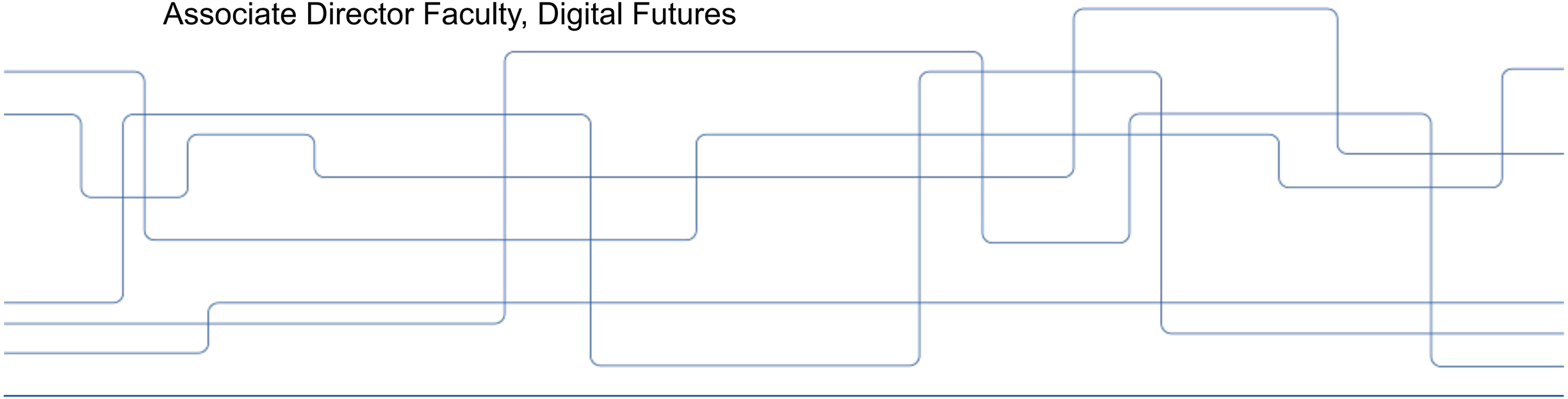
Fundamentals of Bayesian Inference using Probabilistic Programming

Tutorial Day 2
June 7, 2022
Digital Futures Hub, Stockholm, Sweden

digital futures

TECOSA

David Broman
Associate Professor, KTH Royal Institute of Technology
Associate Director Faculty, Digital Futures



Course Overview

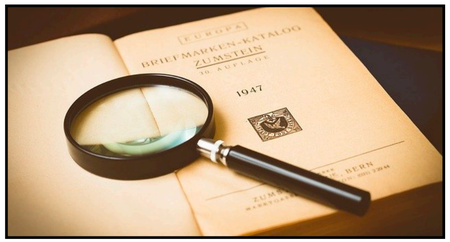
Agenda

14.10 - 15.20 Lecture with mini exercises
15.20 - 15.40 Coffee break
15.40 - 17.00 Lecture with mini exercises

Day I (June 1)
Basics of Bayesian Statistics and Simple Modeling in WebPPL

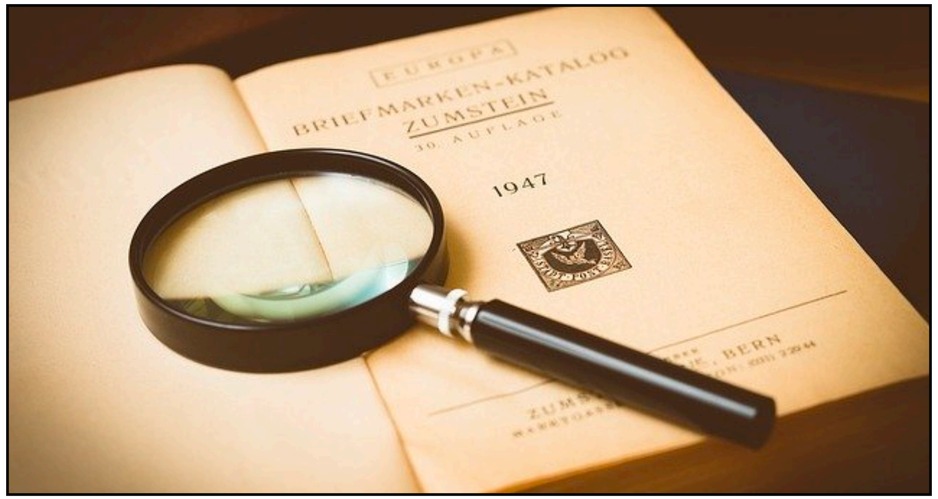
Day II (June 7)
Bayesian Inference Methods and Modeling in Stan

Part I
Inference Methods



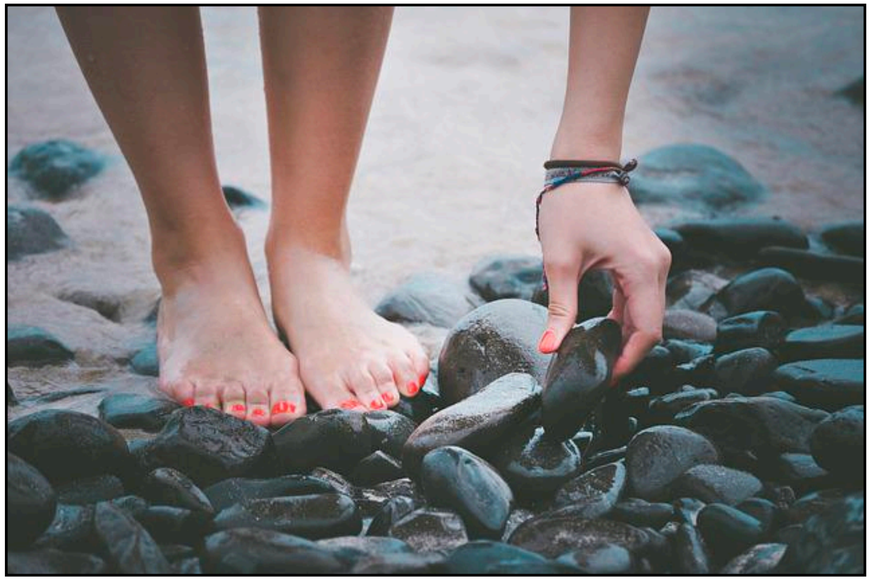
Part II
Modeling in Stan

Part I
Inference Methods



Inference Methods Overview

- Sampling with the CDF
- Rejection Sampling
- Importance Sampling
- Sequential Monte Carlo (SMC)
- Random walk Metropolis
- Metropolis - Hastings
- Hamiltonian Monte Carlo (HMC)



Sampling using the Cumulative Distribution Function (CDF)

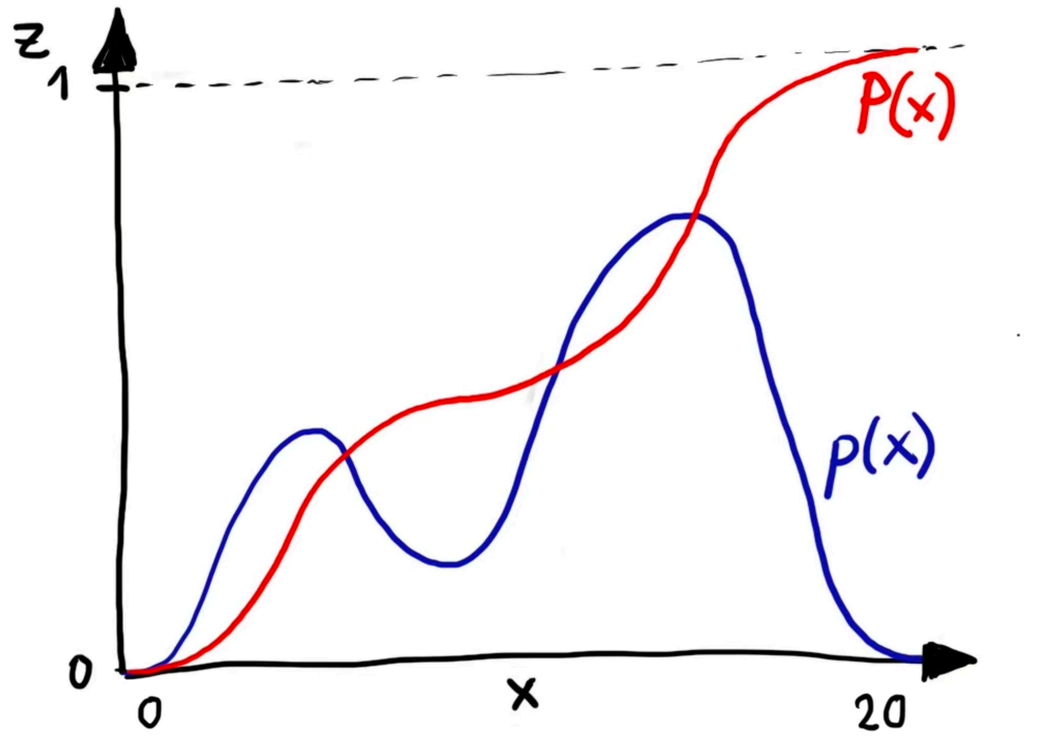
Suppose we can compute the inverse of the CDF (Cumulative distribution function), $P^{-1}(z)$ for some value $z \in [0,1]$.

Procedure:

1. Sample $z \sim U(0,1)$
2. Compute $x = P^{-1}(z)$

$$P(x) = \Pr(X \leq x)$$

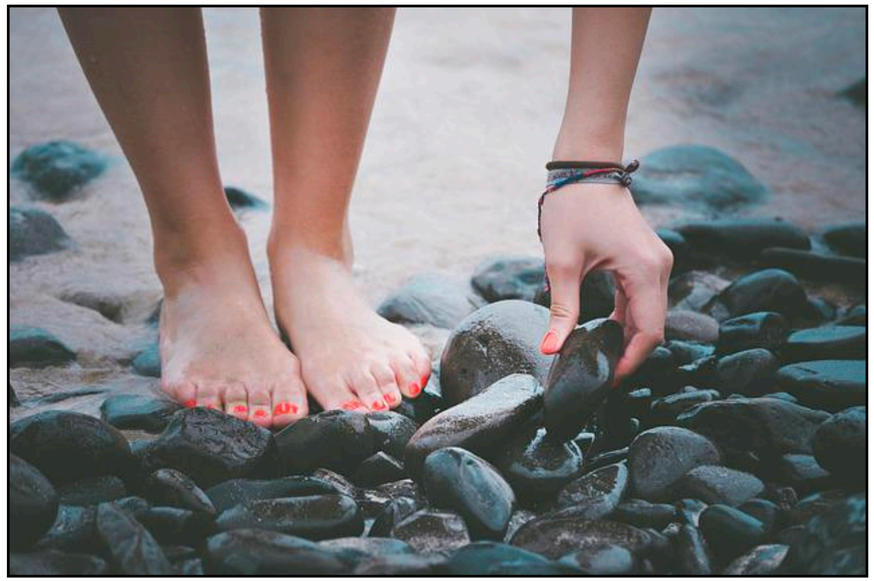
$$p(x) = \frac{d}{dx}P(x)$$ when the derivative exists



Unfortunately, we rarely have the CDF in practice.

Inference Methods Overview

- Sampling with the CDF
- Rejection Sampling
- Importance Sampling
- Sequential Monte Carlo (SMC)
- Random walk Metropolis
- Metropolis - Hastings
- Hamiltonian Monte Carlo (HMC)



Rejection Sampling

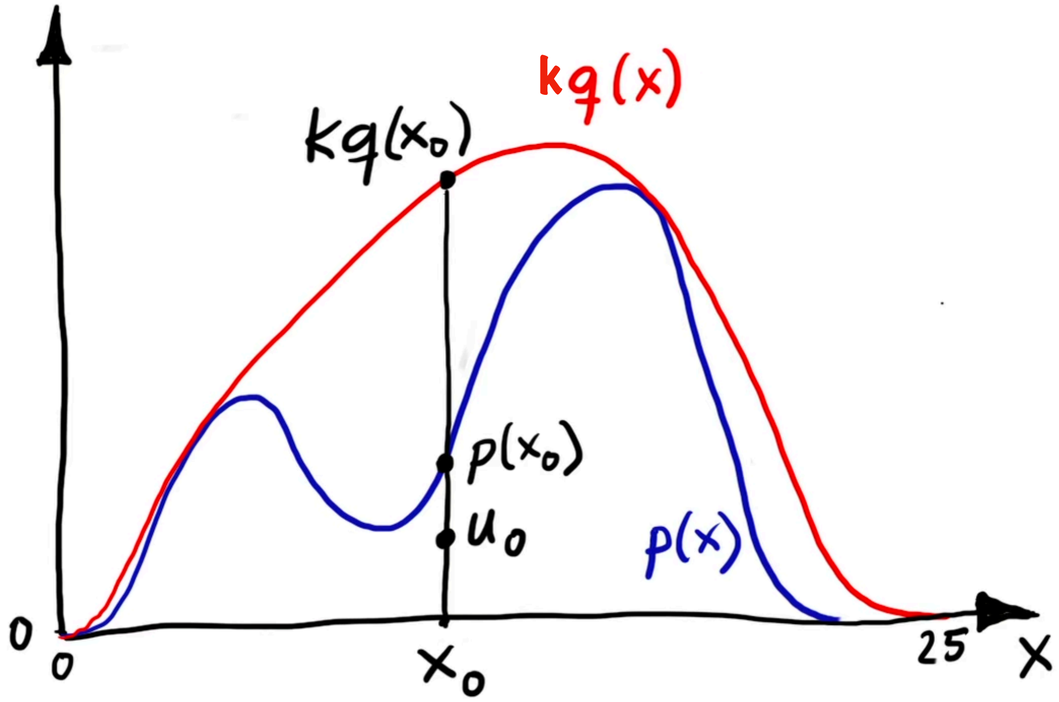
Suppose we cannot draw samples from $p(x)$ directly, but we have another distribution $q(x)$ which we can sample from.

↖ Such $q(x)$ is sometimes called **proposal distribution**.

Suppose we have constant k such that $kq(x) \geq p(x)$ for all x.

