$\LaTeX$ command declarations here.

AERSP 597 - Machine Learning in Aerosapce Engineering

Lecture 10E, Gaussian Process Regression: Model Selection

Instructor: Daning Huang

In [1]:
```python
from warnings import filterwarnings
filterwarnings('ignore')

import numpy as np
import scipy.stats as ss
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

%matplotlib inline
```

TODAY: Gaussian Process Regression - III

- Monte Carlo simulation
- Basics of Markov chain
- Metropolis-Hastings sampling
- Gibbs sampling

References

- [PRML] Chp. 11
- twiecki.io (https://twiecki.io/blog/2015/11/10/mcmc-sampling/)

Model Selection Revisited

Model selection refers to two tasks of interest:

- Task 1: Given a model $\mathcal{M}$ with hyperparameters θ and a dataset $\mathcal{D}$, find θ such that $\mathcal{M}$ fits $\mathcal{D}$ the best.
  - e.g. $\mathcal{M}$: A GPR with a correlation function; θ: Length scales in the correlation function.
- Task 2: Given several models $\mathcal{M}_1, \mathcal{M}_2, \cdots$ and a dataset $\mathcal{D}$, find $\mathcal{M}_j$ that fits $\mathcal{D}$ the best.
  - e.g. GPR's with different correlation/trend functions.

- For Task 1, one can determine θ by maximizing the likelihood

$$\theta_{MLE} = \arg\max_\theta p(\mathcal{D}|\theta, \mathcal{M})$$

> As disscused in the previous lectures for GPR.

- For Task 2, one could use cross validation, or certain information criteria, to assess the fitness of different models.

- For Task 1, one could also employ a Bayesian approach by assigning priors to θ, and evaluate the posterior,
$$p(\theta|\mathcal{D}, \mathcal{M}) = \frac{p(\mathcal{D}|\theta, \mathcal{M})p(\theta|\mathcal{M})}{p(\mathcal{D}|\mathcal{M})} \propto p(\mathcal{D}|\theta, \mathcal{M})p(\theta|\mathcal{M})$$
  - The posterior $p(\theta|\mathcal{D}, \mathcal{M})$ provides an estimation of θ given the dataset and model, with a confidence bound.
- For Task 2, the evidence $p(\mathcal{D}|\mathcal{M})$ provides a "rigorous" and quantitative assessment of the model fitness.

> Sounds like better than MLE+CV?

Two issues with the Bayesian approach:

- For Task 1, the posterior $p(\theta|\mathcal{D}, \mathcal{M})$ typically do not have a closed-form expression. For example, in GPR
  - $p(\mathcal{D}|\theta, \mathcal{M})$ is typically Gaussian - but in some fancier models it is non-Gaussian
  - $p(\theta|\mathcal{M})$ could be a Beta distribution, if θ are the length scales (positive valued)
- As a result, for Task 2, it is non-trivial to evaluate the integral for the evidence:
$$p(\mathcal{D}|\mathcal{M}) = \int_\Theta p(\mathcal{D}|\theta, \mathcal{M})p(\theta|\mathcal{M})d\theta$$

Solution - **Markov chain Monte Carlo** algorithm

- A procedure to generate samples $\{\theta^{(i)}\}_{i=1}^N$ from the posterior distribution $p(\theta|\mathcal{D}, \mathcal{M}) \propto p(\mathcal{D}|\theta, \mathcal{M})p(\theta|\mathcal{M})$.
  - Then one can use a histogram, or a kernel density estimation, to approximate the posterior distribution.
  - With the posterior distribution, you can locate the best parameters.
  - You can also evaluate the evidence to assess the fitness of the model.
- Challenge: How to generate samples satisfying an *unnormalized* distribution $p(\mathcal{D}|\theta, \mathcal{M})p(\theta|\mathcal{M})$?

Monte Carlo Simulation

- "Integration by **Darts**":

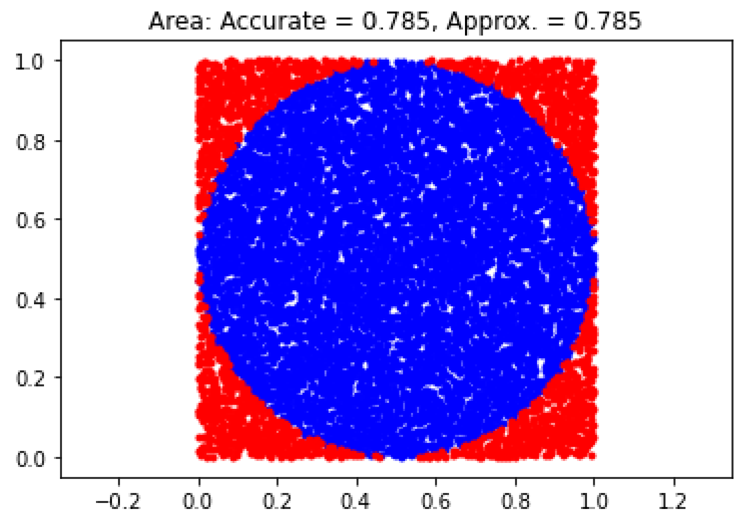
$$\int f(\mathbf{z})p(\mathbf{z})d\mathbf{z} \approx \frac{1}{L}\sum_{l=1}^{L} f(\mathbf{z}^{(l)})$$

