

Statistical inference with Markov chain Monte Carlo algorithms

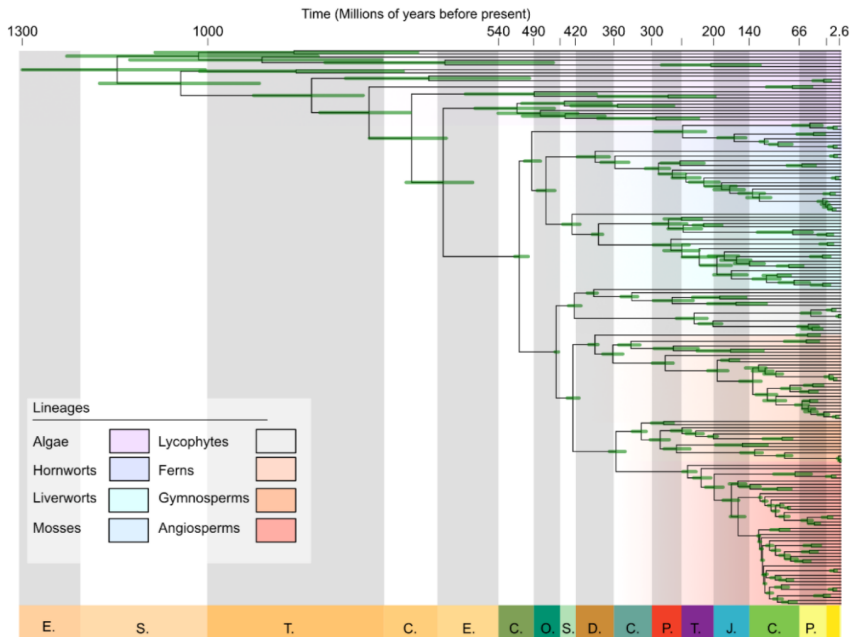
MuniHac 2024

Dominik Schrempf

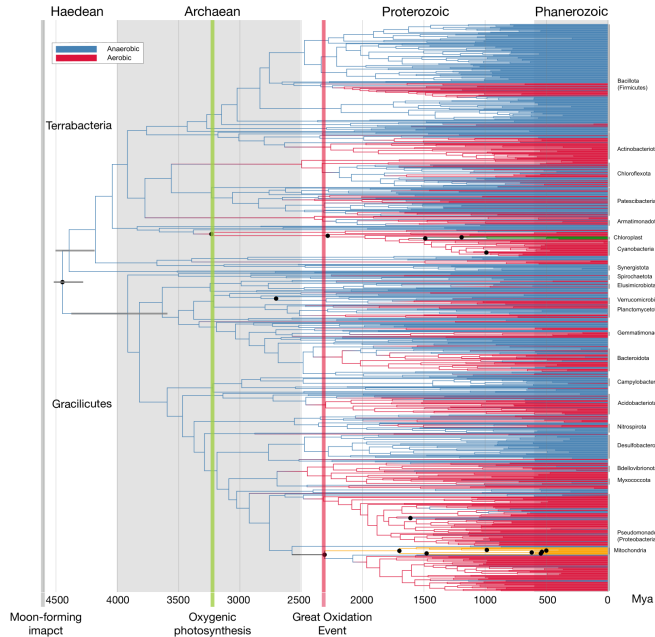
TNG Technology Consulting

October 12, 2024

Evolution of land plants (Harris et al., 2021)



Dated phylogeny of bacteria (Davin et al., 2023)



Markov chains are the simplest mathematical models for random phenomena evolving in time.

At the same time, the class of Markov chains is rich enough to serve in many applications.

This makes Markov chains the first and most important examples of random processes.

J. R. Norris (1998). *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics

A discrete Markov chain is a sequence $(X_n)_{n \geq 0}$ of possible states in which the probability of each state only depends on the previous state

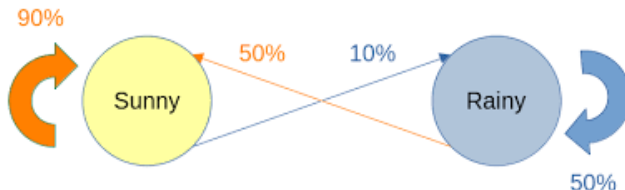
$$\Pr(X_{n+1} = i_{n+1} \mid X_0 = i_1, \dots, X_n = i_n) = \Pr(X_{n+1} = i_{n+1} \mid X_n = i_n), \quad (1)$$

where $n \geq 0$ is a natural number, $i_0, \dots, i_{n+1} \in I$ are states, and I is the **state space** (i.e., the set of all possible states).

