨平台支持、胶水语言、第三方库丰富 …

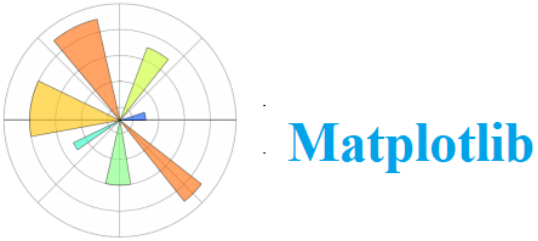


与科学计算领域最流行的商业软件MATLAB 相比，Python 是一门真正的通用程序设计语言，比MATLAB 所采用的脚本语言的应用范围更广泛，有更多程序库的支持，适用于Windows 和Linux 等多种平台，完全免费并且开放源码。

Python科学计算模块

随着NumPy、SciPy、Matplotlib等众多程序库的开发，Python 越来越适合于做科学计算





提供矩阵运算功能　　**添加科学计算函数库**　　**实现数据的可视化**

Tutorial:

https://scipy.org/docs.html
https://numpy.org/doc/
https://docs.scipy.org/doc/scipy/reference/
https://matplotlib.org/contents.html

Anaconda默认安装了Numpy、Scipy、Matplotlib模块

如何让[1, 2] * 2 = [2, 4]

```
1  a = [1, 2]
2  print(a * 2)
3
4  # 列表乘法运算符重载
5  class lst(list):
6      def __init__(self, *x):
7          list.__init__(self)
8          self.number = x
9
10     def __mul__(self, other):
11         return [x*other for x in self.number]
12
13 b = lst(1, 2)
14 print(b * 2)
```

乘法运算符重载

```
[1, 2, 1, 2]
[2, 4]
```

Python中列表中的元素可以是任何对象，而且保存的是对象的引用，而不是数值本身，对数值运算来说比较浪费计算资源

Numpy

NumPy（Numerical Python）支持大量的维度数组与矩阵运算，此外也针对数组运算提供大量的数学函数库

```
1  import numpy as np # 导入numpy模块，简写为np
2  print(np.__doc__)
```

```
NumPy
=====

Provides
  1. An array object of arbitrary homogeneous items
  2. Fast mathematical operations over arrays
  3. Linear Algebra, Fourier Transforms, Random Number Generation

How to use the documentation
----------------------------
Documentation is available in two forms: docstrings provided
with the code, and a loose standing reference guide, available from
`the NumPy homepage <https://www.scipy.org>`_.
```

提供多维数组对象ndarray (n-dimensional array object)

对数组进行处理的函数和方法

与列表不同，NumPy 数组存储在内存中的一个连续位置，因此进程可以非常有效地访问和操纵它们

创建数组对象

np.array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)

object：从Python内置的列表或者元组创建数组
dtype：表示数组所需的数据类型，如果为给定则选择保存对象所需的最小类型，默认为None
ndmin：指定生成数组应该具有的最小维数

创建数组

```
1  # print(np.array.__doc__)
2  a = np.array([1, 2]) # 从列表创建数组
3  print(a * 2)
4  print(type(a), a.__class__)
5
6  # 可以是字符串
7  b = np.array([1, "a"])
8  print(b)
9  # print(b * 2) # 不支持数组的运算
```

数组的乘法运算

没有逗号分隔

```
[2 4]
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
['1' 'a']
```

1也被转换为字符串

多维数组

```
1  # 0维
2  print(np.array(1))
3
4  # 1维
5  print(np.array((1, 2, 3, 4, 5)))
6
7  # 2维
8  print(np.array(([1, 2, 3, 4, 5],(1, 2, 3, 4, 5))))
9
10 # 3维
11 print(np.array([[[1, 2],[3, 4]],[[5, 6],[7, 8]]]))
12
13 # 多维数组
14 c = np.array([1, 2, 3, 4, 5], ndmin = 5)
15 print(c, c.ndim)
```

数组的维数为5

```
1
[1 2 3 4 5]
[[1 2 3 4 5]
 [1 2 3 4 5]]
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
[[[[[1 2 3 4 5]]]]] 5
```

c.ndim：查询数组的维数

其他创建数组的函数

函数	描述
np.arange(x,y,i)	创建由x到y，以i为步长的数组
np.linspace(x,y,n)	创建由x到y，等分成n个元素的数组
np.random.rand(m,n)	创建m行n列的随机数组
np.ones((m,n), dtype)	创建m行n列的全1数组，dtype是数据类型
np.zeros((m,n), dtype)	创建m行n列全0数组，dtype是数据类型
np.indices((m,n))	创建m行n列的矩阵

其他创建数组的函数

```
1  print(np.arange(1,5,1))
2  print(np.linspace(0,2*np.pi,10))
3  print(np.random.rand(2,3))
4  print(np.ones((2,3)))
5  print(np.zeros((2,3)))
6  print(np.indices((2,3)))
7  a, b = np.indices((2,3))
8  print(a, b, sep = "\n")
```

注意参数为元组

```
[1 2 3 4]
[0.          0.6981317   1.3962634   2.0943951   2.7925268   3.4906585
 4.1887902   4.88692191  5.58505361  6.28318531]
[[0.12387708 0.73663296 0.7847359 ]
 [0.44069413 0.8830835  0.54260579]]
[[1. 1. 1.]
 [1. 1. 1.]]
[[0. 0. 0.]
 [0. 0. 0.]]
[[[0 0 0]
  [1 1 1]]

 [[0 1 2]
  [0 1 2]]]
[[0 0 0]
 [1 1 1]]
[[0 1 2]
 [0 1 2]]
```

2 * 2 * 3的数组
(包含2*3数组的索引信息)

$$a = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$b = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix}$$

a为2*3数组的第一个索引

b为2*3数组的第二个索引

数组对象的常用属性

属性	描述
ndarray.ndim	数组的维数
ndarray.shape	数组在每个维度上的元素个数组成的元组
ndarray.size	数组元素的总个数
ndarray.dtype	数组元素的数据类型
ndarray.itemsize	数组中每个元素的字节大小
ndarray.data	包含实际数组元素的缓冲区地址
ndarray.flat	数组元素的迭代器

