数据分析

一、numpy入门

1. numpy是什么?

1) numpy是Numerical Python的缩写，即“数值的python”。
2) numpy是一整套开源的科学计算库。
3) numpy弥补了作为通用编程语言的Python在数值计算方面，能力弱、速度慢的不足。
4) numpy拥有丰富的数学函数、强大的多维数组和优异的运算性能。
5) numpy与scipy、scikits、matplotlib等其它科学计算库可以很好地协调工作。
6) numpy可取代matlab和mathematica的部分功能，并允许用户进行快速交互式原型设计。

2. numpy的历史背景

1) 1995年，numeric，早期的python语言数值计算库，如今已废弃。
2) 2001年，scipy项目启动，用numeric做数值计算，并将一些新特性放到numarray中。
3) 2005年，重构numeric，意图将numarray的部分特性整合进来。
4) 2006年，numpy脱离scipy独立发布，同时拥有numeric和numarray的所有特性。

```
              scipy <----- numpy
            ------------------   ^
numeric -> numeric + numarray --+
```

3. 为什么使用numpy?

1) 同样的数值计算任务，使用numpy比直接编写python代码更加便捷。
2) numpy中数据的存储和输入输出效率远高于python本身，但其通用性不及后者。
3) numpy的大部分代码是用C语言编写的，这令其在运行速度上比纯python代码快得多。
4) numpy是完全开源的，成本低至极限且质量更有保证。

4. numpy的局限性

1) numpy数组的通用性不及python的list，在科学计算之外的领域，其优势并不明显。
2) numpy模块是用C语言实现的，依赖于java虚拟机的jython目前还无法使用numpy的功能。

5. numpy的python环境

1) 安装python

https://www.python.org

2) 安装数据分析工具

numpy+mkl (Intel® Math Kernel Library):
https://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy
numpy-1.13.3+mkl-cp36-cp36m-win_amd64.whl

scipy:
https://www.lfd.uci.edu/~gohlke/pythonlibs/#scipy
scipy-1.0.0-cp36-cp36m-win_amd64.whl

matplotlib:
https://www.lfd.uci.edu/~gohlke/pythonlibs/#matplotlib

matplotlib-2.1.0-cp36-cp36m-win_amd64.whl

6. numpy的核心

1) numpy数组可作为整体参与数值运算，相比python的list容器，可以省略很多循环语句。

例程：向量加法
01_primer/01_vector.py

2) numpy数组的输出与python的list容器不同，元素之间不以逗号分隔。

二、numpy基础

1. 数组对象

1) numpy中的ndarray是一个多维数组类型，其中包括：
   实际的数据；
   描述数据的元数据。
   大部分针对数组的操作仅仅是修改其元数据部分，而并不修改其底层的实际数据。
2) numpy数组一般是同质的，即数组中所有元素的类型完全一致。
3) numpy数组的下标从0开始，最后一个元素的下标为数组长度减1。
4) 数组对象的dtype属性表示其元素的数据类型。
5) 数组对象的shape属性通过元组描述数组的维度，其中每个元素表示一个维度的大小。

numpy.arange: 创建一维数组；
numpy.array : 创建多维数组。

例程：数组的维度
02_basic/01_shape.py

2. 多维数组

1) numpy.array函数可以根据给定的对象创建数组。
   给定的对象应该是类似列表或元组形式的序列。
2) 可以通过numpy.array函数的dtype参数为所创建数组的元素指定数据类型。

例程：创建多维数组
02_basic/02_array.py

3. 元素索引

数组对象[..., 页索引, 行索引, 列索引]

例程：通过索引访问数组中的元素
02_basic/03_index.py

4. 数据类型

1) 所有的类型都是类

type(1) -> <class 'int'>

2) 类型也是对象，类型对象也有类型，类型对象的类型是type

type(type(1)) -> <class 'type'>
type(type(type(1))) -> <class 'type'>

3) python的数据类型

```
bool
int
float
complex
str
tuple
list
dict
set
```

4) numpy的数据类型

```
numpy.bool_
numpy.int8
numpy.int16
numpy.int32
numpy.int64
numpy.uint8
numpy.uint16
numpy.uint32
numpy.uint64
numpy.float16
numpy.float32
numpy.float64
numpy.complex64
numpy.complex128
numpy.str_
```

5) 自定义的数据类型

```
numpy.dtype([('name', numpy.str_, 40), ('age', numpy.uint8)]).type
```

例程：数据类型
02_basic/04_types.py

5. numpy.dtype

用numpy.dtype类型的对象，结合numpy.array函数的dtype参数，
为数组元素指定自定义数据类型。

```
t = numpy.dtype (T)
a = numpy.array ([...], dtype=t)
b = numpy.array ([...], dtype=T)
```

1) T: numpy内置类型

```
numpy.dtype(numpy.int32)
```

Booleans:

Type	Remarks	Character Code
bool_	compatible: Python bool	'?'

bool8	8 bits	

Integers:

Type	Remarks	Character Code
byte	compatible: C char	'b'
short	compatible: C short	'h'
intc	compatible: C int	'i'
int_	compatible: Python int	'l'
longlong	compatible: C long long	'q'
intp	large enough to fit a pointer	'p'
int8	8 bits	
int16	16 bits	
int32	32 bits	
int64	64 bits	

Unsigned integers:

Type	Remarks	Character Code
ubyte	compatible: C unsigned char	'B'
ushort	compatible: C unsigned short	'H'
uintc	compatible: C unsigned int	'I'
uint	compatible: Python int	'L'
ulonglong	compatible: C long long	'Q'
uintp	large enough to fit a pointer	'P'
uint8	8 bits	
uint16	16 bits	
uint32	32 bits	
uint64	64 bits	

Floating-point numbers:

Type	Remarks	Character Code
half		'e'
single	compatible: C float	'f'
double	compatible: C double	
float_	compatible: Python float	'd'
longfloat	compatible: C long float	'g'
float16	16 bits	
float32	32 bits	
float64	64 bits	
float96	96 bits, platform	
float128	128 bits, platform	

Complex floating-point numbers:

Type	Remarks	Character Code
csingle		'F'

complex_	compatible: Python complex	'D'
cfloat		
clongfloat		'G'
complex64	two 32-bit floats	
complex128	two 64-bit floats	
complex192	two 96-bit floats, platform	
complex256	two 128-bit floats, platform	

Any Python object:

Type	Remarks	Character Code
object_	any Python object	'O'

Flexible types:

Type	Remarks	Character Code
bytes_	compatible: Python bytes	'S#'
unicode_	compatible: Python unicode/str	'U#'
void		'V#'

In the character code, '#' is an integer denoting how many elements the data type consists of.

2) T: numpy泛型类型

numpy.dtype(numpy.integer)

NumPy Generic Types	NumPy Array Scalar Types
integer	int_
signedinteger	
unsignedinteger	uint
number	
inexact	float_
floating	
complexfloating	cfloat
character	bytes_
generic	void
flexible	

3) T: python内置类型

numpy.dtype(int)

Python	NumPy
bool	bool_
int	int_
float	float_
complex	cfloat
bytes	bytes_
unicode	unicode_
str (Python2)	bytes_
str (Python3)	unicode_/str_
buffer	void
(all others)	object_

4) T: 类型字符串

```
numpy.dtype('int32')
```

5) T: 类型字符码

```
numpy.dtype('>(2,3)4i4')
```

The first character specifies the kind of data
and the remaining characters specify the number
of bytes per item, except for Unicode, where it
is interpreted as the number of characters:

Character Code	NumPy/Python type
'?'	boolean
'b'	(signed) byte
'B'	unsigned byte
'i'	(signed) integer
'u'	unsigned integer
'f'	floating-point
'c'	complex-floating point
'm'	timedelta
'M'	datetime
'O'	(Python) objects
'S'/'a'	zero-terminated bytes
'U'	Unicode string
'V'	raw data (void)

Recognized strings can be prepended with:
'>' - big-endian,
'<' - little-endian, or
'=' - hardware-native (the default),
to specify the byte order.

6) T: 变长类型

```
numpy.dtype((numpy.str_, 14))
```

7) T: (定长类型, 维度)

```
numpy.dtype((numpy.float32, 3))
numpy.dtype(((numpy.uint8, 3), 2))
```

8) T: 逗号分隔的多个类型字符串

```
numpy.dtype('U14, 3f4, (2,3)u8')
```

元素中的每个字段用'f0'、'f1'、'f2'等访问。

9) T: [(名称, 类型, 维度), ...]

```
numpy.dtype([
    ('st', numpy.str_, 14),
    ('vt', numpy.float32, 3),
    ('mt', numpy.uint64, (2, 3))])
```

元素中的每个字段用'st'、'vt'、'mt'等访问。

10) T: {'names':..., 'formats':..., 'offsets':..., 'titles':..., 'itemsize':...}

```
numpy.dtype({
    'names': ['st', 'vt', 'mt'],
    'formats': ['U14', '3f4', '(2,3)u8']})
```

11) T: {名称: (类型字符串, 偏移字节数), ...}

```
numpy.dtype({
    'st': ('U14', 0),
    'vt': ('3f4', 56),
    'mt': ('(2,3)u8', 68)})
```

12) T: (源类型, 目标类型)

```
numpy.dtype(('<u2', [('lo', 'u1'), ('hi', 'u1')]))
numpy.dtype(('U6', {'codes': ('6u4', 0)}))
```

例程：数据类型对象
02_basic/05_dtype.py

6. 数组切片

数组[第1维起始:终止:步长, 第2维起始:终止:步长, ...]

1) 一维数组的切片

```
arr = numpy.arange(1, 10)
arr                  # [1 2 3 4 5 6 7 8 9]
arr[:3]              # [1 2 3]
arr[3:6]             # [4 5 6]
arr[6:]              # [7 8 9]
arr[::-1]            # [9 8 7 6 5 4 3 2 1]
arr[:-4:-1]          # [9 8 7]
arr[-4:-7:-1]        # [6 5 4]
arr[-7::-1]          # [3 2 1]
arr[:]               # [1 2 3 4 5 6 7 8 9]
arr[::3]             # [1 4 7]
arr[1::3]            # [2 5 8]
arr[2::3]            # [3 6 9]
```

2) 多维数组的切片

```
arr = numpy.arange(1, 25).reshape(2, 3, 4)
arr                     # [[[ 1  2  3  4]
                        #   [ 5  6  7  8]
                        #   [ 9 10 11 12]]
                        #  [[13 14 15 16]
                        #   [17 18 19 20]
                        #   [21 22 23 24]]]
arr[:, 0, 0]            # [ 1 13]
arr[0]                  # [[ 1  2  3  4]
                        #  [ 5  6  7  8]
                        #  [ 9 10 11 12]]
arr[0, :, :]            # [[ 1  2  3  4]
                        #  [ 5  6  7  8]
                        #  [ 9 10 11 12]]
arr[0, ...]             # [[ 1  2  3  4]
                        #  [ 5  6  7  8]
                        #  [ 9 10 11 12]]
arr[0, 1]               # [5 6 7 8]
arr[0, 1, ::2]          # [5 7]
arr[..., 1]             # [[ 2  6 10]
                        #  [14 18 22]]
arr[:, 1]               # [[ 5  6  7  8]
                        #  [17 18 19 20]]
arr[0, 1, 1::2]         # [6 8]
arr[0, :, -1]           # [ 4  8 12]
arr[0, ::-1, -1]        # [12  8  4]
arr[0, ::2, -1]         # [ 4 12]
arr[::-1, ::-1]         # [[[21 22 23 24]
                        #   [17 18 19 20]
                        #   [13 14 15 16]]
                        #  [[ 9 10 11 12]
                        #   [ 5  6  7  8]
                        #   [ 1  2  3  4]]]
arr[..., ::-1]          # [[[ 4  3  2  1]
                        #   [ 8  7  6  5]
                        #   [12 11 10  9]]
                        #  [[16 15 14 13]
                        #   [20 19 18 17]
                        #   [24 23 22 21]]]
arr[-1, 1:, 2:]         # [[19 20]
                        #  [23 24]]
```

例程：切片
02_basic/06_slice.py

7. 改变维度

1) 视图变维

```
a = numpy.arange(1, 7) # [1 2 3 4 5 6]
b = a.reshape(2, 3)    # [[1 2 3]
                       #  [4 5 6]]
c = b.reshape(6)       # [1 2 3 4 5 6]
```

```
d = b.ravel()           # [1 2 3 4 5 6]
```

2) 复制变维

```
e = b.flatten()         # [1 2 3 4 5 6]
```

3) 就地变维

```
a.shape = (2, 3)        # [[1 2 3]
                        #  [4 5 6]]
a.shape = (6,)          # [1 2 3 4 5 6]
a.resize((2, 3))        # [[1 2 3]
                        #  [4 5 6]]
a.resize((6,))          # [1 2 3 4 5 6]
```

4) 视图转置

```
f = b.transpose()       # [[1 4]
                        #  [2 5]
                        #  [3 6]]
```

注意：一维数组不能转置。

```
g = a.transpose()       # [1 2 3 4 5 6]
h = a.reshape(-1, 1)    # [[1]
                        #  [2]
                        #  [3]
                        #  [4]
                        #  [5]
                        #  [6]]
i = h.transpose()       # [[1 2 3 4 5 6]]
j = i.transpose()       # [[1]
                        #  [2]
                        #  [3]
                        #  [4]
                        #  [5]
                        #  [6]]
```

