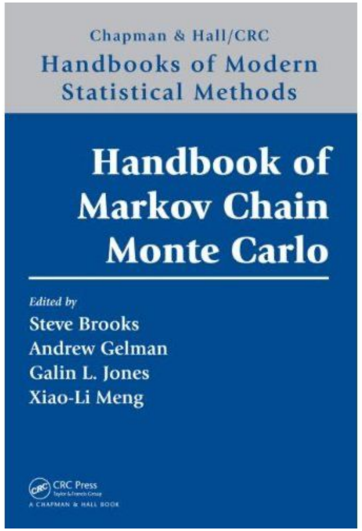
GitHub

An introduction to MCMC

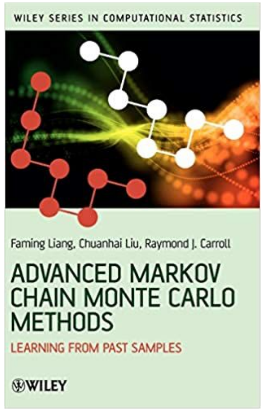
Presented by @jhelie - May 20th 2021


Chapman & Hall/CRC
Handbooks of Modern Statistical Methods

Handbook of Markov Chain Monte Carlo

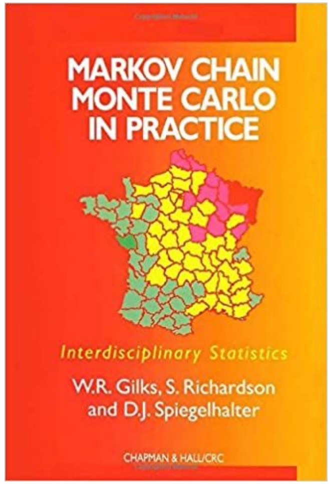
Edited by
Steve Brooks
Andrew Gelman
Galin L. Jones
Xiao-Li Meng

CRC Press
A CHAPMAN & HALL BOOK


WILEY SERIES IN COMPUTATIONAL STATISTICS

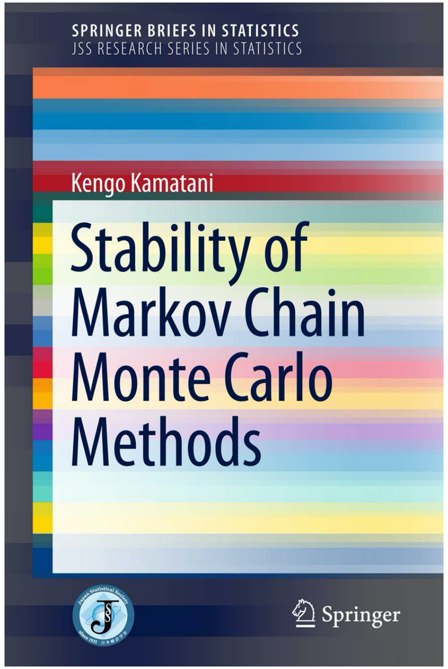
Faming Liang, Chuanhai Liu, Raymond J. Carroll
ADVANCED MARKOV CHAIN MONTE CARLO METHODS
LEARNING FROM PAST SAMPLES

WILEY


MARKOV CHAIN MONTE CARLO IN PRACTICE

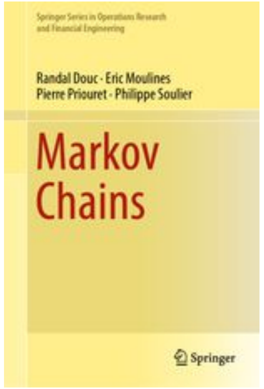
Interdisciplinary Statistics
W.R. Gilks, S. Richardson and D.J. Spiegelhalter

