前言

这是卡尔曼滤波和贝叶斯滤波的入门教材，它用 Jupyter Notebook 编写而成，你可以用浏览器阅读这本书，同时可以运行和修改书中的代码并且能直接看到结果，还有什么更好的方法来学习呢？

卡尔曼和贝叶斯滤波

传感器具有噪声。世界上充满了我们想要测量和追踪的数据和事件，但是我们不能寄希望与传感器来给我们完美的信息。例如，车上的 GPS 会报告高度信息，但是每次当我经过路上同一位置时 GPS 都会显示细微不同的高度。厨房里的食物秤也会在两次称一样的东西时显示不同的数值。

在简单事件中这个问题的解决方法非常明显。如果我的秤有细微不同的示数我只需要称多次然后取平均值就好，或者是我可以使用一个精度更高的秤。但是我们在传感器具有很大噪声或者是数据采集非常困难的时候该如何做？我们可能尝试追踪低空飞行运动、我们可能为无人机创建自动驾驶仪或者是农场器械在整个田野播种。我工作于机器视觉领域，我需要追踪图像中的移动物体并且视觉算法带有很多噪声和不可靠的结果。

这本书教你怎么去解决这一系列的滤波问题。我在书中使用了很多滤波算法，但是它们都是基于贝叶斯概率的。简单来说，贝叶斯概率就是根据过去的信息来确定哪个可能是真实的。

如果我问你我的车在这次运动中的朝向你可能会束手无策，你会在 1~360° 之间随意猜测，这样你有 1/360 的概率猜对。假设现在我告诉你在两秒之前汽车的朝向是 243°，在两秒内我的车转不了多少，所以你可以做出一个更准确的预测。你可以使用过去的信息来更加准确的推断关于现在或者是未来的信息。

世界也存在噪声。前面的预测可以帮助你更好的估计，但是它也受到噪声的影响，我可能会因为一条狗而刹车或者是绕过一个洞。路上的强风和冰雪是车子的外部影响，在控制文献中我们把这部分叫做噪声，虽然你可能不这么认为。

贝叶斯概率还有更多内容，但是你已经了解了主要思想。结果是不确定的并且我们根据证据的分量来改变我们的信任度。卡尔曼滤波器和贝叶斯滤波器将我们对系统如何表现的噪声和有限的结果与噪声和有限的传感器读数相结合，以产生对系统状态的最佳估计。我们的原则是永不丢弃信息。

假设我们正在追踪一个物体并且有一个传感器突然报告物体改变了方向。它真的转了吗？或者是它的数据有噪声？这就要视具体情况而定了。如果这是一架喷气飞机的话，我更倾向于这是一次突然的改变转向。如果是一辆货车在直行我通常会当作噪声而忽略它。我们会根据传感器的精度来进一步修改我们的信任度。我们的信任度依赖于过去的信息、依赖于我们对所追踪系统的了解和传感器的特征。

卡尔曼滤波由 Rudolf Emil Kálmán 发明来解决一系列数学最优解，它首先用在阿波罗登月计划，从那之后它用在各种领域。卡尔曼滤波用在飞行器、潜艇、巡航导弹。华尔街使用卡尔曼滤波来预测市场。卡尔曼滤波还被用在机器人、物联网传感器和实验室仪器方面。化工厂用卡尔曼滤波来控制和监测反应。医学中被用作去除心脏信号的噪声。如果设计传感器或者是时间序列的数据，卡尔曼滤波通常是需要涉及的。

写这本书的动机

我是一名在航天领域从业近 20 年的软件工程师，所以我经常遇到卡尔曼滤波方面的东西，但是从来没有自己实现过一个。它总是臭名昭著于难学。它的理论非常漂亮，如果你没有学过信号处理、控制理论、概率统计、引导与控制理论的话，那将非常难学。当我需要用计算机视觉解决追踪问题时，我迫切需要自己实现卡尔曼滤波。

