The g-h Filter

Building Intuition via Thought Experiments

想象一下，我们生活在一个没有秤的世界。一天上班时，一个同事跑到你面前，向你宣布她发明了一种“秤”。在她解释完之后，你急切地站在上面，宣布结果:“172 磅”。你欣喜若狂——这是你生命中第一次知道自己的体重。更重要的是，当你想象着把这个设备卖给世界各地的减肥诊所时，你的眼睛里闪烁着美元的符号!这是太棒了!另一个同事听到骚动，过来看看是什么让你如此兴奋。你解释了一下这个发明，然后再次走上磅秤，骄傲地宣布结果:“161 磅。”然后你犹豫，困惑。

“几秒钟前还是 172 磅”，你向同事抱怨道。

“我从没说过它是准确的，”她回答说。

传感器是不准确的。这就是大量滤波工作背后的动机，而解决这个问题就是本书的主题。我只能提供过去半个世纪发展出来的解决方案，但这些解决方案是通过问一些非常基本的问题发展出来的这些问题涉及到我们所知道的本质以及我们是如何知道的。在我们尝试数学之前，让我们跟随发现之旅，看看它是否会告诉我们关于滤波的直觉。

Try Another Scale

我们有什么办法可以改进这个结果吗?显而易见，首先要尝试的是获得一个更好的传感器。不幸的是，你的同事告诉你，她做了 10 个秤，它们的精确度都差不多。你让她拿出另一个秤，你在一个秤上称自己，然后在另一个秤上称自己。第一个刻度(A)显示为“160 磅”，第二个刻度(B)显示为“170 磅”。关于你的体重我们能得出什么结论?

我们有什么选择?

我们可以选择只相信 A，并将 160 磅作为我们估计的重量。

我们可以选择只相信 B，并相信我们 170 磅的体重。

我们可以选一个比 A 和 B 都小的数。

我们可以选一个比 A 和 B 都大的数。

我们可以在 A 和 B 之间选择一个数字。

前两种选择看似合理，但我们没有理由偏爱其中一种。为什么我们选择相信 A 而不相信 B?我们没有理由有这样的念头。第三和第四个选择是非理性的。诚然，这些刻度不是很准确，但没有理由选择一个超出他们测量范围的数字。最后的选择是唯一合理的。如果两个秤都不准确，给出的结果有可能高于我的实际体重，也有可能低于实际体重，那么答案通常是在 A 和 B 之间的某个地方。

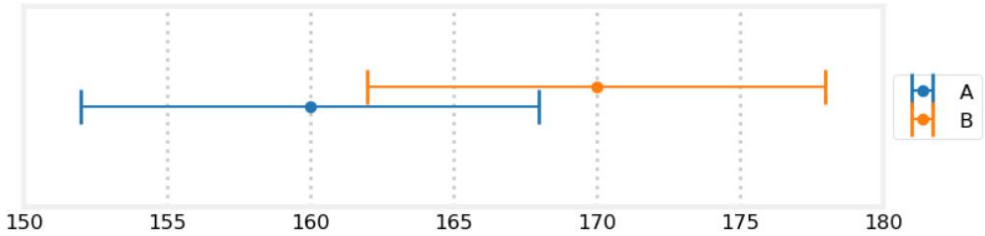
在数学中，这个概念被形式化为期望值，我们稍后将深入讨论它。现在，问问你自己，如果我们读取一百万次数据，会发生什么“通常”的事情。有的时候两个秤的读数都太低，有的时候两个秤的读数都太高，其余的时候它们会跨在实际重量上。如果它们跨越了实际重量，那么我们当然应该在 A 和 B 之间选择一个数字。如果它们没有跨越，我们就不知道它们是过高还是过低，但通过在 A 和 B 之间选择一个数字，我们至少减轻了最糟糕测量的影响。例如，假设我们的实际重量是 180 磅。160 磅是个很大的误差。但如果我们选择 160 磅到 170 磅之间的重量，我们的估计会比 160 磅好。如果两个天平返回的值都大于实际重量，则相同的参数成立。

我们稍后会更正式地处理这个问题，但现在我希望大家清楚，我们的最佳估计是 A 和 B 的平均值。

$$\frac{160 + 170}{2} = 165$$

我们可以直观地看一下。我绘制了 A 和 B 的测量值，假设误差为±8 磅。测量结果在 160 到 170 磅之间，所以唯一合理的重量必须在 160 到 170 磅之间。

```
In [11]: import kf_book.book_plots as book_plots
         from kf_book.book_plots import plot_errorbars
         plot_errorbars([(160, 8, 'A'), (170, 8, 'B')], xlims=(150, 180))
```



谈谈我是如何生成这张图的。我从 kf_book 子目录中的 book_plot 模块导入代码。生成这个情节需要大量读起来毫无意思的 Python 样板文件。我在书中经常采用这种策略。当运行单元格时，将调用 plot_errorbars()，并将 plot 插入到书中。

如果这是您第一次使用 Jupyter Notebook，则上面的代码位于单元格中。文本“In[2]:”将此标记为可以输入输入的单元格，括号中的数字表示此单元格是第二次运行的。要运行单元格，用鼠标单击它，使其具有焦点，然后按下键盘上的 CTRL+ENTER。随着我们继续，您将能够更改单元格内的代码并重新运行它们。尝试将值“160”，“170”和“8”更改为其他值并运行单元格。打印的输出应该根据您输入的内容而改变。

如果想查看 plot_errorbars 的代码，可以在编辑器中打开它，或者创建一个新的单元格，然后输入函数名和两个问号。按下 Ctrl+Enter，您的浏览器将打开一个显示源代码的窗口。这是 Jupyter notebook 的一个特性。如果您只想查看该函数的文档，可以执行相同的操作，但要带一个问号。

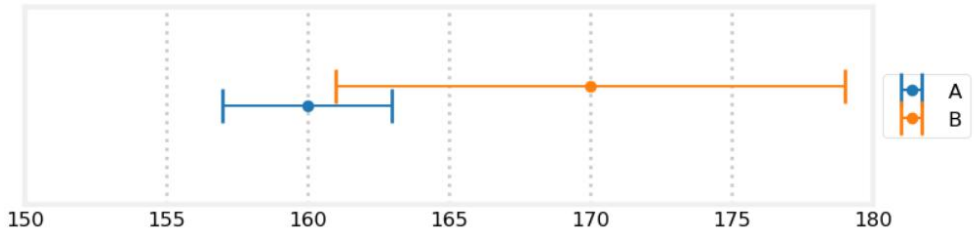
plot_errorbars??或者是 plot_errorbars?

