```
! pip install filterpy
```

```
Requirement already satisfied: filterpy in
/Users/sergeidolin/anaconda3/envs/DS/lib/python3.9/site-packages
(1.4.5)
Requirement already satisfied: numpy in
/Users/sergeidolin/anaconda3/envs/DS/lib/python3.9/site-packages (from
filterpy) (1.26.4)
Requirement already satisfied: scipy in
/Users/sergeidolin/anaconda3/envs/DS/lib/python3.9/site-packages (from
filterpy) (1.11.4)
Requirement already satisfied: matplotlib in
/Users/sergeidolin/anaconda3/envs/DS/lib/python3.9/site-packages (from
filterpy) (3.8.4)
Requirement already satisfied: contourpy>=1.0.1 in
/Users/sergeidolin/anaconda3/envs/DS/lib/python3.9/site-packages (from
matplotlib->filterpy) (1.2.0)
Requirement already satisfied: cycler>=0.10 in
/Users/sergeidolin/anaconda3/envs/DS/lib/python3.9/site-packages (from
matplotlib->filterpy) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/Users/sergeidolin/anaconda3/envs/DS/lib/python3.9/site-packages (from
matplotlib->filterpy) (4.25.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/Users/sergeidolin/anaconda3/envs/DS/lib/python3.9/site-packages (from
matplotlib->filterpy) (1.4.4)
Requirement already satisfied: packaging>=20.0 in
/Users/sergeidolin/anaconda3/envs/DS/lib/python3.9/site-packages (from
matplotlib->filterpy) (24.1)
Requirement already satisfied: pillow>=8 in
/Users/sergeidolin/anaconda3/envs/DS/lib/python3.9/site-packages (from
matplotlib->filterpy) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/Users/sergeidolin/anaconda3/envs/DS/lib/python3.9/site-packages (from
matplotlib->filterpy) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in
/Users/sergeidolin/anaconda3/envs/DS/lib/python3.9/site-packages (from
matplotlib->filterpy) (2.9.0.post0)
Requirement already satisfied: importlib-resources>=3.2.0 in
/Users/sergeidolin/anaconda3/envs/DS/lib/python3.9/site-packages (from
matplotlib->filterpy) (6.1.1)
Requirement already satisfied: zipp>=3.1.0 in
/Users/sergeidolin/anaconda3/envs/DS/lib/python3.9/site-packages (from
importlib-resources>=3.2.0->matplotlib->filterpy) (3.19.2)
Requirement already satisfied: six>=1.5 in
/Users/sergeidolin/anaconda3/envs/DS/lib/python3.9/site-packages (from
python-dateutil>=2.7->matplotlib->filterpy) (1.16.0)
```

```
#format the book
import book_format
```

```
book_format.set_style()

from kf_book.book_plots import figsize
import numpy as np
from matplotlib.patches import Circle, Rectangle, Polygon, Arrow,
FancyArrow
import matplotlib.pylab as plt
```

Фундаментальное уравнение космической геодезии

Определение пространственных координат объекта на Земле возможно при помощи глобальных навигационных спутниковых систем, таких как ГЛОНАСС или GPS (GPS, ГЛОНАСС, Galileo, BeiDou). Множество различных типов устройств содержат ГНСС микросхему, многие мобильные устройства умеют обрабатывать не только кодовые сигналы, но и фазу несущей частоты, что позволяет реализовывать высокоточные методоы спутникого позиционирования.

Данный курс не посвящен теме о том, как именно и что именно передают спутники, а тому, как алгоритмы в приемниках обрабатывают эту информацию. Давайте взглянем на фундаментальное уравнение космической геодезии, которое лежит в основе спутникового позиционирования.

$$\vec{p} = \vec{R} - \vec{r}$$

где $\vec{p}$ - радиус-вектор спутника в геоцентрической системе координат;

$\vec{R}$ - радиус-вектор пункта поверхности в геоцентрической системе координат;

$\vec{r}$ - радиус-вектор спутника в топоцентрической системе координат.

Фундаментальное уравнение космической геодезии представляет собой векторное уравнение, связывающие координаты пункта земной поверхности в общеземной геоцентрической системе координат с координатами спутника в общеземной геоцентрической системе координат и топоцентрической системе координат.

Как вы уже догадались, приемник вычисляет координаты за счет измерения расстояния от него до спутника(ов) выполняя наблюдения.

В этом курсе шаг за шагом, мы будем постигать тайны филтра Калмана, алгоритма, который позволяет из измерений подвежрженных влиянию различных шумов определить искомую случаную величину.

Мысленный эксперимент

При выполнении измерений любым устройством мы вносим ошибку в результат нашей обработки, так как используемые нами устройства или датчики имеют так называемый шум измерений.

Пока отвлечемся от спутниковых измерений и проведем мысленный эксперимент на более простых приборах, которые в каком-то смысле сохраняют общий принцип спутниковых измерений.

Пример N°1:

Представим, что у нас в руках есть два светодальномера и мы хотим получить расстояние между пунктом А и Б. Давайте договоримся о том, что светодальномеры находятся в непосредственной близости друг от друга и выполняют измерения в один момент времени. Эти измерения подвержены влиянию атмосферных эффектов с одинаковой интенсивностью.

- Светодальномер N°1 - 150 м;
- Светодальномер N°2 - 160 км.

Вопрос, какой же вариант нам выбрать? Если вы хоть немного работали с любыми сериями измерений, вам интуитивно захочется взять среднее из этого результата, но при всем при этом никто не осудит вас если вы заходите взять результат первого измерения или второго, а может вы считаете, что оцениваемая величина находится ниже результатов 1 и 2 или выше, но конечно чаще всего ответ лежит где-то посередине.

В математике такой результат называют "Математическим ожиданием", хотя стоит сказать, что этот термин значит несколько больше, чем просто среднее значение.

Давайте зададимся вопросом, а что было бы если бы мы выполнили миллион измерений, какой тогда вариант ответа был бы очевидным для нас?

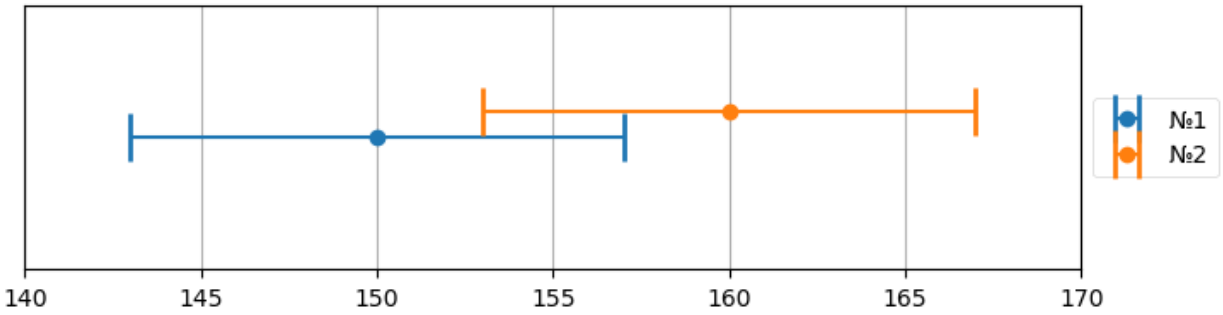
В некоторых случаях показания обоих светодальномеров будут слишком маленькими, иногда - слишком большими, а в остальное время они будут соответствовать близкому значению "истинного" расстояния. Если они соответствуют "истинному" расстоянию, то, конечно, нам следует выбрать число между N°1 и N°2. Если они не совпадают, то мы не знаем, являются ли они оба слишком маленькими или большими, но, выбирая число между N°1 и N°2, мы, по крайней мере, смягчаем эффект наихудшего измерения. Например, предположим, что фактическое расстояние составляет 170 меторо. 150 метров - это большая ошибка. Но если мы выберем значения от 150 до 160 метров, то наша оценка будет лучше, чем 150 метров. Тот же аргумент справедлив, если оба светодальномера вернули значение, превышающее "истинное" расстояние.

Позже мы разберемся с этим более формально, но сейчас, я надеюсь, ясно, что наша наилучшая оценка - это среднее значение N°1 и N°2.

$$\frac{150+160}{2}=155$$

Мы можем посмотреть на это графически. Графики измерений N°1 и N°2 с предполагаемой погрешностью в 7 метров. Результаты измерений находятся в диапазоне от 150 до 160 метров, поэтому единственное расстояние, которое имеет смысл, должно находиться в пределах от 150 до 160 метров.

```
import kf_book.book_plots as book_plots
from kf_book.book_plots import plot_errorbars
plot_errorbars([(150, 7, '№1'), (160, 7, '№2')], xlims=(140, 170))
```

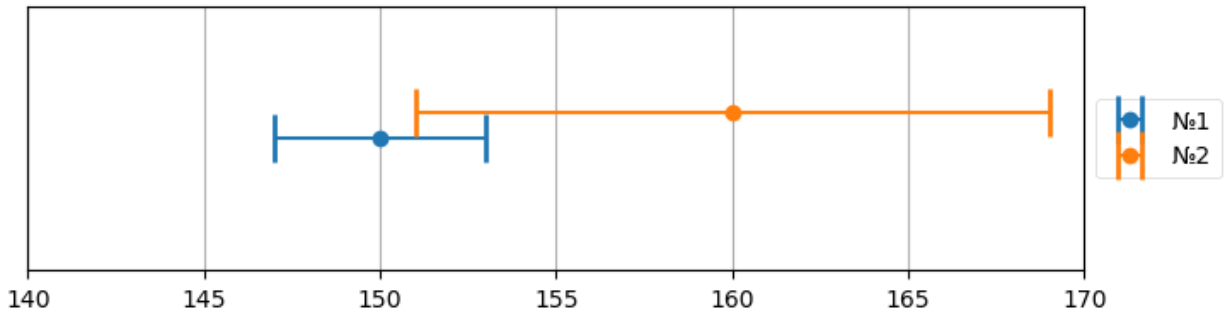


Таким образом, 155 метров выглядит как разумная оценка расстояния, но здесь есть дополнительная информация, которой мы могли бы воспользоваться. Единственные возможные значения расстояния находятся на пересечении столбцов погрешности N°1 и N°2. Например, расстояния в 151 метр невозможно, поскольку светодальномер N°2 не мог показать значение в 160 метров при максимальной погрешности в 7 метров. Аналогично, расстояние в 159 метров невозможно, поскольку светодальномер N°1 не мог показать значение в 150 фунтов с максимальной погрешностью в 7 метров. В этом примере единственно возможные значения расстояния находятся в диапазоне от 153 до 157 метров.

Это пока не позволяет нам получить более точную оценку расстояния, но давайте еще немного поиграем в "что, если". Что, если нам сейчас скажут, что N°1 в три раза точнее, чем N°2? Рассмотрим 5 вариантов, которые мы перечислили выше. Все равно нет смысла выбирать число, выходящее за пределы диапазона N°1 и N°2, поэтому мы не будем рассматривать их. Возможно, кажется более убедительным выбрать N°1 в качестве нашей оценки - в конце концов, мы знаем, что это более точное значение, почему бы не использовать его вместо N°2? Может ли N°2 улучшить наши знания по сравнению с N°1 в одиночку?

Ответ, возможно, противоречащий интуиции, таков: да, это возможно. Сначала давайте рассмотрим те же измерения N°1=150 и N°2=160, но с погрешностью N°1 в 3 метра, а погрешность N°2 в 3 раза больше, 9 метров.

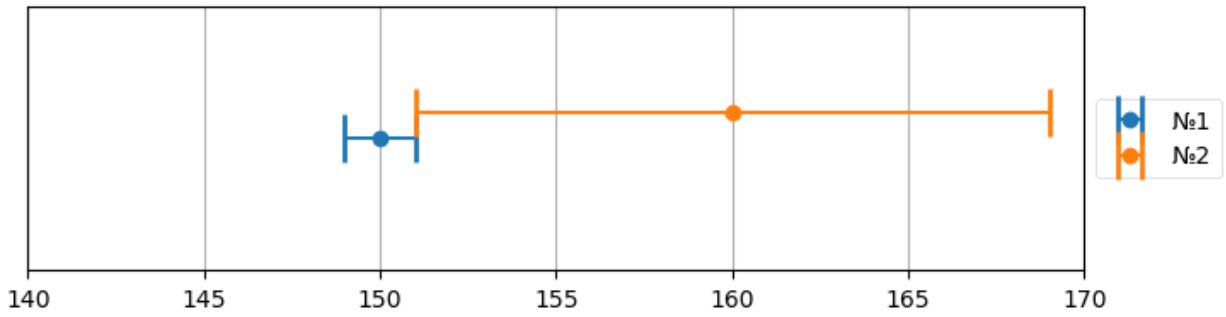
```
plot_errorbars([(150, 3, '№1'), (160, 9, '№2')], xlims=(140, 170))
```



Перекрытие столбцов ошибок N°1 и N°2 - это единственно возможное "истинное" расстояние. Это перекрытие меньше, чем ошибка только в N°1. Что еще более важно, в этом случае мы можем видеть, что перекрытие не включает в себя 150 метров или 155 метров. Если бы мы использовали измерение из пункта N°1 только потому, что оно более точное, чем N°2, мы бы дали оценку в 150 метров. Если мы усредним значения N°1 и N°2, то получим 155 метров. Ни один из этих показателей расстояния невозможен, учитывая наши знания о точности измерений. Включив измерение N°2, мы получим оценку где-то между 151 и 153 метрами, что соответствует пределам пересечения двух погрешностей.

Давайте возьмем это за крайние пределы. Предположим, что мы знаем, что измерение N°1 имеет точность до 1 метра. Другими словами, если расстояние межде А и Б действительно равно 160 метров, то светодальномер может показать 159, 160 или 161 метр. Мы также знаем, что точность измерений составляет 9 метров. Мы измеряем расстояние всеми светодальномерами и получаем N°1=150, а N°2=160. Каким должно быть наше расстояние? Давайте представим это графически.

```
plot_errorbars([(150, 1, '№1'), (160, 9, '№2')], xlims=(140, 170))
```



Видно, что единственно возможное расстояние составляет 161 метр. Это важный результат. Используя два относительно неточных светодальномера, мы можем получить чрезвычайно точный результат.

Таким образом, два светодальномера, даже если один из них менее точен, чем другой, лучше, чем один. Мы никогда не выбрасываем информацию, какой бы скудной она ни была. Мы будем разрабатывать математические методы и алгоритмы, которые позволят нам использовать все возможные источники информации для получения наилучшей оценки.

