Python学习笔记

❀ 赋值

```
x,y,z=5,10,15
print(x+y+z)
```

❀ 在函数外部创建的变量称为**全局变量**。全局变量可以被函数内部和外部的每个人使用。

```
x = "awesome"
def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()
print("Python is " + x)
```

注: myfun 函数中的 x = "fantastic", 而 print 中使用的 x = "awesome"。此外，`:后面的内容要缩进`

NumPy库

- ✓ NumPy 由 Travis Oliphant 于 2005 年创建， 指的是数值 Python（Numerical Python）。
- ✓ NumPy 是用于处理数组的 python 库，它还拥有在线性代数、傅立叶变换和矩阵领域中工作的函数。
- ✓ NumPy 旨在提供一个比传统 Python 列表快 50 倍的数组对象, 其中的数组对象称为 `ndarray`，它提供了许多支持函数，使得利用 `ndarray` 非常容易。

❀ NumPy 通常以 `np` 别名导入:

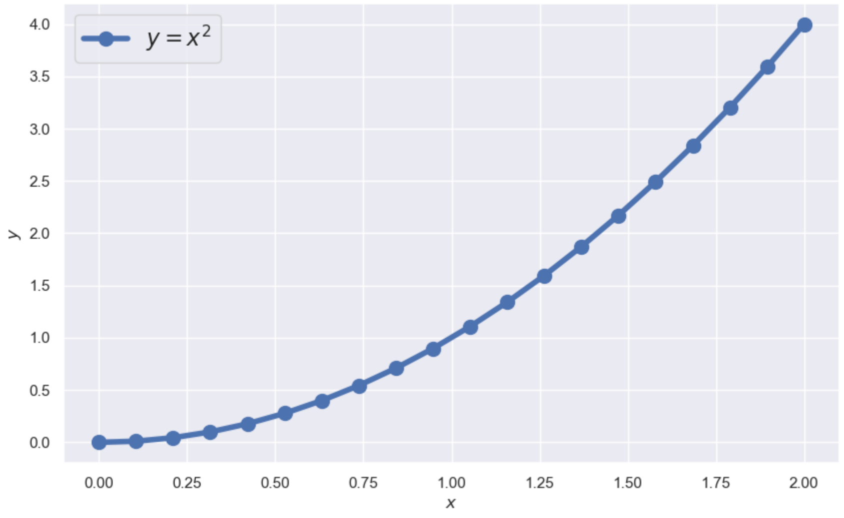
```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

不确定性量化

1.课程简介

❀ 绘图示例

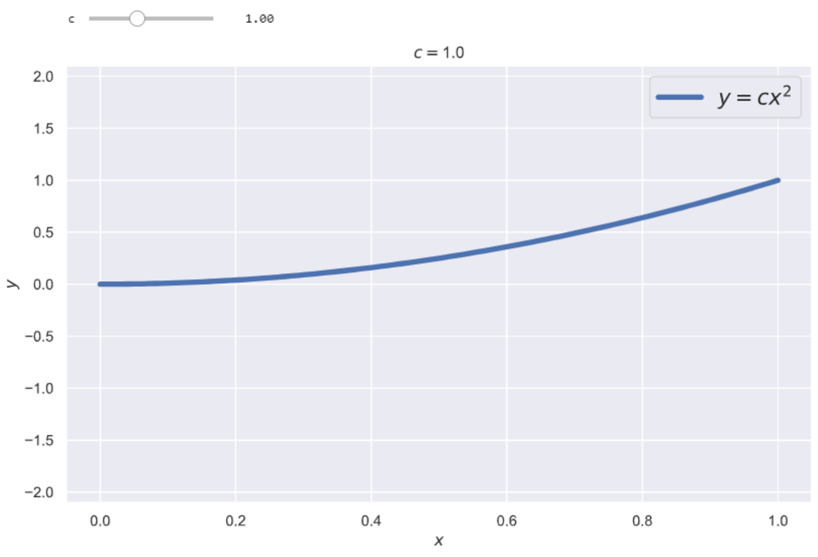
```
##import some libraries
import numpy as np   #matrix and linear algebra
import matplotlib.pyplot as plt   #plotting library
import seaborn as sns     #Make your plot look better than vanilla matplotlib
sns.set()   #use default seaborn settings.

#define a simple function
def f(x):
    return x ** 2

#generate some data
x = np.linspace(0, 2, 20)
y = f(x)

#plot the data and visualize it in the notebook
plt.figure(figsize=(10, 6))
plt.plot(x, y, label = '$y = x^2$', linewidth = 4, marker = 'o', markersize =
10)
plt.xlabel('$x$', fontsize = 12)
plt.ylabel('$y$', fontsize = 12)
plt.legend(loc = 'best', fontsize = 16)
```



❀ 交互式绘图 (似乎只能在 anaconda[`.ipynb`文件] 或者 jupyter 上实现)

```
#import library for interactive plotting
import ipywidgets
from ipywidgets import interactive
import numpy as np   #matrix and linear algebra
import matplotlib.pyplot as plt   #plotting library
import seaborn as sns     #Make your plot look better than vanilla matplotlib
sns.set()   #use default seaborn settings.

def f(x, c):
    return c* (x**2)
```

```
def plot_f(c=1):
    x = list(np.linspace(0, 1, 100))
    y = [f(i, c) for i in x]
    plt.figure(figsize=(10, 6))
    plt.plot(x, y, label = '$y = cx^2$', linewidth = 4)
    plt.title('$c = $'+str(c), fontsize = 12)
    plt.legend(loc = 'best', fontsize = 16)
    plt.xlabel('$x$', fontsize = 12)
    plt.ylabel('$y$', fontsize = 12)
    plt.ylim(-2.1,2.1)

interactive(plot_f, c = (-2, 6, 0.1))
```



❀ 课程所需的一些模块

```
# Remove the '#' from the lines below if you are running on Google Colab
#!pip install GPy
#!pip install pymc3
import GPy
import pymc3
from tensorflow import keras #Not required tensorflow.keras when using functions
keras
```

2.预测建模简介

本节介绍 `structural causal models` 及其 `graphical representation` 。 结构因果模型包含：

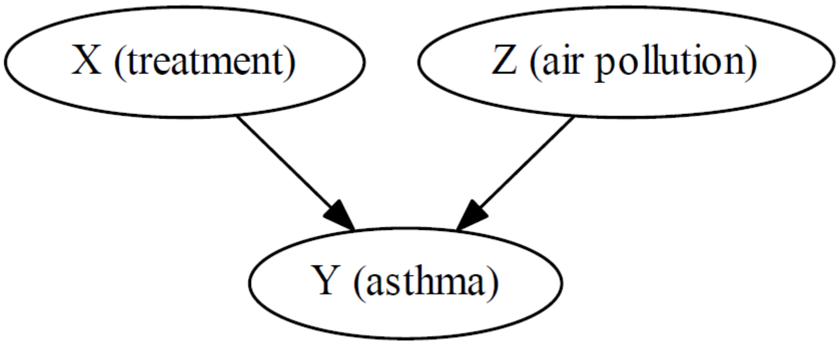
1. 变量集, 即模型试图解释的变量（内生的），也可能是其他可能需要的变量（外生的）。
2. 函数集, 每个变量可基于所有其他变量的值得到。

✸ **Example** : Asthma(哮喘) model - Graphical causal model

Here I am representing each variable with a node. The node at the beginning on an arrow is the direct cause of the node at the end of the arrow.

```
# Import the things we need to plot
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_context('talk')

from graphviz import Digraph # TODO add pygraphviz to dependencies
```

```
g = Digraph('Asthma')
g.node('X', label='X (treatment)')
g.node('Y', label='Y (asthma)')
g.node('Z', label='Z (air pollution)')
g.edge('X','Y')
g.edge('Z', 'Y')
#g.render('asthma_graph', format='png') # Uncomment the line if you want to save
the figure

g.view()
```



🐘 *注*: 关于生成有向图的命令可参考

不确定性分类

Uncertainty may be *aleatory* (偶然的) or *epistemic* (认知的).

- Aleatory uncertainty is associated with inherent system randomness.
- Epistemic uncertainty is associated with lack of knowledge.

🌟 **Example** : 在不平的路上开拖车（1/3）

假设我们是拖车的生产商，这个模型的所有参数有

Variable	Type	Values
k	Manufacturing uncertainty	$[159999, 160001]$ N/m
v	Operating condition	$[80, 150]$ km/hour
m	Loading condition	$[100, 200]$ kg
y_0	Road condition	$[0, 100]$ mm
L	Road condition	$[1, 2]$ m

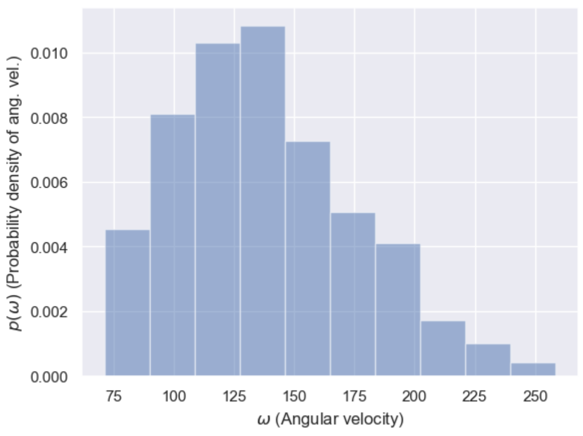
Not being able to come up with more precise information (or any data) we would consider any value within this intervals as equally likely.(即理解为均匀分布)

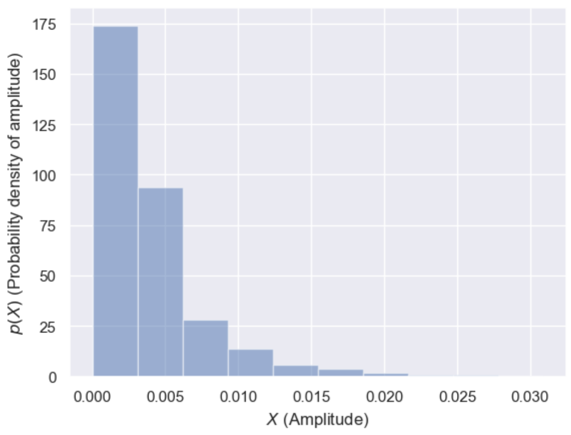
Now, let's write some code to see how this uncertainty affects the angular velocity(角速度) ω and the amplitude(振幅) X.