所以 165 磅看起来是一个合理的估计，但是这里有更多的信息我们可以利用。唯一可能的重量位于误差条 A 和 B 之间的交点。例如，重量为 161 磅是不可能的，因为刻度 B 不能给出最大误差为 8 磅的 170 磅读数。同样，169 磅的重量是不可能的，因为刻度 A 不能给出 160 磅的读数，最大误差为 8 磅。在这个例子中，唯一可能的重量在 162 到 168 磅之间。

这还不能让我们找到一个更好的体重估计，但让我们玩一些“如果”。如果我们现在被告知 A 的准确率是 B 的三倍呢?考虑我们上面列出的 5 个选项。选择 A 和 B 范围之外的数字仍然没有意义，所以我们将不考虑这些。也许选择 A 作为我们的估计似乎更有说服力——毕竟，我们知道它更准确，为什么不使用它而不是 B 呢?相对于 A, B 有可能提高我们的知识吗?

答案是肯定的，可能与直觉相反。首先，让我们看看 A=160 和 B=170 的相同测量，但 A 的误差是±3 磅，B 的误差是 A 的 3 倍，±9 磅。

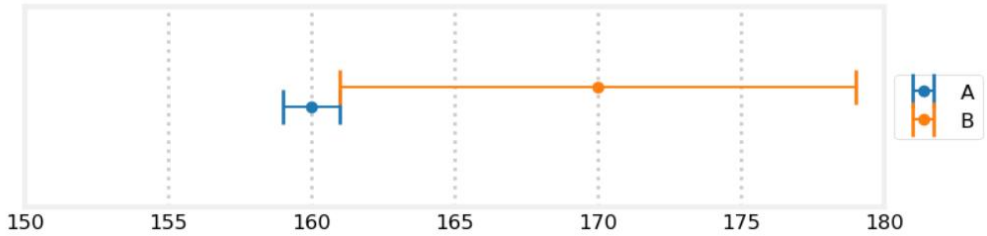
```
In [14]: plot_errorbars([(160, 3, 'A'), (170, 9, 'B')], xlims=(150, 180))
```



A 和 B 误差条的重叠部分是唯一可能的真实权重。这个重叠比单独 A 的误差要小。更重要的是，在这种情况下，我们可以看到重叠部分不包括 160 磅或 165 磅。如果我们只使用 A 的测量值，因为它比 B 更准确，我们会给出 160 磅的估计。如果平均 A 和 B，我们会得到 165 磅。这两种重量都是不可能的因为我们知道秤的精度。通过测量 B，我们可以给出一个介于 161 磅和 163 磅之间的估计值，这是两个物体交点的极限。

让我们把它发挥到极致。假设我们知道秤 A 精确到 1 磅。换句话说，如果我们的真实体重是 170 磅，它可以报告 169、170 或 171 磅。我们还知道刻度 B 精确到 9 磅。我们在每个天平上称重，得到 A=160, B=170。我们应该估计自己的体重是多少?让我们从图形上看一下。

```
In [15]: plot_errorbars([(160, 1, 'A'), (170, 9, 'B')], xlims=(150, 180))
```



在这里我们可以看到，唯一可能的重量是 161 磅。这是一个重要的结果。有了两个相对不准确的传感器，我们就能推断出极其准确的结果。

所以两个传感器，即使一个比另一个精度低，也比一个好。我将在本书的剩余部分反复讨论这个问题。我们从不丢弃信息，不管它有多差。我们将开发数学和算法，使我们能够包括所有可能的信息来源，形成尽可能最佳的估计。

然而，我们已经偏离了我们的问题。没有客户会想要购买多个秤，此外，我们一开始就假设所有的秤都是一样准确的。这种不考虑精度而使用所有度量的见解将在以后发挥很大的作用，所以不要忘记它。

如果我有一个秤，但我称了自己很多次呢?我们的结论是，如果我们有两个精度相等的尺度，我们应该把它们的测量结果平均起来。如果我用一个秤称自己一万次呢?我们已经说过，尺度返回一个太大的数字和返回一个太小的数字的可能性是相等的。证明大量权重的平均值将非常接近实际权重并不难，但现在让我们编写一个模拟。我将使用 NumPy 进行数值计算，它是 SciPy 生态系统的一部分。

```
In [23]: import numpy as np
         measurements = np.random.uniform(160, 170, size=10000)
         mean = measurements.mean()
         print(f'Average of measurements is {mean:.4f}')

Average of measurements is 165.0349
```

打印的确切数字取决于您的随机数生成器，但它应该非常接近 165。

这个代码做出了一个可能不正确的假设——秤上真实体重 165 磅的刻度可能是 160 和 165。这几乎是不可能的。真正的传感器更有可能得到接近真实值的读数，而越来越不可能得到远离真实值的读数。我们将在高斯函数一章中详细讨论这个问题。现在，我将使用 numpy.random.normal()函数，但不作进一步解释，它将在 165 磅附近产生更多的值，在更远的地方产生更少的值。相信现在这将产生噪声测量模拟。

```
In [24]: mean = np.random.normal(165, 5, size=10000).mean()
         print(f'Average of measurements is {mean:.4f}')

Average of measurements is 165.0272
```

