Markov Chain Monte Carlo

Dustin Lang
Perimeter Institute for Theoretical Physics

PSI Numerical Methods 2024

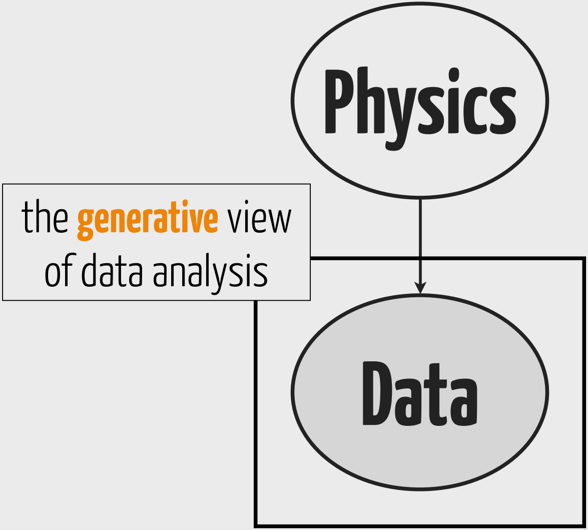
Borrowing heavily from Dan Foreman-Mackey's slides
https://speakerdeck.com/dfm/data-analysis-with-mcmc

These slides are available at
https://github.com/dstndstn/MCMC-talk

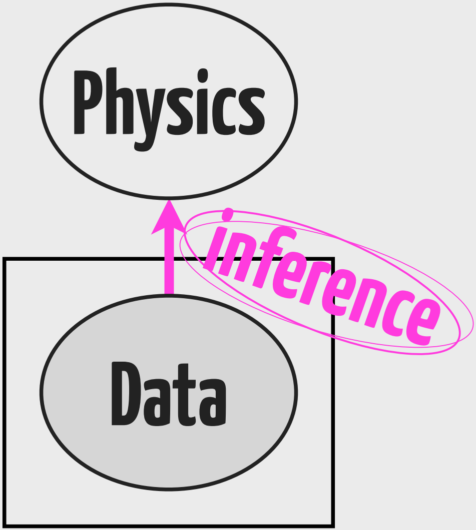
data analysis with

Markov chain Monte Carlo

Dan Foreman-Mackey
CCPP@NYU



The a sketch of graphical model of my research.



a sketch of
The graphical model of my research.

$$p(\mathrm{data} \mid \mathrm{physics})$$

likelihood function/generative model

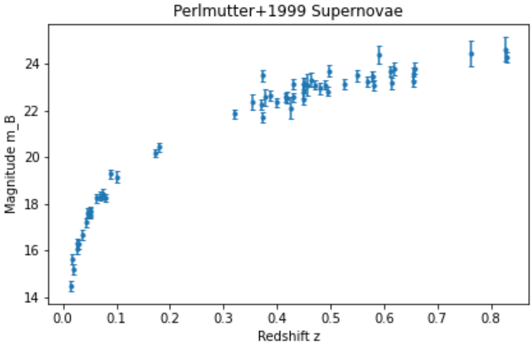
$$\downarrow$$

$$p(\mathrm{physics} \mid \mathrm{data}) \propto p(\mathrm{physics})\, p(\mathrm{data} \mid \mathrm{physics})$$

posterior probability

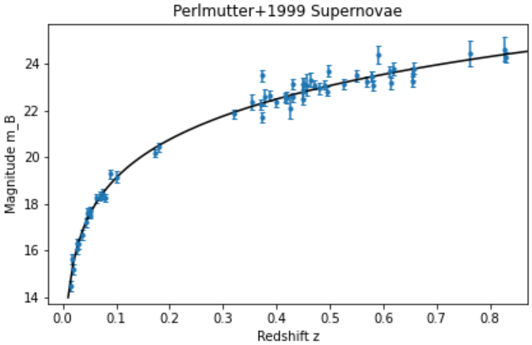
An example

- Perlmutter+1999 (*https://arxiv.org/abs/astro-ph/9812133*)
- Measured the observed peak brightnesses of a sample of type-1a supernovae (in astronomer "mag" units), and the redshifts ("z") of the supernova host galaxies
- $\mathrm{mag} = \mathrm{mag}_{\mathrm{intrinsic}} + \mathrm{luminosity\_distance}(z, \mathrm{parameters})$



