Computing for Economists

Numerical Simulation and Monte Carlo Methods

Toren Fronsdal and Ruby Zhang

Harvard University

Introduction

Goal: how do we sample from complex distributions with accurate coverage while subject to computational resource constraints?

Following resources:

- Ken Judd Course
- Jesús Fernández-Villaverde Course
- Tommaso Rigon Course
- “Computer Age Statistical Inference” by Efron & Hastie

Table of Contents

1. Basic Random Sampling

2. Advanced Random Sampling

3. Markov Chain Monte Carlo (MCMC)

4. Alternative Monte Carlo Methods

5. Variational Inference

6. Python Implementation

7. Resampling Methods

Basic Random Sampling

Monte Carlo Methods

- Idea: use repeated, random sampling to solve complex numerical problems that are deterministic in nature
- Developed by physicist Stanislaw Ulam and inspired by the Monte Carlo Casino where his uncle frequented
- Foundational to generating samples from an underlying probability distribution
- Basic outline:
  1. “Random” numbers are generated deterministically
  2. Special distributions transform “random” uniform draws
  3. Sampling from complex distributions: acceptance-rejection, MCMC
- `numpy.random` package is your friend

Pseudo-Random Number Generator (PRNG)

- Computers draw a “random” numbers via deterministic sequences starting from a seed x_0
  - Sequences pass statistical tests of unbiasedness and zero serial correlation

- Classic: linear congruential generator (LCG) for random integers $x_i \in \{0, 1, .., M\}$:

$$x_i = (ax_{i-1} + b) mod (M + 1)$$

  - Random uniform distribution on $[0, 1]$:

$$x_i = \frac{ax_{i-1} + b}{M + 1}$$

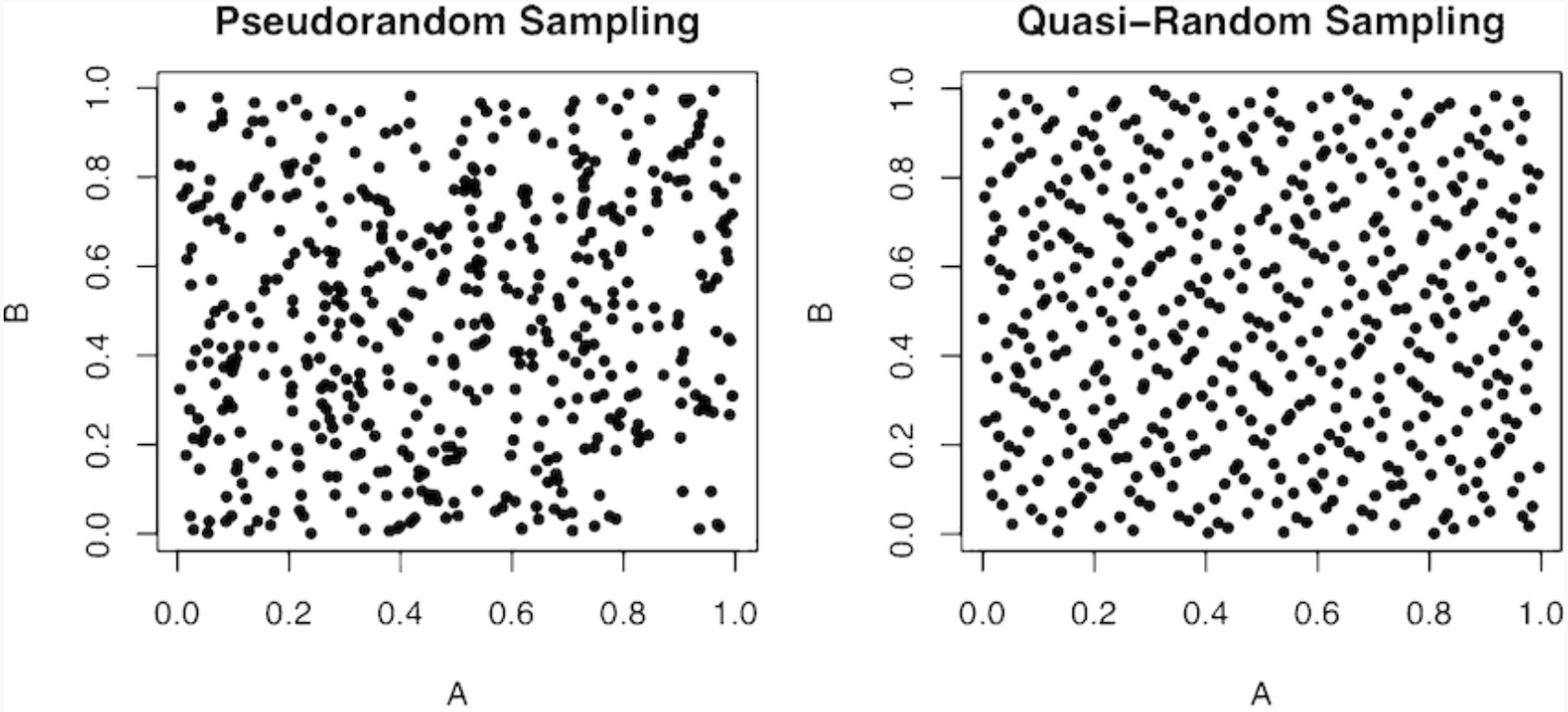
  - Typical default hyperparameters: $a = 7^5, b = 0, M = 2^{31} - 1$ (for long cycles)

- Better PRNGs now: Mersenne Twister and PCG family (NumPy switched from Mersenne Twister to PCG64)

Low Discrepancy Sequences (LDS)

- LDS are *quasi-random* numbers that are uniformly distributed
  - Variance reduction and faster convergence (for low dimensions)
- The numbers are *not* pseudo-random and would fail statistical tests due to the uniformity
- Popular examples (rearrange digits in different bases):
  - Sobol: use the base two numbers and re-order coordinates in subsequent dimensions
  - Halton: use a different prime number base for each dimension
  - Faure: permutations of Halton sequences to provide better uniformity

PRNG vs. LDS



Source

Inverse Cumulative Distribution Function (CDF)

- Possible to generate random draws from any distribution using random uniform draws on $[0, 1]$
  - Use *`numpy.random.uniform()`* in Python
- For U uniform on $[0, 1]$ and some invertible CDF F, the random variable $X = F^{-1}(U)$ is distributed according to F
  - Example: $X \sim exp(\lambda)$, then $F(x; \lambda) = 1 - e^{-\lambda x}$ so $F^{-1}(u; \lambda) = -\frac{ln(1-u)}{\lambda}$ for u drawn from Uniform$[0, 1]$
- Requires knowing and computing F^{-1}, sometimes not possible
  - Analytically complex
  - Multivariate distribution

Box-Muller Transform (1958)

- There are special ways to generate random samples from normal distributions
- If U_1, U_2 are independent uniform variables on $[0, 1]$, then X, Y are independent normal variables distributed $\mathcal{N}(0, 1)$:

$$X = cos(2\pi U_1)(-2lnU_2)^{1/2}$$

$$Y = sin(2\pi U_1)(-2lnU_2)^{1/2}$$

- Idea: joint distribution (X, Y) is circular so Box-Muller converts polar coordinates to Cartesian coordinates

Multivariate Normal

- Recall a random vector $Y = (Y_1, ..., Y_d)$ is multivariate normal if all linear combinations $\Sigma_{i=1}^{d} a_i Y_i$ are normal

- Possible to generate any multivariate normal $Y \sim \mathcal{N}(\mu, \Sigma^T \Sigma)$ from a standard multivariate normal $X \sim \mathcal{N}(0, I)$:

$$Y = \mu + \Sigma X$$

- Σ is the Cholesky decomposition of the variance-covariance matrix
    - Σ is a lower triangular matrix
    - The variance-covariance matrix is symmetric and positive definite

- In Python, this is quite easy with `numpy.random.randn()`

Copulas

- **Question:** How do we generate correlated samples for *arbitrary* multivariate distributions?