Однако мы отклонились от нашей задачи. Врядли кто-то из геодезистов захочет покупать два светодальномера и носить их везде с собой, и, кроме того, мы изначально исходили из предположения, что все светодальномеры одинаково точны. Понимание того, как использовать все измерения независимо от их точности, сыграет большую роль в дальнейшем, так что не забывайте об этом.

Что делать, если у нас один светодальномер, но мы выполнели много измерений? Мы пришли к выводу, что если у нас будут два светодальномера одинаковой точности, то мы должны усреднить результаты их измерений. Что, если я измерю расстояние 10 000 раз одним прибором? Мы уже говорили, что светодальномеры с одинаковой вероятностью выдадут как слишком большие, так и слишком маленькие расстояния. Не так уж сложно доказать, что среднее значение большого числа расстояний будет очень близко к фактическому расстоянию, но давайте пока напишем моделирование. Я буду использовать NumPy, часть экосистемы SciPy для численных вычислений.

```
import numpy as np
measurements = np.random.uniform(150, 160, size=10000)
mean = measurements.mean()
print(f'Average of measurements is {mean:.4f}')

Average of measurements is 154.9725
```

Напечатанное число зависит от вашего генератора (псевдо)случайных чисел, но оно должно быть очень близко к 155.

В этом коде делается предположение, которое, вероятно, не соответствует действительности, а именно, что светодальномеры с одинаковой вероятностью покажут как 150, так и 155 при "истинном" расстоянии в 155 метров. Это почти никогда не соответствует действительности. Реальные датчики с большей вероятностью будут получать показания, близкие к "истинному" значению, и с меньшей вероятностью будут получать показания, удаляющиеся от истинного значения. Мы подробно рассмотрим это в главе, посвященной Гауссу. На данный момент я буду использовать без дальнейших объяснений функцию `numpy.random.normal()`, которая будет выдавать больше значений ближе к 155 метрам. Пока что примите на веру, что это приведет к измерениям с шумом, аналогичным тому, как в настоящих светодальномерах.

```
mean = np.random.normal(155, 5, size=10000).mean()
print(f'Average of measurements is {mean:.4f}')

Average of measurements is 155.0071
```

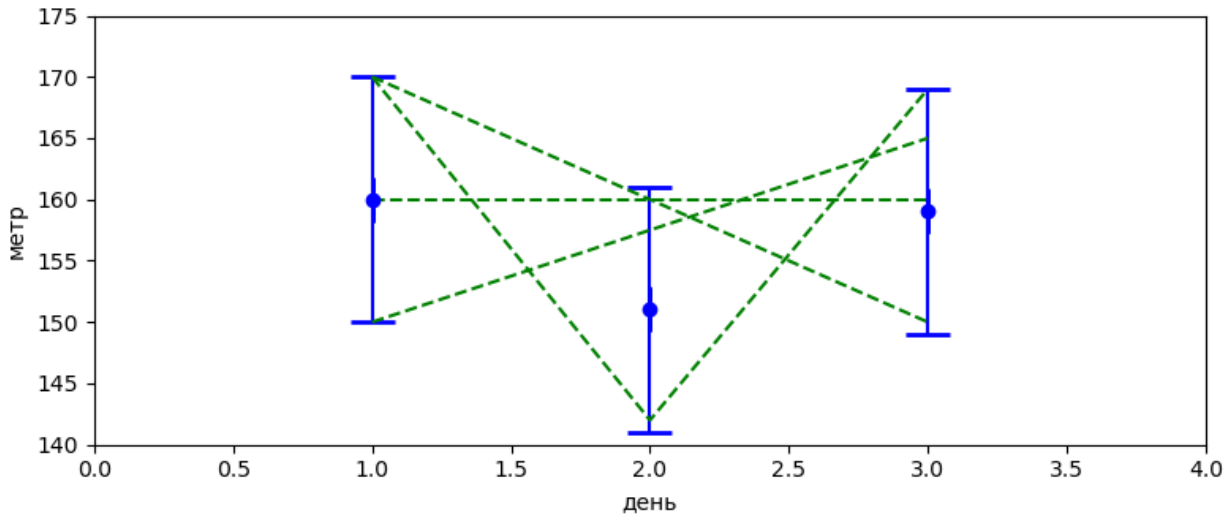
И снова ответ очень близок к 155.

Итак, отлично, у нас есть решение нашей проблемы с светодальномерами! Но это не очень практичный ответ. Ни у кого не хватит терпения измерить расстояние от А до Б десять тысяч или даже сто тысяч раз.

Итак, давайте поиграем в "что, если бы". Что, если бы вы измеряли расстояние один раз в день и получали показания 160, 151, а затем 159? Марка убегает, уворачивается или все это просто шумные измерения?

Мы точно не можем сказать. Первое измерение было 160, а последнее - 159, что означает перемещение марки на 1 метр. Но если точность светодальномера составляет 10 метров,то это можно объяснить шумом. Возможно, марка на самом деле переместилась; возможно, в первый день расстояние составляло 155 метров, а на третий - 162. Эти расстояния можно получить при таком перемещении. Мой светодальномер показывает, что марка отодвинулась на метр,а на самом деле марка пододвинулась к нам! Давайте посмотрим на это в виде диаграммы. Я нанес на график результаты измерений вместе с графиками погрешностей, а затем некоторые возможные приближения / отдаления марки, которые можно было бы объяснить этими измерениями, обозначенными пунктирными зелеными линиями.

```
with figsize(y=3.5):
        plt.figure()
        plt.errorbar([1, 2, 3], [160, 151, 159],
                xerr=0, yerr=10, fmt='bo', capthick=2, capsize=10)

        plt.plot([1, 3], [170, 150], color='g', ls='--')
        plt.plot([1, 3], [160, 160], color='g', ls='--')
        plt.plot([1, 3], [150, 165], color='g', ls='--')
        plt.plot([1, 2, 3], [170, 142, 169], color='g', ls='--')
        plt.xlim(0,4)
        plt.ylim(140, 175)
        plt.xlabel('день')
        plt.ylabel('метр')
        plt.grid(False)
        plt.tight_layout()
```

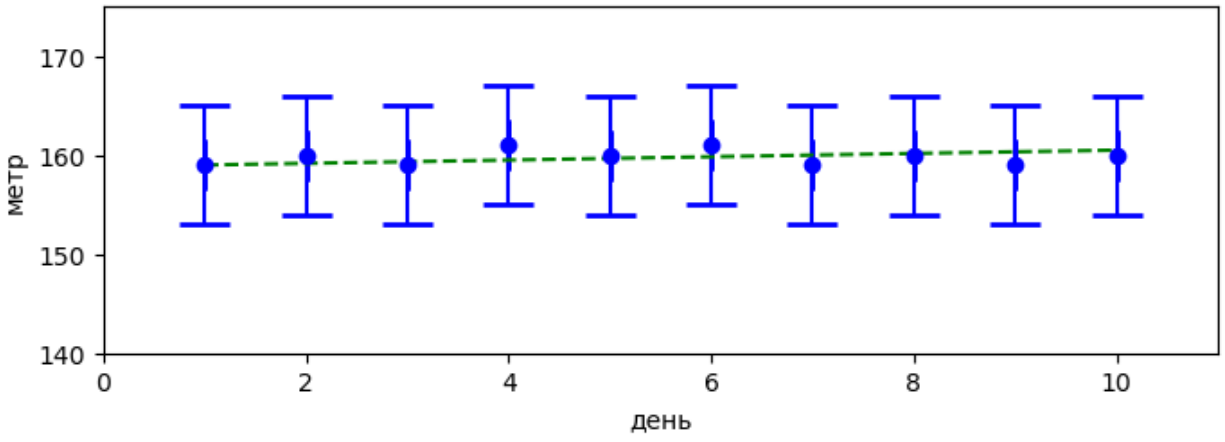


Как мы видим, существует огромный диапазон изменений расстояний, который можно объяснить с помощью этих трех измерений. На самом деле, существует бесконечное количество вариантов. Стоит ли сдаваться? Нет! Напомним, что речь идет об измерении расстояния между пунктами А и Б. Конечно можно придумать объяснение почему точка Б переместилась в один день на несколько метров, но врядли такое возможно.

Поведение физической системы, которую мы измеряем, должно влиять на то, как мы интерпретируем результаты измерений. Если бы мы каждый день измеряли марку на доме, мы бы списали все на шум измерений. Если бы мы измеряли расстояние от спутника до приемника, то могли бы сказать что такие измерения вполне реальны, так как расспространяемый сигнал подвергается большому количеству внешних источников шума.

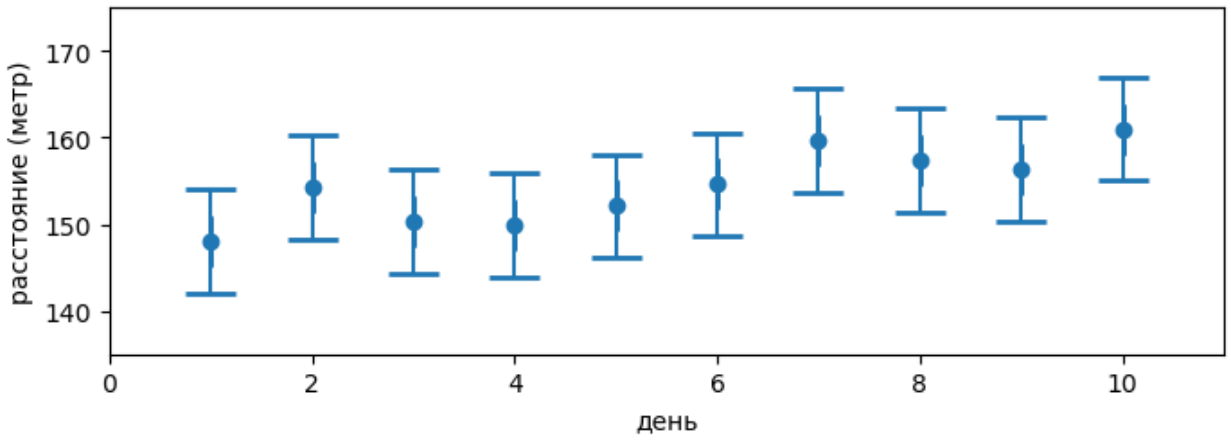
Предположим, я использую другой светодальномер и получаю следующие результаты измерений: 159, 160, 159, 161, 160, 161, 159, 160, 159, 160. Что подсказывает вам ваша интуиция? Например, возможно, что пункт Б перемещается по 1 метру в день, а результаты измерений с помехами выглядят так, будто расстояние осталось прежним. В равной степени пункт Б мог перемещаться по 1 метру в день. Но возможно ли это? Какова вероятность того, что, подбросив монетку, я получу 10 выпадений орлом подряд? Не очень вероятно. Мы не можем доказать это, основываясь только на этих показаниях, но вполне вероятно, что расстояние до пункта Б оставалось неизменным. На приведенной ниже диаграмме я изобразил результаты измерений столбиками с ошибками и вероятным "истинным" расстоянием, обозначенным зеленым пунктиром. Эта пунктирная линия не является "правильным" ответом на этот вопрос, а просто разумным ответом, который может быть объяснен результатами измерений.

```
with book_plots.figsize(y=2.5):
    plt.figure()
    plt.errorbar(range(1, 11), [159, 160, 159, 161, 160, 161, 159,
160, 159, 160],
                 xerr=0, yerr=6, fmt='bo', capthick=2, capsize=10)
    plt.plot([1, 10], [159, 160.5], color='g', ls='--')

    plt.xlim(0, 11)
    plt.ylim(140, 175)
    plt.xlabel('день')
    plt.ylabel('метр')
    plt.grid(False)
```



Другой вопрос, что, если бы: что, если бы показания были такими 148.0, 154.2, 150.3, 149.9, 152.1, 154.6, 159.6, 157.4, 156.4, 161.0? Давайте посмотрим на график, а затем ответим на несколько вопросов.

```
measurements = [148.0, 154.2, 150.3, 149.9, 152.1, 154.6, 159.6,
157.4, 156.4, 161.0]

with book_plots.figsize(y=2.5):
    plt.figure()

    plt.errorbar(range(1, 11), measurements,
                 xerr=0, yerr=6, fmt='o', capthick=2, capsize=10)

    plt.xlim(0, 11)
    plt.ylim(135, 175)
    plt.xlabel('день')
    plt.ylabel('расстояние (метр)')
    plt.grid(False)
```

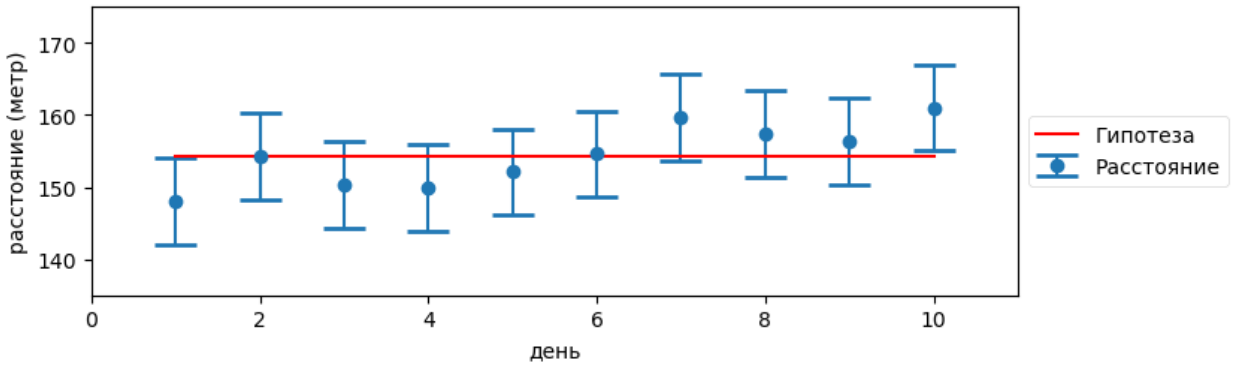


Кажется ли "вероятным", что пункт Б сдвинулся, и это просто очень зашумленные данные? Не совсем. Кажется ли вам вероятным, что пункт Б стоит на месте? И снова нет. Со временем измеренное расстояние увеличивается; не равномерно, но определенно увеличивается. Мы не можем быть уверены, но это похоже на перемещение пункта Б, причем значительное. Давайте проверим это предположение с помощью еще нескольких графиков. Часто данные легче увидеть на графике, чем в таблице.