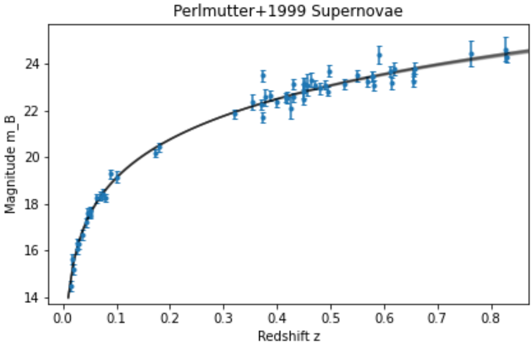
An example

- Generative model:
  $\text{mag}_i = \text{mag}_{\text{intrinsic}} + \text{luminosity\_distance}(z_i, \text{parameters}) + \epsilon_i$
- Probability of a data point given a model ("likelihood"):
  $p(\text{mag}_i \,|\, \text{params}) = \text{Gaussian}(\text{mag}_i \,|\, \mu = f(z_i, \text{params}), \sigma_i^2)$
- Probability of whole data set given a model:
  $p(\{\text{mag}_i\} \,|\, \Omega_M, \Omega_\Lambda) = \prod_i \mathcal{N}(\text{mag}_i \,|\, \text{mag}_{\text{int}} + D_L(z_i, \Omega_M, \Omega_\Lambda), \sigma_i^2)$



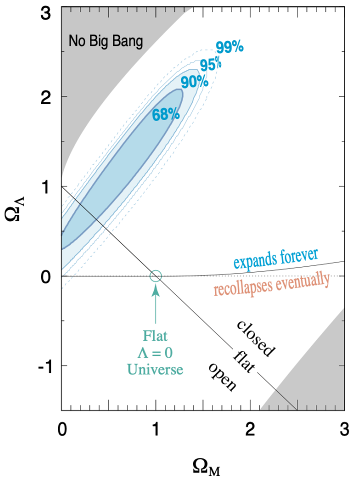
An example

- Use Bayes' theorem to convert data likelihoods into contraints on the model parameters $\theta = \{\Omega_M, \Omega_\Lambda\}$
- $p(\theta \,|\, \mathrm{data}) \propto p(\theta)\, p(\mathrm{data} \,|\, \theta)$
- $p(\Omega_M, \Omega_\Lambda \,|\, \{\mathrm{mag}_i\}) \propto$
  $p(\Omega_M, \Omega_\Lambda) \prod_i \mathcal{N}(\mathrm{mag}_i \,|\, \mathrm{mag}_{\mathrm{int}} + D_L(z_i, \Omega_M, \Omega_\Lambda), \sigma_i^2)$



An example

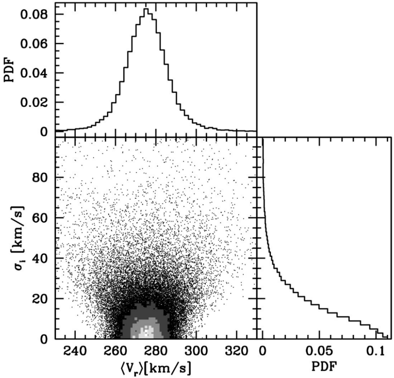
- Then they ran MCMC...
- Resulting parameter constraints (blue ellipse):



Why we often need MCMC

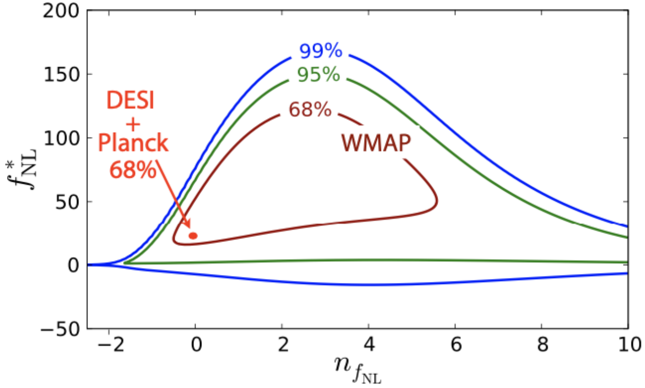
- Real-life models and likelihoods are often complex
- . . . so the resulting constraints have complicated distributions (not Gaussians!)
- . . . but we can represent them with samplings
- MCMC is used for drawing samples from probability distributions that we can compute numerically but cannot solve analytically