- where the samples $\mathbf{z}^{(l)}$ are **drawn from** $p(\mathbf{z})$.
- For the evidence, one can draw $\theta^{(i)}$ from $p(\theta|\mathcal{M})$, and do:

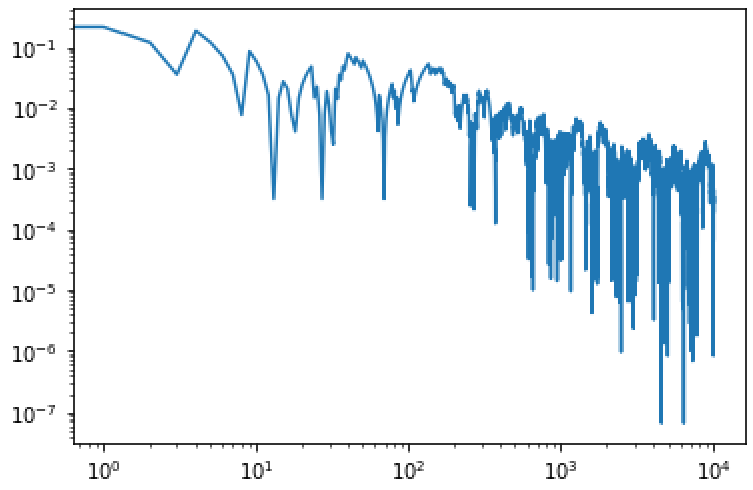
$$p(\mathcal{D}|\mathcal{M}) = \int_{\Theta} p(\mathcal{D}|\theta,\mathcal{M})p(\theta|\mathcal{M})d\theta \approx \frac{1}{N}\sum_{i=1}^{N} p(\mathcal{D}|\theta^{(i)},\mathcal{M})$$

> So the issue for Task 2 is somehow resolved.

In [2]:
```
# Estimating area of circle by MC.
# Very simple as p(z) is a uniform distribution defined on unit square.
# and f(z)=1 in circle and f(z)=0 out of circle.

N = 10000
s = np.random.uniform(size=(2,N))
x, y = s[0], s[1]
r2 = (x-0.5)**2 + (y-0.5)**2
m = r2 <= 0.25

f = plt.figure()
plt.plot(x[m], y[m], 'b.')
plt.plot(x[~m], y[~m], 'r.')
plt.title('Area: Accurate = {0:4.3f}, Approx. = {1:4.3f}'.format(np.pi*0.25
a = plt.axis('equal')
```



```
In [3]: area = np.cumsum(m)/np.arange(1.0,N+1.0)
        _=plt.loglog(np.abs(area-np.pi*0.25))
```



Importance Sampling

For a distribution $p(\mathbf{z})$ that is hard to sample, one can introduce a simpler distribution $q(\mathbf{z})$,

$$\int f(\mathbf{z})p(\mathbf{z})d\mathbf{z} = \int f(\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z})}q(\mathbf{z})d\mathbf{z} \approx \frac{1}{L}\sum_{l=1}^{L} f(\mathbf{z}^{(l)})\frac{p(\mathbf{z}^{(l)})}{q(\mathbf{z}^{(l)})}$$

But maybe the target distribution cannot even be easily normalized, one can do,

$$\int f(\mathbf{z})p(\mathbf{z})d\mathbf{z} \approx \frac{Z_q}{Z_p}\frac{1}{L}\sum_{l=1}^{L}\tilde{r}^{(l)} f(\mathbf{z}^{(l)})$$

where Z_q and Z_p are normalizing factors for $\tilde{q}$ and $\tilde{p}$, respectively, and

$$\tilde{r}^{(l)} = \frac{\tilde{p}(\mathbf{z}^{(l)})}{\tilde{q}(\mathbf{z}^{(l)})}$$

But one also has,

$$\frac{Z_p}{Z_q} = \int \frac{\tilde{p}(\mathbf{z})}{\tilde{q}(\mathbf{z})}q(\mathbf{z})d\mathbf{z} \approx \frac{1}{L}\sum_{l=1}^{L}\tilde{r}^{(l)}$$

so the integral simplifies to

$$\int f(\mathbf{z})p(\mathbf{z})d\mathbf{z} \approx \sum_{l=1}^{L} w^{(l)} f(\mathbf{z}^{(l)}), \quad w^{(l)} = \tilde{r}^{(l)} \Bigg/ \left(\sum_{l}\tilde{r}^{(l)}\right)$$