Procedure (rejection sampling):

1. Sample $x_0 \sim q$
2. Sample $u_0 \sim U(0,\ kq(x_0))$
3. If $u_0 > p(x_0)$ then reject x_0,
   else x_0 is a sample from $p(x)$

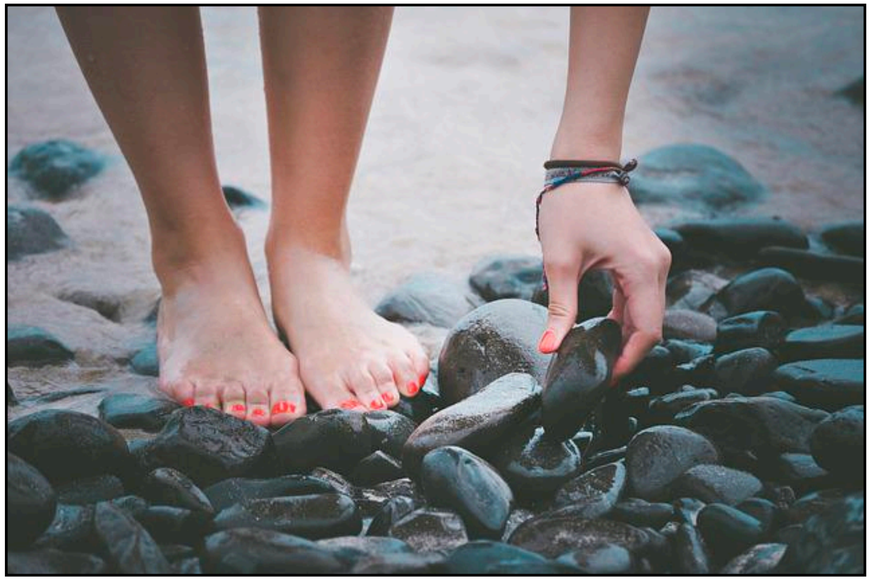


Issues

- Must find $kq(x)$ such that it covers $p(x)$
- If much space in-between, the rejection rate becomes high.
- Can use **adaptive rejection sampling**, but only works on low dimensions

Inference Methods Overview

- Sampling with the CDF
- Rejection Sampling
- Importance Sampling
- Sequential Monte Carlo (SMC)
- Random walk Metropolis
- Metropolis - Hastings
- Hamiltonian Monte Carlo (HMC)



Approximating Expectations

Often we would like to compute the expectation of a function $f(x)$ given a distribution $p(x)$

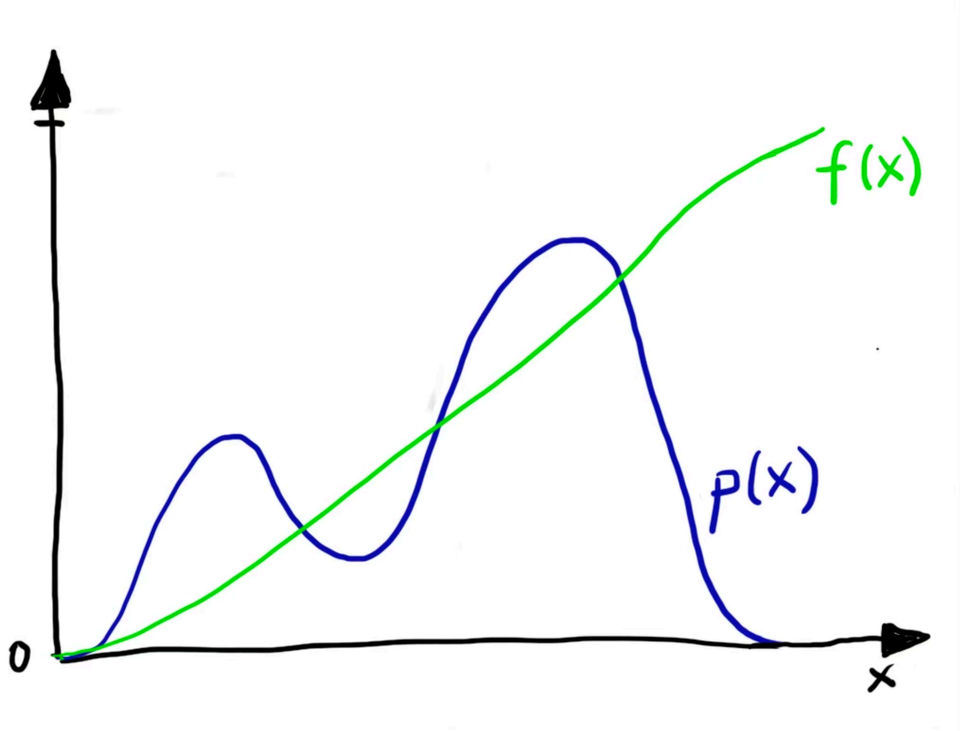
$$\mathbb{E}[f(x)] = \int f(x)p(x)dx$$

If $f(x) = x$ is the identity function, we compute the mean

$$\mathbb{E}[x] = \int x\,p(x)dx$$

If we can get a set of samples $\{x_i\}$ drawn independently from $p(x)$, where $i = 1,\ldots,N$, then we can approximate

$$\mathbb{E}[f(x)] \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$



Note: this assumes that we **can** sample from $p(x)$

Can we compute the expectation without sampling from $p(x)$? Yes, by using **importance sampling**

Importance Sampling

Suppose we want to compute the mean of $p(x)$ but we cannot sample from $p(x)$, only evaluate.

Note the expectation with respect to $p(x)$ written as $\mathbb{E}_p$

$$\mathbb{E}_p[x] = \int x\, p(x) dx$$

$$= \int x \frac{p(x)}{q(x)} q(x) dx$$

Just adding $q(x)$ to the numerator and the denominator

Let $f(x) = x \frac{p(x)}{q(x)}$ then

$$\mathbb{E}_p[x] = \int f(x)\, q(x) dx = \mathbb{E}_q[f(x)]$$

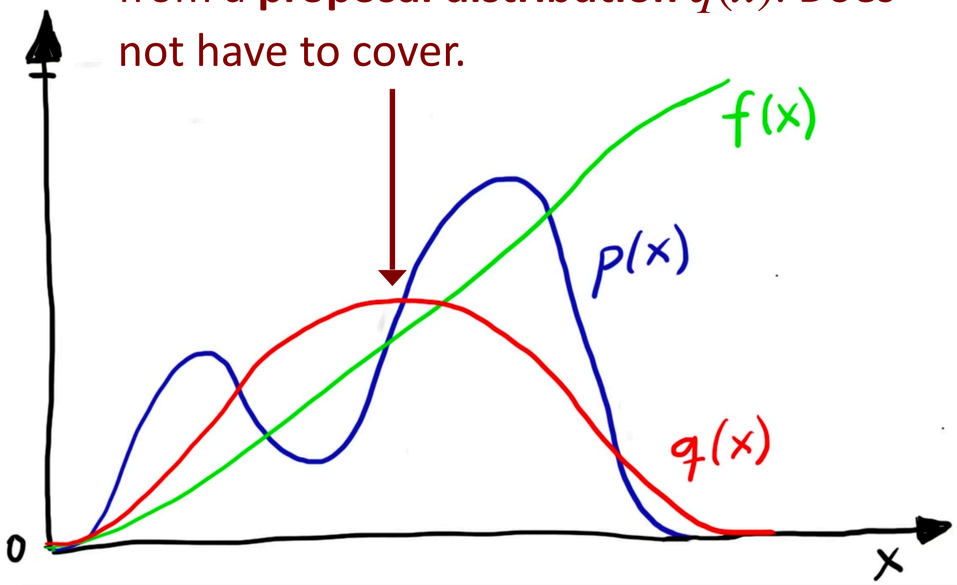
$$\mathbb{E}_q[f(x)] \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i) = \frac{1}{N} \sum_{i=1}^{N} x_i \frac{p(x_i)}{q(x_i)}$$

for samples $\{x_i\}$ sampled i.i.d from $q(x)$

We can approximate by sampling N sampled from q

The importance weight

We compute expectations by sampling from a **proposal distribution** $q(x)$. Does not have to cover.



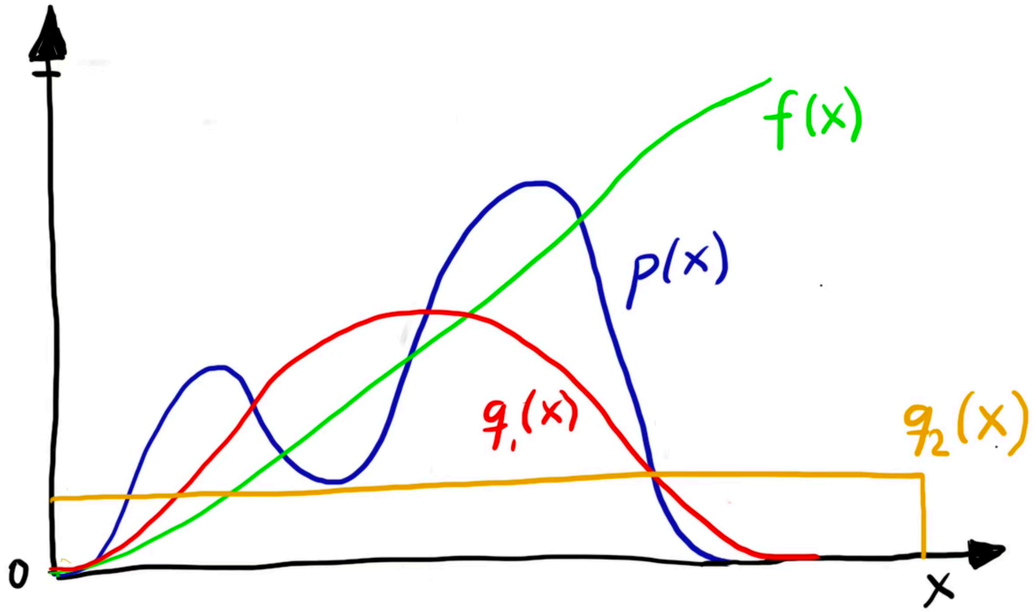
However, often we cannot even evaluated $p(x)$ directly (compare with Bayes' rule)

$$p(x) = \frac{\tilde{p}(x)}{Z}$$

Can easily be evaluated up to a normalizing constant Z

Can still be solved (see e.g., Bishop, 2006)

Importance Sampling



1. Which of the two proposal distributions q_1 and q_2 would give asymptotically the correct result, when $N \to \infty$

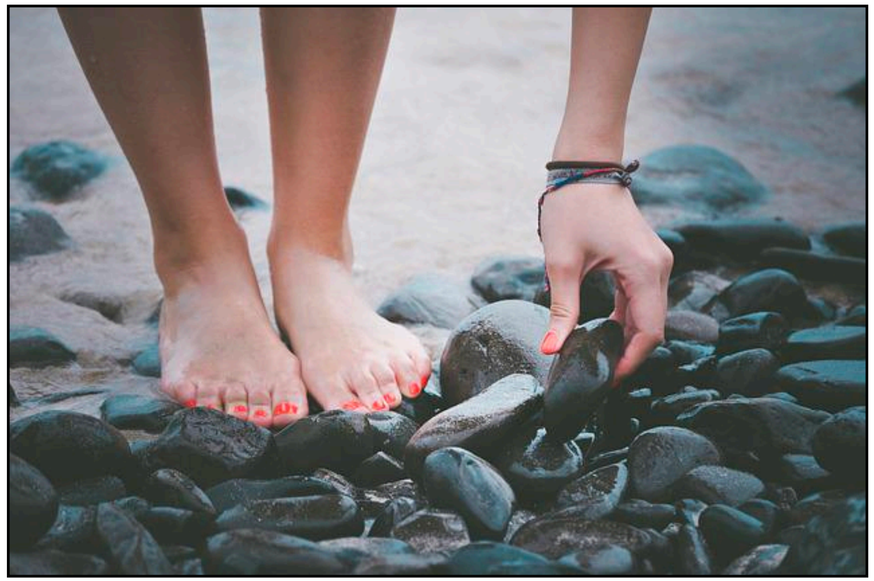
2. Which one would give the best results quickest?

Problems with importance sampling

- Requires the proposal to be close to the posterior, otherwise it may give low sampling efficiency.
- Can give arbitrary error - hard to get diagnostics.

Inference Methods Overview

- Sampling with the CDF
- Rejection Sampling
- Importance Sampling
- Sequential Monte Carlo (SMC)
- Random walk Metropolis
- Metropolis - Hastings
- Hamiltonian Monte Carlo (HMC)



Markov Models

Question: If we have a time series (a sequence of observations), would it then make sense to treat it as i.i.d?

No, there are typically dependencies.

i.i.d = independent and identically distributed

Markov property
Given the present, the future does not depend on the past

The joint distribution can be expressed using the product rule

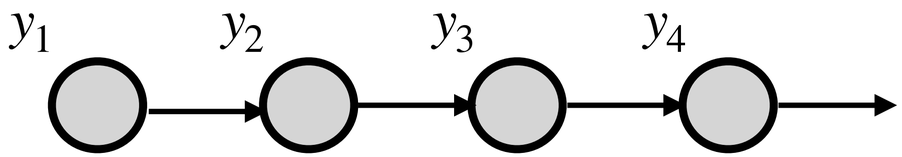
$$p(y_1, \ldots, y_T) = p(y_1) \prod_{k=2}^{T} p(y_k \,|\, y_1, \ldots, y_{k-1})$$

Suppose a given distribution is independent of all previous observations, except the previous one.

Then it is called a **first-order Markov chain**

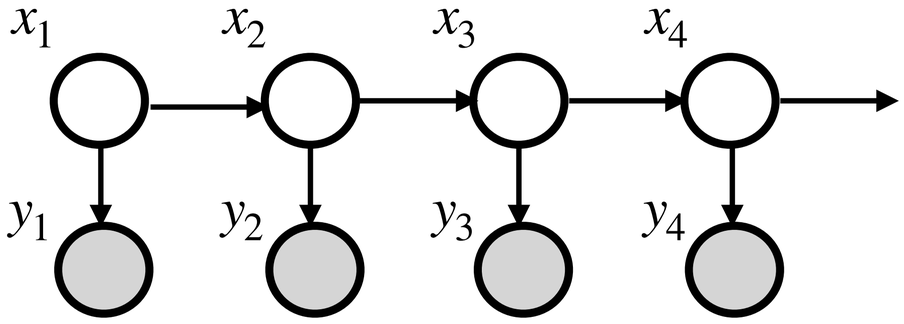
$$p(y_1, \ldots, y_T) = p(y_1) \prod_{k=2}^{T} p(y_k \,|\, y_{k-1})$$

Gray marked nodes are observed variables



A **hidden Markov model** (also called **state-space model**) has the Markov property, latent variables ***x***, and observed variables ***y***.

$$p(x_1, \ldots, x_T, y_1, \ldots, y_T) = p(x_1) p(y_1 \,|\, x_1) \prod_{k=2}^{T} p(x_k \,|\, x_{k-1}) p(y_k \,|\, x_k)$$



Sequential Monte Carlo (SMC)

Key idea illustrated with an example

Uniform prior

Initial position: $X_0 \sim \mathcal{U}(0, 100)$

Observation model: $Y_t \sim \mathcal{N}(\text{map}(X_t), 5)$

Observe Y using noisy altimeter and noisy sensor (aircraft to ground)

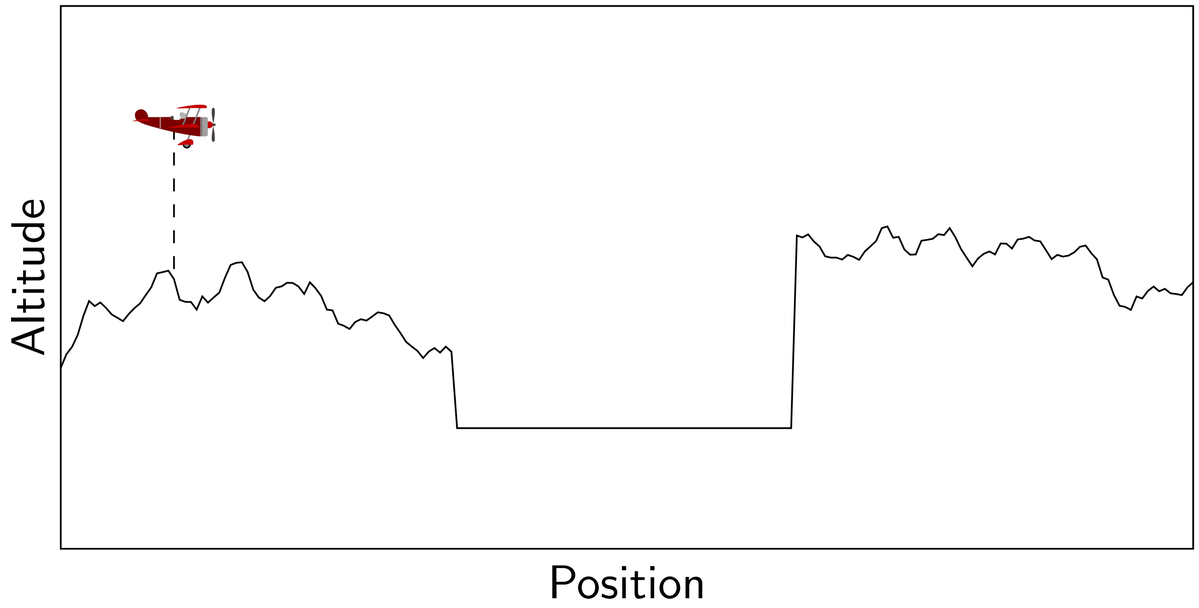
Mean value looked up from a map.