Copulas

- **Question:** How do we generate correlated samples for *arbitrary* multivariate distributions?

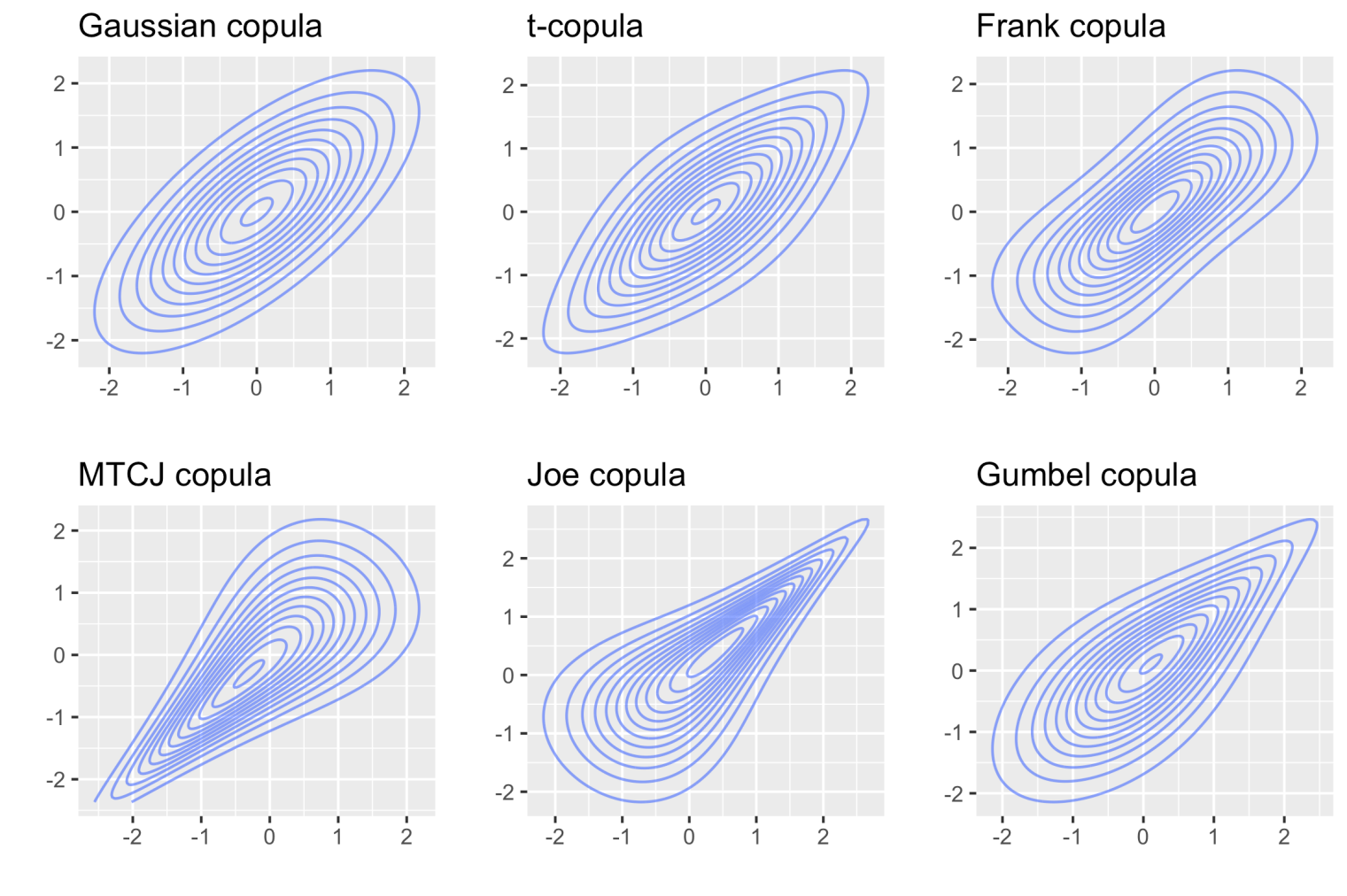
- A **copula** is the CDF that specifies the correlation structure between uniform marginals (recall inverse transform)

- Given RVs X, Y with marginals F_X, F_Y and $u_x, u_y \sim \text{Uniform}[0, 1]$:

$$C(u_x, u_y) = P(F_X(X) \leq u_x, F_Y(Y) \leq u_y)$$

Bivariate Copulas

For arbitrary (X, Y), possible bivariate copulas:



Source

Copula Families

- Gaussian copula: good for symmetric and linear dependencies
- t-copula: good for tail dependencies (i.e. extreme events are correlated)
- Archimedean copulas (specified by a generator function):
  - MTCJ/Clayton: good for strong lower-tail dependencies
  - Gumbel: good for strong upper-tail dependencies
  - Joe: good for even stronger upper-tail dependencies[1]
  - Frank: good for symmetric and non-tail dependencies

[1] Copulas are used quite often in asset pricing and the financial sector.

Sampling from a Gaussian Copula

- Suppose data (X, Y) have marginal distributions of Lognormal and Weibull with a Gaussian copula[2] defined by correlation ρ

- Steps to sample (X, Y):
  1. Generate a pair (z_x, z_y) from a bivariate normal distribution:

$$\mathcal{N} \sim \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \right)$$

  2. Transform (z_x, z_y) to uniform (u_x, u_y): $u_x = \Phi(z_x), u_y = \Phi(z_y)$
  3. Apply inverse transform method to get sample (x, y):

$$x = exp\{\mu + \sigma\Phi^{-1}(u_x)\}, \quad y = \lambda(-log(1 - u_y))^{1/k}$$

  4. Repeat draws until desired sample size

[2] Estimated via MLE or method of moments, see *PyCopula*.

Sampling from an Archimedean Copula

- Gaussian and t-copulas are commonly used by sampling directly from the multivariate distribution
- Archimedean copulas use the *conditional distribution method*
  - A generator function $\phi : [0, 1] \to [0, \infty)$ is continuous, strictly decreasing, and convex with $\phi(1) = 0$
  - Copula function: $C(u_1, ..., u_n) = \phi^{-1}(\phi(u_1) + ... + \phi(u_n))$
- **Idea:** sample independent uniforms (u_x, v) and calculate u_y based on inverse of copula generator function (see Nelsen seminar)

$$u_y = \phi^{(-1)}(\phi(w) - \phi(u_x)) \text{ where } w = \phi'^{(-1)}(\phi'(u_x)/v)$$

Copulas in Economics

- Copulas have cropped up in all sorts of economic applications involving correlated data (apart from finance)

- Examples:
  - Econometrics: instrument-free inference using copulas to model endogeneity (Qian et al., 2024; Yang et al., 2023)
  - Public: joint income distributions that are related (Chetty et al., 2017; Golosov et al., 2014)
  - Marketing: unobserved knowledge in manager decision-making (see Table 1 of Qian et al., 2024)

Advanced Random Sampling

Accept-Reject Sampling

- Suppose we wish to sample from a *target* distribution $f(x)$ but only know how to draw from a *source* distribution $g(x)$

- We can use accept-reject sampling if:
  1. The support of $g(x)$ covers at least that of $f(x)$
  2. Ratio of $f(x)$ to $g(x)$ is bounded (i.e. $g(x)$ has thicker tails):