这个领域有非常多优秀的教课书例如 Grewal and Andrew's *Kalman Filtering*。但是如果你没有必要的背景知识坐下来阅读这些书是非常痛苦的。最典型的是书的前几章用简短的文字来介绍需要大量时间学习的微积分和概率统计知识。这些书是高等本科或者研究生的教材，而且是研究人员和专业人士的宝贵参考，但这对于业余读者来说确实很难。出现的符号没有解释，一个概念使用多种不同的表示方法，而且这些书几乎没有例子或解决过的问题。我发现我自己能够经常理解这些词语和数学定义，但是不知道这些词语和数学定义描述的现实世界的什么。“那意味着什么呢？”是我反复思考的。下面这个是曾经困惑我的一个典型例子：

$$\hat{x}_k = \Phi_k \hat{x}_{k-1} + G_k u_{k-1} + K_k[z_k - H\Phi_k \hat{x}_{k-1} - HG_k u_{k-1}]$$
$$\mathbf{P}_{k|k} = (I - \mathbf{K}_k\mathbf{H}_k)\text{cov}(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k-1})(I - \mathbf{K}_k\mathbf{H}_k)^{\mathrm{T}} + \mathbf{K}_k\text{cov}(\mathbf{v}_k)\mathbf{K}_k^{\mathrm{T}}$$

然而，在我终于理解了卡尔曼滤波时，我意识到它背后的含义是如此直接。如果你知道一些简单的概率规则并且对融合不确定的数据有一定的直觉，卡尔曼滤波的概念对你来说非常简单。卡尔曼滤波以困难著称，但去掉了许多正式的术语后，这门学科和他们的数学之美对我来说变得清晰起来，我爱上了这个东西。

当我开始理解数学和理论时，更多的困难开始出现。一本书或一篇论文会陈述一些事实，并给出图表作为证据。不幸的是，为什么这个陈述是正确的却没有告诉我或者是无法复现这个图像。或者我想知道“如果 R=0，这是真的吗?”或者作者在如此高的级别上提供伪代码，以至于实现不明显。有些书提供 Matlab 代码，但我没有这个昂贵的许可。最后，许多书在每一章结尾都有许多有用的练习。如果你想自己实现卡尔曼滤波器，你需要理解这些练习，但这些练习没有答案。如果你是在教室里使用这本书，也许这是可以的，但这对独立读者来说是可怕的。 我讨厌作者对我隐瞒信息，可能是为了避免学生在课堂上“作弊”。

所有这些都阻碍了学习。我想跟踪屏幕上的图像，或者为我的 Arduino 项目编写一些代码。我想知道书中的图像是如何形成的，以及选择与作者不同的参数。我想进行模拟。我想在信号中加入更多的噪声，看看滤波器的表现。在日常代码中有成千上万的机会使用卡尔曼滤波器，然而，这个相当直接的话题是火箭科学家和学者的起源。

我写这本书就是为了满足这些需求。如果你设计军用雷达，这不是唯一适合你的书。去伟大的 STEM 学校攻读硕士或博士学位吧，因为你会需要它。这本书是为业余爱好者，好奇者，和工作中需要过滤或平滑的数据工程师。如果你是一个业余爱好者，这本书应该提供了你需要的一切，如果你对卡尔曼滤波器是认真的，你需要更多。我的目的是引入足够的概念和数学，使教科书和论文平易近人。

这本书是交互式的。虽然您可以将其作为静态内容在线阅读，但我建议您按我的预期使用它。它是用 Jupyter Notebook 编写的。这允许我将文本、数学、Python 和 Python 输出组合在一个地方。本书中的每一个图像，每一个数据段都是由 Python 生成的。想要将参数的值加倍?只需更改参数的值，然后按 CTRL-ENTER。将出现一个新的绘图或打印输出。

这本书有练习，但也有答案。我信任你。如果你只是需要一个答案，那就去阅读答案。如果你想内化这些知识，试着在阅读答案之前完成练习。因为这本书是交互式的，你在书中

输入并运行你的解决方案-你不需要移动到不同的环境，或者在开始之前处理一堆东西的导入。

这本书免费。我花了几千美元在卡尔曼滤波书籍上。我不敢相信囊中羞涩的人或经济困难的学生能拿到这些东西。我从像 Python 这样的免费软件和 Allen B Downey 的免费书籍中获得了很多，是时候报答这些了。所以，这本书是免费的，它托管在 GitHub 的免费服务器上，它只使用免费开放的软件，如 IPython 和 MathJax。

