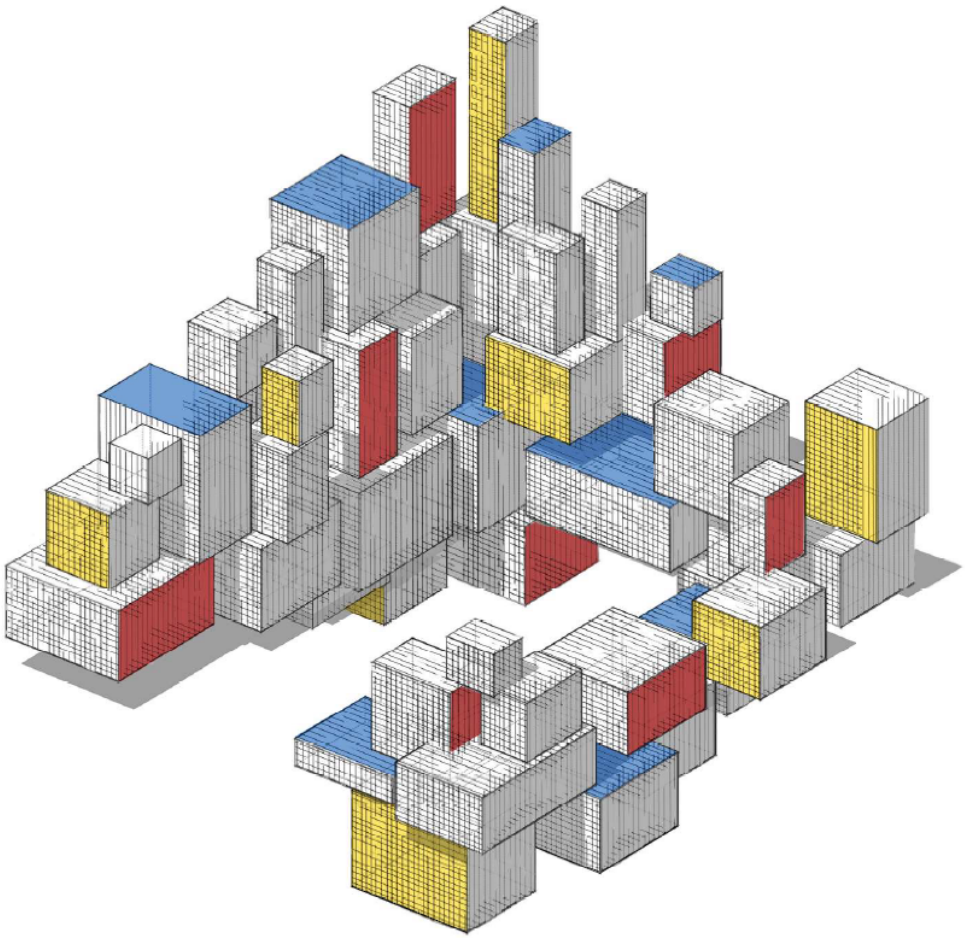
Markov Chain Monte Carlo



Purpose

In this lecture we discuss:

- Simulating Random Vectors
- Simulating Markov Chains
- Resampling Techniques
- Markov Chain Monte Carlo
  - Metropolis–Hastings Sampler
  - Gibbs Sampler

Simulating Random Vectors and Processes

Two general scenarios for simulating random vectors and processes:

1. $X_1, \ldots, X_n$ are independent, with pdfs f_i, $i = 1, \ldots, n$. The random vector $\boldsymbol{X} = [X_1, \ldots, X_n]^\top$ can be simply simulated by drawing each component $X_i \sim f_i$ individually — for example, via the inverse-transform method or acceptance–rejection.
2. For dependent components $X_1, \ldots, X_n$, we can, as a consequence of the *product rule* of probability, represent the joint pdf $f(\boldsymbol{x})$ as

$$f(\boldsymbol{x}) = f(x_1, \ldots, x_n) = f_1(x_1)\, f_2(x_2 \,|\, x_1) \cdots f_n(x_n \,|\, x_1, \ldots, x_{n-1}).$$

   One can generate $\boldsymbol{X}$ by first generating X_1, then, given $X_1 = x_1$, generate X_2 from $f_2(x_2 \,|\, x_1)$, and so on, until generating X_n from $f_n(x_n \,|\, x_1, \ldots, x_{n-1})$.