例程：改变数组的维度
02_basic/07_reshape.py

8. 组合数组

1) 垂直组合

```
v = numpy.vstack((u, d)
v = numpy.concatenate((u, d), axis=0)
```

```
    u             d             v
---------     ---------     ---------
[[a b c]      [[1 2 3]      [[a b c]
 [d e f]  <+>  [4 5 6]]  <V>  [d e f]
 [g h i]]      [7 8 9]]       [g h i]
                              [1 2 3]
                              [4 5 6]
                              [7 8 9]]
```

2) 水平组合

```
h = numpy.hstack((l, r))
h = numpy.concatenate((l, r), axis=1)
```

```
    l            r              h
---------    ---------    ---------------
[[a b c]     [[1 2 3]     [[a b c 1 2 3]
 [d e f]  <+> [4 5 6]  <H> [d e f 4 5 6]
 [g h i]]     [7 8 9]]     [g h i 7 8 9]]
```

3) 深度组合

```
d = numpy.dstack((l, r))
```

```
    l            r                   d
---------    ---------    -----------------------
[[a b c]     [[1 2 3]     [[[a 1] [[d 4] [[g 7]
 [d e f]  <+> [4 5 6]  <D>   [b 2]  [e 5]  [h 8]
 [g h i]]     [7 8 9]]       [c 3]] [f 6]] [i 9]]]
```

4) 行组合

```
r = numpy.row_stack((u, d))
```

```
   u          d          r
-------    -------    ---------
[a b c] <+> [1 2 3] <R> [[a b c]
                         [1 2 3]]
```

二维数组的行组合等价于垂直组合。

5) 列组合

```
c = numpy.column_stack((l, r))
```

```
   l          r          c
-------    -------    -------
[a b c] <+> [1 2 3] <C> [[a 1]
                         [b 2]
                         [c 3]]
```

二维数组的列组合等价于水平组合。

例程：组合数组
02_basic/08_stack.py

9. 分割数组

1) 垂直分割

```
arrs = numpy.vsplit(v, 2)
arrs = numpy.split(v, 2, axis=0)
```

```
    v                   arrs
---------      ----------------------
[[a b c]       [[a b c]       [[1 2 3]
 [d e f]  <V>   [d e f]  <+>   [4 5 6]
 [g h i]        [g h i]]       [7 8 9]]
 [1 2 3]
 [4 5 6]
 [7 8 9]]
```

2) 水平分割

```
arrs = numpy.hsplit(h, 2)
arrs = numpy.split(h, 2, axis=1)
```

```
       h                    arrs
---------------      ----------------------
[[a b c 1 2 3]       [[a b c]       [[1 2 3]
 [d e f 4 5 6]  <H>   [d e f]  <+>   [4 5 6]
 [g h i 7 8 9]]       [g h i]]       [7 8 9]]
```

3) 深度分割

```
arrs = numpy.dsplit(d, 2)
```

```
           d                          arrs
-----------------------      ------------------------------------
[[[a 1] [[d 4] [[g 7]        [[[a] [[d] [[g]       [[[1] [[4] [[7]
  [b 2]  [e 5]  [h 8]   <D>    [b]  [e]  [h]   <+>    [2]  [5]  [8]
  [c 3]] [f 6]] [i 9]]]        [c]] [f]] [i]]]       [3]] [6]] [9]]]
```

例程：分割数组
02_basic/09_split.py

10. 数组的属性与转换

1) 数组的属性

```
numpy.ndarray.dtype    - 元素类型
numpy.ndarray.shape    - 数组维数
numpy.ndarray.ndim     - 数组维度数
numpy.ndarray.size     - 数组元素数
numpy.ndarray.itemsize - 元素字节数
numpy.ndarray.nbytes   - 数组字节数
numpy.ndarray.T        - 转置视图
numpy.ndarray.real     - 实部数组
numpy.ndarray.imag     - 虚部数组
numpy.ndarray.flat     - 扁平迭代器
```

2) 数组的转换

```
numpy.ndarray.tolist() - 转换数组为python列表
numpy.ndarray.astype() - 转换数组中的元素类型
```

例程：数组的属性与转换
02_basic/10_attr.py

三、numpy的通用函数

1. 读取CSV文件

```
dates, opening_prices, highest_prices, \
    lowest_prices, closing_prices = numpy.loadtxt(
        filename, delimiter=',', usecols=(1, 3, 4, 5, 6),
        unpack=True, dtype=numpy.dtype('M8[D], f8, f8, f8, f8'),
        converters={1: dmy2ymd})
```

例程：绘制股价K线图
03_comm/01_candle.py

2. 算数平均值

Samples: S = [s1, s2, ..., sn]

$$\text{Mean: } m = \frac{s1 + s2 + \ldots + sn}{n}$$

```
mean = numpy.mean(closing_prices)
```

例程：用自动方式计算股票的算数平均价格
练习：用手动方式计算股票的算数平均价格
03_comm/02_mean.py

3. 加权平均值

Samples: S = [s1, s2, ..., sn]

Weights: W = [w1, w2, ..., wn]

$$\text{Normalized Weights: } N = [n1, n2, \ldots, nn],\ ni = \frac{wi}{w1 + w2 + \ldots wn}$$

$$\text{Weighted Average: } a = \frac{s1w1 + s2w2 + \ldots + snwn}{w1 + w2 + \ldots + wn} = s1n1 + s2n2 + \ldots + snnn$$

```
vwap = numpy.average(closing_prices, weights=volumes)
```

1) 成交量加权平均价格(Volume Weighted Average Price, VWAP)是一个非常重要的经济学指标，它代表着金融资产的平均价格。某个价格的成交量越大，该价格所占的权重就越高。VWAP就是以成交量为权重计算出来的加权平均价格，常用于算法交易。

例程：计算股票的成交量加权平均价格(VWAP)
03_comm/03_vwap.py

2) 时间加权平均价格(Time Weighted Average Price, TWAP)是另一个非常重要的经济学指标。其基本思想是，相对于远期价格，近期价格的权重更高。

练习：计算股票的时间加权平均价格(TWAP)
03_comm/04_twap.py

4. 最大值和最小值

1) max/min - 数组内取最大/最小

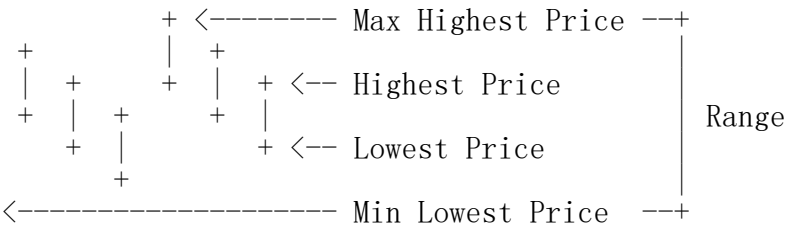
```
    a     numpy.max(a) numpy.min(a)
--------- ------------ ------------
[[9 7 5]
 [3 1 8]       9            1
 [6 6 1]]
```

2) maximum/minimum - 对应元素取最大/最小

```
    a         b     numpy.maximum(a, b) numpy.minimum(a, b)
--------- --------- ------------------- -------------------
[[9 7 5]  [[6 1 9]       [[9 7 9]            [[6 1 5]
 [3 1 8]   [7 1 7]        [7 1 8]             [3 1 7]
 [6 6 1]]  [4 4 5]]       [6 6 5]]            [4 4 1]]
```

例程：最大值和最小值
03_comm/05_max.py

股票在一定时期内最高的最高价和最低的最低价，反映该股在这一时期内的价格变化范围。

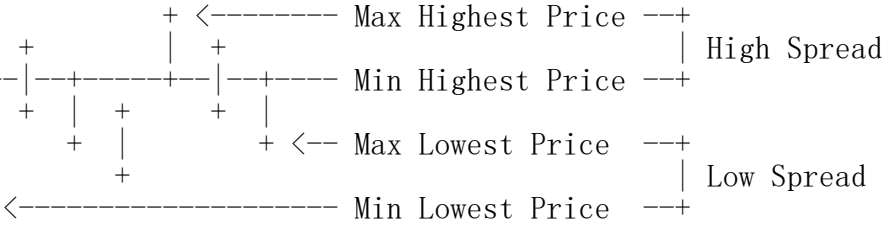


练习：计算股票的价格范围
03_comm/06_range.py

3) ptp - 极差，即数组内最大元素与最小元素之差

```
    a     numpy.max(a) numpy.min(a) numpy.ptp(a)
--------- ------------ ------------ ------------
[[9 7 5]
 [3 1 8]       9            1            8
 [6 6 1]]
```

股票的价格幅度，即该股票的某一种价格在一定时期内的极差。



练习：计算股票的价格幅度
03_comm/07_sprd.py

5. 中位数

将多个样本按大小顺序排列，居于中间位置的元素即为中位数。

奇数个元素：

```
(5-1)/2 == 5/2
        ^

0   1   2   3   4
-----------------
1   3   5   7   9
        |
      Median
```

偶数个元素：

```
(6-1)/2 != 6/2
      ^     ^

0  1  2   3  4  5
----------------
1  2  3   4  5  6
      \  /
 Arithmetic Mean
       \/
     Median
```

无论奇偶，中位数总可以通过公式计算得到：M = (A[(L-1)/2] + A[L/2]) / 2

```
median = numpy.median(closing_prices)
```

例程：用自动方式计算股票的中位价格
练习：用手动方式计算股票的中位价格
03_comm/08_med.py

6. 几个统计学概念

1) 样本

$$\text{Samples: } S = [s1,\ s2,\ \ldots,\ sn]$$

2) 均值

$$\text{Average: } m = \frac{s1 + s2 + \ldots + sn}{n}$$

3) 离差

$$\text{Deviations: } D = [d1, d2, \ldots, dn] \quad (di = si-m)$$

4) 离差方

$$\text{Squares of Deviations: } Q = [q1, q2, \ldots, qn] \quad (qi = di^2)$$

4) (总体)方差

$$\text{Population Variance: } p = \frac{q1 + q2 + \ldots + qn}{n}$$

5) (总体)标准(方)差

Population Standard Deviation: q = sqrt(p) <- 方均根

6) 样本方差

$$\text{Sample Variance: } s = \frac{q1 + q2 + \ldots + qn}{n - 1} = p \frac{n}{n - 1}$$

7) 样本标准(方)差

Sample Standard Deviation: t = sqrt(s) <- 方均根

注意样本方差和总体方差的区别，一个除n-1而另一个除n。样本方差所关心的并非总离差在每个样本上被分摊的平均值，而是每个可以自由变化的样本对总离差贡献的大小。理论上，任何随机变化的量值，其总体均值是一个常量。虽然这里的样本数可能远小于总体样本数，但我们假设样本均值等于总体均值，也是一个常量，因此在n个随机样本中，可以自由变化的样本其实只有n-1个，剩下的那一个样本可以根据均值常量和其它n-1个样本计算得到，也就是说它对于总离差没有任何贡献。这里的n-1被称为自由度，这样得到的方差是一个无偏估计量。

```
population_variance = numpy.var(closing_prices)
sample_variance = population_variance * closing_prices.size / (
        closing_prices.size - 1)
```

例程：用自动方式计算股票价格的总体方差与样本方差
练习：用手动方式计算股票价格的总体方差与样本方差
03_comm/09_var.py

```
            Samples: S = [s1, s2, s3, ..., sn-1, sn]
                          \  /\  /           \  /
                           \/  \/             \/
         Differences: D = [  d1, d2,   ...,    dn-1 ] (di = si+1-si)

      Simple Returns: R = [  r1, r2,   ...,    rn-1 ] (ri = di / si)

Logarithm of Samples: L = [l1, l2, l3, ..., ln-1, ln] (li = log(si))
                          \  /\  /           \  /
                           \/  \/             \/
 Logarithmic Returns: T = [  t1, t2,   ...,    tn-1 ] (ti = li+1-li)

                     ti = li+1-li = log(si+1)-log(si) = log(si+1/si)
                      |                                      ^
                      +--------------------------------------+
```

```
dif_prices          = numpy.diff(closing_prices)
sim_returns_std     = numpy.std(sim_returns)
log_prices          = numpy.log(closing_prices)
```

```
pos_returns_indices = numpy.where(sim_returns > 0)[0]
```

例程：计算股票的简单收益率和对数收益率
03_comm/10_ret.py

Volatility: v = std(T)/mean(T)/sqrt(1/trading_days)

练习：计算股票的价格波动率
03_comm/11_vol.py

7. 针对时间的数据分析

1) converters

numpy.loadtxt()函数的converters参数是一个反映列号和转换函数间对应关系的字典，numpy.loadtxt()函数在其执行过程中，将调用该函数，以完成对指定列中数据的转换。

```
def dmy2weekday(dmy):
    return datetime.datetime.strptime(str(dmy, encoding='utf-8'),
                                      '%d-%m-%Y').date().weekday()

weekdays, closing_prices = numpy.loadtxt(
    filename, delimiter=',', usecols=(1, 6),
    unpack=True, converters={1: dmy2weekday})
```

例程：分析星期数据
03_comm/12_week.py

2) apply_along_axis

numpy.apply_along_axis()函数会调用另一个由参数指定的函数，同时沿着指定的轴向传入一个数组，并将该函数的返回值沿相同轴向汇集成一个数组，整体返回给调用者。

```
summaries = numpy.apply_along_axis(
    get_summary, 1, indices, opening_prices,
    highest_prices, lowest_prices, closing_prices)
```