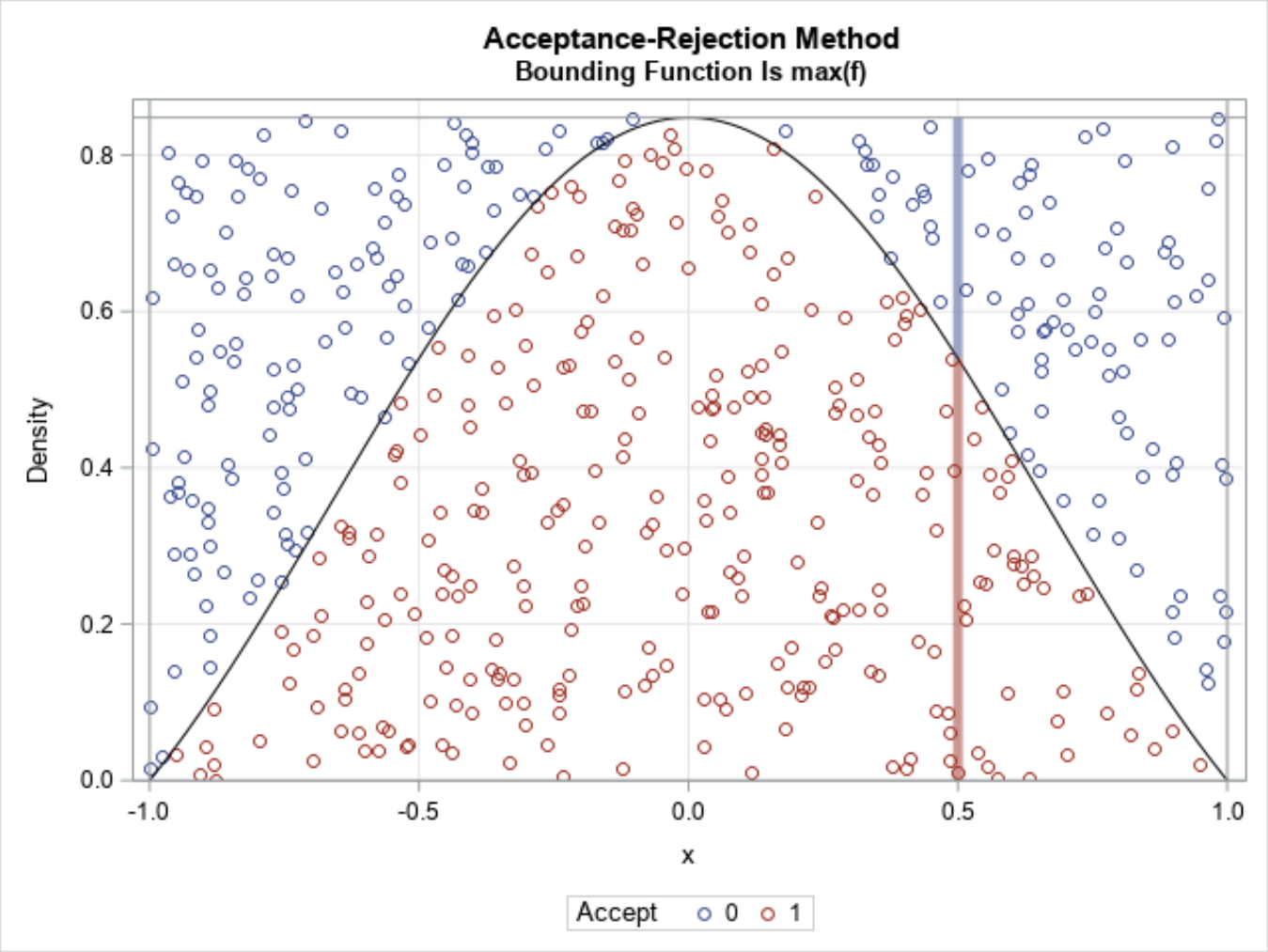
$$sup_{x \in Supp(f)} \frac{f(x)}{g(x)} = a < \infty$$

Algorithm steps:

1. Draw $u_i \sim \text{Uniform}[0, 1]$ and $x_i \sim g(x)$
2. Accept x_i into sample set if $u_i < \frac{f(x_i)}{ag(x_i)}$
3. Repeat steps until desired number of draws is reached

Accept-Reject Sampling

Note: many draws are wasted if source is poor *and* bound a is hard to calculate in practice!



Source

Importance Sampling

- An *approximation* method related to accept-reject sampling
- Idea: instead of wasting rejected draws, weight by $\frac{f(x)}{g(x)}$
- Assume the same set-up of accept-reject sampling

Algorithm steps:

1. Draw N samples of $\{x_i\} \sim g(x)$ for $i \in \{1, ..., N\}$
2. Compute weights using likelihood ratio: $w_i = \frac{f(x_i)}{g(x_i)}$
3. Compute the statistic of interest with the appropriate weights:
   - Expectation of $h(x)$ over f: $\frac{\sum_{i=1}^{N} w_i h(x_i)}{\sum_{i=1}^{N} w_i}$
   - Higher-order moments of $h(x)$ over f, numerical integration

Note that ideal $g(x) \propto |f(x)h(x)|$ can lower estimator variance

Markov Chain Monte Carlo (MCMC)

Bayesian Models

- Suppose now that your data X is drawn from some sampling distribution $f(x|\theta)$ (i.e. likelihood function)
  - Prior density of θ is given by $\pi(\theta)$
  - Then posterior density of θ is:

$$\pi(\theta|X) = \frac{\pi(\theta)f(X|\theta)}{\int \pi(\theta)f(X|\theta)d\theta}$$

- The denominator is the integration of seeing the data X over all possible values of θ – we cannot rely on analytic solutions
- The posterior mean is also hard to analytically compute:

$$\mathbb{E}[\pi(\theta|X)] = \frac{\int \theta\pi(\theta)f(X|\theta)d\theta}{\int \pi(\theta)f(X|\theta)d\theta}$$

Markov Chain Monte Carlo (MCMC)

- However, we can still *sample* from this posterior using Monte Carlo methods even without a closed analytical form
- **Idea**: Instead of any source distribution $g(x)$, let's take small steps for the proposal point
- Establish a proposal distribution that forms a *Markov chain* that converges to the target posterior $\pi(\theta|X)$
- We follow similar principle to accept-reject in order to "walk" to higher-density areas of the target

Bayesian Modeling and MCMC in Economics

Ubiquitous in economics and across fields, for example:

- "An MCMC approach to classical estimation" (Chernozhukov & Hong, 2003)
- Using quasi-Bayes to deal with weakly-identified GMMs (Andrews & Mikusheva, 2021)
- Spatial dependence and updating (Hodgson, 2021; LeSage et al., 2018)
- Estimation of DSGE models (see Fernández-Villaverde & Guerrón-Quintana (2021) for a review of computational methods)

...and many more (e.g. behavioral, public, IO, finance)

Markov Chains

- A Markov chain is a stochastic model that predicts the probability of events where future states depend *only* on the current state
- Markov chains consist of **states** and **transition probabilities** between states
- Under certain conditions, Markov chains converge to a stationary distribution as the number of steps goes to infinity
  - All states are visited, and in finite time but aperiodically
- **Goal:** Construct a Markov chain where the target posterior is the stationary distribution, this is called the Metropolis-Hastings algorithm

Metropolis-Hastings (MH) Algorithm

- Sampling from the posterior only requires the numerator since the denominator is a constant:

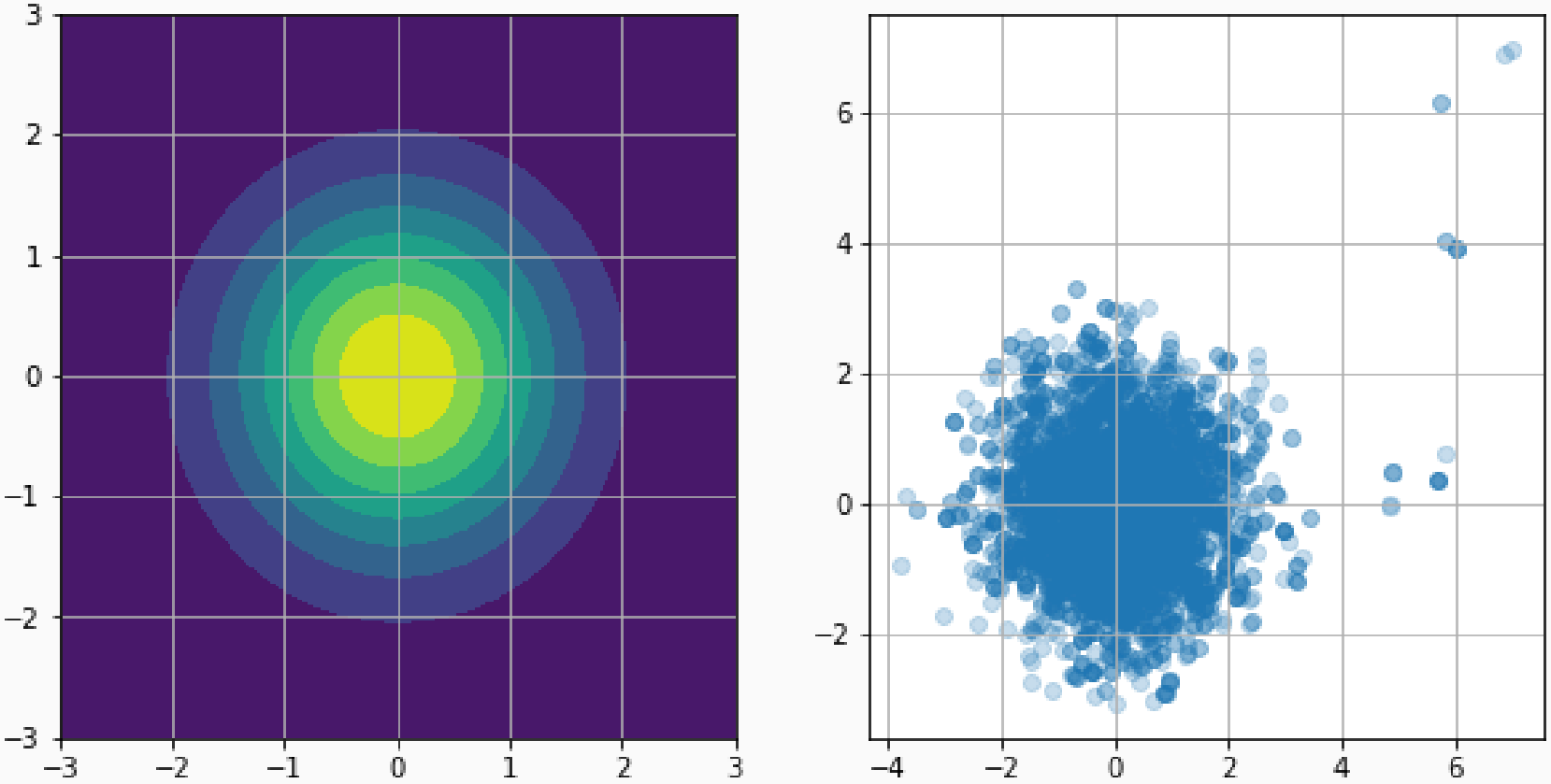
$$\text{Goal: } \pi(\theta)f(X|\theta)$$

1. Start from some value θ_0 (it should have positive density on the posterior distribution)
2. At step k for the current value θ_k, draw a proposal value $\theta_P = \theta_k + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$[3]
   - 2.1 There are other proposal distributions, e.g. can also draw θ_P from Uniform$[\theta_k - S, \theta_k + S]$
   - 2.2 Parameters σ and S govern the step-size, which governs convergence speed
3. Draw $u_k \sim$ Uniform$[0, 1]$ and accept θ_P as θ_{k+1} if

$$u_i < \frac{\pi(\theta_P)f(X|\theta_P)}{\pi(\theta_k)f(X|\theta_k)}$$

[3]This is known as Random-Walk MH.