Simulating Markov Chains

A Markov chain is a stochastic process $\{X_t, t = 0, 1, 2, \ldots\}$ that satisfies the *Markov property*; meaning that for all t and s the conditional distribution of X_{t+s} given $X_u, u \leqslant t$, is the same as that of X_{t+s} given only X_t.

As a result, each conditional density $f_t(x_t \mid x_1, \ldots, x_{t-1})$ can be written as a one-step transition density $q_t(x_t \mid x_{t-1})$; that is, the probability density to go from state x to state y in one step.

In many cases of interest the chain is time-homogeneous, meaning that the transition density q_t does not depend on t. Such Markov chains can be generated *sequentially*, as given in the following algorithm.

Algorithm 1: Simulate a Markov Chain

input: Number of steps N, initial pdf f_0, transition density q.

Draw X_0 from the initial pdf f_0.

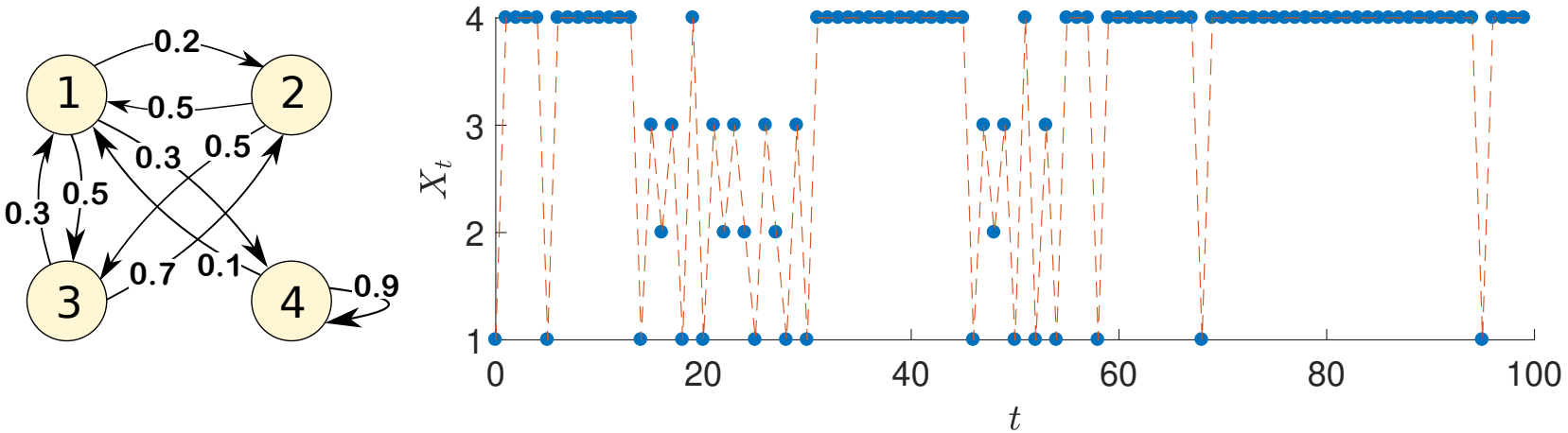
for $t = 1$ **to** N **do**

    Draw X_t from the distribution corresponding to the transition density $q(\cdot \mid X_{t-1})$

return $X_0, \ldots, X_N$

Example: Markov Chain Simulation

For time-homogeneous Markov chains with a discrete state space, we can visualize the one-step transitions by means of a transition graph, where arrows indicate possible transitions between states and the labels describe the corresponding probabilities. The following figure shows (on the left) the transition graph of the Markov chain $\{X_t, t = 0, 1, 2, \ldots\}$ with state space $\{1, 2, 3, 4\}$, starting at state 1, and on the right a typical path of the Markov chain.



MCsim.py

```
import numpy as np
import matplotlib.pyplot as plt

n = 101
P = np.array([[0, 0.2, 0.5, 0.3],
              [0.5, 0, 0.5, 0],
              [0.3, 0.7, 0, 0],
              [0.1, 0, 0, 0.9]])
x = np.array(np.ones(n, dtype=int))
x[0] = 0
for t in range(0,n-1):
    x[t+1] = np.min(np.where(np.cumsum(P[x[t],:]) >
                    np.random.rand()))
x = x + 1  #add 1 to all elements of the vector x
plt.plot(np.array(range(0,n)),x, 'o')
plt.plot(np.array(range(0,n)),x, '--')
plt.show()
```

Resampling

The idea behind resampling is very simple: an iid sample $\tau := \{x_1, \ldots, x_n\}$ from some unknown cdf F represents our best knowledge of F if we make no further *a priori* assumptions about it. If it is not possible to simulate more samples from F, the best way to "repeat" the experiment is to resample each x_i with equal probability and repeat this N times.

