Bayesian Methods for Machine Learning

Lecture 6 - Markov Chain Monte Carlo

Simon Leglaive

CentraleSupélec

Let $\mathbf{x} \in \mathcal{X}$ and $\mathbf{z} \in \mathcal{Z}$ denote the observed and latent random variables, respectively.

In Bayesian methods, it's all about posterior computation:

$$p(\mathbf{z}|\mathbf{x};\theta) = \frac{p(\mathbf{x}|\mathbf{z};\theta)p(\mathbf{z};\theta)}{p(\mathbf{x};\theta)} = \frac{p(\mathbf{x}|\mathbf{z};\theta)p(\mathbf{z};\theta)}{\int p(\mathbf{x}|\mathbf{z};\theta)p(\mathbf{z};\theta)d\mathbf{z}}$$

- Easy to compute for conjugate prior.
- Hard for many models were conjugacy does not hold.

For simplicity of notations, we denote the parameters of the likelihood and prior by θ even though they are generally different.

More precisely, we are very often interested in computing expectations of some function g of the latent variable, taken with respect to the posterior distribution:

$$\mathbb{E}_{p(\mathbf{z}|\mathbf{x};\theta)}[f(\mathbf{z})] = \int f(\mathbf{z})p(\mathbf{z}|\mathbf{x};\theta)d\mathbf{z}.$$

For instance:

- Posterior mean: $\mathbb{E}_{p(\mathbf{z}|\mathbf{x};\theta)}[\mathbf{z}]$.
- Posterior expected loss (see Lecture 1): $\mathcal{L}(\hat{\mathbf{z}}) = \mathbb{E}_{p(\mathbf{z}|\mathbf{x};\theta)}[\ell(\hat{\mathbf{z}}, \mathbf{z})]$
- Predictive posterior distribution (see Lecture 1): $p(\mathbf{x}_{\text{new}}|\mathbf{x};\theta) = \mathbb{E}_{p(\mathbf{z}|\mathbf{x};\theta)}[p(\mathbf{x}_{\text{new}}|\mathbf{z};\theta_x)]$
- E-step of the EM algorithm (see Lecture 3 part 2): $Q(\theta, \theta_{\text{old}}) = \mathbb{E}_{p(\mathbf{z}|\mathbf{x};\theta_{\text{old}})}[\ln p(\mathbf{x}, \mathbf{z};\theta)]$.

We need approximate inference techniques when exact posterior computation is infeasible:

- Variational inference
- Markov Chain Monte Carlo (focus of today)

Monte Carlo

The Monte Carlo method, first introduced in (Metropolis and Ulam, 1949), is a stochastic approach to computing expectations of functions of random variables.

Let

- $p : \mathbb{R}^D \mapsto \mathbb{R}_+$ be a probability density function (pdf) over a random vector $\mathbf{x} \in \mathbb{R}^D$,
- $f : \mathbb{R}^D \mapsto \mathbb{R}$ be an arbitrary function.

The expected value of f over p is defined as:

$$\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})] = \int_{\mathbb{R}^D} f(\mathbf{x}) p(\mathbf{x}) d\mathbf{x}.$$

The Monte Carlo method approaches $\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})]$ with a sample average:

$$\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})] \approx \hat{f}_N = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i), \qquad \mathbf{x}_i \overset{i.i.d}{\sim} p(\mathbf{x}),$$

where $\mathbf{x}_i \overset{i.i.d}{\sim} p(\mathbf{x})$ means $\mathbf{x}_i$ is independently and identically drawn from $p(\mathbf{x})$.

Video credits: https://t.co/x393c5aMAZ?amp=1=

Properties of the Monte Carlo estimate

- The strong law of large numbers (LLN) states that the sample average converges almost surely (a.s.) to the expected value:

$$\hat{f}_N \xrightarrow{\text{a.s.}} \mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})] \qquad \text{as} \qquad N \to +\infty.$$

  Convergence *almost surely* means that:

$$Pr\left\{\lim_{N\to+\infty} \hat{f}_N = \mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})]\right\} = 1.$$

- Moreover, the estimator is unbiased:

$$\mathbb{E}_{p(\mathbf{x})}\left[\hat{f}_N\right] = \mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})]$$

- While fluctuations around the true expectation are inevitable, we wish these fluctuations to be small.

  This is guaranteed by the central limit theorem (CLT).

  For sufficiently large sample size N, the fluctuations are approximately Gaussian distributed:

$$\hat{f}_N \sim \mathcal{N}\left(\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})], \frac{1}{N}\mathbb{V}_{p(\mathbf{x})}[f(\mathbf{x})]\right),$$

  where $\mathbb{V}_{p(\mathbf{x})}[f(\mathbf{x})] = \mathbb{E}_{p(\mathbf{x})}\left[\left(f(\mathbf{x}) - \mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})]\right)^2\right]$.

  It is important to note that the variance of the Monte Carlo estimate decreases as $\frac{1}{N}$ independently of the dimensionality of $\mathbf{x}$.

- This last point suggests that, at least in principle, with a reasonable number of samples one can compute approximate solutions for random vectors $\mathbf{x}$ of arbitrary dimension.
- All we need is independent and identically distributed samples.
- The difficulty with the Monte Carlo approach, however, is precisely in generating independent samples from a target distribution (think of the posterior distribution).
- Various Monte Carlo methods aim to provide techniques for this.

Two potential reasons for the Monte Carlo method to fail:

- sampling from $p(\mathbf{x})$ is difficult or impossible,
- the density $p(\mathbf{x})$ is low for regions where $f(\mathbf{x})$ is high-valued, and vice-versa, so that the terms $f(\mathbf{x}_i)$ used to build the sample average

  $$\hat{f}_N = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i), \qquad \mathbf{x}_i \overset{i.i.d}{\sim} p(\mathbf{x})$$

  are most of the time very low. Thus the effective sample size can be much smaller than the apparent sample size N.

Importance sampling

Let's assume that sampling from $p(\mathbf{x})$ is difficult or impossible.

The word "sampling" is a bit misleading as this method does not provide samples from the target distribution, it can only be used to approximate expectations.

We have

$$\begin{aligned}\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})] &= \int_{\mathbb{R}^D} f(\mathbf{x})p(\mathbf{x})d\mathbf{x} \\ &= \int_{\mathbb{R}^D} f(\mathbf{x})\frac{p(\mathbf{x})}{q(\mathbf{x})}q(\mathbf{x})d\mathbf{x} \\ &= \mathbb{E}_{q(\mathbf{x})}\left[f(\mathbf{x})\frac{p(\mathbf{x})}{q(\mathbf{x})}\right],\end{aligned}$$

for some arbitrary distribution $q(\mathbf{x})$ such that $q(\mathbf{x}) > 0$ when $f(\mathbf{x})p(\mathbf{x}) \neq 0$.