数组的常用属性

```
1  a = np.ones((3,3))
2  print(a)
3  print(a.ndim)
4  print(a.shape)
5  print(a.size)
6  print(a.dtype, a.itemsize, a.data)
7  for x in a.flat:
8      print(x, end = " ")
9
10 print("")
11 b = np.array([[1,2,3],[4,5,6]])
12 for x in b.flat:
13     print(x, end = " ")
```

ndarray.flat按行的先后顺序返回迭代器

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
2
(3, 3)
9
float64 8 <memory at 0x00000000071C86C0>
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1 2 3 4 5 6
```

ndarray.shape返回元组

数组对象的方法

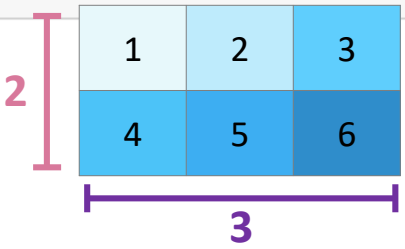
变形

```
1  a = np.array([[1, 2, 3], [4, 5, 6]])
2  print(a.shape)
3  b = a.reshape(3, 2)
4  print(b)
5  b.resize(1, 6)
6  print(b)
7  b.resize(6)
8  print(b)
9
10 c = a.swapaxes(0, 1)
11 d = a.flatten()
12 print(c, d)
```

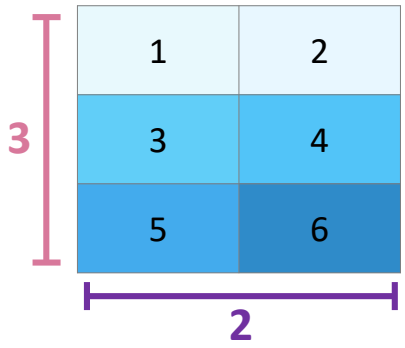
```
(2, 3)
[[1 2]
 [3 4]
 [5 6]]
[[1 2 3 4 5 6]]
[1 2 3 4 5 6]
[[1 4]
 [2 5]
 [3 6]] [1 2 3 4 5 6]
```

ndarray.reshape(n, m)：返回维度为(n,m)的数组，不改变ndarray.flat的顺序
（ ndarray.resize(n, m)无返回值，直接修改 ）

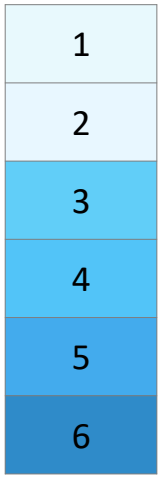
a



a.reshape(3,2)



a.reshape(6)



变形

```
1  a = np.array([[1, 2, 3], [4, 5, 6]])
2  print(a.shape)
3  b = a.reshape(3, 2)
4  print(b)
5  b.resize(1,6)
6  print(b)
7  b.resize(6)
8  print(b)
9
10 c = a.swapaxes(0, 1)
11 d = a.flatten()
12 print(c, d)
```

```
(2, 3)
[[1 2]
 [3 4]
 [5 6]]
[[1 2 3 4 5 6]]
[1 2 3 4 5 6]
[[1 4]
 [2 5]
 [3 6]] [1 2 3 4 5 6]
```

ndarray.swapaxes(0, 1)：交换第0个和第1个维度的数据

a

1	3	5
2	4	6

a.swapaxes(0,1)

1	2
3	4
5	6

ndarray.flatten()：返回折叠后的一维数组

变形

```
1  a = np.arange(1,13)
2  print(a,  a.shape)
3  b = a.reshape(2,  3,  2)
4  print(b)
5  c = b.swapaxes(0,2)
6  print(c)
7  d = b.flatten()
8  print(d)
```

多维数组交换第0个和第2个维度的数据

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

 [[ 7  8]
  [ 9 10]
  [11 12]]]
[[[ 1  7]
  [ 3  9]
  [ 5 11]]

 [[ 2  8]
  [ 4 10]
  [ 6 12]]]
[ 1  2  3  4  5  6  7  8  9 10 11 12]
```

奇偶分开，结合np.indices()进行分析

```
1  b1, b2, b3 = np.indices(b.shape)
2  print(b1, b2, b3, sep = "\n")
```

```
[[[0 0]
  [0 0]
  [0 0]]

 [[1 1]
  [1 1]
  [1 1]]]
[[[0 0]
  [1 1]
  [2 2]]

 [[0 0]
  [1 1]
  [2 2]]]
[[[0 1]
  [0 1]
  [0 1]]

 [[0 1]
  [0 1]
  [0 1]]]
```

	第0维索引	第1维索引	第2维索引
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	0	2	0
6	0	2	1
7	1	0	0
8	1	0	1
9	1	1	0
10	1	1	1
11	1	2	0
12	1	2	1

索引和切片

```
# 一维数组
a = np.arange(12)
print(a[1], a[-1], a[1:4], a[1:6:2])

# 多维数组
b = a.reshape(2, 2, 3)
print(b)
print(b[0, 1, 2])
print(b[0, 0, :])
print(b[0, 0, :-1])
print(b[0, :, :-1])
```

一维数组与列表类似

每项用逗号分隔，依次代表第n个维度

```
1 11 [1 2 3] [1 3 5]
[[[ 0  1  2]
  [ 3  4  5]]

 [[ 6  7  8]
  [ 9 10 11]]]
5
[0 1 2]
[0 1]
[[0 1]
 [3 4]]
```

注意多维数组与列表的区别！！！

```
# 列表索引与切片
listb = [[[0, 1, 2], [3, 4, 5]], [[6, 7, 8], [9, 10, 11]]]
print(listb)
print(listb[0][1][2])
print(listb[0][0][:])
```

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

```
print(b[np.array([[0, 0], [0, 0]]), np.array([[0, 0], [1, 1]]), np.array([[0, 2], [0, 2]])])
```

```
[[0 2]
 [3 5]]
```