The probabilities

$$\Pr(X_{n+1} = j \mid X_n = i) = p_{ij} \quad (2)$$

are called **transition probabilities**.



- The state space I is {Sunny, Cloudy}.
- For example, the probability to have sun today when there was rain yesterday is $\Pr(X_{\text{today}} = \text{Sunny} \mid X_{\text{yesterday}} = \text{Rainy}) = 50\%$.
- A possible, albeit improbable, realization of the Weather Markov chain is: Rainy, Rainy, Rainy, Rainy, Rainy.

Figure from lancaster.ac.uk.

The obvious way to find out the thermodynamic equilibrium is to simulate the dynamics of the system, and let it run until it reaches equilibrium.

The practice of MCMC is simple. Set up a Markov chain having the required invariant distribution, and run it on a computer. The folklore of simulation makes this seem more complicated than it really is.

S. Brooks et al., eds. (2011). *Handbook of Markov Chain Monte Carlo*. CRC press

- We would like to simulate *processes*.
- We can do that by
 - ① constructing a probabilistic model that describes the process we seek to simulate, and by
 - ② sampling realizations of the process given our probabilistic model.

That is, **we want to sample from probability distributions**.

But isn't this supposed to be easy?

Well, it depends. In fact, it is usually exceedingly difficult.

Markov chain Monte Carlo algorithms construct and sample from specific Markov chains that **converge to the probability distributions we desire** to have samples from.

For a given state x of our process with probability density $h(x)$, we are interested in *proposing* a new state such that we preserve the probability density of states $h(\cdot)$.

The Metropolis-Hastings algorithm does the following:

- Propose to move from x to y with probability density $q(x, y)$.
- Calculate the Hastings ratio

$$r(x, y) = \frac{h(y)q(y, x)}{h(x)q(x, y)}. \quad (3)$$

- Accept the proposal with probability

$$a(x, y) = \min(1, r(x, y)). \quad (4)$$

Recall the Metropolis-Hastings algorithm

$$r(x, y) = \frac{h(y)q(y, x)}{h(x)q(x, y)}, \text{ and } a(x, y) = \min(1, r(x, y)). \quad (5)$$

We want to simulate tosses of a coin. The coin has a probability p to show head H , and $1 - p$ to show tail T . Hence, $I = \{H, T\}$, and $h(H) = p$, and $h(T) = 1 - p$.

We use the following proposal: If the current state is $x = H$, we propose $y = T$; if the current state is $x = T$, we propose $y = H$. That is, $q(x, y) = 1$ if $x \neq y$, otherwise 0.

- If $p = 0.5$, the acceptance probability is 1.0, and $\Pr(x = H) = \Pr(x = T) = 0.5$.
- If $0 < p < 0.5$, the acceptance probability of moving from H to T is 1, and of moving from T to H is $p/(1 - p)$. On average, we remain $(1 - p)/p$ iterations at T until a proposal to H gets accepted. That is,

$$\Pr(x = H) = \frac{1}{1 + (1 - p)/p} = \frac{1}{\frac{p+1-p}{p}} = p. \quad (6)$$

Usually, we split up the probability density into a product of

- a *prior* function describing the probability density of model parameters, and
- a *likelihood* function describing the probability density of observing some data given a particular model configuration.

Then, we call the probability function *posterior* function.

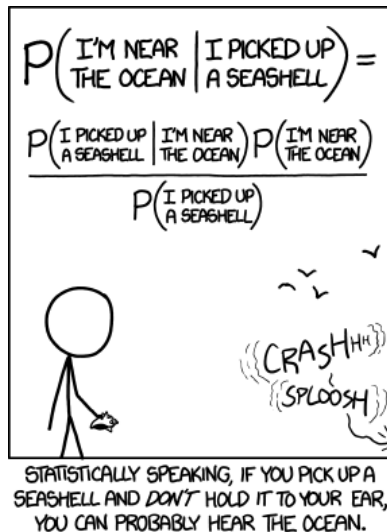


Figure from <https://xkcd.com/1236/>.