在线阅读

GitHub

这本书在 GitHub 上，你可以通过点击它的名字来阅读任何章节。GitHub 静态渲染 Jupyter 笔记本。您不能运行或修改代码，但可以读取所有内容。

这个项目的 GitHub 页面在：

https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python

binder

binder 为在线交互式笔记本提供服务，因此您可以在浏览器中运行代码并更改代码，而无需下载书籍或安装 Jupyter。使用这个链接可以通过活页夹访问这本书:

http://mybinder.org/repo/rlabbe/Kalman-and-Bayesian-Filters-in-Python

nbviewer

nbviewer 网站将以静态格式呈现任何笔记本电脑。我发现它比 GitHub 渲染器做得稍微好一点，但它使用起来有点困难。它直接访问 GitHub;无论我已经检查到 GitHub 将由 nbviewer 渲染。

你可以在这里通过 nbviewer 访问这本书:

http://nbviewer.ipython.org/github/rlabbe/Kalman-and-Bayesian-Filters-in-Python/blob/master/table_of_contents.ipynb

PDF 版本

我定期从笔记本中生成这本书的 PDF 文件。你可以在这里访问:

https://drive.google.com/file/d/0By_SW19c1BfhSVFzNHc0SjduNzg/view?usp=sharing

下载和运行这本书

然而，这本书是交互式的，我建议以这种形式使用它。设置起来要花点功夫，但值得。如果您在计算机上安装了 IPython 和一些支持库，然后克隆这本书，您将能够自己运行书中的所有代码。您可以执行实验，查看过滤器如何响应不同的数据，查看不同的过滤器如何响应相同的数据，等等。我发现这种即时反馈既重要又令人振奋，你不必去想“如果……会发生什么”。试试看!

安装说明可以在安装附录中找到。

软件安装完成后，您可以导航到安装目录并使用命令行指令运行 Juptyer notebook。

这将打开一个浏览器窗口，显示基本目录的内容。这本书按章节编写，每一章的名称为 xx-name.lpynb，其中 xx 为章节编号，.ipynb 是 Notebook 文件扩展名。要阅读第 2 章，请点击第 2 章的链接，这会让浏览器打开该子目录。在每个子目录中将有一个或多个 IPython notebook(所有 notebook 都有一个.ipynb 文件扩展名)。章节内容在笔记本中，与章节名称同名。有时候支持笔记本可以做一些事情，比如生成显示在章节中的动画，这些内容并不打算让终端用户阅读，但如果你对动画是如何制作的感到好奇，当然可以看一看。

无可否认，这对书籍来说是一个很麻烦的界面，我正在跟随其他几个项目的脚步，重新利用 Jupyter Notebook 来生成完整的书籍。我觉得这种轻微的烦恼带来了巨大的回报——当您试图阅读一本书时，您不必下载一个单独的代码库并在 IDE 中运行它，而是所有的代码和文本都在一个地方。如果您想更改代码，您可以这样做，并立即看到更改的效果。如果您发现了一个错误，您可以进行修复，并将其推回我的存储库，这样全世界的每个人都能从中受益。当然，您永远不会遇到我在阅读传统书籍时经常遇到的问题——书籍和代码彼此不同步，您不知道该信任哪一个源代码。

Jupyter

首先，有一些关于如何使用 Jupyter Notebooks 的说明。这本书是交互式的，如果您想运行代码示例，特别是如果您想看到动画绘图，您将需要运行代码单元格。我不能教你所有关于 Jupyter Notebooks 的东西，然而，有几件事会阻碍读者学习。你可以去 http://jupyter.org/ 获取详细的文档。

首先，您必须始终运行最上面的代码单元格，即带有#格式化该书的注释的单元格。它就在正上方。这不仅设置了您可能不关心的格式，而且还加载了一些必要的模块，并对绘图和打印进行了一些全局设置。所以，一直运行这个单元格，除非你只是被动地读取。