MCMC and Bivariate Gaussian



Source

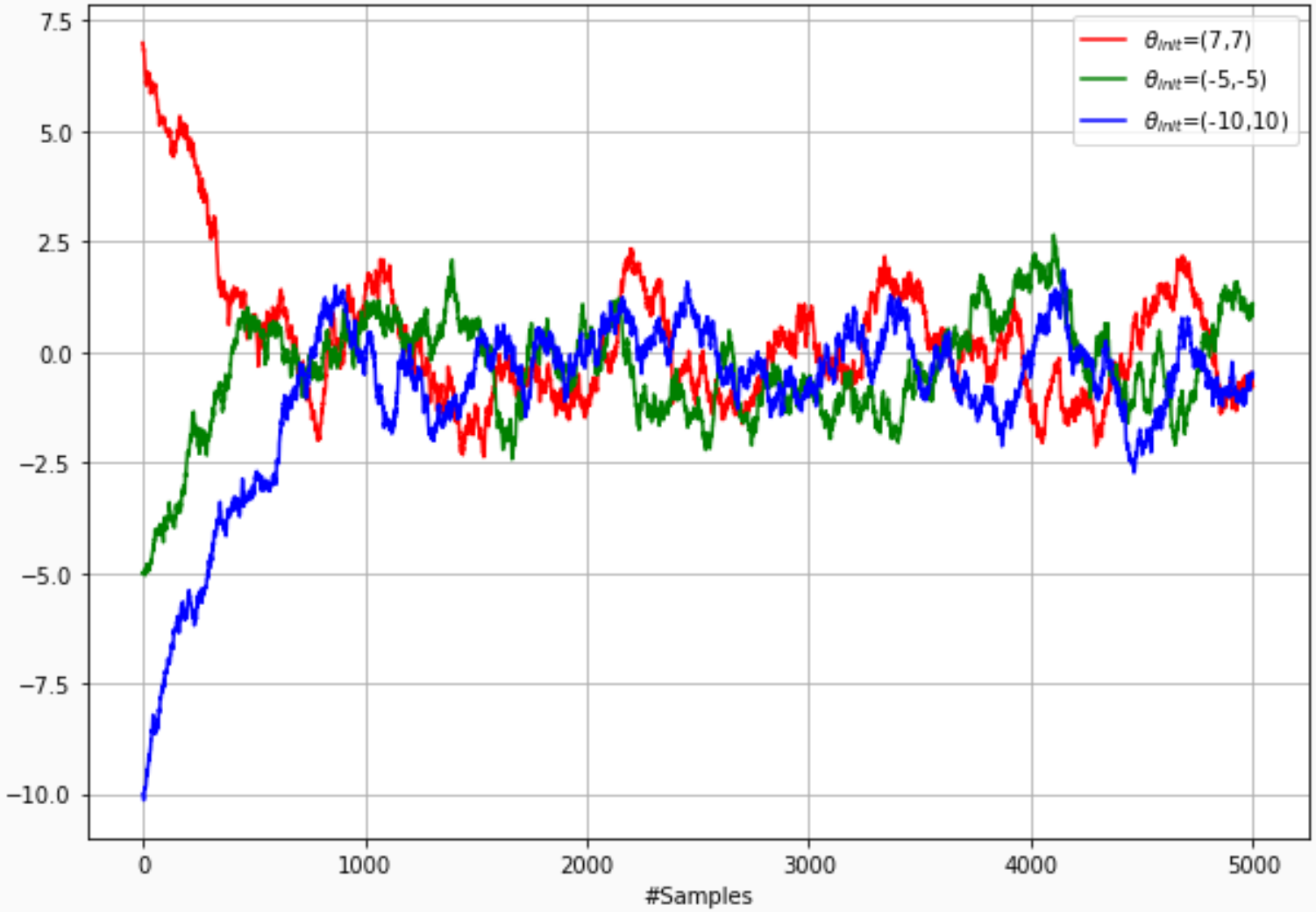
Gibbs Sampling

- Gibbs sampling is MCMC method useful for a multidimensional setting: $\theta = (\theta_1, ..., \theta_n)$
- Requires knowing the full *conditional* distribution $\pi(\theta_i|\theta_1, ..., \theta_n, X)$ and how to *sample* from it
- Special proposal distribution which successively draws parameter components from conditional:
  1. Start from some vector $\theta^{(0)}$
  2. Sample $\theta_1^{(1)} \sim \pi(\theta_1|\theta_2^{(0)}, ..., \theta_n^{(0)}, X)$
  3. Sample $\theta_2^{(1)} \sim \pi(\theta_2|\theta_1^{(1)}, \theta_3^{(0)}, ..., \theta_n^{(0)}, X)$
  4. Successively sample $\theta_i^{(1)}$ with updated parameter components
  5. This is one round of Gibbs sampling, repeat for many samples
- *All* proposals are accepted, but if parameters are correlated convergence can be slow

When Does MCMC “Converge”?

- There are no statistical tests to check, but there are proxies:
  - Trace plots: time series of sampled parameter over time
  - Multiple chains: start with many different starting values (should converge)
  - Chain length: longer chain length to ensure reaching steady-state
  - Acceptance ratio: acceptance ratio should be around 0.2 to 0.5
    - Acceptance ratios should be lower in higher dimensions
    - Large step sizes $\rightarrow$ low acceptance rates, but small step sizes $\rightarrow$ slow convergence. See Rigon slides for further discussion.
- Usual practice is to have an initial “burn-in” sample which is discarded

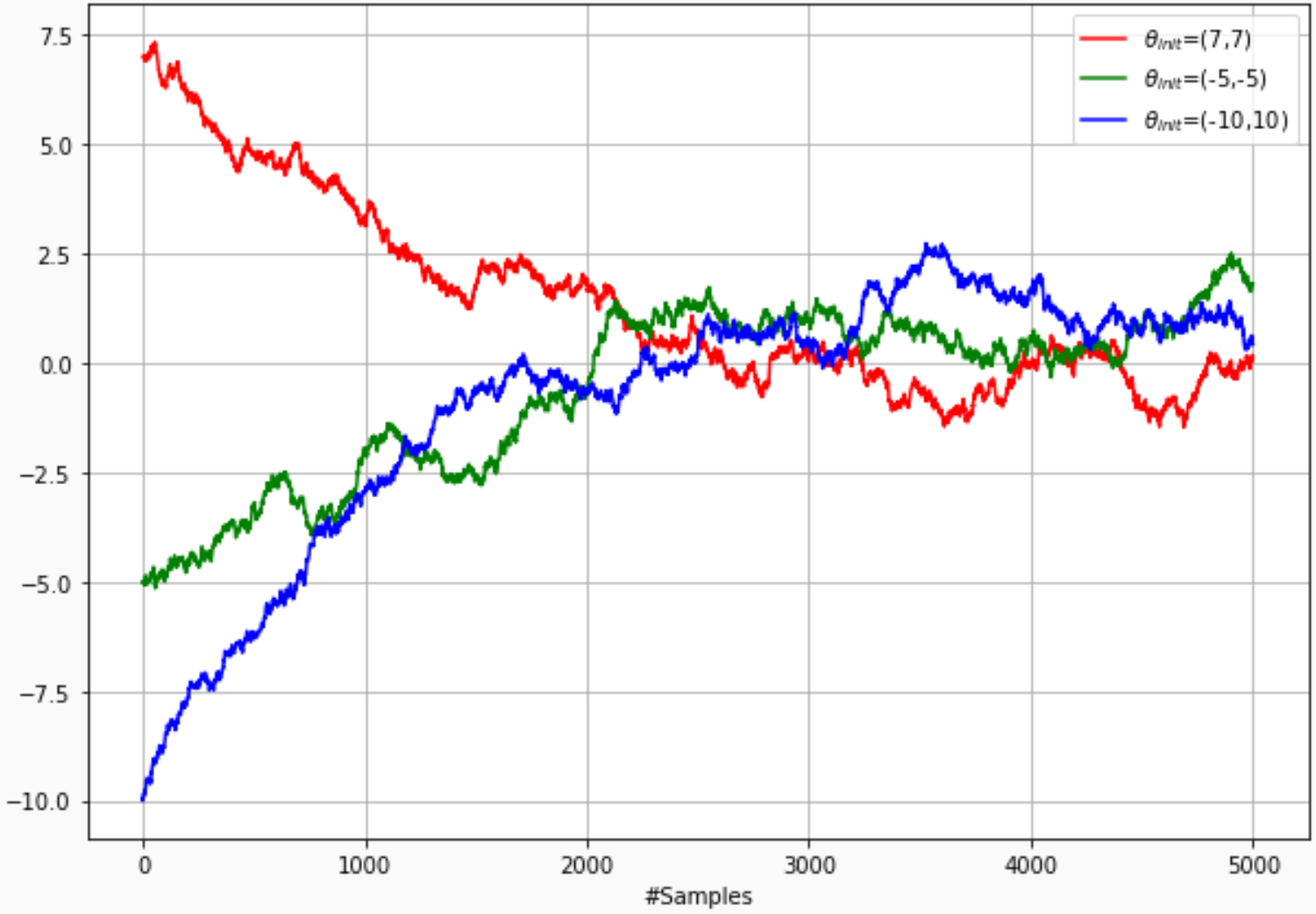
Trace Plot of Multiple Chains



Source

Trace Plot of Multiple Chains

Smaller step sizes takes the chain longer to converge



Source

Metropolis-Hastings using Gradient Information

- For complex targets, random walk MH can converge very slowly
- **Idea:** what if the proposal distribution uses information about the posterior shape?
- MALA: proposal distribution error $\epsilon \sim \mathcal{N}\left(\frac{\sigma^2}{2}\Delta log\{\pi(\theta_k|X)\}, \sigma^2 I\right)$
  - Note that $\Delta log\{\pi(\theta_k|X)\} = \Delta log\{\pi(\theta_k)\} + \Delta log\{f(X|\theta_k)\}$
  - Ascending the gradient to move towards high density areas
- Tradeoff: faster mixing vs. gradient calculation (worth it in high dimensions)
- Gradient based methods are also more fragile, need to be pre-conditioned and calibrated