```
# Import the things we need to plot
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_context('talk')  # sns.set()
```

```
# for numerical arrays and linear algebra:
import numpy as np
# The number of samples we wish to take
num_samples = 1000
# Two arrays in which to store the samples we take
Xs = np.ndarray((num_samples, )) # To store the samples
omegas = np.ndarray((num_samples, ))
for i in range(num_samples):
    k = 160000.0 + np.random.rand() # np.random.rand() samples a number
uniformly between 0 and 1
    m = 100.0 + (200.0 - 100.0) * np.random.rand() # Here we sample a random
number in [100, 200]
    y0 = 100 * np.random.rand() * 1e-3 # Turning it to m
    v = (80.0 + (150.0 - 80.0) * np.random.rand()) * 1e3 / 3600.0 # Turning it
to m/s
    lam = 1.0 + (2.0 - 1.0) * np.random.rand()
    omega = 2.0 * np.pi * v / lam
    X = np.abs(k * y0 / (k - m * omega ** 2))
    omegas[i] = omega
    Xs[i] = X

# Plot the angular velocity
fig, ax = plt.subplots() #第一个是figure图片对象，第二个是轴axes对象或者是轴对象组成的数
组。两个参数是装在tuple里返回的，所以直接用两个变量fig和ax分别接收了
ax.hist(omegas, bins=10, alpha=0.5, density=True)
ax.set_xlabel('$\omega$ (Angular velocity)')
ax.set_ylabel('$p(\omega)$ (Probability density of ang. vel.)')

# Plot the amplitude
fig, ax = plt.subplots()
ax.hist(Xs, bins=10, alpha=0.5, density=True)
ax.set_xlabel('$X$ (Amplitude)')
ax.set_ylabel('$p(X)$ (Probability density of amplitude)');
```

因此角速度 $\omega = \frac{2\pi v}{L}$，其中 $v \sim U(\frac{80}{3.6}, \frac{150}{3.6}), L \sim U(1,2)$，此外振幅 $X = |\frac{ky_0}{k-m\omega^2}|$.





Questions

- What does the probability density in the figures above represent? Is the uncertainty aleatory or epistemic? (推导相关概率分布, 判断不确定性来源)
- Rerun(重新运行) the code above giving different values to `num_samples`. Can you trust the results when you pick small values?(小样本结果是否可信) How can you pick the right value for `num_samples`? (如何选取样本量)

***The model calibration problem* 模型校准**

The model calibration problem is the inverse of the uncertainty propagation problem. (不确定传播的反问题) That is why such problems are also called **inverse problems**. It goes as follows. One observes a quantity that is predicted by the model and they want to go back and characterize how this observation changes the state of knowledge about the parameters of the model.(反推模型参数)

✸ **Example** : 在不平的路上开拖车（2/3）

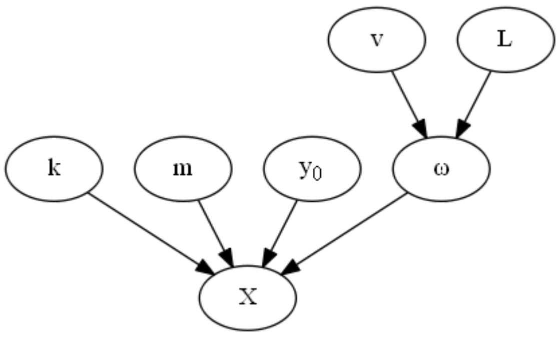
In this example, imagine that we put sensors on the suspension of the trailer to measure the amplitude of oscillation(悬挂拖车上的传感器来测量振动的振幅). Now, the amplitude we measure is not exactly the prediction of the model. Why is that?

- First, there is observation noise.
- Second, our model has a systematic error (e.g., we have completely ignored any dumping effects).

So, *the measurement is not the same as the model prediction*. This means that we need to add one more node to the graphical representation of the model. The new node, let's call it is the result of the measurement.

```
g = Digraph('Trailer')
g.node('k')
g.node('m')
g.node('y0', label='<y<sub>0</sub>>')
g.node('omega', label='<&omega;>')
g.node('v')
g.node('L')
g.node('X')
g.edge('v', 'omega')
g.edge('L', 'omega')
g.edge('y0', 'X')
g.edge('omega', 'X')
g.edge('k', 'X')
g.edge('m', 'X')
```

```
g.node('Xm', label='<X<sub>m</sub>>', style='filled')
g.edge('X', 'Xm')
g.render('trailer_m_g', format='png')
```



Solving inverse problems

We will need a couple of lectures to understand what is the right way to pose and solve the problem.

But here is the answer:

- Quantify our **prior** state of knowledge about all the model parameters (by assigning probability densities to them).
- Use Bayes' rule to condition the prior knowledge on the observations. This updated knowledge is our **posterior knowledge**. (后验分布) Unfortunately, this posterior knowledge is rarely analytically available. This is why we need the third step.
- Create a practical procedure that characterizes our posterior state of knowledge.

The majority of the lectures of this class are about the third step.

Hands-on activity: Catalytic Conversion of Nitrate(硝酸) to Nitrogen(氮气)

Consider the catalytic conversion of nitrate (NO_3^-) to nitrogen (N_2) and other by-products by electrochemical means. The mechanism that is followed is complex and not well understood.

The experiment of (Katsounaros, 2012) confirmed the production of nitrogen (N_2), ammonia (NH_3), and nitrous oxide $(\mathrm{N}_2\mathrm{O})$ as final products of the reaction, as well as the intermediate production of nitrite (NO_2^-).

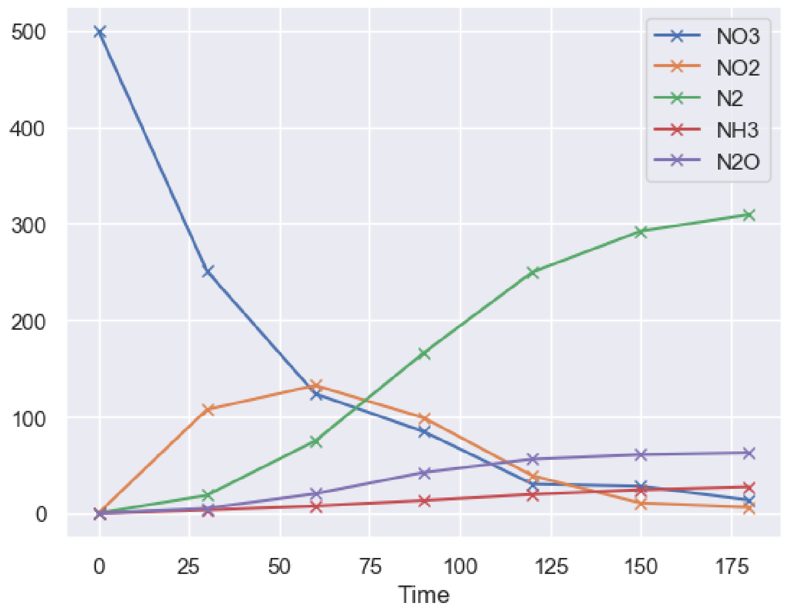
The data are reproduced in Comma-separated values (CSV) and stored in data/catalysis.csv. The time is measured in minutes and the conentrations are measured in $\mathrm{mmol}\cdot\mathrm{L}^{-1}$.

Let's load the data into this notebook using the Pandas Python module:

```
import pandas as pd
import io
import requests
url="https://raw.githubusercontent.com/PredictiveScienceLab/uq-
course/master/lectures/catalysis.csv"
s=requests.get(url).content
catalysis_data = pd.read_csv(io.StringIO(s.decode('utf-8')))
print(catalysis_data)
```

```
   Time     NO3     NO2     N2   NH3   N2O
0     0  500.00    0.00   0.00  0.00  0.00
1    30  250.95  107.32  18.51  3.33  4.98
```

```
2    60  123.66  132.33   74.85   7.34  20.14
3    90   84.47   98.81  166.19  13.14  42.10
4   120   30.24   38.74  249.78  19.54  55.98
5   150   27.94   10.42  292.32  24.07  60.65
6   180   13.54    6.11  309.50  27.26  62.54

import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
catalysis_data.plot(x='Time', style='-x')   #visualize the data using Matplotlib
plt.show()
```



The theory of catalytic reactions guarantees that the total mass must be conserved.(催化反应总质量守恒) However, this is not the case in our dataset:

```
catalysis_data.sum(axis=1)

0    500.00
1    415.09
2    418.32
3    494.71
4    514.28
5    565.40
6    598.95
dtype: float64
```

This inconsistency suggests the existence of an intermediate unobserved reaction product X. The dynamical system associated with the reaction is:

$$
\begin{aligned}
\frac{d\left[\mathrm{NO}_3^-\right]}{dt} &= -k_1\left[\mathrm{NO}_3^-\right],\\
\frac{d\left[\mathrm{NO}_2^-\right]}{dt} &= k_1\left[\mathrm{NO}_3^-\right] - (k_2+k_4+k_5)[\mathrm{NO}_2^-],\\
\frac{d\left[\mathrm{X}\right]}{dt} &= k_2\left[\mathrm{NO}_2^-\right] - k_3[X],\\
\frac{d\left[\mathrm{N}_2\right]}{dt} &= k_3\left[\mathrm{X}\right],\\
\frac{d\left[\mathrm{NH}_3\right]}{dt} &= k_4\left[\mathrm{NO}_2^-\right],\\
\frac{d\left[\mathrm{N}_2O\right]}{dt} &= k_5\left[\mathrm{NO}_2^-\right],
\end{aligned}
$$

where $[\cdot]$ denotes the concentration of a quantity, and $k_i > 0$, $i = 1, \ldots 5$ are the *kinetic rate constants*.

Computational Model

We will develop a generic computational model for the solution of dynamical systems and we will use it to study the catalysis problem. The code relies on the Fourth-order Runge-Kutta method and is a modified copy of http://www.math-cs.gordon.edu/courses/ma342/python/diffeq.py developed by Jonathan Senning. The code solves:

$$
\begin{aligned}
\dot{\mathbf{y}} &= f(\mathbf{y}, t),\\
\mathbf{y}(0) &= \mathbf{y}_0.
\end{aligned}
$$

```
import numpy as np
def rk45( f, y0, t, args=() ):
    """Fourth-order Runge-Kutta method with error estimate.