CHAPMAN & HALL/CRC


SPRINGER BRIEFS IN STATISTICS
JSS RESEARCH SERIES IN STATISTICS

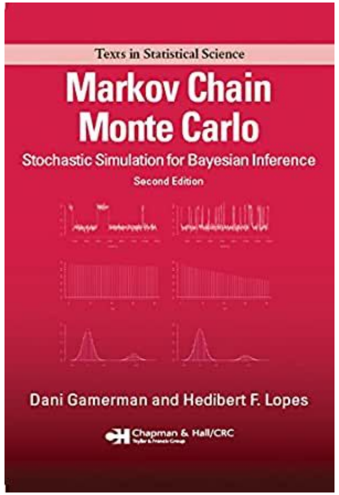
Kengo Kamatani
Stability of Markov Chain Monte Carlo Methods

Springer


Springer Series in Operations Research and Financial Engineering

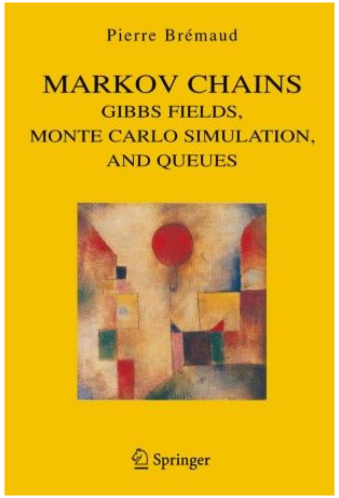
Randal Douc · Eric Moulines
Pierre Priouret · Philippe Soulier
Markov Chains

Springer


Texts in Statistical Science
Markov Chain Monte Carlo
Stochastic Simulation for Bayesian Inference
Second Edition

Dani Gamerman and Hedibert F. Lopes

Chapman & Hall/CRC


Pierre Brémaud
MARKOV CHAINS
GIBBS FIELDS,
MONTE CARLO SIMULATION,
AND QUEUES

Springer

What’s in a name?

MCMC = Markov Chain Monte Carlo

straight from Wikipedia:

A **Markov chain** is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event.

Monte Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results.

So MCMC is...?

Techniques to draw samples from a distribution given its pdf.

That’s it.

Nothing more.

Note: any function can be considered a “distribution” on an interval where it’s positive and integrable.

Why is that interesting?

Use Cases

When was the multivariate Gaussian distribution added to scipy?

2014

I just want to draw samples

- libraries allow us to easily draw samples from common distributions
- but what do you do if you want to draw from an arbitrary distribution?

I have no idea what my distribution *looks* like

- you have the pdf… but do you know *where* you should evaluate it?
- particularly problematic with multidimensional pdfs

I want to compute a gnarly looking integral

- integrals are just averages, i.e. sums…
- that's easy to compute if you can find out over which set to evaluate

MCMC algorithm(s)

There are *many* MCMC algorithms, but the basic idea remains the same and corresponds to the **Metropolis algorithm**.

Say we want to draw samples from distribution $p(x)$.

```
1.  Initialise your chain to an arbitrary value: [$x_1$]
2.  Randomly choose $x^*$ using a proposal distribution
    q centered on $x_1$
3.  Compute ratio $p(x^*)/p(x_1)$
4.  Draw a random u number from Uniform([0,1])
    4.1.  append $x_2=x^*$ if $p(x^*)/p(x_0) > u$: [$x_1$,$x_2$]
    4.2.  append $x_2=x_1$ if $p(x^*)/p(x_0) < u$: [$x_1$,$x_1$]
5.  Repeat from step 2
```

Say x^* is 5 times less likely than x_1:
$p(x^*)/p(x_1)=0.2$

Then there's a 20% chance to keep it.

By following this rule every time we'll end up with 5 times fewer x^* values than x_1 values in our chain.

When running an MCMC algorithm we always accept more likely proposals, but can also accept less likely proposals (so we don't get stuck in local maxima).

This allows us to build a chain whose stationary distribution is the same as that of our target distribution $p(x)$ since $n_{xi}/n_{xj} \to p(x_i)/p(x_j)$.