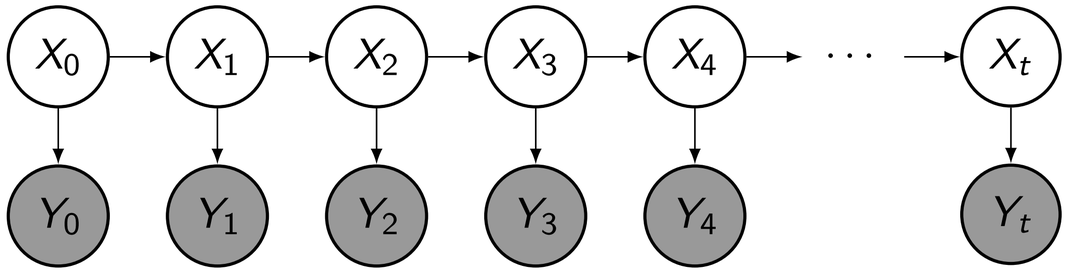
Transition model: $X_t \sim \mathcal{N}(X_{t-1} + 2, 0.5)$

Constant speed, measured with uncertainty

Problem: Find $p(x_t \mid y_{0:t})$

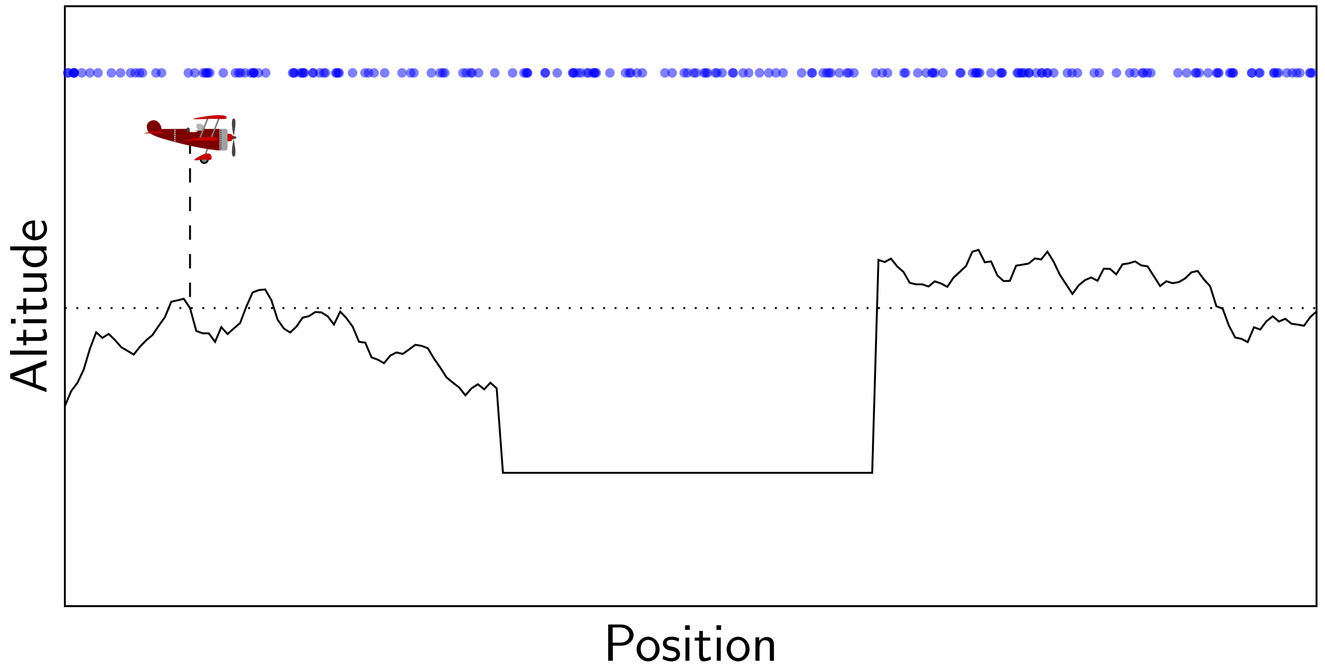
That is, where is the plane?





(Thanks to Daniel Lundén for the illustration and Andreas Svensson for the aircraft example)

Sequential Monte Carlo (SMC)



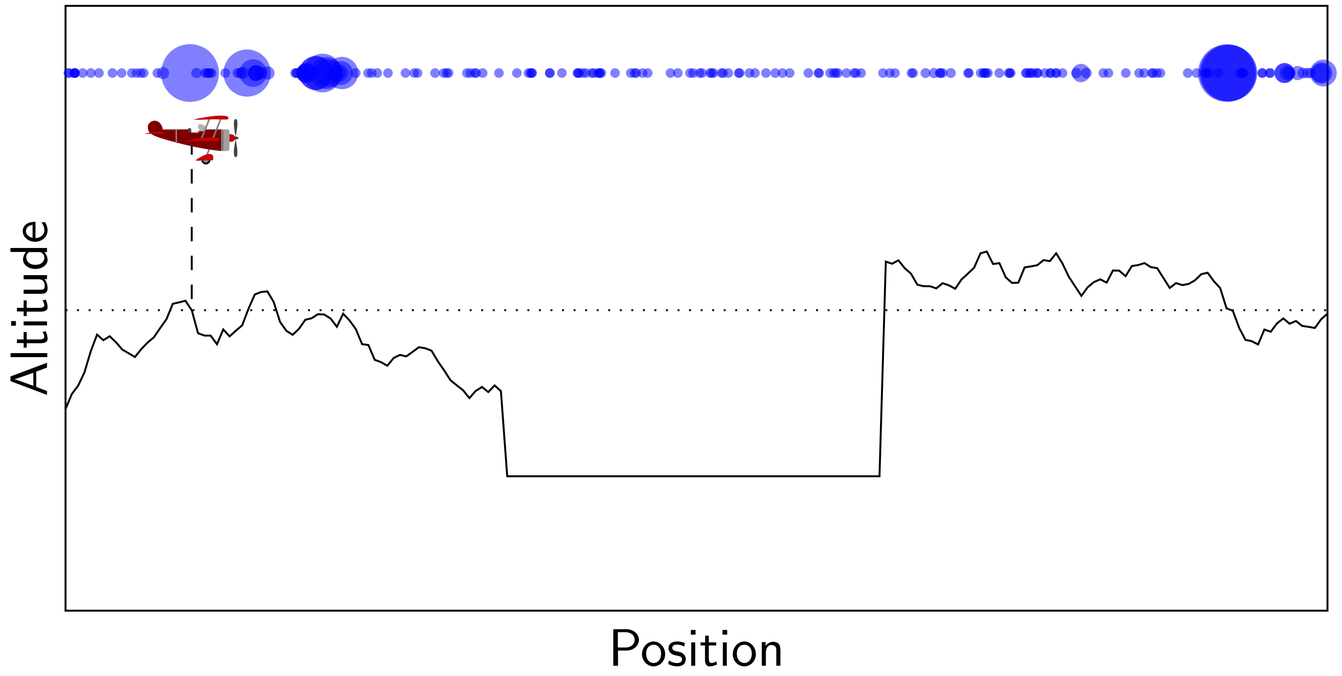
Initialize N number of particles
(N = 200 in this case)

Initial position: $X_0 \sim \mathcal{U}(0, 100)$

Observation model: $Y_t \sim \mathcal{N}(\text{map}(X_t), 5)$

Transition model: $X_t \sim \mathcal{N}(X_{t-1} + 2, 0.5)$

Sequential Monte Carlo (SMC)



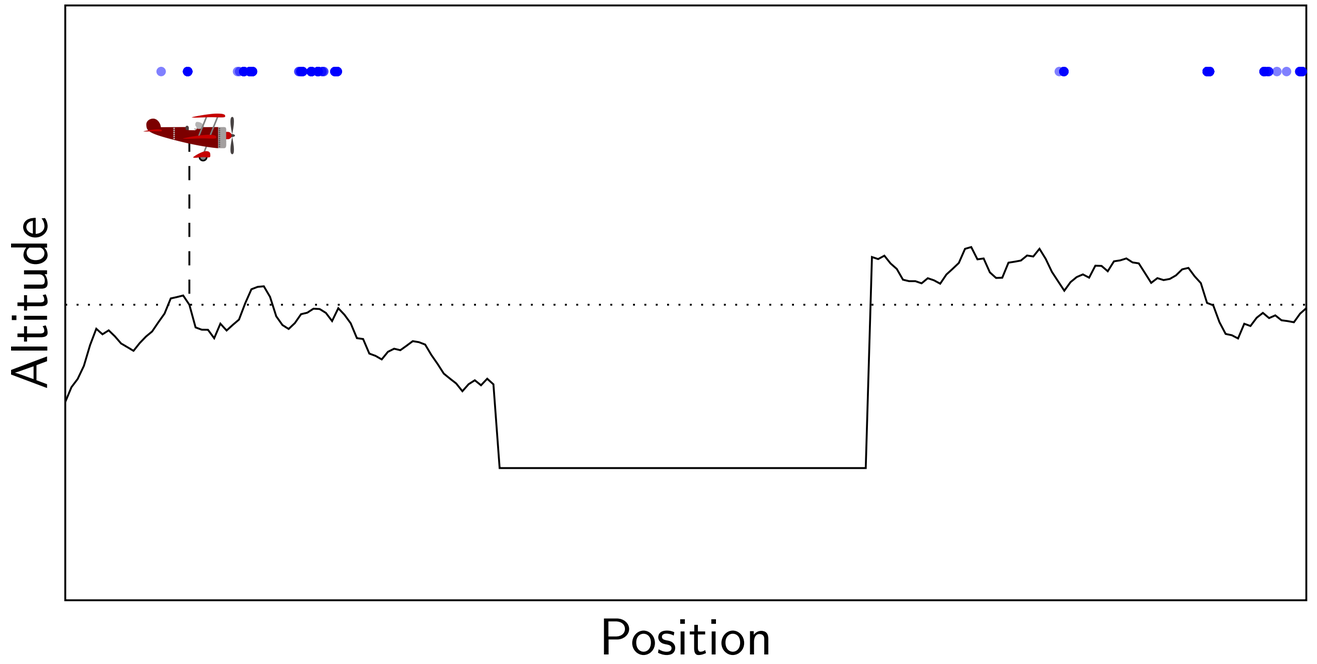
Weight samples using the observations.

Initial position: $X_0 \sim \mathcal{U}(0, 100)$

Observation model: $Y_t \sim \mathcal{N}(\text{map}(X_t), 5)$

Transition model: $X_t \sim \mathcal{N}(X_{t-1} + 2, 0.5)$

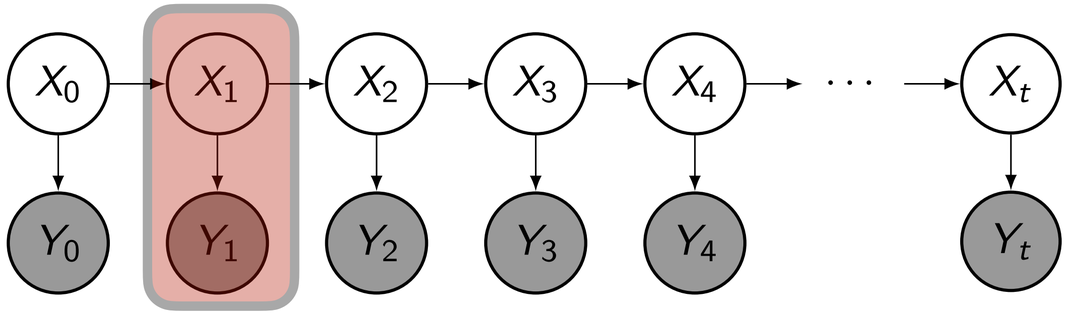
Sequential Monte Carlo (SMC)



Resample according to their weights (key step of SMC). Still 200 particles.

Also called **bootstrap particle filter**, specific algorithm within the SMC

Refocuses the state on areas that are the most likely ones.