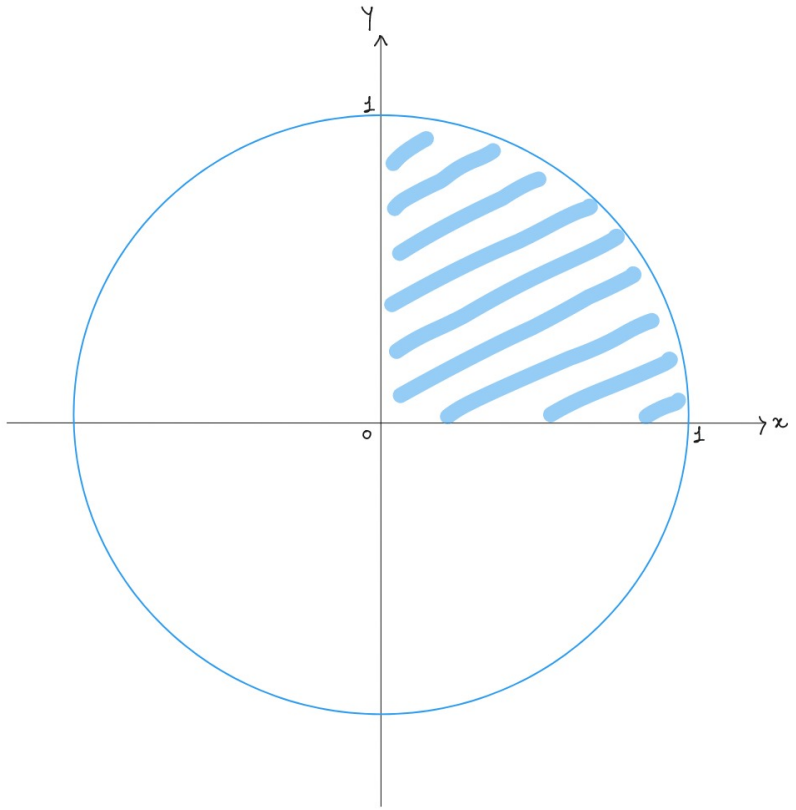
We can therefore reformulate the approximation as:

$$\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})] \approx \tilde{f}_N = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i)\frac{p(\mathbf{x}_i)}{q(\mathbf{x}_i)}, \qquad \mathbf{x}_i \overset{i.i.d}{\sim} q(\mathbf{x}).$$

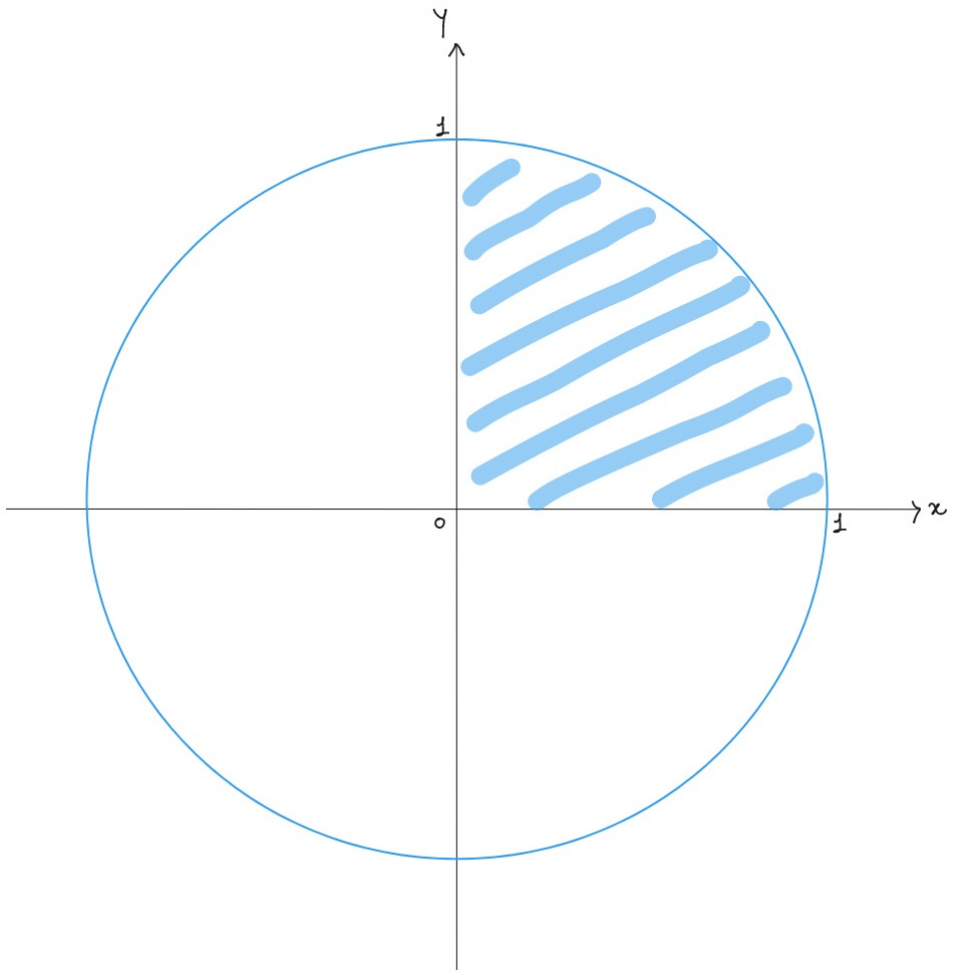
The quantities $p(\mathbf{x}_i)/q(\mathbf{x}_i)$ are known as importance weights. This method is called importance sampling, and given a careful choice of $q(\mathbf{x})$ can allow easier sampling and help reduce the variance of the estimator.

Example: approximating π with Monte Carlo

Consider a unit square and a circle arc joining two opposite corners of the square.



The area of a circle with radius 1 is π, so the area of the quarter circle is $\pi/4$.

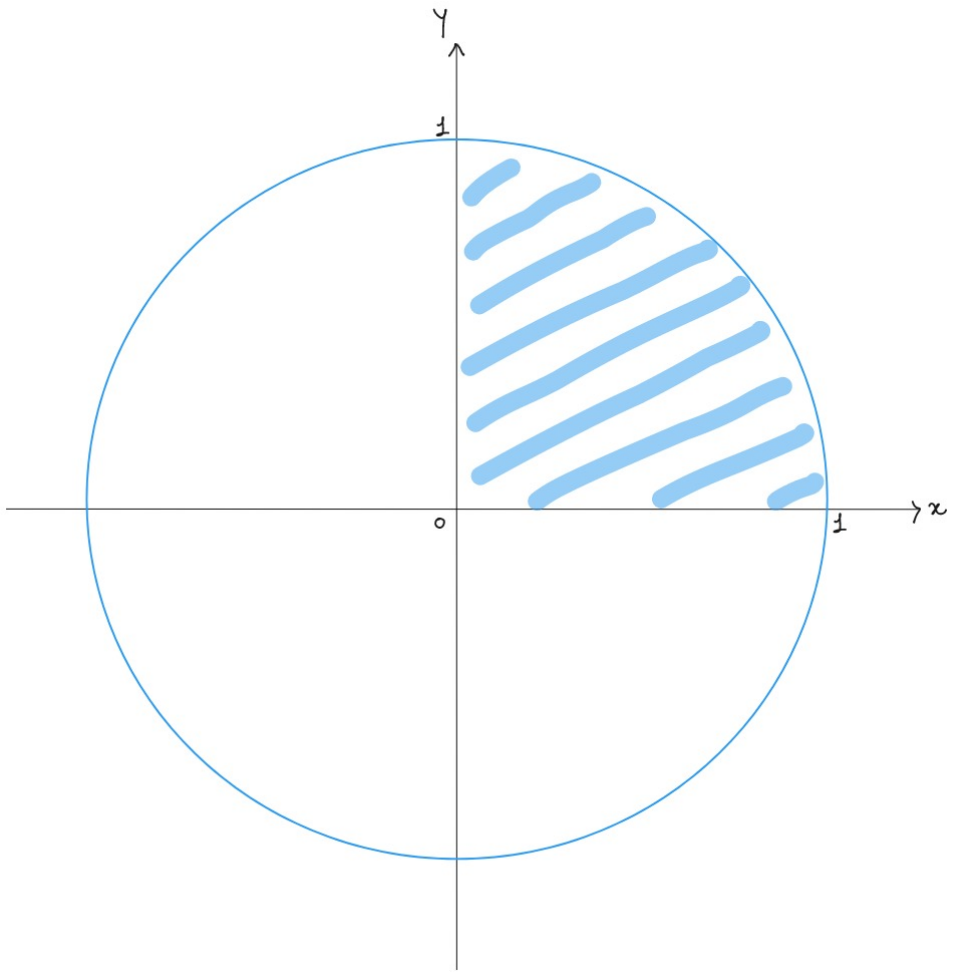


We have

$$\frac{\pi}{4} = \int_0^1 \int_0^1 f(x,y)dxdy,$$

with

$$f(x,y) = \begin{cases} 1, & x^2 + y^2 \leq 1, \\ 0, & \text{otherwise.} \end{cases}$$