Samplings to represent constraints - examples



- From https://arxiv.org/abs/1910.04899
- With a sampling: Marginalize over a parameter by projecting it out

Samplings to represent constraints - examples



- From https://arxiv.org/abs/1611.00036

MCMC

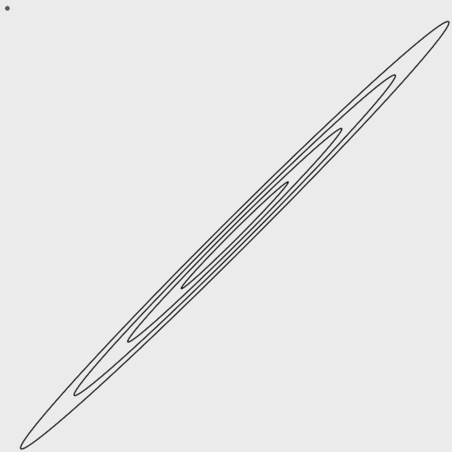
draws samples from a probability function

and all you need to be able to do is

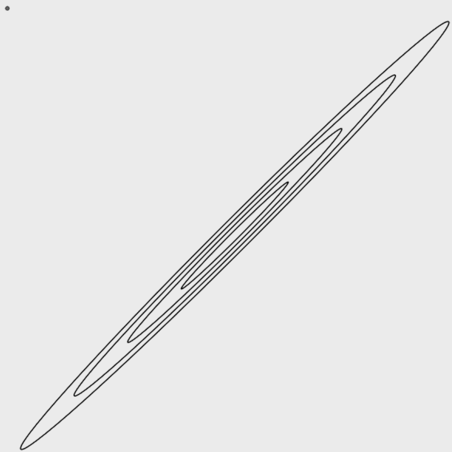
evaluate

the function

(up to a constant)