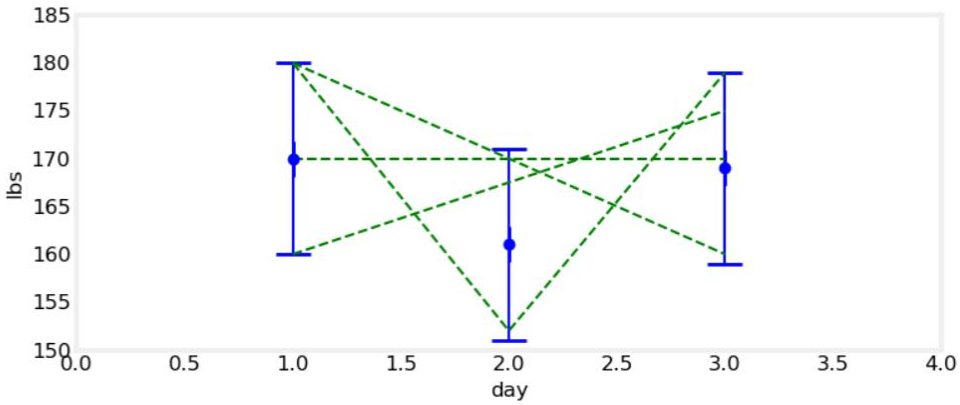
答案再次非常接近 165。

太好了，我们有办法解决传感器的问题了!但这不是一个非常实际的答案。没有人有耐心把自己称一万次，甚至十几次。

所以，让我们来玩“如果”。如果你每天测量一次体重，读数分别是 170、161 和 169。你是变胖了，还是瘦了，还是这些都是测量结果?

我们真的不能说。第一次测量是 170 磅，最后一次测量是 169 磅，这意味着减掉了 1 磅。但如果刻度只能精确到 10 磅，那就可以用噪音来解释了。我本可以增重的;也许我第一天的体重是 165 磅，第三天是 172 磅。在体重增加的同时获得这些体重读数是可能的。我的体重秤告诉我，我的体重在下降，而实际上我的体重在增加!我们来看看图表。我画出了测量值和误差条，然后一些可能的体重增加/减少可以用这些点表示的测量值来解释。

```
In [25]: import kf_book.gh_internal as gh
         gh.plot_hypothesis1()
```



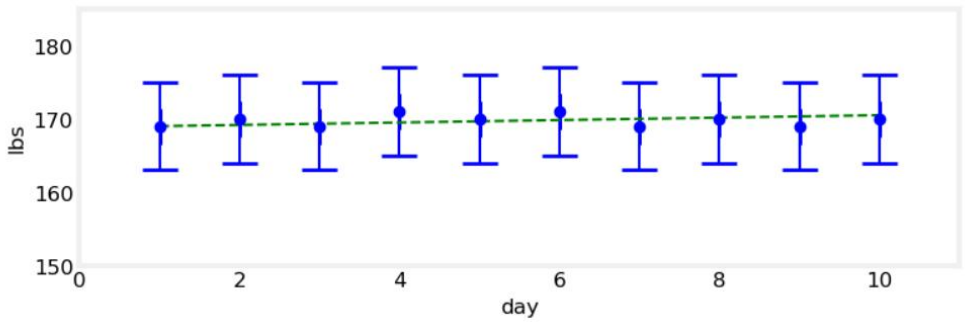
正如我们所看到的，这三次测量方法可以解释一个极端范围的体重变化。事实上，有无数种选择。我们放弃好吗?不是!回想一下，我们正在讨论测量一个人的体重。没有一种合理的方法可以让一个人在第一天重 180 磅，第三天重 160 磅。或者在一天内减掉 30 磅体重，第二天又会反弹回来(我们假设这个人没有遭受截肢或其他创伤)。

我们正在测量的物理系统的行为应该会影响我们如何解释测量结果。如果我们每天都称一块石头的重量，我们会把所有的变化归因于噪音。如果我们给一个被雨水浇灌并用来做家务的蓄水池称重，我们可能会相信这样的重量变化是真实的。

假设我取一个不同的刻度，我得到以下的测量值:169 170 169 171 170 171 171 169 170 169 169 169 170。你的直觉告诉你什么?例如，有可能你每天增重 1 磅，而这些嘈杂的测量结果恰好显示你的体重保持不变。同样，你也可以每天减掉 1 磅，得到相同的读数。但这有可能吗?抛一枚硬币连续得到 10 次正面的可能性有多大?不太可能。我们不能仅仅根据这些读数来证明，但我的体重很可能保持稳定。在下面的图表中，我用误差条绘制了测量值，用绿色虚线表示可能的真实重量。这条虚线并不是这个问题的“正确”答案，它只是一个合理

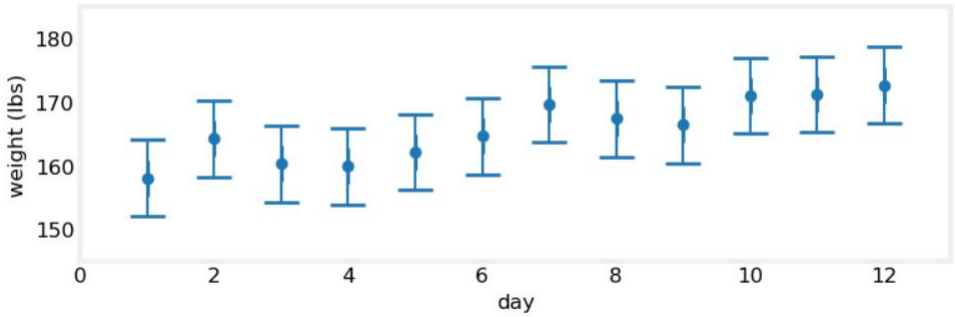
的，可以通过测量来解释的答案。

```
In [26]: gh.plot_hypothesis2()
```