    USAGE:
        y = rk45(f, x0, t, args=())

    INPUT:
        f     - function of x and t equal to dx/dt.  x may be multivalued,
                in which case it should a list or a NumPy array.  In this
                case f must return a NumPy array with the same dimension
                as x.
        y0    - the initial condition(s).  Specifies the value of x when
                t = t[0].  Can be either a scalar or a list or NumPy array
                if a system of equations is being solved.
        t     - list or NumPy array of t values to compute solution at.
                t[0] is the the initial condition point, and the difference
                h=t[i+1]-t[i] determines the step size h.
        args  - any other parameters of the function f.

    OUTPUT:
        y     - NumPy array containing solution values corresponding to each
                entry in t array.  If a system is being solved, x will be
                an array of arrays.

    NOTES:
        This version is based on the algorithm presented in "Numerical
        Mathematics and Computing" 6th Edition, by Cheney and Kincaid,
        Brooks-Cole, 2008.
    """
    # Coefficients used to compute the independent variable argument of f

    c20  =   2.500000000000000e-01  #  1/4
    c30  =   3.750000000000000e-01  #  3/8
    c40  =   9.230769230769231e-01  #  12/13
    c50  =   1.000000000000000e+00  #  1
    c60  =   5.000000000000000e-01  #  1/2

    # Coefficients used to compute the dependent variable argument of f

    c21 =   2.500000000000000e-01  #  1/4
    c31 =   9.375000000000000e-02  #  3/32
    c32 =   2.812500000000000e-01  #  9/32
    c41 =   8.793809740555303e-01  #  1932/2197
    c42 =  -3.277196176604461e+00  # -7200/2197
    c43 =   3.320892125625853e+00  #  7296/2197
    c51 =   2.032407407407407e+00  #  439/216
    c52 =  -8.000000000000000e+00  # -8
    c53 =   7.173489278752436e+00  #  3680/513
    c54 =  -2.058966861598441e-01  # -845/4104
    c61 =  -2.962962962962963e-01  # -8/27
    c62 =   2.000000000000000e+00  #  2
    c63 =  -1.381676413255361e+00  # -3544/2565
    c64 =   4.529727095516569e-01  #  1859/4104
    c65 =  -2.750000000000000e-01  # -11/40

    # Coefficients used to compute 4th order RK estimate

    a1  =   1.157407407407407e-01  #  25/216
    a2  =   0.000000000000000e-00  #  0
    a3  =   5.489278752436647e-01  #  1408/2565
    a4  =   5.353313840155945e-01  #  2197/4104
    a5  =  -2.000000000000000e-01  # -1/5

    b1  =   1.185185185185185e-01  #  16.0/135.0
    b2  =   0.000000000000000e-00  #  0
    b3  =   5.189863547758284e-01  #  6656.0/12825.0
    b4  =   5.061314903420167e-01  #  28561.0/56430.0
    b5  =  -1.800000000000000e-01  # -9.0/50.0
    b6  =   3.636363636363636e-02  #  2.0/55.0

    n = len( t )
    y = np.array( [ y0 ] * n )
    for i in range( n - 1 ):
        h = t[i+1] - t[i]
        k1 = h * f( y[i], t[i], *args )
        k2 = h * f( y[i] + c21 * k1, t[i] + c20 * h, *args )
        k3 = h * f( y[i] + c31 * k1 + c32 * k2, t[i] + c30 * h, *args )
        k4 = h * f( y[i] + c41 * k1 + c42 * k2 + c43 * k3, t[i] + c40 * h, *args
)
        k5 = h * f( y[i] + c51 * k1 + c52 * k2 + c53 * k3 + c54 * k4, \
                        t[i] + h, *args )
        k6 = h * f( y[i] + c61 * k1 + c62 * k2 + c63 * k3 + c64 * k4 + c65 * k5,
\
            t[i] + c60 * h, *args )

        y[i+1] = y[i] + a1 * k1 + a3 * k3 + a4 * k4 + a5 * k5
        y5 = y[i] + b1 * k1 + b3 * k3 + b4 * k4 + b5 * k5 + b6 * k6

    return y
```

Calibrating the Catalysis Model to the Experimental Data(通过实验数据校准参数)

Now that we are certain that our generic ODE solver works, let us use it to develop a solver for the catalysis model. All, we need to do is define the right hand side of the dynamics:

```
def f_catalysis(y, t, kappa):
    rhs = np.zeros((6,))
    rhs[0] = -kappa[0] * y[0]
    rhs[1] = kappa[0] * y[0] - (kappa[1] + kappa[3] + kappa[4]) * y[1]
    rhs[2] = kappa[1] * y[1] - kappa[2] * y[2]
    rhs[3] = kappa[2] * y[2]
    rhs[4] = kappa[3] * y[1]
    rhs[5] = kappa[4] * y[1]
    return rhs
```

Let's try to calibrate the parameters of the model to the data, manually. Because the parameters are too small, let us work with the transformed version:

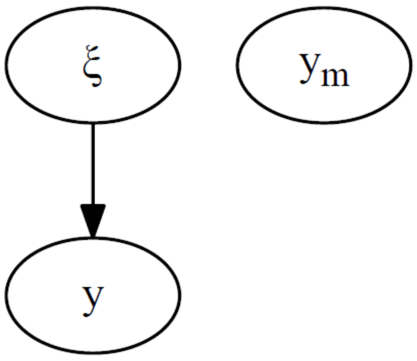
$$\xi_i = \log(180k_i).$$

Also, let's draw the graph corresponding to this model. We have the following variables:

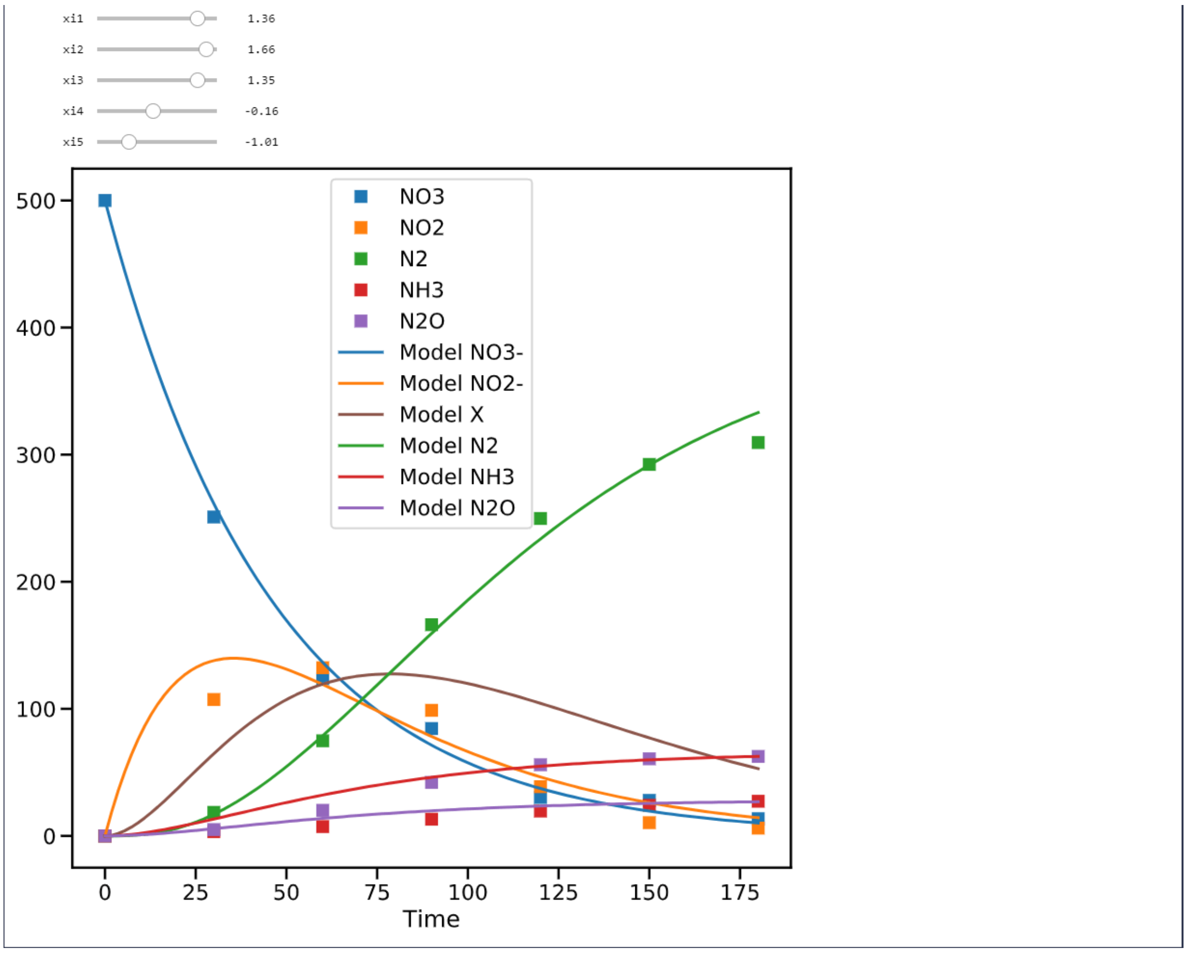
- ξ corresponding to the scaled unknown parameters
- y which is the prediction of our model at all timesteps for which we have data.
- y_m which are the measured data.

The graph will look as follows:

```
gc = Digraph('Catalysis')
gc.node('xi', label='<&xi;>')
gc.node('y')
gc.node('ym', label='<y<sub>m</sub>>')
gc.edge('xi', 'y')
gc.view()
```



```
from ipywidgets import interactive
def compare_model_to_data(xi1 = 1.359, xi2 = 1.657, xi3 = 1.347, xi4 = -.162,
xi5 = -1.009):
    """
    Compare the model predictions to the data.
    """
    t = np.linspace(0, 180, 100)
    kappa = np.exp([xi1, xi2, xi3, xi4, xi5]) / 180.
    y = rk45(f_catalysis, (500., 0., 0., 0., 0., 0.), t, args=(kappa,))
    fig, ax = plt.subplots(figsize=(10, 10))
    catalysis_data.plot(x='Time', ax=ax, style='s')
    ax.plot(t, y[:, 0], color=sns.color_palette()[0], label='Model NO3-')
    ax.plot(t, y[:, 1], color=sns.color_palette()[1], label='Model NO2-')
    ax.plot(t, y[:, 2], color=sns.color_palette()[5], label='Model X')
    ax.plot(t, y[:, 3], color=sns.color_palette()[2], label='Model N2')
    ax.plot(t, y[:, 4], color=sns.color_palette()[3], label='Model NH3')
    ax.plot(t, y[:, 5], color=sns.color_palette()[4], label='Model N2O')
    plt.legend()

interactive(compare_model_to_data, xi1 = (-2, 2, 0.05), xi2 = (-2, 2, 0.05), xi3
= (-2, 2, 0.05), xi4 = (-2, 2, 0.05), xi5 = (-2, 2, 0.05) )
```



3.概率导论

The probability mass function (概率质量函数,pmf)

对于概率空间 $(\Omega, \mathcal{F}, \mathbb{P})$ 上的离散随机变量 X , 不失一般性, 假设 X取值与 $\mathbb{N} = \{1, 2, \dots\}$. 相应的概率质量函数记为 $f_X(x)$. 数学上的定义为:

$$f_X(x) := \mathbb{P}(X = x) = \mathbb{P}\left(\{\omega : X(\omega) = x\}\right).$$

If you are 100% sure about which random variable you are talking about, feel free to use the much simpler notation:

$$p(x) \equiv p(X = x) \equiv f_X(x) = \mathbb{P}\left(\{\omega : X(\omega) = x\}\right).$$

This is the notation we will employ from this point on. We will only use the strict mathematical notation when we have no choice.

🐘 *注*: 如果想体现 background information , 我们会记 pmf 为 $p(x|I)$.

🌲 **Properties of the probability mass function** There are some standard properties of the probability mass function that is worth memorizing:

- The probability mass function is nonnegative:

$$p(x) \geq 0,$$

for all x in $\mathbb{N}$.

- The probability mass function is normalized:

$$\sum_{x=0}^{\infty} p(x) = 1.$$

This is a direct consequence of the fact that X must take a value.

- Take any set of possible values of X, A. The probability of X taking values in A is:

$$p(X \in A) = \sum_{x \in A} p(x).$$

🍁 **Example: The Bernoulli random variable (1/2)**

Bernoulli random variable generalizes the concept of a coin toss. You can think of it as the result of an experiment with two possible outcomes 0 and 1. One just needs to specify the probability of one of the outcomes, typically the probability of zero. So, how do we denote mathematically a Bernoulli random variable X that takes the value 1 with probability θ in $[0, 1]$? We can write:

$$X = \begin{cases} 1, & \text{with probability } \theta, \\ 0, & \text{otherwise.} \end{cases}$$

The other way we can write this is as follows:

$$X \sim \text{Bernoulli}(\theta).$$

The expectation of the Bernoulli is:

$$\mathbb{E}[X] = \sum_x x p(X = x) = 0 \cdot (1 - \theta) + 1 \cdot \theta = \theta.$$

Similarly, the variance of the Bernoulli is:

$$\mathbb{V}[X] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2 = \theta - \theta^2 = \theta(1 - \theta).$$

☘ use the functionality(功能) of `scipy.stats` to define a Bernoulli random variable and sample from it.

```
# Import the scipy.stats library
import scipy.stats as st
# This is the probability of 1:
theta = 0.6
# Define the random variable, Bernoulli(theta)
```

```
X = st.bernoulli(theta)
# Here is the **support** of the random variable. It tells you which variables
it takes:
print('X takes values in', X.support())

X takes values in (0, 1)

# Evaluate the cumulative distribution function at every point of the support
for x in X.support():
    print('p(X={0:d}) = {1:1.2f}'.format(x, X.cdf(x)))

p(X=0) = 0.40
p(X=1) = 1.00

# Sample the random variable 100 times:
xs = X.rvs(100)
print(xs)

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

# The expectation of the Bernoulli:
print('E[X] = {0:1.2f}'.format(X.expect()))
E[X] = 0.60

# The variance of the Bernoulli:
print('V[X] = {0:1.2f}'.format(X.var()))
V[X] = 0.24
```

🐘 *注*: 一种格式化字符串的方法是使用字符串的`format()`方法，它会用传入的参数依次替换字符串内的占位符`{0}`、`{1}`……，不过这种方式写起来比%要麻烦得多：

```
print('Hello, {0}, 成绩提升了 {1:.1f}%'.format('小明', 17.125))

        'Hello, 小明, 成绩提升了 17.1%'
```

Joint probability mass function of random variables

Consider two random variables X and Y. The *joint probability mass function* of the pair (X, Y) is the function $f_{X,Y}(x, y)$ giving the probability that $X = x$ and $Y = y$. Mathematically (and introducing a simplified notation), we have:

$$p(x, y) \equiv p(X = x, Y = y) \equiv f_{X,Y}(x, y) := \mathbb{P}(\{\omega : X(\omega) = x, Y(\omega) = y\}).$$

🌲 **Properties of the joint probability mass function**

- It is nonnegative:

$$p(x, y) \geq 0.$$

- If you sum over all the possible values of all random variables, you should get one:

$$\sum_x \sum_y p(x, y) = 1.$$

- If you *marginalize* over the values of one of the random variables you get the pmf of the other. For example:

$$p(x) = \sum_y p(x,y),$$

and

$$p(y) = \sum_x p(x,y).$$

The covariance operator

The covariance operator measures how correlated two random variables X and Y are. Its definition is:

$$\mathbb{C}[X,Y] = \mathbb{E}\left[(X-\mathbb{E}[X])\,(Y-\mathbb{E}[Y])\right].$$

If $\mathbb{C}[X,Y]$ is positive, then we say that the two random variables are correlated. If it is negative, then we say that the two random variables are anti-correlated. If it is zero, then we say that the two random variables are not correlated.

A usefull property of the covariance operator is that it can give tell you something about the variance of the sum of two random variables. It is:

$$\mathbb{V}[X+Y] = \mathbb{V}[X] + \mathbb{V}[Y] + 2\mathbb{C}[X,Y].$$

Joint probability mass function of many random variables . Take N random variables $X_1,\dots,X_N$. We can define their joint probability mass function in the same way we did it for two:

$$p(x_1,\dots,x_N) \equiv p(X_1=x_1,\dots,X_N=x_N) \equiv f_{X_1,\dots,X_N}(x_1,\dots,X_N) := \mathbb{P}\left(\{\omega : X_1(\omega)=x_1,\dots,X_N(\omega)=x_N\}\right).$$

Just like before, we can marginalize over any subset of random variables to get the pmf of the remaining ones. For example:

$$p(x_i) = \sum_{x_j, j\neq i} p(x_1,\dots,x_N).$$

Example: The binomial distribution

Suppose that you tossing n times a coin with probability of heads θ and let X be the number of heads. The random variable X is called the binomial random variable. We write:

$$X \sim B(n,\theta).$$

It is easy to show that its pmf is:

$$p(X=k) = \binom{n}{k}\theta^k(1-\theta)^{n-k},$$

where $\binom{n}{k}$ is the number of k combinations out of n elements:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

The expectation of the binomial is:

$$\mathbb{E}[X] = \sum_{k=0}^n p(X=k) = \sum_{k=0}^n \binom{n}{k}\theta^k(1-\theta)^{n-k} = ?.$$

It does seem very easy to carry out this sum. However, you can do something smarter. Notice that X counts the number of heads in n independent trials. Let's introduce the *independent* random variables $X_1,\dots,X_n$ corresponding to the result of these trials. All these variables are:

$$X_i \sim \text{Bernoulli}(\theta).$$

The number of heads is simply:

$$X = X_1 + \dots X_n.$$

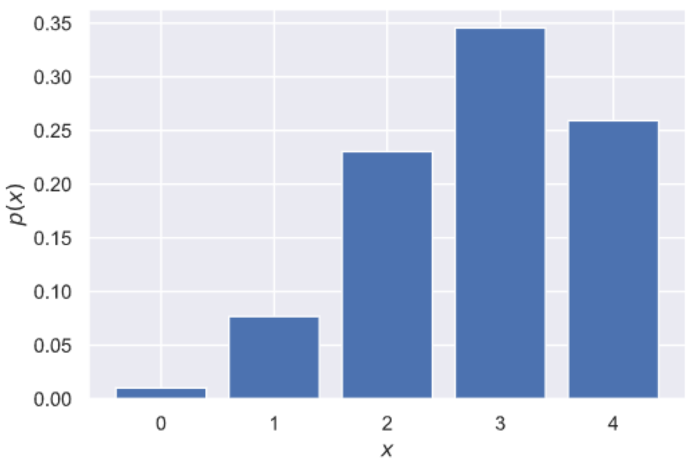
Since all the random variables on the right hand-side are independent, we get:

$$\mathbb{E}[X] = \mathbb{E}[X_1] + \dots + \mathbb{E}[X_n] = \theta + \dots + \theta = n\theta.$$

Similarly, we can find the variance of X:

$$\mathbb{V}[X] = \mathbb{V}[X_1] + \dots + \mathbb{V}[X_n] = n\theta(1-\theta).$$