It can be rewritten as:

$$\frac{\pi}{4} = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x,y)p(x,y)dxdy$$

with

$$f(x,y) = \begin{cases} 1, & x^2 + y^2 \leq 1, \\ 0, & \text{otherwise.} \end{cases}$$

and

$$p(x,y) = \begin{cases} 1, & 0 \leq x \leq 1 \text{ and } 0 \leq y \leq 1, \\ 0, & \text{otherwise.} \end{cases}$$

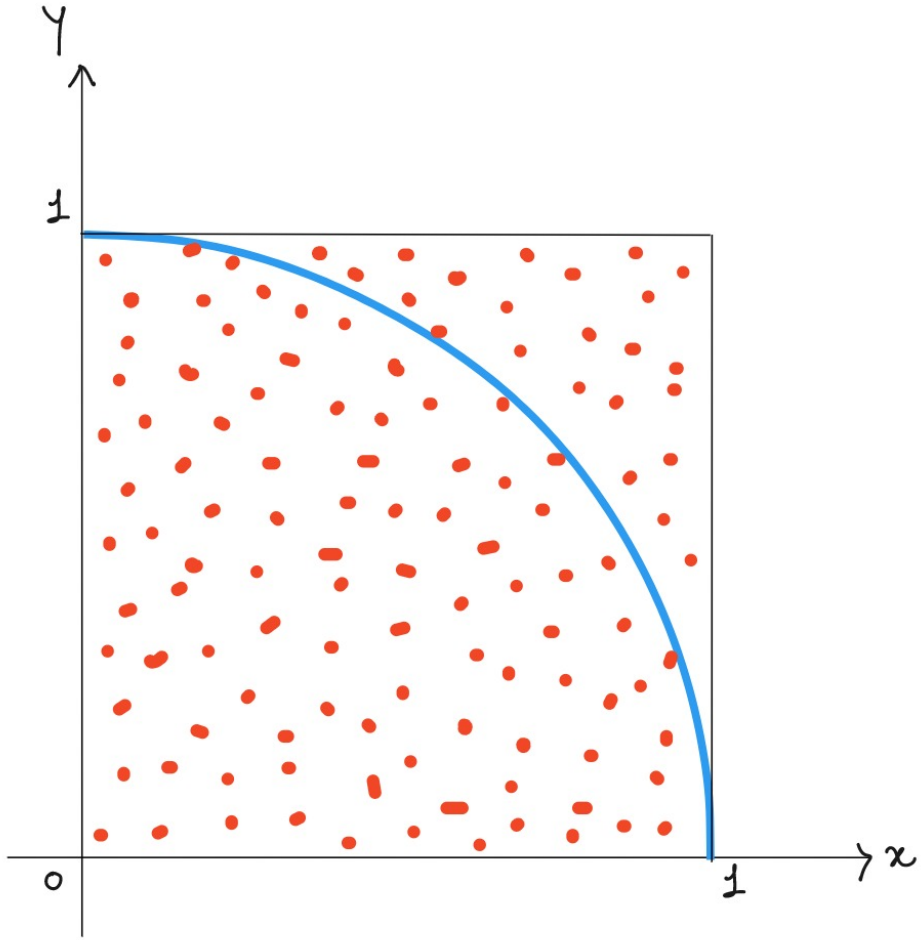
The numerical value of π can be interpreted as the expected value of $f(\mathbf{x})$ with $\mathbf{x}$ being a two-dimensional random vector uniformly distributed over the unit square:

$$\frac{\pi}{4} = \mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})], \qquad p(\mathbf{x}) = \mathcal{U}(\mathbf{x}; [0,1] \times [0,1]).$$

This allows us to approximate $\pi/4$ as:

$$\frac{\pi}{4} \approx \hat{f}_N = \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}_i), \qquad \mathbf{x}_i \overset{i.i.d}{\sim} \mathcal{U}(\mathbf{x}; [0,1] \times [0,1]).$$

To approximate $\pi/4$, we need to draw N uniformly-distributed samples across the unit square and count the proportion of those points which fall into the quarter circle.



Activity #1

Using numpy, implement this method to approximate π and observe the accuracy for different numbers of samples.

Sampling from probability distributions

In the following, for simplicity, we consider scalar random variables.

"While computer-assisted pseudo-random number generation is computationally cheap and fast, it relies on technology which might not be available in the event of a zombie apocalypse."

A Ballistic Monte Carlo Approximation of π

Vincent Dumoulin*
Département d'informatique et de recherche opérationnelle
Université de Montréal

Félix Thouin†
Département de physique
Université de Montréal
(Dated: April 10, 2014)

We compute a Monte Carlo approximation of π using importance sampling with shots coming out of a Mossberg 500 pump-action shotgun as the proposal distribution. An approximated value of 3.131 is obtained, corresponding to a 0.33% error on the exact value of π. To our knowledge, this represents the first attempt at estimating π using such method, thus opening up new perspectives towards computing mathematical constants using everyday tools.

V. Dumoulin and F. Thouin, "A Ballistic Monte Carlo Approximation of π", arXiv:1404.1499v2, 2014.

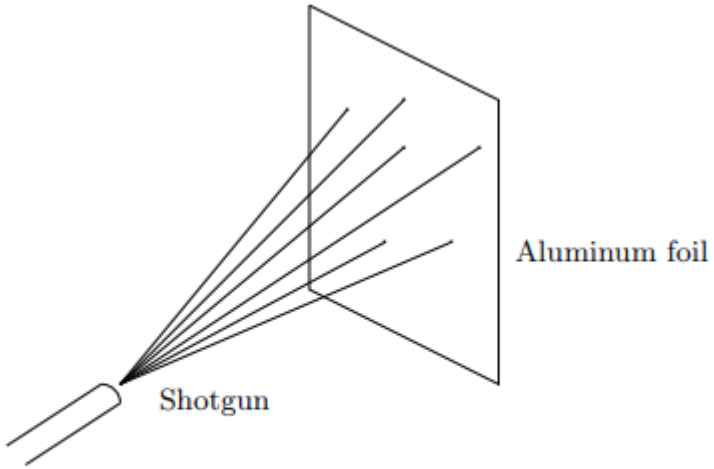


FIG. 2. Experimental setup. The shotgun is pointed at an aluminum foil to record its shot pattern. Samples are iteratively drawn by shooting the shotgun.