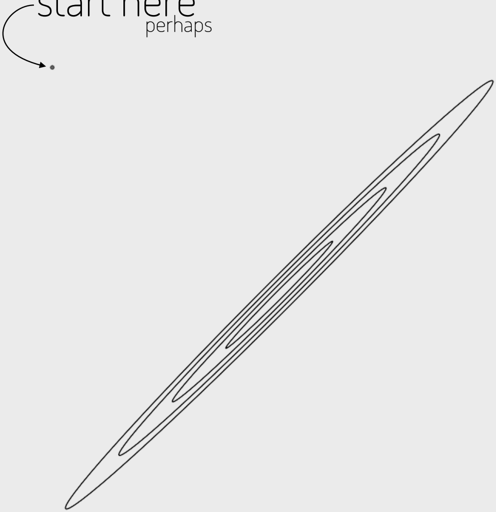


Metropolis–Hastings

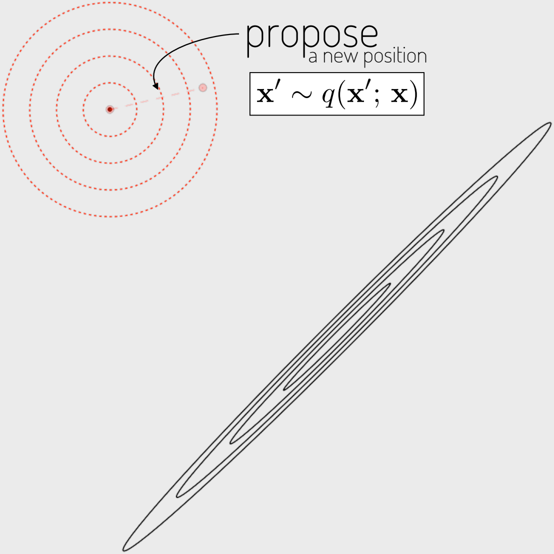


Metropolis–Hastings

in an ideal world



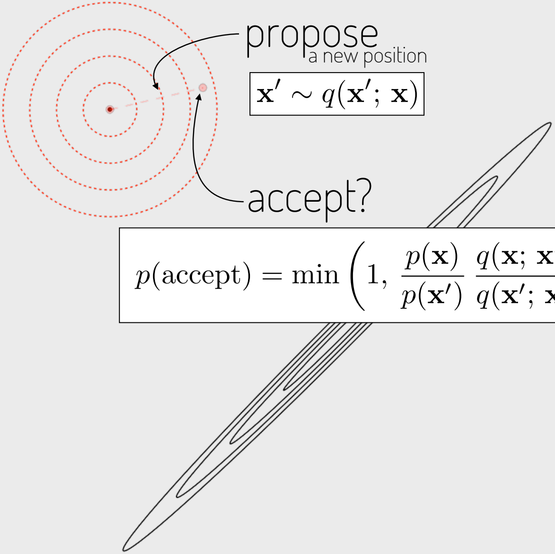
Metropolis–Hastings
in an ideal world



$$\boxed{\mathbf{x}' \sim q(\mathbf{x}'; \mathbf{x})}$$

Metropolis–Hastings

in an ideal world

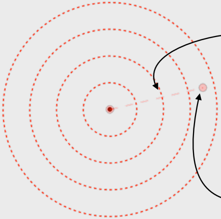


$$\boxed{\mathbf{x}' \sim q(\mathbf{x}';\, \mathbf{x})}$$

$$\boxed{p(\text{accept}) = \min\left(1,\ \frac{p(\mathbf{x})}{p(\mathbf{x}')}\,\frac{q(\mathbf{x};\,\mathbf{x}')}{q(\mathbf{x}';\,\mathbf{x})}\right)}$$

Metropolis–Hastings

in an ideal world

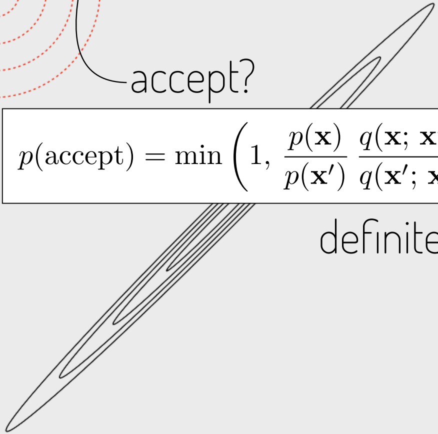


propose
a new position

$$\boxed{\mathbf{x}' \sim q(\mathbf{x}';\, \mathbf{x})}$$

accept?

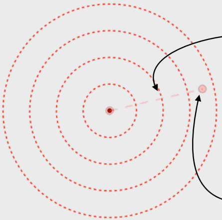
$$\boxed{p(\text{accept}) = \min\left(1,\ \frac{p(\mathbf{x})}{p(\mathbf{x}')}\ \frac{q(\mathbf{x};\, \mathbf{x}')}{q(\mathbf{x}';\, \mathbf{x})}\right)}$$



definitely.

Metropolis–Hastings

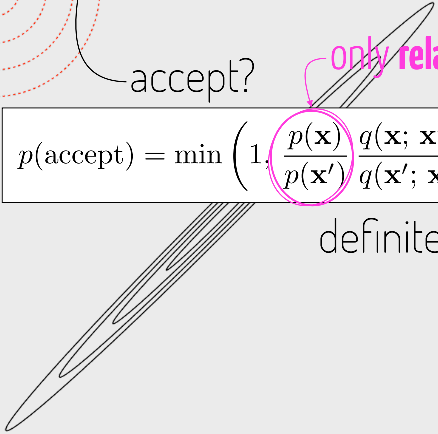
in an ideal world



propose
a new position

$$\boxed{\mathbf{x}' \sim q(\mathbf{x}';\, \mathbf{x})}$$

accept?

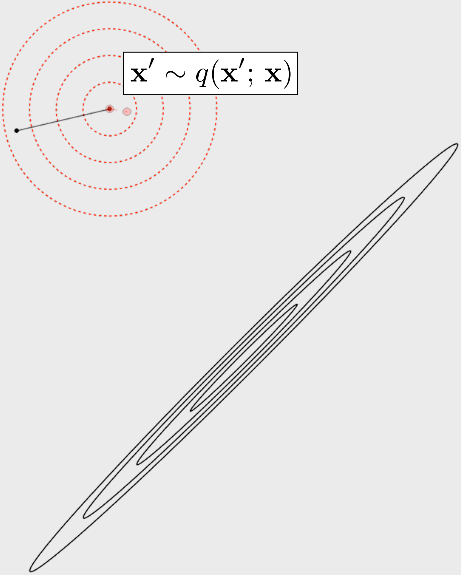


only **relative** probabilities

$$\boxed{p(\text{accept}) = \min\left(1, \frac{p(\mathbf{x})}{p(\mathbf{x}')} \frac{q(\mathbf{x};\, \mathbf{x}')}{q(\mathbf{x}';\, \mathbf{x})}\right)}$$