另一个假设是:如果读数是 158.0,164.2,160.3,159.9,162.1,164.6,169.6,167.4,166.4,171.0 呢?让我们看一下图表，然后回答一些问题。

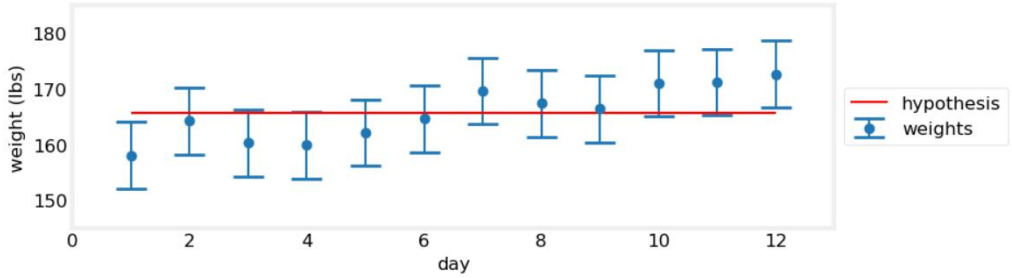
```
In [27]: gh.plot_hypothesis3()
```



是否“看起来”我减肥了，而这只是真正嘈杂的数据?不是真的。我的体重看起来有可能是一样的吗?再一次,没有。随着时间的推移，数据呈上升趋势;不是平均的，但肯定是向上的。我们不能确定，但这看起来像是体重增加了，而且是显著的体重增加。让我们用更多的图来检验这个假设。用图表来“观察”数据通常比用表格更容易。

我们来看两个假设。首先，我们假设体重没有变化。为了得到这个数字，我们同意我们应该平均测量值。我们来看看。

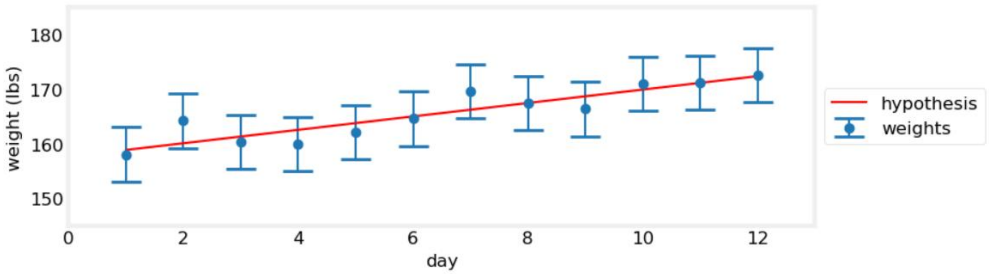
```
In [28]: gh.plot_hypothesis4()
```



这看起来不太有说服力。事实上，我们可以看到，没有一条水平线可以在所有的误差条中画出来。

现在，让我们假设我们增加了体重。增加了多少?我不知道，但是 NumPy 知道!我们想通过测量画一条看起来“大约”正确的线。NumPy 有一些函数可以根据“最小二乘拟合”的规则完成此操作。让我们不要担心计算的细节(如果您感兴趣，我使用 polyfit())，只绘制结果。

```
In [29]: gh.plot_hypothesis5()
```



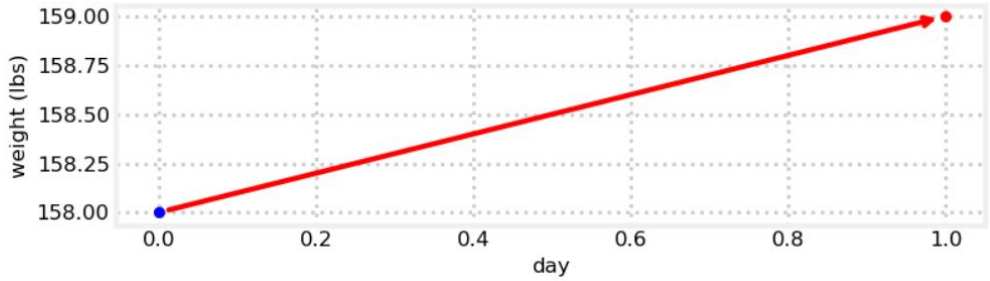
这个看起来好多了，至少在我看来是这样。注意，假设离测量值很近，而在之前的图中假设离测量值很远。我体重增加的可能性比我没有增加任何体重的可能性要大得多。我真的增重了 13 磅吗?谁能说什么?这似乎不可能回答。

“但这是不可能的吗?”一位同事大声问道。

让我们尝试一些疯狂的事情。让我们假设我知道我每天增重一磅。现在我怎么知道的并不重要，假设我知道它是近似正确的。也许我每天摄入 6000 卡路里的食物，会导致体重增加。或者也许有另一种方法来估计体重增加。这是一个思想实验，细节并不重要。让我们看看如果有这样的信息，我们能否利用它。

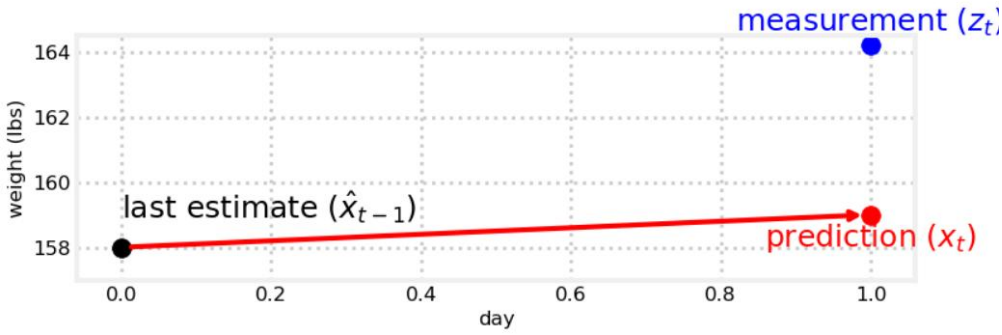
第一次测量是 158。我们没有办法知道任何不同，所以让我们接受这是我们的估计。如果我们今天的体重是 158，明天会是多少?嗯，我们认为我们的体重每天增加 1 磅，所以我们的预测是 159 磅，像这样:

```
In [30]: gh.plot_estimate_chart_1()
```