利用多维数组进行索引

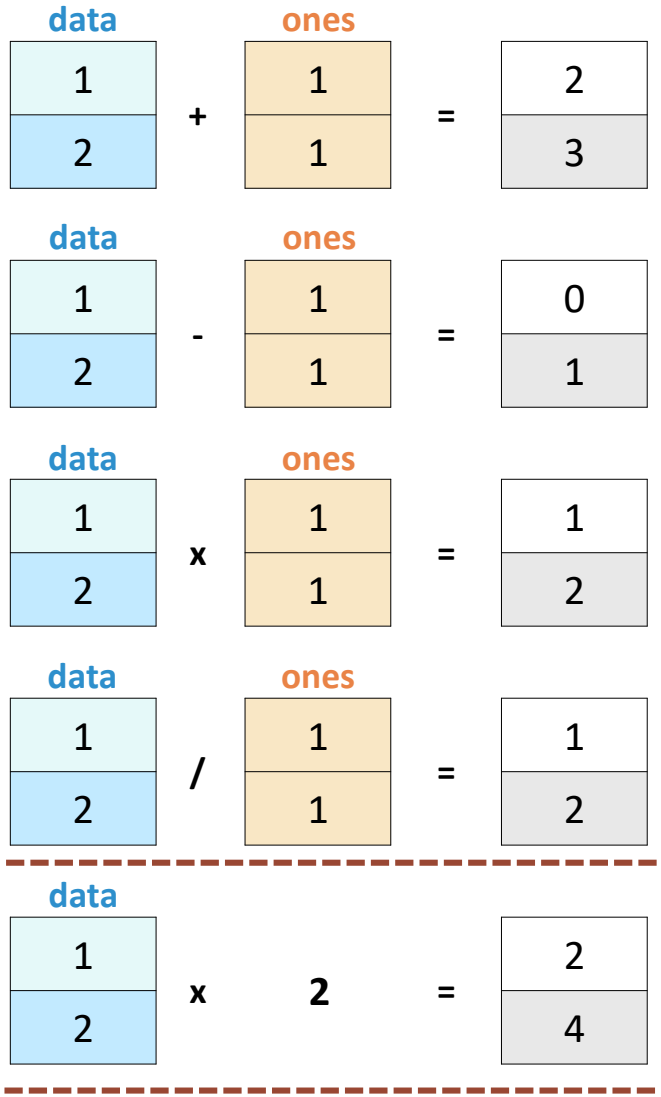
数组对象的运算

四则运算

- **按位置运算**（即对应元素进行加减乘除）
- 数组和单个数字之间也可以进行运算操作（即向量和标量之间的运算）

```
1  data = np.array([1, 2])
2  ones = np.ones(2)
3  print(data, ones)
4
5  print("data + ones = ", data+ones)
6  print("data - ones = ", data-ones)
7  print("data * ones = ", data*ones)
8  print("data / ones = ", data/ones)
9  print("data * 2 = ", data*2)
```

```
[1 2] [1. 1.]
data + ones =  [2. 3.]
data - ones =  [0. 1.]
data * ones =  [1. 2.]
data / ones =  [1. 2.]
data * 2 =  [2 4]
```

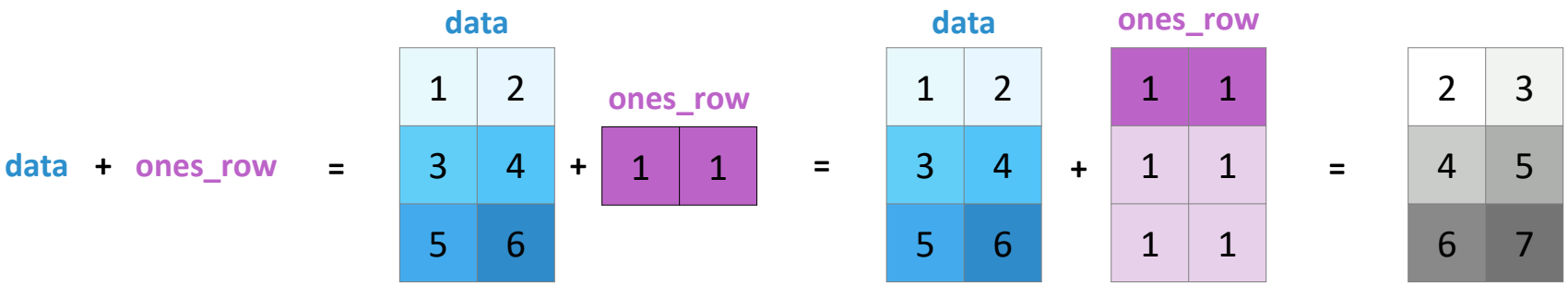


广播机制

广播原则

对不同形状的数组之间执行算术运算，需遵循：

- 让所有输入数组都向其中shape最长的数组看齐，shape中不足的部分通过在前面加1补齐
- 输出数组的shape是输入数组shape各维度的最大值
- 如果**输入数组的某个维度和输出数组的对应维度的长度相同或者某个维度长度为1时**，这个数组可以用来计算，否则出错
- 当输入数组的某个维度的长度为1时，沿着此维度运算时都用此维度上的第一组值



data为3×2的数组，ones_row为1×2的数组，相加时后者被扩充（内容重复填充）为3×2的数组然后按位执行加法

广播原则

```
1  data = np.arange(1,7).reshape(3,2)
2  ones = np.ones((1,2))
3  print(data+ones)
4
5  # 各维度长度相同或者有维度长度为1才能计算
6  ones = np.ones((2,2))
7  print(data+ones)
```

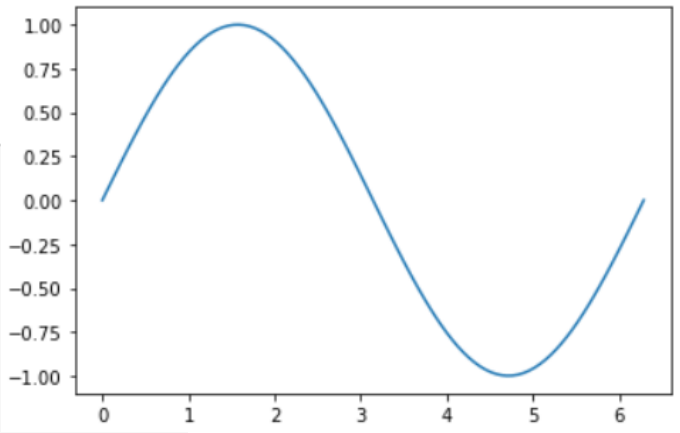
```
[[2. 3.]
 [4. 5.]
 [6. 7.]]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-142-1b682e04eee8> in <module>
      4
      5 ones = np.ones((2,2))
----> 6 print(data+ones)

ValueError: operands could not be broadcast together with shapes (3,2) (2,2)
```

数组运算函数

np.sin()

```
1  import matplotlib.pyplot as plt
2  x = np.linspace(0,2*np.pi,10)
3  y = np.sin(x)
4  print(x, y, sep = "\n")
5
6  x = np.linspace(0,2*np.pi,100)
7  y = np.sin(x)
8  plt.plot(x, y)
9  plt.show()
```




```
[0.          0.6981317   1.3962634   2.0943951   2.7925268   3.4906585
 4.1887902   4.88692191  5.58505361  6.28318531]
[ 0.00000000e+00  6.42787610e-01  9.84807753e-01  8.66025404e-01
  3.42020143e-01 -3.42020143e-01 -8.66025404e-01 -9.84807753e-01
 -6.42787610e-01 -2.44929360e-16]
```