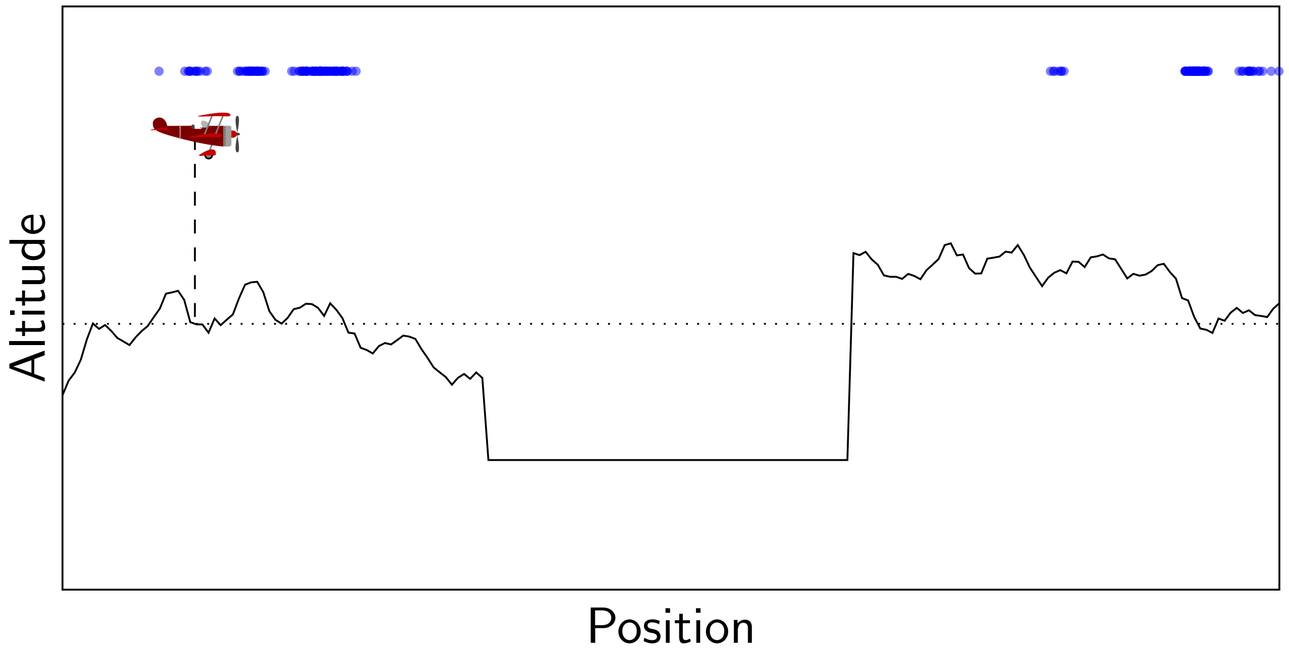


Initial position: $X_0 \sim \mathcal{U}(0, 100)$

Observation model: $Y_t \sim \mathcal{N}(\text{map}(X_t), 5)$

Transition model: $X_t \sim \mathcal{N}(X_{t-1} + 2, 0.5)$

Sequential Monte Carlo (SMC)

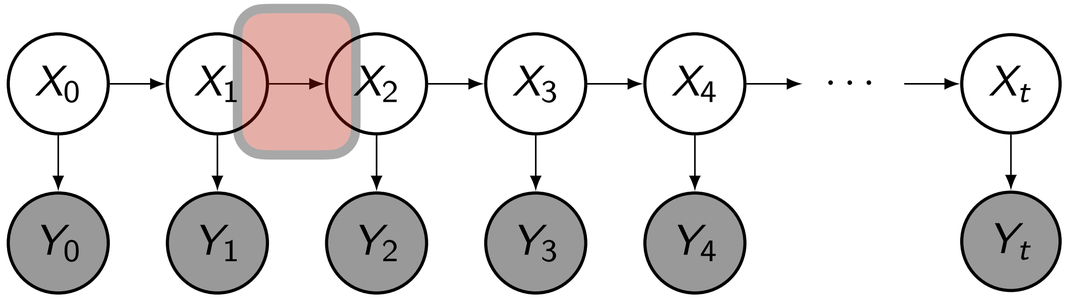


Transition - stretches out slightly because of uncertainty

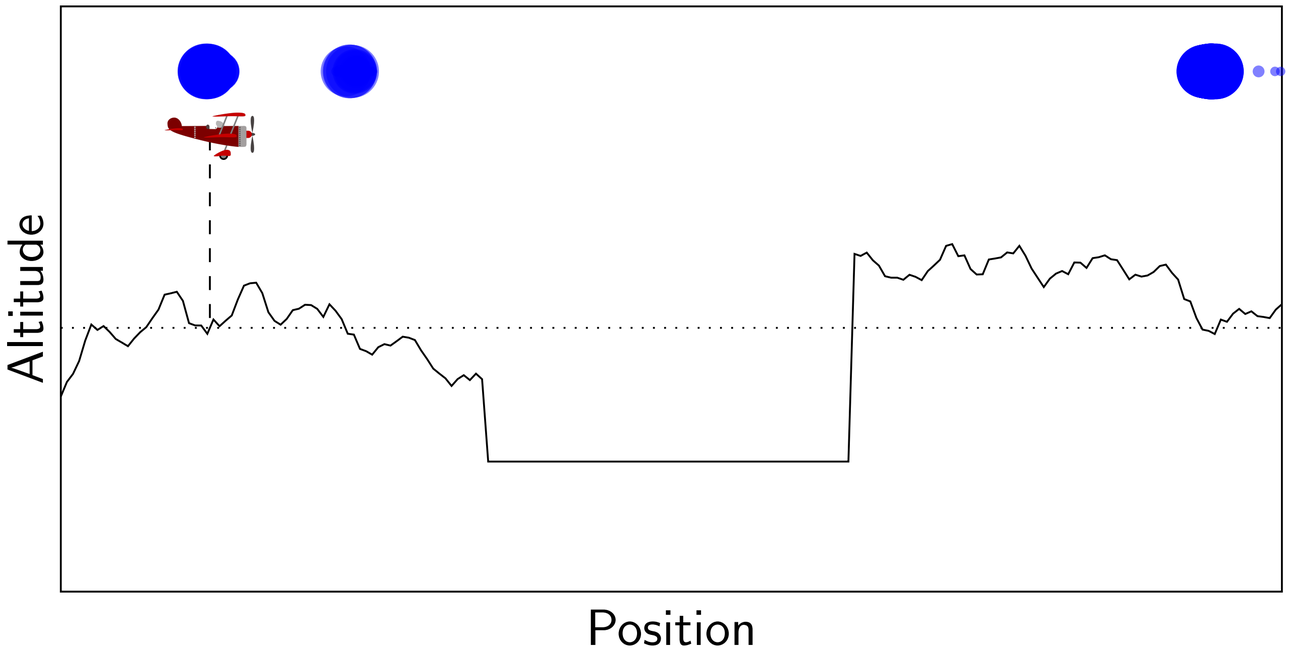
Initial position: $X_0 \sim \mathcal{U}(0, 100)$

Observation model: $Y_t \sim \mathcal{N}(\text{map}(X_t), 5)$

Transition model: $X_t \sim \mathcal{N}(X_{t-1} + 2, 0.5)$



Sequential Monte Carlo (SMC)

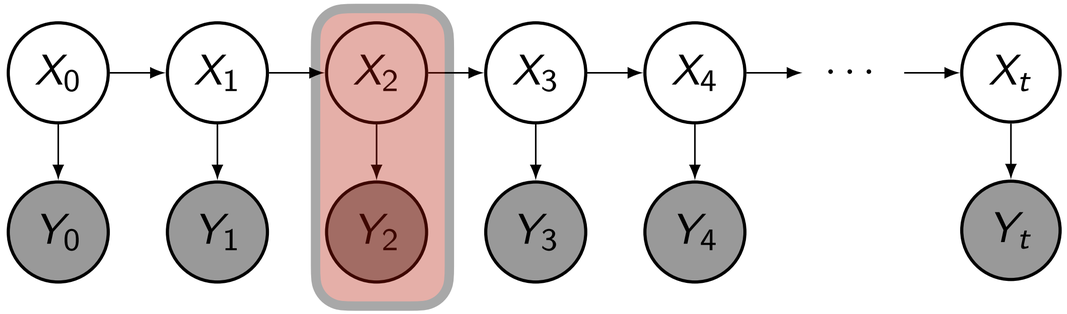


Weight again using the **observation**

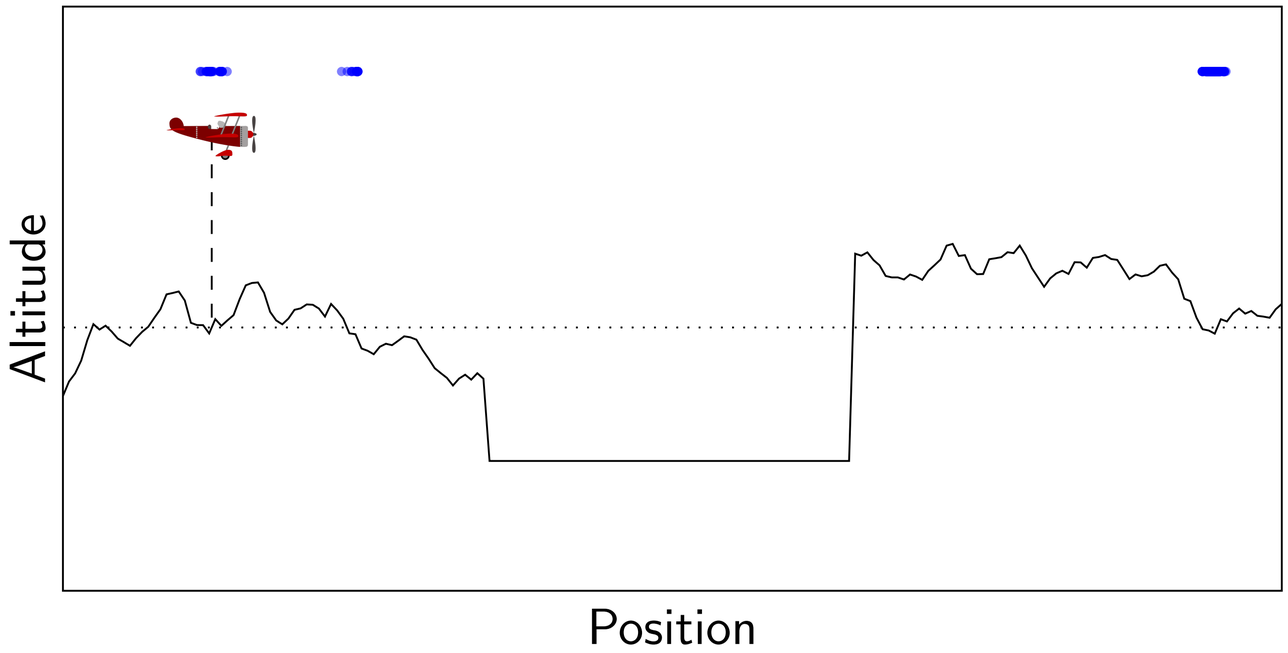
Initial position: $X_0 \sim \mathcal{U}(0, 100)$

Observation model: $Y_t \sim \mathcal{N}(\text{map}(X_t), 5)$

Transition model: $X_t \sim \mathcal{N}(X_{t-1} + 2, 0.5)$



Sequential Monte Carlo (SMC)

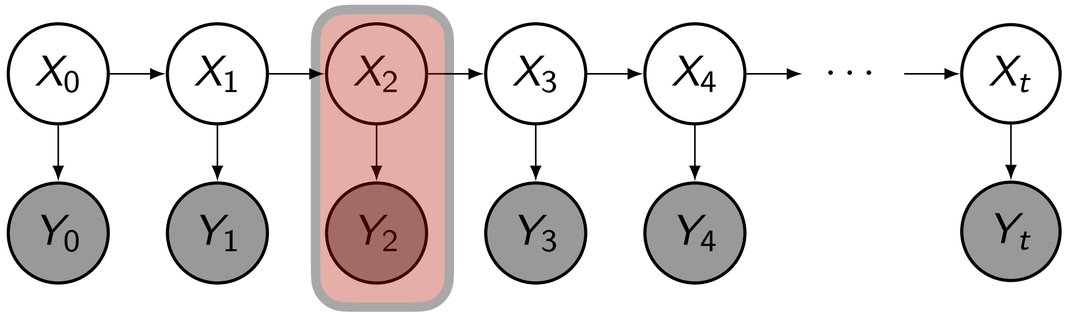


Resample again

Initial position: $X_0 \sim \mathcal{U}(0, 100)$

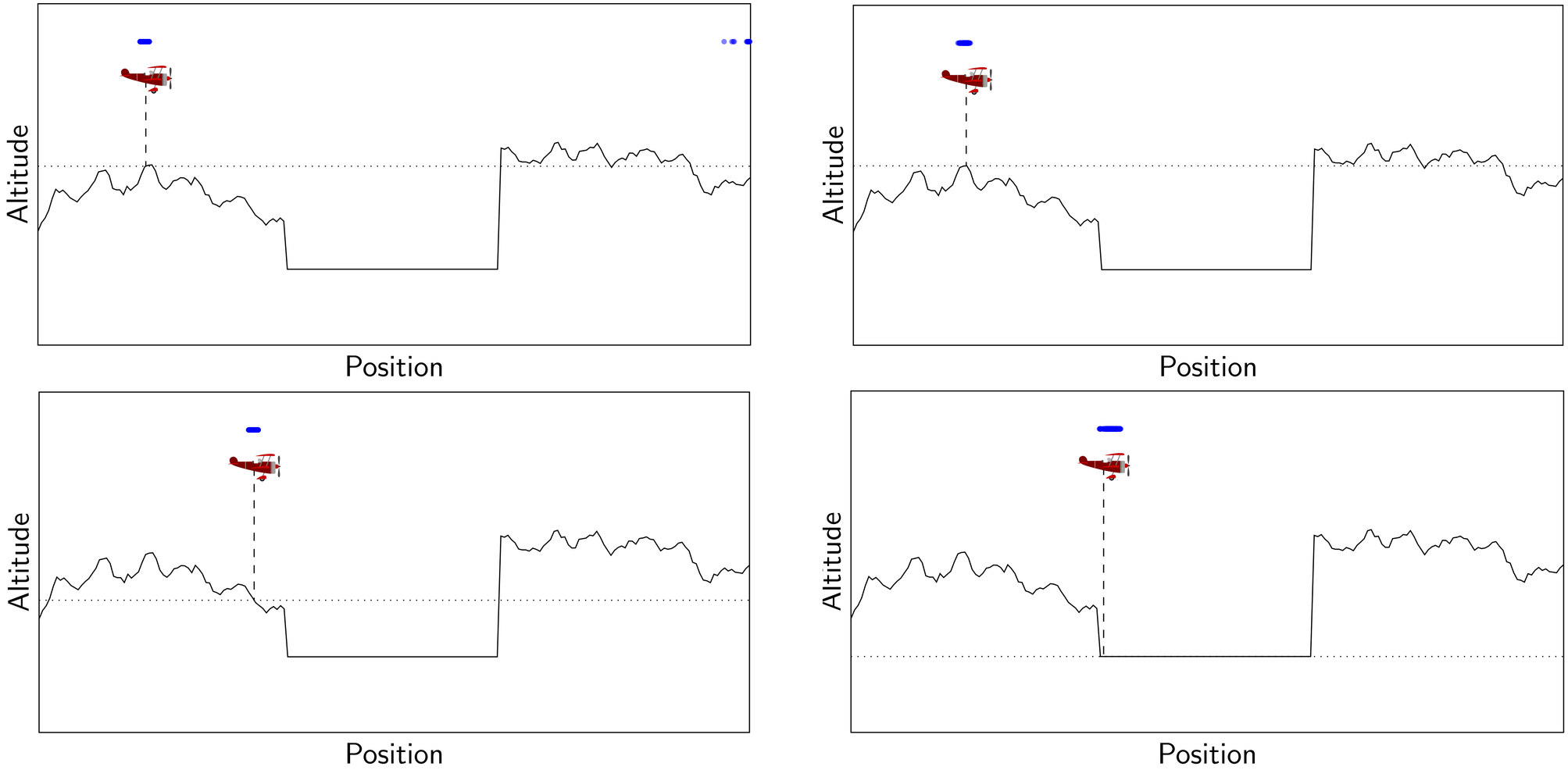
Observation model: $Y_t \sim \mathcal{N}(\text{map}(X_t), 5)$