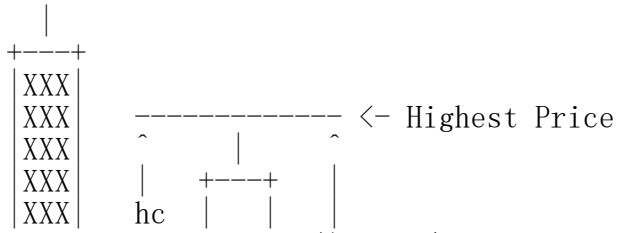
例程：按周汇总数据
03_comm/13_sum.py

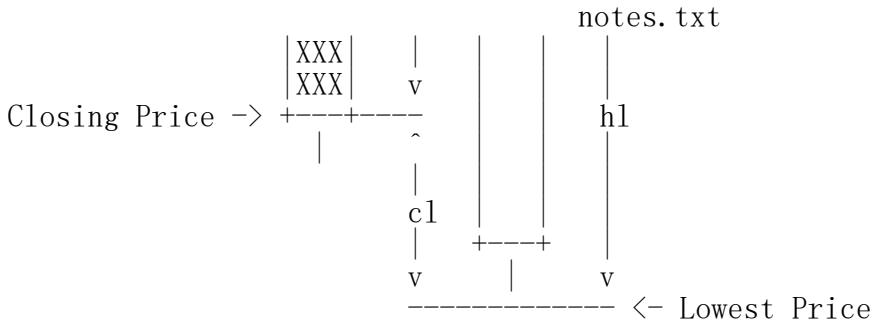
3) maximum/minimum

numpy.maximum()和numpy.minimum()函数可以根据两个数组中对应元素的最大和最小值构建一个新数组。

```
tr = numpy.maximum(numpy.maximum(hc, cl), hl)
```

平均真实波幅(Average True Range, ATR)是一个用来衡量股价波动性的技术指标。





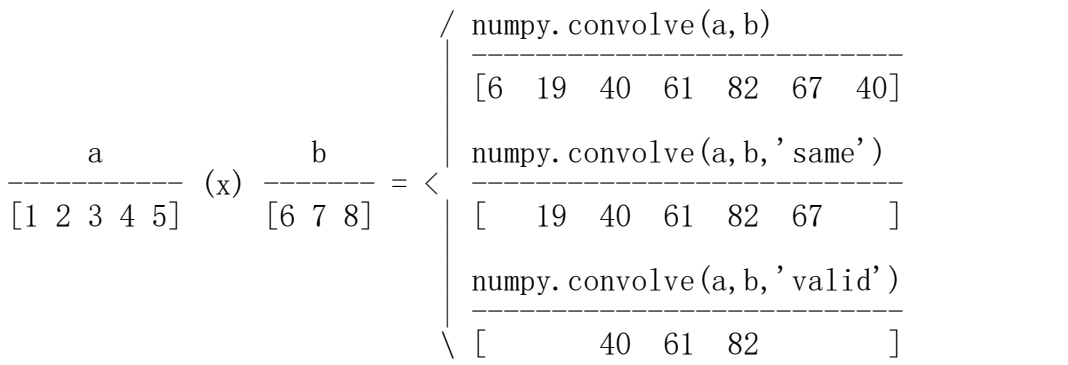
取最近N(通常20)个交易日的最高价、最低价和前日收盘价；
针对每个交易日计算三个价差：
    hc = 当日最高价 - 前日收盘价
    cl = 前日收盘价 - 当日最低价
    hl = 当日最高价 - 当日最低价
三个价差中的最大值即为该交易日的真实波幅(True Range, TR)；
N个交易日中的首日平均真实波幅取各交易日真实波幅的平均值；
其它交易日的平均真实波幅按如下公式计算：

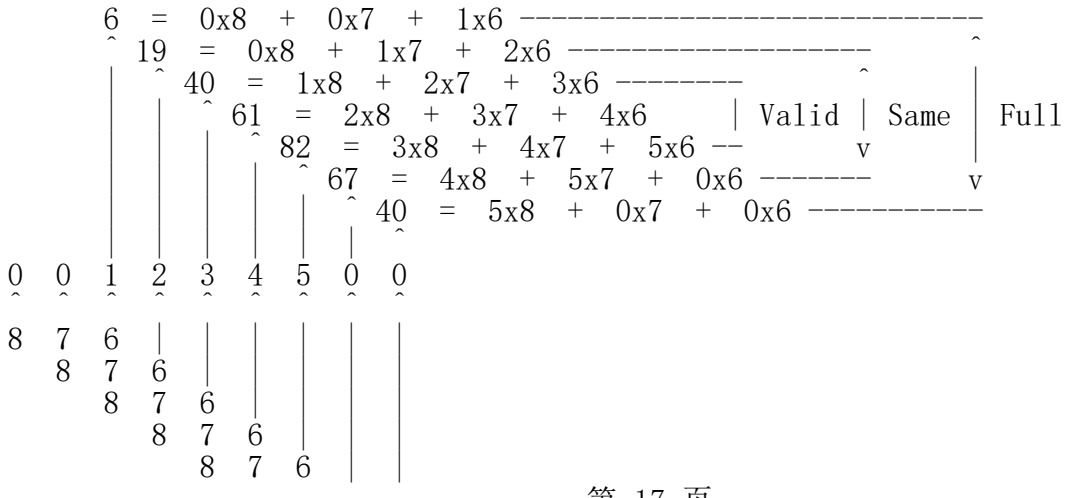
$$ATR[i] = \frac{ATR[i-1] \times (N-1) + TR[i]}{N} \quad (i = 1, 2, \ldots, N-1)$$

练习：计算股票价格的平均真实波幅(ATR)
03_comm/14_atr.py

8. 数组的卷积运算

1) convolve



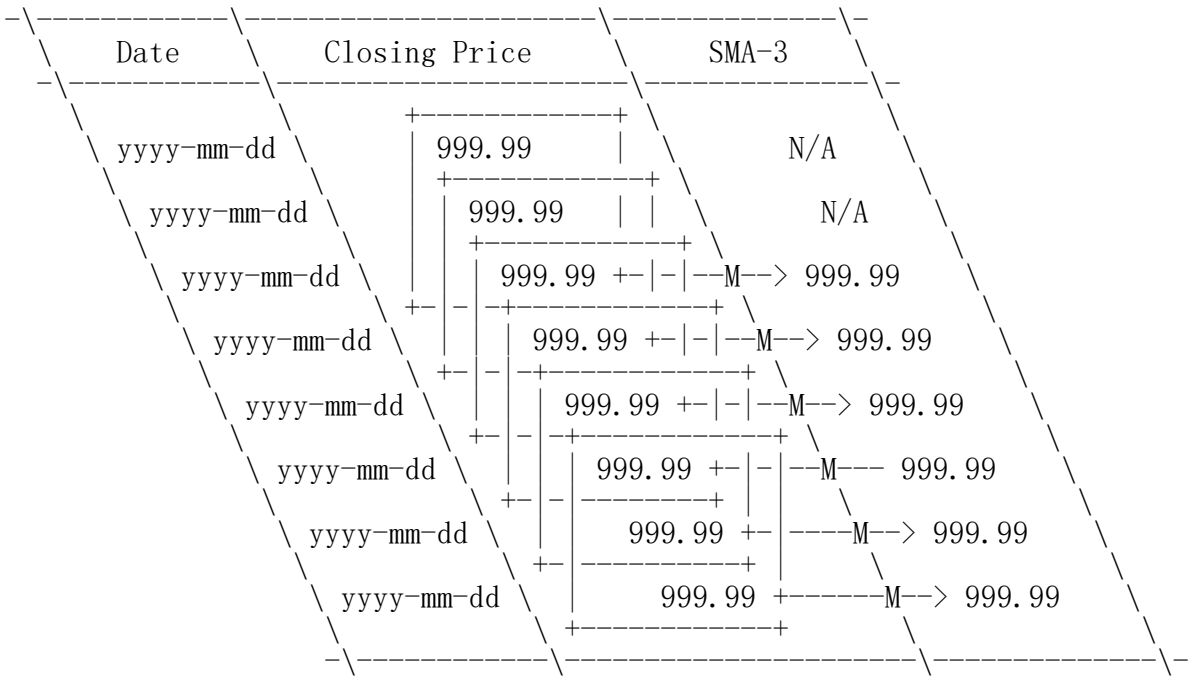


```
              8  7  6  |
                 8  7  6
```

例程：卷积
03_comm/15_conv.py

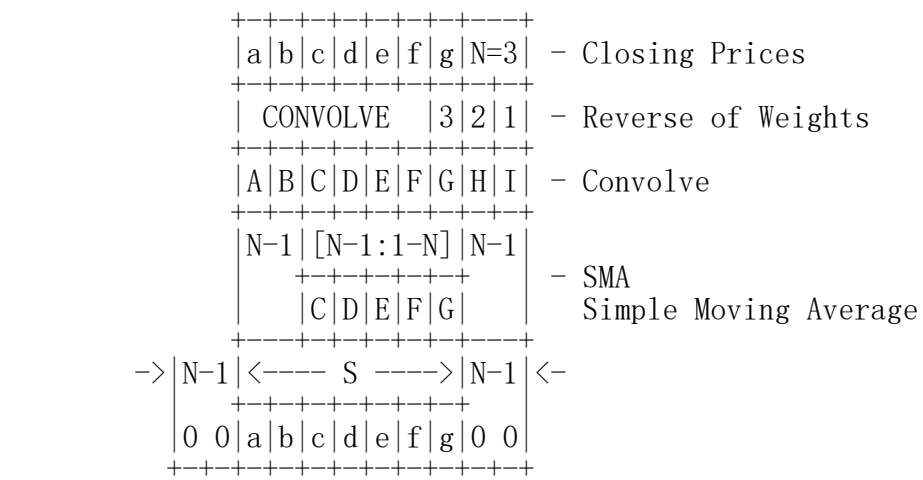
2) 移动平均线

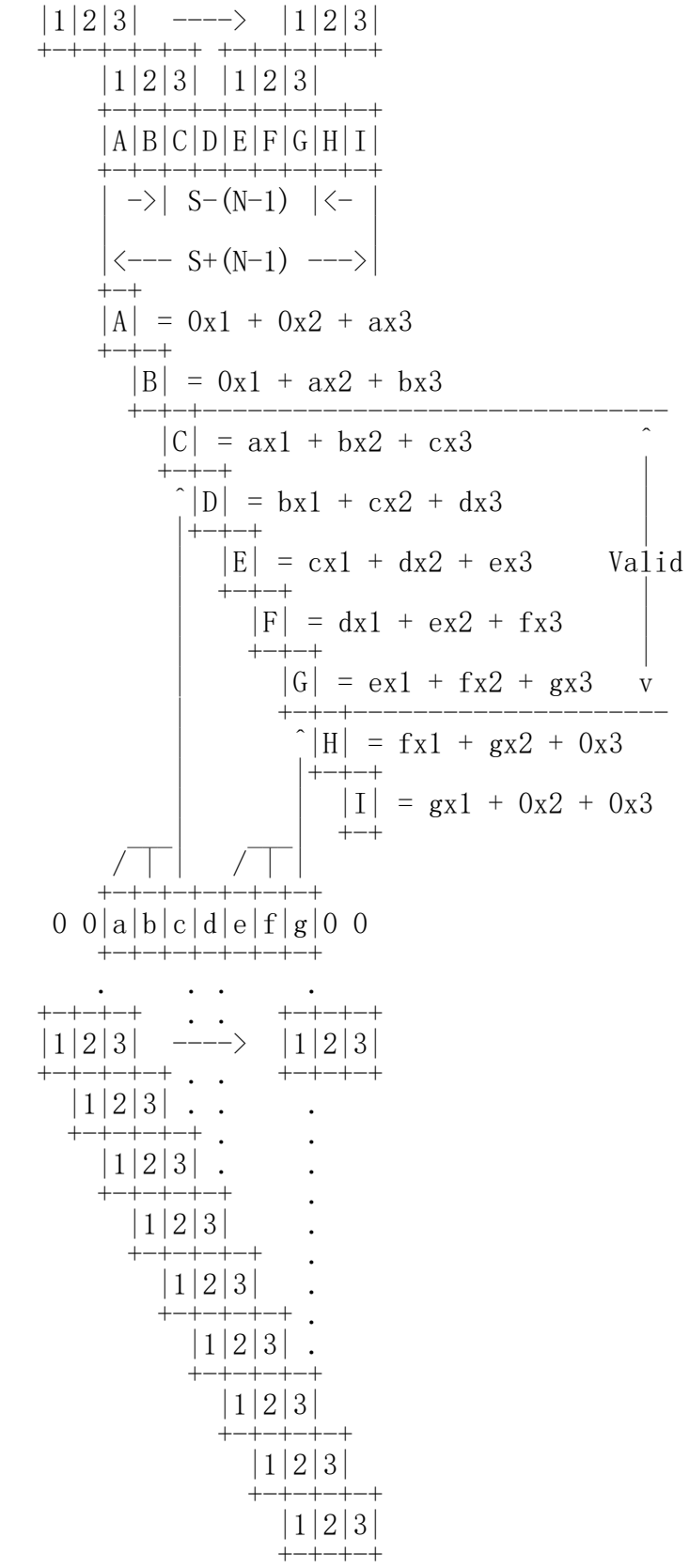
移动平均线(Moving Average, MA)通常用于分析时间序列上的数据。为了计算它，需要事先定义一个包含N个周期(比如N个交易日)的移动窗口，然后按照时间序列滑动该窗口，并计算窗口内数据的加权平均值。若取均权，则为简单移动平均线(Simple Moving Average, SMA)




```
smas = numpy.zeros(closing_prices.size - (time_cycle - 1))
for i in range(smas.size):
    smas[i] = closing_prices[i:i + time_cycle].mean()
```

滑动窗口并求其内部数据平均值的过程可被视作卷积运算，卷积核取归一化的权重之逆。
例如N=5均权重及其卷积核: [0.2 0.2 0.2 0.2 0.2]






```
weights = numpy.ones(time_cycle)
weights /= weights.sum()
smas = numpy.convolve(closing_prices, weights[::-1], 'valid')
```