可以直接将数组作为参数输入np.sin()进行运算

np.sin() vs math.sin()

```python
import time, math
x = [i for i in range(10000000)]
y, z = [], []
# math.sin()
start_time = time.perf_counter()
for i in x:
    y.append(math.sin(i))
during_time = time.perf_counter() - start_time
print("math.sin: {}".format(during_time))

# np.sin()
start_time = time.perf_counter()
X = np.array(x)
Y = np.sin(X)
during_time = time.perf_counter() - start_time
print("np.sin: {}".format(during_time))

# np.sin()
start_time = time.perf_counter()
for i in x:
    z.append(np.sin(i))
during_time = time.perf_counter() - start_time
print("np.sin: {}".format(during_time))
```

```
math.sin: 2.5630429979937617
np.sin: 0.635888981007156
np.sin: 10.226119162005489
```

计算1千万次正弦值

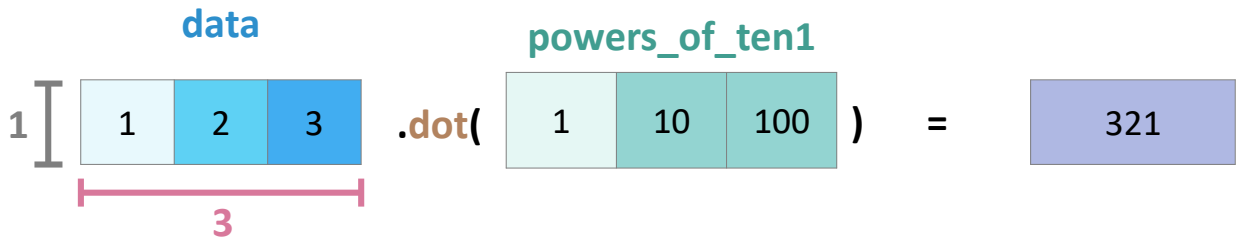
numpy.sin在C语言级别的循环计算

numpy.sin为了同时支持数组和单个值的计算，其C语言的内部实现要比math.sin复杂很多

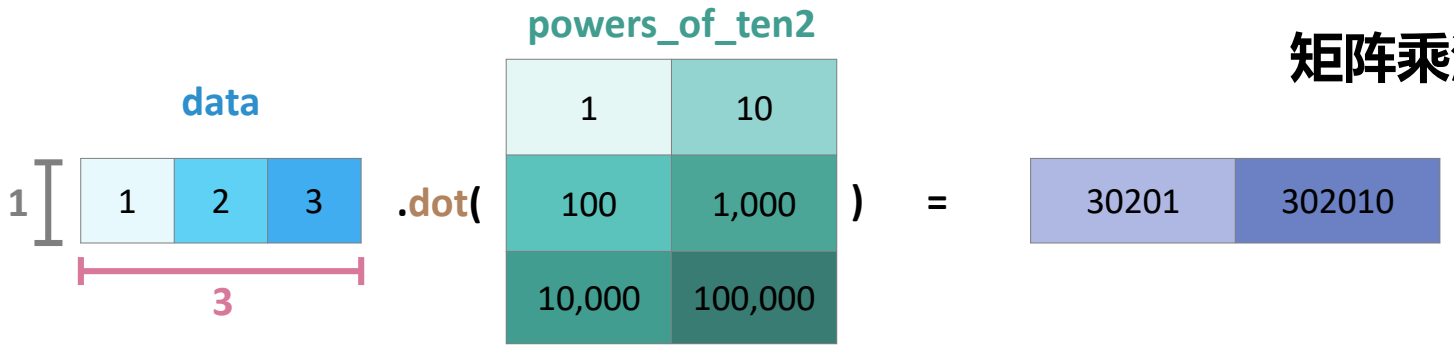
数组运算函数

np.dot()

向量乘法



矩阵乘法



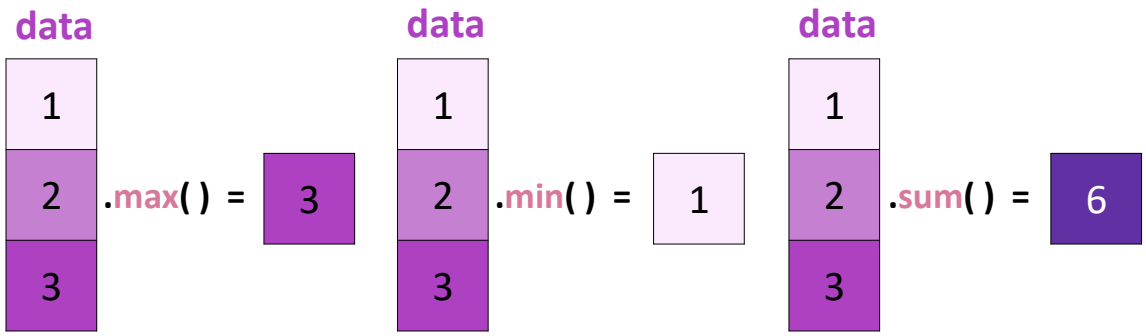
Matrix dimensions: 1 x 3 3 x 2 1 x 2

```
1  data = np.array([1, 2, 3])
2  powers_of_ten1 = np.array([1, 10, 100])
3  powers_of_ten2 = np.array([[1, 10], [100, 1000], [10000, 100000]])
4  print(np.dot(data, powers_of_ten1))
5  print(np.dot(data, powers_of_ten2))
```

```
321
[ 30201 302010]
```