Algorithm 2: Sampling from an Empirical Cdf.

---

input: Original iid sample $x_1, \ldots, x_n$ and sample size N.

output : Iid sample $X_1^*, \ldots, X_N^*$ from the empirical cdf.

1 **for** $t = 1$ **to** N **do**

2 Draw $U \sim \mathcal{U}(0, 1)$

3 Set $I \leftarrow \lceil nU \rceil$ `// discrete uniformly from` $1, \ldots, n$

4 Set $X_t^* \leftarrow x_I$

5 **return** $X_1^*, \ldots, X_N^*$

Example: Quotient of Uniforms

Let $U_1, \ldots, U_n, V_1, \ldots, V_n$ be iid $\mathcal{U}(0,1)$ random variables and define $X_i = U_i/V_i$, $i = 1, \ldots, n$.

Suppose we wish to investigate the distribution of the sample median $\widetilde{X}$ and sample mean $\overline{X}$ of the (random) data $\mathcal{T} := \{X_1, \ldots, X_n\}$.

Since we know the model for $\mathcal{T}$ exactly, we can generate a large number, N say, of independent copies of it, and for each of these copies evaluate the sample medians $\widetilde{X}_1, \ldots, \widetilde{X}_N$ and sample means $\overline{X}_1, \ldots, \overline{X}_N$.

For $n = 100$ and $N = 1000$ the empirical cdfs might look like the left and right curves in Figure 1, respectively.

The distributions of the sample median and sample mean do not match at all. The sample median is quite concentrated around 1, whereas the distribution of the sample mean is much more spread out.

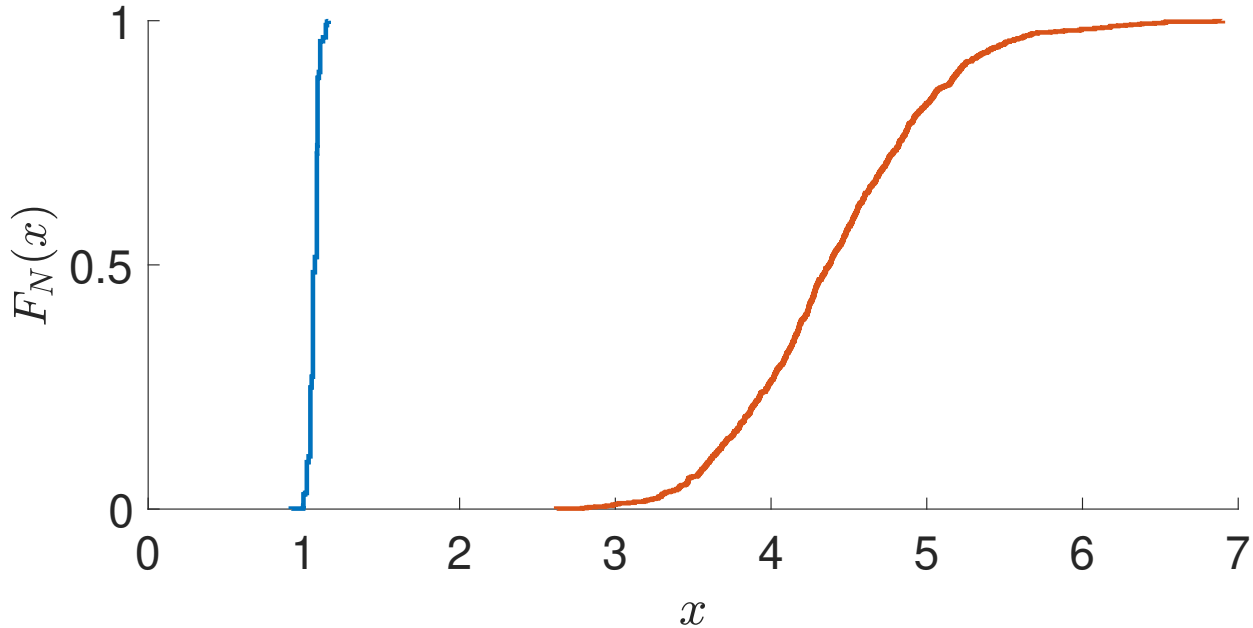


Figure: Empirical cdfs of the medians of the resampled data (left curve) and sample means (right curve) of the resampled data.