这行%matplotlib inline

使绘图在笔记本中显示。Matplotlib 是一个绘图包，如下所述。由于某些原因，我不理解 Jupyter notebook 的默认行为是在外部窗口中生成绘图。

%matplotlib 中的%符号用于 IPython magic - 这些命令用于内核执行不属于 Python 语言的事情。有许多有用的魔法命令，你可以在这里阅读它们:

http://ipython.readthedocs.io/en/stable/interactive/magics.html

在单元中运行代码很容易。单击它，使其具有焦点(将在其周围绘制一个框)，然后按 CTRL-Enter。

其次，单元格必须按顺序运行。我把问题分成几个单元;如果您试图直接跳过并运行第 10 个代码单元格，它几乎肯定不会工作。如果你还没有运行任何东西，只需从单元格菜单项中选择 run All Above。这是确保一切都已运行的最简单方法。

一旦单元运行，您可以经常跳转并以不同的顺序重新运行单元，但并非总是如此。我在努力弥补，但有个折衷的办法。我将在单元格 10 中定义一个变量(比方说)，然后在单元格 11 和 12 中运行修改该变量的代码。如果您返回并再次运行单元格 11，变量将具有在单元格 12 中设置的值，而代码将期望在单元格 10 中设置的值。所以，偶尔你会得到奇怪的结果，如果你运行单元格无序。我的建议是后退一点，并运行单元格，以便再次回到一个适当的状态。这很烦人，但 Jupyter notebook 的交互性足以弥补这一点。更好的是，在 GitHub 上提交一个问题，这样我就知道这个问题并修复它!

最后，一些读者报告了一些浏览器中动画绘图功能的问题。我还没能复制出来。在本书

的某些部分，我使用了%matplotlib 笔记本魔法，它支持交互式绘图。如果这些绘图对您不起作用，请尝试将其更改为内联读取%matplotlib。您将失去动画绘图，但它似乎可以在所有平台和浏览器上工作。

SciPy, NumPy, and Matplotlib

SciPy 是一个数学软件的开源集合。SciPy 中包含 NumPy，它提供数组对象、线性代数、随机数等。Matplotlib 提供 NumPy 数组的绘图。SciPy 的模块复制了 NumPy 中的一些功能，同时添加了优化、图像处理等特性。

为了使我为本书所做的努力易于管理，我选择假设您知道如何用 Python 编程，并且您也熟悉这些包。尽管如此，我还是会花一些时间来说明它们各自的一些特性;实际上，你必须找到外部资源来教你细节。SciPy 的主页 https://scipy.org 是一个完美的起点，不过您很快就会想要搜索相关的教程或视频。

NumPy, SciPy 和 Matplotlib 没有附带默认的 Python 发行版;如果没有安装，请参阅安装附录。

在整本书中，我都使用 NumPy 的数组数据结构，所以现在让我们来了解一下它们。我将教你如何开始;如果您想成为专家，请参阅 NumPy 的文档。

Numpy.array 实现一维或多维数组。它的类型是 numpy.ndarray，我们简称为 ndarray。您可以使用任何列表类对象来构造它。下面从列表中构造一个 1-D 数组:

```
In  [4]: import numpy as np
         x = np.array([1, 2, 3])
         print(type(x))
         x

         <class 'numpy.ndarray'>

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

使用 import numpy as np 已经成为行业标准。你也可以使用元组:

```
In  [5]: x = np.array((4,5,6))
         x

Out[5]: array([4, 5, 6])
```

创建带有嵌套括号的多维数组:

```
In  [6]: x = np.array([[1, 2, 3],
                       [4, 5, 6]])
         print(x)

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

你可以创建三维或三维以上的数组，但这里不需要，所以我就不细说了。

默认情况下，数组使用列表中值的数据类型；如果有多个类型，那么它将选择最准确地代表所有值的类型。因此，如果列表包含 int 和 float 的混合，数组的数据类型将是 float 类