其他常用运算函数

可使用 min、max 和 sum 进行最值、求和统计

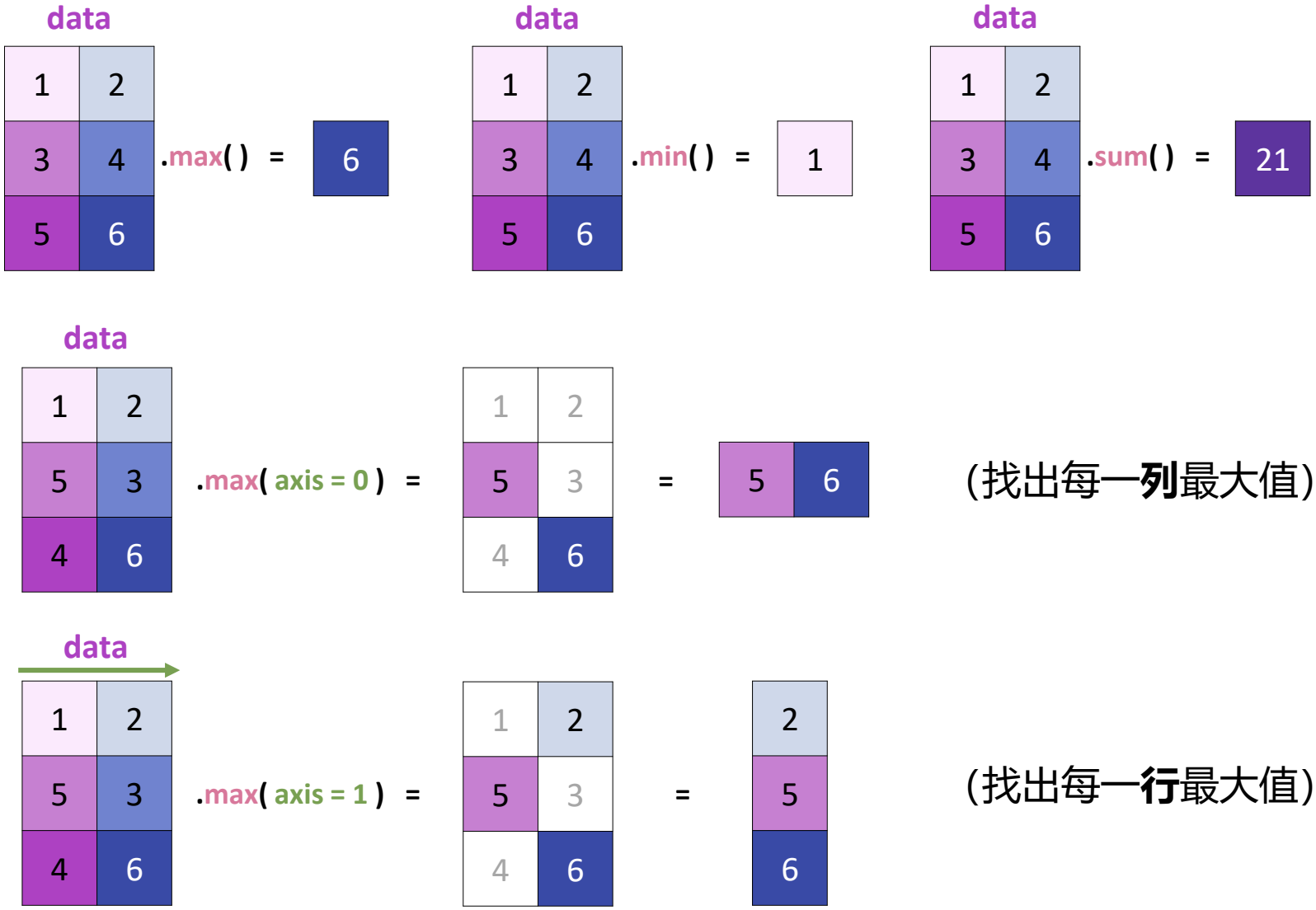


可使用 mean 得到平均值、 prod 得到所有元素乘积、 std 得到标准差

```
1 data = np.array([1, 2, 3])
2 print(np.min(data), np.max(data), np.sum(data))
3 print(np.mean(data), np.prod(data), np.std(data))
```

```
1 3 6
2.0 6 0.816496580927726
```

其他常用运算函数


data
1 2
3 4
5 6
.max( ) = 6
data
1 2
3 4
5 6
.min( ) = 1
data
1 2
3 4
5 6
.sum( ) = 21
data
1 2
5 3
4 6
.max(axis = 0) =
1 2
5 3
4 6
= 5 6
(找出每一列最大值)
data
1 2
5 3
4 6
.max(axis = 1) =
1 2
5 3
4 6
= 2 5 6
(找出每一行最大值)

Scipy

SciPy是构建在Numpy的基础之上的，它提供了许多的操作Numpy的数组的函数

```
1  import scipy
2  # print(dir(scipy))
3  print(scipy.__doc__)
```

```
SciPy: A scientific computing package for Python
================================================

Documentation is available in the docstrings and
online at https://docs.scipy.org.

Contents
--------
SciPy imports all the functions from the NumPy namespace, and in
addition provides:

Subpackages
-----------
Using any of these subpackages requires an explicit import. For example,
``import scipy.cluster``.
```

包括了统计、优化、整合以及线性代数模块、傅里叶变换、信号和图像图例，常微分方程的求解等

optimize模块

SciPy的optimize模块提供了许多数值优化算法

最小二乘法

假设有一组实验数据 (xi,yi)，我们事先知道它们之间应该满足某函数关系：yi= f(xi)。通过这些已知信息，需要确定函数f的一些参数。例如，如果函数f是线性函数f(x)= kx+ b，那么参数k和b就是需要确定的值。

$$S(p)=\sum_{i=1}^{m}\left[y_i-f\left(x_i,p\right)\right]^2$$

如果用p表示函数中需要确定的参数，则目标是找到一组p使得函数S的值最小

线性回归模型

```python
import numpy as np
from scipy.optimize import curve_fit
# 产生数据
x = [i * 0.1 for i in range(100)]
y = list(map(lambda i:i+2, x))

# 添加噪音
X, Y = np.array(x), np.array(y)
Yn = Y + 0.9 * np.random.normal(size=len(x))

# 函数模型拟合
def func(x, a, b):
    return a*x+b
popt, pcov = curve_fit(func, X, Yn)
print(popt)
print(pcov)
```

```
[0.99335615 2.10865386]
[[ 0.00095351 -0.00471988]
 [-0.00471988  0.03130857]]
```

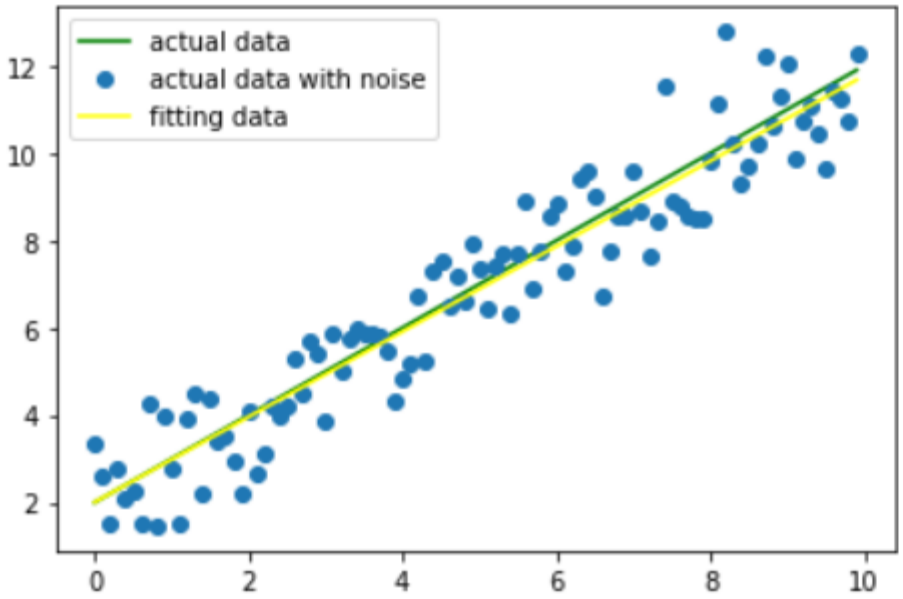
curve_fit(func, x, y)