Instead of sampling completely new data, we could also *reuse* the original data by resampling them via Algorithm 2. This gives independent copies $\widetilde{X}_1^*, \ldots, \widetilde{X}_N^*$ and $\overline{X}_1^*, \ldots, \overline{X}_N^*$, for which we can again plot the empirical cdf.

The results will be similar to the previous case. In fact, in the figure above, the empirical cdfs of the *resampled* sample medians and sample means were plotted.

The corresponding Python code is given below. The essential point of this example is that resampling of data can greatly add to the understanding of the probabilistic properties of certain measurements on the data, *even if the underlying model is not known*.

quotunif.py

```
import numpy as np
from numpy.random import rand, choice
import matplotlib.pyplot as plt
from statsmodels.distributions.empirical_distribution import ECDF

n = 100
N = 1000
x = rand(n)/rand(n)  # data
med = np.zeros(N)
ave = np.zeros(N)
for i in range(0,N):
    s = choice(x, n, replace=True) # resampled data
    med[i] = np.median(s)
    ave[i] = np.mean(s)

med_cdf = ECDF(med)
ave_cdf = ECDF(ave)
plt.plot(med_cdf.x, med_cdf.y)
plt.plot(ave_cdf.x, ave_cdf.y)
plt.show()
```

Markov Chain Monte Carlo

Markov chain Monte Carlo (MCMC) is a Monte Carlo sampling technique for (approximately) generating samples from an arbitrary distribution — often referred to as the target distribution.

The basic idea is to run a Markov chain long enough such that its limiting distribution is close to the target distribution.

The initial random variables in the Markov chain may have a distribution that is significantly different from the target (limiting) distribution. The random variables that are generated during this burn-in period are often discarded.

The remaining random variables form an *approximate* and *dependent* sample from the target distribution.

We discuss two popular MCMC samplers: the Metropolis–Hastings sampler and the Gibbs sampler.

Metropolis–Hastings Sampler

We wish to sample from a target pdf $f(\boldsymbol{x})$, where $\boldsymbol{x}$ takes values in some d-dimensional set. The aim is to construct a Markov chain $\{\boldsymbol{X}_t, t = 0, 1, \ldots\}$ in such a way that its limiting pdf is f.

Suppose the Markov chain is in state $\boldsymbol{x}$ at time t. A transition of the Markov chain from state $\boldsymbol{x}$ is carried out in two phases. First a proposal state $\boldsymbol{Y}$ is drawn from a transition density $q(\cdot \,|\, \boldsymbol{x})$. This state is accepted as the new state, with acceptance probability

$$\alpha(\boldsymbol{x}, \boldsymbol{y}) = \min\left\{\frac{f(\boldsymbol{y})\, q(\boldsymbol{x} \,|\, \boldsymbol{y})}{f(\boldsymbol{x})\, q(\boldsymbol{y} \,|\, \boldsymbol{x})}, 1\right\}, \tag{1}$$

or rejected otherwise. In the latter case the chain remains in state $\boldsymbol{x}$. The algorithm just described can be summarized as follows.

Algorithm 3: Metropolis–Hastings Sampler

input: Initial state $\boldsymbol{X}_0$, sample size N, target pdf $f(\boldsymbol{x})$, proposal function $q(\boldsymbol{x}, \boldsymbol{y})$.

output: $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ (dependent), approximately distributed according to $f(\boldsymbol{x})$.

1 **for** $t = 0$ **to** $N - 1$ **do**

2 Draw $\boldsymbol{Y} \sim q(\boldsymbol{y} \mid \boldsymbol{X}_t)$ `// draw a proposal`

3 $\alpha \leftarrow \alpha(\boldsymbol{X}_t, \boldsymbol{Y})$ `// acceptance probability as in` (1)

4 Draw $U \sim \mathcal{U}(0, 1)$

5 **if** $U \leqslant \alpha$ **then** $\boldsymbol{X}_{t+1} \leftarrow \boldsymbol{Y}$

6 **else** $\boldsymbol{X}_{t+1} \leftarrow \boldsymbol{X}_t$

7 **return** $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$

Note that one only needs to know the target pdf $f(\boldsymbol{x})$ *up to a constant*; that is $f(\boldsymbol{x}) = c\, \overline{f}(\boldsymbol{x})$ for some known function $\overline{f}(\boldsymbol{x})$ but unknown constant c.