There are only two alternatives available on Hackage:

- An amazing set of libraries developed by Jared Tobin: `mcmc-flat`, `mighty-metropolis`, `speedy-slice`, and `hasty-hamiltonian`. They are more lightweight, but also less flexible.
- `mcmc` developed by myself (← we are going to work with this one).
- (There are probabilistic programming libraries such as `monad-bayes` hiding the MCMC algorithm behind scenes; in my experience, they are not yet flexible enough for real world problems).

- Samples in state space (polymorphic type `a` , usually called `I` ; user-defined).
- Prior and likelihood functions

```
1 type PriorFunction a = a -> Log Double
2 type LikelihoodFunction a = a -> Log Double
```

- List of proposals performed per iteration

```
1 type Proposal a = a -> Gen -> (a, Log Double, Gen) -- Simplified.
2 type Cycle a = [Proposal a] --Simplified.
```

- Monitors (not strictly necessary, but very convenient).

Note, probabilities are handled in log domain (log-domain package)

```
1 newtype Log a = Exp { ln :: a }
```

- A `Chain` contains current sample (`link`), trace, prior function, likelihood function, cycle, and monitors.
- The Metropolis-Hastings-Green algorithm `MHG` is a small wrapper around a `Chain` and specifies algorithmic details. Another algorithm is the Metropolis-coupled Markov chain Monte Carlo algorithm `MC3`.
- The function `mcmc` runs an MCMC algorithm (terminal output, burn-in with auto tuning, saves chain on early exit, etc.).

Clone the workshop repository¹. Have a look at the slides and the exercise sheet.

If you have the Nix package manager:

- Either use `nix develop` to enter a shell having all dependencies, or activate `nix-direnv` using `direnv allow`.
- Build the project using `cabal build`.

Otherwise:

- Set up a Haskell toolchain using `GHCup`². I am using `GHC 9.6`.
- Try building the project using `cabal build`.
- Install `tracer`³, if you want to inspect traces.

Let me know if you have problems!

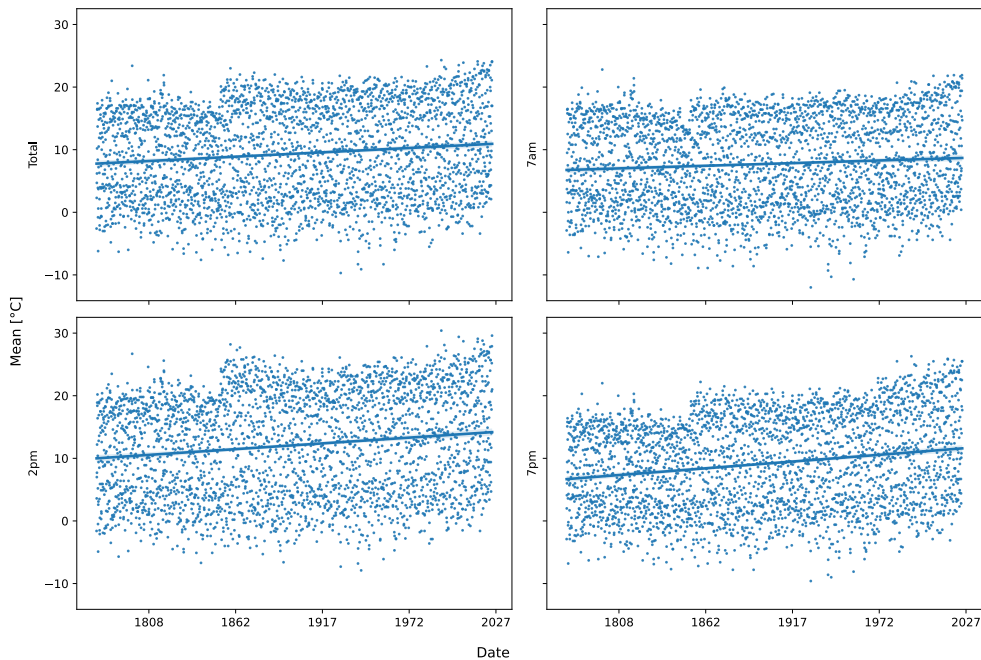
¹<https://github.com/dschrempf/munihac24-mcmc>