- func：自定义的拟合的函数形式
- x,y：原始数据

如果有一个很好的拟合效果，**popt返回的是给定模型的最优参数**。可以使用pcov的值检测拟合的质量，其对角线元素值代表着每个参数的方差。

线性回归模型

```
1  import matplotlib.pyplot as plt
2  yfit = func(X, popt[0], popt[1])
3  plt.plot(X, Y, color="green", label="actual data")
4  plt.plot(X, Yn, "o", label="actual data with noise")
5  plt.plot(X, yfit, color="yellow", label="fitting data")
6  plt.legend(loc="best")
7  plt.show()
```



非线性回归模型

通过最小二乘拟合高斯分布（Gaussian profile），一种非线性函数：

$$\alpha * \exp\left(-(x-\mu)^2 / 2\sigma^2\right)$$

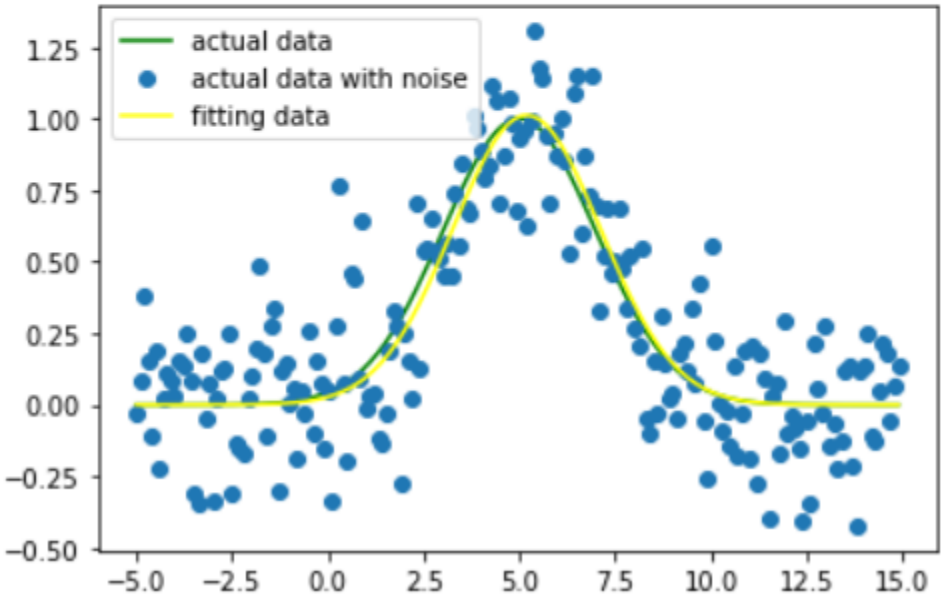
α表示一个标量，μ是期望值，而σ是标准差

```python
import numpy as np
from scipy.optimize import curve_fit
# 函数模型拟合
def func(x, a, b, c):
    return a*np.exp(-(x-b)**2/(2*c**2))

# 产生数据
X = np.array([i * 0.1 for i in range(-50, 150)])
Y = func(X, 1, 5, 2)

# 添加噪音
Yn = Y + 0.2 * np.random.normal(size=len(X))

p0 = [1.2, 4, 3] # 初步猜测参数，如果没有默认全为1
#popt, pcov = curve_fit(func, X, Yn)
popt, pcov = curve_fit(func, X, Yn, p0 = p0)
print(popt)
```

```
[1.01225994 5.19004858 1.93395595]
```

非线性回归模型

```
1  import matplotlib.pyplot as plt
2  yfit = func(X, *tuple(popt))
3  plt.plot(X, Y, color="green", label="actual data")
4  plt.plot(X, Yn, "o", label="actual data with noise")
5  plt.plot(X, yfit, color="yellow", label="fitting data")
6  plt.legend(loc="best")
7  plt.show()
```



Matplotlib

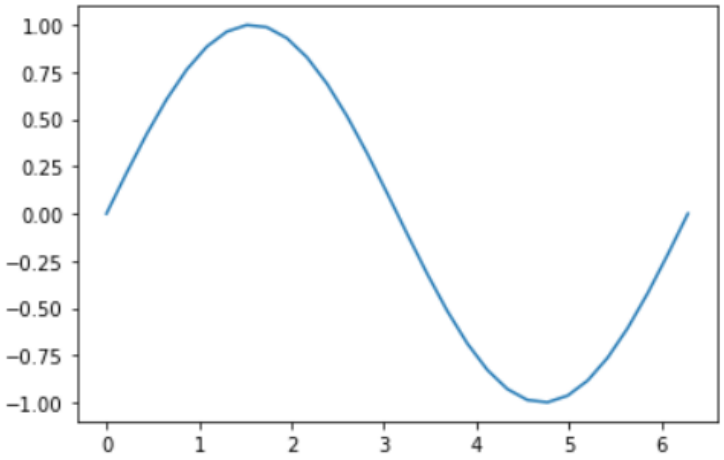
Matplotlib是一个绘图库，包含了大量的工具，可以创建各种图形，包括简单的曲线、散点图、条形图，甚至是三维图形