Detailed Balance Equations

Because a transition of the Metropolis–Hastings Markov chain consists of two steps, the one-step transition probability to go from $\boldsymbol{x}$ to $\boldsymbol{y}$ is not $q(\boldsymbol{y} \mid \boldsymbol{x})$ but

$$\widetilde{q}(\boldsymbol{y} \mid \boldsymbol{x}) = \begin{cases} q(\boldsymbol{y} \mid \boldsymbol{x})\, \alpha(\boldsymbol{x}, \boldsymbol{y}), & \text{if } \boldsymbol{y} \neq \boldsymbol{x}, \\ 1 - \sum_{\boldsymbol{z} \neq \boldsymbol{x}} q(\boldsymbol{z} \mid \boldsymbol{x})\, \alpha(\boldsymbol{x}, \boldsymbol{z}), & \text{if } \boldsymbol{y} = \boldsymbol{x}. \end{cases}$$

This transition density satisfies the detailed balance equations:

$$f(\boldsymbol{x})\, \widetilde{q}(\boldsymbol{y} \mid \boldsymbol{x}) = f(\boldsymbol{y})\, \widetilde{q}(\boldsymbol{x} \mid \boldsymbol{y}) \quad \text{for all } \boldsymbol{x}, \boldsymbol{y}.$$

This is the reason why the limiting distribution of the Metropolis–Hastings Markov chain is equal to the target distribution.

Choice of the Proposal

Ideally, we would like $q(\boldsymbol{y} \mid \boldsymbol{x})$ to be "close" to the target $f(\boldsymbol{y})$, irrespective of $\boldsymbol{x}$. Two common approaches are:

1. Independence Sampler: Choose $q(\boldsymbol{y} \mid \boldsymbol{x}) = g(\boldsymbol{y})$ independent of $\boldsymbol{x}$ for some pdf $g(\boldsymbol{y})$. The acceptance probability is thus

$$\alpha(\boldsymbol{x}, \boldsymbol{y}) = \min\left\{\frac{f(\boldsymbol{y})\, g(\boldsymbol{x})}{f(\boldsymbol{x})\, g(\boldsymbol{y})},\ 1\right\}.$$

2. Random Walk Sampler: Choose a symmetric proposal density $q(\boldsymbol{y} \mid \boldsymbol{x}) = q(\boldsymbol{x} \mid \boldsymbol{y})$. Then the acceptance probability has the simple form

$$\alpha(\boldsymbol{x}, \boldsymbol{y}) = \min\left\{\frac{f(\boldsymbol{y})}{f(\boldsymbol{x})},\ 1\right\}. \tag{2}$$

A typical example is when, for a given current state $\boldsymbol{x}$, the proposal state $\boldsymbol{Y}$ is of the form $\boldsymbol{Y} = \boldsymbol{x} + \boldsymbol{Z}$, where $\boldsymbol{Z}$ is generated from some spherically symmetric distribution, such as $\mathcal{N}(\boldsymbol{0}, \mathbf{I})$.

Example: Random Walk Sampler

For an unspecified constant c, consider the two-dimensional pdf

$$f(x_1, x_2) = c\, \mathrm{e}^{-\frac{1}{4}\sqrt{x_1^2+x_2^2}} \left(\sin\left(2\sqrt{x_1^2 + x_2^2} \right) + 1 \right), \quad [x_1, x_2] \in [-2\pi, 2\pi]^2.$$