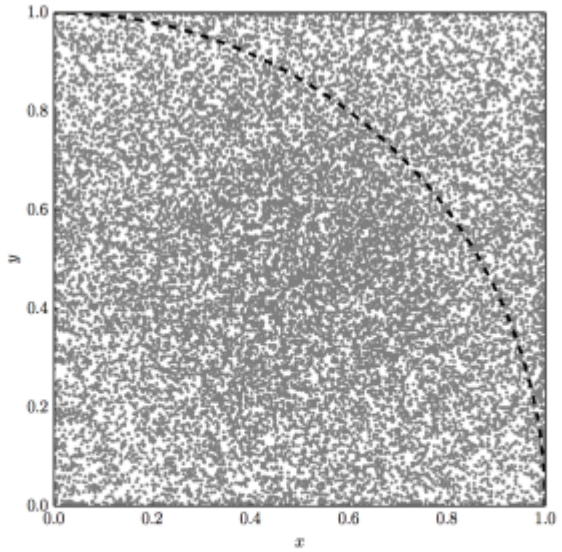


FIG. 4. Remaining 20857 samples used to approximate π. Positions were normalized to be in $[0, 1] \times [0, 1]$. A quarter circle was drawn for visual reference.

Of course, do not take this too seriously...

Pseudo-random number generators

- A Linear Congruential Generator (LCG) is defined by a seed x_0 and the recurrence relation:

$$x_{n+1} = ax_n + c \pmod{M}, \qquad n \geq 0.$$

  If the a and c are chosen carefully, the samples will be roughly uniformly distributed between 0 and $M-1$.

- Example proposed by Lewis, Goodman, and Miller (1969) for the IBM System/360:

$$x_{n+1} = 16807x_n \pmod{2^{31} - 1}.$$

- With LCG, we can generate random integers (approximately) uniformly distributed between 0 and $M-1$. Assume $M = 2^{64}$, if we simply divide by $M - 1$ we can generate double-precision floating-point numbers uniformly distributed in $[0, 1]$.

In the following, we assume that we have access to a uniform random number generator of real numbers in $[0, 1]$, that we denote by $\mathcal{U}([0, 1])$.

Given this random number generator, how can we sample from more complex distributions?

Example: The Box–Muller transform, by George Edward Pelham Box and Mervin Edgar Muller, is a random number sampling method for generating pairs of independent, standard, normally distributed (zero expectation, unit variance) random numbers, given a source of uniformly distributed random numbers.

Inverse transform sampling

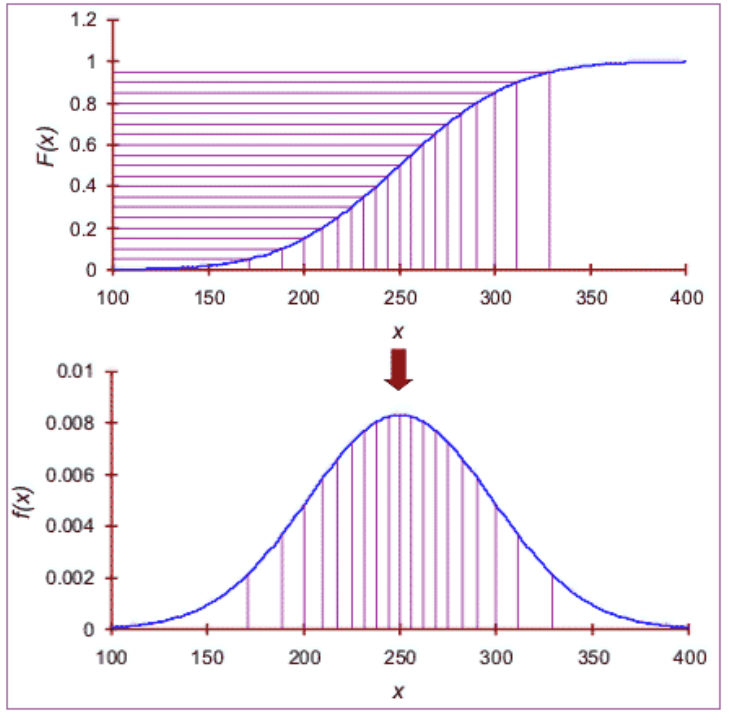
Inverse transform sampling is a method for generating samples from any probability distribution given the inverse of its cumulative distribution function (CDF).

The CDF of a continuous random variable x with pdf $p(x)$ is defined by:

$$F_x(x_0) = Pr(x \leq x_0) = \int_{-\infty}^{x_0} p(x)dx.$$

Inverse transform sampling consists in:

1. Generating $u \sim \mathcal{U}([0, 1])$
2. Computing $x = F_x^{-1}(u)$.



For several common distributions (including the Gaussian), we cannot compute the inverse CDF analytically.
For the Gaussian, we have extremely accurate approximations using moderate-degree polynomials.

Image credit: http://www.epixanalytics.com/modelassist/AtRisk/images/9/figure43.gif

Density transformation

Let x be a real-valued random variable with pdf $p(x)$.

Let $g : \mathbb{R} \mapsto \mathbb{R}$ be an invertible mapping.

The pdf $q(y)$ of $y = g(x)$ is given by:

$$q(y) = p(g^{-1}(y)) \left| \frac{d}{dy} g^{-1}(y). \right|$$

The derivative of g^{-1} is denoted as the Jacobian and the absolute value ensures the positivity of the density.

In the multivariate setting, we consider the determinant of the Jacobian.

```
mu = 1
sigma = 0.1

x = np.random.randn() # ~N(0,1)
y = mu + sigma*x # ~N(mu, sigma^2)
```