Metropolis

- the proposal density q must be symmetrical
- acceptance probability:

$$a1 = \min\left(\frac{p(\mathbf{x}_*)}{p(\mathbf{x}_{t-1})}, \quad 1\right)$$

Metropolis-Hasting

- no symmetry constraint on the proposal density q
- acceptance probability is modified to take asymmetry into account:

$$a = \min\left(\frac{p(\mathbf{x}_*)q(\mathbf{x}_{t-1}|\mathbf{x}_*)}{p(\mathbf{x}_{t-1})q(\mathbf{x}_*|\mathbf{x}_{t-1})}, \quad 1\right)$$

The Metropolis algorithm is a special case of the Metropolis-Hasting algorithm.

Concepts: creating the chain

- **Trace**
  The sequence of accepted values in the MCMC chain.
- **Proposal density**
  The distribution used to randomly select a new candidate value given the current value.
- **Proposal width**
  The standard deviation of the proposal density
- **Mixing time**
  The speed with which the stationary distribution is reached and sampled.

Concepts: checking convergence

Until the chain has reached its stationery distribution it is not useful.

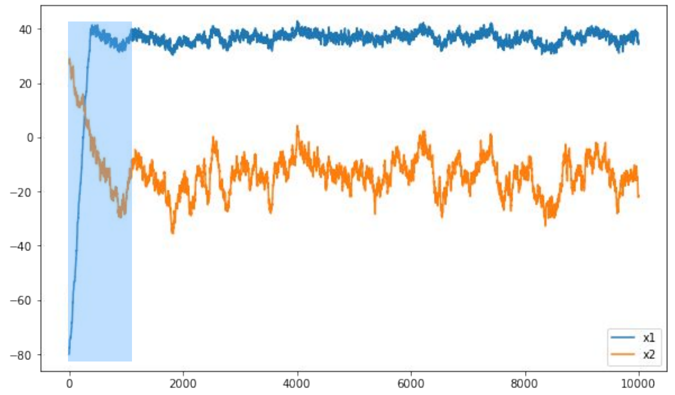
Formal proof of convergence is usually not possible but several heuristics exist to assess whether we’re close enough to the target distribution:

- split chain into chunks and check their statistical properties
- check statistical properties of different chains
- check autocorrelation plots
- compare inter and intra chain variance (Gelman-Rubin diagnostic)

Concepts: using the sampler

Burn-in

The beginning of the trace before it reaches its stationary distribution.



Thinning

By construction consecutive elements of the trace are correlated.

So to generate i.i.d. samples we can take only every `n-th` value of the chain where `n` is such that the retained elements are independent.

Issues

The challenge is to set up our MCMC sampler so that we can achieve rapid mixing.

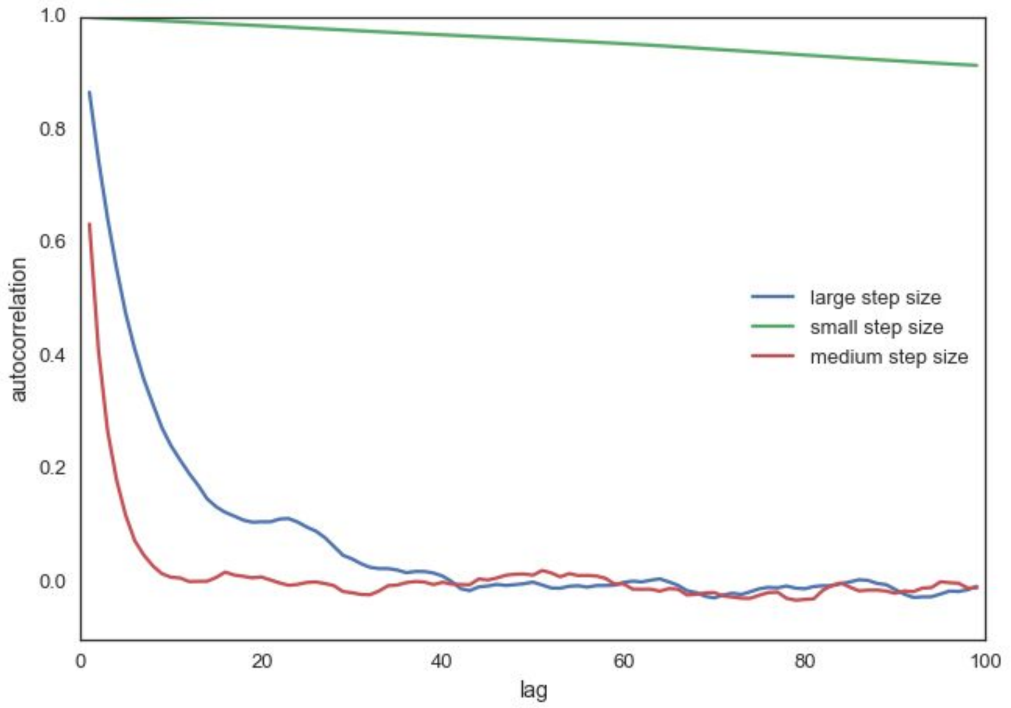
Two major issues, compounding each other, are:

- finding a suitable proposal dist and width
- efficient sampling in multi-dimensions

Some sophisticated MCMC algorithms try to address these issues but there's no silver bullet and defining the best MCMC sampler can be a bit of a dark art.

Proposal width

- determines the acceptance rate
- a too high and a too low acceptance rate both impact the sampling efficiency
- can be tuned to achieve an optimum average acceptance rate (e.g. 0.234 for Gaussian proposal density under certain assumptions)



Multivariate distributions

- the standard Metropolis algorithm updates the entire parameter vector at once with a single step: a p-dimensional parameter vector must have a p-dimensional proposal distribution.

- tuning the MCMC algorithm becomes harder as the number of dimensions increases.

- Single Component MCMC algorithms can help address these issues.
    - a univariate proposal dist is used to update each dimension sequentially

- **Gibbs sampling** is one such algorithm and very commonly used when doing Bayesian statistics
    - the conditional dist is used to update each dimension sequentially
    - more efficient than naive SC MCMC
    - requires ability to draw from conditionals…
    - ...but if need we can use MCMC to do that: “hybrids Gibbs” or “Metropolis within Gibbs”

Bayes and MCMC

Bayesian distributions recap

- We have a **model** that takes a vector of parameters θ and that gives us the probabilities of observing the data X:
  - `p(X|θ)`
- We also have a **prior** for θ, i.e. some intuition of how likely a given vector θ is:
  - `p(θ)`
- We're usually interested in determining the **posterior** of θ i.e., having seen the data, an updated belief about how likely a given vector θ is:
  - `p(θ|X) = p(X|θ)*p(θ)/p(X) ~ p(X|θ)*p(θ)` since `p(X)=cste`
- The **posterior predictive** can also be of interest to predict new values, having seen the evidence:
  - $p(x_{new}|X) = \int p(x_{new}|\theta) * p(\theta|X)\, d\theta$

Estimating the posterior distribution

- In general the posterior distribution is likely to be hard to compute:
  - it can be arbitrary
  - it can be high dimensions (the simplest of models usually have at least 2 parameters)

- We can go down the conjugate prior route
  - prior and model such that the posterior is in the same distribution family as the prior
  - Examples: Gaussian/Gaussian, Binomial/Beta, etc...

- Or we can use MCMC

Note: *given the choice we should always try to choose a model and a prior that make the posterior as manageable as possible - but that might still require using a sampling technique like MCMC to evaluate it*

Some clarification

A common justification for the need for MCMC is the difficulty of computing `p(X)`.