例程：绘制简单移动平均线(SMA)
03_comm/16_sma.py

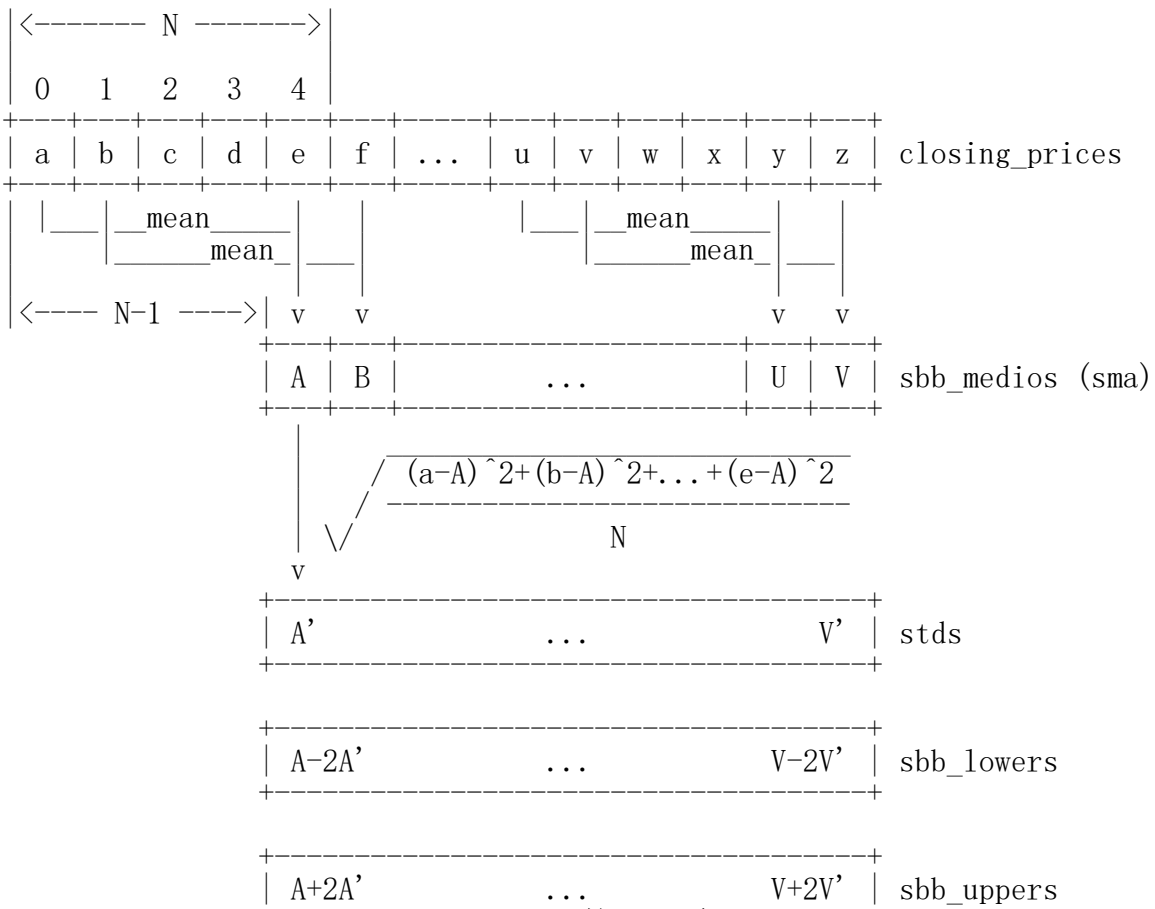
指数移动平均线(Exponential Moving Average)也是一种流行的技术指标。它对移动窗口内数据求加权平均值所使用的权重是一个在时间上由近及远依次递减的归一化指数序列，考虑到越是相对晚近的数据价值越高。例如N=5指数权重[0.114 0.146 0.188 0.241 0.310]，卷积核[0.310 0.241 0.188 0.146 0.114]。

```
weights = numpy.exp(numpy.linspace(-1, 0, time_cycle))
weights /= weights.sum()

emas = numpy.zeros(closing_prices.size - (time_cycle - 1))
for i in range(emas.size):
    emas[i] = closing_prices[i:i + time_cycle].dot(weights)

emas = numpy.convolve(closing_prices, weights[::-1], 'valid')
```

练习：绘制指数移动平均线(EMA)
03_comm/17_ema.py

3) 布林带

布林带(Bollinger Band, BB)的基本形态是由三条轨道线组成的带状通道，其中：
中轨就是简单或指数移动平均线；
上轨比中轨高两倍的标准差；
下轨比中轨低两倍的标准差。



例程：绘制简单布林带(SBB)
03_comm/18_sbb.py

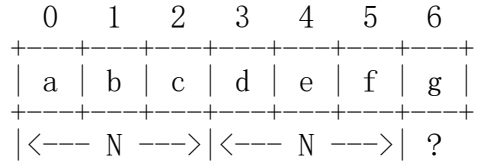
练习：绘制指数布林带(EBB)
03_comm/19_ebb.py

9. 线性模型

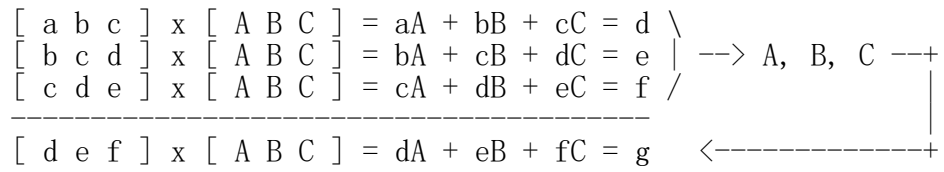
1) 线性预测

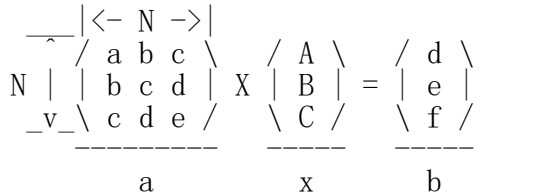
我们姑且假设，一个股价可以用之前股价的线性组合来表示，也就是说，这个股价等于之前一段时间内的股价与各自系数相乘后再相加。一旦获得这些系数，就可以对未来股价进行预测。其实这就是一个线性代数中的最小二乘法问题。

Samples: N = 3

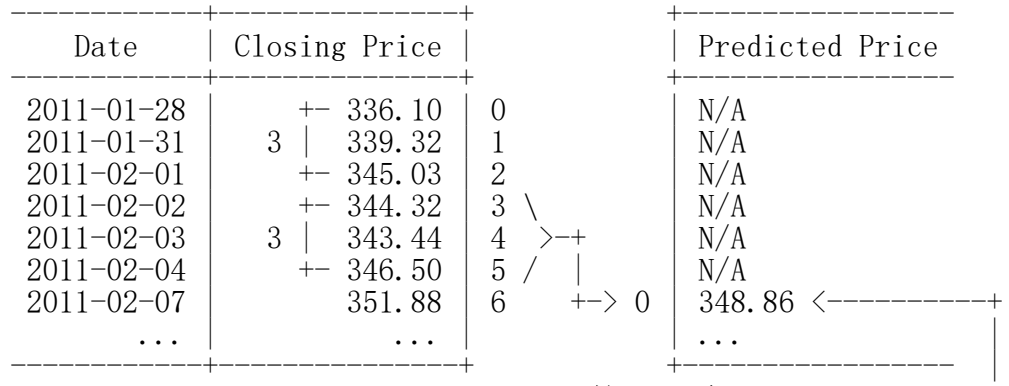


Linearmodel: [A B C]



ax = b
x = numpy.linalg.lstsq (a, b)
g = bx



```
N = 3

    / 336.10 339.32 345.03 \
a = | 339.32 345.03 344.32 |
    \ 345.03 344.32 343.44 /

    / 344.32 \
b = | 343.44 |
    \ 346.50 /

    / 0.60910662 \
x = |-0.39855765 |
    \ 0.79656213 /

bx = 344.32 x  0.60910662 +
     343.44 x -0.39855765 +
     346.50 x  0.79656213 = 348.855745282
```

例程：利用线性模型预测次日股价
练习：验证不同时间周期下线性模型的准确性
03_comm/20_line.py

2) 线性拟合

```
 kx  + b = y

 kx1 + b = y1
 kx2 + b = y2
 kx3 + b = y3
  .    .   .
  .    .   .
  .    .   .
 kxn + b = yn

/ x1 , 1 \           / y1 \
| x2 , 1 |           | y2 |
| x3 , 1 |   / k \   | y3 |
| .    . | X |   | = | .  |
| .    . |   \ b /   | .  |
| .    . |           | .  |
\ xn , 1 /           \ yn /
----------   -----   ------
     a         x        b

ax = b
 x = numpy.linalg.lstsq (a, b)[0]
 k = x[0]
 b = x[1]
[x1 ... xn] x k + b -> [y1 ... yn]
     |                      |
     +----------+----------+
                |
           拟合直线
```

基准位：每个交易周期最高价、最低价和收盘价的算数平均值。

日价差：每个交易周期最高价与最低价之差。
支撑位：每个交易周期基准位下方一个日价差的位置。
阻力位：每个交易周期基准位上方一个日价差的位置。
基准位、支撑位和阻力位的拟合直线，分别构成趋势线、支撑线和阻力线。

例程：绘制股票价格的趋势线
练习：绘制股票价格的支撑线和阻力线
03_comm/21_trend.py

10. ndarray对象的方法

1) clip

numpy.ndarray.clip()方法返回一个修剪过的数组，原数组中所有比给定最大值大的元素全部被设为给定的最大值，而所有比给定最小值小的元素则全部被设为给定的最小值。

a	a.clip(min=3,max=4)
1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 1 1 2 3 3 3 3 3 2 1 1 2 3 4 4 4 3 2 1 1 2 3 4 5 4 3 2 1 1 2 3 4 4 4 3 2 1 1 2 3 3 3 3 3 2 1 1 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1	3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 3 3 3 3 3 3 4 4 4 3 3 3 3 3 3 4 4 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

2) compress

numpy.ndarray.clip()方法返回一个根据给定条件筛选后的数组。

a	a.compress(3<a.ravel()). reshape((3,3))
1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 1 1 2 3 3 3 3 3 2 1 1 2 3 4 4 4 3 2 1 1 2 3 4 5 4 3 2 1 1 2 3 4 4 4 3 2 1 1 2 3 3 3 3 3 2 1 1 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1	4 4 4 4 5 4 4 4 4

3) prod/cumprod

numpy.ndarray.prod()方法返回数组中所有元素的乘积。

```
a: 1     2     3     4     5     6     7     8     9
    \_x_/ \_x_/ \_x_/ \_x_/ \_x_/ \_x_/ \_x_/ \_x_/|
                                                   v
                                        a.prod() -> 362880
```

numpy.ndarray.cumprod()方法返回数组中所有元素的乘积数组。

```
a:              1     2     3     4     5     6     7     8     9
                | _x_/| _x_/| _x_/| _x_/| _x_/| _x_/| _x_/| _x_/|
                v/    v/    v/    v/    v/    v/    v/    v/    v
a.cumprod() -> 1     2     6     24    120   720   5040  40320 362880
```

例程：数组的裁剪、压缩和阶乘
03_comm/22_ndarr.py

四、numpy的便捷函数

1. 相关性

1) 协方差

Samples:

    a: [a1, a2, ..., an]
    b: [b1, b2, ..., bn]

Average:

$$ave(a) = \frac{a1 + a2 + ... + an}{n}$$

$$ave(b) = \frac{b1 + b2 + ... + bn}{n}$$

Deviations:

    dev(a) = [a1, a2, ..., an] - ave(a)
    dev(b) = [b1, b2, ..., bn] - ave(b)

Variance:

    var(a) = ave(dev(a)dev(a))
    var(b) = ave(dev(b)dev(b))

Standard Deviation:

$$std(a) = \sqrt{var(a)}$$

$$std(b) = \sqrt{var(b)}$$

A. 自协方差(即方差)

    cov(a,a) = ave(dev(a)dev(a)) = var(a)

cov(b,b) = ave(dev(b)dev(b)) = var(b)

B. 互协方差

cov(a,b) = ave(dev(a)dev(b))
cov(b,a) = ave(dev(b)dev(a))

随机信号a和b越一致，dev(a)和dev(b)中同正同负的对应元素就越多，乘积均值cov(a,b)就越有可能是一个较大的正数。

随机信号a和b越相悖，dev(a)和dev(b)中一正一负的对应元素就越多，乘积均值cov(a,b)就越有可能是一个较大的负数。

随机信号a和b越无关，dev(a)和dev(b)中同号和异号的比例接近相等，乘积求和正负相抵，其均值cov(a,b)越接近于零。

2) 协方差矩阵

Correlation Matrix:

$$corr = \begin{pmatrix} \frac{cov(a,a)}{std(a)std(a)} & \frac{cov(a,b)}{std(a)std(b)} \\ \frac{cov(b,a)}{std(b)std(a)} & \frac{cov(b,b)}{std(b)std(b)} \end{pmatrix} = \begin{pmatrix} 1 & coco(a,b) \\ coco(b,a) & 1 \end{pmatrix}$$

$$= \frac{\begin{pmatrix} cov(a,a) & cov(a,b) \\ cov(b,a) & cov(b,b) \end{pmatrix}}{\begin{pmatrix} std(a)std(a) & std(a)std(b) \\ std(b)std(a) & std(b)std(b) \end{pmatrix}} \quad \begin{matrix} = covs \\ \\ = stds \end{matrix}$$