```
# Let's draw histograms of the binomial
n = 5
theta = 0.6
X = st.binom(n, theta)
# Here are some samples
print(X.rvs(100))
[3 3 2 3 3 3 3 4 4 1 4 3 2 2 4 4 3 4 2 2 2 4 3 3 4 3 3 4 4 1 2 2 1 1 4 4 4
 3 4 4 3 2 3 3 3 3 1 4 4 2 3 3 4 3 4 5 3 4 1 2 1 4 2 3 2 2 3 1 4 3 2 2 3 3
 4 4 3 1 5 0 2 1 4 3 3 3 5 3 3 2 1 2 4 4 3 3 4 3 2 1]

# Here is the expectation
print('E[X] = {0:1.2f}'.format(X.expect()))
E[X] = 3.00

# Here is the variance
print('V[X] = {0:1.2f}'.format(X.var()))
V[X] = 1.20

# Let's draw the pmf
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
xs = range(n)     # range(star,stop,step);range(n)=range(1,n-1,1)
ax.bar(xs, X.pmf(xs))    ##ax.plot()
ax.set_xlabel('$x$')
ax.set_ylabel('$p(x)$')
plt.show()
```



Something more on probability spaces

It turns out that when Ω is a continuous space, like $\mathbb{R}$ for example, it is not possible to take $\mathcal{F}$ to be all the subsets of Ω (you need to take a measure-theoretic probability theory class to understand why this is). However, many "nice" subsets of Ω are usually in $\mathcal{F}$. For example, in the case of $\Omega = \mathbb{R}$, $\mathcal{F}$ can include all intervals, and any countable unions and intersections of intervals. That's a lot of sets.

In any case, $\mathcal{F}$ must satisfy certain properties for everything to be well-defined. These properties are:

- $\Omega \in \mathcal{F}$
- For any A in $\mathcal{F}$, the complement A^c is in $\mathcal{F}$.
- For any $A_1, A_2, \dots$ in $\mathcal{F}$, the union $\cup_n A_n$ is in $\mathcal{F}$.

When a set of subsets $\mathcal{F}$ satisfies these properties, we say that it forms a σ-algebra.

The probability density function

The probability density function (PDF) is a "function" $f_X(x)$ that can give us the probability that X is in any "good" subset A of $\mathbb{R}$ as follows:

$$\mathbb{P}(X \in A) = \int_A f_X(x)dx.$$

Note that certain random variables may not have a PDF that is a function. That's why I put the word "function" in quotes. However, if you allow the PDF to include Dirac's δ, then any random variable has a PDF. We will ignore this complication for the moment.

In this class, we will simplify the notation and we will be writing:

$$p(x) \equiv f_X(x),$$

when there is no ambiguity.

🌲 **Properties of the probability density function**

- $p(x) \geq 0$ for all x.
- $\int_{-\infty}^{\infty} p(x)dx = 1$.
- The derivative of the CDF is the PDF, i.e., $F_X'(x) = p(x)$.

Dirac's delta and a unified view of all random variables**

Dirac's δ is a special function, actually called a distribution, which is defined as follows:

$$\delta(x) = 0,$$

for $x \neq 0$, and

$$\int_{-\infty}^{\infty} \delta(x)dx = 1$$

You can think of Dirac's δ as the PDF of a discrete random variable that **takes the value 0 with probability one**.

利用Dirac函数可将任一离散随机变量视为连续随机变量(相应的有概率密度函数). 例如, 分类型随机变量取值 $x_1, \dots, x_K$ with probabilities $p_1, \dots, p_K$. The PDF of this random variable can be written as:

$$p(x) = \sum_{k=1}^{K} p_k \delta(x - x_k).$$

PDF更一般的形式如下：

$$p(x) = f_X^n(x) + f_X^\delta(x),$$

a part $f_X^n(x)$ that is a nice proper function and a part $f_X^\delta(x)$ that consists of a weighted sum of Dirac δ's.

✸ Example: The uniform distribution

The uniform distribution is the most common continuous distribution. It corresponds to a random variable that is equally likely to take a value within a given interval. We write:

$$X \sim U([0,1]),$$

and we read X follows a uniform distribution taking values in $[0,1]$.

The probability density of the uniform is constant in $[0,1]$ and zero outside it. We have:

$$p(x) := U(x|[0,1]) := f_X(x) = \begin{cases} 1, & 0 \le x \le 1, \\ 0, & \text{otherwise.} \end{cases}$$

The cumulative distribution function of the uniform is:

$$F_X(x) = \mathbb{P}(X \le x) = \int_0^x f_X(u)du = \int_0^x du = x.$$

$$\mathbb{P}(a \le X \le b) = F_X(b) - F_X(a) = b - a.$$

The expectation of the uniform is:

$$\mathbb{E}[X] = \int_0^1 x dx = \frac{1}{2}.$$

The variance of the uniform is:

$$\mathbb{V}[X] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2 = \frac{1}{3} - \frac{1}{4} = \frac{1}{12}.$$

```python
# Let's demonstrate how you can sample from the uniform
import scipy.stats as st
X = st.uniform(3,10)
X.rvs(size=100)

# An alternative way is to use the functionality of numpy
np.random.rand(100)
```

✸ Example: Inferring the probability of a coin toss from data (1/3)

This is our first Bayesian inference example!

现在抛一枚硬币 N 次, 我们希望计算出正面朝上(success)的概率, 此时观测值为 $x_1, \dots, x_N$. For notational convenience we will be writing:

$$x_{1:N} := (x_1, \dots, x_N).$$

- [x] 首先, 记 the probability of success of the coin toss 为 θ. 为了描述对其的不确定性, 给参数 θ 施加(assign)一个先验概率分布, 由于只知道 θ 取值与0和1, 因此假定

$$\theta \sim U([0,1]).$$

- [x] 其次, N 次抛硬币结果对应与独立同分布的Bernoulli variable with the same probability of success θ. We write:

$$X_n|\theta \sim \text{Bernoulli}(\theta), \quad \text{for } n=1,\dots,N.$$

Note that these random variables depend on θ. That's why we are conditioning like this.

- [x] 最后, 为使用Bayesian inference, we need the joint probability density of all variables. It is:

$$\begin{aligned} p(x_1,\dots,x_N,\theta) = p(x_{1:N}|\theta)p(\theta) &= \left(\prod_{n=1} p(x_n|\theta)\right)p(\theta), \\ &= \theta^{\sum_{n=1}^N x_n}(1-\theta)^{N-\sum_{n=1}^N x_n}1_{[0,1]}(\theta) \end{aligned}$$

which has a nice interpretation as it depends only on the total number of heads $\sum_{n=1}^N x_n$.

Now, we are in a position to apply Bayes rule to condition on the data. We have:

$$p(\theta|x_{1:N}) = \frac{p(x_{1:N},\theta)}{p(x_{1:N})} \propto p(x_{1:N},\theta) = \theta^{\sum_{n=1}^N x_n}(1-\theta)^{N-\sum_{n=1}^N x_n}1_{[0,1]}(\theta).$$

因此, θ 的后验分布为 Beta distribution. 首先简单介绍 Beta 分布.

Example: The Beta distribution

The Beta distribution is suitable for random variables that take values in $[0,1]$ but are not necessarily uniform. We write:

$$X \sim \text{Beta}(\alpha,\beta),$$

where α and β are positive shape parameters. The interpretation of the parameters is more or less this:

- The bigger α is, the more the distribution is pulled towards zero.
- The bigger β is, the more the distribution is pulled towards one.

The PDF of the Beta is:

$$p(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)},$$

where

$$B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)},$$

where Γ is the Gamma function. It's expectation is:

$$\mathbb{E}[X] = \frac{\alpha}{\alpha+\beta}.$$

Example: Inferring the probability of a coin toss from data (2/2)

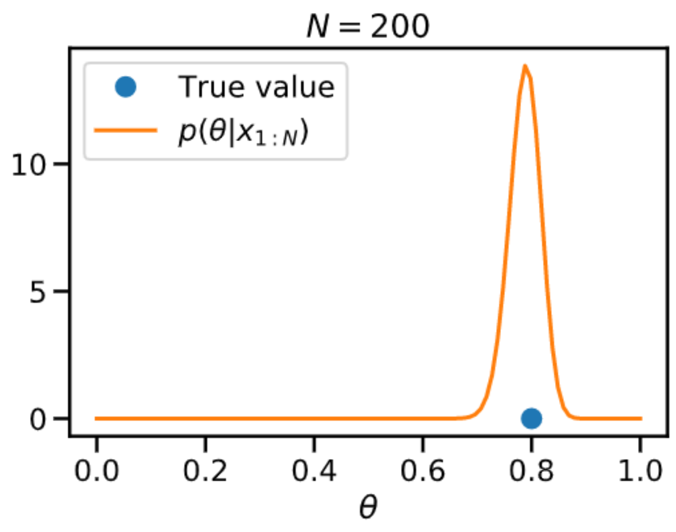
Now, that we know about the Beta distribution, we can write for the posterior of θ:

$$p(\theta|x_{1:N}) = \text{Beta}\left(\theta\middle|1+\sum_{n=1}^{N}x_n, 1+N-\sum_{n=1}^{N}x_n\right).$$

where with $\text{Beta}(\theta|\alpha,\beta)$ we mean the PDF of the $\text{Beta}(\alpha,\beta)$ evaluated at $\theta$ (this is a very useful notation). So, we see that the α parameter is just one plus the number of heads and the β parameter is one plus the number of tails.

❋ Let's try this out with some fake data. 假定抛硬币 success 的概率为 0.8, 据此生成200个观测值 $x_1,\dots,x_{200}$, 根据训练数据, 得到 $\theta$ 的后验密度.