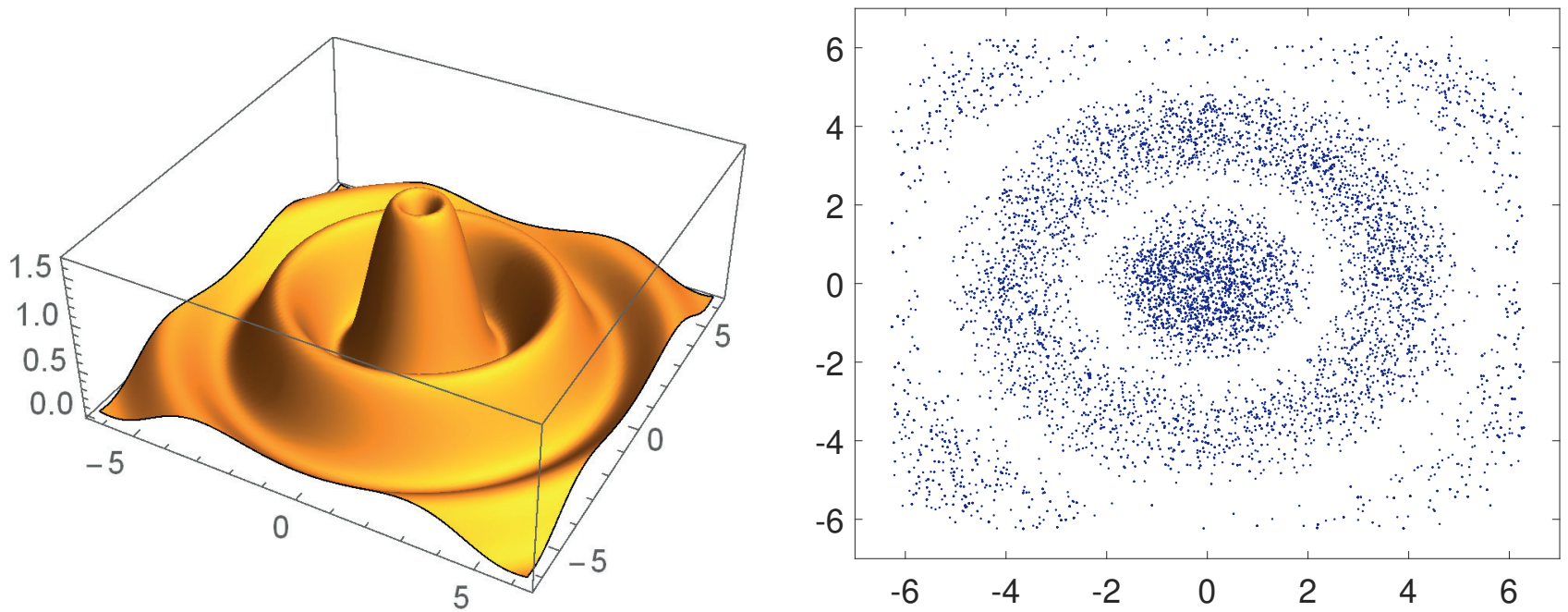


Figure: Left panel: the two-dimensional target pdf. Right panel: points from the random walk sampler. At each step, given a current state $\boldsymbol{x}$, a proposal $\boldsymbol{Y}$ is drawn from the $\mathcal{N}(\boldsymbol{x}, \mathbf{I})$ distribution.

rwsamp.py

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import pi, exp, sqrt, sin
from numpy.random import rand, randn
N = 10000
a = lambda x: -2*pi < x
b = lambda x: x < 2*pi
f = lambda x1, x2: exp(-sqrt(x1**2+x2**2)/4)*(
        sin(2*sqrt(x1**2+x2**2))+1)*a(x1)*b(x1)*a(x2)*b(x2)

xx = np.zeros((N,2))
x = np.zeros((1,2))
for i in range(1,N):
    y = x + randn(1,2)
    alpha = np.amin((f(y[0][0],y[0][1])/f(x[0][0],x[0][1]),1))
    r = rand() < alpha
    x = r*y + (1-r)*x
    xx[i,:] = x