Transition model: $X_t \sim \mathcal{N}(X_{t-1} + 2, 0.5)$



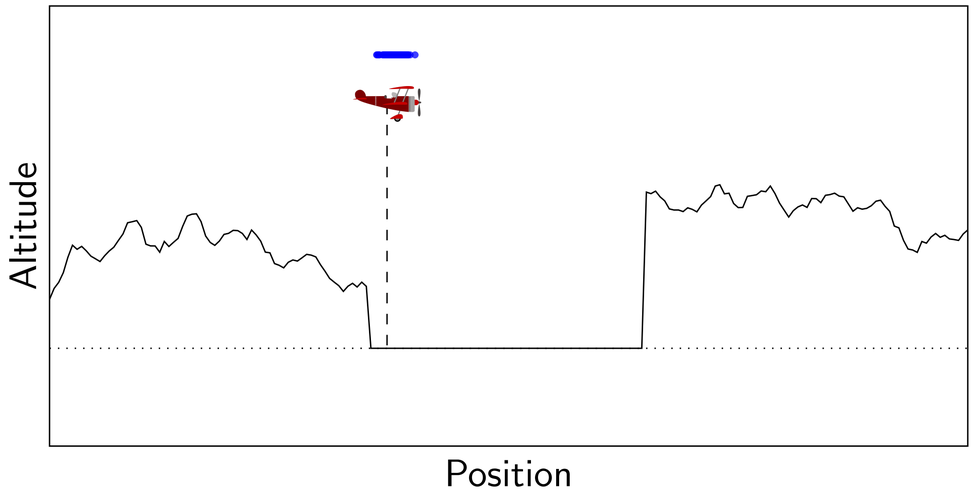
Sequential Monte Carlo (SMC)

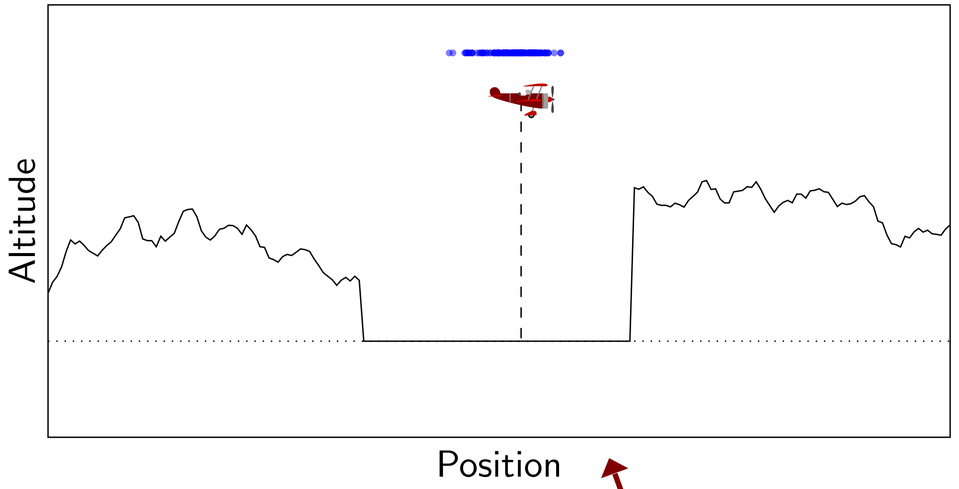
Propagate, weight, resample, over and over again…

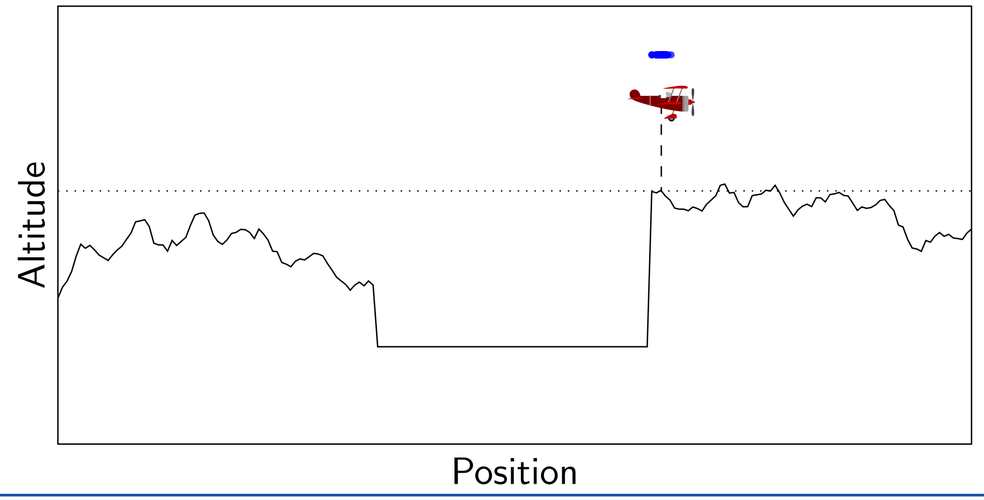


Sequential Monte Carlo (SMC)

Propagate, weight, resample, over and over again…







Note how the **uncertainty increases** when flying over water (less information)

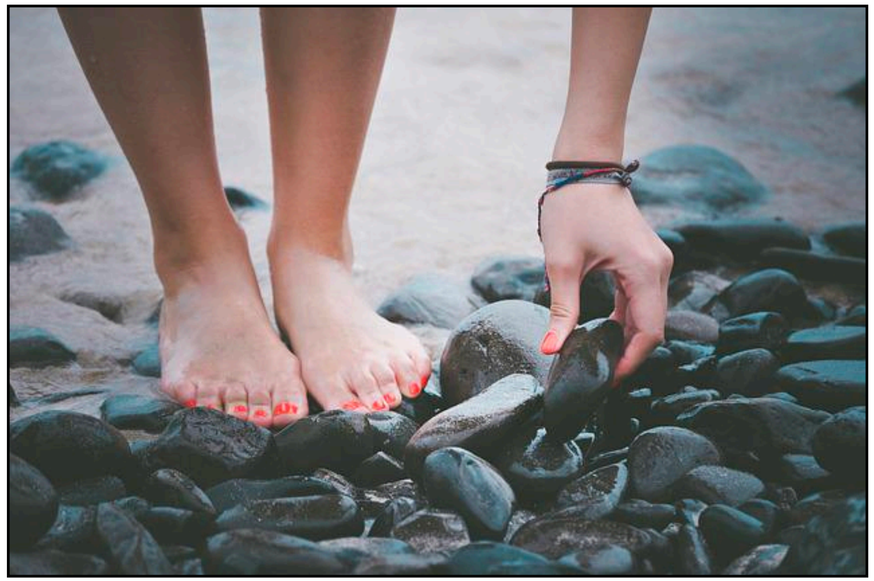
Note how the estimation gets more certain again, when flying over land.

When SMC is used in PPLs, resampling typically happens at observations

Inference Methods Overview

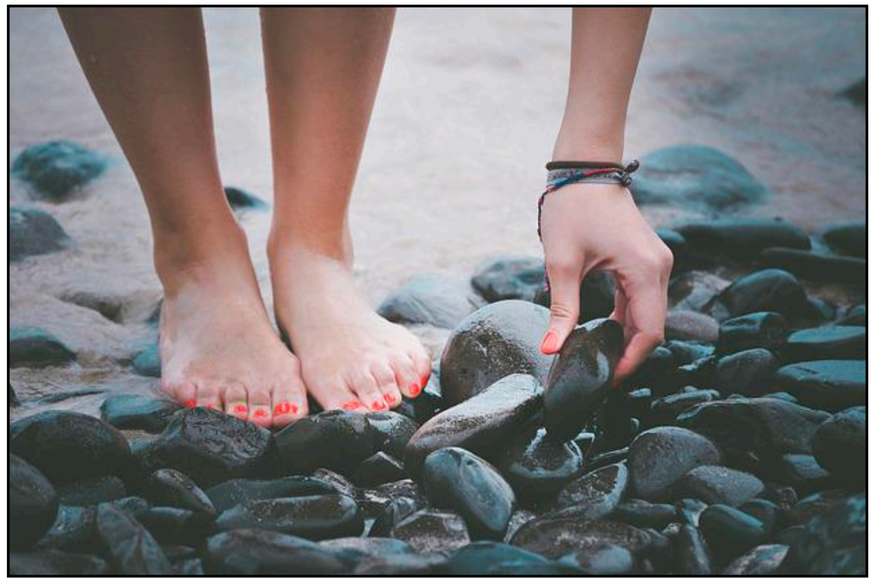
- Sampling with the CDF
- Rejection Sampling
- Importance Sampling
- Sequential Monte Carlo (SMC)
- Random walk Metropolis
- Metropolis - Hastings
- Hamiltonian Monte Carlo (HMC)

Markov chain Monte Carlo (MCMC) methods



Inference Methods Overview

- Sampling with the CDF
- Rejection Sampling
- Importance Sampling
- Sequential Monte Carlo (SMC)
- Random walk Metropolis
- Metropolis - Hastings
- Hamiltonian Monte Carlo (HMC)



Markov chain Monte Carlo (MCMC)

It is hard to make importance sampling and SMC to scale to higher **dimensional problems**.

We will now give the intuition for a common type of **dependent sampler**:

Markov chain Monte Carlo

Walks over the the posterior, only remembering the previous step.

Each step is **dependent** on its previous state.

The direction of the step taken is **random** (hence Monte Carlo).

Sometimes called **"random walk"**

Recall: Bayesian Inference and Bayes' Rule

Goal: Estimate the parameters θ (random variables) given the observation of data x.

Likelihood of generating data x given the parameters θ

Prior, our beliefs of the parameters right now.

$$p(\theta \mid x) = \frac{p(x \mid \theta)p(\theta)}{p(x)}$$

Posterior, the inferred parameters θ given the observed data.

Normalizing constant. Determined by prior and likelihood.

But, to calculate $p(x)$, we needed to compute the integral $p(x) = \int p(x \mid \theta)p(\theta)d\theta$, which can be very hard!

Computing Posterior Ratio

Suppose we want to compute the posterior for two parameters θ_1 and θ_2 in the parameter space.

$$p(\theta \mid x) = \frac{p(x \mid \theta)p(\theta)}{p(x)}$$

$p(\theta_1 \mid x) \quad p(\theta_2 \mid x)$ ← Not possible directly, if we cannot compute $p(x)$

But, can we not compute the ratio?

$$\frac{p(\theta_1 \mid x)}{p(\theta_2 \mid x)} = \frac{\frac{p(x \mid \theta_1)p(\theta_1)}{p(x)}}{\frac{p(x \mid \theta_2)p(\theta_2)}{p(x)}} = \frac{p(x \mid \theta_1)p(\theta_1)}{p(x \mid \theta_2)p(\theta_2)}$$

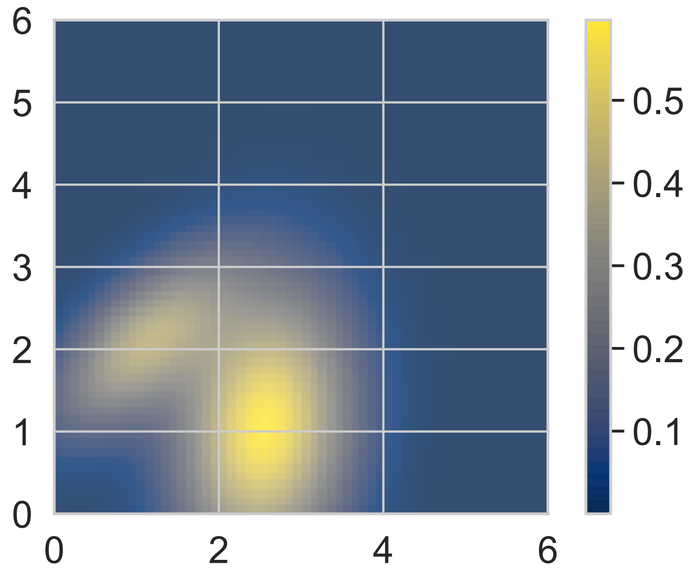
Knowing the **un-normalized posterior** is enough (the numerator of Bayes' rule)

The normalization constant $p(x)$ is cancelled out!

$$p(\theta \mid x) = \frac{p(x \mid \theta)p(\theta)}{p(x)} \propto p(x \mid \theta)p(\theta)$$

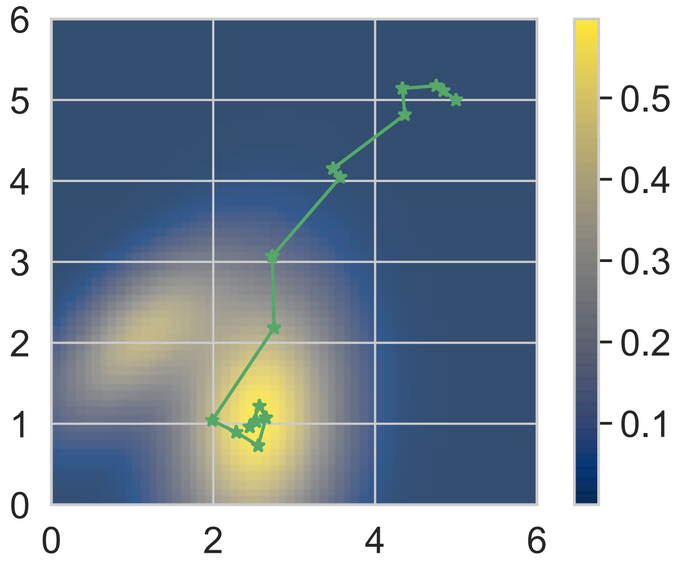
The un-normalized posterior i proportional to the posterior.

Climbing the Hill of the Posterior



A posterior (2 dim) with two hills.

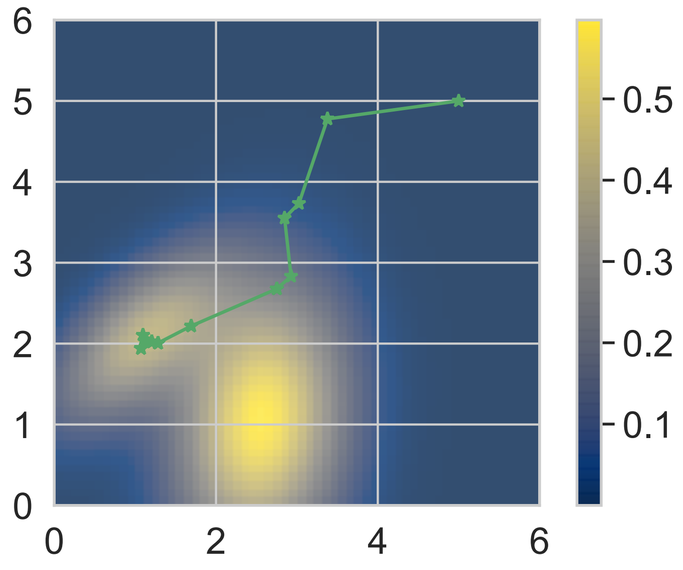
We write $f(\theta) \propto p(x \mid \theta)p(\theta)$



Acceptance rate: 0.87

Procedure:

1. Select a start position θ.
2. Randomly select a new possible move to θ' (e.g. using multivariate normal dist)
3. If $f(\theta') > f(\theta)$, then accept and move $\theta \leftarrow \theta'$, else reject.
4. Go to step 2.