Accept-Reject Sampling

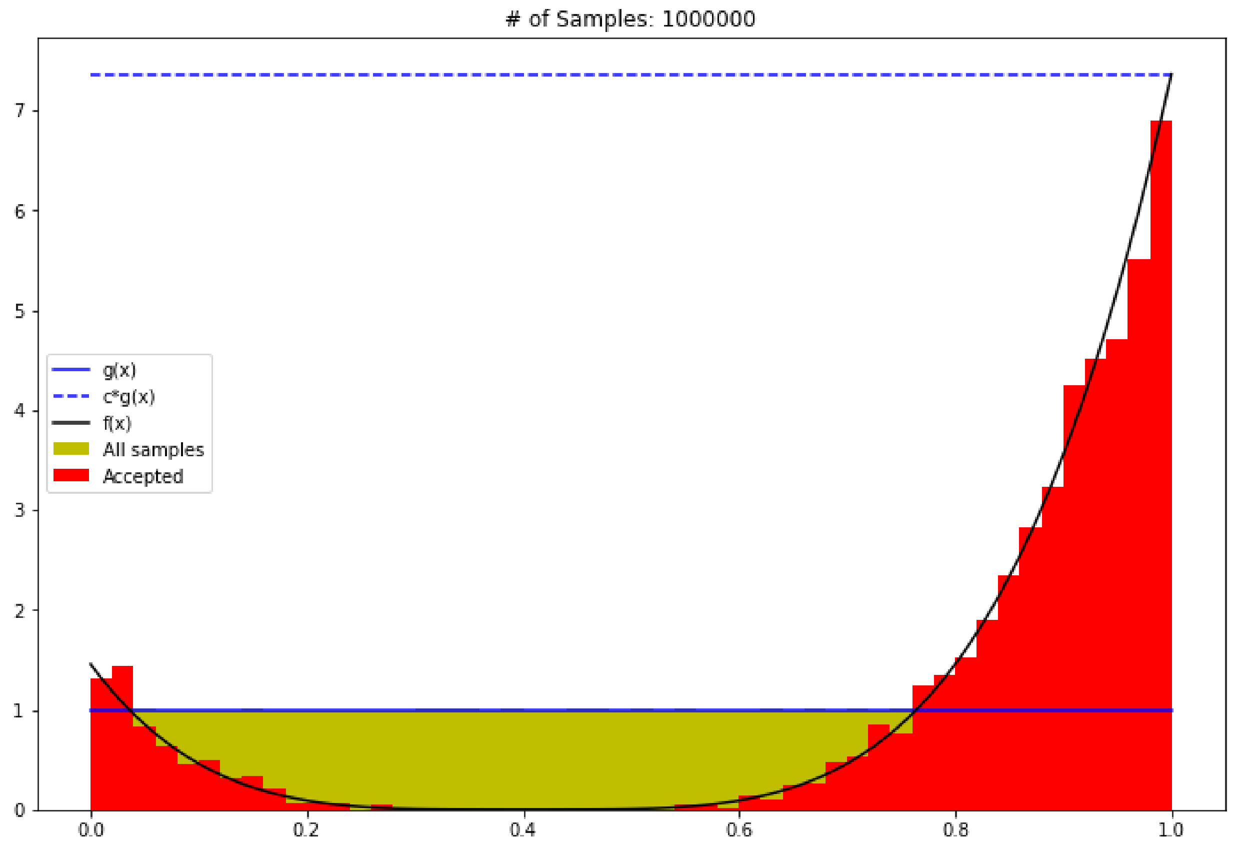
Given a difficult, maybe unnormalized, distribution $p(\mathbf{z})$:

- Pick $q(\mathbf{z})$ and a constant k such that $p(\mathbf{z}) < kq(\mathbf{z})$.
- Sample $\mathbf{z}_0$ from $q(\mathbf{z})$
- Sample $u \sim \mathrm{Uniform}[0, kq(\mathbf{z}_0)]$
- If $u \leq p(\mathbf{z}_0)$, accept $\mathbf{z}_0$, otherwise reject.

> The samples of $p(\mathbf{z})$ is represented by a carefully selected subset of samples of $q(\mathbf{z})$.

In [4]:
```
# Constants
power = 4
t = 0.4
Nsmp = 1000000
# Compute some coefficients to be used later
_pow = power+1
_sum = ((1-t)**_pow - (-t)**_pow) / _pow
c = 0.6**4/_sum

def targetPDF(x):
    return (x-t)**power

x = np.linspace(0, 1, 100)

# Proposal PDF g(x), uniform
g = np.ones_like(x)

# Target PDF f(x)
y = (x-t)**power / _sum

# Accept-Reject sampling
ss_all = []
ss_acc = []
for _i in range(Nsmp):
    _x = np.random.uniform(0, 1)
    _y = np.random.uniform(0, 1)
    ss_all.append(_x)
    if _y*c <= targetPDF(_x):
        ss_acc.append(_x)
```

In [5]:
```
f = plt.figure(figsize=(12,8))
plt.plot(x, g, 'b-', label='g(x)')
plt.plot(x, c*g, 'b--', label='c*g(x)')
plt.plot(x, y, 'k-', label='f(x)')
plt.hist(ss_all, bins=50, density=True, color='y', label='All samples')
plt.hist(ss_acc, bins=50, density=True, color='r', label='Accepted')
plt.legend()
_=plt.title('# of Samples: {0}'.format(Nsmp))
```



Markov Models

> Uses material from **[MLAPP]**

Markov Models

A **Markov chain** is a series of random variables $X_1, \ldots, X_N$ satisfying the *Markov property*:

> The future is independent of the past, given the present.
> $$p(x_n|x_1, \ldots, x_{n-1}) = p(x_n|x_{n-1})$$

A chain is **stationary** if the transition probabilities do not change with time.