Итак, давайте рассмотрим две гипотезы. Во-первых, давайте предположим, что положение пункт Б, а следовательно расстояние между А и Б не изменилось. Чтобы получить расстояние, мы договорились, что нам следует усреднить результаты измерений. Давайте посмотрим на это.

```
measurements = [148.0, 154.2, 150.3, 149.9, 152.1, 154.6, 159.6,
157.4, 156.4, 161.0]

with book_plots.figsize(y=2.5):
    plt.figure()
    ave = np.sum(measurements) / len(measurements)
    plt.errorbar(range(1,11), measurements, label='Расстояние',
                 yerr=6, fmt='o', capthick=2, capsize=10)
```

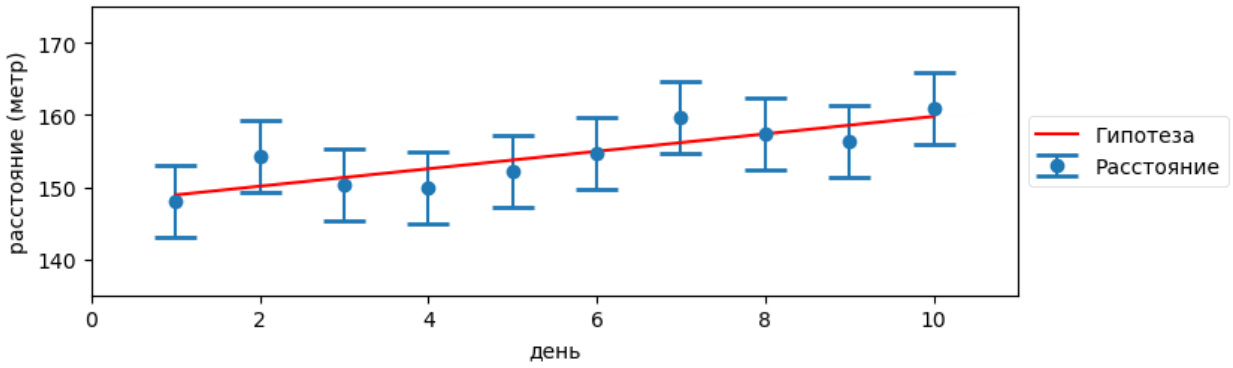
```
    plt.plot([1, 10], [ave,ave], c='r', label='Гипотеза')
    plt.xlim(0, 11)
    plt.ylim(135, 175)
    plt.xlabel('день')
    plt.ylabel('расстояние (метр)')
    book_plots.show_legend()
    plt.grid(False)
```



Не очень убедительно. На самом деле, мы видим, что внутри не всех столбцов ошибок проходит горизонтальная линия, которую мы могли бы провести.

Теперь давайте предположим, что расстояние изменилось. На сколько? Я не знаю, но NumPy знает! Мы хотим провести линию между измерениями, которая выглядит "примерно" правильно. В NumPy есть функции, которые будут делать это в соответствии с правилом, называемым "соответствие наименьшим квадратам". Давайте не будем беспокоиться о деталях этого вычисления (я использую polyfit(), если вам интересно), а просто построим график результатов.

```
measurements = [148.0, 154.2, 150.3, 149.9, 152.1, 154.6, 159.6,
157.4, 156.4, 161.0]

xs = range(1, len(measurements)+1)
line = np.poly1d(np.polyfit(xs, measurements, 1))

with figsize(y=2.5):
    plt.figure()
    plt.errorbar(range(1, 11), measurements, label='Расстояние',
                 yerr=5, fmt='o', capthick=2, capsize=10)
    plt.plot (xs, line(xs), c='r', label='Гипотеза')
    plt.xlim(0, 11)
    plt.ylim(135, 175)
    plt.xlabel('день')
    plt.ylabel('расстояние (метр)')
    book_plots.show_legend()
    plt.grid(False)
```



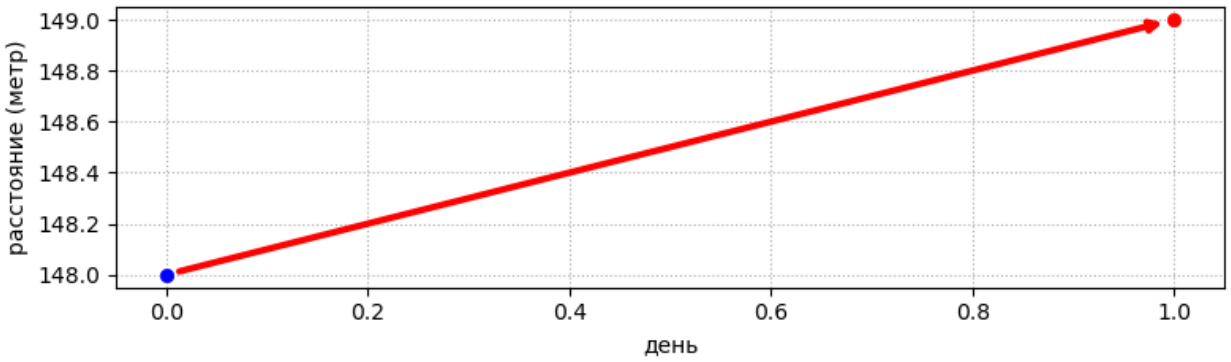
Намного лучше, по крайней мере, на мой взгляд. Обратите внимание, что теперь гипотеза очень близка к каждому измерению, тогда как на предыдущем графике гипотеза часто была довольно далека от результатов измерений. Кажется, что гораздо более вероятно, что расстоние увеличивется, чем остается неизменным. Действительно ли пункт Б переместился 13 метров? Кто может сказать? Кажется, на этот вопрос невозможно ответить.

"Но разве это невозможно?"

Давайте попробуем что-нибудь безумное. Давайте предположим, что я знаю, что пункт Б перемещается на метр в день. Неважно, откуда я это знаю, предположим, что это приблизительно верное утверждение. Это мысленный эксперимент, детали не важны. Давайте посмотрим, сможем ли мы воспользоваться такой информацией, если бы она была доступна.

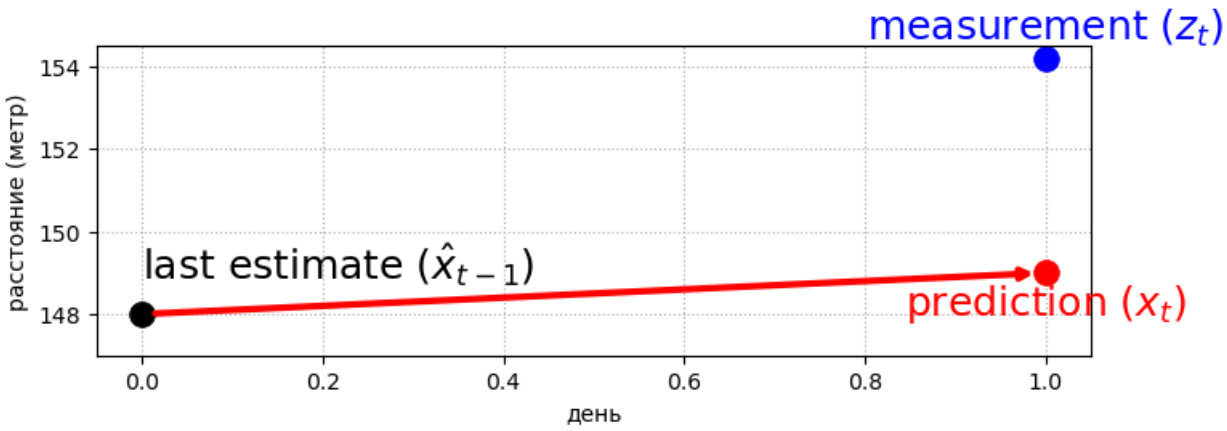
Первое измерение было 148. У нас нет возможности узнать разницу, поэтому давайте примем это за нашу оценку. Если расстояние от А до Б сегодня равно 148, каким он будет завтра? Что ж, мы думаем, что набираем расстояние изменится со скоростью 1 метр в день, поэтому наш прогноз - 149, примерно так:

```
with figsize(y=2.5):
    plt.figure()
    ax = plt.axes()
    ax.annotate('', xy=[1,149], xytext=[0,148],
                arrowprops=dict(arrowstyle='->', ec='r',shrinkA=6,
lw=3,shrinkB=5))
    plt.scatter ([0], [148], c='b')
    plt.scatter ([1], [149], c='r')
    plt.xlabel('день')
    plt.ylabel('расстояние (метр)')
    ax.xaxis.grid(True, which="major", linestyle='dotted')
    ax.yaxis.grid(True, which="major", linestyle='dotted')
    plt.tight_layout()
```



Хорошо, но что это нам дало? Конечно, мы могли бы считать, что 1 метр в день - это точно, и предсказать расстояние на следующие 10 дней, но тогда зачем вообще использовать светодальномер, если мы не учитываем его показания? Итак, давайте рассмотрим следующее измерение. Мы снова измеряем расстояние, и светодальномер показывает 154,2 метра.

```
with figsize(y=2.5):
    plt.figure()
    ax = plt.axes()
    ax.annotate('', xy=[1,149], xytext=[0,148],
                arrowprops=dict(arrowstyle='->',
                                ec='r', lw=3, shrinkA=6, shrinkB=5))
    plt.scatter ([0], [148.0], c='k',s=128)
    plt.scatter ([1], [154.2], c='b',s=128)
    plt.scatter ([1], [149], c='r', s=128)
    plt.text (1.0, 148.8, "prediction ($x_t)$",
ha='center',va='top',fontsize=18,color='red')
    plt.text (1.0, 154.4, "measurement
($z_t$)",ha='center',va='bottom',fontsize=18,color='blue')
    plt.text (0.0, 149.8, "last estimate ($\hat{x}_{t-1}$)",
ha='left', va='top',fontsize=18)
    plt.xlabel('день')
    plt.ylabel('расстояние (метр)')
    ax.xaxis.grid(True, which="major", linestyle='dotted')
    ax.yaxis.grid(True, which="major", linestyle='dotted')
    plt.ylim(147, 154.5)
```



У нас проблема. Наш прогноз не соответствует результатам измерений. Но это именно то, чего мы ожидали, верно? Если бы прогноз всегда в точности совпадал с результатами измерений, мы бы не смогли добавить какую-либо информацию в фильтр. И, конечно, не было бы смысла проводить какие-либо измерения, поскольку наши прогнозы безупречны.

> ** Ключевая идея всего этого курса содержится в следующем абзаце. Прочтите ее внимательно!**

Итак, что же нам делать? Если мы будем формировать оценки только на основе результатов измерений, то прогноз не повлияет на результат. Если мы будем формировать оценки только на основе прогнозов, то результаты измерений будут проигнорированы. Если мы хотим, чтобы наш анализ данных сработал, нам нужно каким-то образом сочетать прогнозирование и измерение.

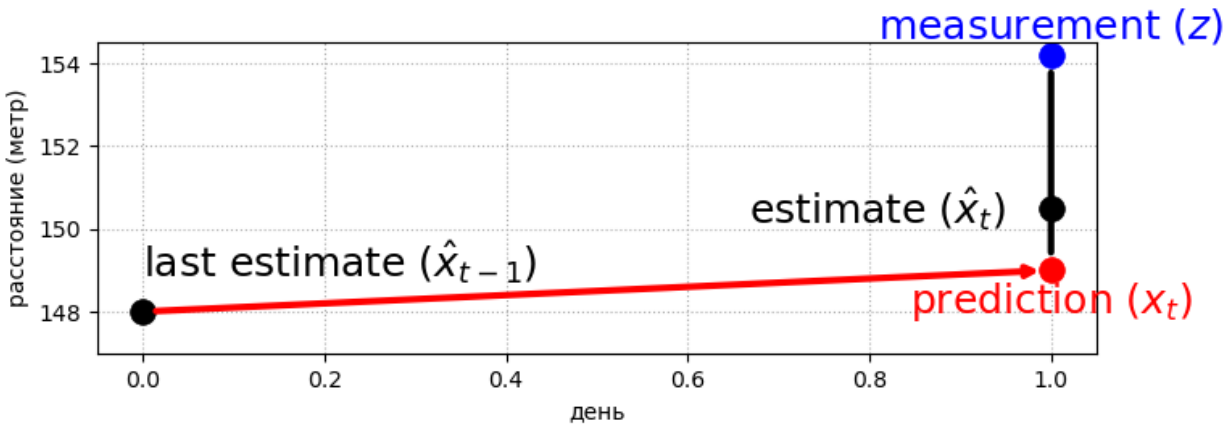
Смешение двух значений - это очень похоже на задачу о двух светодальномерах, рассмотренную ранее. Используя те же рассуждения, что и раньше, мы можем видеть, что единственное, что имеет смысл, - это выбрать число между прогнозом и измерением. Например, оценка в 155 не имеет смысла, как и 147. Наши оценки должны находиться в диапазоне от 149 (прогноз) до 154,2 (измерение).

Еще раз подчеркну, что это очень важно. Мы согласились с тем, что при представлении двух значений с ошибками мы должны получить промежуточную оценку между этими двумя значениями. Не имеет значения, как были получены эти значения. В начале главы у нас было два измерения, но теперь у нас есть одно измерение и один прогноз. Логика и, следовательно, математические расчеты в обоих случаях одинаковы. Мы никогда не выбрасываем информацию на ветер.

Я вынужден настаивать на том, чтобы вы остановились и по-настоящему задумались об этом. Все, что я сделал, - это заменил неточный светодальномер на неточный прогноз расстояния, основанный на модели движения пункта Б. Это все еще данные. Математика не знает, были ли данные получены с помощью светодальномера или прогноза. У нас есть два фрагмента данных с определенным количеством шума, и мы хотим их объединить. В оставшейся части этой главы мы рассмотрим некоторые довольно сложные математические методы для выполнения этих вычислений, но математику никогда не волнует, откуда берутся данные, она производит вычисления, основанные только на значении и точности этих значений.