型。您可以使用 dtype 参数来重写它。

```
In [7]: x = np.array([1, 2, 3], dtype=float)
        print(x)

[1. 2. 3.]
```

你可以使用下标位置访问数组元素（索引从 0 开始）：

```
In [8]: x = np.array([[1, 2, 3],
                      [4, 5, 6]])

        print(x[1,2])

6
```

您可以使用片访问列或行。冒号(:)用作下标是对该行或列中所有数据的选择。所以 x[:，0]返回第一列所有数据的数组(0 指定第一列):

```
In [9]: x[:, 0]
Out[9]: array([1, 4])
```

我们可以得到第二行:

```
In [10]: x[1, :]
Out[10]: array([4, 5, 6])
```

获取第二行的最后两个元素:

```
In [11]: x[1, 1:]
Out[11]: array([5, 6])
```

与 Python 列表一样，可以使用负索引来指向数组的末尾。-1 表示最后一个索引。所以另一种获取第二(最后)行的最后两个元素的方法是:

```
In [12]: x[-1, -2:]
Out[12]: array([5, 6])
```

可以用+运算符执行矩阵加法，但矩阵乘法需要 dot 方法或函数。操作符*执行元素级的乘法，这不是线性代数所需要的。

```
In [13]: x = np.array([[1., 2.],
                       [3., 4.]])
         print('addition:\n', x + x)
         print('\nelement-wise multiplication\n', x * x)
         print('\nmultiplication\n', np.dot(x, x))
         print('\ndot is also a member of np.array\n', x.dot(x))

addition:
 [[2. 4.]
 [6. 8.]]

element-wise multiplication
 [[ 1.  4.]
 [ 9. 16.]]

multiplication
 [[ 7. 10.]
 [15. 22.]]

dot is also a member of np.array
 [[ 7. 10.]
 [15. 22.]]
```

Python 3.5 引入了@运算符用于矩阵乘法。

```
In [14]: x @ x
Out[14]: array([[ 7., 10.],
                [15., 22.]])
```

这只在使用 Python 3.5+时有效。这本书需要 3.6 或更高的版本，所以我只要有机会就会使用它。注意，操作符要求两个值都是数组。因此，x @ 3.引发 ValueError，而 np.dot(X, 3)没问题。

你可以用.T 求转置，用 numpy.linalginv 求逆。SciPy 包也提供了逆函数。

```
In [15]: import scipy.linalg as linalg
         print('transpose\n', x.T)
         print('\nNumPy ninverse\n', np.linalg.inv(x))
         print('\nSciPy inverse\n', linalg.inv(x))

transpose
 [[1. 3.]
 [2. 4.]]

NumPy ninverse
 [[-2.   1. ]
 [ 1.5 -0.5]]

SciPy inverse
 [[-2.   1. ]
 [ 1.5 -0.5]]
```

有一些辅助函数，比如 zeros，用来创建一个全是 0 的矩阵，ones 用来得到全是 1 的矩

阵，eye 用来得到单位矩阵。如果需要多维数组，请使用元组指定形状。

```
In [16]: print('zeros\n', np.zeros(7))
         print('\nzeros(3x2)\n', np.zeros((3, 2)))
         print('\neye\n', np.eye(3))
```

```
zeros
 [0. 0. 0. 0. 0. 0. 0.]

zeros(3x2)
 [[0. 0.]
 [0. 0.]
 [0. 0.]]

eye
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

我们有函数来创建等间距的数据。arange 的工作原理很像 Python 的 range 函数，除了它返回一个 NumPy 数组这一点外。linspace 的工作方式略有不同，你用 linspace (start, stop, num)调用它，其中 num 是你想要的数组长度。