```
# Take a fake coin which is a little bit biased
theta_true = 0.8
# This is the random variable corresponding to a coin toss
X = st.bernoulli(theta_true)
# Sample from it a number of times to generate our data = (x1, ..., xN)
N = 200
data = X.rvs(size=N)
# Now we are ready to calculate the posterior which the Beta we have above
alpha = 1.0 + data.sum()
beta = 1.0 + N - data.sum()
Theta_post = st.beta(alpha, beta)
# Now we can plot the posterior PDF for theta
fig, ax = plt.subplots()
thetas = np.linspace(0, 1, 100)
ax.plot([theta_true], [0.0], 'o', markeredgewidth=2, markersize=10, label='True
value')
ax.plot(thetas, Theta_post.pdf(thetas), label=r'$p(\theta|x_{1:N})$')
ax.set_xlabel(r'$\theta$')
ax.set_title('$N={0:d}$'.format(N))
plt.legend(loc='best')
```



Credible Intervals（置信区间） The posterior $p(\theta|x_{1:N})$ captures everything that we have to say about $\theta$. Credible intervals are a way to summarize it. A credible interval is basically an interval inside which the parameter $\theta$ lies with high probability. Specifically, a 95% credible interval (ℓ, u) (for lower and upper bounds) for $\theta$ is such that:

$$p(\ell \le \theta \le u|x_{1:N}) = 0.95.$$

Of course, there is not a unique credible interval. You can move (ℓ, u) to the left or to the right in a way that keeps the probability contained in it at 0.95.

The *central credible interval* is particularly common. It is defined by solving the following problems

$$p(\theta \leq \ell | x_{1:N}) = 0.025,$$

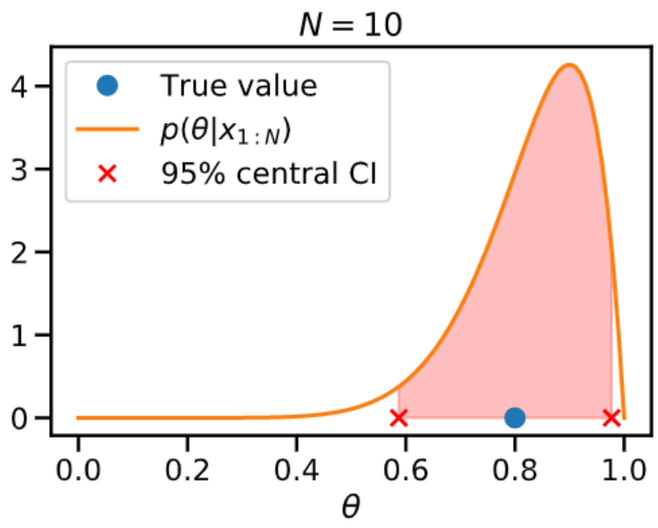
and

$$p(\theta \leq u | x_{1:N}) = 0.975,$$

for ℓ and u, respectively. Here is how you can do it for the coin toss example above since you have the posterior analytically:

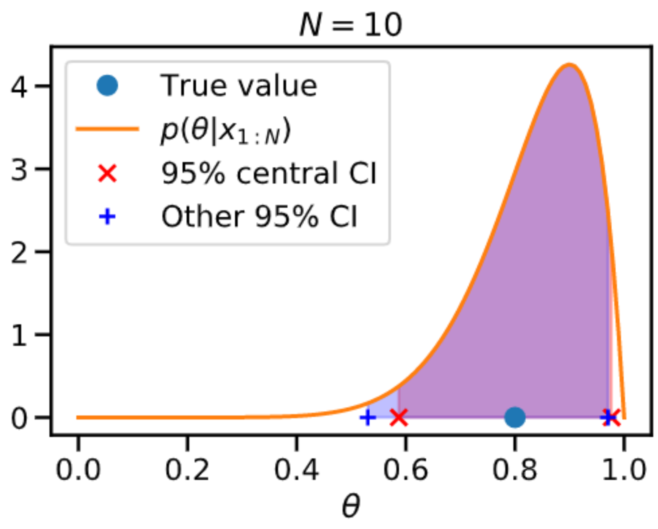
❄ 仿照上一程序, 改用小样本量 $N = 10$， 得到 $\theta$ 的后验密度 Theta_post, 利用 `ppf` 返回分位数.

```
# Take a fake coin which is a little bit biased
theta_true = 0.8
# This is the random variable corresponding to a coin toss
X = st.bernoulli(theta_true)

# Sample from it a number of times to generate our data = (x1, ..., xN)
N = 10
data = X.rvs(size=N)
# Now we are ready to calculate the posterior which the Beta we have above
alpha = 1.0 + data.sum()
beta = 1.0 + N - data.sum()
Theta_post = st.beta(alpha, beta)
theta_low = Theta_post.ppf(0.025)
theta_up = Theta_post.ppf(0.975)
print('Theta is in [{0:1.2f}, {1:1.2f}] with 95% probability'.format(theta_low,
theta_up))
Theta is in [0.59, 0.98] with 95% probability

# Here is a visualization of this credible interval:
fig, ax = plt.subplots()
ax.plot([theta_true], [0.0], 'o', markeredgewidth=2, markersize=10, label='True
value')
ax.plot(thetas, Theta_post.pdf(thetas), label=r'$p(\theta|x_{1:N})$')
thetas_int = np.linspace(theta_low, theta_up, 100)
# 填充范围为thetas_int,下端为0,其形状为thetas_int.shape,上端为
Theta_post.pdf(thetas_int)
ax.fill_between(thetas_int, np.zeros(thetas_int.shape),
Theta_post.pdf(thetas_int), color='red', alpha=0.25)
ax.plot([theta_low, theta_up], np.zeros((2,)), 'x', color='red',
markeredgewidth=2, label='95% central CI') #np.zeros(2)也可以
ax.set_xlabel(r'$\theta$')
ax.set_title('$N={0:d}$'.format(N))
plt.legend(loc='best')
```



置信区间还可以是1%-96%分位数给出

```
theta_low_o = Theta_post.ppf(0.01)
theta_up_o = Theta_post.ppf(0.96)
print('Theta is in [{0:1.2f}, {1:1.2f}] with 95%
probability'.format(theta_low_o, theta_up_o))
Theta is in [0.53, 0.97] with 95% probability

fig, ax = plt.subplots()
ax.plot([theta_true], [0.0], 'o', markeredgewidth=2, markersize=10, label='True
value')
ax.plot(thetas, Theta_post.pdf(thetas), label=r'$p(\theta|x_{1:N})$')
thetas_int = np.linspace(theta_low, theta_up, 100)
ax.fill_between(thetas_int, np.zeros(thetas_int.shape),
Theta_post.pdf(thetas_int), color='red', alpha=0.25)
ax.plot([theta_low, theta_up], np.zeros((2,)), 'x', color='red',
markeredgewidth=2, label='95% central CI')
thetas_int_o = np.linspace(theta_low_o, theta_up_o, 100)
ax.fill_between(thetas_int_o, np.zeros(thetas_int_o.shape),
Theta_post.pdf(thetas_int_o), color='blue', alpha=0.25)
ax.plot([theta_low_o, theta_up_o], np.zeros((2,)), '+', color='blue',
markeredgewidth=2, label='Other 95% CI')
ax.set_xlabel(r'$\theta$')
ax.set_title('$N={0:d}$'.format(N))
plt.legend(loc='best')
```



贝叶斯决策(Bayesian Decision Making)

So, we learned about credible intervals. But what if someone asks you to report a single value for θ in the coin toss example? What is the correct way of doing this?

这是所谓的决策问题(decision-making problem), 为此我们需要量化决策犯错的损失, 并最小化损失. 为了formalize 这个概念, 假定有随机变量 θ, 根据训练数据 $x_{1:N}$ 得到估计值 θ'. Let $\ell(\theta',\theta)$ be the loss we incur when we guess θ' and the true value is θ. However, here are some ideas:

- The 0-1 loss:

$$\ell_{01}(\theta',\theta) = \begin{cases} 0, & \text{if } \theta' = \theta \\ 1, & \text{if } \theta' \neq \theta. \end{cases} = 1_{\{\theta' \neq \theta\}}.$$

- The square loss:

$$\ell_2(\theta',\theta) = (\theta' - \theta)^2.$$

- The absolute loss:

$$\ell_1(\theta',\theta) = |\theta' - \theta|.$$

合理的选取 θ 估计值 θ' 应为, 期望(对应于 θ 的后验密度)损失最小(minimize our *expected loss* where the expectation is taken over our posterior state of knowledge about θ). 因此求解问题：

$$\theta^* = \arg\min_{\theta'} \mathbb{E}[\ell(\theta',\theta)|x_{1:N}] = \arg\min_{\theta'} \int \ell(\theta',\theta)p(\theta|x_{1:N})d\theta.$$

> This, in general, is not a problem with an analytical solution. However, for the two special loss functions above the answer is:
>
> - The choice that minimizes the 0-1 loss is the one maximizing the posterior:
>
> $$\theta^*_{01} = \arg\max_{\theta} p(\theta|x_{1:N}).$$
>
> - The choice that minimizes the square loss is the expectation of the random variable:
>
> $$\theta^*_2 = \mathbb{E}[\theta|x_{1:N}] = \int \theta p(\theta|x_{1:N})d\theta.$$
>
> - The choice that minimizes the absolute loss is the median:
>
> $$p(\theta \leq \theta^*_1|x_{1:N}) = 0.5.$$

现在考虑计算0-1损失, 平方损失, 绝对损失：

❀ 0-1损失最小等价于后验密度最大