好吧，但这有什么好处?当然，我们可以假设 1 磅/天是准确的，并预测我们未来 10 天的体重，但如果我们不把读数纳入磅秤，为什么要使用它呢?让我们来看下一个测量。我们再次站到秤上，显示是 164.2 磅。

```
In [31]: gh.plot_estimate_chart_2()
```



我们有麻烦了。我们的预测与测量结果不符。但是，这就是我们期望的，对吧?如果预测总是与测量完全相同，那么它将无法向滤波器添加任何信息。当然，我们没有理由去测量，因为我们的预测是完美的。

整本书的关键见解在下面一段。仔细阅读!

那么我们该怎么办呢?如果我们只是从测量中形成估计，那么预测将不会影响结果。如果我们只是从预测中形成估计，那么测量将被忽略。如果这是工作，我们需要采取某种混合的预测和测量(我已加粗的关键点)。

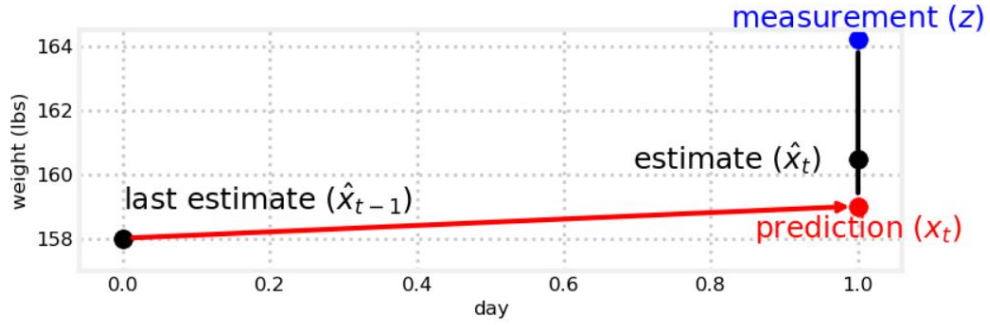
混合两个值，这听起来很像前面的两个尺度问题。使用同样的推理，我们可以看到，唯一有意义的事情是在预测和测量之间选择一个数字。例如，165 的估计没有意义，157 也没有意义。我们的估计应该在 159(预测)和 164.2(测量)之间。

再说一遍，这很重要。我们一致认为，当给出两个有误差的值时，我们应该在两个值之间形成一个估计部分。这些价值是如何产生的并不重要。在本章的开始，我们有两个测量，但现在我们只有一个测量和一个预测。这两种情况的推理和计算方法是相同的。我们从不丢弃信息。我的意思是它。我看到很多商业软件抛弃了嘈杂的数据。不要这样做!我们对体重增加的预测可能不是很准确，但只要有一些信息，我们就应该这样做。

我一定要你停下来好好想想。我所做的只是用基于人类生理的不准确的体重预测代替了不准确的体重秤。它仍然是数据。数学不知道这些数据是来自一个量表还是一个预测。我们有两组带有一定噪声的数据，我们想把它们结合起来。在本书的其余部分，我们将开发一些相当复杂的数学来执行这个计算，但数学从不关心数据来自哪里，它只根据这些值的值和精度进行计算。

估计是否应该介于测量和预测之间?也许吧，但总的来说，我们似乎知道我们的预测与测量相比或多或少是准确的。也许我们预测的准确度与刻度的准确度不同。回想一下，当 A 比 B 精确得多的时候我们是怎么做的，我们把答案缩小到 A 比 B 更接近 A，我们在图表中看一下。

```
In [32]: gh.plot_estimate_chart_3()
```



现在让我们随机选择一个数字 4/10 来放大我们的估计。我们的估计是测量值的十分之四，其余的来自预测。换句话说，我们在这里表达一种相信，一种相信预测比测量更有可能是正确的。我们可以这样计算：

$$\text{estimate} = \text{prediction} + \frac{4}{10}(\text{measurement} - \text{prediction})$$

测量值和预测值之间的差值称为残差，在上图中用黑色竖线表示。这将成为以后使用的一个重要值，因为它是测量值和滤波器输出之间的差值的精确计算。更小的残差意味着更好的性能。

让我们编写代码，并在与上面的权重序列进行测试时查看结果。我们必须考虑到另一个因素。体重增加的单位是磅/时间，所以通常我们需要添加一个时间步。

我手工生成的体重数据对应于真正的初始体重 160 磅，每天增加 1 磅。换句话说，第一天(第 0 天)的真实体重是 160 磅，第二天(第一天，称重的第一天)的真实体重是 161 磅，以

此类推。

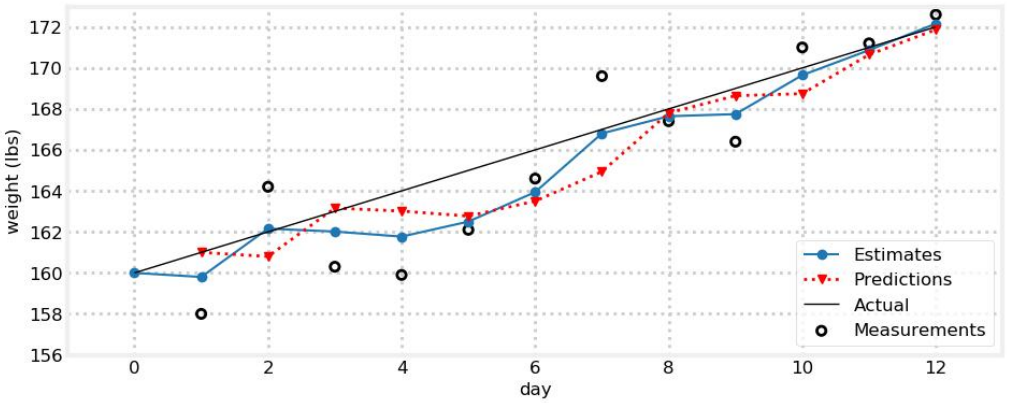
我们需要对初始权重进行猜测。现在谈论初始化策略还为时过早，所以现在我将假设 160 磅。