Alternative Monte Carlo Methods

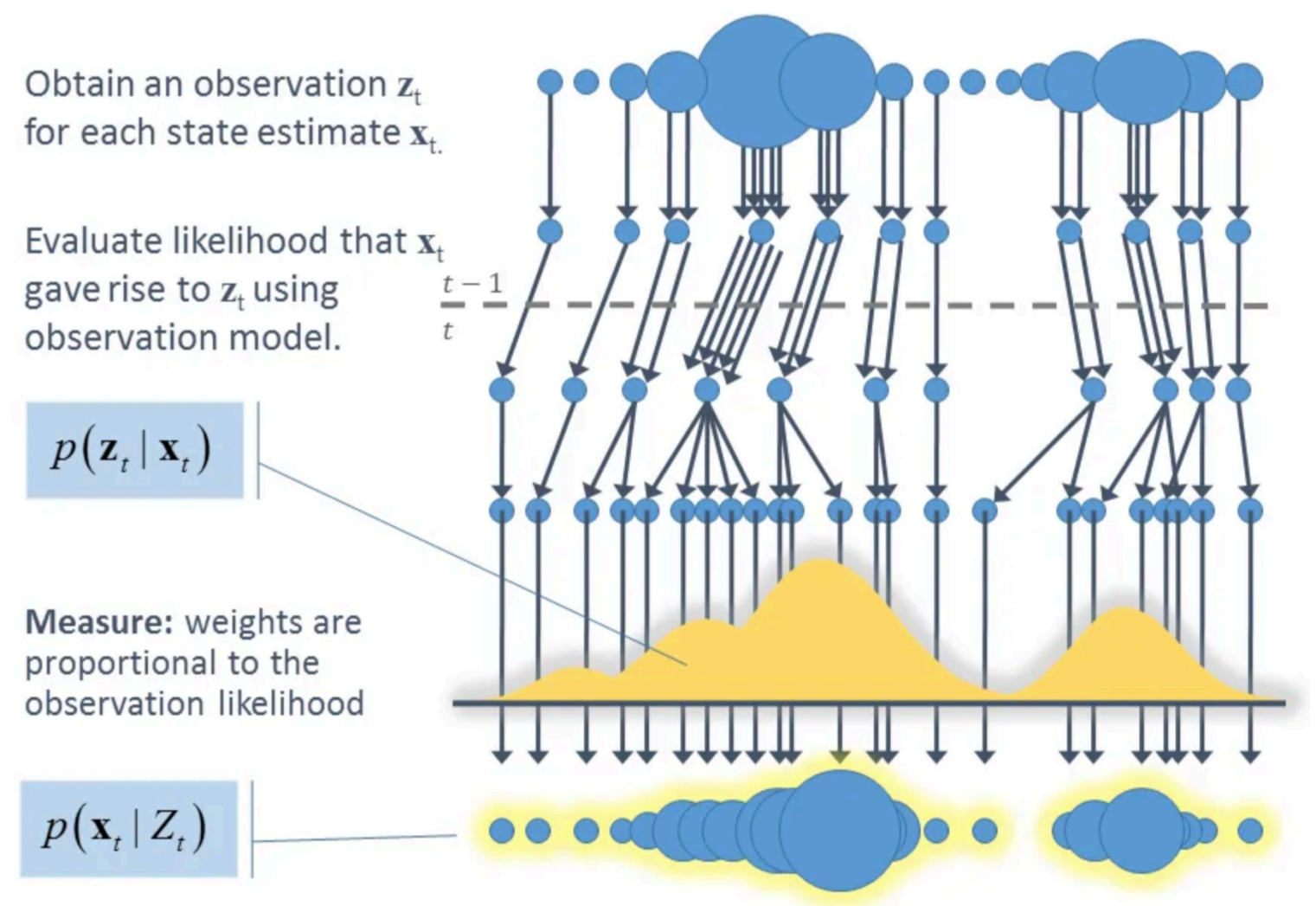
Quasi-Monte Carlo

- Instead of using stochastic sequences generated by PRNGs, use low-discrepancy sequences
- LDS cover the sample space more uniformly $\rightarrow$ **faster convergence** to target
- However, the benefits are limited to certain problems (e.g. high dimensional up to a point)
- Very complex targets may be challenging for both quasi-MCMC and traditional MCMC methods

Time Dependence: Particle Filtering

- Special case of data dependency: *time* data
- **Particle filtering** is based on sequential Monte Carlo methods
- At each time t, assume underlying state x_t that evolves via process (e.g. AR(1)) and gives rise to observation z_t
- At each step t:
  1. Generate sample of "particles" based on prior $p(x_t|x_{t-1}, z_{t-1})$
  2. Observe data z_t and re-weight particles $w_t \propto p(z_t|x_t)$
  3. Resample particles with replacement according to the weights (to prevent degeneracy)
  4. Estimate state of the system from weighted particles and form $p(x_{t+1}|x_t, z_t)$

Particle Filtering in a Nutshell



Source

Variational Inference

Why not MCMC?

- MCMC is a *sampling* method used to approximate a distribution
- This can be computationally expensive and take time to converge, especially in high dimensional settings, for example:
  - Recent VAR models in macro use large datasets (Gefang et al., 2023)
  - Multinomial probit models of choice behavior depend on many variables (Loaiza-Maya & Nibbering, 2022)
- **Idea:** What if we directly tried to approximate the posterior using parametric distributions?
- Now we have an optimization problem!

Variational Inference

- Variational inference is based on finding **the best parametric approximation**
  - **Pros:** fast to compute, no sampling chain/convergence required
  - **Cons:** accuracy is dictated by the parameterization
- Ingredients:
  - Parametric family of distributions $p_\omega \in \mathcal{P}$ with parameters $\omega \in \Omega$
  - Distance metric between distributions
- Choice of distribution family dictates tradeoff between simple and biased vs. complex and accurate
  - Should be computationally tractable and should *not* include posterior

Distance between Distributions

- Common metric is **Kullback-Leibler** (KL) divergence:

$$KL\{p_\omega(\theta), \pi(\theta|X)\} = \int p_\omega(\theta) log \left\{ \frac{p_\omega(\theta)}{\pi(\theta|X)} \right\} d\theta = \mathbb{E}_{p_\omega} \left[log \left\{ \frac{p_\omega(\theta)}{\pi(\theta|X)} \right\} \right]$$

- Note that KL divergence is *not* symmetric

- Variational Bayes problem:

$$\boxed{\omega* = argmin_{\omega \in \Omega} KL\{p_\omega(\theta), \pi(\theta|X)\}}$$

- However, since the integral contains the posterior $\pi(\theta|X)$, recall

$$\pi(\theta|X) = \frac{\pi(\theta, X)}{\pi(X)}$$

Evidence Lower Bound (ELBO)

- Note that we have:

$$KL\{p_\omega(\theta), \pi(\theta|X)\} = -\underbrace{(\mathbb{E}_{p_\omega}\left[log\{\pi(\theta, X)\}\right] - \mathbb{E}_{p_\omega}\left[log\{p_\omega(\theta)\}\right])}_{\text{ELBO}} + \overbrace{log\{\pi(X)\}}^{\text{constant}}$$

- Equivalent, easier optimization problem:

$$\omega* = argmax_{\omega\in\Omega} \int p_\omega(\theta) log\left\{\frac{\pi(\theta, X)}{p_\omega(\theta)}\right\} d\theta$$

  - Cannot confirm KL divergence is small due to the $log\{\pi(X)\}$

- Mean-field variational family is common assumption: independent components

$$p(\theta) = \Pi_{j=1}^{k} p_j(\theta_j)$$

Distribution Family Matters

Source

Summary

Method	Strengths	Limitations
MCMC	Efficient and gold-standard of sampling	Slow convergence; auto-correlated samples
Quasi-MCMC	Reduce estimate variance due to uniform sampling	Potential convergence issues
MALA	Faster convergence in multimodal distributions	Requires gradient computation; critical step-size tuning
Particle Filtering	Effective in non-linear/time-dependent problems	Computationally expensive; suffers from particle degeneracy
Variational Bayes	Fast and scalable to high dimensions	May miss true posterior complexity, family choice important

Python Implementation

Python Implementation of MCMC

- *PyMC* package is used for Bayesian modeling
  - See Patil et al. (2010) for a nice overview of the first version
- Website provides examples of Metropolis-Hastings, variational inference, sequential Monte Carlo, etc.
- *ArviZ* is a library used for Bayesian model visualization (together with PyMC)

PyMC Example

Example from Patil et al. (2010):

- Suppose we have a binomial logit model where the number of successes Y is based on covariate X:

$$\theta = \frac{exp(\alpha + \beta X)}{1 + exp(\alpha + \beta X)}$$

$$Y \sim Binomial(n, \theta)$$

- We define priors on (α, β):

$$\alpha, \beta \sim \mathcal{N}(0, 10)$$

- Given data points (X, Y) and number of trials n, what are the posteriors of α and β?