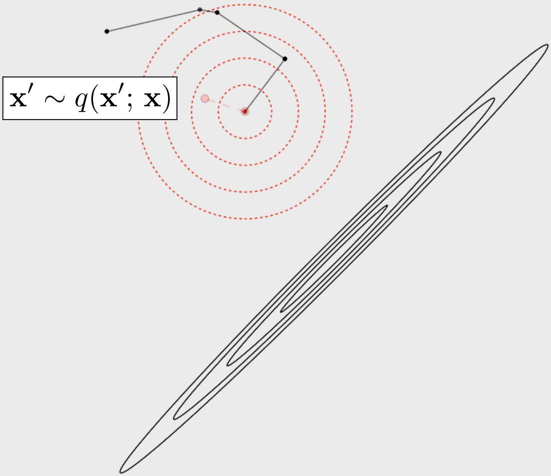
definitely.

Metropolis–Hastings
in an ideal world



Metropolis–Hastings

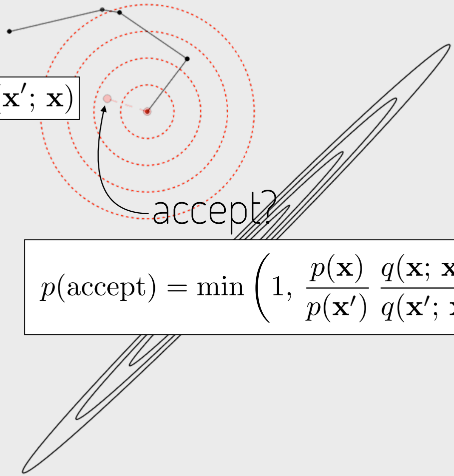
in an ideal world



Metropolis–Hastings

in an ideal world

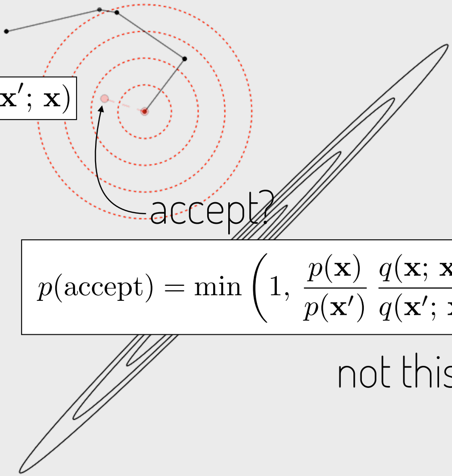
$$\mathbf{x}' \sim q(\mathbf{x}';\, \mathbf{x})$$



$$p(\text{accept}) = \min\left(1,\ \frac{p(\mathbf{x})}{p(\mathbf{x}')}\ \frac{q(\mathbf{x};\, \mathbf{x}')}{q(\mathbf{x}';\, \mathbf{x})}\right)$$

Metropolis–Hastings
in an ideal world

$$\boxed{\mathbf{x}' \sim q(\mathbf{x}'; \mathbf{x})}$$

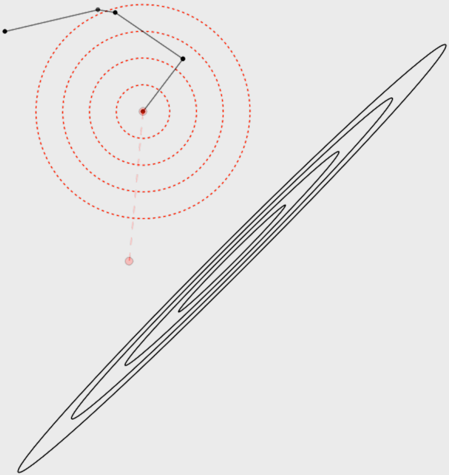


$$\boxed{p(\text{accept}) = \min\left(1, \frac{p(\mathbf{x})}{p(\mathbf{x}')} \frac{q(\mathbf{x}; \mathbf{x}')}{q(\mathbf{x}'; \mathbf{x})}\right)}$$

not this time.