Должна ли оценка находиться на полпути между измерением и прогнозом? Возможно, но в целом кажется, что мы можем знать, что наш прогноз более или менее точен по сравнению с измерениями. Вероятно, точность нашего прогноза отличается от точности светодальномера. Вспомните, что мы сделали, когда светодальномер N°1 оказалась намного точнее, чем N°2 - мы масштабировали ответ так, чтобы он был ближе к N°1, чем к N°2. Давайте посмотрим на это на диаграмме.

```
with figsize(y=2.5):
    plt.figure()
    ax = plt.axes()
    ax.annotate('', xy=[1,149], xytext=[0,148],
                arrowprops=dict(arrowstyle='->',
                                ec='r', lw=3, shrinkA=6, shrinkB=5))

    ax.annotate('', xy=[1,149], xytext=[1,154.2],
                arrowprops=dict(arrowstyle='-',
                                ec='k', lw=3, shrinkA=8, shrinkB=8))

    est_y = (148 + .4*(154.2-148))
    plt.scatter ([0,1], [148.0,est_y], c='k',s=128)
    plt.scatter ([1], [154.2], c='b',s=128)
    plt.scatter ([1], [149], c='r', s=128)
    plt.text (1.0, 148.8, "prediction ($x_t)$",
ha='center',va='top',fontsize=18,color='red')
    plt.text (1.0, 154.4, "measurement
($z$)",ha='center',va='bottom',fontsize=18,color='blue')
    plt.text (0, 149.8, "last estimate ($\hat{x}_{t-1}$)", ha='left',
va='top',fontsize=18)
    plt.text (0.95, est_y, "estimate ($\hat{x}_{t}$)", ha='right',
va='center',fontsize=18)
    plt.xlabel('день')
    plt.ylabel('расстояние (метр)')
    ax.xaxis.grid(True, which="major", linestyle='dotted')
    ax.yaxis.grid(True, which="major", linestyle='dotted')
    plt.ylim(147, 154.5)
```



Теперь давайте попробуем использовать случайно выбранное число для масштабирования нашей оценки: $\frac{4}{10}$. Наша оценка будет составлять четыре десятых от измерения, а остальное будет получено из прогноза. Другими словами, мы выражаем здесь уверенность в том, что прогнозирование с большей вероятностью окажется правильным, чем результаты измерений. Мы вычисляем это как

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

Разница между измерением и прогнозом называется "остаточной невязкой", которая показана черной вертикальной линией на графике выше. Это значение станет важным для дальнейшего использования, поскольку оно позволяет точно рассчитать разницу между измерениями и выходными данными фильтра. Меньшие значения невязок означают лучшую производительность.

Давайте закодируем это и посмотрим на результаты, когда мы проверим это на основе приведенных выше весовых коэффициентов. Мы должны принять во внимание еще один фактор. Увеличение расстояние измеряется в метрах за время, поэтому для общего понимания нам нужно будет добавить временной шаг t, который мы установим равным 1 (день).

Я вручную сгенерировал данные о расстоянии, чтобы они соответствовали истинному исходному расстоянию в 150 метров и увеличению на 1 метр в день. Другими словами, в первый день (нулевой день) истинное расстояние составляет 150 метров, на второй день (первый день, первый день измерения) истинное расстояние составляет 151 метр и так далее.

Нам нужно сделать предположение о первоначальном расстояние. Пока еще слишком рано говорить о стратегиях инициализации, поэтому пока я буду считать, что расстояни равно 150 метров.

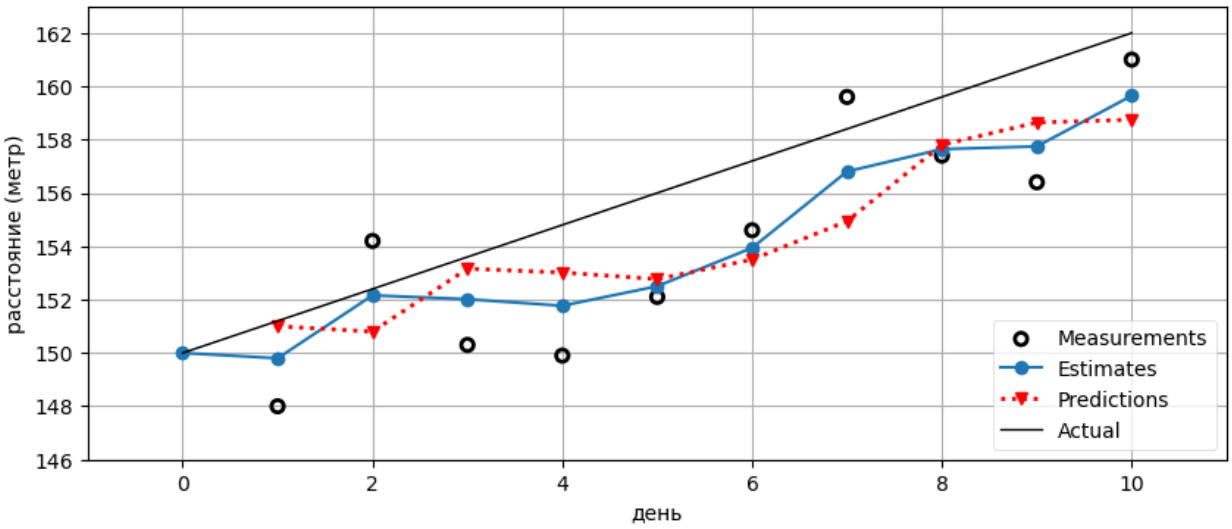
```
def plot_results(weights, estimates, predictions, actual,
time_step=0):
    n = len(weights)
    if time_step > 0:
        rng = range(1, n+1)
    else:
        rng = range(n, n+1)
    xs = range(n+1)
    book_plots.plot_measurements(range(1, len(weights)+1), weights,
color='k', lines=False)
    book_plots.plot_filter(xs, estimates, marker='o',
label='Estimates')
    book_plots.plot_track(xs[1:], predictions, c='r', marker='v',
label='Predictions')
    plt.plot([xs[0], xs[-1]], actual, c='k', lw=1, label='Actual')
    plt.legend(loc=4)
    book_plots.set_labels(x='день', y='расстояние (метр)')
```

```
    plt.xlim([-1, n+1])
    plt.ylim([146.0, 163])

from kf_book.book_plots import figsize
import matplotlib.pyplot as plt

measurements = [148.0, 154.2, 150.3, 149.9, 152.1, 154.6, 159.6,
157.4, 156.4, 161.0]

time_step = 1.0  # day
scale_factor = 4.0/10

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

    # most filter literature uses 'z' for measurements
    for z in measurements:
        # predict new position
        predicted_mes = estimated_mes + gain_rate * time_step

        # update filter
        estimated_mes = predicted_mes + scale_factor * (z -
predicted_mes)

        # save and log
        estimates.append(estimated_mes)
        predictions.append(predicted_mes)
        if do_print:
            print('previous estimate: {:.2f}, prediction: {:.2f},
estimate {:.2f}'.format(
                estimates[-2], predicted_mes, estimated_mes))

    return estimates, predictions

initial_estimate = 150.
estimates, predictions = predict_using_gain_guess(
    estimated_mes=initial_estimate, gain_rate=1, do_print=True)
```

```
previous estimate: 150.00, prediction: 151.00, estimate 149.80
previous estimate: 149.80, prediction: 150.80, estimate 152.16
previous estimate: 152.16, prediction: 153.16, estimate 152.02
previous estimate: 152.02, prediction: 153.02, estimate 151.77
previous estimate: 151.77, prediction: 152.77, estimate 152.50
previous estimate: 152.50, prediction: 153.50, estimate 153.94
previous estimate: 153.94, prediction: 154.94, estimate 156.80
previous estimate: 156.80, prediction: 157.80, estimate 157.64
previous estimate: 157.64, prediction: 158.64, estimate 157.75
previous estimate: 157.75, prediction: 158.75, estimate 159.65
```

```
# plot results
book_plots.set_figsize(10)
plot_results(measurements, estimates, predictions, [150, 162])
measurements
```

```
[148.0, 154.2, 150.3, 149.9, 152.1, 154.6, 159.6, 157.4, 156.4, 161.0]
```

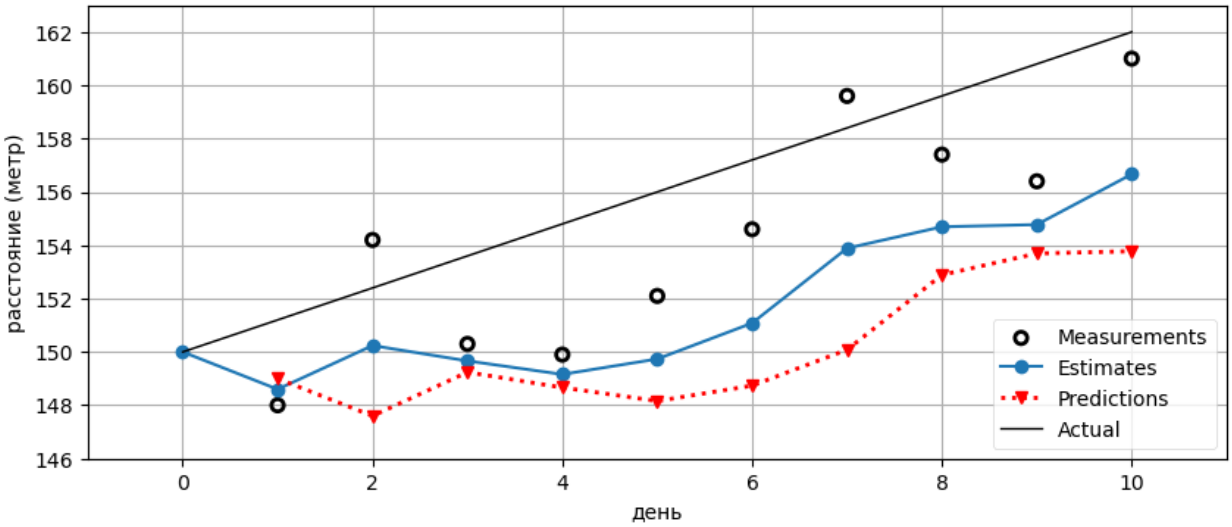


Очень хорошо! Здесь много данных, поэтому давайте поговорим о том, как их интерпретировать. Жирная синяя линия показывает оценку, полученную с помощью фильтра. Все начинается в день 0 с первоначальной оценки в 150 метров. Красной линией показан прогноз, сделанный на основе расстояния за предыдущий день. Итак, в первый день предыдущее расстояние составляло 150 фунтов, прибавка в расстоянии составила 1 метр, и, таким образом, первый прогноз - 151 метр. Таким образом, оценка на первый день находится на полпути между прогнозом и измерением и составляет 149,8 метра. Ниже приведена распечатка предыдущего расстояния, прогнозируемого расстояния и новой оценки за каждый день. Наконец, тонкая черная линия показывает фактическую прибавку в расстояние между А и Б.

Полученные оценки не являются линейными, но они более прямые, чем результаты измерений, и несколько близки к созданной нами линии тренда. Кроме того, со временем, похоже, ситуация улучшается.

Результаты фильтрации могут показаться вам довольно глупыми; конечно, данные будут выглядеть хорошо, если мы сделаем вывод, что наш прирост в расстоянии составляет около 1 метра в день! Давайте посмотрим, что делает фильтр, если наше первоначальное предположение неверно. Давайте спрогнозируем, что расстояние сокращается на 1 метр в день:

```
e, p = predict_using_gain_guess(initial_estimate, -1.)
plot_results(measurements, e, p, [150, 162])
```



Не так впечатляет. Оценки быстро расходятся с результатами измерений. Очевидно, что фильтр, который требует от нас правильного определения скорости изменения, не очень полезен. Даже если наше первоначальное предположение было правильным, фильтр не сработает, как только скорость изменения изменится. Если пункт Б перестанет отдаляться, фильтру будет крайне сложно приспособиться к этому изменению. Обратите внимание, что он приспосабливается! Показатели растут, хотя мы говорим ему, что пункт Б приближается по 1 метру в день. Он просто не может адаптироваться достаточно быстро.

Но что, если? Что, если вместо того, чтобы оставить отдаление пунтка Б на первоначальном уровне в 1 метр (или сколько угодно еще), мы рассчитаем ее на основе существующих измерений и оценок. В первый день наша оценка расстояния составляет:

$$(150+1)+\frac{4}{10}(148-151)=149.8$$

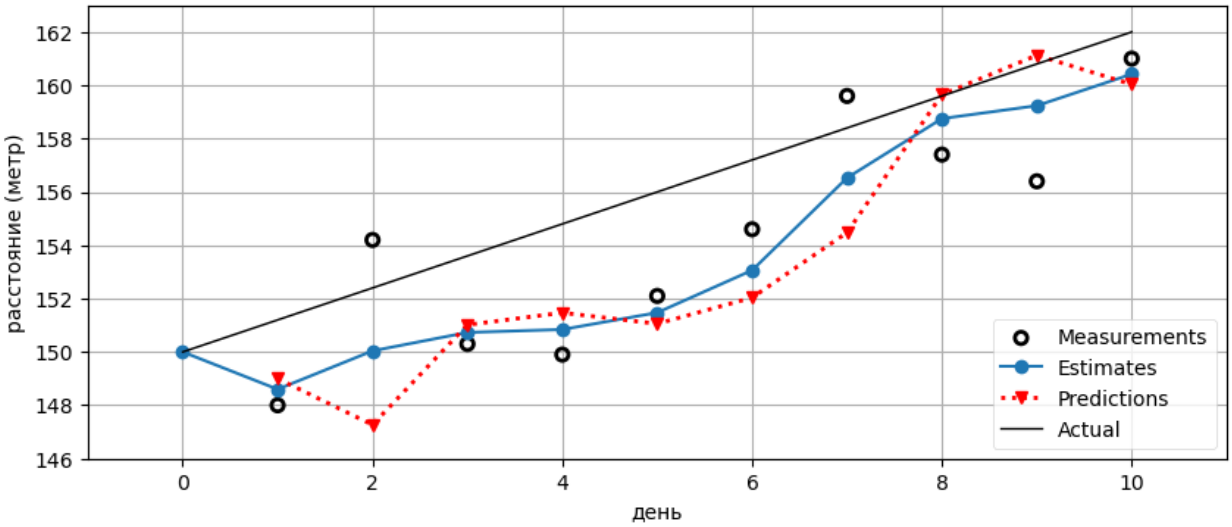
На следующий день мы измерили 154,2, что означает увеличение расстояния на 4,4 метра (поскольку 154,2 - 149,8 = 4,4), а не на 1. Можем ли мы как-то использовать эту информацию? Похоже на то. В конце концов, само измерение расстояния основано на реальном измерении расстояния, поэтому в нем содержится полезная информация. Наша оценка увеличения расстояния может быть не идеальной, но это, безусловно, лучше, чем просто предполагать, что прирост составляет 1 метр. Данные лучше, чем предположение, даже если они зашумлены.