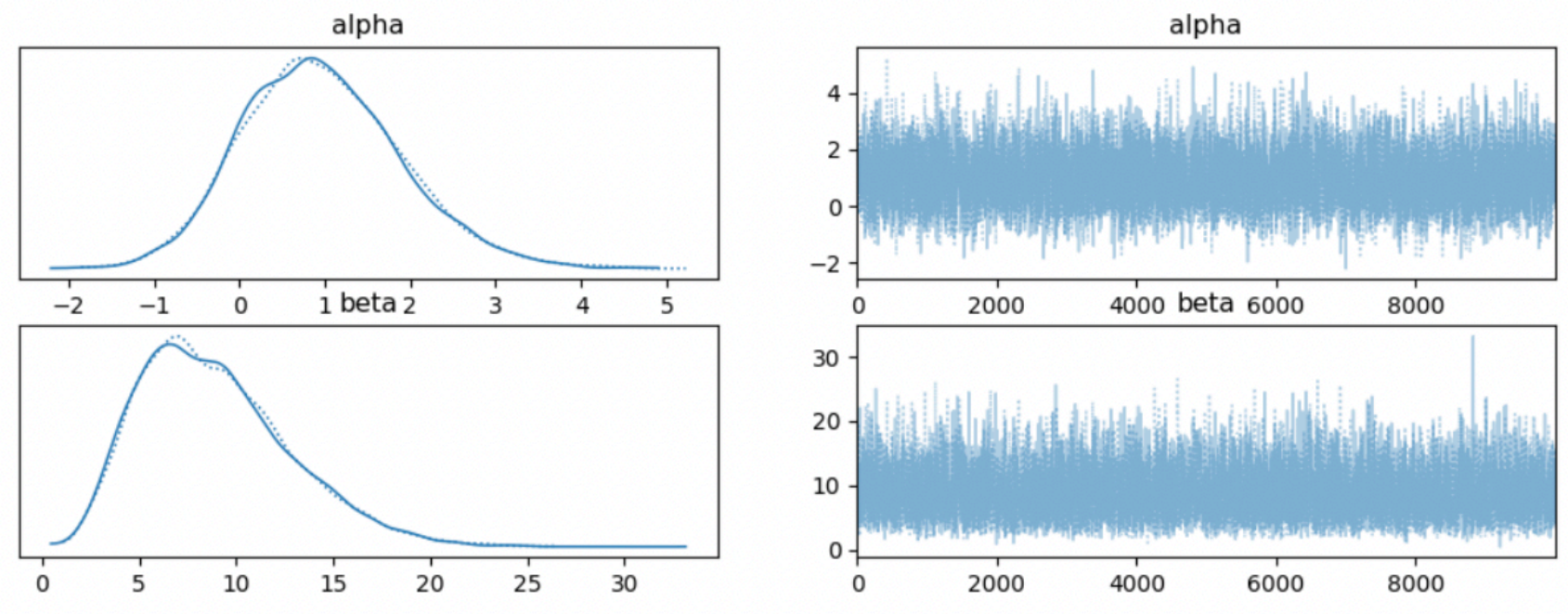
PyMC Example Code

```
import pymc as pm
import numpy as np
import arviz as az

# Data
n = 5 * np.ones(4, dtype=int)
x = np.array([-0.86, -0.3, -0.05, 0.73])
y = np.array([0, 1, 3, 5])

# Model setup
with pm.Model() as model:
    # Priors for unknown model parameters
    alpha = pm.Normal('alpha', mu=0, sigma=10)
    beta = pm.Normal('beta', mu=0, sigma=10)

    # Expected value (logit link function)
    theta = pm.math.invlogit(alpha + beta * x)

    # Likelihood (sampling distribution) of observations
    d = pm.Binomial('d', n=n, p=theta, observed=y)

    # Posterior distribution and sampling
    trace = pm.sample(10000, tune=5000, return_inferencedata=True)
```

```
100.00% [15000/15000 00:15<00:00 Sampling chain 0, 0 divergences]
100.00% [15000/15000 00:14<00:00 Sampling chain 1, 0 divergences]
```

Bayesian Visualization with ArviZ

```
# Plotting the results using ArviZ
az.plot_trace(trace)
```



Resampling Methods

Why Resampling?

- **Goal:** perform inference on estimators (e.g. standard errors, confidence intervals) or perform hypothesis tests
  - What if the data sample is small?
  - What if the data is skewed and non-normal?
  - What if we just don't want to impose an analytical distribution on errors!
- **Solution:** numerical resampling methods $\rightarrow$ no distributional assumptions vs. computational power
  - Reweighting of the data
  - Parallelization

Jackknife & Bootstrap

- Quenouille (1956) and Tukey (1958) introduced the jackknife (i.e. "leave-one-out" estimator)
  - Efron (1979) introduced bootstrap in relation to the jackknife
- Assume data $(X_1, ..., X_N)$ is sampled iid from population
- Each resample is a weight vector over data points:

$$W^{(i)} = (W_1, ..., W_N) \text{ for resamples } i = \{1, ..., B\}$$

  - Jackknife: $W^{(i)} = (1, 1, ..., 0, ..., 1)$ where $W_i^{(i)} = 0$ for $i = \{1, ..., N\}$
  - Bootstrap: $W^{(i)} \sim \text{Multinomial}(N \text{ trials}, N \text{ events})$ for $i = \{1, ..., B\}$
  - Bayesian Bootstrap:[4] $W^{(i)} \sim \text{Dirichlet} \times N$ for $i = \{1, ..., B\}$

[4] Introduced by Rubin (1981).

Bayesian Bootstrap Endorsed by Peter Hull

Peter Hull
@instrumenthull

Ok, so I come bearing good news for ~93% of you: esp. those bootstraping complex models (e.g. w/many FEs)

Instead of resampling, which can be seen as reweighting by a random integer W that may be zero, you can reweight by a random non-zero non-integer W

> **Peter Hull** @instrumenthull · Jan 28, 2022
> Question for economists who do empirical work:
>
> When you need to bootstrap your SEs, do you:
> Show this poll

11:54 AM · Jan 29, 2022

Source

Reweighting the Data

- Resampling is Monte Carlo draw of data weights
- Suppose we have sample mean $\hat{\theta} = f(X_1, ..., X_N) = \frac{1}{N} \sum_{j=1}^{N} X_j$

$$\hat{\theta}^{(i)} = \frac{1}{\sum_{j=1}^{N} W_j^{(i)}} \sum_{j=1}^{N} W_j^{(i)} X_j \text{ for all } i \in \{1, ..., B\}$$