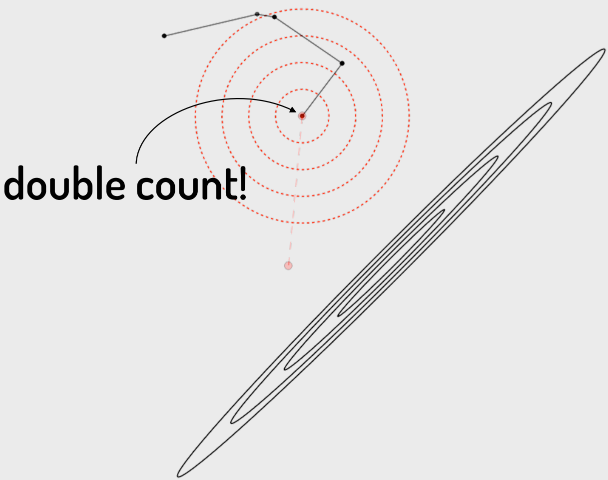
Metropolis–Hastings

in an ideal world



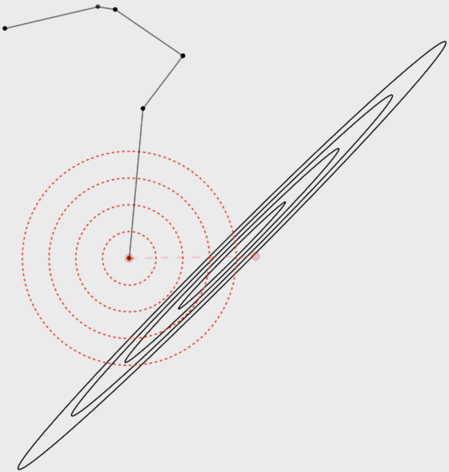
Metropolis–Hastings

in an ideal world



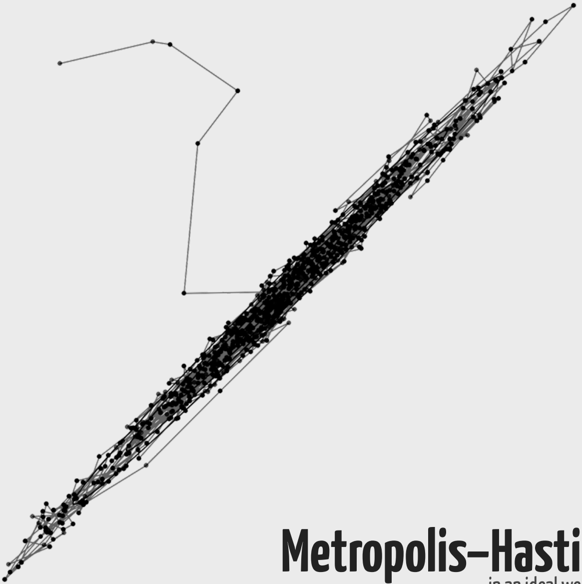
Metropolis–Hastings

in an ideal world



Metropolis–Hastings

in an ideal world



Metropolis–Hastings
in an ideal world

Note

- In the previous slides, there is an unfortunate typo! In all cases,
$$p(\text{accept}) = \min(1, \frac{p(x)}{p(x')}\frac{q(x;x')}{q(x';x)})$$
should actually be
$$p(\text{accept}) = \min(1, \frac{p(x')}{p(x)}\frac{q(x';x)}{q(x;x')})$$
That is, how good is the new place divided by how good was the old place?

About the name

- Monte Carlo: a reference to the famous Monte Carlo Casino in Monaco, alluding to the randomness used in the algorithm
- Markov Chain: a list of samples, where each one is generated by a process that only looks at the previous one.
- Markov: a 19th-century Russian mathematician and impressive-moustache-haver with an extensive list of things named after him
- Metropolis–Hastings: lead authors of 1953 and 1970 papers (resp.) giving the algorithm with symmetric and general proposal distributions (resp.)