Markov Models: Joint Distribution

If a sequence has the Markov property, then the joint distribution factorizes according to

$$p(x_1, \ldots, x_N) = p(x_1) \prod_{n=2}^{N} p(x_n|x_{n-1})$$

Markov Models: Transition Matrix

Suppose $X_t \in \{1, \ldots, K\}$ is discrete. Then, a stationary chain with N states can be described by a **transition matrix**, $A \in \mathbb{R}^{N \times N}$ where

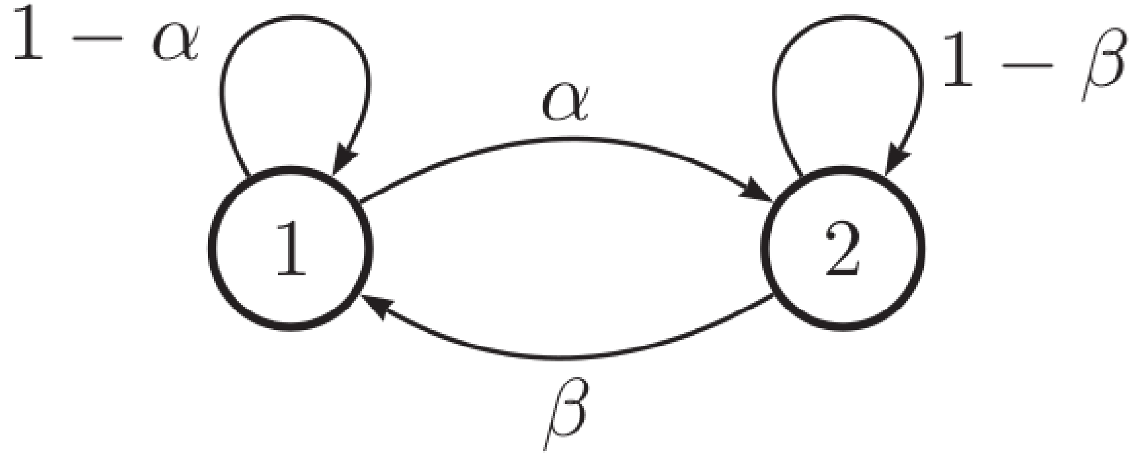
$$a_{ij} = p(X_t = j \mid X_{t-1} = i)$$

is the probability of transitioning from state i to j.

> Each row sums to one, $\sum_{j=1}^{K} A_{ij} = 1$, so A is a *row-stochastic matrix*.

Markov Models: Transition Diagram

Transitions between states can be represented as a graph:



Markov Models: State Vectors

Consider a row vector $x_t \in \mathbb{R}^{K\times 1}$ with entries $x_{tj} = p(X_t = j)$. Then,

$$\begin{aligned} p(X_t = j) &= \sum_{i=1}^{K} p(X_t = j \mid X_{t-1} = i) p(X_{t-1} = i) \\ &= \sum_{i=1}^{K} A_{ij} x_{t-1,i} \end{aligned}$$

Therefore, we conclude $x_t = x_{t-1}A$. Note $\sum_{j=1}^{K} x_{tj} = 1$.

Markov Models: Matrix Powers

Since $x_t = x_{t-1}A$, this suggests that in general,

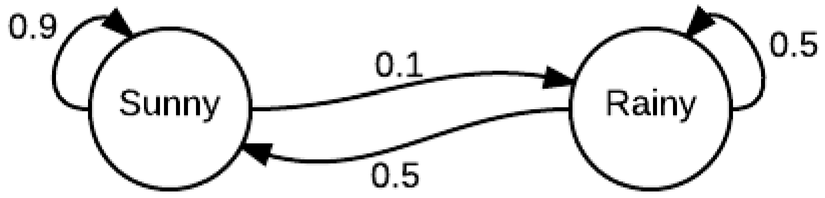
$$x_t = x_{t-1}A = x_{t-2}A^2 = \cdots x_0 A^t$$

If we know the initial state probabilities x_0, we can find the probabilities of landing in any state at time $t > 0$.

Example: Weather

Suppose the weather is either $R = \text{Rainy}$ or $S = \text{Sunny}$,

$$A = \begin{bmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{bmatrix}$$



Example: Weather

Suppose today is sunny, $x_0 = \begin{bmatrix} 1 & 0 \end{bmatrix}$. We can predict tomorrow's weather,

$$x_1 = x_0 A = \begin{bmatrix} 0.9 & 0.1 \end{bmatrix}$$

The weather over the next several days will be

$$\begin{aligned} x_2 = x_1 A &= \begin{bmatrix} 0.86 & 0.14 \end{bmatrix} \\ x_3 = x_2 A &= \begin{bmatrix} 0.844 & 0.156 \end{bmatrix} \\ x_4 = x_3 A &= \begin{bmatrix} 0.8376 & 0.1624 \end{bmatrix} \end{aligned}$$

Question: What happens to $x_0 A^n$ as $n \to \infty$?