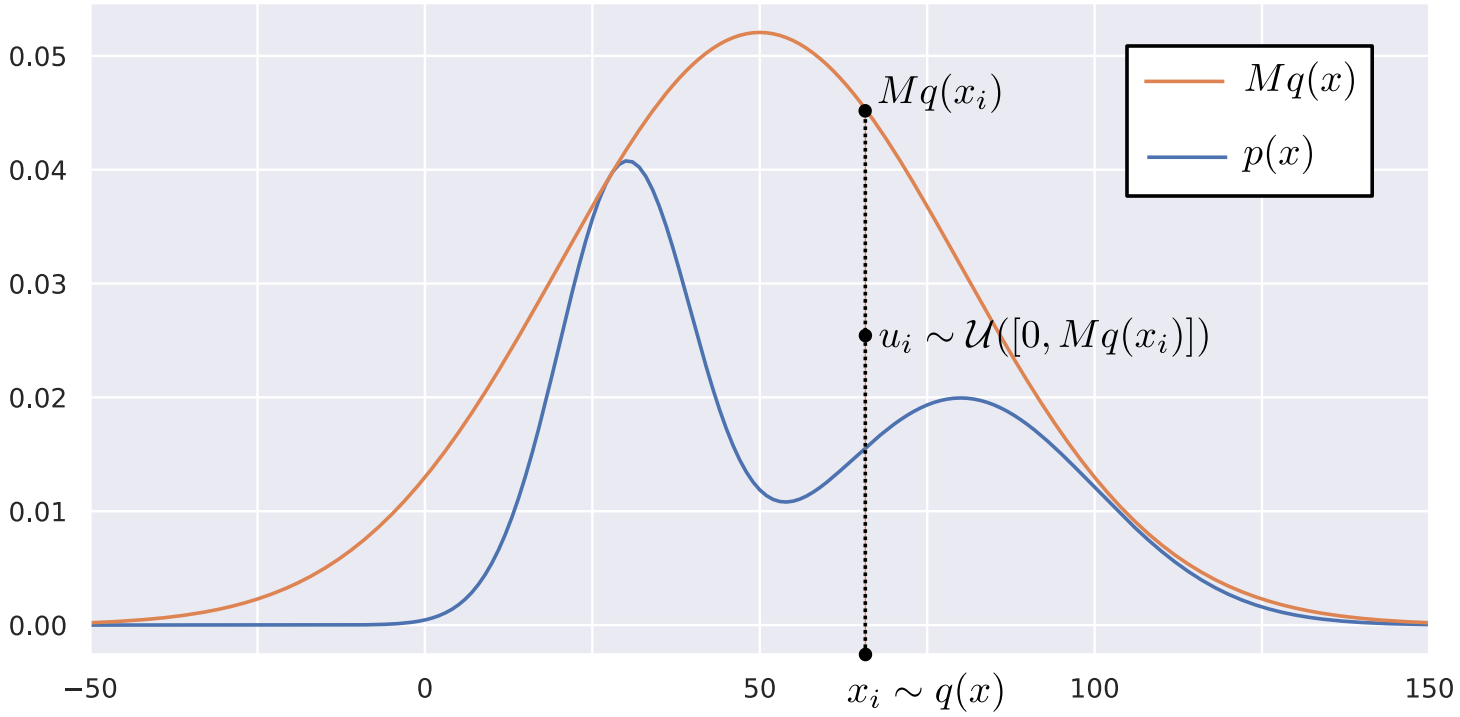
Rejection sampling

Rejection sampling is a technique for indirectly sampling from a target distribution $p(x)$ by sampling from a proposal distribution $q(x)$.

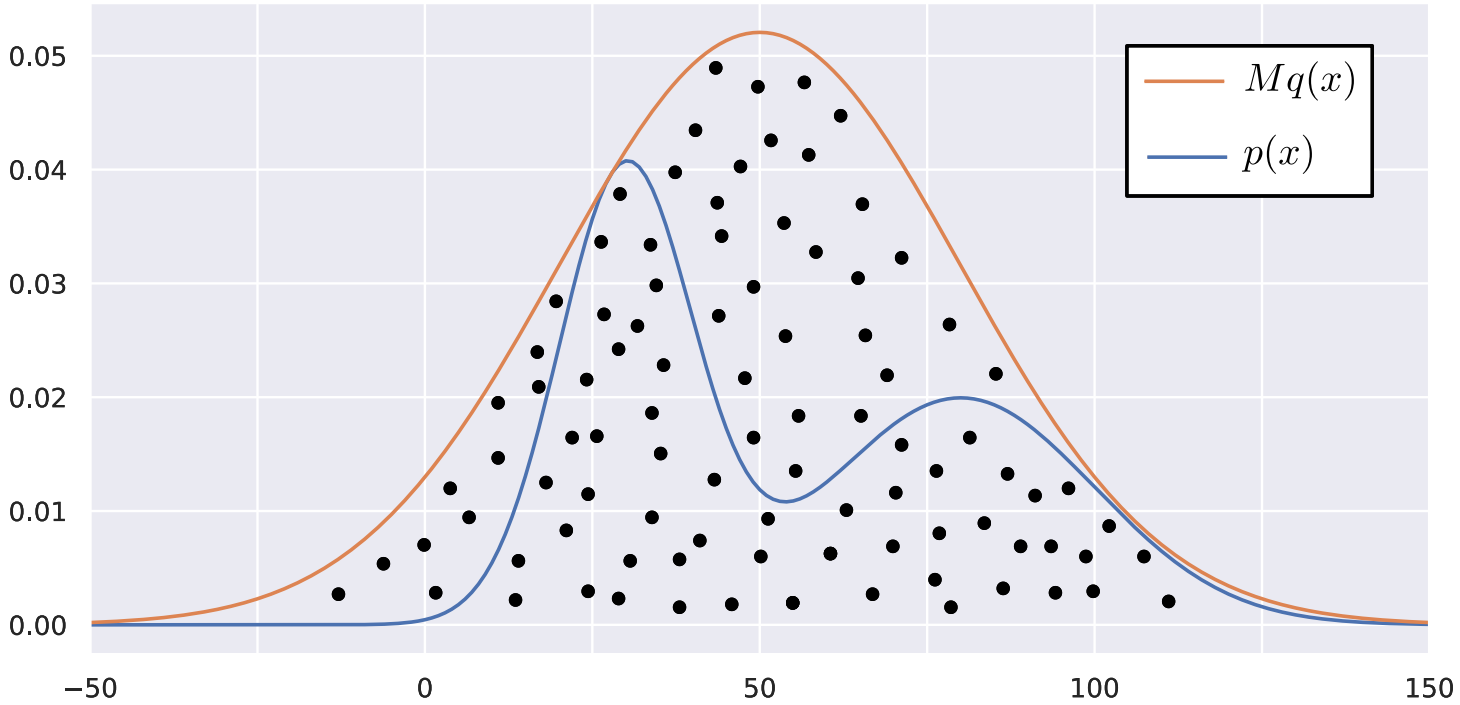
We reject some of the generated samples to compensate for the fact that $q(x) \neq p(x)$.

The algorithm for rejection sampling goes as follows for $i = 1, ..., N$:

- Sample x_i independently from $q(x)$.
- Sample u_i from from the uniform distribution over $[0, Mq(x_i)]$, where M is a positive number that guarantees $p(x) \leq Mq(x)$ for all x.