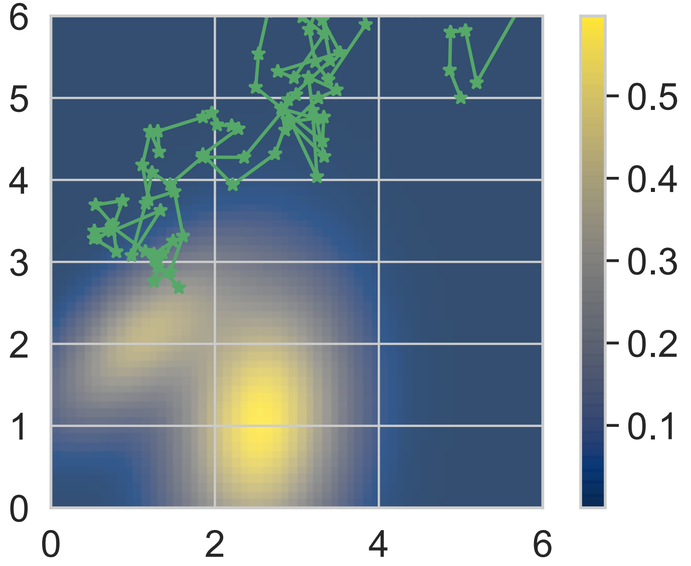
Acceptance rate: 0.9

Can end up in different local maximums.

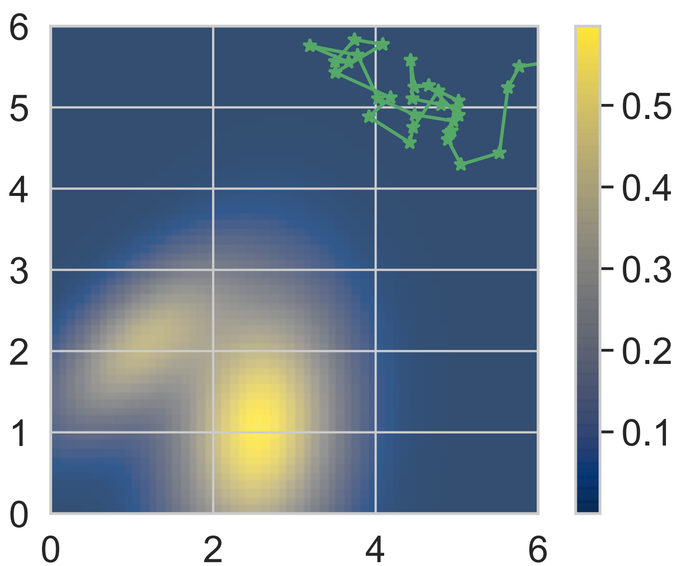
What do we get? Problems?

Potentially the **mode**, but not the posterior. Does not **explore** the space.

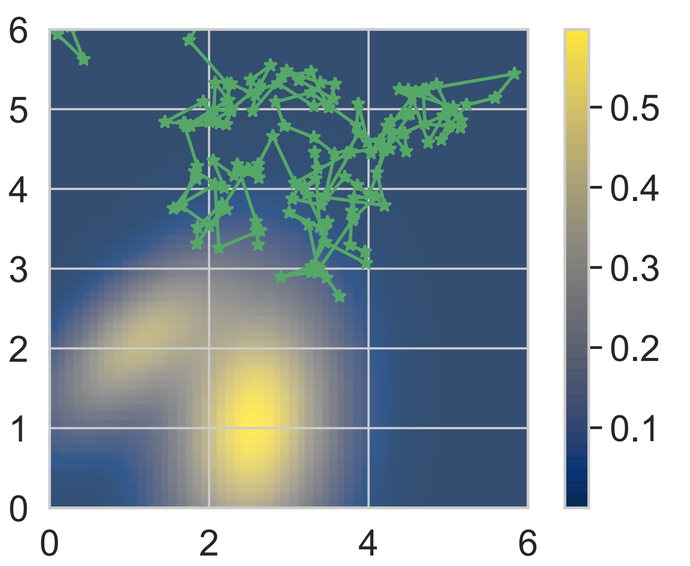
Random Walk (Really)



Acceptance rate: 1.0



Acceptance rate: 1.0



Acceptance rate: 1.0

Procedure:

1. Select a start position θ.
2. Randomly select a new possible move to θ' (e.g. using multivariate normal dist)
3. Accept and move $\theta \leftarrow \theta'$, and go to step 2.

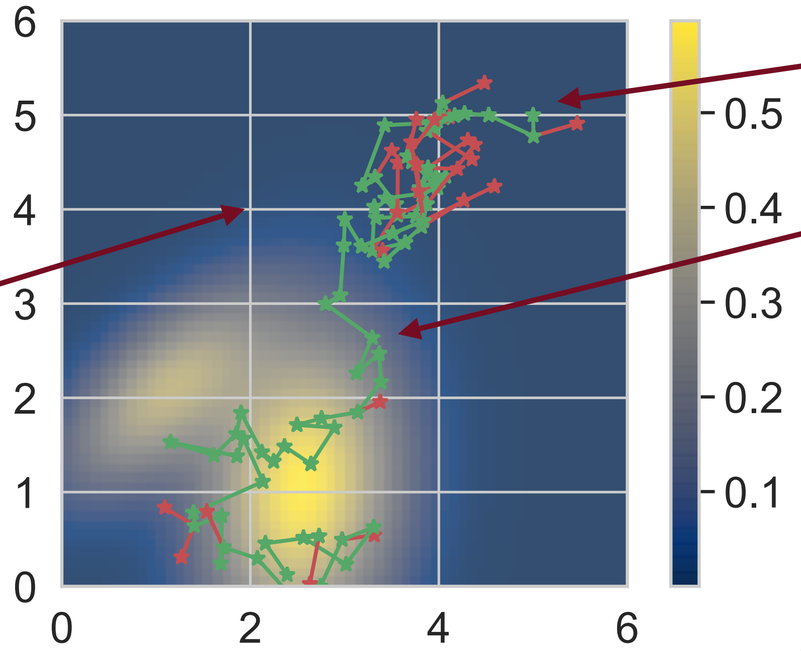
What do we get now?

Explores randomly, but without considering the proportion to the pdf.

Can we get something in between?

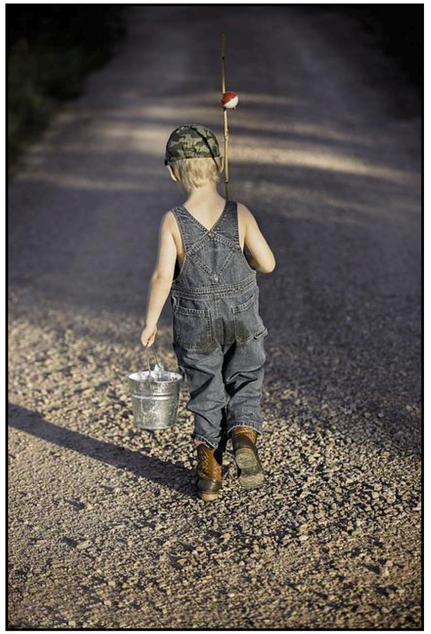
Random Walk Metropolis

Acceptance rate: 0.74



Accepts in green, rejects in red.

Must discard the "warm-up" before finding the **typical set**.



The distribution used to make the move is called the **jumping** or **proposal distribution**.

Procedure:

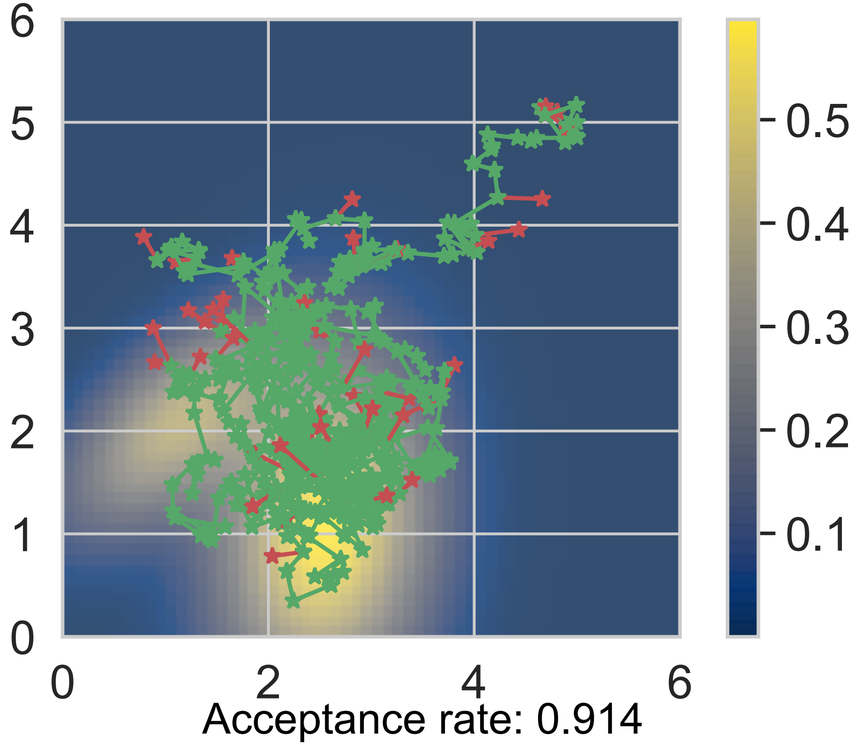
1. Select a start position θ.
2. Randomly select a new possible move to θ' (e.g. using multivariate normal dist.)
3. Compute r. Sample $x \sim U(0,1)$.
4. If $x \leq r$, then accept and move $\theta \leftarrow \theta'$, else reject.
5. Go to step 2.

Compute ratio r:

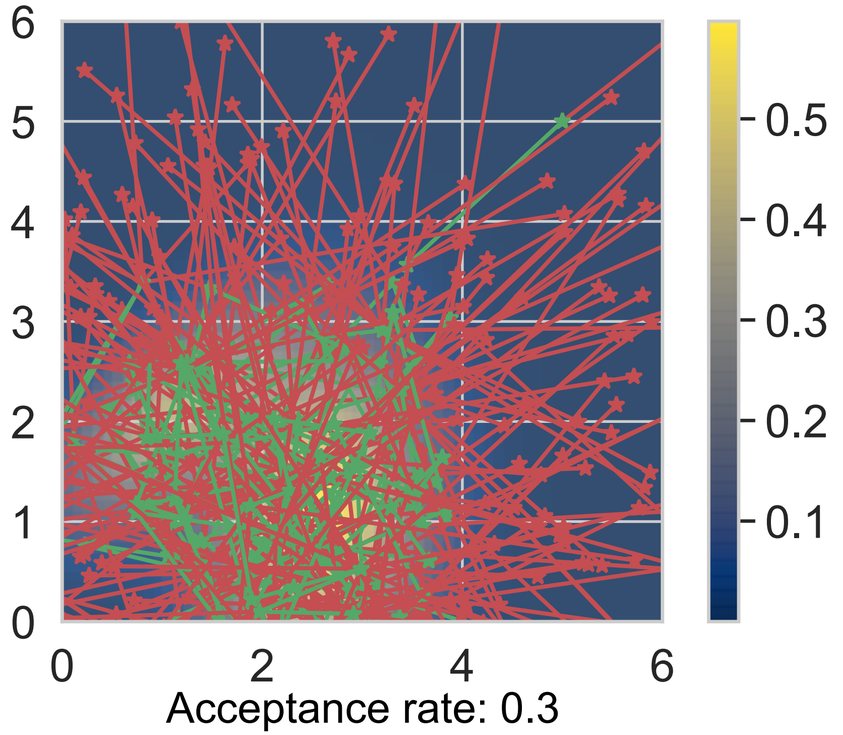
if $f(\theta') \geq f(\theta)$
then $r = 1$
else $r = \dfrac{f(\theta')}{f(\theta)}$

How do we select the proposal distribution?

Proposal Distribution and Step Size



Too small step sizes (configured using e.g. variance of the proposal distribution) lead to:
- Too many steps to get to the typical set.
- Explores slowly the whole posterior.



Larger step sizes lead to:
- Better exploration of the posterior.
- High rejection rate.

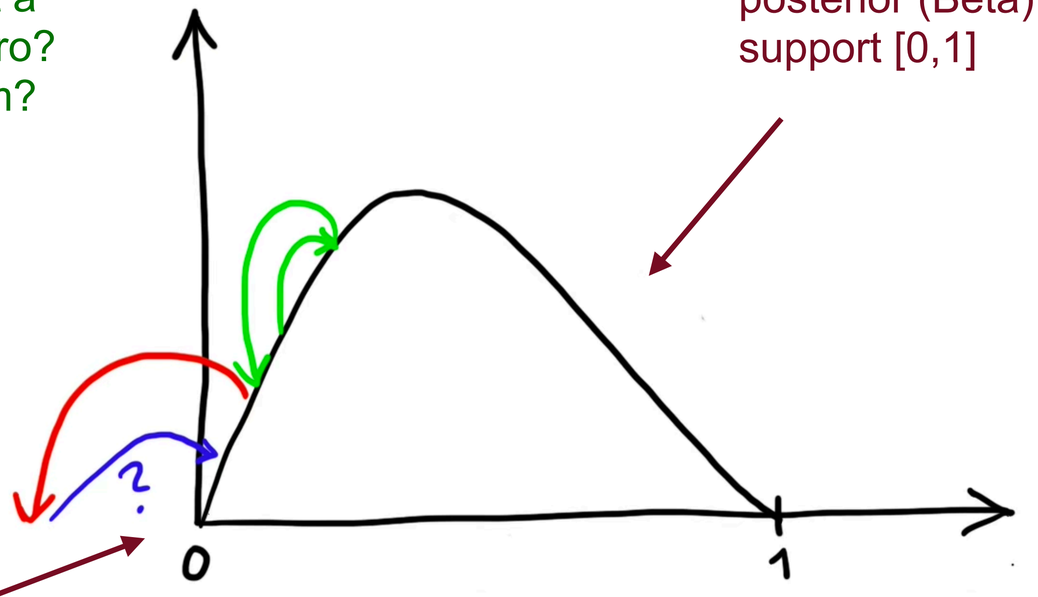
Earlier investigations show that an acceptance rate of 0.23 is good for higher dimensional models (Lambert, 2018)

Problems with Random Walk Metropolis

Random walk Metropolis has a few problems:
1. It cannot handle constrained parameter spaces
2. Proposal distributions need to be symmetric.