plt.scatter(xx[:,0], xx[:,1], alpha =0.4,s =2)
plt.axis('equal')
```

Gibbs Sampler

The Gibbs sampler is useful for generating n-dimensional random vectors. The key idea of the Gibbs sampler is to update the components of the random vector one at a time, by sampling them from *conditional* pdfs.

Specifically, suppose that we wish to sample a random vector $\boldsymbol{X} = [X_1, \ldots, X_n]^\top$ according to a target pdf $f(\boldsymbol{x})$.

Using Bayesian notation, let $f(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ represent the conditional pdf of the i-th component, X_i, given the other components $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$.

Systematic Gibbs Sampler

Algorithm 4: Systematic Gibbs Sampler

input: Initial point $\boldsymbol{X}_0$, sample size N, and target pdf f.

output: $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ approximately distributed according to f.

1 **for** $t = 0$ **to** $N - 1$ **do**

2 Draw Y_1 from the conditional pdf $f(y_1 \mid X_{t,2}, \ldots, X_{t,n})$.

3 **for** $i = 2$ **to** n **do**

4 Draw Y_i from the conditional pdf $f(y_i \mid Y_1, \ldots, Y_{i-1}, X_{t,i+1}, \ldots, X_{t,n})$.

5 $\boldsymbol{X}_{t+1} \leftarrow \boldsymbol{Y}$

6 **return** $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$

Note that here the components are updated in a fixed order $1 \to 2 \to \cdots \to n$. For this reason Algorithm 4 is also called the systematic Gibbs sampler.

Variants of the Gibbs Sampler

There exist many variants of the Gibbs sampler, depending on cycle steps required to update $\boldsymbol{X}_t$ to $\boldsymbol{X}_{t+1}$.

- Random-order Gibbs sampler: update the components in the order of a random permutation of $\{1, \ldots, n\}$.
- Update the components in blocks (i.e., several at the same time).
- Update only a random selection of components; e.g., only a single component (random Gibbs sampler).
- Reversible Gibbs sampler: update according to $1 \to 2 \to \cdots \to n-1 \to n \to n-1 \to \cdots \to 2 \to 1$.

In all cases, except for the systematic Gibbs sampler, the resulting Markov chain $\{\boldsymbol{X}_t, t = 1, 2, \ldots\}$ is *reversible* and hence its limiting distribution is precisely $f(\boldsymbol{x})$.

Instead, the systematic Gibbs sampler satisfies a related balance condition (Hammersley–Clifford), which guarantees that the limiting pdf is indeed $f(\boldsymbol{x})$.

Example: Gibbs Sampler for the Bayesian Normal Model

Gibbs samplers are often applied in Bayesian statistics, to sample from the posterior pdf. Consider for instance the Bayesian normal model

$$f(\mu, \sigma^2) = 1/\sigma^2$$
$$(\boldsymbol{x} \mid \mu, \sigma^2) \sim \mathcal{N}(\mu \mathbf{1}, \sigma^2 \mathbf{I}).$$

The prior for (μ, σ^2) is here improper. In some sense this prior conveys the least amount of information about μ and σ^2.

Following the usual procedure, we find the posterior pdf:

$$f(\mu, \sigma^2 \mid \boldsymbol{x}) \propto \left(\sigma^2\right)^{-n/2-1} \exp\left\{-\frac{1}{2}\frac{\sum_i (x_i - \mu)^2}{\sigma^2}\right\}. \tag{3}$$