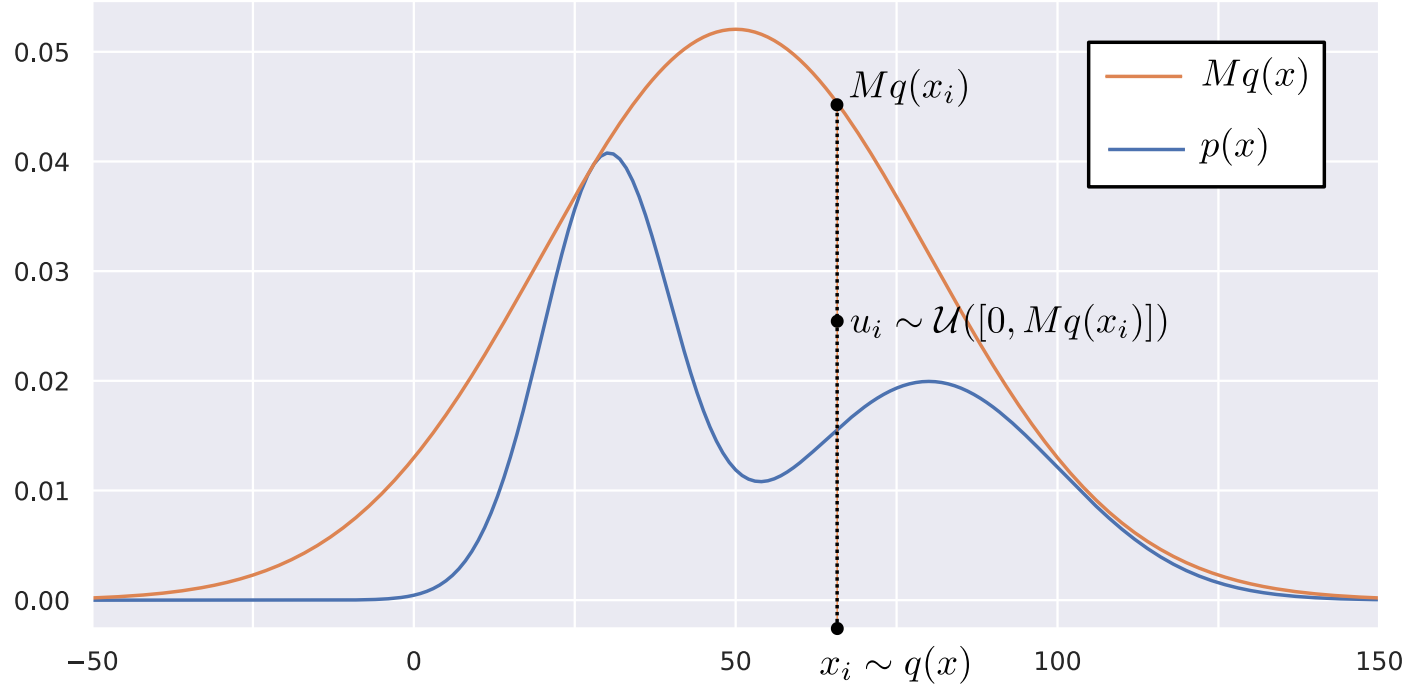


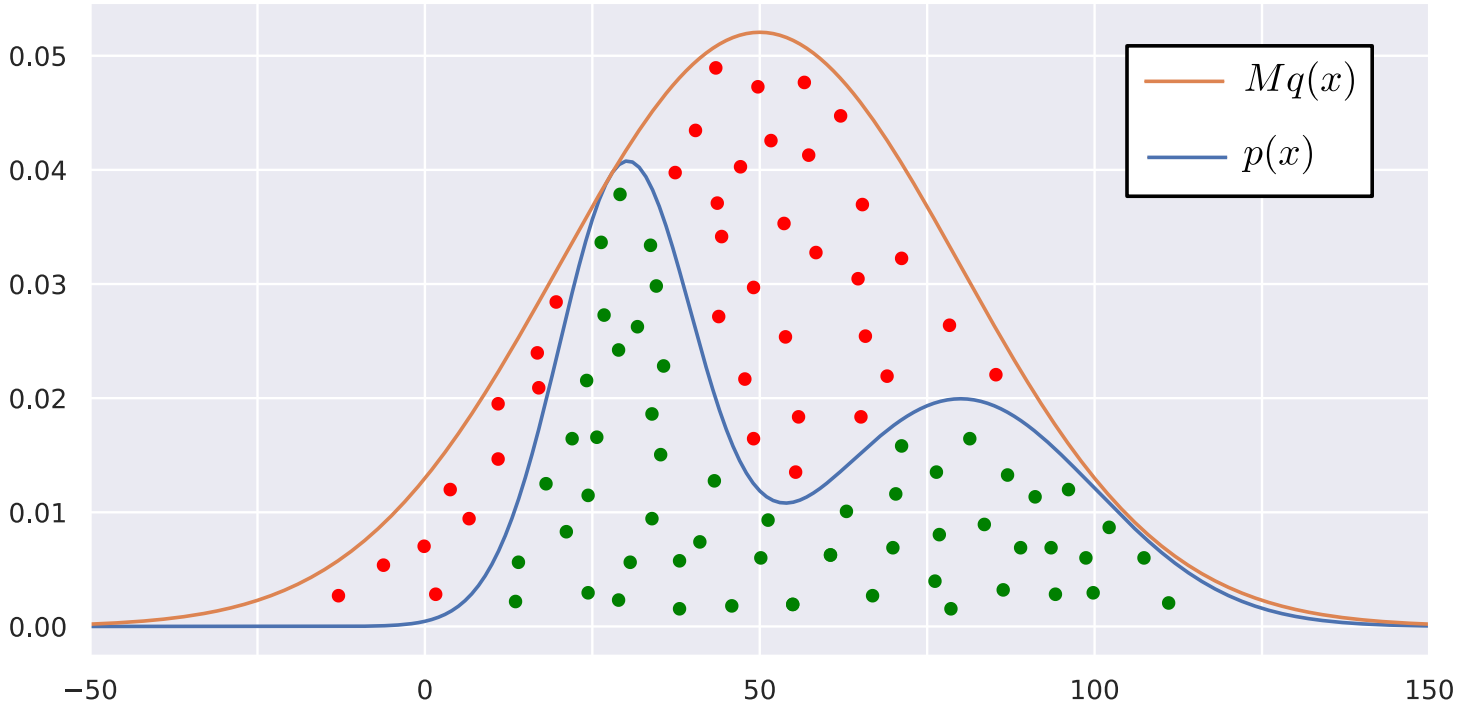
The sampled pairs (x_i, u_i) are uniformly distributed under the curve of $Mq(x)$, i.e. on the set $\{(x, u) : 0 \leq u \leq Mq(x)\}$.



- If $u_i \leq p(x_i)$, i.e. u_i is under the curve of $p(x)$, we accept the sample, otherwise we reject it.

  The corresponding acceptance probability is equal to the ratio $p(x_i)/Mq(x_i)$.





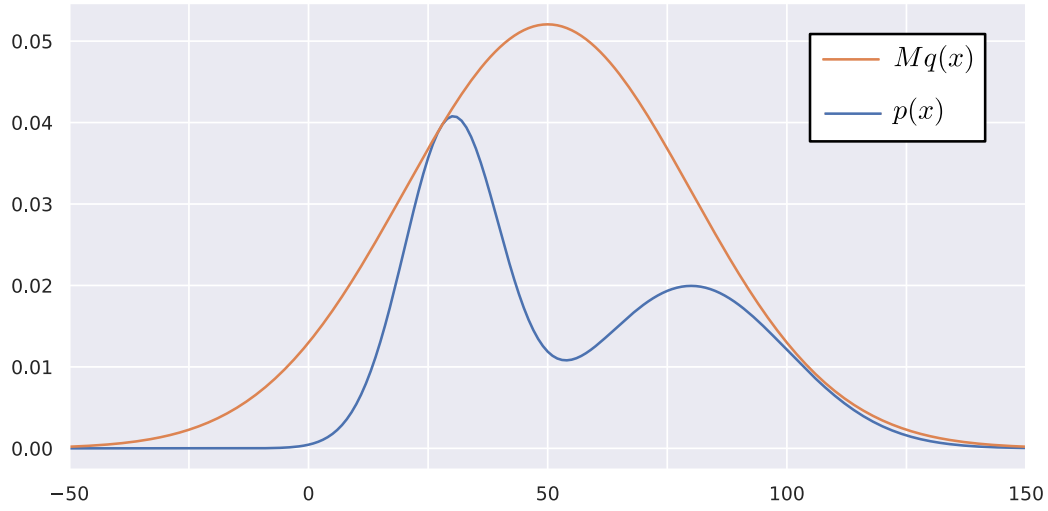
The accepted pairs (u_i, x_i) have a uniform distribution under the curve $p(x)$, i.e. on the set $\{(x, u) : 0 \leq u \leq p(x)\}$.