Discussion: Can we simply reject a Metropolis step if it goes below zero? Why/why not? What is the problem?

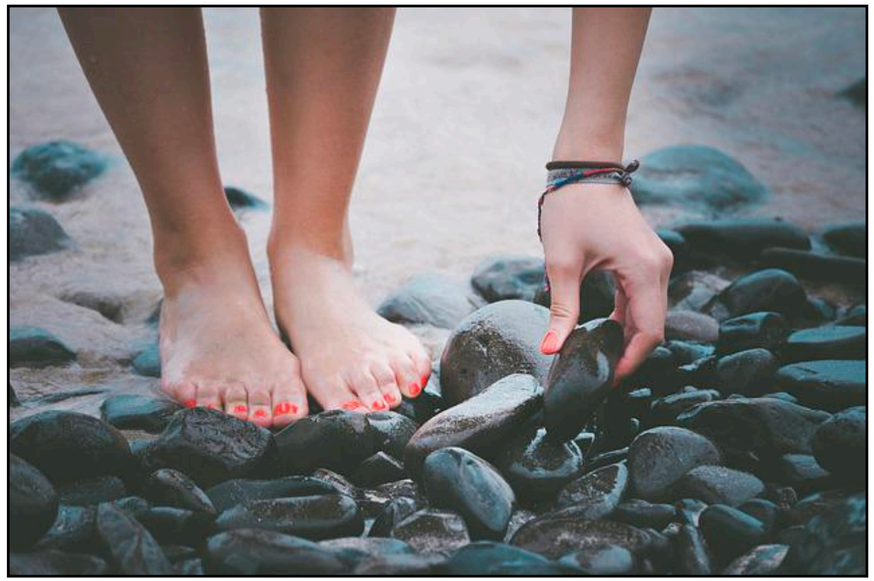
Consider the following simple constrained posterior (Beta) with support [0,1]



There will be too few samples just to the right of zero, because contributing steps only come from the right, not from the left of zero.

Inference Methods Overview

- Sampling with the CDF
- Rejection Sampling
- Importance Sampling
- Sequential Monte Carlo (SMC)
- Random walk Metropolis
- Metropolis - Hastings
- Hamiltonian Monte Carlo (HMC)



Metropolis-Hastings

Solution: Metropolis-Hastings, that can handle asymmetric proposals!

Procedure (same as for Metropolis):

1. Select a start position θ.
2. Randomly select a new possible move to θ' (e.g. using multivariate normal dist.)
3. Compute r. Sample $x \sim U(0,1)$.
4. If $x \leq r$, then accept and move $\theta \leftarrow \theta'$, else reject.
5. Go to step 2.

Compute ratio r:

if $f(\theta') \geq f(\theta)$

then $r = 1$

else $r = \dfrac{f(\theta')}{f(\theta)} \dfrac{J(\theta \mid \theta')}{J(\theta' \mid \theta)}$

New way of computing the ratio.

The first fraction is the same as for random walk Metropolis.

The second fraction compensates for asymmetric proposals (where J is the proposal distribution)

Question: What happens if $J(\theta \mid \theta') = J(\theta' \mid \theta)$?

Answer: then the proposal distribution is symmetric, and it collapses to random walk Metropolis.

Problems with Metropolis-Hastings

Is Metropolis-Hastings efficient? Are there better alternatives? Problems?

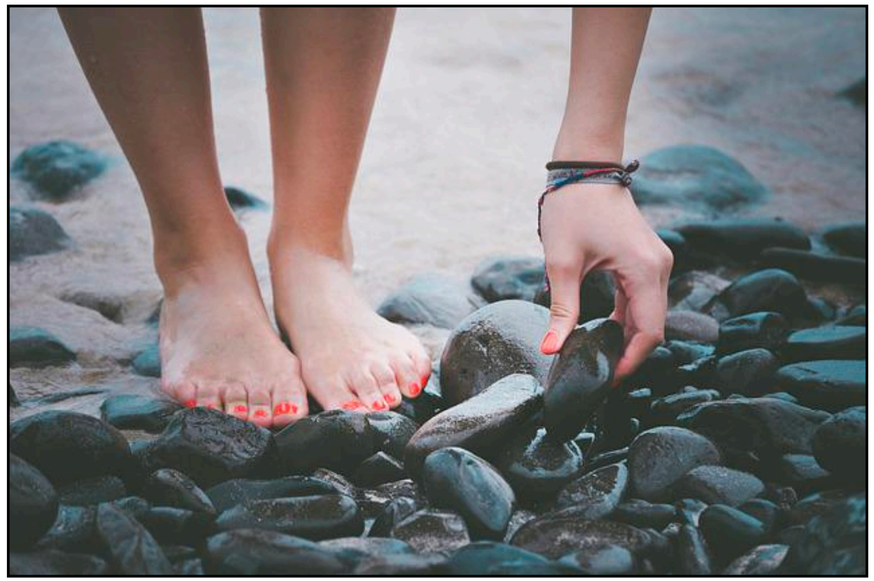
Metropolis-Hastings is blindfolded. It does not make use of the underlying shape of the posterior.

If we can see a few meters in the fog, we have an idea about the direction.

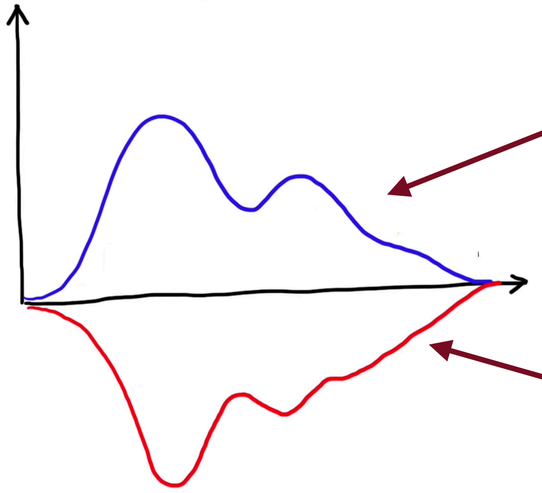
Solution: Hamiltonian Monte Carlo

Inference Methods Overview

- Sampling with the CDF
- Rejection Sampling
- Importance Sampling
- Sequential Monte Carlo (SMC)
- Random walk Metropolis
- Metropolis - Hastings
- Hamiltonian Monte Carlo (HMC)



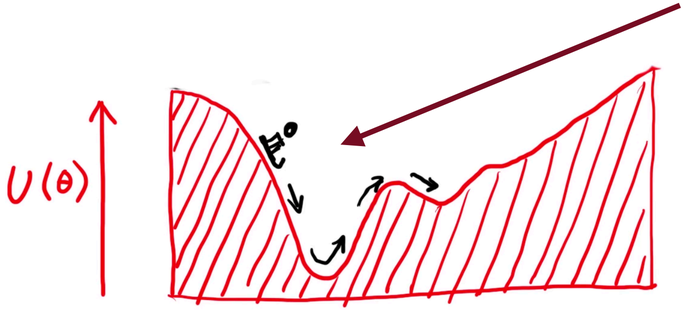
Hamiltonian Monte Carlo (HMC)



Posterior (in blue)

HMC uses an inverted landscape: Negative Log Posterior (NLP) (in red)

The "sledge" descends in NLP space (ascends in posterior space).



Idea: Model the problem using total energy (Hamiltonians), summing up potential energy $U(\theta)$ and kinetic energy $KE(m)$, which depends on the momentum m.

Key steps (informal):

1. Initiate the momentum.
2. The sledge moves over NLP space for a time T.
3. Stop. Compute r (also compensate for momentum) and check if accept or reject.

The U-turn problem: sledge can come back (forces shorted time steps, if not handled)

Solution: No-U-Turn Sampler (NUTS). Monitors the distance. Used in Stan.

Conjugate Priors and Delayed Sampling

In certain special cases, we can compute the posterior analytically!

Special pairs of likelihoods and priors

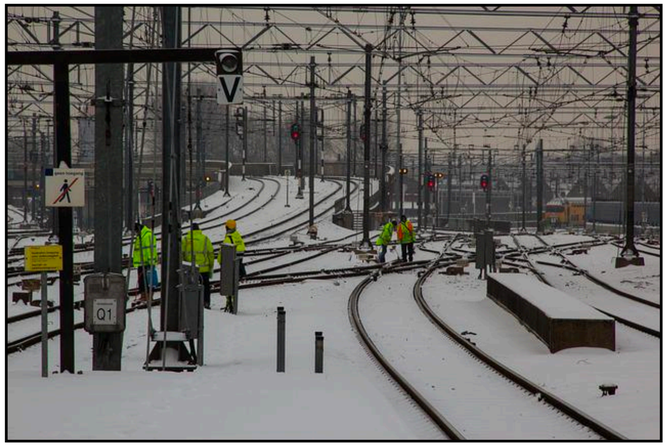
- Bernoulli (likelihood) and Beta (prior)
- Binomial (likelihood) and Beta (prior)
- Poisson (likelihood) and Gamma (prior)
- Multinomial (likelihood) and Dirichlet (prior)
- … and more.

Such priors are called **conjugate priors**

Example with Bernoulli-Beta

With n number of observations $\{x_i\}$

Likelihood	**Prior**	**Posterior**
Bern	$\text{Beta}(\alpha, \beta)$	$\text{Beta}(\alpha + \sum_{i=1}^{n} x_i, \beta + n - \sum_{i=1}^{n} x_i)$



Can we automate this?

Yes! See our work on delayed sampling.

Lawrence M. Murray, Daniel Lundén, Jan Kudlicka, David Broman, and Thomas B. Schön. **Delayed Sampling and Automatic Rao-Blackwellization of Probabilistic Programs.** In *Proceeding of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*, Lanzarote, Canary Islands, 2018.

Part II
Modeling in Stan

Copyright Los Alamos National Laboratory

Stan

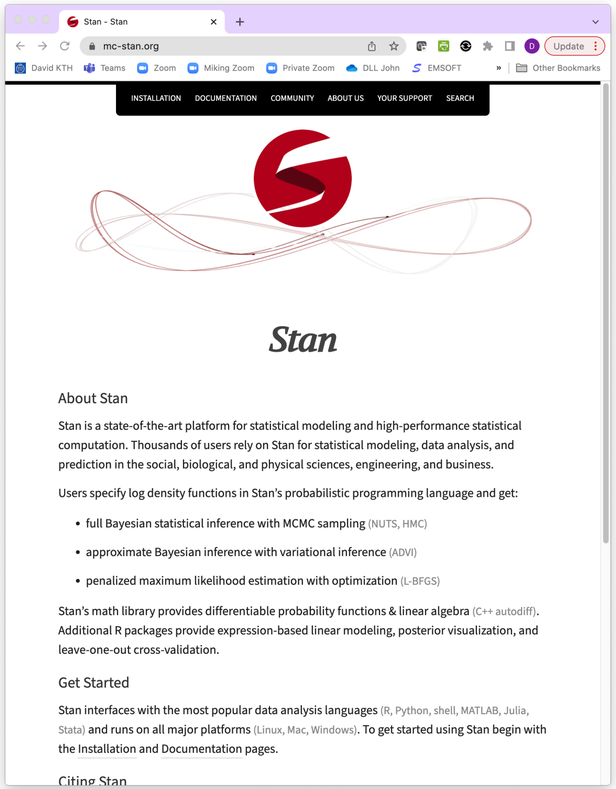
Copyright Los Alamos National Laboratory

The language is named Stan to honor Stanislaw Ulam (made significant contributions to Monte Carlo methods)

- Stan is a probabilistic programming/modeling language (imperative and Turing complete).
- The number of random variables (parameters) is fixed at compile time (not a universal PPL in the sense of WebPPL).
- It is very fast, using a special version of Hamiltonian Monte Carlo (HMC) called No-U-Turn (NUTS).
- It also supports Black-box variational inference.

Designed by Carpenter et al. and developed by the Stan team. See:

Carpenter, B., Gelman, A., Hoffman, M.D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P. and Riddell, A., 2017. **Stan: A probabilistic programming language**. Journal of statistical software, 76(1).



https://mc-stan.org/

Please check out the comprehensive documentation.

Coin Flip - Hello World!

A Stan model consists of a number of **blocks**.

Model: Compute the posterior probability after one flip (head)

Block "parameters":
Specifies random variables that will be inferred.

Here, one parameter, the probability of head. Note that type must be a real with the interval [0,1]

Why?
Answer: constrained by the support of the Beta distribution.

```
parameters {
  real <lower=0, upper=1> p;
}

model {
  p ~ beta(2.0,2.0);
  1 ~ bernoulli(p);
}
```

The prior distribution for the **latent variable** p is the Beta distribution.

Block "model":
Specifies the probabilistic model by connecting data, parameters, and distributions.

The observations have fixed values left to $\sim$.

In this case, we observe a "head" (integer value 1) using the Bernoulli distribution.

Installing and Running Stan with PyStan

Import PyStan and libraries for plotting.

The model is defined as an inline string, or in an external file.

Simple to install using python3

```
python3 -m pip install pystan
```

Install other packages

```
python3 -m pip install pandas seaborn
```

Many alternatives exist, for example for R, Matlab, command line etc. See https://mc-stan.org/

Compiles the model.

Using HMC (NUTS) to compute samples. In this case 4 times 5000 (runs 4 chains in parallel).

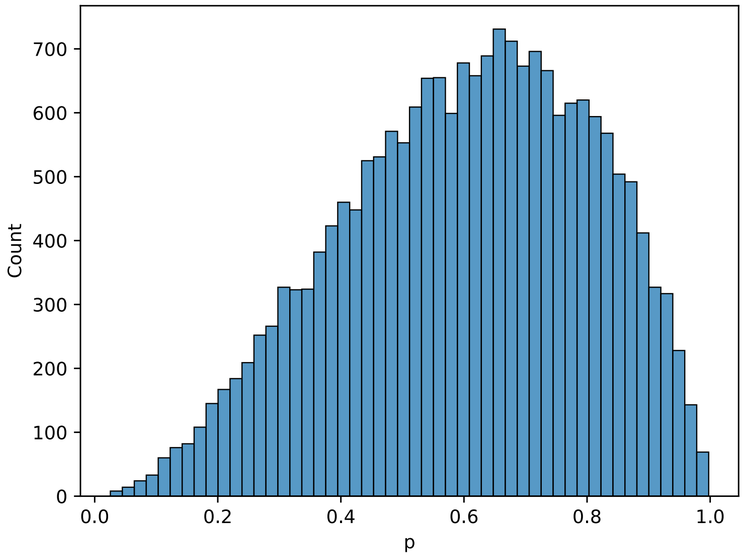
Display summary statistics using the pandas package.