互协方差不仅反映了两个随机信号相关性的强弱和方向，同时也包含了每个随机信号自身的波动性，为了消除后者的影响，需要做标准化，即除以它们各自标准差的乘积，而这就是所谓的相关系数：coco(a,b)=cov(a,b)/std(a)std(b)。

3) 相关矩阵

$$\begin{pmatrix} cov(a,a) & cov(a,b) \\ cov(b,a) & cov(b,b) \end{pmatrix} = covs$$

4) 标准差矩阵

$$\begin{pmatrix} std(a)std(a) & std(a)std(b) \\ std(b)std(a) & std(b)std(b) \end{pmatrix} = stds$$

5) 相关系数

Correlation Coefficient:

$$coco(a,b) = \frac{cov(a,b)}{std(a)std(b)} = coco(b,a) \ (\in[-1,1])$$

$$coco(b,a) = \frac{cov(b,a)}{std(b)std(a)} = coco(a,b) \ (\in[-1,1])$$

相关系数的大小反映了两个随机信号间相关性的强弱，其绝对值越接近于1，相关性越强，反之越接近于0，相关性越弱。相关系数的正负则反映了两个随机信号相关性的方向，正值表示正相关，即变化趋势一致，负值表示反相关，即变化趋势相悖。

6) cov/corrcoef

numpy.cov(a,b) -> covs

numpy.corrcoef(a,b) -> corr

例程：用自动和半自动方式计算两只股票收益率的相关系数
练习：用手动方式计算两只股票收益率的相关系数
04_conv/01_corr.py

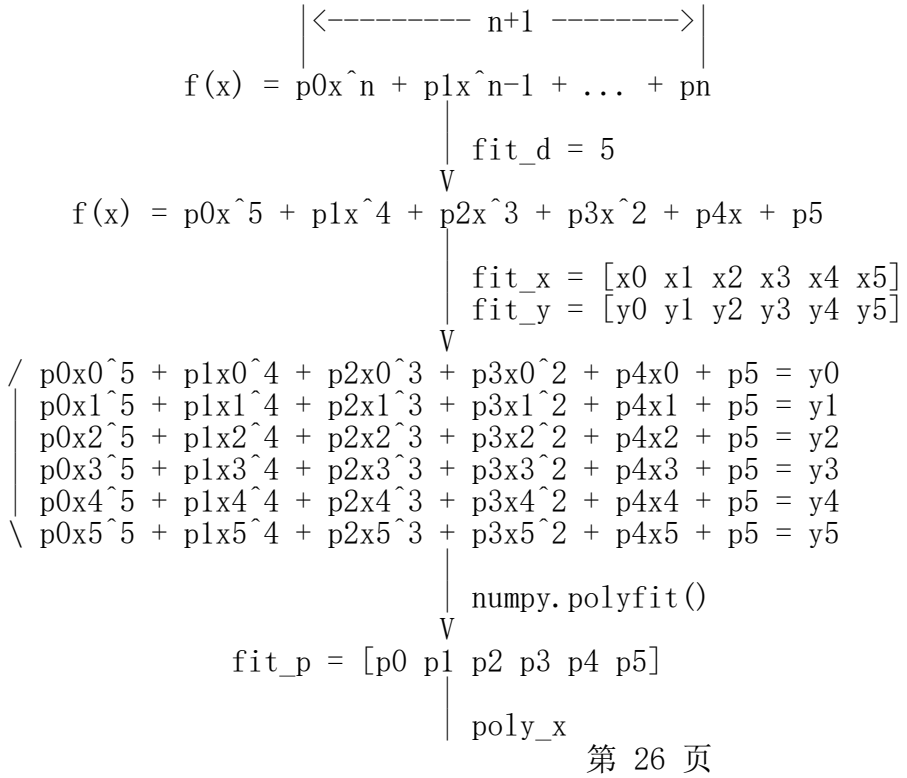
2. 多项式拟合

在微积分里有泰勒展开的概念，即用一个无穷级数表示一个可微的函数。实际上任何可微的函数都可以用一个N次多项式来估计，而比N次幂更高阶的部分作为无穷小量而被忽略不计。

1) 拟合与求值：polyfit()和polyval()

Polynomial Fitting Based On Taylor's Theorem

Any differentiable function f(x) always can be expanded as following:



```
                    | numpy.polyval()
                    V
                  poly_y
```

2) 解多项式方程：numpy.roots ()

```
p0x^n + p1x^n-1 + ... + pn-1x + pn = 0, fit_p = [p0 p1 ... pn  ]
   \_____|_________________/                      |
         |                                        | numpy.roots ()
       x = [...] <------------------------------+
```

3) 多项式函数的导函数：numpy.polyder ()

```
f (x) = p0x^n + p1x^n-1 + ... + pn-1x + pn, fit_p = [p0 p1 ... pn  ]
        |<------------- n+1 ------------>|       |
                                                 | numpy.polyder ()
                                                 v
f'(x) = p0x^n-1 + p1x^n-2 + ... + pn-1    , der_p = [p0 p1 ... pn-1]
        |<------------- n ---------->|
```

例程：用一个5次多项式拟合两只股票收盘价的差价
练习：尝试改变拟合多项式的次数，比较拟合效果的差异
04_conv/02_poly.py

3. 符号数组

1) sign

numpy.sign()函数返回数组中每个元素的正负和零，分别用+1、-1和0表示。

```
signs = numpy.sign(diffs)
```

2) piecewise

numpy.piecewise()函数根据给定的条件对数组中的元素分段取值。

```
signs = numpy.piecewise(diffs, [diffs < 0, diffs == 0, diffs > 0], [-1, 0, 1])
```

净额成交量(On-Balance Volume, OBV)是最简单的股价指标之一，它可以由当日收盘价、前日收盘价和当日成交量计算得出。若当日收盘价高于前日收盘价，则当日OBV等于当日成交量；若当日收盘价低于前日收盘价，则当日OBV取当日成交量之负。若当日收盘等于前日收盘价，则当日OBV为0。

例程：计算股票的净额成交量(OBV)
04_conv/03_obv.py

4. 矢量化

numpy.vectorize()函数可以将一个针对单个数值的处理函数转换为针对数组的处理函数。

```
def calc_profit(buying_rate, opening_price, highest_price,
                lowest_price, closing_price):
    ...
    return profit
```

```
calc_profits = numpy.vectorize(calc_profit)
profits = calc_profits(
    buying_rates, opening_prices, highest_prices,
    lowest_prices, closing_prices)
```

在calc_profits()函数的执行过程中，会用其参数数组中的每一个元素调用calc_profit()函数，并将其返回值组织成一个数组返回给调用者。

模拟每一个交易日，尝试在股价比开盘价下跌一小部分时买入，如果这个价格不在当日股价范围内，则尝试失败，否则以当日收盘价卖出，分别计算所有交易日、盈利交易日和亏损交易日的利润率及其均值。

例程：模拟交易过程
04_conv/04_sim.py

5. 数据平滑

汉宁窗函数numpy.hanning()返回一个余弦函数序列：

[0 0.1882551 0.61126047 0.95048443 0.95048443 0.61126047 0.1882551 0]

将其归一化后可作为卷积核被用于曲线的平滑处理。

[0 0.0537871 0.17464585 0.27156698 0.27156698 0.17464585 0.0537871 0]

```
weights = numpy.hanning(time_cycle)
weights /= weights.sum()
returns = numpy.diff(closing_prices) / closing_prices[:-1]
smrs = numpy.convolve(weights, returns, 'valid')
```

多项式函数f1(x)的系数序列：fit_p1
多项式函数f2(x)的系数序列：fit_p2
多项式函数f3(x)=f1(x)-f2(x)的系数序列：fit_p3 = numpy.polysub(fit_p1, fit_p2)

```
fit_p1 = numpy.polyfit(fit_x, fit_y1, fit_d)
fit_p2 = numpy.polyfit(fit_x, fit_y2, fit_d)
fit_p3 = numpy.polysub(fit_p1, fit_p2)
```

例程：利用汉宁窗平滑两只股票的收益率
练习：找到两支股票收益率的交叉点
04_conv/05_smr.py

五、矩阵与通用函数

1. 创建矩阵

numpy中的矩阵是matrix类型的对象，matrix是ndarray的子类，二者计算规则不同。

```
a = numpy.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
b = numpy.array([[9, 8, 7], [6, 5, 4], [3, 2, 1]])
c = a * b
```

```
    a           b           c
/ 1 2 3 \   / 9 8 7 \   /  9 16 21 \
| 4 5 6 | X | 6 5 4 | = | 24 25 24 |
\ 7 8 9 /   \ 3 2 1 /   \ 21 16  9 /
```

```
a = numpy.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
b = numpy.matrix([[9, 8, 7], [6, 5, 4], [3, 2, 1]])
c = a * b
```

```
                b
          /  9   8  7 \
          |  6   5  4 |
          \  3   2  1 /
        X       =
/ 1 2 3 \ / 30  24 18 \
| 4 5 6 | | 84  69 54 |
\ 7 8 9 / \138 114 90 /
    a           c
```

```
a = numpy.matrix ('1 2 6; 3 5 7; 4 8 9') --- 通过字符串创建矩阵
b = numpy.matrix (a)                     --- b从a复制了一份数据
c = numpy.matrix (a, copy=False)         --- c和a共享同一份数据
```

```
    a
/ 1 2 3 \
| 4 5 6 |    \
\ 7 8 9 /     |
              |
a[1,1] = 0     \ 修改a
               / a变了
              |
    a         |
/ 1 2 3 \     |
| 4 0 6 |    /
\ 7 8 9 /
```

```
    b
/ 1 2 3 \
| 4 5 6 |    --- b的数据独立于a，不随之而变
\ 7 8 9 /
```

```
    c
/ 1 2 3 \
| 4 0 6 |    --- c的数据与a共享，反映其变化
\ 7 8 9 /
```

numpy.mat(...)等价于numpy.matrix(..., copy=False)

```
a = numpy.mat('1 2 6; 3 5 7; 4 8 9') --- 通过字符串创建矩阵
```

```
    a
/ 1 2 6 \
| 3 5 7 |
\ 4 8 9 /
```

```
b = numpy.mat(a)                      --- b和a共享同一份数据
```

```
    b
/ 1 2 6 \
| 3 5 7 |
\ 4 8 9 /
```

```
c = b.T                                  --- 转置

    c
/ 1 3 4 \
| 2 5 8 |
\ 6 7 9 /

d = c.I                                  --- 求逆

                           d
/ -7.33333333e-01  6.66666667e-02  2.66666667e-01 \
|  2.00000000e+00 -1.00000000e+00  5.20417043e-18 |
\ -1.06666667e+00  7.33333333e-01 -6.66666667e-02 /

e = c * d                                --- 原矩阵x逆矩阵=单位矩阵

                           e
/  1.00000000e+00  0.00000000e+00  0.00000000e+00 \
| -1.77635684e-15  1.00000000e+00  0.00000000e+00 |
\ -2.22044605e-16 -1.11022302e-16  1.00000000e+00 /

f = numpy.mat(numpy.array([1, 2, 6, 3, 5, 7, 4, 8, 9]).reshape(3, 3))

    f
/ 1 2 6 \
| 3 5 7 |
\ 4 8 9 /
```

例程：创建矩阵
05_mat/01_mat.py

2. 块拼接

numpy.bmat()函数用于将若干小矩阵组合成大矩阵。

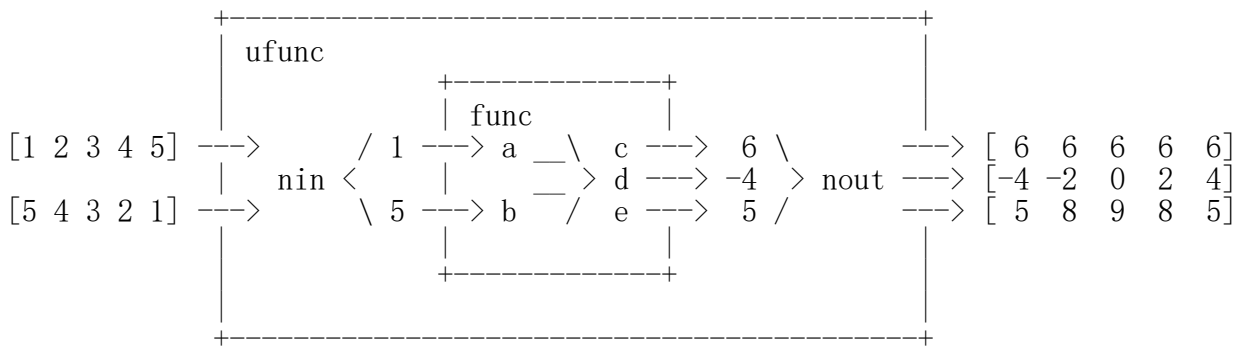
```
a = numpy.ones(dtype=int, shape=(3, 3))

    a
/ 1 1 1 \
| 1 1 1 |
\ 1 1 1 /

b = a * 2

    b
/ 2 2 2 \
| 2 2 2 |
\ 2 2 2 /

c = a + b

    c
/ 3 3 3 \
| 3 3 3 |
\ 3 3 3 /
```

```
d = c + 1

    d
/ 4 4 4 \
| 4 4 4 |
\ 4 4 4 /

e = numpy.bmat('a b; c d')

         e
/ 1 1 1 | 2 2 2 \
| 1 1 1 | 2 2 2 |
| 1 1 1 | 2 2 2 |
| ------+------ |
| 3 3 3 | 4 4 4 |
| 3 3 3 | 4 4 4 |
\ 3 3 3 | 4 4 4 /
```