In [35]:
```
from kf_book.book_plots import figsize
import matplotlib.pyplot as plt

weights = [158.0, 164.2, 160.3, 159.9, 162.1, 164.6,
           169.6, 167.4, 166.4, 171.0, 171.2, 172.6]

time_step = 1.0  # day
scale_factor = 4.0/10

def predict_using_gain_guess(estimated_weight, gain_rate, do_print=False):
    # storage for the filtered results
    estimates, predictions = [estimated_weight], []

    # most filter literature uses 'z' for measurements
    for z in weights:
        # predict new position
        predicted_weight = estimated_weight + gain_rate * time_step

        # update filter
        estimated_weight = predicted_weight + scale_factor * (z - predicted_weight

        # save and log
        estimates.append(estimated_weight)
        predictions.append(predicted_weight)
        if do_print:
            gh.print_results(estimates, predicted_weight, estimated_weight)

    return estimates, predictions

initial_estimate = 160.
estimates, predictions = predict_using_gain_guess(
    estimated_weight=initial_estimate, gain_rate=1, do_print=True)
```

```
previous estimate: 160.00, prediction: 161.00, estimate 159.80
previous estimate: 159.80, prediction: 160.80, estimate 162.16
previous estimate: 162.16, prediction: 163.16, estimate 162.02
previous estimate: 162.02, prediction: 163.02, estimate 161.77
previous estimate: 161.77, prediction: 162.77, estimate 162.50
previous estimate: 162.50, prediction: 163.50, estimate 163.94
previous estimate: 163.94, prediction: 164.94, estimate 166.80
previous estimate: 166.80, prediction: 167.80, estimate 167.64
previous estimate: 167.64, prediction: 168.64, estimate 167.75
previous estimate: 167.75, prediction: 168.75, estimate 169.65
previous estimate: 169.65, prediction: 170.65, estimate 170.87
previous estimate: 170.87, prediction: 171.87, estimate 172.16
```

```
In [36]: # plot results
book_plots.set_figsize(10)
gh.plot_gh_results(weights, estimates, predictions, [160, 172])
weights
```

```
Out[36]: [158.0,
 164.2,
 160.3,
 159.9,
 162.1,
 164.6,
 169.6,
 167.4,
 166.4,
 171.0,
 171.2,
 172.6]
```



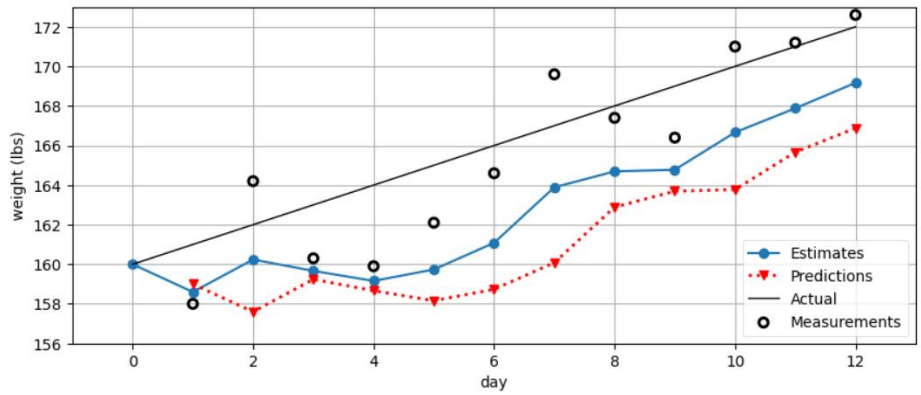
太好了!这里有很多数据，所以我们来谈谈如何解释它。粗蓝线显示的是滤波器的估计结果。从第 0 天开始，最初的猜测是 160 磅。红线显示的是根据前一天的体重做出的预测。第一天之前的体重是 160 磅，增加了 1 磅，所以第一个预测是 161 磅。第一天的估计是在预测和测量之间的一部分，159.8 磅。下面的图表是打印出以前的重量，预测重量，和每天的新估计。最后，这条细黑线显示了被称重者的实际体重增加情况。

每一天都进行一遍，确保您理解每一步的预测和评估是如何形成的。注意估计总是落在测量和预测之间。

估计结果并不是一条直线，但它们比测量结果要直，并且在某种程度上接近我们创建的趋势线。而且，随着时间的推移，它似乎会变得更好。

滤波器的结果可能会让你觉得很愚蠢;当然，如果我们假设我们的体重增加约 1 磅/天的结论，数据会看起来很好!让我们看看如果我们最初的猜测是错误的，滤波器会怎么做。我们来预测一下，每天会减少 1 磅的体重:

```
In [18]: e, p = predict_using_gain_guess(initial_estimate, -1.)
gh.plot_gh_results(weights, e, p, [160, 172])
```