Markov Chains: Stationary Distribution

If we ever reach a stage x where

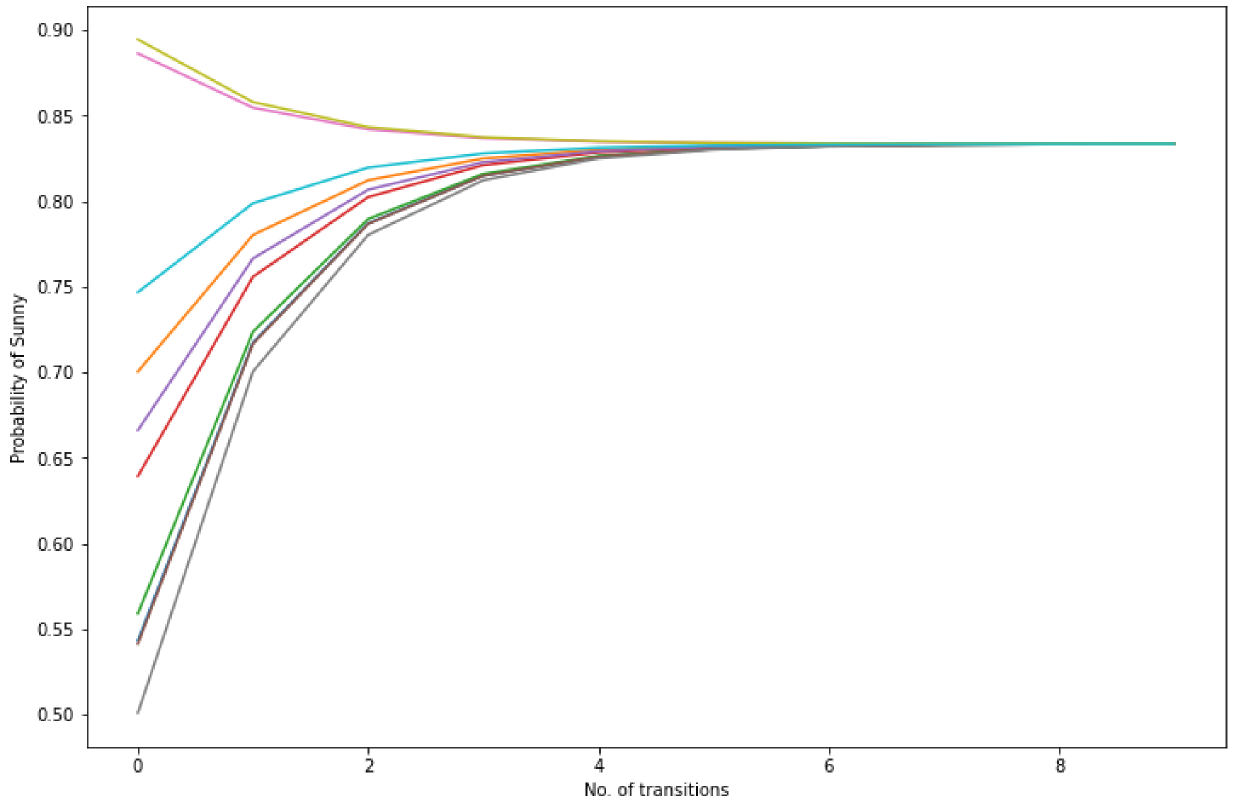
$$x = xA$$

then we have reached the **stationary distribution** of the chain.

- To find $x = v^T$, solve the eigenvalue problem $A^T v = v$
- Under certain conditions, the limiting distribution $\lim_{n \to \infty} x_0 A^n = x$
- Stationary distribution x does not depend on the starting state x_0

In [6]:
```
transition = np.array([[0.9,0.1], [0.5,0.5]])
_rand = np.random.uniform(0, 1, (10,))
states = np.vstack([_rand, 1.0-_rand]).T

_tmp = []
for _i in range(10):
    states = states.dot(transition)
    _tmp.append(states)
history = np.moveaxis(np.array(_tmp), 0, 2)
```

In [7]:
```
f = plt.figure(figsize=(12,8))
for _s in history:
    plt.plot(_s[0])
plt.xlabel('No. of transitions')
_=plt.ylabel('Probability of Sunny')
```



MCMC

Detailed Balance

- A stationary distribution $\pi(x)$ satisfies the **detailed balance** condition:
$$\pi(i)P(i,j) = \pi(j)P(j,i)$$

Proof:

$$\sum_{i=1}^{\infty} \pi(i)P(i,j) = \sum_{i=1}^{\infty} \pi(j)P(j,i) = \pi(j)\sum_{i=1}^{\infty} P(j,i) = \pi(j)$$

i.e. $\pi P = \pi$

- So it is possible to find the transition matrix from the stationary distribution, which could be the distribution to be sampled.
- Yet it is nontrivial to satisfy detailed balance. With a random guess $Q(x,y)$, one gets
$$\pi(i)Q(i,j) \neq \pi(j)Q(j,i)$$

Similar to the idea from accept-reject sampling, let's introduce a correction $\alpha(i,j)$, called acceptance rate, such that:

$$\pi(i)Q(i,j)\alpha(i,j) = \pi(j)Q(j,i)\alpha(j,i)$$

To make the above happen, α should satisfy,

$$\alpha(i,j) = \pi(j)Q(j,i)$$
$$\alpha(j,i) = \pi(i)Q(i,j)$$

And we have,

$$P(i,j) = Q(i,j)\alpha(i,j)$$

Basic MCMC

We do a loop of sampling:

- Sample x_* from $Q(x, x_{t-1})$
- Sample $u \sim \text{Uniform}[0,1]$
- If $u < \alpha(x_{t-1}, x_*) = \pi(x_*)Q(x_*, x_{t-1})$, then $x_t = x_*$, else $x_t = x_{t-1}$ The set of samples $\{x_0, x_1, \cdots, x_n\}$ are then the MCMC samples.

Issue: α could be low, resulting in many rejected samples.