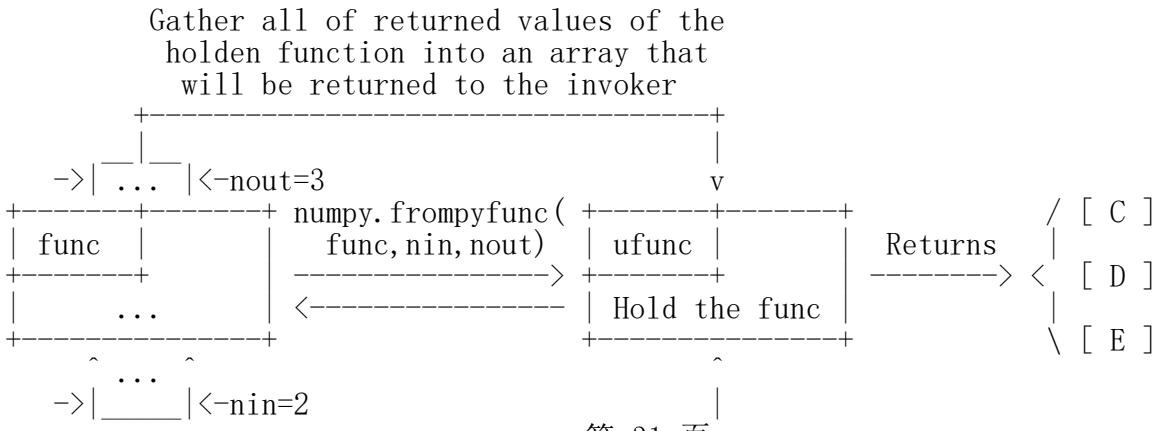
例程：块拼接
05_mat/02_bmat.py

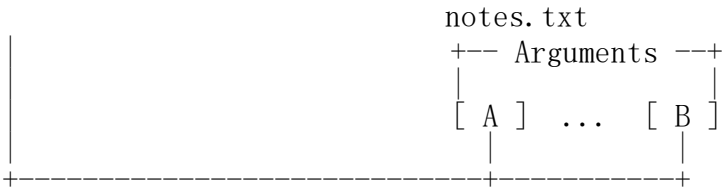
3. 通用函数

通用函数(universal function, ufunc)是numpy.ufunc类型的对象，其并非函数却可被当作函数。通用函数的内部封装了一个函数(func)，及其参数个数(nin)和返回值个数(nout)。用nin个数组作为参数调用通用函数，通用函数将从这些数组中依次提取每一个元素，调用被封装的func函数，并将其返回的nout个值组织成nout个数组返回给通用函数的调用者。



通用函数可通过numpy.frompyfunc(func, nin, nout)函数获得。






```
def func(a, b):
    c = a + b
    d = a - b
    e = a * b
    return c, d, e

A = numpy.array([1, 2, 3, 4, 5])
B = numpy.array([5, 4, 3, 2, 1])
ufunc = numpy.frompyfunc(func, 2, 3)
C, D, E = ufunc(A, B)
print(C, D, E, sep='\n', end='\n\n')
```

```
[ 6  6  6  6  6]
[-4 -2  0  2  4]
[ 5  8  9  8  5]
```

例程：通用函数
05_mat/03_ufunc.py

4. 实现加法运算的通用函数

```
a = numpy.arange(1, 7)
```

```
a : 1   2   3   4   5   6
```

```
b = numpy.add.reduce(a)
```

```
    1 + 2 + 3 + 4 + 5 + 6
    \___\___\___/___/___/
b :           21
```

```
c = numpy.add.accumulate(a)
```

```
    1   2   3   4   5   6
    |/+/|/+/|/+/|/+/|/+/|
c : 1   3   6   10  15 21
```

```
d = numpy.add.reduceat(a, [0, 2, 4])
```

```
    0       2       4
    1 + 2   3 + 4   5 + 6
    \___/   \___/   \___/
d :   3       7       11
```

```
e = numpy.arange(1, 4) * 10
```

```
e : 10 20 30
```

```
f = numpy.add.outer(e, a)

f : |  1   2   3   4   5   6
----+------------------------
 10 | 11  12  13  14  15  16
 20 | 21  22  23  24  25  26
 30 | 31  32  33  34  35  36
```

注意，numpy.add.outer()与numpy.outer()是不一样的。

```
g = numpy.linspace(-3, 3, 7)

g : -3  -2  -1   0   1   2   3

h = numpy.outer(g, g)

h : | -3  -2  -1   0   1   2   3
----+----------------------------
 -3 |  9   6   3   0  -3  -6  -9
 -2 |  6   4   2   0  -2  -4  -6
 -1 |  3   2   1   0  -1  -2  -3
  0 |  0   0   0   0   0   0   0
  1 | -3  -2  -1   0   1   2   3
  2 | -6  -4  -2   0   2   4   6
  3 | -9  -6  -3   0   3   6   9
```

例程：通用函数add的方法
05_mat/04_add.py

5. 实现除法和模运算的通用函数

1) 除法运算

A. 真除：无论运算数是整型还是浮点，运算结果都是浮点数，保留小数。

```
[ 5  5 -5 -5] ÷ [ 2 -2  2 -2] = [ 2.5 -2.5 -2.5  2.5]

numpy.true_divide()
numpy.divide()
/
```

B. 地板除：运算数是整型或浮点，运算结果也是整型或浮点，向下取整。

```
[ 5  5 -5 -5] ÷ [ 2 -2  2 -2] ≈ [ 2 -3 -3  2]

numpy.floor_divide()
//
```

C. 天花板除：运算数是整型或浮点，运算结果也是整型或浮点，向上取整。

```
[ 5  5 -5 -5] ÷ [ 2 -2  2 -2] ≈ [ 3 -2 -2  3]
```

D. 截断除：运算数是整型或浮点，运算结果也是整型或浮点，舍弃小数。

```
[ 5  5 -5 -5] ÷ [ 2 -2  2 -2] ≈ [ 2 -2 -2  2]
```

例程：数组的除法运算
05_mat/05_div.py

2) 模运算

A. 地板模：地板除的余数，余数与除数同号。

```
[ 5  5 -5 -5] ÷ [ 2 -2  2 -2] = [ 2 -3 -3  2] ... [ 1 -1  1 -1]
```

```
numpy.remainder()
numpy.mod()
%
```

B. 截断模：截断除的余数，余数与被除数同号。

```
[ 5  5 -5 -5] ÷ [ 2 -2  2 -2] = [ 2 -2 -2  2] ... [ 1  1 -1 -1]
```

```
numpy.fmod()
```

例程：数组的模运算
05_mat/06_mod.py

6. 算术运算符与相应的通用函数隐式关联

numpy中的数组和矩阵已对基本算术运算符进行了重载，并在其重载实现中调用了相应的通用函数，因此针对numpy数组和矩阵使用基本算术运算符，与直接调用通用函数效果等价。

1) 递归法求斐波那契数列第n项：

```
------+-------------------------------------------------
   x  |   1    2    3    4    5 ...  n-2    n-1    n   ...
------+-------------------------------------------------
      |   1    1    2    3    5 ... 17711  28657 46368 ...
      |   |    |    |    |    |      |      |     |
      | f(1)+f(2)=f(3)   |    |      |      |     |
 f(x) |      f(2)+f(3)=f(4)  |      |      |     |
      |           f(3)+f(4)=f(5)    |      |     |
      |                ...          |      |     |
      |                        f(n-2)+f(n-1)=f(n)
      |                                            ...
------+-------------------------------------------------
```

```
def fibo_recursion(n):
    if n < 3:
        fn = 1
    else:
        fn = fibo_recursion(n - 2) + fibo_recursion(n - 1)
    return fn
```

2) 循环法求斐波那契数列第n项：

```
def fibo_loop(n):
    fn_2, fn_1 = 1, 0
    for i in range(n):
        fn = fn_2 + fn_1
        fn_2, fn_1 = fn_1, fn
```

```
    return fn
```

3) 矩阵法求斐波那契数列第n项：

```
------+-------------------------------------------------------------
   x  |   1 2     3       4       5     ...     n              ...
------+-------------------------------------------------------------
      |   F >   / 1 1 \ / 1 1 \ / 1 1 \
      |   v     \ 1 0 / \ 1 0 / \ 1 0 /
 f(x) | /*1*1 \ /*2 1 \ /*3 2 \ /*5 3 \ ... /*f(n)   f(n-1) \ ...
      | \ 1 0 / \ 1 1 / \ 2 1 / \ 3 2 /     \ f(n-1) f(n-2) /
------+-------------------------------------------------------------
   F  |   F^1     F^2     F^3     F^4   ... F^n-1[0,0] = f(n) ...
------+-------------------------------------------------------------
```

```
def fibo_matrix(n):
    F = numpy.mat('1 1; 1 0')
    fn = (F ** (n - 1))[0, 0]
    return fn
```

4) 公式法求斐波那契数列第n项：

$$f(n) = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

```
def fibo_formula(n):
    V5 = numpy.sqrt(5)
    fn = (((1 + V5) / 2) ** n - ((1 - V5) / 2) ** n) / V5
    return fn
```

例程：用多种方法求取斐波那契数列特定项的值
05_mat/07_fibo.py

7. 实现三角函数的通用函数

1) linspace

```
t = numpy.linspace(-numpy.pi, numpy.pi, 201)
```

在实轴上，将[-π, π]闭区间等分成200份，取包括端点在内的201个点的值。

2) sin/cos/tan

在numpy中所有的标准三角函数均有对应的通用函数，可对数组或矩阵中的每个元素求取其三角函数的值，构成值数组或值矩阵。

$$\text{Lissajous} \begin{cases} x = A\sin(at+\pi/2) \\ y = B\sin(bt) \end{cases}$$

```
def lissajous(A, a, B, b):
```

```
    def lissajou(t):
        x = A * numpy.sin(a * t + numpy.pi / 2)
        y = B * numpy.sin(b * t)
        return x, y
    return numpy.frompyfunc(lissajou, 1, 2)

x, y = lissajous(A, a, B, b)(t)
```

例程：绘制利萨茹曲线
05_mat/08_lissa.py

Squarewaves can be expanded into Fourier Series:

n	1	2	3	...	N	N->∞
f(x)	$\frac{4\sin(x)}{\pi} + \frac{4\sin(3x)}{3\pi} + \frac{4\sin(5x)}{5\pi} + \dots + \frac{4\sin((2N-1)x)}{(2N-1)\pi}$					R->0

$$f(x) = \frac{4}{\pi} \sum_{k=1,3,5,\dots,N}^{N\to\infty} \frac{\sin(kx)}{k}$$

```
def squarewaves(N):
    k = numpy.arange(1, 2 * N, 2)

    def squarewave(x):
        return (numpy.sin(k * x) / k).sum() * 4 / numpy.pi
    return numpy.frompyfunc(squarewave, 1, 1)

y = squarewaves(N)(x)
```

例程：方波信号发生器
05_mat/09_squr.py

Sawtoothwaves can be expanded into Fourier Series:

n	1	2	3	...	N	N->∞
f(x)	$\frac{-2\sin(2\pi x)}{\pi} + \frac{-2\sin(4\pi x)}{2\pi} + \frac{-2\sin(6\pi x)}{3\pi} + \dots + \frac{-2\sin(2N\pi x)}{N\pi}$					R->0

$$f(x) = \frac{-2}{\pi} \sum_{k=1,2,3,\dots,N}^{N\to\infty} \frac{\sin(2k\pi x)}{k}$$

```
def sawtoothwaves(N):
    k = numpy.arange(1, N + 1)

    def sawtoothwave(x):
        return (numpy.sin(2 * k * numpy.pi * x) / k).sum() * (-2) / numpy.pi
    return numpy.frompyfunc(sawtoothwave, 1, 1)
```

Triangularwaves (x) = |Sawtoothwaves (x)|

练习：锯齿波和三角波信号发生器
05_mat/10_saw.py

8. 实现位运算的通用函数

1) ^/__xor__/bitwise_xor

Bitwise exclusive-or:

```
1 ^ 0 = 1
1 ^ 1 = 0
0 ^ 0 = 0
0 ^ 1 = 1

   5 00000101
  -5 11111011
^)-----------
  -2 11111110
```

If a^b<0, their sign are different.

```
a = numpy.arange(-5, 6)
b = -a
c = a ^ b
d = a.__xor__(b)
e = numpy.bitwise_xor(a, b)
f = e < 0
```

2) &/__and__/bitwise_and

Bitwise and:

```
1 & 0 = 0
1 & 1 = 1
0 & 0 = 0
0 & 1 = 0

  32 00100000
  31 00011111
&)-----------
   0 00000000
```

If a&(a-1)==0, it's a power of 2.

```
a = numpy.arange(1, 21)
b = a - 1
c = a & b
d = a.__and__(b)
e = numpy.bitwise_and(a, b)
f = a[e == 0]
```

3) left_shift

Bitwise left shift:

```
   1 00000001 2^0 == 1<<0
   2 00000010 2^1 == 1<<1
   4 00000100 2^2 == 1<<2
   8 00001000 2^3 == 1<<3
  16 00010000 2^4 == 1<<4
         ...
     0010...0 2^n == 1<<n

      ->| n |<-
  91 01011011
  15 00001111 <------+
&)----------        |-1
  11 00001011 == 91%16
```

If b==2^n, a%b==a&(b-1)

```
a = numpy.ones(5, dtype=int)
b = a * 2
c = numpy.arange(5)

d = b ** c
e = a << c
f = a.__lshift__(c)
g = numpy.left_shift(a, c)

h = numpy.arange(24)
i = h % 4
j = h & ((1 << 2) - 1)
```

例程：位运算
05_mat/11_bit.py

六、numpy的模块

1. 线性代数模块(linalg)

线性代数是数学的一个重要分支。numpy.linalg模块包含线性代数的函数。使用这个模块，我们可以计算逆矩阵、求特征值、解线性方程组以及求解行列式等。

1) inv

在线性代数中，矩阵A与其逆矩阵A^-1的乘积是一个单位矩阵I。

Inversion:

AA^-1=E	-0.4375 0.25 0.3125	-0.16666667 0.33333333 -0.16666667	0.39583333 -0.41666667 0.14583333
1 2 3 8 9 4 7 6 5	1. 0. 0.	0. 1. 0.	0. 0. 1.