这并不令人印象深刻。估计很快就偏离了测量结果。显然，要求我们正确猜测变化率的滤波器不是很有用。即使我们最初的猜测是正确的，只要变化的速度改变，滤波器就会失败。

如果我停止暴饮暴食，滤波器将很难适应这种变化。注意，它正在调整!尽管我们告诉它我们每天减重 1 磅，但估计数字还在攀升。它调整得不够快。

但是,“如果”?如果我们根据现有的测量和估计来计算体重增加，而不是最初猜测的 1 磅(或其他什么)，会怎么样呢?第一天，我们对权重的估计是:

$$(160 + 1) + \frac{4}{10}(158 - 161) = 159.8$$

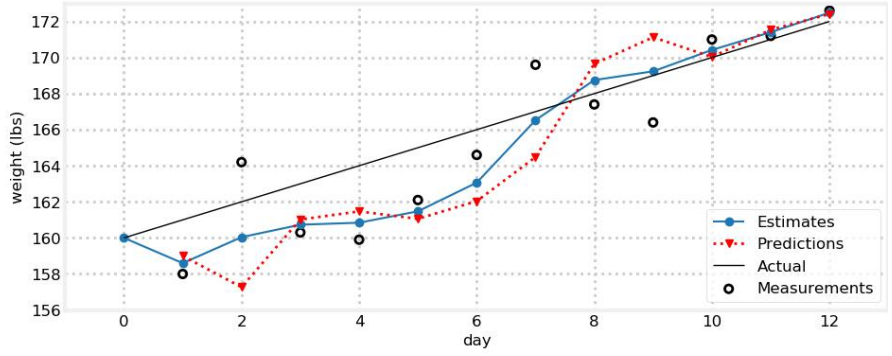
第二天我们的体重是 164.2，这意味着体重增加了 4.4 磅(从 164.2 - 159.8 = 4.4)，而不是 1 磅。我们能利用这些信息吗?这似乎是可信的。毕竟，体重测量本身是基于现实世界对我们体重的测量，所以有有用的信息。我们对体重增加的估计可能并不完美，但这肯定比仅仅猜测体重增加了 1 磅要好。数据比猜测要好，即使它有噪声。

人们在这一点上真的很犹豫，所以确保你们是一致的。两个嘈杂的体重测量给我们一个隐含的体重增加/减少。如果测量不准确，这个估计将是非常不准确的，但是计算中仍然有信息。想象一下，用精确到 1 磅的秤给一头牛称重，结果显示这头牛增重了 10 磅。这头牛可能增重了 8 磅到 12 磅，这取决于误差，但我们知道它增重了，大概增重了多少。这是信息。我们怎么处理信息?永远不要扔掉它!

回到我的饮食。我们应该把新增体重设定为每天 4.4 磅吗?昨天我们认为体重增加了 1 磅，今天我们认为是 4.4 磅。我们有两个数，想把它们结合起来。听起来我们又遇到了同样的问题。我们用同样的工具，目前唯一的工具，在两者之间选择一个值。这次我将使用另一个任意选择的数字，1/3。这个方程与体重估计是一样的，除了我们需要加入时间，因为这是一个速率(增重/天):

$$\text{new gain} = \text{old gain} + \frac{1}{3}\frac{\text{measurement - predicted weight}}{\text{1 day}}$$

In [37]:
```
weight = 160.  # initial guess
gain_rate = -1.0  # initial guess

time_step = 1.
weight_scale = 4./10
gain_scale = 1./3
estimates = [weight]
predictions = []

for z in weights:
    # prediction step
    weight = weight + gain_rate*time_step
    gain_rate = gain_rate
    predictions.append(weight)

    # update step
    residual = z - weight

    gain_rate = gain_rate + gain_scale   * (residual/time_step)
    weight    = weight    + weight_scale * residual

    estimates.append(weight)

gh.plot_gh_results(weights, estimates, predictions, [160, 172])
```



我觉得情况开始好转了。由于最初对体重增加的猜测是-1，所以滤波器需要几天时间来准确地预测体重，但一旦做到了这一点，它就开始准确地跟踪体重。我们没有使用任何方法来选择 4/10 和 1/3 的比例因子(实际上，它们对于这个问题来说是糟糕的选择)，但除此之外，所有的数学计算都是基于非常合理的假设。回想一下，如果您想查看逐步绘制的图形，可以将参数 time_step 的值更改为更大的值并重新运行单元格。

在我们继续之前，还有最后一点。在预测步骤中，我写了这一行：

gain_rate = gain_rate

这显然没有影响，可以去除。我写这篇文章是为了强调，在预测步骤中，您需要预测所有变量的下一个值，包括权重和增益率。这将很快变得相关。在这种情况下，我们假设增益是不变的，但当我们推广这个算法时，我们将去掉这个假设。

The g-h Filter

这个算法被称为 g-h 滤波器或 α - β 滤波器。g 和 h 参考我们在示例中使用的两个比例因子。g 是我们用于测量的比例(在我们的例子中是权重)，h 是测量随时间变化的比例(在我们的例子中是磅/天)。 α - β 只是这些因子的不同名称。

这个滤波器是包括卡尔曼滤波器在内的大量滤波器的基础。换句话说，卡尔曼滤波器是 g-h 滤波器的一种形式，我将在本书后面证明它。还有最小二乘滤波器，你可能听说过，还有 Benedict-Bordner 滤波器，你可能没听说过。每个滤波器都有不同的方法给 g 和 h 赋值，但在其他方面算法是相同的。例如，Benedict-Bordner 滤波器将一个常量赋值给 g 和 h 并限制在一定的值范围内。其他滤波器，如卡尔曼将在每个时间步动态变化 g 和 h。

让我重复一下重点，因为它们很重要。如果你不理解这些，你就无法理解本书的其余部分。如果你确实理解了它们，那么本书的其余部分将自然地为你展开，就像我们将问 g 和 h 的各种“如果”问题的数学阐述一样。数学可能看起来有很大不同，但算法将是完全相同的。