Metropolis-Hastings Sampling

Note that α is insusceptible to scales:

$$\pi(i)Q(i,j)\alpha(i,j) = \pi(j)Q(j,i)\alpha(j,i) \Leftrightarrow \pi(i)Q(i,j)(5\alpha(i,j)) = \pi(j)Q(j,i)(5\alpha(j,i))$$

So one could define

$$\alpha(i,j) = \min\left\{\frac{\pi(j)Q(j,i)}{\pi(i)Q(i,j)}, 1\right\}$$

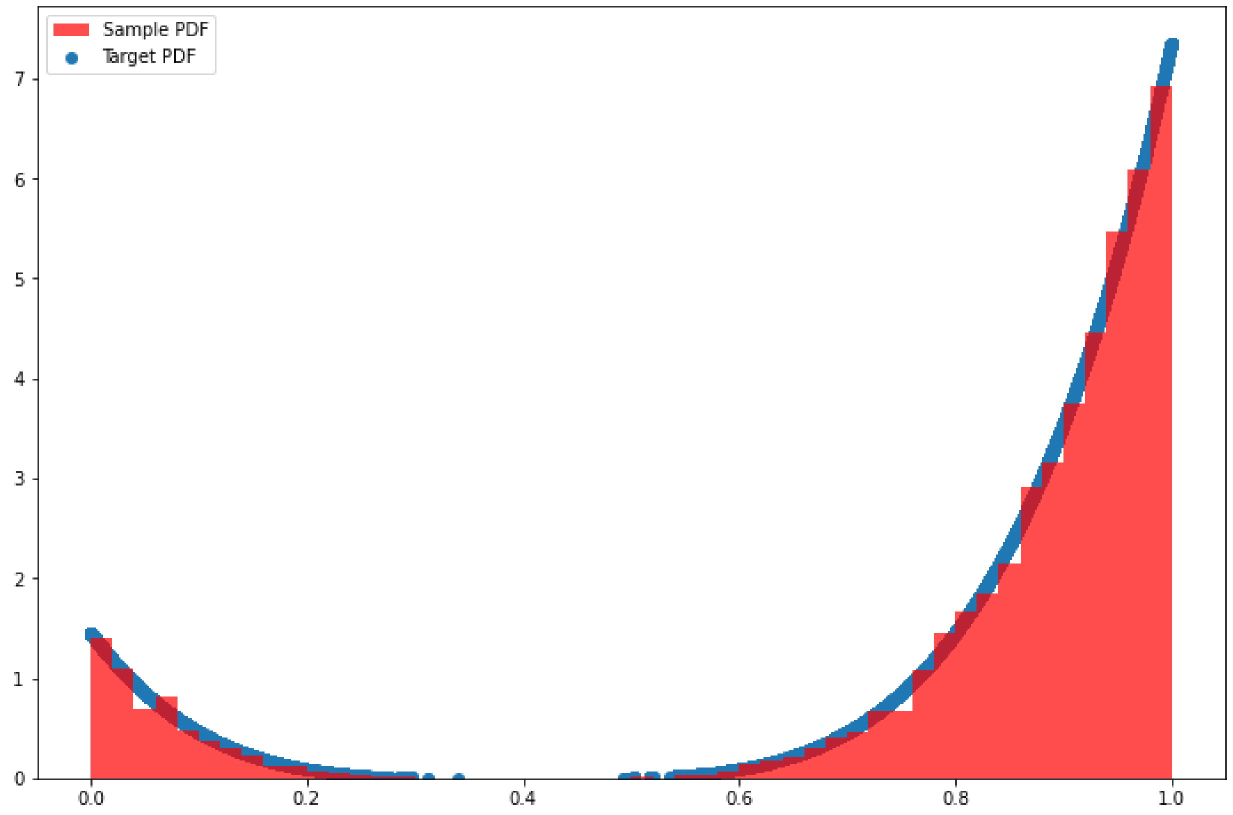
to obtain an acceptance rate α that is maximum possible.

Proof that the definition works:

$$\begin{aligned}\pi(i)Q(i,j)\alpha(i,j) &= \pi(i)Q(i,j)\min\left\{\frac{\pi(j)Q(j,i)}{\pi(i)Q(i,j)}, 1\right\} \\ &= \min\{\pi(j)Q(j,i), \pi(i)Q(i,j)\} = \pi(j)Q(j,i)\min\left\{1, \frac{\pi(i)Q(i,j)}{\pi(j)Q(j,i)}\right\} \\ &= \pi(j)Q(j,i)\alpha(j,i)\end{aligned}$$

In [8]:
```
# PDF for sanity check
# def targetPDF(x):
#     return ss.norm.pdf(x, loc=3, scale=2)

# The same PDF for the Accept-Reject example
def targetPDF(x):
    if x >= 0.0 and x <= 1.0:
        return (x-0.4)**4
    return 0.0

Nsmp = 100000
p = np.zeros((Nsmp,))
step = 1.0  # Step size for sample proposal
# step = 0.001  # Too small a step would take the sampler forever to reach
# step = 100.0  # Too large a step would result in strong oscillation in th

for _i in range(1, Nsmp):
    _q = p[_i - 1]
    # Propose sample
    _p = np.random.normal(loc=_q, scale=step, size=1)[0]
    # Acceptance model
    _a = min(1, (targetPDF(_p) / targetPDF(_q)))
    # Accept-Reject
    _u = np.random.uniform(0, 1)
    if _u < _a:
        p[_i] = _p
    else:
        p[_i] = _q
```

In [9]:
```
f = plt.figure(figsize=(12,8))
plt.hist(p, 50, density=True, facecolor='red', alpha=0.7, label='Sample PDF
plt.scatter(p, [56.818*targetPDF(_) for _ in p], label='Target PDF')
_=plt.legend()
```



Issues

Curse of dimensionality:

- For example, volume of high prod. region of high-dim Gaussian ~ σ^{-D}

- Computational cost: $\frac{\pi(j)Q(j,i)}{\pi(i)Q(i,j)}$, and the proposed sample could be rejected
- Sometimes it is even hard to get joint distribution.