Люди на самом деле не согласны с этим, поэтому убедитесь, что вы согласны. Два измерения расстояния с помехами дают нам предполагаемое увеличение / сокращение расстояния. Эта оценка будет очень неточной, если измерения будут неточными, но в этом расчете все равно есть информация. Представьте, что вы мониторите вертикальное перемещение пункта геодинамического полигона до 1 мм, и нивелир показывает, что пункт поднялся больше чем по оценки модели движения литосферных плит на 10 мм. Вертикальная составляющая перемещения пункта могла меняться от 8 до 12 мм, в зависимости от ошибок измерений, но мы знаем, что пункт "поднялся", и примерно на сколько. Это информация. Что мы делаем с информацией? Никогда не выбрасывайте ее!

Вернемся к расстоянию между А и Б. Следует ли нам установить новое правило изменения расстояния в 4,4 метра в день? Вчера мы думали, что увеличение в расстоянии составило 1 метр, сегодня мы думаем, что это 4,4 метра. У нас есть две цифры, и мы хотим их как-то объединить. Хм, похоже, у нас снова та же проблема. Давайте воспользуемся тем же инструментом, и единственным инструментом, который у нас есть на данный момент, - выберем значение, среднее между этими двумя. На этот раз я буду использовать другое произвольно выбранное число, $\frac{1}{3}$. Уравнение такое же, как и для оценки веса, за исключением того, что мы должны учитывать время, потому что это показатель (прирост в день).:

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

```
measurement = 150.  # initial guess
gain_rate = -1.0  # initial guess

time_step = 1.
measurement_scale = 4./10
gain_scale = 1./3
estimates = [measurement]
predictions = []

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

    # update step
    residual = z - measurement

    gain_rate = gain_rate + gain_scale   * (residual/time_step)
    measurement    = measurement    + measurement_scale * residual

    estimates.append(measurement)

plot_results(measurements, estimates, predictions, [150, 162])
```



Так выглядит намного лучше. Из-за неверного первоначального предположения о увеличении расстояния, равной -1, фильтру требуется несколько дней, чтобы точно спрогнозировать расстояния, но как только он это сделает, он начнет точно отслеживать это расстояние. Мы не использовали никакой методологии для выбора наших коэффициентов масштабирования $\frac{4}{10}$ и $\frac{1}{3}$ (на самом деле, это плохой выбор для данной задачи), но в остальном все математические расчеты основывались на очень разумных предположениях. Напомним, что вы можете изменить значение параметра "time_step" на большее и повторно запустить ячейку, если хотите увидеть график, нарисованный шаг за шагом.

И последнее замечание, прежде чем мы продолжим. На этапе прогнозирования я написал строку

```
gain_rate = gain_rate
```

Очевидно, что это не имеет никакого эффекта и может быть удалено. Я написал это, чтобы подчеркнуть, что на этапе прогнозирования вам необходимо спрогнозировать следующее значение для всех переменных, как "measurement", так и `gain_rate`. Вскоре это станет актуальным. В этом случае мы предполагаем, что коэффициент усиления не меняется, но когда мы обобщим этот алгоритм, мы уберем это предположение.

g-h Фильтр

Этот алгоритм известен как g-h filter или фильтр α-β. g и h относятся к двум коэффициентам масштабирования, которые мы использовали в нашем примере. g - это масштабирование, которое мы использовали для измерения (расстояние в нашем примере), а h - масштабирование для изменения измерения с течением времени (метр в день в нашем примере). α и β - это просто разные названия, используемые для этих коэффициентов.

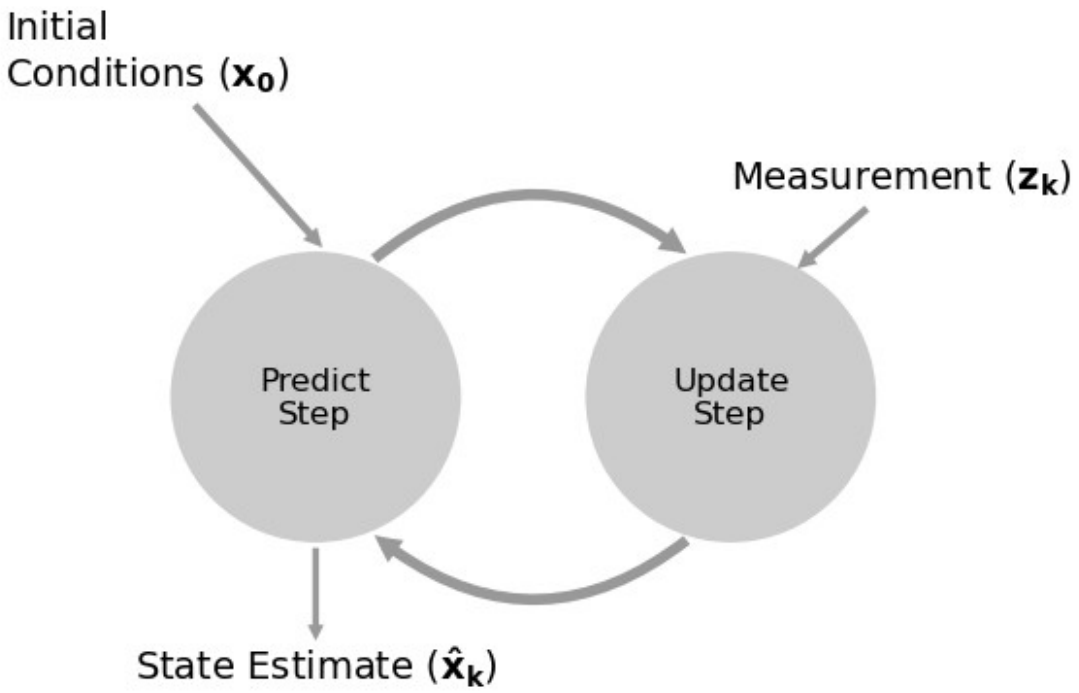
Этот фильтр является основой для огромного количества фильтров, включая фильтр Калмана. Другими словами, фильтр Калмана - это разновидность фильтра g-h, что мы и увидем позже. Так же, как и фильтр (метод) наименьших квадратов, о котором вы, возможно, слышали, и фильтр Бенедикта-Борднера, о котором вы, вероятно, не слышали. Каждый фильтр по-своему присваивает значения g и h, но в остальном алгоритмы идентичны. Например, фильтр Бенедикта-Борднера присваивает константы g и h, ограниченные определенным диапазоном значений. Другие фильтры, такие как фильтр Калмана, будут динамически изменять значения g и h на каждом временном шаге.

Позвольте мне повторить ключевые моменты, поскольку они очень важны. Если вы не поймете их, вы не поймете и остальную часть курса. Поняв их остальная часть курса будет для вас естественным образом раскрываться в виде математических пояснений к различным вопросам "что, если", которые мы будем задавать о g и h. Математика может выглядеть совершенно по-другому, но алгоритм будет точно таким же.

- Несколько измерений лучше, чем одно измерение, поэтому ничего не выбрасывайте, какими бы неточными они ни были.
- Всегда выбирайте интервал между двумя измерениями, чтобы получить более точную оценку.
- Прогнозируйте следующее измерение и скорость изменения на основе текущей оценки и того, насколько, по нашему мнению, оно изменится.
- Затем новая оценка выбирается как промежуточный результат между прогнозом и следующим измерением, масштабируемым по степени точности каждого из них.

Давайте посмотрим на наглядное описание алгоритма.

```
book_plots.predict_update_chart()
```



Введем более формальную терминологию. Система - это *объект*, который мы хотим оценить. В этой главе система - это то, что мы пытаемся измерить. Эта терминология заимствована из теории систем управления. https://en.wikipedia.org/wiki/Plant_ (контрольная теория)

Состояние системы (англ. *state*) - это текущая конфигурация или значения этой системы, которые нас интересуют. Нас интересуют только показания расстояния. Если я измерю расстояние в 100 метров, то значение будет равно 100 метров. Мы определяем состояние, исходя из того, что для нас важно.

Измерение (англ. *measurement*) - это измеренное значение системы. Измерения могут быть неточными, поэтому оно может не совпадать с состоянием.

Оценка состояния (англ. *state estimate*) - это оценка состояния нашим фильтром. Например, для расстояния в 100 метров наша оценка может составлять 99,327 метра из-за ошибок светодальномера. Обычно это сокращенно обозначается как "оценка".

Другими словами, состояние следует понимать как фактическое значение системы. Обычно это значение для нас "скрыто". Если бы я воспользовался светодальномером, вы бы получили "измерение". Мы называем это "наблюдением", поскольку вы можете непосредственно наблюдать это измерение. Напротив, вы никогда не сможете непосредственно наблюдать за расстоянием между А и Б, вы можете только измерить его.

Эти слова "скрытый" и "наблюдаемый" очень важены. Любая задача оценки состоит в формировании оценки скрытого состояния с помощью наблюдаемых измерений. Если вы

читали литературу, то эти термины используются при определении проблемы, поэтому вам нужно с ними ознакомиться.

Мы используем *модель процесса* (англ. *process model*) для математического моделирования системы. В этой главе наша модель процесса основана на предположении, что расстояние между А и Б равно вчерашнему расстоянию плюс прибавка за последний день. Модель процесса не моделирует датчики и не учитывает их каким-либо иным образом. Другим примером может служить модель процесса для автомобиля. Модель процесса может быть такой: "Расстояние равно скорости, умноженной на время". Эта модель не идеальна, поскольку скорость автомобиля может изменяться в течение ненулевого периода времени, шины могут скользить по дороге и так далее. *Системная ошибка* (англ. *system error*) или *ошибка процесса* (англ. *process error*) - это ошибка в данной модели. Мы никогда не знаем это значение в точности; если бы мы знали, мы могли бы усовершенствовать нашу модель, чтобы получить нулевую ошибку.

Этап прогнозирования известен как "распространение системы". Он использует "модель процесса" для формирования новой "оценки состояния". Из-за "ошибки процесса" эта оценка является неполной. Предполагая, что мы отслеживаем данные с течением времени, мы говорим, что мы "распространяем" состояние в будущее. В некоторых текстах это называется "эволюцией".

Этап обновления известен как "обновление измерений". Одна итерация распространения системы и обновления измерений называется "эпохой".

Теперь давайте рассмотрим несколько различных проблемных областей, чтобы лучше понять этот алгоритм. Рассмотрим задачу отслеживания движения поезда на рельсах. Рельсы ограничивают положение поезда в строго определенном направлении. Кроме того, поезда большие и медленные. Им требуется много минут, чтобы значительно замедлиться или ускориться. Итак, если я знаю, что в момент времени t поезд находится на отметке 23 км и движется со скоростью 18 км/ч, я могу с абсолютной уверенностью спрогнозировать его местоположение в момент времени t + 1 секунда. Почему это так важно? Предположим, мы можем оценить его местоположение только с точностью до 250 метров. Поезд движется со скоростью 18 км/ч, что составляет 5 метров в секунду. В момент времени t+1 секунда поезд будет находиться на отметке 23,005 км, однако результаты измерений могут варьироваться от 22,755 км до 23,255 км. Таким образом, если при следующем измерении будет указано, что точка находится на отметке 23,4, мы знаем, что это, должно быть, неточно. Даже если в момент времени t машинист нажал на тормоза, поезд все равно будет находиться очень близко к 23,005 км, потому что поезд не может сильно замедлиться за 1 секунду. Если бы мы должны были разработать фильтр для решения этой проблемы мы разработали фильтр, который придавал бы очень высокий вес прогнозированию по сравнению с измерением.

Теперь рассмотрим задачу отслеживания брошенного груза в вакууме, к примеру в баллистическом гравиметре если бы мы жили на нормальной Земле (Нормальная Земля - модель Земли в которой плотности распределены равномерно от центра до края поверхности и сохраняется постоянная масса). Мы знаем, что баллистический объект движется параллельно вектору силы тяжести в вакууме, находясь в гравитационном поле нормальной Земли. Но на груз, брошенный на ("сумасшедшей") реальной Земле, влияет неоднородность самого гравитационного поля (свободно падающий груз меняет свою ортометрическую высоту, а на разной высоте разное нормальное значения силы тяжести).

Точность отслеживания падающего груза с помощью лазеров может быть очень высокой, а построенная модель гравитационного поля мягко говоря не очень точно. В этом случае мы, вероятно, разработали бы фильтр, который делал бы акцент на измерениях, а не на прогнозах.

И естественно существуют случаи когда мы будет строить фильтр который бы приравнивал равный вес измерениям и прогнозу.

Большая часть курса посвящена математическому выражению проблем, изложенных в последних трех абзацах, что затем позволяет нам найти оптимальное решение (в некотором математическом смысле). В этой главе мы просто будем присваивать разные значения g и h более интуитивно понятным и, следовательно, менее оптимальным способом. Но основная идея заключается в том, чтобы объединить несколько неточные измерения с несколько неточными моделями поведения систем, чтобы получить отфильтрованную оценку, которая лучше, чем любой источник информации сам по себе.

Мы можем выразить это в виде алгоритма:

Инициализация

```
1. Инициализируем состояние фильтра
2. Инициализируем нашу "уверенность" в состоянии
```

Прогнозируем

```
1. Используем поведение системы для прогнозирования состояния на
следующем временном шаге
2. Корректируем нашу "увереность", чтобы учесть неопределенность в
прогнозировании
```

Обновляем

```
1. Получите результат измерения и связанное с ним представление о его
точности
2. Вычислите разницу между оцененным состоянием и измерением
3. Новая оценка находится где-то на линии остатка
```

Мы будем использовать этот же алгоритм на протяжении всего курса, хотя и с некоторыми изменениями.

Условные обозначения

Я начну знакомить вас с обозначениями и именами переменных, используемыми в литературе. Некоторые из них уже использовались в приведенных выше диаграммах. Измерение обычно обозначается как z, и именно это обозначение мы будем использовать в этом курсе (в некоторых источниках используется y). Индекс k указывает на временной шаг, поэтому z_k - это измерения на момент времени k. Жирный шрифт обозначает вектор или матрицу. До сих пор мы рассматривали только один датчик и, следовательно, одно измерение с помощью датчика, но в целом у нас может быть n датчиков и n измерений. x

обозначает состояние нашей системы и он выделен жирным шрифтом, чтобы обозначить, что это вектор. В нашем примере со светодальномерами он представляет как начальное расстояния, так и начальную скорость увеличения расстояния, например, так:

$$x = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$

Здесь я использую ньютоновскую систему счисления с точкой над x для обозначения скорости. Точнее, точка означает производную от x по времени, которая, конечно же, является скоростью. При расстоянии в 150 метров и скорости 1 метр в день мы имеем

$$x = \begin{bmatrix} 150 \\ 1 \end{bmatrix}$$

Итак, алгоритм прост. Состояние инициализируется с помощью x_0, начальной оценки. Затем мы входим в цикл, прогнозируя состояние на время или шаг k на основе значений времени (или шага) $k-1$. Затем мы получаем измерение z_k и выбираем некоторую промежуточную точку между измерениями и прогнозом, создавая оценку x_k.

Напишите g-h фильтр

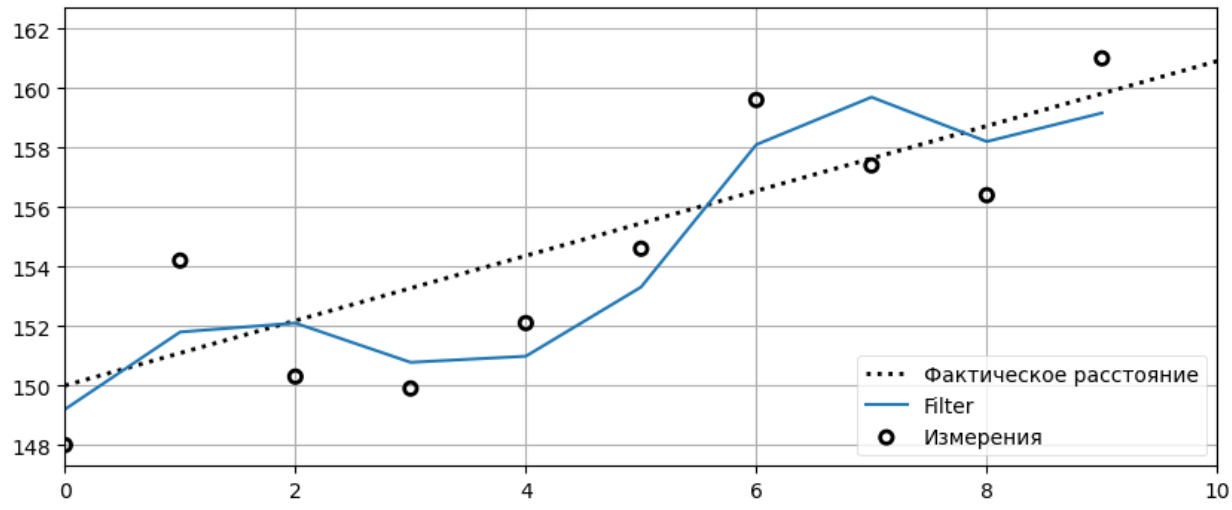
Хорошие алгоритм обязаны быть одинаковыми для любой задачи. Поэтому давайте перепишем код выше, чтобы он был универсальным и подходил для работы с любой проблемой. Используйте эту сигнатуру функции:

```
def g_h_filter(data, x0, dx, g, h, dt):
    """
    Выполняет g-h фильтрацию для 1 переменной состояния с
фиксированными значениями g и h.

    'data' содержит данные, подлежащие фильтрации;
    'x0' начальное значение для нашей переменной состояния;
    'dx' начальная скорость изменения для нашей переменной состояния;
    'g' масштабный коэффициент g для g-h;
    'h' масштабный коэффициент h для g-h;
    'dt' длина временного шага.
    """
```

Функция (алгоритм) должна возвращать данные в виде числового массива, а не списка. Протестируйте его, введя те же данные о расстоянии, что и раньше, выведите результаты на график и визуально убедитесь, что он работает.

```
def plot_g_h_results(measurements, filtered_data,
                     title='', z_label='Измерения',
                     **kwargs):

    book_plots.plot_filter(filtered_data, **kwargs)
    book_plots.plot_measurements(measurements, label=z_label)
    plt.legend(loc=4)
```

```
    plt.title(title)
    plt.gca().set_xlim(left=0,right=len(measurements))

    return


def g_h_filter(data, x0, dx, g, h, dt) -> np.array:
    est_x = x0
    result = []
    for z in data:
        # prediction step
        x_pred = est_x + (dx * dt)

        # update step
        residual = z - x_pred
        dx = dx + h * (residual) / dt

        est_x = x_pred + g * residual

        result.append(est_x)
    return np.array(result)


# uncomment to run the filter and plot the results
book_plots.plot_track([0, 11], [150, 162], label='Фактическое
расстояние')
data = g_h_filter(data=measurements, x0=150., dx=1., g=6./10, h=2./3,
dt=1.)
plot_g_h_results(measurements, data)
```



Выбор g и h

Фильтр g-h - это не один фильтр, это классификация для семейства фильтров. Эли Брукнер в книге "Отслеживание и фильтрация по Калману: просто" перечисляет 11

фильтров, и я уверен, что их еще больше. Кроме того, у каждого типа фильтров есть множество подтипов. Каждый фильтр отличается тем, как выбраны значения g и h. Поэтому я не могу дать здесь универсального совета. Некоторые фильтры устанавливают значения g и h в качестве констант, другие изменяют их динамически. Фильтр Калмана динамически изменяет их на каждом шаге. Некоторые фильтры позволяют g и h принимать любое значение в пределах диапазона, другие ограничивают зависимость одного от другого с помощью некоторой функции

$f(\dot{x})$, where $g = f(h)$.

Однако в типичной формулировке фильтра Калмана вообще не используются g и h. Фильтр Калмана является g-h фильтром, поскольку математически он сводится к этому алгоритму. Когда мы разрабатываем фильтр Калмана, мы используем критерии проектирования, которые могут быть математически сведены к g и h, но форма фильтра Калмана обычно является гораздо более эффективным способом осмысления проблемы. Не волнуйтесь, если сейчас не совсем понятно о чем речь, все станет ясно, как только мы разработаем теорию фильтра Калмана.

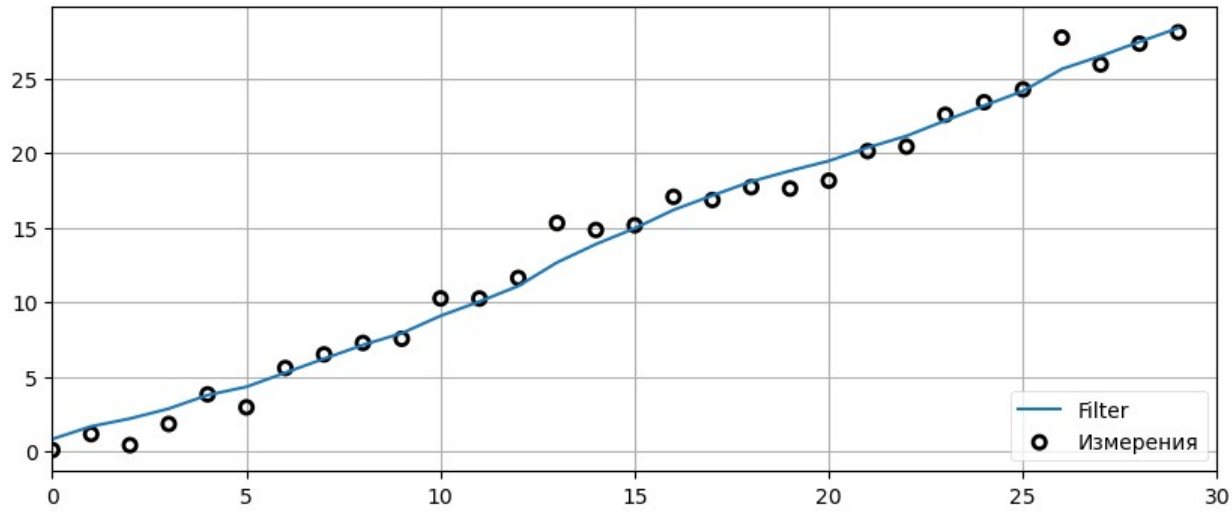
Стоит посмотреть, как изменение значений g и h влияет на результаты, поэтому мы рассмотрим несколько примеров. Это даст нам четкое представление об основных преимуществах и ограничениях этого типа фильтров и поможет нам понять поведение гораздо более сложного фильтра Калмана.

Функция измерений

Давайте напишем функцию, которая генерирует зашумленные измерения. В этом курсе наши измерения буду содержать в себе так называемый белый шум. Мы еще не рассматривали статистику, чтобы полностью понять определение белого шума. По сути, думайте об этом как о данных, которые случайным образом изменяются то в большую, то в меньшую сторону. Это последовательно некоррелированная случайная величина с нулевым средним значением и конечной дисперсией.

Белый шум может генерироваться с помощью `numpy.random.randn()`. Нам нужна функция, которую мы будем вызывать с начальным значением, величиной изменения за шаг, количеством шагов и уровнем шума, который мы хотим добавить. Она должна возвращать список данных. Протестируйте его, создав 30 точек, отфильтровав их с помощью `g_h_filter()` и построив результаты с помощью `plot_g_h_results()`.

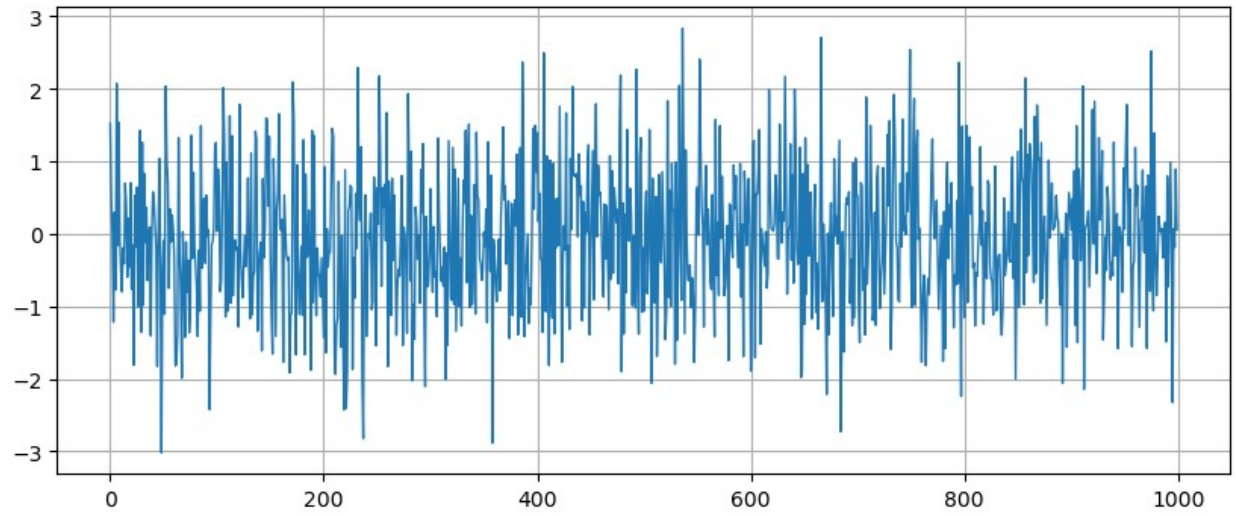
```
from numpy.random import randn
def gen_data(x0, dx, count, noise_factor):
    return [x0 + dx*i + randn()*noise_factor for i in range(count)]

measurements = gen_data(0, 1, 30, 1)
data = g_h_filter(data=measurements, x0=0., dx=1., dt=1., g=.2,
h=0.02)
plot_g_h_results(measurements, data)
```



Обсуждение

`rand()` возвращает случайные числа с центром около 0. Значение варьируется на *одно стандартное отклонение*. Я нанес на график 1000 вызовов `randn()` - вы можете видеть, что значения сосредоточены вокруг нуля и в основном варьируются от -1 до +1, хотя иногда они намного больше/меньше.

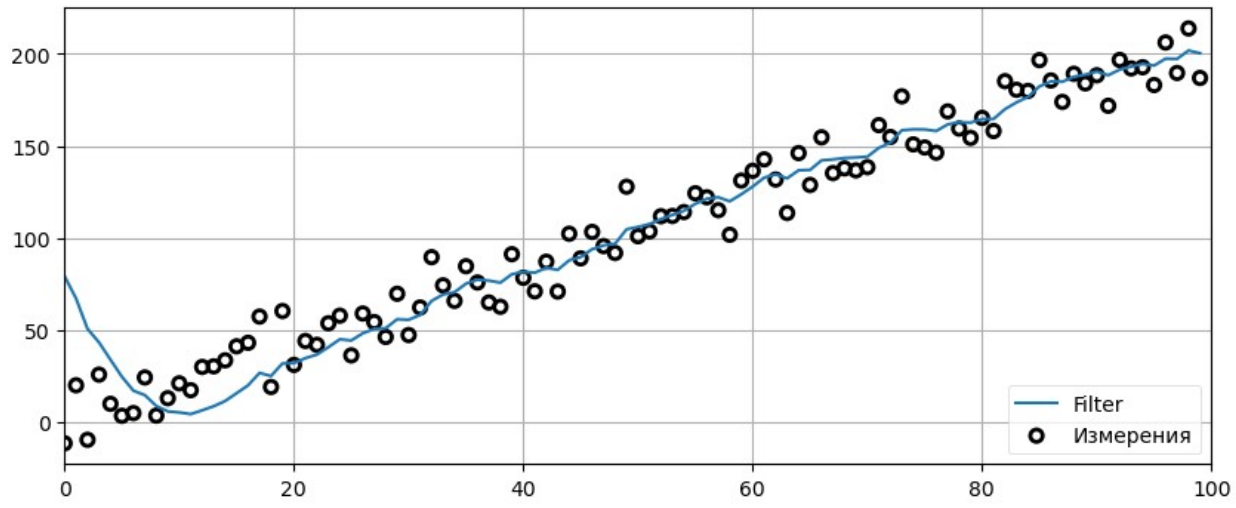
```
plt.plot([randn() for _ in range(1000)], lw=1);
```



Плохие начальные условия

Теперь напишем код, который использует `gen_data` и `g_h_filter` для фильтрации 100 точек, которые начинаются с 5, имеют производную от 2, коэффициент масштабирования шума равен 10 и используют g=0,2 и h=0,02. Установите ваше первоначальное предположение о том, что x равно 100.

```
zs = gen_data(x0=5., dx=2., count=100, noise_factor=10)
data = g_h_filter(data=zs, x0=100., dx=2., dt=1., g=0.2, h=0.02)
plot_g_h_results(measurements=zs, filtered_data=data)
```

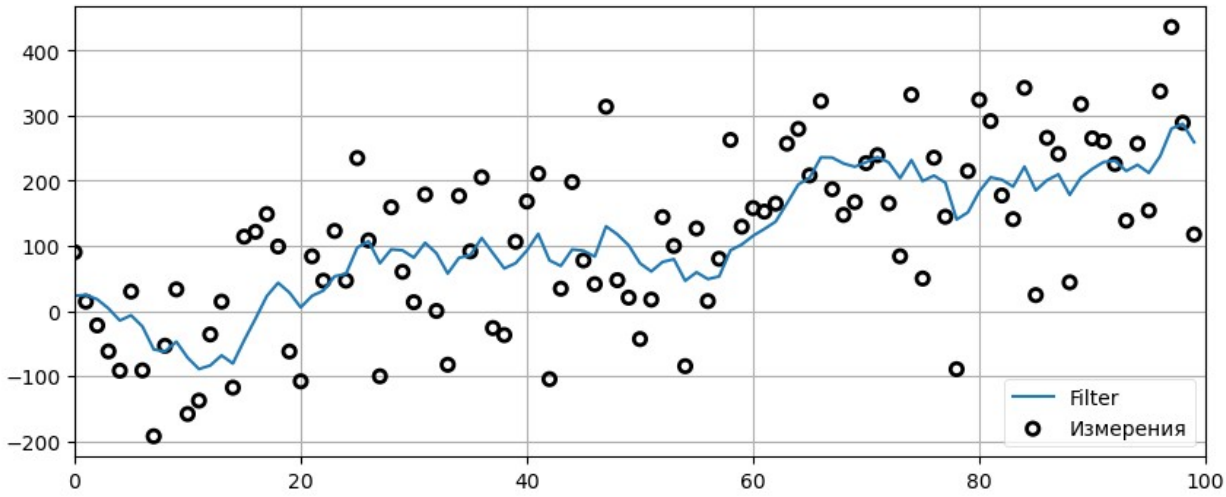


Мы видим, что фильтр начинает с оценок, которые далеки от измеренных данных из-за неверного первоначального предположения о 100. Вы можете видеть, что он делает "прозвон", прежде чем остановиться на измеренных данных. "Перезвон" означает, что сигнал пропускает данные по синусоидальной схеме. Это очень распространенное явление в фильтрах, и большая часть работы по проектированию фильтров посвящена минимизации перезвона. Это тема, к рассмотрению которой мы еще не готовы, но я хотел показать вам это явление.

Сильный шум

Повторим тот же тест, но на этот раз используем коэффициент шума, равный 100. Устраним "прозвон", изменив начальное условие со 100 на 5.

```
zs = gen_data(x0=5., dx=2., count=100, noise_factor=100)
data = g_h_filter(data=zs, x0=5., dx=2., g=0.2, h=0.02, dt=1)
plot_g_h_results(measurements=zs, filtered_data=data)
```



Не так уж и здорово. Мы видим, что, возможно, отфильтрованный сигнал меняется меньше, чем зашумленный, но он далек от прямой линии. Если бы мы отображали только отфильтрованный результат, никто бы не догадался, что сигнал начинается с 5 и увеличивается на 2 на каждом временном шаге. И хотя в некоторых местах фильтр, по-видимому, уменьшает шум, в других местах он, похоже, превышает или занижает уровень.

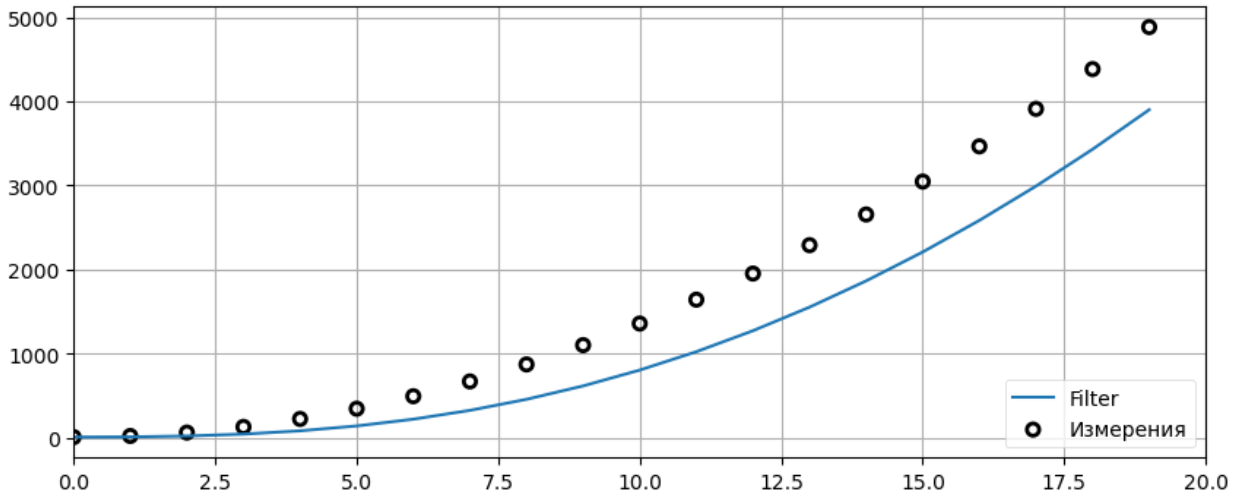
На данный момент мы знаем недостаточно, чтобы по-настоящему судить об этом. Мы добавили много шума; возможно, это все, на что способна фильтрация. Однако,я надеюсь, к концу курса мы сможем сделать фильтр гораздо лучше.

Эффект ускорения

Напишим новую функцию генерации данных, которая добавляет постоянный коэффициент ускорения к каждой точке. Другими словами, увеличивайте dx при вычислении каждой точки, чтобы скорость (dx) постоянно увеличивалась. Установим уровень шума равным 0, $g=0,2 и $h=0,02 и построим график результатов, используя `plot_g_h_results`.

Задание: Поиграйте с различными ускорениями и временными интервалами. Объясните, что вы видите.

```
def gen_data(x0, dx, count, noise_factor, accel=0.):
    zs = []
    for i in range(count):
        zs.append(x0 + accel * (i**2) / 2 + dx*i +
randn()*noise_factor)
        dx += accel
    return zs

predictions = []
zs = gen_data(x0=10., dx=0., count=20, noise_factor=0, accel=9.)
data = g_h_filter(data=zs, x0=10., dx=0., g=0.2, h=0.02, dt=1)
plot_g_h_results(measurements=zs, filtered_data=data)
```



Каждый прогноз отстает от сигнала. Если вдуматься в происходящее, это имеет смысл. Наша модель предполагает, что скорость постоянна. Фильтр g-h вычисляет первую производную от x (мы используем $\dot{x}$ для обозначения производной), но не вторую производную $\ddot{x}$. Итак, мы предполагаем, что $\ddot{x}=0$. На каждом шаге прогнозирования мы прогнозируем новое значение x как $x+\dot{x}*t$. Но из-за ускорения прогноз обязательно должен отставать от фактического значения. Затем мы пытаемся вычислить новое значение для $\dot{x}$, но из-за фактора h мы лишь частично приспосабливаем $\dot{x}$ к новой скорости. На следующей итерации мы снова потерпим неудачу.

Обратите внимание, что для устранения этой проблемы мы не можем настроить параметры g или h. Это называется "ошибкой задержки" или "системной ошибкой" системы. Это фундаментальное свойство фильтров g-h. Возможно, ваш разум уже предлагает решения или обходные пути для решения этой проблемы. Как вы могли бы ожидать, этой проблеме было посвящено множество исследований, и в этом курсе мы рассмотрим различные решения этой проблемы.

> Суть в том, что фильтр хорош настолько, насколько хороша математическая модель, используемая для описания системы.

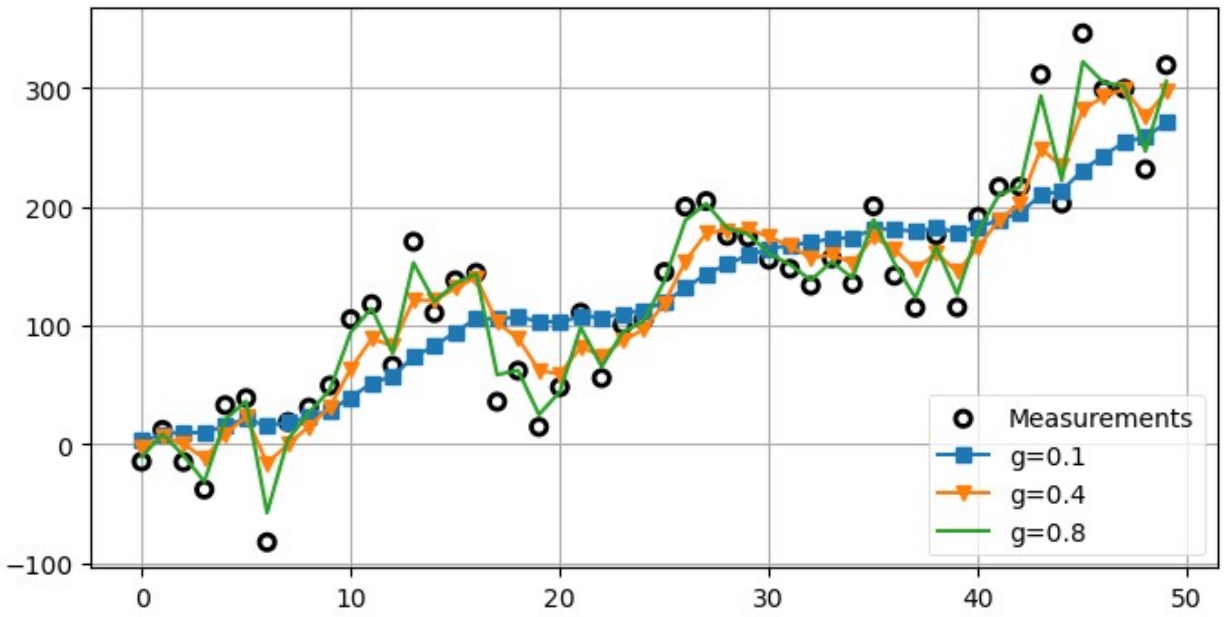
Изменяющийся g

Теперь давайте посмотрим на эффект от изменения g. Прежде чем мы это сделаем, вспомним, что g - это масштабный коэффициент для выбора между измерением и прогнозированием. Как вы думаете, каков будет эффект от большого значения g? Небольшой?

Теперь пусть `noise_factor=50` и `dx=5`. Постройте график результатов для $g=0.1, 0.4, 0.8$.

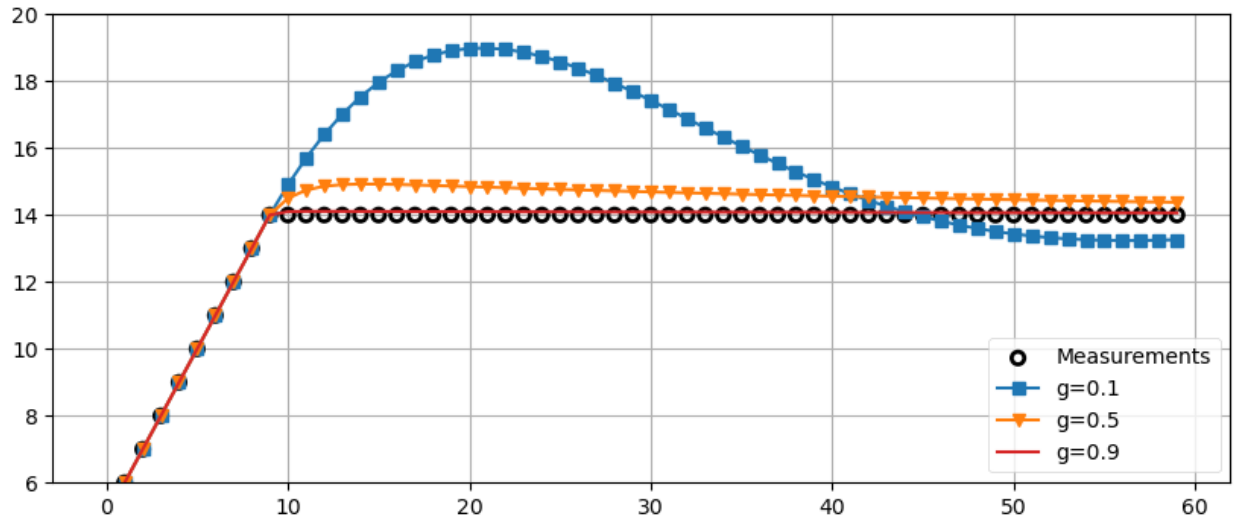
```
zs = gen_data(x0=5., dx=5., count=50, noise_factor=50)
data1 = g_h_filter(data=zs, x0=0., dx=5., dt=1., g=0.1, h=0.01)
data2 = g_h_filter(data=zs, x0=0., dx=5., dt=1., g=0.4, h=0.01)
data3 = g_h_filter(data=zs, x0=0., dx=5., dt=1., g=0.8, h=0.01)
```

```
with book_plots.figsize(y=4):
    book_plots.plot_measurements(zs, color='k')
    book_plots.plot_filter(data1, label='g=0.1', marker='s', c='C0')
    book_plots.plot_filter(data2, label='g=0.4', marker='v', c='C1')
    book_plots.plot_filter(data3, label='g=0.8', c='C2')
    plt.legend(loc=4)
```



Очевидно, что чем больше g, тем тщательнее мы следим за измерением, а не за прогнозированием. Когда $g=0,8$, мы практически точно отслеживаем сигнал и почти не отбрасываем шум. Можно было бы наивно заключить, что значение g всегда должно быть очень маленьким, чтобы максимально снизить уровень шума. Однако это означает, что мы в основном игнорируем результаты измерений в пользу нашего прогноза. Что происходит, когда сигнал изменяется не из-за шума, а из-за фактического изменения состояния? Давайте посмотрим. Я создам данные, которые будут иметь $\dot{x}=1$, в течение 9 шагов, прежде чем изменить их на $\dot{x}=0$.