The Algorithm (1)

```
function mcmc(prob_func, propose_func, initial_pos, nsteps)
    p = initial_pos
    prob = prob_func(p)
    chain = []
    for i in 1:nsteps
        # propose a new position in parameter space
        # ...
        # compute probability at new position
        # ...
        # decide whether to jump to the new position
        if # ...
            # ...
            # ...
        end
        # save the position
        append!(chain, p)
    end
    return chain
end
```

The Algorithm (2)

```
function mcmc(prob_func, propose_func, initial_pos, nsteps)
    p = initial_pos
    prob = prob_func(p)
    chain = []
    for i in 1:nsteps
        # propose a new position in parameter space
        p_new = propose_func(p)
        # compute probability at new position
        prob_new = prob_func(p_new)
        # decide whether to jump to the new position
        if prob_new / prob > uniform_random()
            p = p_new
            prob = prob_new
        end
        # save the position
        append!(chain, p)
    end
    return chain
end
```

The Algorithm (3)

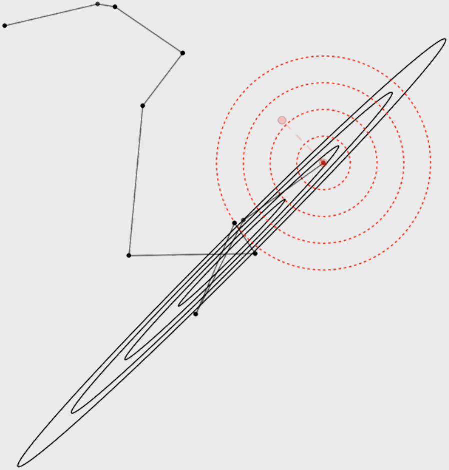
```julia
function mcmc(logprob_func, propose_func, initial_pos, nsteps)
    p = initial_pos
    logprob = logprob_func(p)
    chain = []
    for i in 1:nsteps
        # propose a new position in parameter space
        p_new = propose_func(p)
        # compute probability at new position
        logprob_new = logprob_func(p_new)
        # decide whether to jump to the new position
        if exp(logprob_new - logprob) > uniform_random()
            p = p_new
            logprob = logprob_new
        end
        # save the position
        append!(chain, p)
    end
    return chain
end
```

The Algorithm (4)

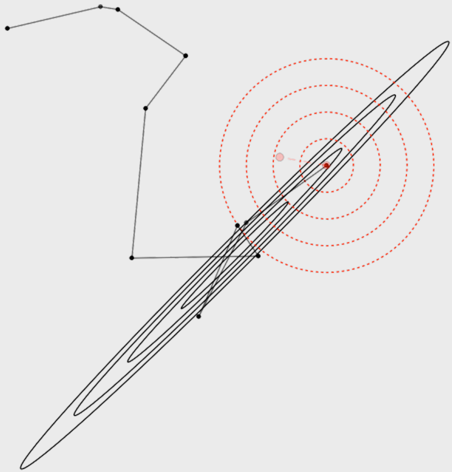
```
function mcmc(logprob_func, propose_func, initial_pos, nsteps)
    p = initial_pos
    logprob = logprob_func(p)
    chain = []
    naccept = 0
    for i in 1:nsteps
        # propose a new position in parameter space
        p_new = propose_func(p)
        # compute probability at new position
        logprob_new = logprob_func(p_new)
        # decide whether to jump to the new position
        if exp(logprob_new - logprob) > uniform_random():
            p = p_new
            logprob = logprob_new
            naccept += 1
        end
        # save the position
        append!(chain, p)
    end
    return chain, naccept/nsteps
```

Practicalities

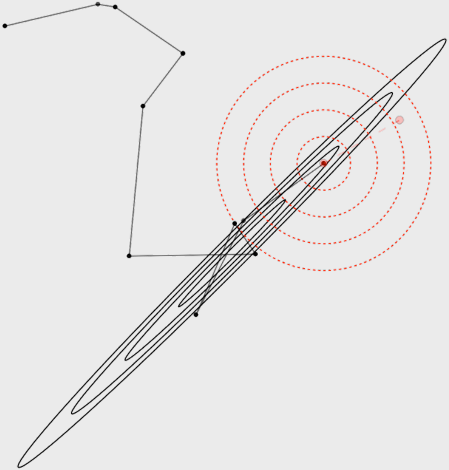
- How do I choose a proposal distribution?
- How many steps do I have to take?