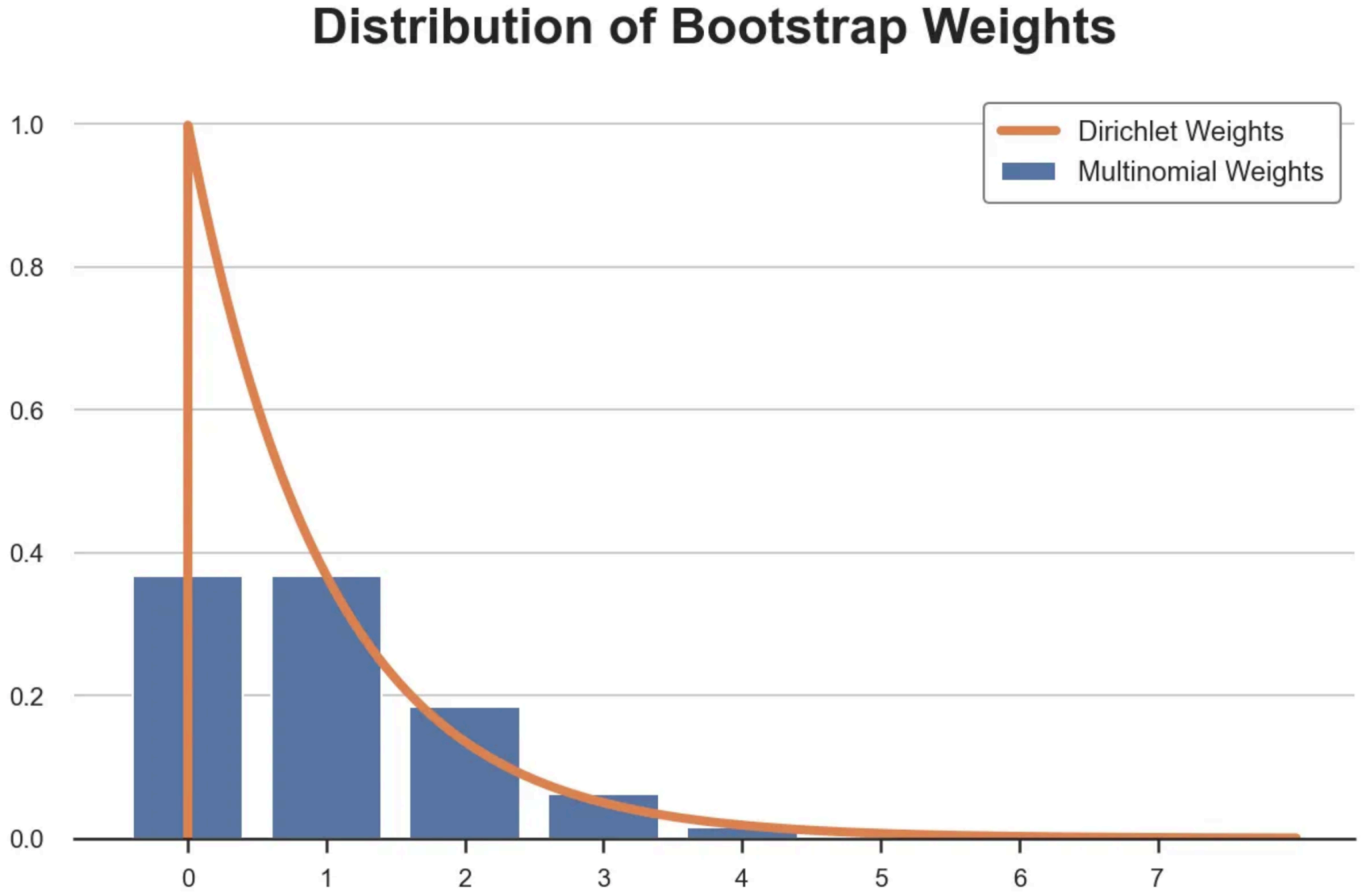
$$\text{Resampling Distribution of } \hat{\theta} : \{\hat{\theta}^{(1)}, ..., \hat{\theta}^{(B)}\}$$

- Dirichlet distribution is parameterized by vector of concentration parameters: $\alpha = (\alpha_1, ..., \alpha_N)$
  - Uninformed prior places uniform weight on each data point

$$\alpha = (1, ..., 1)$$

  - All data points have *non-zero* weight which prevents corner cases
- Choosing weights and computing is easily vectorized and much faster than data sampling

Continuous vs. Discrete Weights



Source

Block Bootstrap & Parallelization

- Bootstrapping is based on data samples being independent
  - Each resample can be computed **in parallel**
  - Greatly increases the efficiency of resampling inference
- Same principle applies to bootstrapping on dependent data:
  - If $X = (X_1, ..., X_N)$ is time-dependent, choose block size M and split the data into

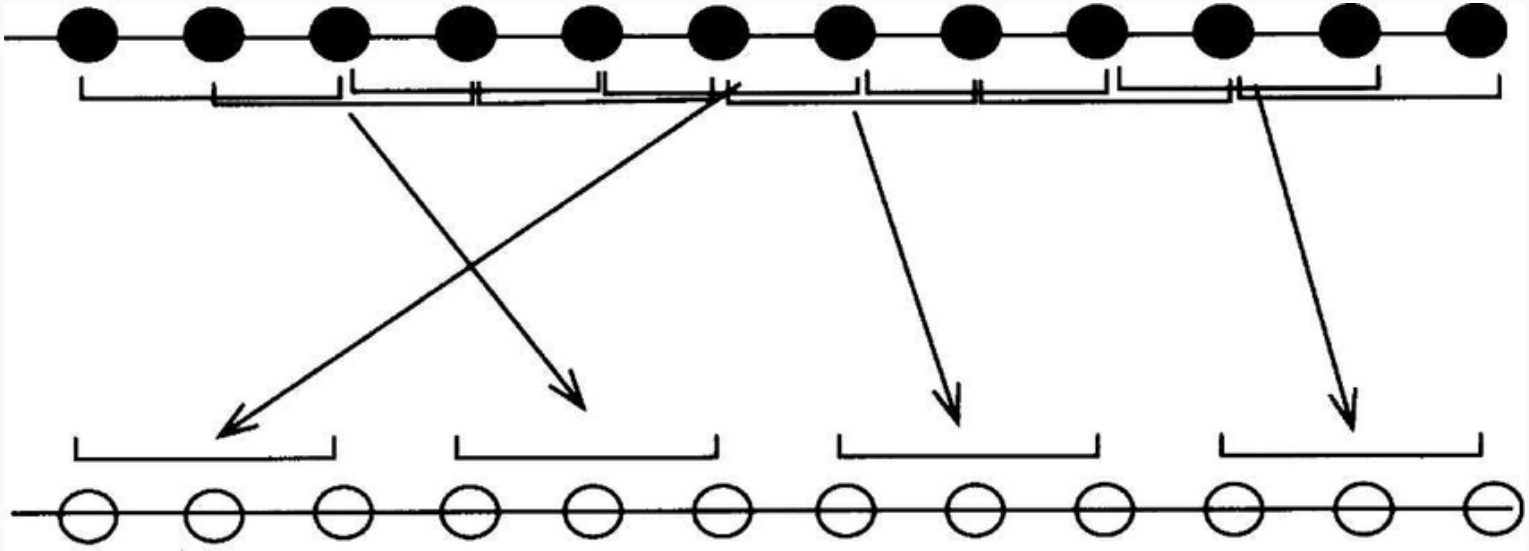
$$\mathcal{B} = \{(X_1, ..., X_M), (X_2, ..., X_{M+1}), ..., (X_{N-M}, ..., X_N)\}$$

  - Resample blocks of data and arrange them to construct a bootstrap sample
  - Ensure data points further than M have negligible dependence

Moving Block Bootstrap

Block 1	R_1	R_2	R_3	R_4	R_5	R_6	...	R_{n-2}	R_{n-1}	R_n
Block 2	R_1	R_2	R_3	R_4	R_5	R_6	...	R_{n-2}	R_{n-1}	R_n
Block 3	R_1	R_2	R_3	R_4	R_5	R_6	...	R_{n-2}	R_{n-1}	R_n
...					...					
Block n-L+1	R_1	R_2	R_3	R_4	R_5	R_6	...	R_{n-2}	R_{n-1}	R_n

Source



Source

Permutation Tests

- Used for *hypothesis testing* of outcomes in two or more groups
- Under null hypothesis H_0, data is *exchangeable* (i.e. treatment has no effect)
- Permute data labels and calculate estimators $\rightarrow$ from null distribution
  - Large datasets $\rightarrow$ too many permutations $\rightarrow$ Monte Carlo sample of possible permutations
- Permuting and test statistic calculation are *parallelizable*
- `numpy.random.shuffle()` shuffles array in place