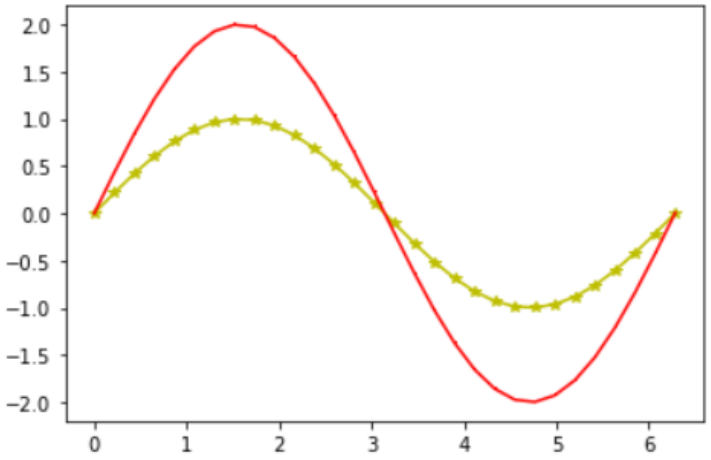
```
1 import matplotlib
2 # print(dir(matplotlib))
3 print(matplotlib.__doc__)
```

```
This is an object-oriented plotting library.

A procedural interface is provided by the companion pyplot module,
which may be imported directly, e.g.::

    import matplotlib.pyplot as plt

or using ipython::

    ipython

at your terminal, followed by::

    In [1]: %matplotlib
    In [2]: import matplotlib.pyplot as plt
```

数据曲线

```
1  import matplotlib.pyplot as plt
2  X = np.linspace(0, 2*np.pi, 30)
3  Y = np.sin(X)
4  plt.plot(X,Y)
5  plt.show()
6
7  # 调整样式
8  plt.plot(X, Y, "y*-")
9  plt.plot(X, Y*2, "r--")
10 plt.show()
```



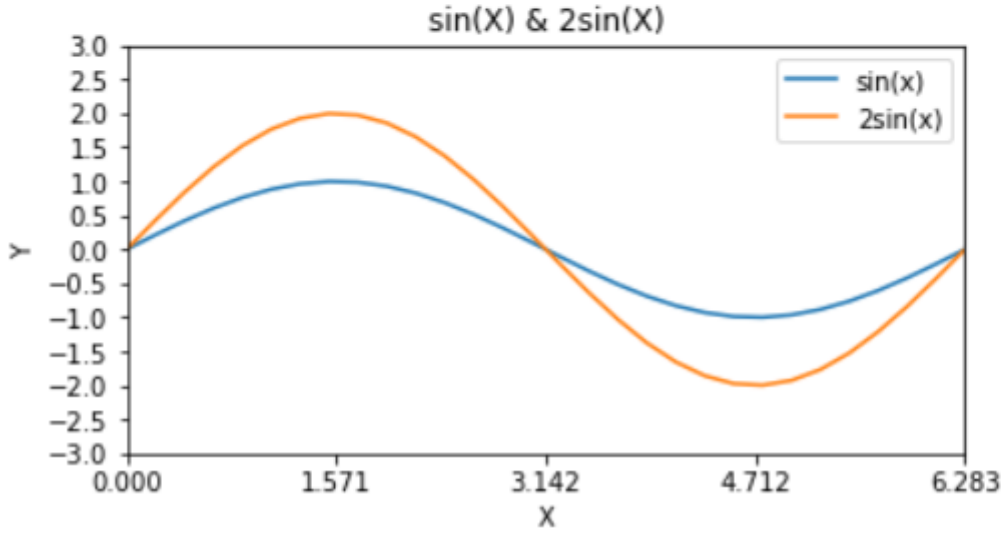


增加一个字符串参数调整样式：

- 常见的颜色表示方式：　'b' 蓝色　'g' 绿色　'r' 红色　'y' 黄色　'k' 黑色　'w' 白色
- 常见的点的表示方式：　'.' 点　',' 像素　'o' 圆　's' 方形　'^' 三角形
- 常见的线的表示方式：　'-' 直线　'—' 虚线　':' 点线　'-.' 点画线

图注

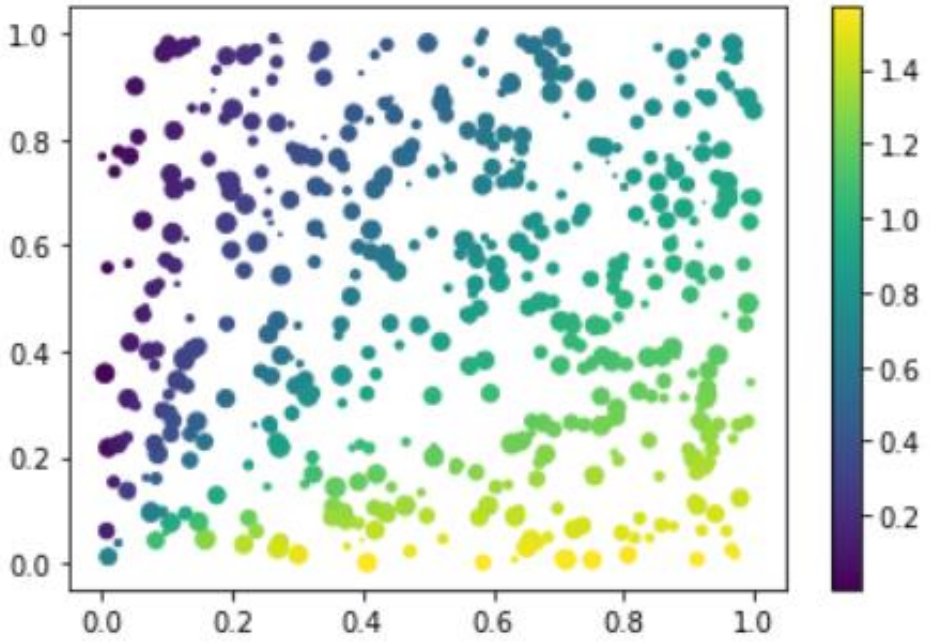
```
plt.figure(figsize=(6,3)) # 设置图片大小
plt.plot(X, Y, label = "sin(x)") # 曲线标注
plt.plot(X, Y*2, label = "2sin(x)") # 曲线标注
plt.legend(loc="best")
plt.title("sin(X) & 2sin(X)") # 标题
plt.xlim((0, np.pi+1)) # x轴范围
plt.ylim((-3,3)) # y轴范围
plt.xlabel("X") # x轴名称
plt.ylabel("Y") # y轴名称
plt.xticks((0, np.pi*0.5, np.pi, np.pi*1.5, np.pi*2)) # x轴刻度精度
plt.yticks(np.linspace(-3,3,13)) # y轴刻度精度
plt.show()
```



散点图

plt.scatter()