One of the cures is the Gibbs sampling method.

Gibbs Sampling

Detailed balance requires:

$$\pi(i)P(i,j) = \pi(j)P(j,i)$$

Consider a 2D case $\pi(x_1, x_2)$ and two points: $A(x_1^{(1)}, x_2^{(1)})$ and $B(x_1^{(1)}, x_2^{(2)})$. One has

$$\pi(x_1^{(1)}, x_2^{(1)})\pi(x_2^{(2)}|x_1^{(1)}) = \pi(x_1^{(1)})\pi(x_2^{(1)}|x_1^{(1)})\pi(x_2^{(2)}|x_1^{(1)})$$
$$\pi(x_1^{(1)}, x_2^{(2)})\pi(x_2^{(1)}|x_1^{(1)}) = \pi(x_1^{(1)})\pi(x_2^{(2)}|x_1^{(1)})\pi(x_2^{(1)}|x_1^{(1)})$$

This is saying:

$$\pi(x_1^{(1)}, x_2^{(1)})\pi(x_2^{(2)}|x_1^{(1)}) = \pi(x_1^{(1)}, x_2^{(2)})\pi(x_2^{(1)}|x_1^{(1)})$$
$$\text{or } \pi(A)\pi(x_2^{(2)}|x_1^{(1)}) = \pi(B)\pi(x_2^{(1)}|x_1^{(1)})$$

Or, $\pi(x_2|x_1^{(1)})$ can be used to satisfy **detailed balance** of Markov chain!

Algorithm - 2D version

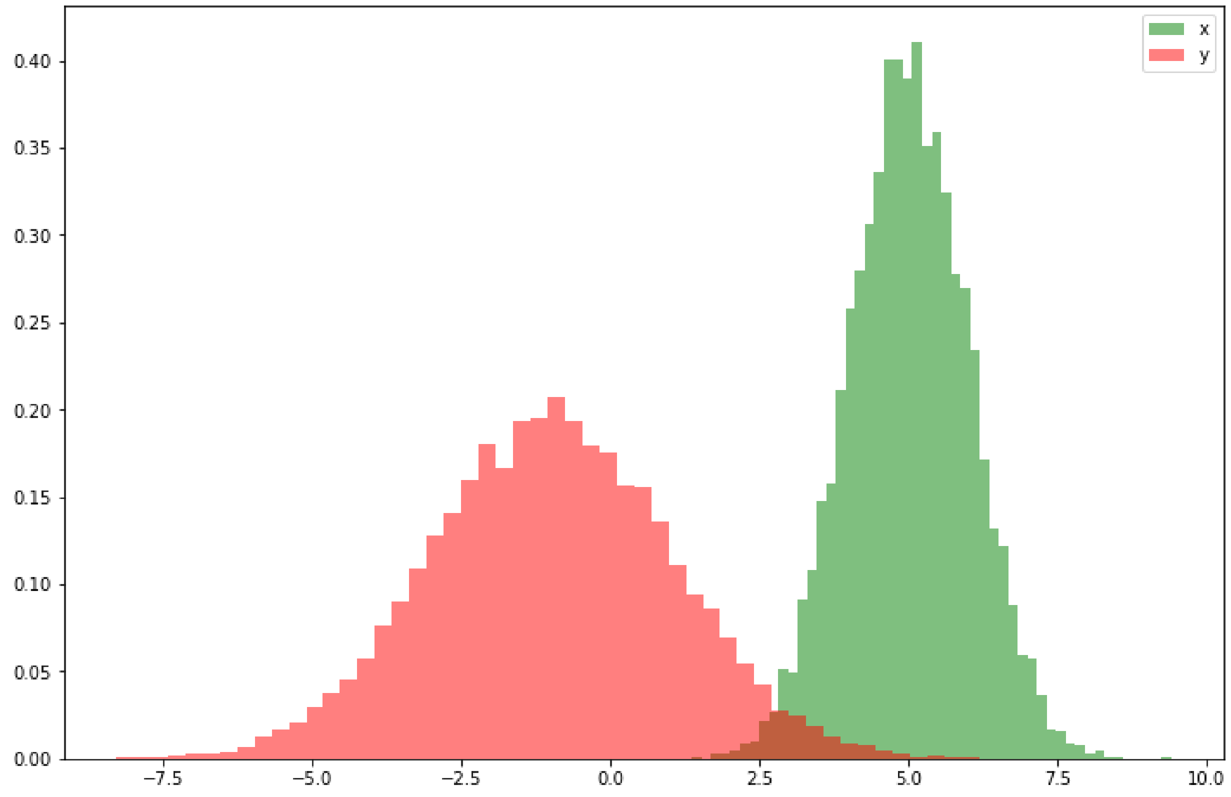
- Given stationary distribution $\pi(x_1, x_2)$
- Set initial states $x_1^{(0)}$ and $x_2^{(0)}$
- Loop for sampling:
- (a) Sample x_2^{t+1} from $P(x_2|x_1^{(t)})$
- (b) Sample x_1^{t+1} from $P(x_1|x_2^{(t)})$