²<https://www.haskell.org/ghcup/>

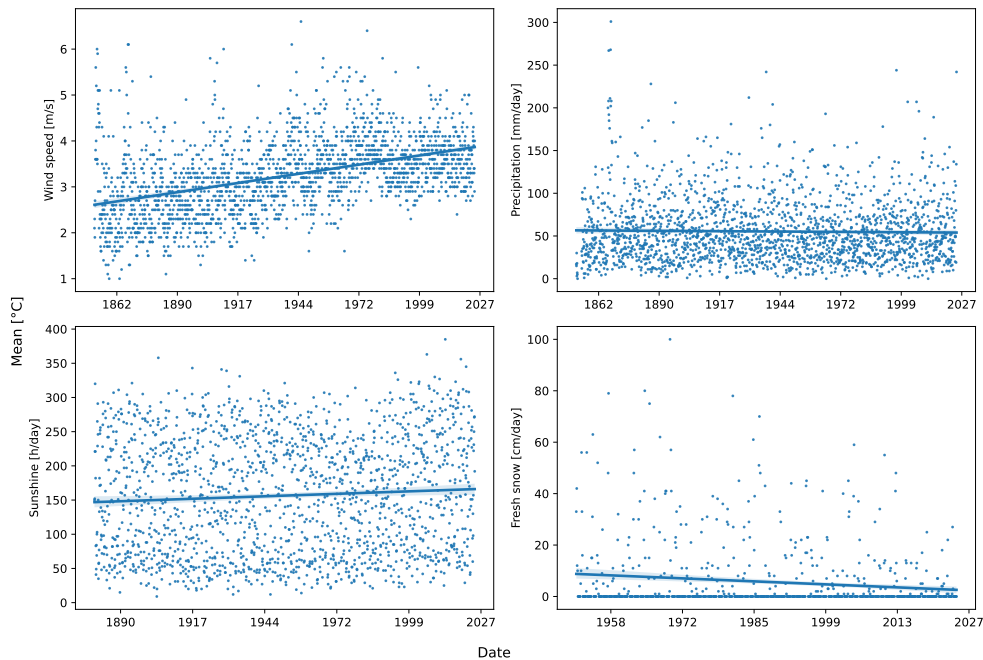
³<https://github.com/beast-dev/tracer>

- Data from [Geosphere Austria](#) "Messstationen Monatsdaten v2".
- Air temperatures at Hohe Warte (Vienna).
- Monthly means (total, 7am, 2pm, 7pm) from 1775 to today.

Air temperature at Hohe Warte (Vienna)



Other climate data at Hohe Warte (Vienna)







There are more advanced algorithms available. For example, Hamiltonian Monte Carlo (HMC) uses the gradient of the likelihood function to propose new values.

The `mcmc` library offers two implementations of proposals based on Hamiltonian dynamics: classical HMC, and the No-U-Turn sampler (NUTS).

We can use automatic differentiation to calculate the gradient! (But there is a lot left to be desired.)

Thank you!

-  Brooks, S., A. Gelman, G. Jones, and X.-L. Meng, eds. (2011). *Handbook of Markov Chain Monte Carlo*. CRC press.
-  Davín, A. A., B. J. Woodcroft, R. M. Soo, B. Morel, R. Murali, D. Schrempf, J. Clark, B. Boussau, E. R. R. Moody, L. L. Szánthó, E. Richy, D. Pisani, J. Hemp, W. Fischer, P. C. Donoghue, A. Spang, P. Hugenholtz, T. A. Williams, and G. J. Szöllősi (2023). “An evolutionary timescale for Bacteria calibrated using the Great Oxidation Event.” In: *bioRxiv*. DOI: 10.1101/2023.08.08.552427.
-  Harris, B. J., J. W. Clark, D. Schrempf, G. J. Szöllősi, P. C. Donoghue, A. M. Hetherington, and T. A. Williams (2021). “Divergent evolutionary trajectories of bryophytes and tracheophytes from a complex common ancestor of land plants.” In: *bioRxiv*. DOI: 10.1101/2021.10.28.466308.
-  Norris, J. R. (1998). *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics.