Kalman-and-Bayesian-Filters-in-Python

Preface

群名称：AibotBeginer
群　号：710288823

Mar 12th, 2022 https://github.com/duyongquan/LTSLAM

AibotBeginer 视觉SLAM
quandy2020@126.com

Outline

- Kalman and Bayesian Filters
- Motivation for this Book
- Reading Online
- PDF Version
- Downloading and Running the Book
- Jupyter
- SciPy, NumPy, and Matplotlib

3

Kalman and Bayesian Filters
卡尔曼和贝叶斯滤波

4 Kalman and Bayesian Filters

Sensors are noisy

- GPS in my car reports altitude, Each time pass the same point, it reports a slightly different altitude
- Kitchen scale gives different readings weigh the same object twice.
- Sensor is very noisy, environment makes data collection difficult. How can track?

Rudolf Emil Kálmán
鲁道夫·卡尔曼 匈牙利裔美国数学家

Introduce

- Apollo missions to the moon
- Medical imaging
- Aaircraft, submarines, cruise missiles
- Robots, IoT (Internet of Things) , laboratory instruments

5

Motivation for this Book
书的目标

6

Motivation for this Book

- The theory is beautiful, but quite difficult to learn
  - signal processing
  - control theory
  - probability and statistics
- Ohter books quite difficult to learn
  - notation is introduced without explanation
  - books are almost devoid of examples or worked problems
  - no idea as to what real world phenomena these words and math were attempting to describe

7

Reading Online
在线阅读

8 Reading Online

- GitHub
  - https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python
- ninder
  - binder serves interactive notebooks online
  - http://mybinder.org/repo/rlabbe/Kalman-and-Bayesian-Filters-in-Python
- nbviewe
  - https://nbviewer.org/github/rlabbe/Kalman-and-Bayesian-Filters-in-Python/tree/master/

9

PDF Version
PDF 版本

10

PDF Version

- PDF book

https://drive.google.com/file/d/0By_SW19c1BfhSVFzNHc0SjduNzg/view?resourcekey=0-41olC9ht9xE3wQe2zHZ45A

11

Downloading and Running the Book

下载和运行

12

Downloading and Running the Book

Installation

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

Introduce

This book is intended to be interactive and I recommend using it.
Its a little more effort to set up.
You can perform experiments, see how filters react to different data, see how different filters react to the same data, and so on

13

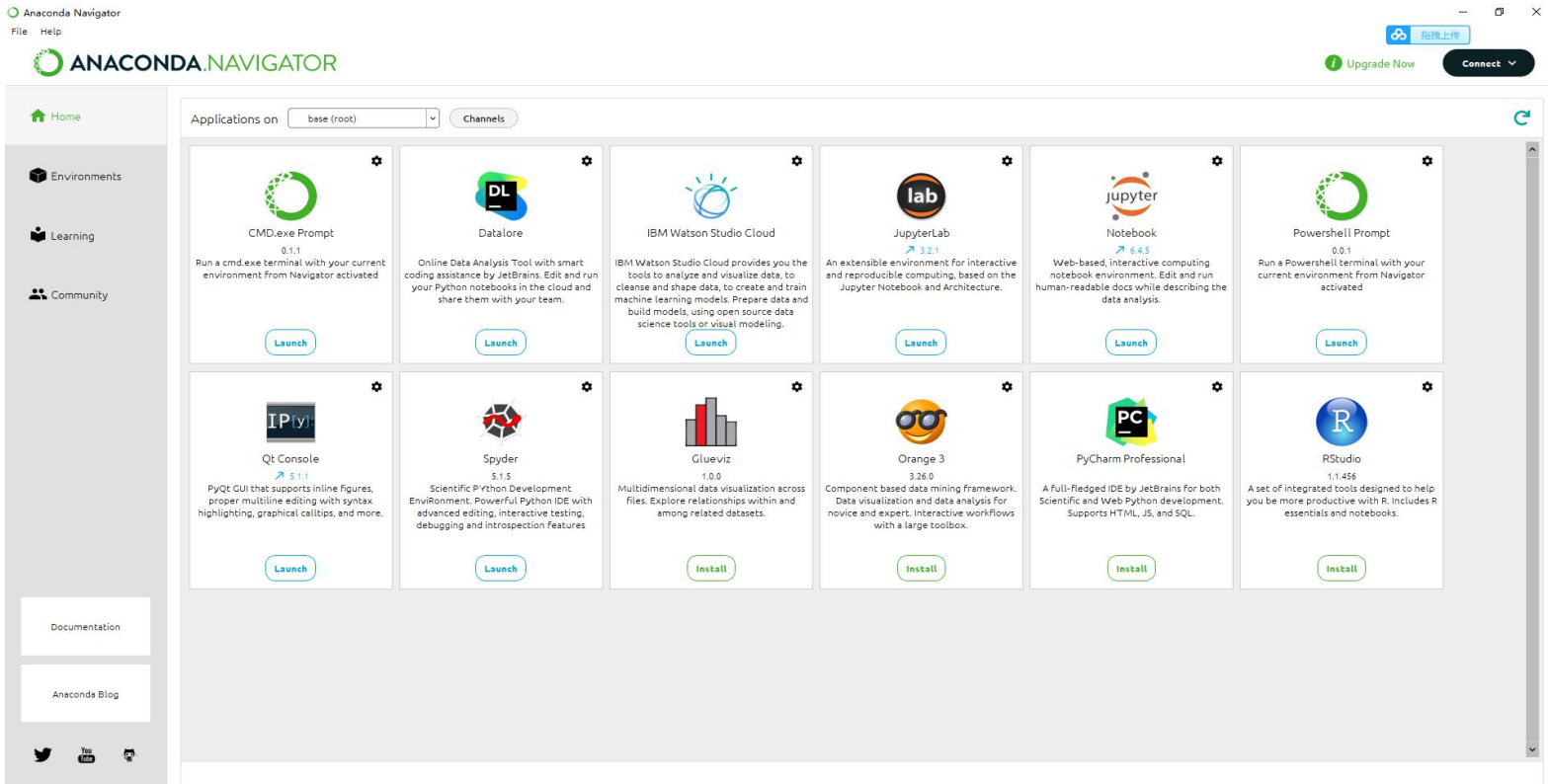
Jupyter
开发环境

14

Jupyter

jupyter notebook

安装Anaconda

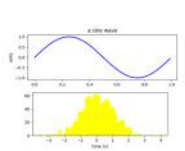


15

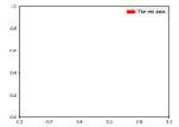
SciPy, NumPy, and Matplotlib Python 库

SciPy, NumPy, and Matplotlib

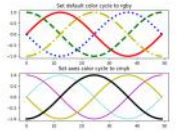
- SciPy is a mathematic
  - array objects
  - linear algebra
  - random numbers
  - https://scipy.org/
- NumPy
- Matplotlib
  - https://matplotlib.org/stable/tutorials/index



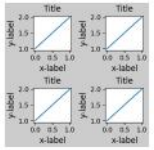
Artist tutorial



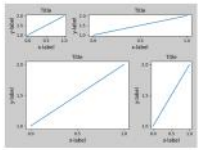
Legend guide



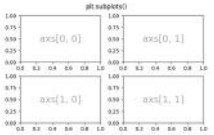
Styling with cycler



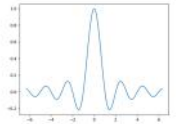
Constrained Layout Guide



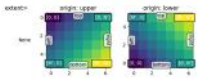
Tight Layout guide



Arranging multiple Axes in a Figure



Autoscaling



origin and extent in imshow

SciPy, NumPy, and Matplotlib

- numpy.array implements a one or more dimensional array

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

<class 'numpy.ndarray'>


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

SciPy, NumPy, and Matplotlib

- numpy.array implements a one or more dimensional array

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

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

Create multidimensional arrays with nested brackets:

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

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

SciPy, NumPy, and Matplotlib

- numpy.array implements a one or more dimensional array

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

[1. 2. 3.]
```

You can access the array elements using subscript location:

```
In [7]: x = np.array([[1, 2, 3],
                      [4, 5, 6]])

        print(x[1,2])
```

SciPy, NumPy, and Matplotlib

- numpy.array implements a one or more dimensional array

You can access a column or row by using slices. A colon (:) used as a subscript is shorthand for all data in that row or column. So `x[:,0]` returns an array of all data in the first column (the 0 specifies the first column):

```
In [8]: x[:, 0]

Out[8]: array([1, 4])
```

We can get the second row with:

We can get the second row with:

```
In [9]: x[1, :]

Out[9]: array([4, 5, 6])
```

SciPy, NumPy, and Matplotlib

Get the last two elements of the second row with:

```
In [10]: x[1, 1:]

Out[10]: array([5, 6])
```

As with Python `lists`, you can use negative indexes to refer to the end of the array. -1 refers to the last index. So another way to get the last two elements of the second (last) row would be:

```
In [11]: x[-1, -2:]

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

SciPy, NumPy, and Matplotlib

You can perform matrix addition with the + operator, but matrix multiplication requires the `dot` method or function. The * operator performs element-wise multiplication, which is **not** what you want for linear algebra.

```
In [12]: 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.]]
```

SciPy, NumPy, and Matplotlib

- Python 3.5 introduced the `@` operator for matrix multiplication.

```
In [13]: x @ x

Out[13]: array([[ 7., 10.],
                [15., 22.]])
```

- transpose and the inverse

```
In [14]: 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]]
```

SciPy, NumPy, and Matplotlib

- zeros matrix , ones matrix, eye identity matrix

```
In [15]: 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.]]
```

SciPy, NumPy, and Matplotlib

```
In [16]: np.arange(0, 2, 0.1)

Out[16]: 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])
```

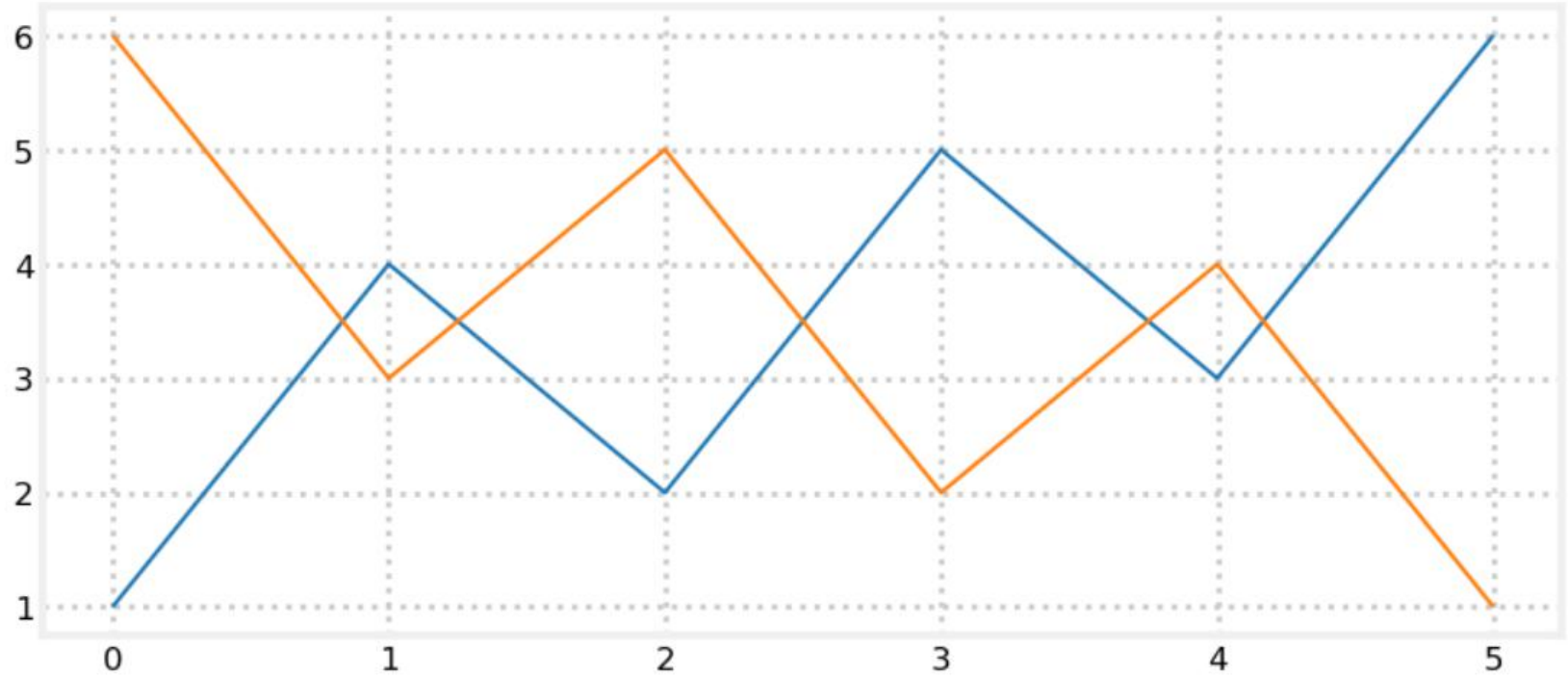
```
In [17]: np.linspace(0, 2, 20)

Out[17]: 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.   ])
```

SciPy, NumPy, and Matplotlib

```
In [18]: 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[18]: [<matplotlib.lines.Line2D at 0x298bcc7b4c8>]
```




SUNSPRING