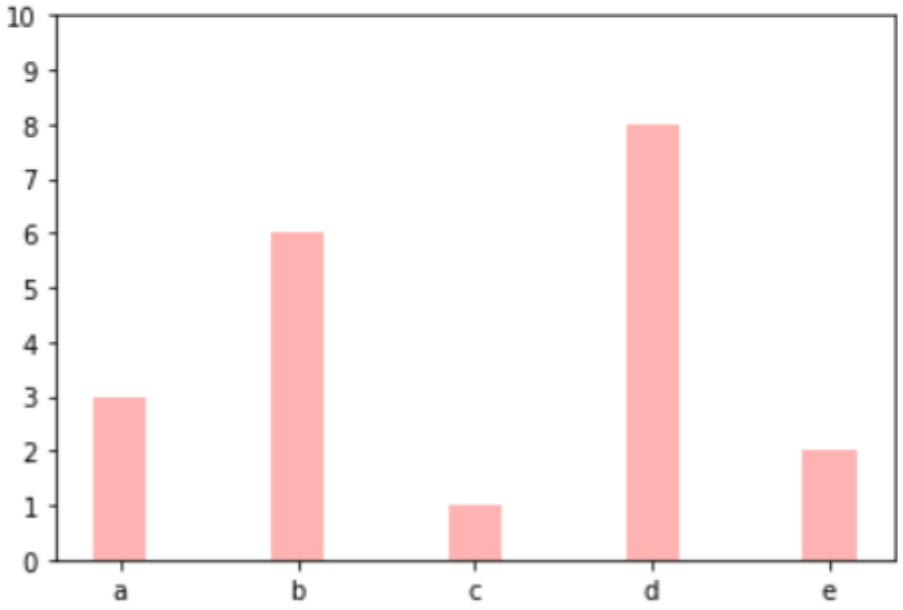
```
1  import matplotlib.pyplot as plt
2
3  # 生成数据点
4  k = 500
5  x = np.random.rand(k)
6  y = np.random.rand(k)
7
8  # 生成每个点的大小、颜色
9  size = 50*np.random.rand(k)
10 colour = np.arctan2(x, y)
11
12 # 画图并添加颜色栏（colorbar）
13 plt.scatter(x, y, s=size, c=colour)
14 plt.colorbar()
15 plt.show()
```



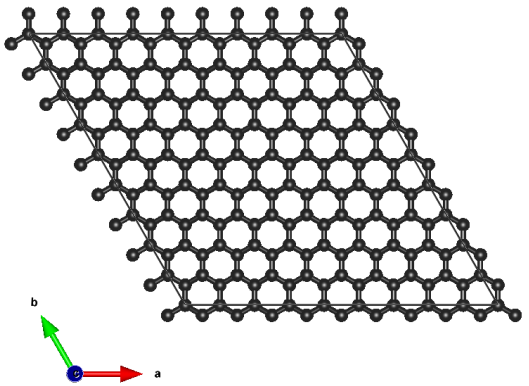
柱状图

plt.bar()

```
1  import matplotlib.pyplot as plt
2  x = [1,2,3,4,5]
3  y = [3,6,1,8,2]
4  plt.bar(x,y, width=0.3, color='r', alpha=0.3)
5  plt.xticks(x, ["a","b","c","d","e"])
6  plt.yticks(np.linspace(0,10,11))
7  plt.show()
```



计算文件操作示例




```
graphene
1.0
        22.1399993896          0.0000000000          0.0000000000
       -11.0699996948         19.1738019112          0.0000000000
         0.0000000000          0.0000000000          6.8000001907
     C
   162
Direct
     -0.000000000         -0.000000000          0.250000000
      0.037039998          0.074069997          0.250000000
      0.111110007         -0.000000000          0.250000000
      0.148149991          0.074069997          0.250000000
      0.222220013         -0.000000000          0.250000000
      0.259259987          0.074069997          0.250000000
     -0.000000003          0.111110000          0.250000000
      0.037039995          0.185190007          0.250000000
      0.111110003          0.111110000          0.250000000
      0.148149991          0.185190007          0.250000000
      0.222220016          0.111110000          0.250000000
      0.259259987          0.185190007          0.250000000
     -0.000000007          0.222220000          0.250000000
      0.037039995          0.296299982          0.250000000
      0.111110000          0.222220000          0.250000000
      0.148149991          0.296299982          0.250000000
      0.222220007          0.222220000          0.250000000
      0.259259987          0.296299982          0.250000000
      0.333330009         -0.000000000          0.250000000
      0.370369983          0.074069997          0.250000000
      0.444440027         -0.000000000          0.250000000
      0.481480022          0.074069997          0.250000000
      0.555559973         -0.000000000          0.250000000
      0.592589975          0.074069997          0.250000000
      0.333330012          0.111110000          0.250000000
      0.370369972          0.185190007          0.250000000
      0.444440007          0.111110000          0.250000000
      0.481480011          0.185190007          0.250000000
      0.555559954          0.111110000          0.250000000
      0.592589964          0.185190007          0.250000000
      0.333329992          0.222220000          0.250000000
      0.370369983          0.296299982          0.250000000
      0.444440031          0.222220000          0.250000000
```

```
162
graphite
 C      0.000000     0.000000     1.700000
 C      0.000111     1.420203     1.700000
 C      2.459975     0.000000     1.700000
 C      2.460086     1.420203     1.700000
 C      4.919951     0.000000     1.700000
 C      4.920061     1.420203     1.700000
 C     -1.229988     2.130401     1.700000
 C     -1.229988     3.550797     1.700000
 C      1.229988     2.130401     1.700000
 C      1.229987     3.550797     1.700000
 C      3.689963     2.130401     1.700000
 C      3.689963     3.550797     1.700000
 C     -2.459975     4.260802     1.700000
 C     -2.459975     5.681197     1.700000
 C      0.000000     4.260802     1.700000
 C      0.000000     5.681197     1.700000
 C      2.459975     4.260802     1.700000
 C      2.459975     5.681197     1.700000
 C      7.379926     0.000000     1.700000
 C      7.380036     1.420203     1.700000
 C      9.839902     0.000000     1.700000
 C      9.840013     1.420203     1.700000
 C     12.300097     0.000000     1.700000
 C     12.299987     1.420203     1.700000
 C      6.149939     2.130401     1.700000
 C      6.149938     3.550797     1.700000
 C      8.609914     2.130401     1.700000
 C      8.609914     3.550797     1.700000
 C     11.070109     2.130401     1.700000
 C     11.069888     3.550797     1.700000
 C      4.919950     4.260802     1.700000
 C      4.919950     5.681197     1.700000
```

小结

- ndarray对象，相关属性和方法
- Numpy中数组的运算和常用函数
- Scipy中的optimize模块和Matplotlib作图方法