```
In [17]: np.arange(0, 2, 0.1)
```

```
Out[17]: array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1,
                1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9])
```

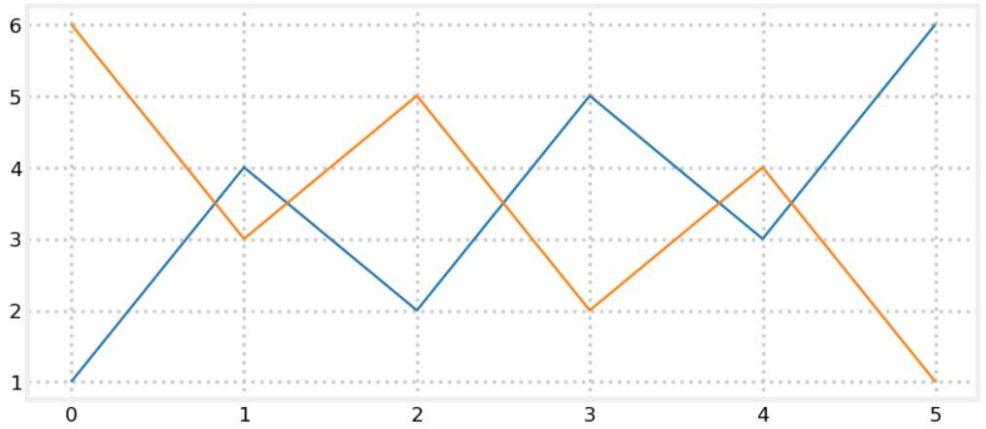
```
In [20]: np.linspace(0, 2, 20)
```

```
Out[20]: array([0.    , 0.105, 0.211, 0.316, 0.421, 0.526, 0.632, 0.737, 0.842,
                0.947, 1.053, 1.158, 1.263, 1.368, 1.474, 1.579, 1.684, 1.789,
                1.895, 2.   ])
```

现在我们来绘制一些数据。在大多数情况下，这是非常简单的。Matplotlib 包含一个绘图库 pyplot。按行业标准进口。导入后，通过调用 plt 来绘制数字。使用数字列表或数组进行绘图。如果您进行多次调用，它将绘制多个系列，每个系列都有不同的颜色。

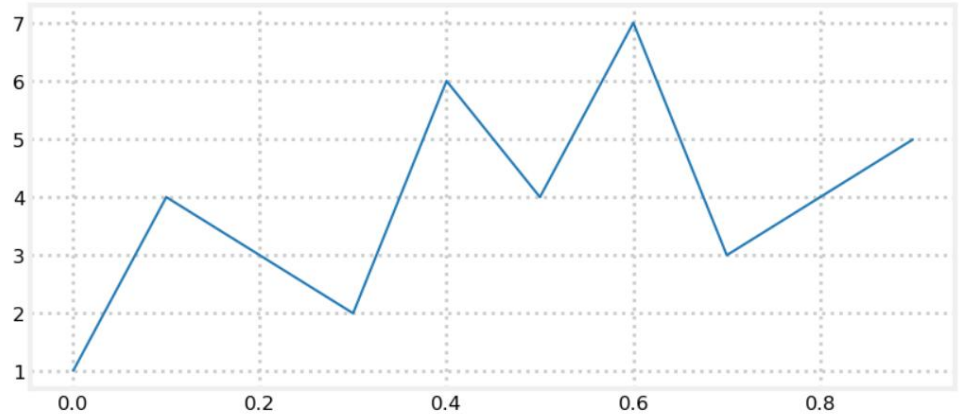
```
In [23]: import matplotlib.pyplot as plt
         a = np.array([6, 3, 5, 2, 4, 1])
         plt.plot([1, 4, 2, 5, 3, 6])
         plt.plot(a)
```

```
Out[23]: [<matplotlib.lines.Line2D at 0x1eff51f7d48>]
```



[<matplotlib.lines.Line2D at 0x2ba160bed68>]是因为 plt.plot 返回的对象刚被创建。通常我们不想看到这个，所以我加上;到我的最后一个绘图命令，以抑制输出。

```
In [24]: plt.plot(np.arange(0, 1, 0.1), [1, 4, 3, 2, 6, 4, 7, 3, 4, 5]);
```



我在本书中使用的这些包还有更多的特性。通常情况下，我会不加解释地介绍它们，相信你可以根据上下文推断用法，或者上网搜索解释。和往常一样，如果你不确定，在 Notebook 中创建一个新单元格或启动 Python 控制台并进行实验!