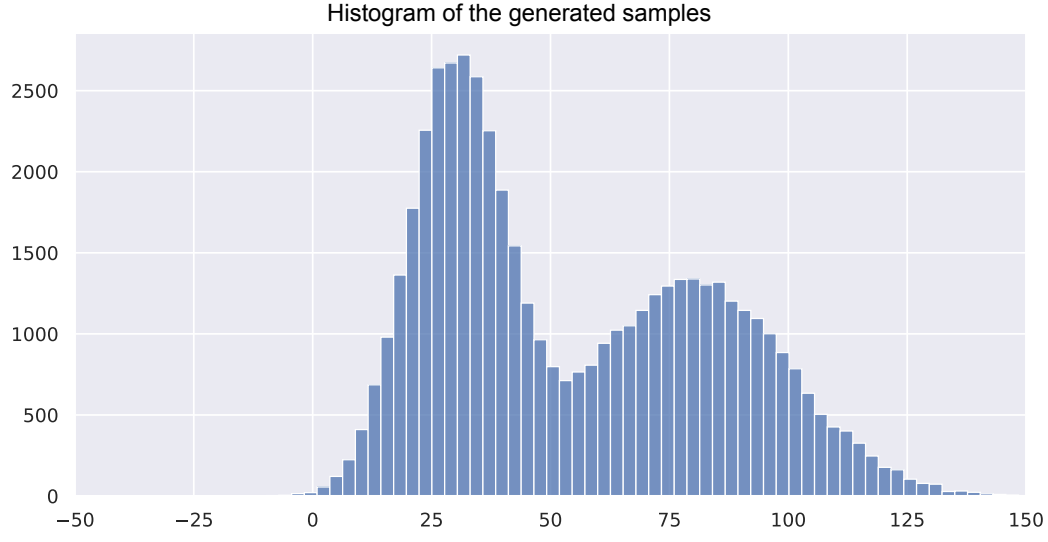
The marginal of this uniform density is the target distribution:

$$\int \mathbb{I}\{0 \leq u \leq p(x)\} du = \int_0^{p(x)} du = p(x).$$

Activity #2

Implementation of rejection sampling.


0.05
0.04
0.03
0.02
0.01
0.00
−50
0
50
100
150
$Mq(x)$
$p(x)$


Histogram of the generated samples
2500
2000
1500
1000
500
0
−50
−25
0
25
50
75
100
125
150

```python
import numpy as np
import scipy.stats as st
import seaborn as sns
import matplotlib.pyplot as plt

sns.set()

def p(x):
    return st.norm.pdf(x, loc=30, scale=10) + st.norm.pdf(x, loc=80, scale=20)

def q(x):
    return st.norm.pdf(x, loc=50, scale=30)

x = np.arange(-50, 151)
M = max(p(x) / q(x)) # p(x) / q(x) <= M <--> p(x) <= M q(x)

def rejection_sampling(iter=1000):
    samples = []

    # TO COMPLETE
    # Use np.random

    return np.array(samples)

s = rejection_sampling(iter=100000)

fig = plt.figure(figsize=(10,5))
ax = fig.subplots(1,1)
ax.plot(x, p(x))
ax.plot(x, M*q(x))
plt.xlim([-50, 150])

fig = plt.figure(figsize=(10,5))
ax = fig.subplots(1,1)
sns.histplot(s, ax=ax)
plt.xlim([-50, 150])
```

```python
import numpy as np
import scipy.stats as st
import seaborn as sns
import matplotlib.pyplot as plt

sns.set()

def p(x):
    return st.norm.pdf(x, loc=30, scale=10) + st.norm.pdf(x, loc=80, scale=20)

def q(x):
    return st.norm.pdf(x, loc=50, scale=30)

x = np.arange(-50, 151)
M = max(p(x) / q(x)) # p(x) / q(x) <= M <--> p(x) <= M q(x)

def rejection_sampling(iter=1000):
    samples = []

    # TO COMPLETE
    # Use np.random

    return np.array(samples)

s = rejection_sampling(iter=100000)

fig = plt.figure(figsize=(10,5))
ax = fig.subplots(1,1)
ax.plot(x, p(x))
ax.plot(x, M*q(x))
plt.xlim([-50, 150])

fig = plt.figure(figsize=(10,5))
ax = fig.subplots(1,1)
sns.histplot(s, ax=ax)
plt.xlim([-50, 150])
```

https://wiseodd.github.io/techblog/2015/10/21/rejection-sampling/

Rejection sampling summary

- Construct an easy to sample density $q(x)$ and a positive number M such that $p(x) \leq Mq(x)$.
- Sample x_i for $i = 1, ..., N$ independently from $q(x)$.
- Accept the sample x_i with acceptance probability $a(x_i)$ where

$$a(x) = \frac{p(x)}{Mq(x)}.$$

In high dimension, both rejection sampling and importance sampling may be inefficient, due to very low acceptance probability or importance weights, respectively (cf. last paragraph of Section 11.1.3 of PRML by Bishop.)

Markov Chain Monte Carlo

- Suppose we want to sample from a target distribution $\pi(x)$. We can evaluate $\pi(x)$ as a function but have no means to directly generate a sample.
- We have seen methods (inverse transform, density transform, rejection sampling) that all produce independent realizations from $\pi(x)$.

  If these methods are inefficient or difficult to implement (e.g. in high dimension), we can drop the independence criteria and generate instead a dependent sequence $\{x_n\}_{n\geq 1}$ such that the marginal distribution of each x_n is the target distribution π.
- Going a step further, we may allow the marginal distribution to be different from π, but converge to π in some sense.
- By relaxing this independence constraint, it becomes possible to overcome some key problems of the previous sampling methods. A practical framework for constructing dependent sequences satisfying the above-mentioned convergence goal is provided by Markov chains.

Markov chain

- A Markov chain is a sequence of random variables $\{x_n\}_{n \geq 1}$ such that

$$p(x_n | x_{n-1}, ..., x_0) = p(x_n | x_{n-1}),$$

that is, x_n is conditionally independent of $x_{n-2}, ..., x_0$ given x_{n-1}. This is called the (1st order) Markov property.

- A Markov chain if fully characterized by the initial distribution $p(x_0)$ and the transition distribution $p(x_n | x_{n-1})$.

For some integer $N \geq 1$:

$$p(x_0, x_1, ..., x_N) = p(x_0) \prod_{n=1}^{N} p(x_n | x_{n-1}).$$

For example, the Gaussian random walk defined by

$$p(x_0) = \mathcal{N}(x_0; 0, 1)$$

and

$$p(x_n|x_{n-1}) = \mathcal{N}(x_n, x_{n-1}, \sigma^2), \qquad n \geq 1$$

is a Markov chain.

Stationary distribution

π is a stationary distribution for the Markov chain defined by the transition distribution $p(x_n|x_{n-1})$ if

$$\pi(x_n) = \int p(x_n|x_{n-1})\pi(x_{n-1})dx_{n-1}.$$

Interpretation using ancestral sampling:

Draw a sample $x' \sim \pi$, use the transition distribution to get a sample $x \sim p(x|x')$, repeat this process many times. If the empirical distribution of the samples x corresponds to π it means that π is a stationary distribution for the Markov chain: we started from π, moved through the Markov chain, and arrived to π again.

- Assume that the initial distribution corresponds to the target distribution:

$$p(x_0) = \pi(x_0).$$

- What is the marginal distribution of x_1?

$$p(x_1) = \int p(x_1, x_0) dx_0 = \int p(x_1|x_0)p(x_0)dx_0 = \int p(x_1|x_0)\pi(x_0)dx_0 = \pi(x_1).$$

  if π is a stationary distribution for the Markov chain.

- This result generalizes to

$$p(x_n) = \pi(x_n), \qquad \forall n \geq 1.$$

- However, in practice, we cannot draw x_0 from the target distribution π, so the last equality generally does not hold.

Instead, we have a sequence of marginal distributions generally defined by:

$$\pi_n(x_n) = \int p(x_n|x_{n-1})\pi_{n-1}(x_{n-1})dx_{n-1}.$$

Does this sequence of distributions converge and, if so, what does it converge to?