```
idx = np.argmax(Theta_post.pdf(thetas))
theta_star_01 = thetas[idx]
print('theta_star_01 = {0:1.2f}'.format(theta_star_01))
theta_star_01 = 0.90
```

❀ 平方损失最小对应于期望 Now, let's the theta θ that minimizes the square loss. We just have to find the expectation of the posterior $p(\theta|x_{1:N})$ (which is just a Beta). It is:

$$\theta^*_N = \mathbb{E}[\theta|x_{1:N}] = \frac{1+\sum_{n=1}^N x_n}{1+\sum_{n=1}^N x_n + N + 1 - \sum_{n=1}^N x_n} = \frac{1+\sum_{n=1}^N x_n}{N+2}.$$

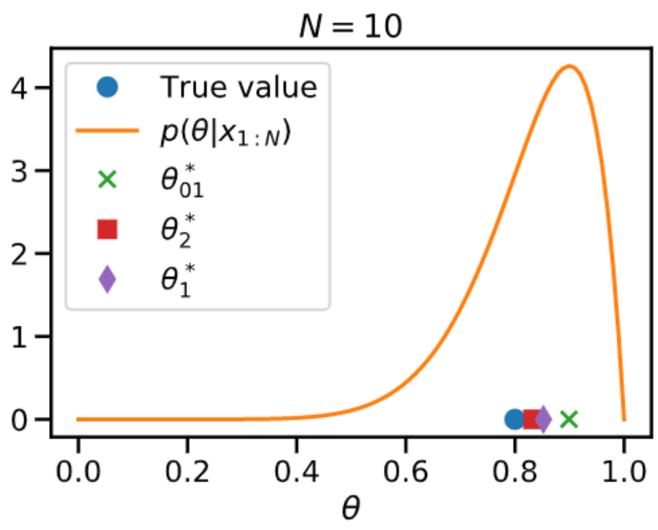
```
# In the example we had above:
theta_star_2 = Theta_post.expect()
print('theta_star_2 = {0:1.2f}'.format(theta_star_2))
theta_star_2 = 0.83
```

❀ 绝对损失最小对应于中位数

```
# In the example we had above:
theta_star_1 = Theta_post.median()
print('theta_star_1 = {0:1.2f}'.format(theta_star_1))
theta_star_1 = 0.85
```

❀ See them all together in the same plot:

```
fig, ax = plt.subplots()
ax.plot([theta_true], [0.0], 'o', markeredgewidth=2, markersize=10, label='True
value')
ax.plot(thetas, Theta_post.pdf(thetas), label=r'$p(\theta|x_{1:N})$')
ax.plot(theta_star_01, 0, 'x', markeredgewidth=2, label=r'$\theta^*_{01}$')
ax.plot(theta_star_2, 0, 's', markeredgewidth=2, label=r'$\theta^*_{2}$')
ax.plot(theta_star_1, 0, 'd', markeredgewidth=2, label=r'$\theta^*_{1}$')
ax.set_xlabel(r'$\theta$')
ax.set_title('$N={0:d}$'.format(N))
plt.legend(loc='best')
```



✸ **Example: The Normal distribution**

The normal (or Gaussian) distribution is a ubiquitous(无处不在) one. It appears over and over again. There are two explanations as to why it appears so often:

- It is the distribution of maximum uncertainty that matches a known mean and a known variance variance.（用以匹配不确定性最常用的分布）
- It is the distribution that arises when you add a lot of random variables together.（CLT）

We will learn about both these in the next lectures. We write:

$$X|\mu,\sigma \sim N(\mu,\sigma),$$

and we read "X conditioned on μ and σ follows a normal distribution with mean μ and variance σ^2.

When $\mu = 0$ and $\sigma^2 = 1$, we say that we have a *standard normal* distribution. Let

$$Z \sim N(0,1).$$

The PDF of the standard normal is:

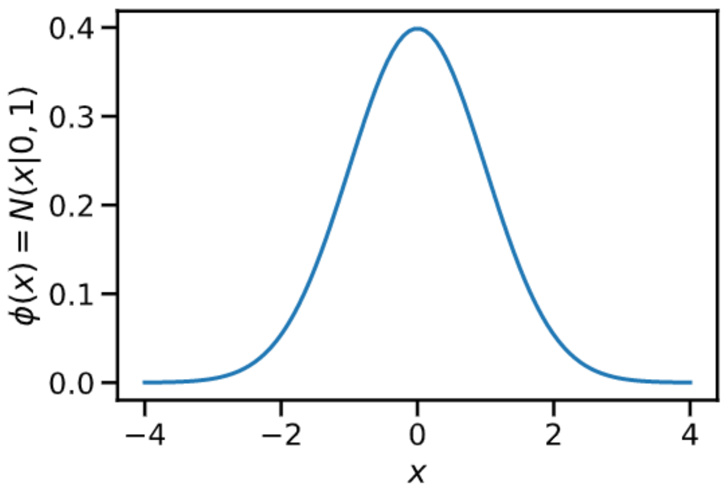
$$\phi(z) := N(z|0,1) = \frac{1}{\sqrt{2\pi}}\exp\left\{-\frac{z^2}{2}\right\}.$$

The CDF of the standard normal is:

$$\Phi(z) := \mathbb{P}(Z \le z) = \int_{-\infty}^{z}\phi(z')dz',$$

is not analytically available. However, there are codes that can compute it.

```
# Here is how you can get the PDF of the standard normal
Z = st.norm()
fig, ax = plt.subplots()
zs = np.linspace(-4.0, 4.0, 100)
ax.plot(zs, Z.pdf(zs))  #ax.plot(zs, Z.cdf(zs))
ax.set_xlabel('$x$')
ax.set_ylabel('$\phi(x) = N(x|0,1)$');
```

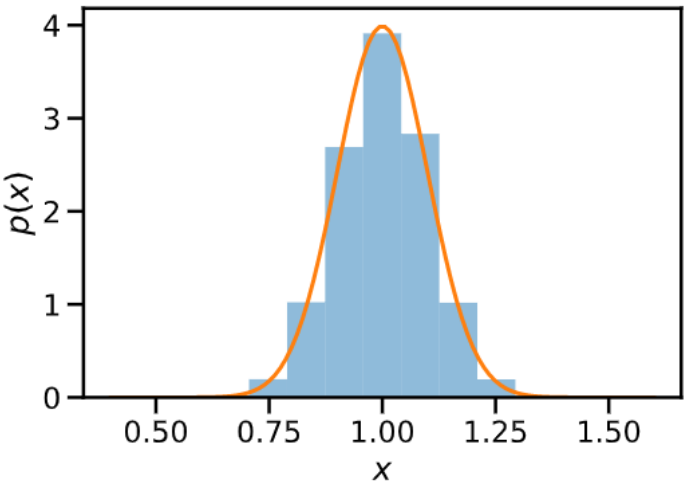


Using the standard normal, we can express any normal. It is easy to show that:

$$X = \mu + \sigma Z,$$

follows a $N(\mu, \sigma^2)$ if Z follows and $N(0,1)$. For example, using this relationship you can sample from any normal using samples from the standard normal. Let's take some samples exploiting this relationship and then compare the histogram to the true PDF.