Then, samples are generated from rotating the coordinate axes

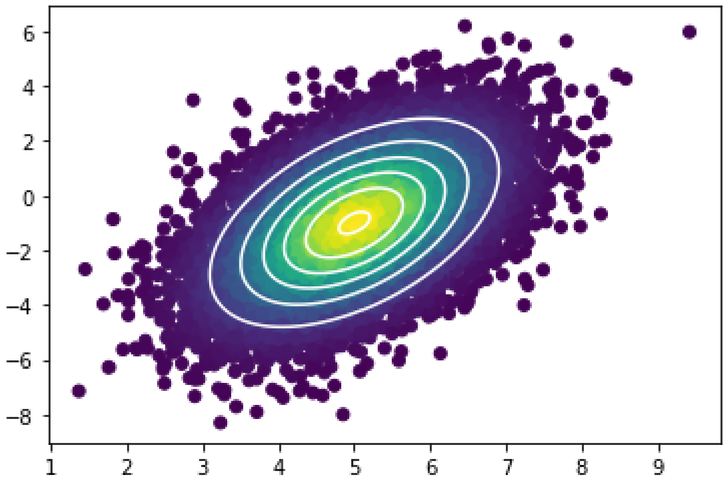
$$(x_1^{(1)}, x_2^{(1)}) \to (x_1^{(1)}, x_2^{(2)}) \to (x_1^{(2)}, x_2^{(2)}) \to \ldots \to (x_1^{(n)}, x_2^{(n)})$$

```
In [10]: rho = 0.5
         m1, m2 = 5, -1
         s1, s2 = 1, 2
         target = ss.multivariate_normal(mean=[m1,m2], cov=[[s1**2,rho*s1*s2],[rho*s

         def p_ygivenx(x, m1, m2, s1, s2):
             return np.random.normal(m2 + rho * s2/s1 * (x-m1), np.sqrt((1 - rho**2)
         def p1(s):
             return p_ygivenx(s, m2, m1, s2, s1)  # x given y
         def p2(s):
             return p_ygivenx(s, m1, m2, s1, s2)  # y given x

         Nsmp = 10000
         _tmp = []
         _y = 0.0
         for _i in range(Nsmp):
             _x = p1(_y)  # Fix y and sample x
             _y = p2(_x)  # Fix x and sample y
             _z = target.pdf([_x,_y])  # Target PDF
             _tmp.append([_x,_y,_z])
         p = np.array(_tmp).T
```

```
In [11]: f = plt.figure(figsize=(12,8))
         plt.hist(p[0], 50, density=True, facecolor='green', alpha=0.5, label='x')
         plt.hist(p[1], 50, density=True, facecolor='red', alpha=0.5, label='y')
         _=plt.legend()
```



In [12]:
```python
Nb = 1000
Ng = 101
x = np.linspace(2, 8, Ng)
y = np.linspace(-7, 5, Ng)
X, Y = np.meshgrid(x, y)
Z = target.pdf(np.vstack([X.reshape(-1), Y.reshape(-1)]).T).reshape(Ng, Ng)
plt.scatter(p[0][Nb:], p[1][Nb:], c=p[2][Nb:])
_=plt.contour(X, Y, Z, colors='w', linewidth=2)
```



In [13]:
```python
Nb = 1000
M1 = np.mean(p[0][Nb:])
M2 = np.mean(p[1][Nb:])
dx = p[0][Nb:]-M1
dy = p[1][Nb:]-M2
S1 = np.mean(dx**2)
S2 = np.mean(dy**2)
RR = np.mean(dx*dy)
print([M1, M2, S1, S2, RR])
```

```
[5.016763907726373, -0.9874205825714542, 0.9963200421004105, 4.0931138181
43411, 1.021406979986205]
```

There are many other methods

One example - auxiliary variable method:

- Introduce a new variable $\mathbf{u}$, $\mathbf{z}, \mathbf{u} \sim p(\mathbf{z}, \mathbf{u})$, so that
$$\int f(\mathbf{z})p(\mathbf{z})d\mathbf{z} = \int\int f(\mathbf{z})p(\mathbf{z},\mathbf{u})d\mathbf{z}d\mathbf{u} = \frac{1}{N}\sum_{i=1}^{N} f(x^{(i)})$$
- Efficient if $p(\mathbf{z}|\mathbf{u})$ or $p(\mathbf{u}|\mathbf{z})$ is easy to sample, or if $p(\mathbf{z}, \mathbf{u})$ is easy to explore.

One specific method is **Hamiltonian Monte Carlo**

- Let $p(\mathbf{z}) \propto \exp[-E(\mathbf{z})]$, and

$$p(\mathbf{z}, \mathbf{u}) \propto \exp[-E(\mathbf{z})] \exp(-\mathbf{u}^T \mathbf{u}/2) \equiv \exp[-H(\mathbf{z}, \mathbf{u})]$$

- $E(\mathbf{z})$ and $\frac{1}{2}\mathbf{u}^T \mathbf{u}$ can be considered potential and kinetic energy, respectively. Due to energy conservation,

$$p(\mathbf{z}, \mathbf{u}) = p(\mathbf{z}', \mathbf{u}')$$

- So one can solve Hamitonian dynamics to do sampling!
  - A practical variant: HMC with No U-Turns Sampling (NUTS)
  - A challenge: Hamitonian dynamics requires the evaluation of $\frac{\partial E}{\partial \mathbf{z}}$ - not always available.
  - Not a problem for gradient-enabled frameworks, e.g. TensorFlow, pyTorch, FluxML, etc.