多个数据点比一个数据点更准确，所以无论多么不准确，都不要扔掉任何东西。

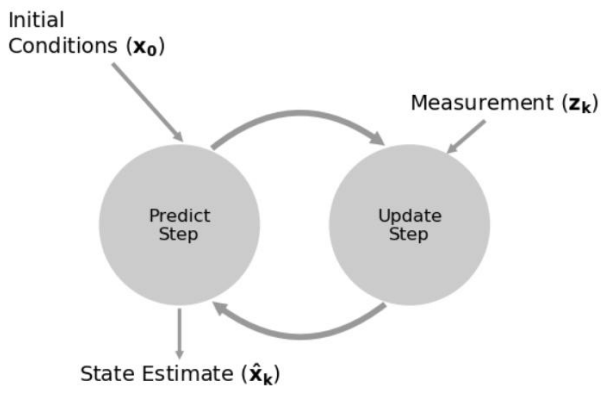
总是在两个数据点之间选择一个数字，这样可以得到更准确的估计。

根据当前的估计以及我们认为它会发生多大的变化，预测下一个测量和变化的速度。

然后选择新的估计作为预测和下一个测量之间的一部分，根据每个测量的准确度进行衡量。

让我们看一下算法的可视化描述。

```
In [42]: book_plots.predict_update_chart()
```



让我介绍一些更正式的术语。系统就是我们要估计的对象。在本章中，系统是我们试图衡量的任何东西。一些文本称其为植物。这个术语来自于控制系统理论。https://en.wikipedia.org/wiki/Plant_ (control_theory)。

系统的状态是我们感兴趣的系统的当前配置或值。我们只对重量读数感兴趣。如果我把一个 100kg 的物体放在磅秤上，状态是 100kg。我们根据与我们相关的东西来定义状态。刻度的颜色与我们无关，所以我们在状态中不包括这些值。制造商的 QA 工程师可能会在状态中包含颜色，这样她就可以跟踪和控制制造过程。

测量值是系统的测量值。测量可能不准确，因此它可能与状态的值不相同。

状态估计是我们的滤波器对状态的估计。例如，对于 100 公斤的重量，由于传感器误差，我们的估计可能是 99.327 公斤。这通常被缩写为估计值，我在本章中已经做过了。

换句话说，状态应该被理解为系统的实际值。这个值通常对我们是隐藏的。如果我站在一个秤上，你就有一个测量。我们称它为可观察的，因为你可以直接观察到这个测量结果。相反，你永远不能直接观察到我的体重，你只能测量它。

这种隐藏和可观察的语言很重要。任何估计问题都包括通过可观测测量形成对隐藏状态的估计。如果您阅读了相关文献，这些术语在定义问题时被使用，那么您需要熟悉它们。

我们使用过程模型对系统进行数学建模。在本章中，我们的过程模型是假设我今天的体重等于昨天的体重加上最后一天增加的体重。流程模型不建模或以其他方式解释传感器。另一个例子是汽车的流程模型。过程模型可能是“距离等于速度乘以时间”。这个模型并不完美，因为汽车的速度会在非零的时间内变化，轮胎会在路面上打滑，等等。系统错误或过程错误是这个模型中的错误。我们永远不知道。

预测步骤被称为系统传播。它使用过程模型来形成一个新的状态估计。由于过程误差，这个估计是不完善的。假设我们在一段时间内跟踪数据，我们说我们将状态传播到未来。一些文献称之为进化。

更新步骤称为测量更新。系统传播和测量更新的一次迭代被称为 epoch。

现在，让我们探索几个不同的问题领域，以更好地理解这个算法。考虑在轨道上跟踪火车的问题。轨道将列车的位置限制在一个非常特定的区域。此外，火车又大又慢。他们需要花很多分钟的时间来减速或大幅加速。所以，如果我知道火车在时间 t 时在里程标记 23 公里处，以 18 公里每小时的速度行驶，我就可以非常自信地预测它在时间 t + 1 秒的位置。为什么这很重要?假设我们只能测量它的位置，精度为±250 米。火车的速度是 18 公里每小时，也就是 5 米每秒。在 t+1 秒时，火车将到达 23.005 公里，但测量结果可能在 22.755 公里至 23.255 公里之间。所以如果下一次测量结果是 23.4 我们知道这肯定是不准确的。即使当时工程师猛踩刹车，火车仍将非常接近 23.005 公里，因为火车不可能在 1 秒内大幅减速。如果我们要为这个问题设计一个滤波器(我们将在本章中进一步讨论!)，我们会希望设计一个对预测和测量给予很高权重的滤波器。

现在考虑追踪一个扔出去的球的问题。我们知道，在引力场中，一个弹道物体在真空中沿抛物线运动。但扔到地球上的球会受到空气阻力的影响，所以它不会沿完美的抛物线运动。棒球投手在投曲线球时就利用了这一点。假设我们正在使用计算机视觉跟踪球场内的球，这是我工作时做的事情。计算机视觉跟踪的准确性可能不高，但通过假设球沿着抛物线运动来预测它未来的位置也不是非常准确。在这种情况下，我们可能会设计一个滤波器，让测量和预测的权重大致相等。

现在考虑在飓风中追踪氦气派对气球。我们没有一个合法的模型可以让我们预测气球的行为，除非是在非常短的时间尺度上(例如，我们知道气球不能在 1 秒内走 10 英里)。在这种情况下，我们将设计一个滤波器，强调测量而不是预测。

这本书的大部分内容都是用数学的方法来表达最后三段中的问题，这样我们就可以找到

一个最优的解决方案(在某种数学意义上)。在本章中，我们将以一种更直观的、因而不是最优的方式为 g 和 h 赋值。但其基本思想是将有些不准确的测量与有些不准确的系统行为模型结合起来，从而得到一个过滤后的估计，这个估计比任何一个信息源本身都要好。

我们可以用算法来表示:

Initialization

1. 初始化滤波器的状态
2. 初始化我们对状态的信任权重

Predict

1. 利用系统行为来预测下一个时间步骤的状态
2. 调整信任权重以解释预测中的不确定性

Update

1. 得到一个关于其准确性的测量和相关的信任权重
2. 计算估计状态和测量值之间的残差
3. 新的估计是在残差线上的某个地方

Notation

测量值：z
下标 k 表示时间步长
粗体表示向量和矩阵