B = numpy.linalg.inv(A)

例程：计算逆矩阵
06_mod/01_inv.py

2) solve

numpy.linalg中的solve()函数可以解形如Ax=b的线性方程组：

```
/  x - 2y + z = 0
| 2y - 8z - 8 = 0
\-4x + 5y + 9z + 9 = 0

/ 1x + -2y +  1z =  0
| 0x +  2y + -8z =  8
\-4x +  5y +  9z = -9
```

```
            x
          +---+
          | x |
          | y |
          | z |
+---------+---+
| 1 -2  1 | 0 |
| 0  2 -8 | 8 |
|-4  5  9 |-9 |
+---------+---+
     A      b
```

Ax = b
 x = numpy.linalg.solve(A,b)

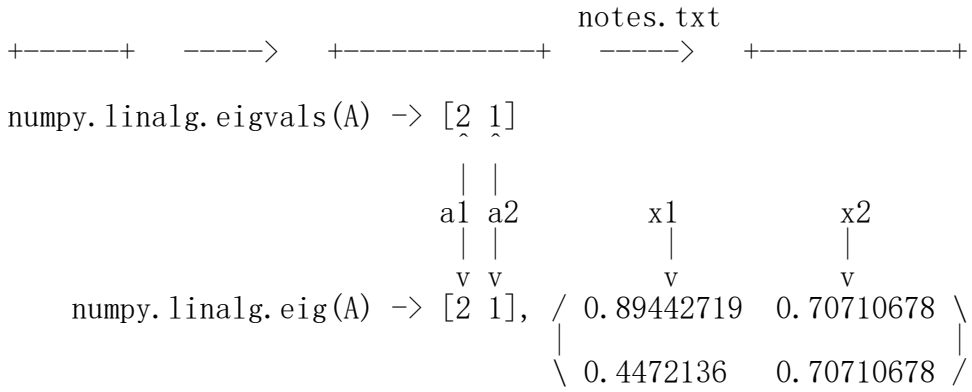
例程：解线性方程组
06_mod/02_solve.py

3) eigvals/eig

对于n阶方阵A，如果存在数a和非零n维列向量x，使得Ax=ax成立，则称a是矩阵A的一个特征值，x是矩阵A属于特征值a的特征向量。

Ax1 = a1x1
Ax2 = a2x2
...
Eigen values of A: a1, a2, ...
Eigen vectors of A: x1, x2, ...

```
                        a1                          a2
                  +------------+              +------------+
          x1      |      2     |      x2      |      1     |
       +----------+------------+--------------+------------+
       | 0.89442719 | a1x1 |       | 0.70710678 | a2x2 |
   A   | 0.4472136  |      v       | 0.70710678 |      v
+------+------------+------------+--------------+------------+
| 3 -2 |            | 1.78885438 |              | 0.70710678 |
| 1  0 |    Ax1     | 0.89442719 |     Ax2      | 0.70710678 |
```



例程：求矩阵的特征值和特征向量
06_mod/03_eig.py

4) svd

奇异值分解(Singular Value Decomposition, SVD)是一种因子分解运算，将一个矩阵分解为三个矩阵的乘积。numpy.linalg模块的svd()函数可以对矩阵进行奇异值分解。该函数返回参数矩阵M的三个因子矩阵U、Σ和V，即M=UΣV，其中U和V是正交矩阵，即UU^T=E，VV^T=E (E表示单位矩阵)，Σ被称为M的奇异值矩阵，其主对角线上的值被称为M的奇异值，非主对角线上的值均为0。

注意svd()函数只返回Σ矩阵主对角线上的值，即所谓奇异值，可以通过numpy.diag()函数得到完整的奇异值矩阵。

Singular Value Decomposition, SVD

M = UΣV

M	Σ	V
4. 11. 14. 8. 7. -2.	18.9736 0. 0. 9.4868	-0.3333 -0.6666 -0.6666 0.6666 0.3333 -0.6666
U	UΣ	UΣV
-0.9486 -0.3162 -0.3162 0.9486	-18. -3. - 6. 9.	4. 11. 14. 8. 7. -2.

U and V are orthogonal matrices:

```
UU^T = E
VV^T = E
```

The diagonal of Σ are singular values of M:

```
18.97366596
 9.48683298
```

例程：对矩阵做奇异值分解
06_mod/04_svd.py

5) pinv

广义逆矩阵将矩阵求逆的运算法则由方阵推广到非方阵，numpy.linalg模块的pinv()函数可

用于对任意矩阵求逆。

Pseudoinversion:

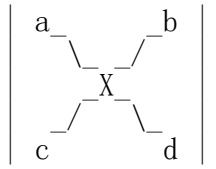
AA^-1=E	-0.18055556 -0.08333333 0.23611111 -0.04305556 0.04166667 -0.00138889 0.09444444 0.16666667 -0.23888889 0.1625 -0.125 0.0375
11 12 13 14 20 21 22 15 19 18 17 16	1. 0. 0. 0. 1. 0. 0. 0. 1.

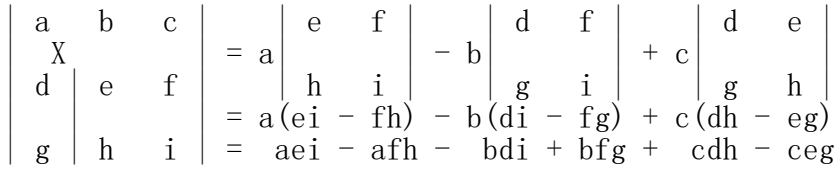
例程：计算广义逆矩阵
06_mod/05_pinv.py

6) det

行列式是与方阵相关的一个标量值。可以调用numpy.linalg模块的det()函数计算方阵的行列式的值。

Determinant:



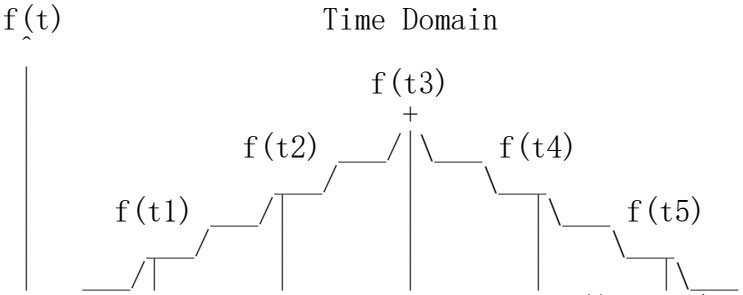


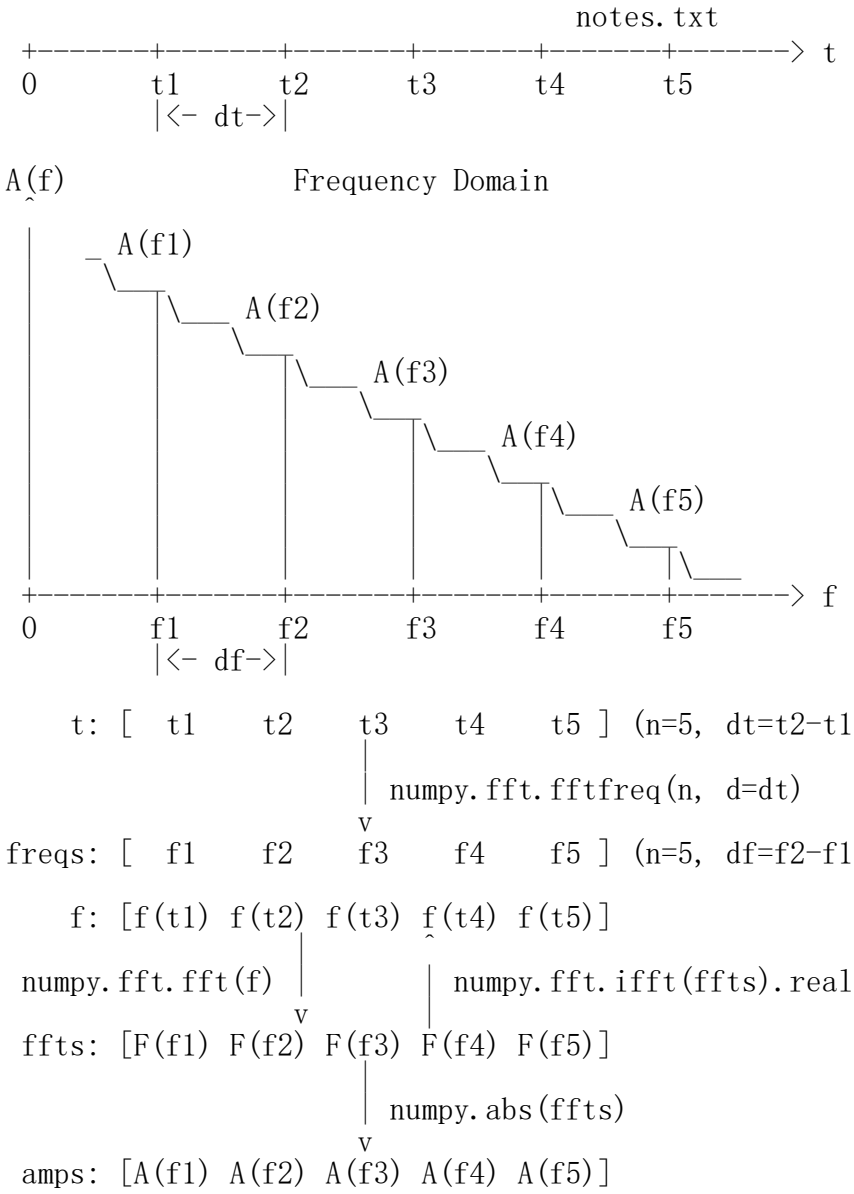
例程：计算矩阵的行列式
06_mod/06_det.py

2. 快速傅里叶变换模块(fft)

1) 快速傅里叶变换(Fast Fourier Transform, FFT)

在numpy中，有一个名为fft的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，也就是说许多函数都存在对应的逆操作函数。



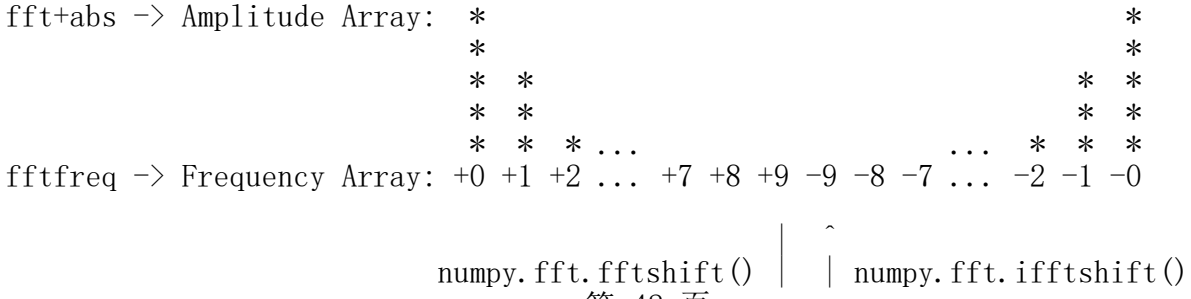


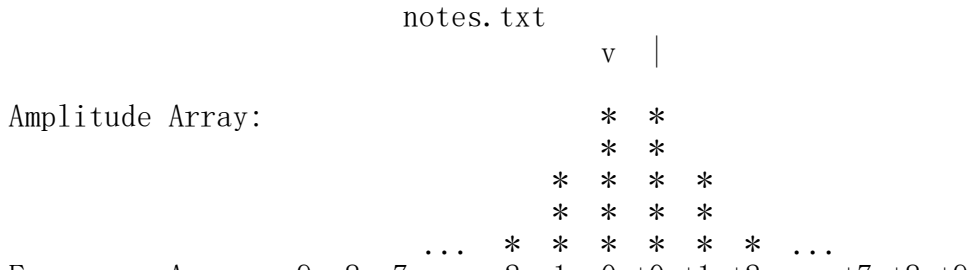

```
    t: [  t1    t2    t3    t4    t5 ] (n=5, dt=t2-t1)
                        |
                        | numpy.fft.fftfreq(n, d=dt)
                        v
freqs: [  f1    f2    f3    f4    f5 ] (n=5, df=f2-f1=1/(ndt)

    f: [f(t1) f(t2) f(t3) f(t4) f(t5)]
                  |           ^
 numpy.fft.fft(f) |           | numpy.fft.ifft(ffts).real
                  v           |
 ffts: [F(f1) F(f2) F(f3) F(f4) F(f5)]
                        |
                        | numpy.abs(ffts)
                        v
 amps: [A(f1) A(f2) A(f3) A(f4) A(f5)]
```