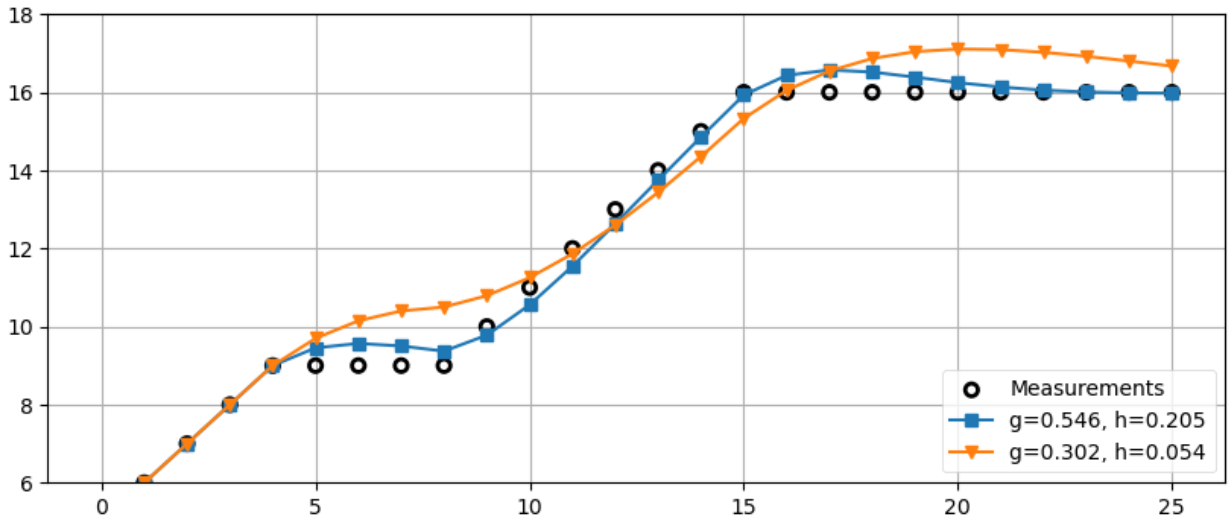
```
zs = [5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
for i in range(50):
    zs.append(14)

data1 = g_h_filter(data=zs, x0=4., dx=1., dt=1., g=0.1, h=0.01)
data2 = g_h_filter(data=zs, x0=4., dx=1., dt=1., g=0.5, h=0.01)
data3 = g_h_filter(data=zs, x0=4., dx=1., dt=1., g=0.9, h=0.01)

book_plots.plot_measurements(zs)
book_plots.plot_filter(data1, label='g=0.1', marker='s', c='C0')
book_plots.plot_filter(data2, label='g=0.5', marker='v', c='C1')
book_plots.plot_filter(data3, label='g=0.9', c='C3')
plt.legend(loc=4)
plt.ylim([6, 20]);
```



Здесь мы можем увидеть последствия игнорирования измерений. Мы не только отфильтровываем шум, но и корректируем изменения в сигнале.

Может быть, нам нужен фильтр "Златовласка", где g не слишком большой, не слишком маленький, а в самый раз? Ну, не совсем. Как упоминалось ранее, разные фильтры выбирают g и h по-разному в зависимости от математических свойств задачи. Например, фильтр Бенедикта-Борднера был изобретен для минимизации временной ошибки в нашем примере, где $\dot{x}$ совершает ступенчатый скачок. Мы не будем обсуждать этот фильтр в этом курсе, но вы можете оценить работу фильтра из анализа этих двуъ графиков, выбранных с разными допустимыми парами g и h. Такая конструкция фильтра сводит к минимуму временные ошибки при скачках шага в $\dot{x}$ за счет того, что он не является оптимальным для других типов изменений в $\dot{x}$.

```
zs = [5,6,7,8,9,9,9,9,9,10,11,12,13,14,
      15,16,16,16,16,16,16,16,16,16,16,16]
data1 = g_h_filter(data=zs, x0=4., dx=1., dt=1., g=.302, h=.054)
data2 = g_h_filter(data=zs, x0=4., dx=1., dt=1., g=.546, h=.205)

book_plots.plot_measurements(zs)
book_plots.plot_filter(data2, label='g=0.546, h=0.205', marker='s',
c='C0')
book_plots.plot_filter(data1, label='g=0.302, h=0.054', marker='v',
c='C1')
plt.legend(loc=4)
plt.ylim([6, 18]);
```

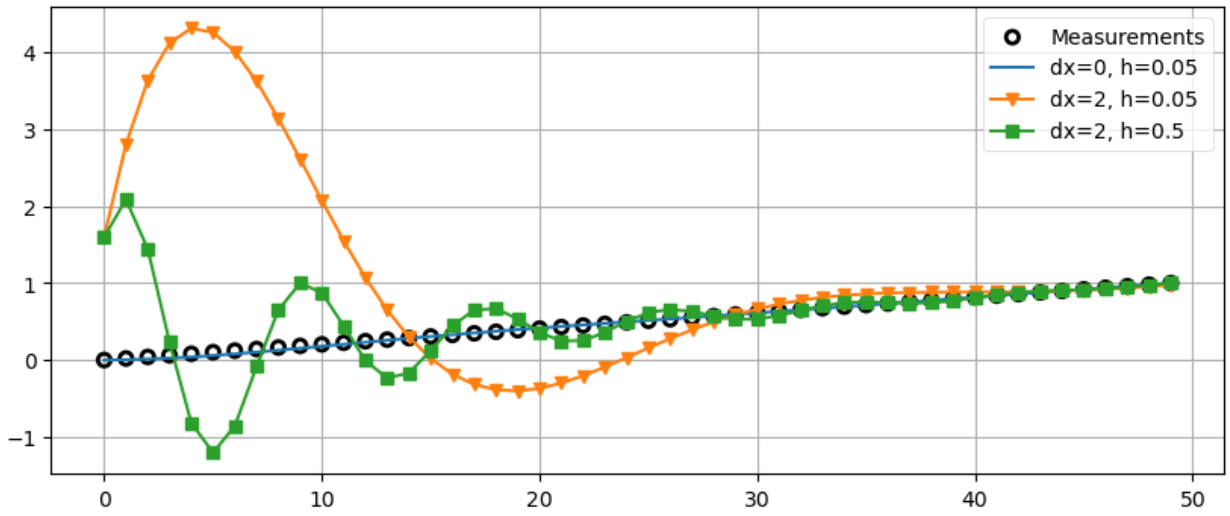


Изменения h

Теперь давайте оставим g без изменений и исследуем влияние изменения h. Мы знаем, что h влияет на то, насколько мы предпочитаем измерение $dotx$ по сравнению с нашим прогнозом. Но что это *означает*? Если наш сигнал сильно меняется (быстро по сравнению с временным шагом нашего фильтра), то большое значение h заставит нас быстро реагировать на эти временные изменения. Меньшее значение h заставит нас реагировать медленнее.

Мы рассмотрим три примера. У нас есть бесшумное измерение, которое медленно переходит от 0 к 1 за 50 шагов. Наш первый фильтр использует почти правильное начальное значение для $\dot{x}$ и небольшое значение h. Из выходных данных видно, что фильтр очень близок к сигналу. Второй фильтр использует грубое предположение о $\dot{x}=2$. Здесь мы видим, что фильтр "звонит", пока не успокоится и не найдет сигнал. Третий фильтр использует те же условия, но теперь он устанавливает $h=0.5$. Если вы посмотрите на амплитуду звукового сигнала, то увидите, что она намного меньше, чем на втором графике, но частота больше. Он также сходится немного быстрее, чем при использовании второго фильтра, хотя и ненамного.

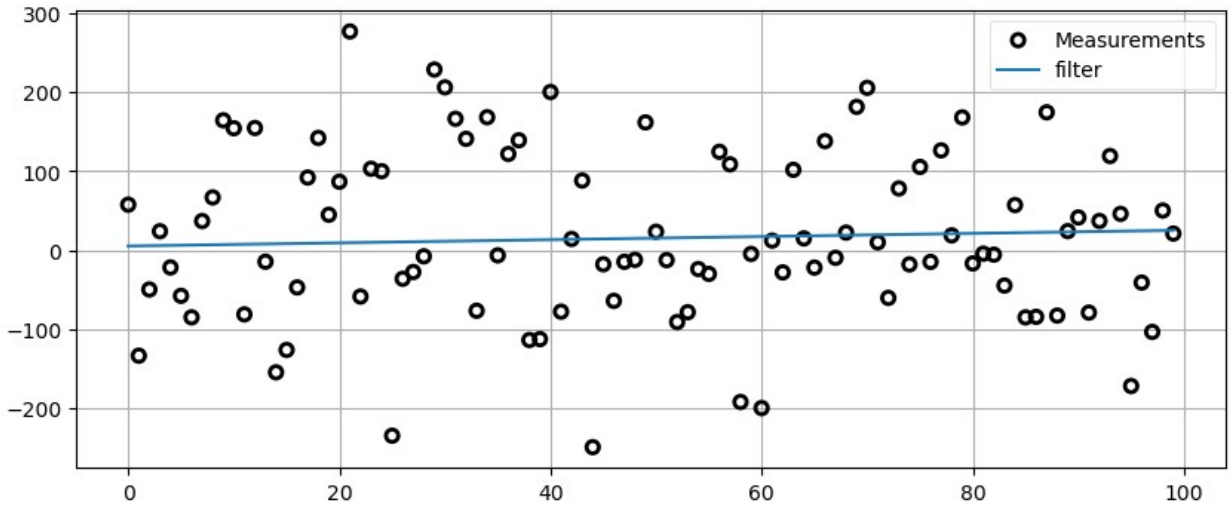
```
zs = np.linspace(0, 1, 50)

data1 = g_h_filter(data=zs, x0=0, dx=0., dt=1., g=.2, h=0.05)
data2 = g_h_filter(data=zs, x0=0, dx=2., dt=1., g=.2, h=0.05)
data3 = g_h_filter(data=zs, x0=0, dx=2., dt=1., g=.2, h=0.5)

book_plots.plot_measurements(zs)
book_plots.plot_filter(data1, label='dx=0, h=0.05', c='C0')
book_plots.plot_filter(data2, label='dx=2, h=0.05', marker='v',
c='C1')
book_plots.plot_filter(data3, label='dx=2, h=0.5',  marker='s',
c='C2')
plt.legend(loc=1);
```



Не обманывайте фильтр

Вы можете установить для g и h любые значения. Вот фильтр, который отлично работает, несмотря на сильный шум.

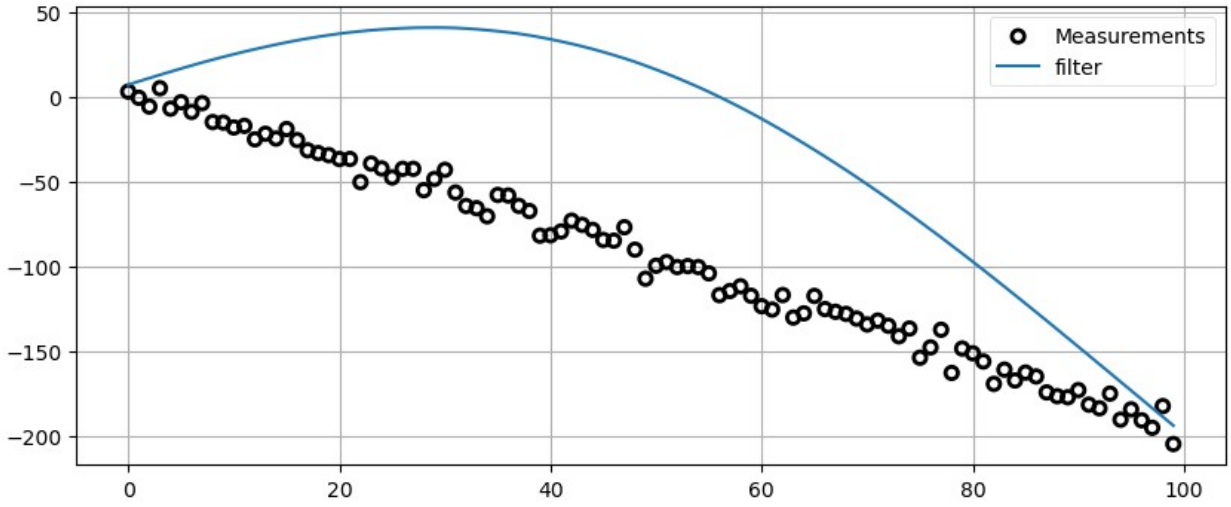
```
zs = gen_data(x0=5., dx=.2, count=100, noise_factor=100)
data = g_h_filter(data=zs, x0=5., dx=.2, dt=1., g=0., h=0.)

book_plots.plot_measurements(zs)
book_plots.plot_filter(data, label='filter')
plt.legend(loc=1);
```



Я блестяще вывел прямую линию из очень зашумленных данных! Возможно, мне пока не стоит пытаться получить медаль Филдса по математике. Я сделал это, установив значения для g и h равными 0. Что это дает? Это заставляет фильтр игнорировать измерения, и поэтому при каждом обновлении он вычисляет новую позицию как $x+\Delta x\,\Delta t$. Конечно, результат будет прямой, если мы проигнорируем измерения.

Фильтр, который игнорирует измерения, бесполезен. Я верю, что вы бы никогда не установили значения для g и h равными нулю, поскольку для этого требуется особый талант, которым обладаю только я. Вы всегда можете получить отличные результаты на основе тестовых данных. Когда вы попробуете применить свой фильтр к другим данным, вы будете разочарованы результатами, потому что вы точно настроили константы для определенного набора данных. g и h должны отражать реальное поведение системы, которую вы фильтруете, а не поведение одного конкретного набора данных. В последующих главах мы узнаем много нового о том, как это сделать. Пока я могу только сказать, что будьте осторожны, иначе вы получите отличные результаты с вашими тестовыми данными, но результаты, подобные этим, будут, когда вы перейдете к реальным данным:

```
zs = gen_data(x0=5, dx=-2, count=100, noise_factor=5)
data = g_h_filter(data=zs, x0=5., dx=2., dt=1., g=.005, h=0.001)
book_plots.plot_measurements(zs)
book_plots.plot_filter(data, label='filter')
plt.legend(loc=1);
```



Практика фильтрации