Note that μ and σ^2 here are the "variables" and $\boldsymbol{x}$ is a fixed data vector.

Simulate from the Posterior

To simulate samples μ and σ^2 from (3) using the Gibbs sampler, we need the distributions of both $(\mu \mid \sigma^2, \boldsymbol{x})$ and $(\sigma^2 \mid \mu, \boldsymbol{x})$.

To find $f(\mu \mid \sigma^2, \boldsymbol{x})$, view the right-hand side of (3) as a function of μ only, regarding σ^2 as a constant. This gives

$$f(\mu \mid \sigma^2, \boldsymbol{x}) \propto \exp\left\{-\frac{n\mu^2 - 2\mu \sum_i x_i}{2\sigma^2}\right\} = \exp\left\{-\frac{\mu^2 - 2\mu\bar{x}}{2(\sigma^2/n)}\right\}$$

$$\propto \exp\left\{-\frac{1}{2}\frac{(\mu - \bar{x})^2}{\sigma^2/n}\right\}. \tag{4}$$

This shows that $(\mu \mid \sigma^2, \boldsymbol{x})$ has a normal distribution with mean $\bar{x}$ and variance σ^2/n.

Similarly, to find $f(\sigma^2 \,|\, \mu, \boldsymbol{x})$, view the right-hand side of (3) as a function of σ^2, regarding μ as a constant. This gives

$$f(\sigma^2 \,|\, \mu, \boldsymbol{x}) \propto (\sigma^2)^{-n/2-1} \exp\left\{-\frac{1}{2}\sum_{i=1}^{n}(x_i - \mu)^2/\sigma^2\right\}, \qquad (5)$$

showing that $(\sigma^2 \,|\, \mu, \boldsymbol{x})$ has an inverse-gamma distribution with parameters $n/2$ and $\sum_{i=1}^{n}(x_i - \mu)^2/2$. The Gibbs sampler thus

involves the repeated simulation of

- $(\mu \,|\, \sigma^2, \boldsymbol{x}) \sim \mathcal{N}\left(\overline{x},\ \sigma^2/n\right)$ and
- $(\sigma^2 \,|\, \mu, \boldsymbol{x}) \sim \mathsf{InvGamma}\left(n/2,\ \sum_{i=1}^{n}(x_i - \mu)^2/2\right)$.

Simulating $X \sim \mathsf{InvGamma}(\alpha, \lambda)$ is achieved by first generating $Z \sim \mathsf{Gamma}(\alpha, \lambda)$ and then returning $X = 1/Z$.

gibbsamp.py

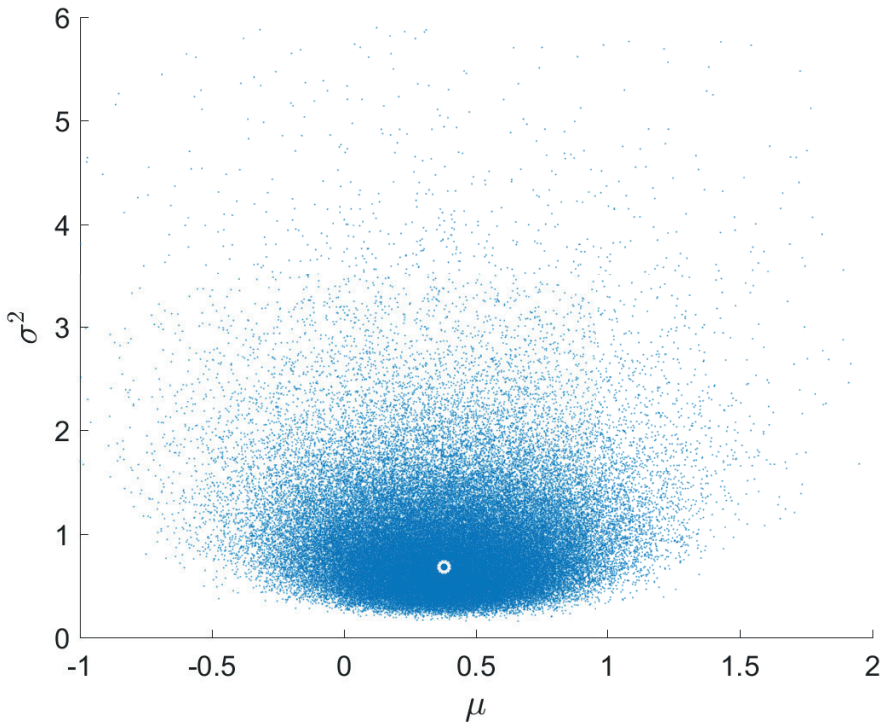
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([[-0.9472, 0.5401, -0.2166, 1.1890, 1.3170,
               -0.4056, -0.4449, 1.3284, 0.8338, 0.6044]])
n=x.size
sample_mean = np.mean(x)
sample_var = np.var(x)
sig2 = np.var(x)
mu=sample_mean

N=10**5
gibbs_sample = np.array(np.zeros((N, 2)))
for k in range(N):
    mu=sample_mean + np.sqrt(sig2/n)*np.random.randn()
    V=np.sum((x-mu)**2)/2
    sig2 = 1/np.random.gamma(n/2, 1/V)
    gibbs_sample[k,:]= np.array([mu, sig2])
plt.scatter(gibbs_sample[:,0], gibbs_sample[:,1],alpha =0.1,s =1)
plt.plot(np.mean(x), np.var(x),'wo')
```

Output from the Gibbs Sampler

The figure shows the posterior point cloud; i.e., the (μ, σ^2) points generated by the Gibbs sampler. The white circle is the point $(\overline{x}, s^2)$, where $\overline{x} = 0.3798$ is the sample mean and $s^2 = 0.6810$ the sample variance.



Posterior for μ

By projecting the (μ, σ^2) points onto the μ-axis — that is, by ignoring the σ^2 values — one obtains (approximate) samples from the posterior pdf of μ; that is, $f(\mu \mid \boldsymbol{x})$.

The figure shows a kernel density estimate of this pdf.

The corresponding 0.05 and 0.95 sample quantiles were found to be -0.2054 and 0.9662, respectively, giving the 95% credible interval $(-0.2054, 0.9662)$ for μ, which contains the true expectation 0.