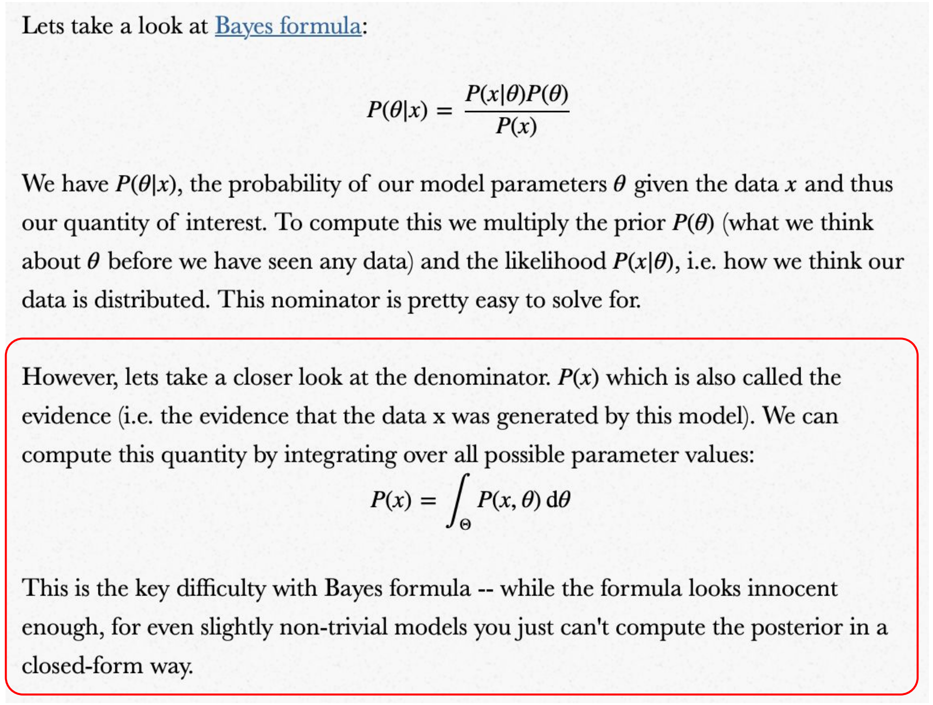
It is indeed difficult but… it's a constant and we couldn't care less about it.

```
p(θ|X) ~ p(X|θ)*p(θ)
```

If we couldn't compute the above for a given θ, there's not much we could do.

The real issue regarding evaluating the posterior is that we don't know ***where*** to do so.

From PyMC3 author's blog post MCMC sampling for dummies:



Lets take a look at Bayes formula:

$$P(\theta|x) = \frac{P(x|\theta)P(\theta)}{P(x)}$$

We have $P(\theta|x)$, the probability of our model parameters θ given the data x and thus our quantity of interest. To compute this we multiply the prior $P(\theta)$ (what we think about θ before we have seen any data) and the likelihood $P(x|\theta)$, i.e. how we think our data is distributed. This nominator is pretty easy to solve for.

However, lets take a closer look at the denominator. $P(x)$ which is also called the evidence (i.e. the evidence that the data x was generated by this model). We can compute this quantity by integrating over all possible parameter values:

$$P(x) = \int_{\Theta} P(x, \theta)\, d\theta$$

This is the key difficulty with Bayes formula -- while the formula looks innocent enough, for even slightly non-trivial models you just can't compute the posterior in a closed-form way.

misleading/irrelevant

Estimating the **posterior** using MCMC

- we simply apply MCMC to: `f(θ) = p(X|θ)*p(θ)`
- when theta is multivariate we'll use Gibbs Sampling

Estimating the posterior predictive using MCMC

- We want to estimate the distribution:

$$p(x_{new}|\text{data}) = \int p(x_{new}|\theta) * p(\theta|\text{data})\, d\theta$$

- This is fairly easy to do by repetitively sampling via the following steps:
  - draw θi from the posterior $p(\theta|\text{data})$ using MCMC
  - draw xi from $p(x|\theta i)$ using the dist modeling the process

The obtained $\{xi\}$ are distributed according to $p(x|\text{data})$. That's because the above amounts to sampling from $p(x,\theta|\text{data})$ and marginalising over θ.