Plot using package `seaborn`.

```
import stan
import seaborn as sns
import matplotlib.pyplot as plt

model = """
parameters {
  real <lower=0, upper=1> p;
}

model {
  p ~ beta(2.0,2.0);
  1 ~ bernoulli(p);
}
"""

posterior = stan.build(model)

fit = posterior.sample(num_chains=4, num_samples=5000)
df = fit.to_frame()
print(df.describe().T)
sns.histplot(data=df, x="p", bins=50)
plt.savefig('plot.pdf')
```

hello-coin.py

Installing and Running Stan with PyStan



	count	mean	std	min	25%	50%	75%	max
parameters								
lp__	20000.0	-3.927449	0.796823	-12.090559	-4.113380	-3.615265	-3.421164	-3.365058
accept_stat__	20000.0	0.909745	0.135527	0.037345	0.878151	0.969057	1.000000	1.000000
stepsize__	20000.0	0.969405	0.060270	0.921199	0.934746	0.941851	0.976509	1.072718
treedepth__	20000.0	1.381600	0.490605	1.000000	1.000000	1.000000	2.000000	3.000000
n_leapfrog__	20000.0	2.397000	1.088508	1.000000	1.000000	3.000000	3.000000	7.000000
divergent__	20000.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
energy	20000.0	4.437345	1.067545	3.365103	3.672399	4.113744	4.836225	12.497514
p	20000.0	0.602667	0.200825	0.025453	0.459164	0.617964	0.760641	0.997623

```
import stan
import seaborn as sns
import matplotlib.pyplot as plt

model = """
parameters {
  real <lower=0, upper=1> p;
}

model {
  p ~ beta(2.0,2.0);
  1 ~ bernoulli(p);
}
"""

posterior = stan.build(model)

fit = posterior.sample(num_chains=4, num_samples=5000)
df = fit.to_frame()
print(df.describe().T)
sns.histplot(data=df, x="p", bins=50)
plt.savefig('plot.pdf')
```

hello-coin.py

Run using: `python3 hello-coin.py`

Using Data and Iterating Over Arrays

Block "data":
Declares data that will be observed.

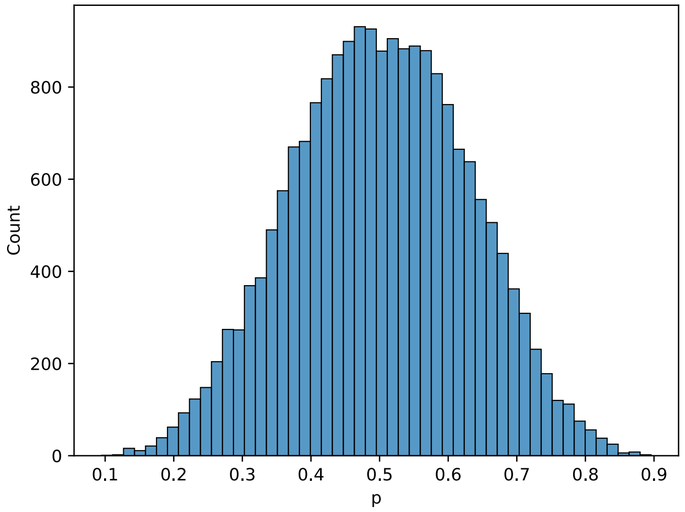
```
data {
  int<lower=0> N;
  array[N] int flips;
}

parameters {
  real <lower=0, upper=1> p;
}

model {
  for(i in 1:N) {
    flips[i] ~ bernoulli(p);
  }

  p ~ beta(2.0,2.0);
}
```

coin-flips.stan

An integer N states the number of coin flips.

The data is declared as an integer array of size N.

Iterate over the array and observe each element.



	count	mean	std
p	20000.0	0.501788	0.128438

Question: Given the data flips=[1, 0, 1, 1, 0, 1, 0, 1, 0, 0], where N=10, what do we expect the posterior of *p* to be?

Loading Data and Running From Python

Create a dictionary with the data. Note that the string names must match.

Read the model from a Stan file.

```
import stan
import seaborn as sns
import matplotlib.pyplot as plt

mdata = {"flips" : [1, 0, 1, 1, 0, 1, 0, 1, 0, 0]}
mdata["N"] = len(mdata["flips"])

with open("coin-flips.stan") as f:
    model = f.read()

posterior = stan.build(model, data=mdata, random_seed=1)

fit = posterior.sample(num_chains=4, num_samples=5000)
df = fit.to_frame()
print(df.describe().T)
sns.histplot(data=df, x="p", bins=50)
plt.savefig('plot.pdf')
```

coin-flips.py

Compile the model. Pass the data. In this case, we use a **seed** to get a same result each time we run.

Print and plot as in the previous example.

Some Alternatives

```
data {
  int<lower=0> N;
  array[N] int flips;
}

parameters {
  real <lower=0, upper=1> p;
}

model {
  for(i in 1:N) {
    flips[i] ~ bernoulli(p);
  }

  p ~ beta(2.0,2.0);
}
```

coin-flips.stan

Instead of writing an explicit `for` loop, we can use a vector operation (implicitly mapping each element)

```
model {
  flips ~ bernoulli(p);
  p ~ beta(2.0,2.0);
}
```

We can also explicitly add to the accumulated log weight (global variable called **`target`** in Stan).

Note the explicit `log-pmf` function.

```
model {
  for(i in 1:N) {
    target += bernoulli_lpmf(flips[i] | p);
  }
  p ~ beta(2.0,2.0);
}
```

Log Weights and the Log-Sum-Exp Trick

Why log weights?

- **Numerically more stable** when approximating real numbers using floating-point numbers on a computer.
- **Efficiency**: Where we would have needed multiplications and divisions, we instead perform additions and subtractions on log weights.

Sometimes we want to compute the sum of logs, using a LogSumExp (LSE) function.

Definition (LSE):

$$LSE(x_1, \ldots, x_n) = \textbf{log}(\textbf{exp}(x_1) + \cdots + \textbf{exp}(x_n))$$

A naive implementation is numerically unstable.

Solution: The log-sum-exp trick (stable):

$$LSE(x_1, \ldots, x_n) = \textbf{log}(\textbf{exp}(x_1 - x_{max}) + \cdots + \textbf{exp}(x_n - x_{max})) + x_{max}$$

where $x_{max} = \textbf{max}(\{x_1, \ldots, x_n\})$

Assignments (If Time Allows)

Implement the following assignments using either Stan or WebPPL.

Task A. Go to https://people.kth.se/~dbro/ppl-tutorial-june-2022.html (you got the link in your email) and download the slides for Stan (just presented).

A1. Type in and test the model with many coin flips. Play around.

A2. Rewrite the model using conjugate priors.

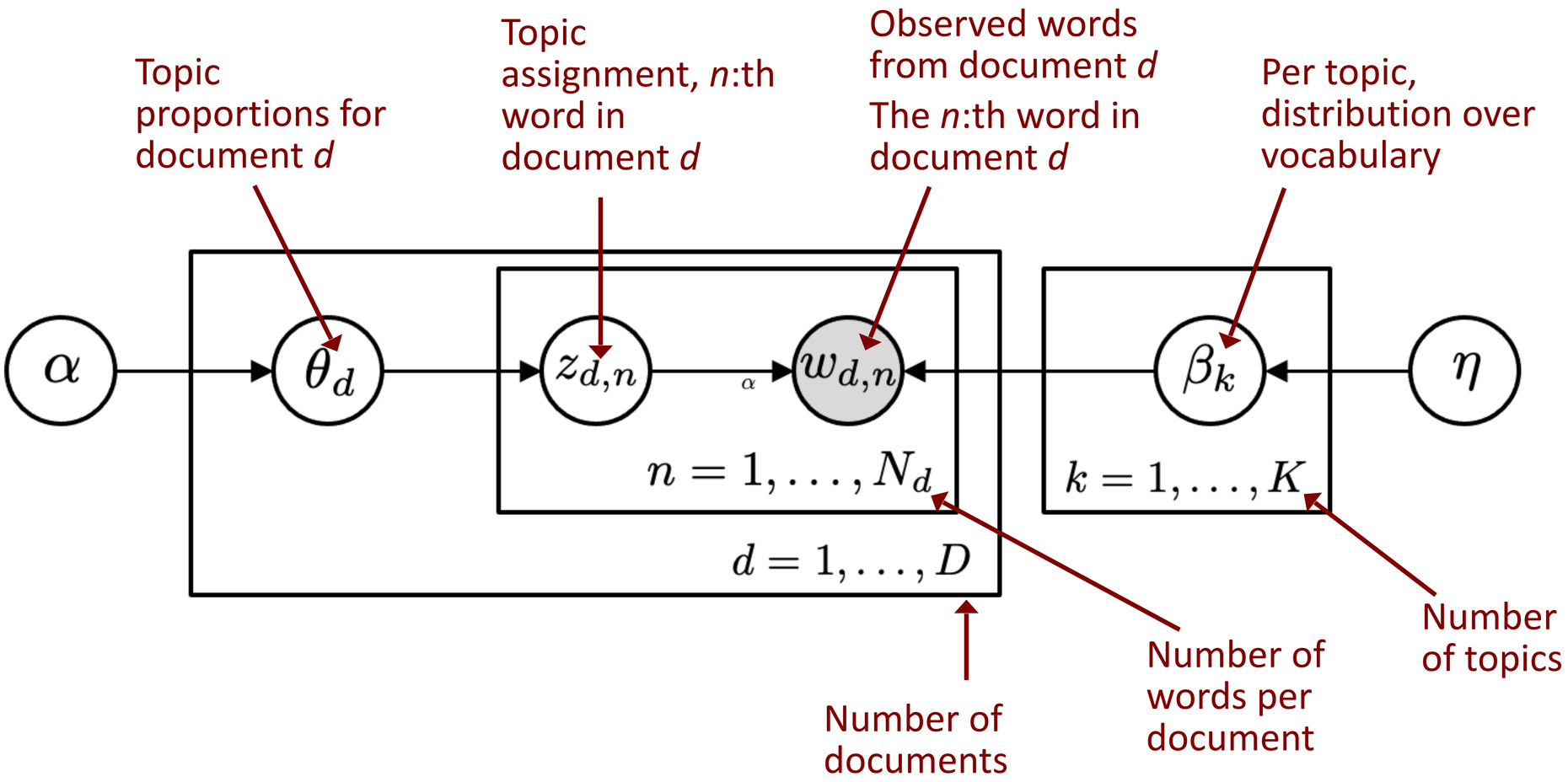
Task B. Suppose you have the following data: `length = [55, 57, 52, 64, 53, 64]` and `age = [4, 10, 2, 17, 6, 20]`, where the tuple `(length[i], age[i])` represents the length in cm and age in weeks for a baby. Create a WebPPL/Stan script that infers a posterior distribution over the length of six months old babies by using Bayesian linear regression of the form $l_i \sim N(\alpha + \beta a_i, \sigma)$, where N is the normal distribution, l_i the length, a_i the age, and α, β, and σ are random variables.

B1. Visualize the result and compute the expected value (in WebPPL, use function `expectation(INF)`, where `INF` is the call to the Infer function). What is the uncertainty of the estimation? Discuss the results.

B2. Discuss what reasonable priors for α, β, and σ can be. Test and explain how different priors affect the result.

B3. Suppose you extend the data set with one more data point, where the length is 100 cm, and the age is 5 weeks. How do the mean value and the uncertainty in the estimation change?

Latent Dirichlet Allocation (LDA)



Other Concepts

- **Posterior Predictive Checks (PPC)** - from the inferred model, generate fake data and compare with original data. In Stan, use block “generated quantities”. See p. 289 (Lambert, 2018)
- **Cross-validation:** k-fold cross-validation. Rotate test and training sets.
- **Validation set:** divide into training, validation, and test set, where the validation set is used, for instance, to tune hyper parameters.
- Stan does not support **discrete parameters** directly (because of HMC). However, discrete parameters can be used by marginalizing them out (combining blocks “transformed parameters” and “model”). See p. 401 - 406 (Lambert, 2018).

Are we already done?

References and Further Reading (1/2)

Books

Christopher M. Bishop. **Pattern Recognition and Machine Learning**, Springer, 2006

Ben Lambert, **A Student's Guide to Bayesian Statistics**, ISBN: 978-1473916364, SAGE Publications, 2018

John N McDonald and Neil A Weiss, **A Course in Real Analysis**, 2nd Edition,
ISBN: 978-0123877741, Academic Press, 2012

Kevin P. Murphy, **Probabilistic Machine Learning: An Introduction**, MIT Press, 2022. Available online: https://probml.github.io/pml-book/book1.html

Kevin P. Murphy, **Probabilistic Machine Learning: Advanced Topics**, MIT Press, 2023. Available online: https://probml.github.io/pml-book/book2.html

Judea Pearl, **The book of why: the new science of cause and effect**,
ISBN: 978-0141982410, Penguin Books, 2019

Derek Rowntree, **Statistics without Tears**, ISBN: 978-0141987491,
Penguin Books Ltd, 2018

Stuart Russell and Peter Nervig, **Artificial Intelligence: A Modern Approach**,
4th Edition, Pearson, ISBN: 978-0134610993, 2020

References and Further Reading (2/2)

Papers

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. **Latent Dirichlet Allocation**. Journal of Machine Learning Research, 2003.

Lawrence M. Murray, Daniel Lundén, Jan Kudlicka, David Broman, and Thomas B. Schön. **Delayed Sampling and Automatic Rao-Blackwellization of Probabilistic Programs**. In the Proceeding of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS), Lanzarote, Canary Islands, 2018.
Online: http://proceedings.mlr.press/v84/murray18a

Daniel Lundén, Johannes Borgström, and David Broman. **Correctness of Sequential Monte Carlo Inference for Probabilistic Programming Languages**. In Proceedings of 30th European Symposium on Programming (ESOP 2021), LNCS vol. 12648, Springer, 2021.
Online: https://link.springer.com/chapter/10.1007/978-3-030-72019-3_15

Lawrence M. Murray, Anthony Lee, and Pierre E. Jacob. **Parallel Resampling in the Particle Filter**. Journal of Computational and Graphical Statistics, 25:3, 789-805, Taylor & Francis, 2016.
Online: https://doi.org/10.1080/10618600.2015.1062015

Christian A. Naesseth, Fredrik Lindsten, Thomas B. Schön. **Elements of Sequential Monte Carlo**. ArXiv. https://arxiv.org/abs/1903.04797, 2019.

Assignments (Optional)

- If you are participating in the course, send your solution at the latest at 11 am on Thursday, June 23. Submit the solution as a Zip file, where you include code, graphs, and discussions that explain your solutions (explanations in a PDF file).
- The code tasks must be executable, easy to install, with a reasonable README file.
- If you are not in the course, you do not need to send any solutions.

A1. Implement the LDA model in Stan. Import data using the format defined here http://archive.ics.uci.edu/ml/datasets/Bag+of+Words. Test with small data sets (designed by yourself), as well as some of the ones in the above repo. Make sure to print out the top words in each topic. You may take inspiration from existing solutions (a partial solution is available on the Stan website) but you need to write all the code yourself.

Conclusions

Some key take away points:

- Several approximate inference algorithms use **sampling techniques** (rejection sampling, SMC, MCMC variants etc.)
- Modern **MCMC algorithms**, such as HMC, scales better to high-dimensional problems in PPLs.
- **Stan** is a PPL that implements HMC and NUTS.
- I might give a **follow-up tutorial** in the fall, focusing on advanced PPL techniques and our research results.

Thanks for listening!