```
mu = 1.0
sigma = 0.1
X = st.norm(mu, sigma)
Z = st.norm()
xs = np.linspace(mu - 6.0 * sigma, mu + 6.0 * sigma, 100)
x_samples = mu + sigma * Z.rvs(size=10000) #x_samples = X.rvs(size=10000)
fig, ax = plt.subplots()
ax.hist(x_samples, density=True, alpha=0.5)
ax.plot(xs, X.pdf(xs))
ax.set_xlabel('$x$')
ax.set_ylabel('$p(x)$');
```



Example: Inferring the mean of a normal with a known variance（方差已知估均值）

Assume that we are performing an experiment X_n that measures the acceleration of gravity(重力加速度) and that we know that the measurement variance is $\sigma = 0.1$. So, we have:

$$X_n|g,\sigma \sim N(g,\sigma^2).$$

该模型表明, 测量的加速度 X_n 符合均值为 g (真实的加速度), 标准差 $\sigma = 0.1$ 的正态分布, 那么问题来了, g 的认知我们有多少呢？假定

$$g|g_0,s_0 \sim N(g_0,s_0^2),$$

其中 $g_0 = 10$, $s_0 = 0.4$. 在 σ, g_0 和 s_0 条件下的联合概率密度为

$$\begin{aligned} p(x_{1:N},g|\sigma,g_0,s_0) =& p(x_{1:N}|g,\sigma)p(g|g_0,s_0) \\ =& \left(\prod_{n=1}^N p(x_n|g,\sigma)\right) p(g|g_0,s_0) \\ =& \left(\prod_{n=1}^N N(x_n|g,\sigma)\right) N(g|g_0,s_0) \\ =& \left(\prod_{n=1}^N \frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{(x_n-g)^2}{2\sigma^2}\right\}\right) \frac{1}{\sqrt{2\pi}s_0}\exp\left\{-\frac{(g-g_0)^2}{2s_0^2}\right\} \\ \propto& \exp\left\{-\frac{\sum_{n=1}^N(x_n-g)^2}{2\sigma^2}-\frac{(g-g_0)^2}{2s_0^2}\right\}, \end{aligned}$$

注: $p(x_{1:N},g|\sigma,g_0,s_0) = p(x_{1:N}|g,\sigma,g_0,s_0)p(g|\sigma,g_0,s_0)$，其中由于 g_0,s_0 的信息可决定 g，即有

$$\sigma(g) \subset \sigma(g_0,s_0) \Rightarrow p(x_{1:N}|g,\sigma,g_0,s_0) = p(x_{1:N}|g,\sigma)$$

此外, σ 与 g 独立, 故有 $p(g|\sigma,g_0,s_0) = p(g|g_0,s_0)$.

where we have ignored all proportionality constants for convenience. From Bayes' rule, we have that our posterior state of knowledge:

$$\begin{aligned} p(g|x_{1:N},\sigma,g_0,s_0) =& \frac{p(x_{1:N},g|\sigma,g_0,s_0)}{p(x_{1:N}|\sigma,g_0,s_0)} \\ \propto& p(x_{1:N},g|\sigma,g_0,s_0) \\ \propto& \exp\left\{-\frac{\sum_{n=1}^N(x_n-g)^2}{2\sigma^2}-\frac{(g-g_0)^2}{2s_0^2}\right\}. \end{aligned}$$

That's pretty much the answer without the normalization constant. But in this particular case, we can actually match this posterior to a normal distribution by following a technique known as "completing the square."(完全平方) Let's try it out (you can lump into an additive "const" everything that doesn't depend on $g$):

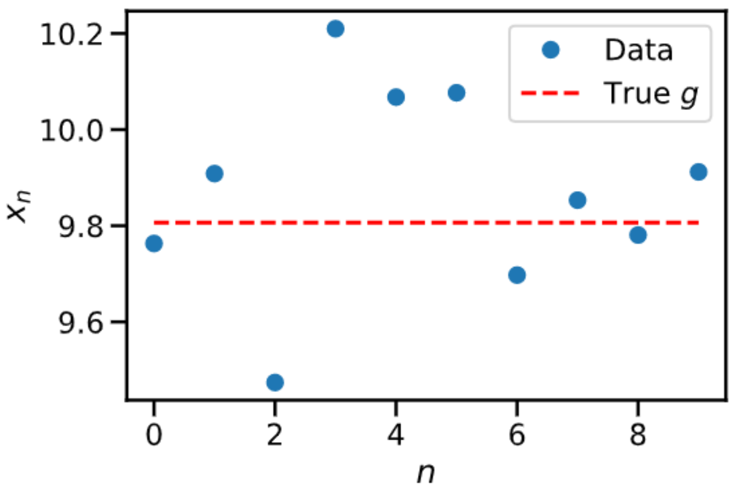
$$
\begin{aligned}
\frac{\sum_{n=1}^N(x_n-g)^2}{2\sigma^2}+\frac{(g-g_0)^2}{2s_0^2} &= \frac{s_0^2\sum_{n=1}^N(x_n-g)^2+\sigma^2(g-g_0)^2}{2\sigma^2 s_0^2}\\
&= \frac{s_0^2Ng^2-2s_0^2g\sum_{n=1}^N x_n+s_0^2\sum_{n=1}^N x_n+\sigma^2g^2-2\sigma^2gg_0+\sigma^2g_0^2}{2\sigma^2 s_0^2}\\
&= \frac{g^2(s_0^2N+\sigma^2)-2g(s_0^2\sum_{n=1}^N x_n+\sigma^2g_0)}{2\sigma^2 s_0^2}+\text{const}\\
&= \frac{g^2-2g(s_0^2\sum_{n=1}^N x_n+\sigma^2g_0)(s_0^2N+\sigma^2)^{-1}}{2\sigma^2 s_0^2(s_0^2N+\sigma^2)^{-1}}+\text{const}\\
&= \frac{g^2-2g\frac{1}{\frac{N}{\sigma^2}+\frac{1}{s_0^2}}\left(\frac{g_0}{s_0^2}+\frac{\sum_{n=1}^N x_n}{\sigma^2}\right)}{2\left(\frac{1}{s_0^2}+\frac{N}{\sigma^2}\right)^{-1}}+\text{const.}
\end{aligned}
$$

Ok, it takes a bit more algebra than usual. Now, we put this back into the exponential (with the minus sign in front of it), and we observe that it gives a normal for the posterior:

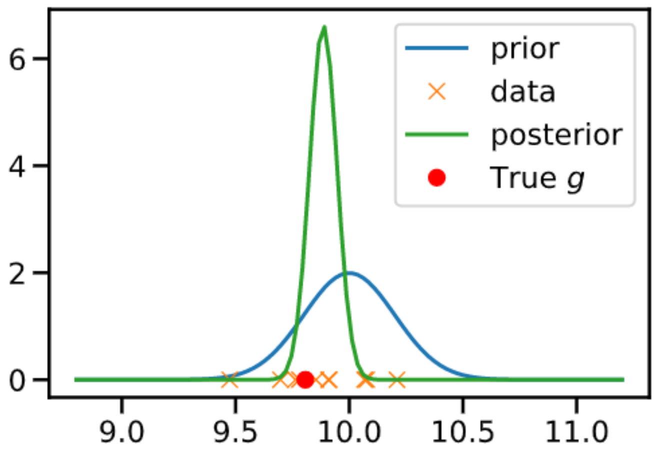
$$
p(g|x_{1:N},\sigma,g_0,s_0)=N\left(g\middle|\frac{1}{\frac{N}{\sigma^2}+\frac{1}{s_0^2}}\left(\frac{g_0}{s_0^2}+\frac{\sum_{n=1}^N x_n}{\sigma^2}\right),\left(\frac{1}{s_0^2}+\frac{N}{\sigma^2}\right)^{-1}\right).
$$

Not very pretty. But it is what it is. Let's experiment with this.

```
# Get the true acceleration of gravity from scipy
import scipy.constants
g_true = scipy.constants.g
# Generate some synthetic data
N = 10
sigma = 0.2
gs = g_true + sigma * np.random.randn(N) ##观测值取自N(9.8,0.2)
fig, ax = plt.subplots()
ax.plot(gs, 'o', label='Data')
ax.plot(range(N), [g_true] * N, 'r--', label='True $g$')
ax.set_xlabel('$n$')
ax.set_ylabel('$x_n$')
plt.legend(loc='best')
```



```
# Our prior state of knowledge
g0 = 10
s0 = 0.2
G_prior = st.norm(g0, s0) ##先验估计为N(10,0.2^2)
# We shoed that the posterior for g conditioned on data is a Gaussian with mean
gn = 1.0 / (N / sigma ** 2 + 1.0 / s0 ** 2) * (g0 / s0 ** 2 + gs.sum() / sigma
** 2)
# And standard deviation
sigman = np.sqrt(1.0 / (1.0 / s0 ** 2 + N / sigma ** 2))
print('The posterior of g is N({0:1.2f}, ({1:1.2f})^2)'.format(gn, sigman))
The posterior of g is N(9.89, (0.06)^2)

# Let's draw this
G_post = st.norm(gn, sigman)  ##后验估计为 N(9.89, (0.06)^2)
fig, ax = plt.subplots()
ggs = np.linspace(g0 - 6.0 * s0, g0 + 6.0 * s0, 100)
ax.plot(ggs, G_prior.pdf(ggs), label='prior')
ax.plot(gs, np.zeros(gs.shape), 'x', label='data')
ax.plot(ggs, G_post.pdf(ggs), label='posterior')
ax.plot([g_true], [0.0], 'ro', label='True $g$')
plt.legend(loc='best');

# Let's get a credible interval:
g_025 = G_post.ppf(0.025)
g_975 = G_post.ppf(0.975)
print('g is in [{0:1.2f}, {1:1.2f}] with 95% probability'.format(g_025, g_975))
g is in [9.77, 10.00] with 95% probability
```



伪随机数生成器(Pseudo-random number generators,PRNG)

Random number generation is the backbone of Bayesian inference. Computers are deterministic. So, how can they generate random numbers? Well they cannot! But they can produce sequence of numbers that look like random numbers! These "fake" random number generators are called Pseudo-random number generators (PRNG). They are used to generate random numbers between zero and a maximum integer, say m. As we will argue later this is sufficient to generate pretty much any random variable you want.

The middle square algorithm (Von Neumann)

The middlesquare algorithm is the simplest PRNG.

1. Take a number and square it.

2. Pad the result with zeros to get to the desired number of digits.(用零填充结果以得到所需的位数)
3. Take the middle digits of the resulting number.
4. Repeat.

Here is an implementation:

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_context('talk')
import numpy as np

def middlesquare(s, digits=4):
    """
    :param s:      The initial seed.
    :param digits: How many digits do you want.
    """
    # Square the number
    s2 = s ** 2
    # Turn the resulting number into a string padding with zeros to get to the
desired number of digits
    s2_str = str(s2).zfill(2*digits)  ##前面用0补充至字符串长度为2*digits
    # Keep only the middle; [1:]第二位到最后,[:-2]最前位到倒数第三位
    middle_str = s2_str[int(np.floor(digits/2)):][:-int(np.floor(digits/2))]
    return int(middle_str)

seed = 1234
s = seed
for _ in range(20):
    s = middlesquare(s, digits=4)
    print(s)
```

Unfortunately, the middle square algorithms results in periodic sequences with very small period.(可能出现周期循环结果)

线性同余生成器(Linear congruential generator ,LCG)

The linear congruential generator works as follows. You pick three big integers a, b and m. Pick a seed x_0. Then iterate:

$$x_{i+1} = (ax_i + b) \mod m$$

Mersenne Twister PRNG

Numpy uses the Mersenne Twister to generate random numbers. Its details are more complicated than LCG, but it is still initialized by an integer seed. You can test it as follows:

```
# set the seed
np.random.seed(12345)
# print 5 integers from 0 to 6012119
for _ in range(5):
    print(np.random.randint(0, 6012119))

# see what the seed does - Here is what happens if you rerun the code above:
for _ in range(5):
    print(np.random.randint(0, 6012119))
```

```python
# And here is what happens if you reset the seed to its original value and rerun
the code
np.random.seed(12345)
for _ in range(5):
    print(np.random.randint(0, 6012119))
```

So, resetting the seed gives you the same sequence. In your numerical simulations you should always set the seed by hand in order to ensure the reproducibility(再现) of your work.

Sampling from the uniform distribution

If we have a PRNG that samples between zero and a big integer, say m, we can create a generator that samples from the uniform distribution. If d is the sample from the PRNG, then

$$x = \frac{d}{m},$$

is approximately uniformly distributed. Let's experiment with this idea.

```python
# The maximum integer
m = 6012119

# First a uniform random generator based on lcg
lcg_seed = 123456 # A seed of lcg
lcg_state = lcg_seed # Internal state of lcg
def unif_lcg():
    """
    Samples from the uniform using LCG.
    """
    global lcg_state
    lcg_state = lcg(lcg_state)
    return lcg_state / (1. * m) # The 1. in the denominator ensures
                                # that the division is done in floating point
arithmetic
print('LCG Uniform Samples:')
for _ in range(5):
    print(unif_lcg())

# And let's also do it with Mersenne Twister from numpy
np.random.seed(123456)
def unif_mt():
    """
    Samples from the uniform using the MT.
    """
    return np.random.randint(0, m) / (1. * m)
print('\nMT Uniform Samples:')
for _ in range(5):
    print(unif_mt())
```