例程：快速傅里叶变换及逆变换
06_mod/07_fft.py

2) 移频

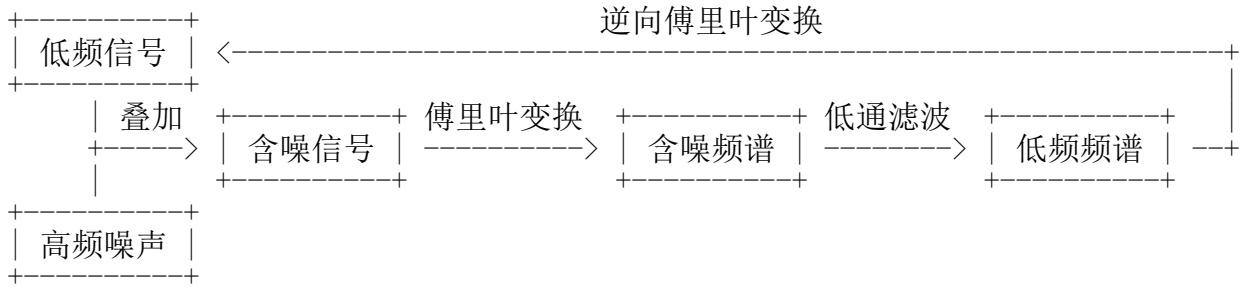
由numpy.fft.fftfreq()和numpy.fft.fft()函数返回的频谱数组，按照从0到+∞再从-∞到0的顺序排列。通过numpy.fft.fftshift()函数的移频操作，可将其变为从-∞到+∞的排列顺序，而numpy.fft.ifftshift()函数则执行相反的移频操作。注意通过numpy.fft.ifft()函数执行逆向傅里叶变换，需要提供按照从0到+∞再从-∞到0顺序排列的频谱数组。





例程：移频
06_mod/08_shift.py

3) 滤波



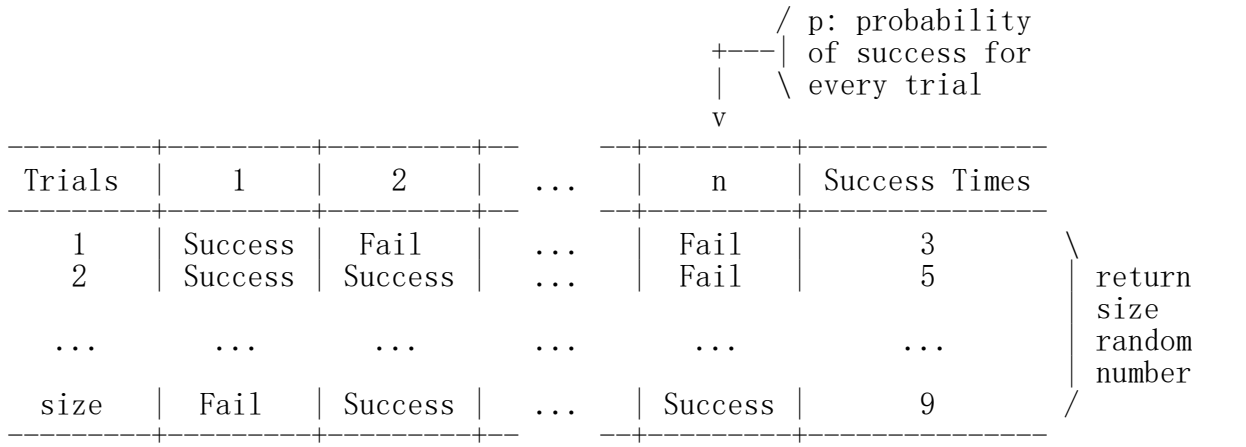
练习：基于快速傅里叶变换的低通滤波器
06_mod/09_filter.py

3. 随机数模块(random)

1) 二项分布

numpy.random.binomial(n, p, size)

产生size个随机数，每个随机数来自n次尝试中的成功次数，其中每次尝试的成功概率为p。



猜硬币游戏：初始筹码1000，每轮猜9次(每次猜对的概率0.5)，猜对5次及5次以上为赢，筹码加1，否则为输，筹码减1，问10000轮以后手里有多少筹码？

outcomes = numpy.random.binomial(9, 0.5, size=rounds)

例程：基于二项分布随机数的猜硬币游戏

06_mod/10_bi.py

2) 超几何分布

numpy.random.hypergeometric(ngood, nbad, nsample, size)

产生size个随机数，每个随机数来自随机抽取的nsample个样本中好样本数，总样本由ngood个好样本和nbad个坏样本组成。

```
---------+-----------------------+---------
         | nsample/(ngood+nbad)  | Number
 Trials  +-----------+-----------+   Of
         |   Good    |    Bad    |  Good
---------+-----------+-----------+---------
    1    |     3     |     0     |    3     \
    2    |     2     |     1     |    2     |
                                             | return size
   ...       ...         ...        ...     | samples
                                             |
  size   |     3     |     0     |    3     /
---------+-----------+-----------+---------
```

摸球游戏：将25个好球1个坏球放在一起，每轮摸3个球，全为好球加1分，摸到坏球减1分，问100轮以后得多少分？

outcomes = numpy.random.hypergeometric(25, 1, 3, size=rounds)

例程：基于超几何分布随机数的摸球游戏
06_mod/11_hyper.py

3) 标准正态分布

numpy.random.normal(size)

产生size个随机数，服从标准正态分布：

f(x) = e^(-x^2/2)/√2π

例程：验证正态分布随机数的理论符合性
06_mod/12_norm.py

4) 标准对数正态分布

numpy.random.lognormal(size)

产生size个随机数，服从标准对数正态分布：

f(x) = e^(-log(x)^2/2)/(x√2π)

例程：验证对数正态分布随机数的理论符合性
06_mod/13_log.py

七、numpy的专用函数

1. 排序和查找

1) lexsort

```
       0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
   a: [0 2 6 3 2 7 9 4 1 9  2  2  5  4  3  4  2  6  3  7]
   b: [4 2 4 7 8 6 4 2 4 7  1  8  6  6  0  6  1  0  6  1]
```

```
c = numpy.lexsort((b, a))
```

返回a数组按升序排列的索引数组，对于a数组中值相同的元素参考其在b数组中对应元素的升序排列。

```
   c: [0 8 10 16 1 4 11 14 18 3 7 13 15 12 17 2 19 5 6 9]

a[c]: [0 1  2  2 2 2  2  3  3 3 4  4  4  5  6 6  7 7 9 9]
b[c]: [4 4  1  1 2 8  8  0  6 7 2  6  6  6  0 4  1 6 4 7]
```

例程：多键间接排序
07_spec/01_sort.py

2) sort_complex

numpy中有专门的复数类型，使用两个浮点数来表示复数。这些复数可以使用numpy的sort_complex()函数进行排序，返回数组按实部的升序排列，实部相同者参考虚部的升序。

```
0+8j 5+3j 0+1j 1+8j 2+7j 3+1j 7+3j 8+5j 6+3j 5+4j
6+2j 4+1j 9+3j 8+6j 3+1j 8+3j 4+2j 3+5j 5+6j 6+8j
                          |
                          | numpy.sort_complex()
                          V
0+1j 0+8j 1+8j 2+7j 3+1j 3+1j 3+5j 4+1j 4+2j 5+3j
5+4j 5+6j 6+2j 6+3j 6+8j 7+3j 8+3j 8+5j 8+6j 9+3j
```

例程：复数排序
07_spec/02_cmp.py

3) argmax/argmin

返回数组中最大/最小值的下标，NaN值既是最大值也是最小值。

4) nanargmax/nanargmin

返回数组中最大/最小值的下标，忽略NaN值。

5) argwhere/where

返回数组中满足给定条件的元素下标。

6) searchsorted

```
     0 1 2 3 4 5 6
a : [1 2 4 5 6 8 9]
b : [7 3]

c = numpy.searchsorted(a, b)
c : [5 2]
```

将b数组中的元素，插入到有序的a数组的c位置上，能得到有序数组。

d = np.insert(a, c, b)

d : [1 2 3 4 5 6 7 8 9]

7) extract
8) nonzero

例程：在数组中查找满足特定条件的元素并获得其下标
练习：从数组中抽取满足特定条件的元素
07_spec/03_search.py

2. 金融计算

1) fv
2) pv
3) npv
4) irr
5) pmt
6) rate
7) nper

例程：金融计算
07_spec/04_fin.py

3. 窗函数

1) bartlett
2) blackman
3) hamming
4) hanning
5) kaiser

例程：分别在时间域和频率域绘制窗函数曲线
07_spec/05_wfunc.py

练习：绘制经不同窗函数平滑后的移动平均线
07_spec/06_ma.py

4. 科学计算

1) i0

例程：绘制贝塞尔函数曲线
07_spec/07_bsl.py

2) sinc

例程：绘制一维及二维辛克函数曲线
07_spec/08_sinc.py

八、断言和单元测试

1. 判等断言

例程：判等断言
08_test/01_equal.py

2. 基于文档字符串的单元测试
08_test/02_docstr.py

九、基于matplotlib的数据可视化

1. 简单绘图

1) title
2) xlabel/ylabel
3) tick_params
4) grid
5) show
6) plot

例程：绘制多项式函数曲线
09_mplot/01_poly.py

例程：在同一张坐标图中绘制多项式函数及其导函数曲线
09_mplot/02_deriv.py

2. 子坐标图

subplot

例程：在三张子坐标图中绘制多项式函数及其一阶、二阶导函数曲线
09_mplot/03_splot.py

3. 定制绘图

1) set_major_locator
2) set_major_formatter
3) candlestick_ohlc
4) autofmt_xdate

例程：绘制股价K线图
09_mplot/04_candle.py

4. 直方图

hist

例程：绘制股价分布直方图
09_mplot/05_hist.py

5. 对数坐标图

semilogy

例程：分别用普通方式和对数方式绘制股票成交量曲线图
09_mplot/06_log.py

6. 散点图

scatter

例程：绘制散点图，用点的位置、大小和
颜色表示股票的价格、成交量及其变化率
09_mplot/07_scatter.py
09_mplot/08_scatter3d.py

练习：随机漫步
09_mplot/09_walk.py

7. 着色

fill_between

例程：对满足特定条件的特定区域着色
09_mplot/10_fill.py

8. 图例和注释

1) legend
2) annotate

例程：图例和注释
09_mplot/11_legend.py

9. 三维绘图

1) from mpl_toolkits.mplot3d import axes3d
2) projection = '3d'
3) meshgrid
4) plot_wireframe

例程：绘制三维线框图
09_mplot/12_wf.py

5) plot_surface

例程：绘制三维表面图
09_mplot/13_sf.py

10. 等高线图

1) contour
2) contourf

例程：绘制普通和着色等高线图
09_mplot/14_cntr.py

11. 条形图

例程：绘制条形图
09_mplot/15_bar.py

12. 饼图

例程：绘制饼图

09_mplot/16_pie.py

13. 热力图

例程：绘制热力图
09_mplot/17_heat.py

14. 日期

例程：显示日期
09_mplot/18_date.py

15. 动画

FuncAnimation

例程：显示动画
09_mplot/19_bub.py
09_mplot/20_sig.py

十、基于pygal的数据可视化

1. 掷骰子

例程：掷一个骰子
10_pygal/01_die.py

练习：同时掷两个面数相同的骰子
10_pygal/02_dice.py

练习：同时掷两个面数不同的骰子
10_pygal/03_diff.py

2. 世界人口

例程：绘制世界人口地图
10_pygal/04_wpop.py

3. Python仓库

例程：调查当前被GitHub托管的Python项目
10_pygal/05_proj.py

十一、基于scipy的科学计算

1. 读写matlab数据文件

1) loadmat
2) savemat

例程：读写matlab数据文件
11_scipy/01_io.py

2. 统计

1) rvs

2) fit
3) skewtest
4) kurtosistest
5) normaltest
6) scoreatpercentile
7) percentileofscore

例程：分析随机数
11_scipy/02_stats.py

 8) ttest_ind
 9) ks_2samp
10) jarque_bera

例程：对比两只股票的对数收益率
11_scipy/03_comp.py

3. 信号处理

1) detrend

例程：绘制趋势线
11_scipy/04_line.py

2) fftfreq
3) fft/ifft

例程：基于快速傅里叶变换的信号滤波
11_scipy/05_filter.py

4. 数学优化

leastsq

例程：基于最小二乘法的曲线拟合
11_scipy/06_fit.py

5. 积分

quad

例程：计算积分
11_scipy/07_integ.py

6. 插值

interpld

例程： 线性插值和三次插值
11_scipy/08_inter.py

7. 多媒体

1) imshow
2) median_filter
3) rotate

4) prewitt

例程：图像处理
11_scipy/09_image.py

5) read
6) tile
7) write

例程：音频处理
11_scipy/10_audio.py

十二、基于pygame的游戏与人工智能

1. pygame基础

1) init
2) set_mode
3) set_caption
4) SysFont
5) render
6) blit
7) get
8) quit
9) update

例程：简单游戏
12_pygame/01_simple.py

2. 显示动画

1) Clock
2) load

例程：显示动画
12_pygame/02_anim.py

3. 在pygame中使用matplotlib

1) use
2) FigureCanvasAgg
3) draw
4) get_renderer

例程：使用matplotlib
12_pygame/03_mplot.py

4. 访问屏幕像素

1) array2d
2) blit_array

例程：访问屏幕像素
12_pygame/04_sfa.py

5. 数据点聚类

1) 机器学习与scikit-learn
2) fit
3) polygon

例程：数据点聚类
12_pygame/05_clust.py

6. OpenGL

1) pyopengl
2) glClear
3) gluOrtho2D
4) glColor3f
5) glBegin
6) glVertex2fv
7) glEnd
8) glFlush

例程：绘制谢尔宾斯基地毯
12_pygame/06_st.py

7. 模拟游戏

例程：生命游戏
12_pygame/07_life.py