A key result is that for an ergodic Markov chain there exists a unique stationary distribution π to which all the marginal distributions π_n converge, irrespective of the initial distribution π_0.

The Markov chain is ergodic if it is irreducible, aperiodic and positive recurrent.

Defining these properties is out of the scope of this lecture.

- The previous discussion suggests that, if we can design a transition distribution $p(x_n|x_{n-1})$ such that the target π is its stationary distribution, at least in principle we can generate samples from the Markov chain that eventually will tend to be drawn from the target distribution.
- After ignoring samples obtained from an initial "burn in" period, as the chain moves towards the stationary distribution, the generated samples can be subsequently used to estimate expectations under the target distribution, as if they are independent.
- Formally we require the chain to be ergodic, otherwise the chain might never reach the desired stationary distribution.
- Determining ergodicity and stationarity for an arbitrary Markov chain is difficult, except in cases where a stronger condition known as detailed balance holds.

Detailed balance

If the transition distribution of the Markov chain satisfies

$$p(x_{n-1}|x_n)\pi(x_n) = p(x_n|x_{n-1})\pi(x_{n-1})$$

then π is a stationary distribution.

Proof: Integrating on both sides we have

$$\pi(x_n) = \int p(x_{n-1}|x_n)\pi(x_n)dx_{n-1} = \int p(x_n|x_{n-1})\pi(x_{n-1})dx_{n-1}$$

This is a sufficient condition; a Markov chain may have π as the stationary distribution while not satisfying detailed balance.

MCMC summary

General idea

- Construct a Markov chain such that its stationary distribution is π,
- Simulate that chain to obtain samples,
- Discard the first samples associated with the so-called burn-in period,
- Build Monte Carlo estimates to approximate intractable expectations.

Theory

The MCMC algorithm is correct if the Markov chain is ergodic with stationary distribution π.

A sufficient condition for the Markov chain to admit π as a stationary distribution is the detailed balance condition.

Examples

The Metropolis-Hastings algorithm and the Gibbs sampler.

Metropolis-Hastings algorithm

Suppose we are given

- a target density $\pi(x) = \phi(x)/Z$, where Z is a possibly unknown normalization constant,
- and a proposal density $q(x'|x)$.

We can evaluate the target density but we cannot sample from it directly.

Metropolis-Hastings algorithm

Given an arbitrary initial sample x_0, the Metropolis-Hastings (MH) algorithm iterates for $n \geq 1$:

- Sample x'_n from the proposal $q(x'_n|x_{n-1})$
- Accept the sample with acceptance probability

$$\alpha(x'_n, x_{n-1}) = \min\left\{1, \frac{q(x_{n-1}|x'_n)\pi(x'_n)}{q(x'_n|x_{n-1})\pi(x_{n-1})}\right\}.$$

  If the sample is rejected, the chain stays at the previous state.

Note that to compute the acceptance probability, we only need to evaluate π up to a normalization, since the normalization constant cancels out.

Metropolis-Hastings algorithm

Given an arbitrary initial sample x_0, the Metropolis-Hastings (MH) algorithm iterates for $n \geq 1$:

- Sample x'_n from the proposal $q(x'_n|x_{n-1})$
- Compute the acceptance probability

$$\alpha(x'_n, x_{n-1}) = \min\left\{1, \frac{q(x_{n-1}|x'_n)\pi(x'_n)}{q(x'_n|x_{n-1})\pi(x_{n-1})}\right\}.$$

- Sample u from the uniform distribution $\mathcal{U}([0,1])$.
- If $u < \alpha(x'_n, x_{n-1})$, accept the sample and set

$$x_n = x'_n,$$

  otherwise, reject the sample and set

$$x_n = x_{n-1}.$$

The MH algorithm implicitly defines a transition probability $p(x|x')$ (not simply equal to the proposal $q(x|x')$) which can be shown to satisfy the detailed balance condition.

Random walk MH demo

By tuning the random walk variance σ^2 we see that there is a trade-off between the number of accepted samples and the ability to explore quickly the complete target density.

The Gibbs sampler

- The MH schema is a very general technique for deriving MCMC algorithms. This generality stems partially from the arbitrariness of the proposal q.
- However, in complex models the design of a reasonable proposal can require a lot of work.
- It would be desirable if somehow the proposal design phase could be automated. The Gibbs sampler is an attempt in this direction.

The Gibbs sampler is suitable for sampling a multivariate random variable $\mathbf{x} = \{x_1, x_2, ..., x_D\}$ with intractable joint target density $p(\mathbf{x})$.

Gibbs sampling proceeds by partitioning the set of variables $\mathbf{x}$ into a chosen variables x_i and the rest $\mathbf{x}_{-i}$, such that $\mathbf{x} = \{x_i, \mathbf{x}_{-i}\}$.

It is assumed that the full conditionals $p(x_i|\mathbf{x}_{-i})$ are tractable and easy to sample.

Remember that we can rely on the Makov blanket of x_i to simplify its full conditional (see Lecture 3). This is fundamental for Gibbs sampling in Bayesian networks!

It can be shown that the Gibbs sampler is actually a MH algorithm with a specific proposal that leads to an acceptance probability of 1, i.e. every sample is accepted.

Gibbs sampler

From an abitrary initial sample $\mathbf{x}_0 = \{x_{1,0}, x_{2,0}, \ldots, x_{D,0}\}$, the Gibbs sampler iterates for $n \geq 1$:

- sample $x_{1,n}$ from $p(x_1 | x_{2,n-1}, x_{3,n-1}, \ldots, x_{D,n-1})$
- sample $x_{2,n}$ from $p(x_2 | x_{1,n}, x_{3,n-1}, \ldots, x_{D,n-1})$
- ...
- sample $x_{D,n}$ from $p(x_D | x_{1,n}, x_{2,n}, \ldots, x_{D-1,n})$

The order for scanning the set of variables $\mathbf{x}$ can be either fixed or defined randomly at each iteration.

Gibbs sampling demo

In 2D, at each iteration of a Gibbs sampler we perform two moves, one corresponding to the sampling of $p(x_1|x_2)$ and the other one corresponding to the sampling of $p(x_2|x_1)$.

Mixing time of MCMC

A key parameter an MCMC algorithm is the number of burn-in steps.

Intuitively, this corresponds to the number of steps needed to converge to our limit (stationary) distribution.

This is called the mixing time of the Markov chain. Unfortunately, this time may vary dramatically, and may sometimes take essentially forever.

There exist many heuristics to determine whether a Markov chain has mixed. Typically these heuristics involve plotting certain quantities such as the sample path or the autocorrelation.

- The sample path is a plot of the realizations x_n along the iterations n. If a chain is mixing poorly, it will remain at or near the same value for many iterations.
- The autocorrelation at lag τ is the correlation between samples that are τ iterations apart. A Markov chain that has poor mixing properties will exhibit slow decay of the autocorrelation as the lag between samples increases.

References

- A. Taylan Cemgil, A Tutorial Introduction to Monte Carlo methods, Markov Chain Monte Carlo and Particle Filtering, in Academic Press Library in Signal Processing (Vol. 1, pp. 1065-1114). Elsevier.
- MCMC course by Professor Kerby Shedden at University of Michigan
- Sampling methods, part of lecture notes for CS 228 - Probabilistic Graphical Models course at Standford.
- Course on Bayesian Methods for Latent Variable Model by Olivier Cappé, 5ème école d'été de Peyresq, 2010.