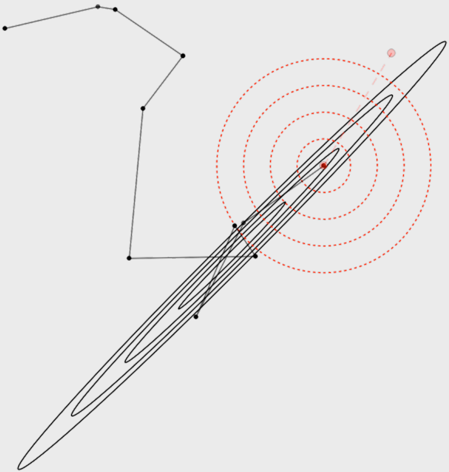
Metropolis–Hastings
in the real world



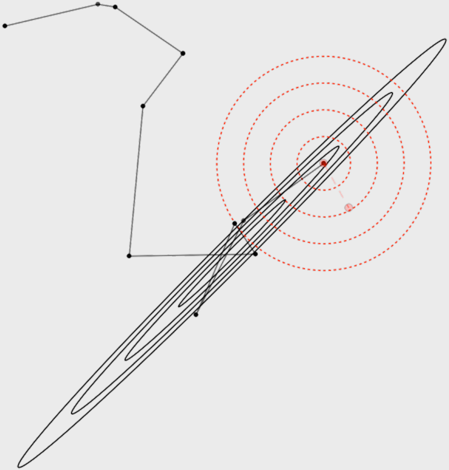
Metropolis–Hastings
in the real world



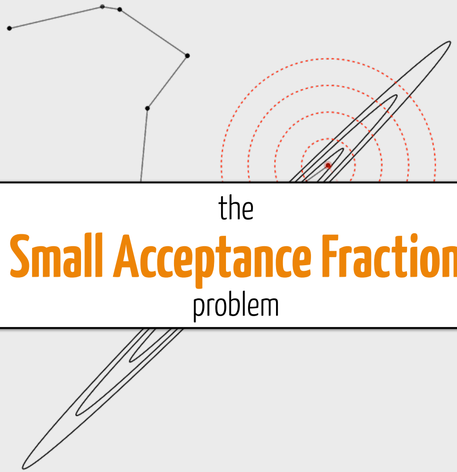
Metropolis–Hastings
in the real world



Metropolis–Hastings
in the real world



Metropolis–Hastings
in the real world



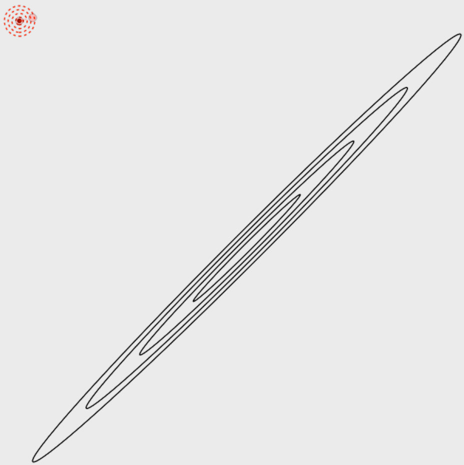
the

Small Acceptance Fraction

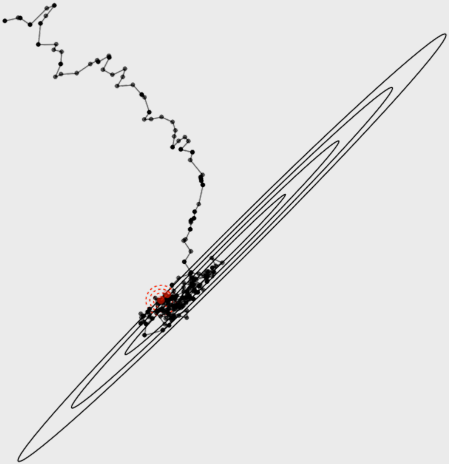
problem

Metropolis–Hastings

in the real world



Metropolis–Hastings
in the real world



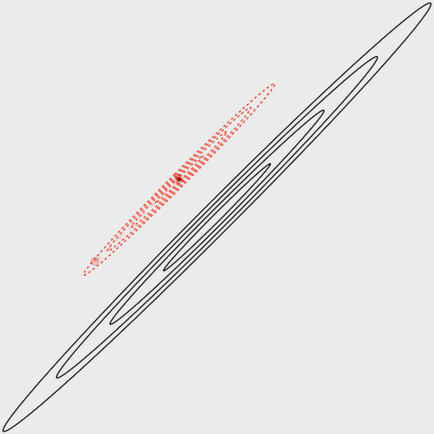
Metropolis–Hastings
in the real world

the

Huge Acceptance Fraction

problem

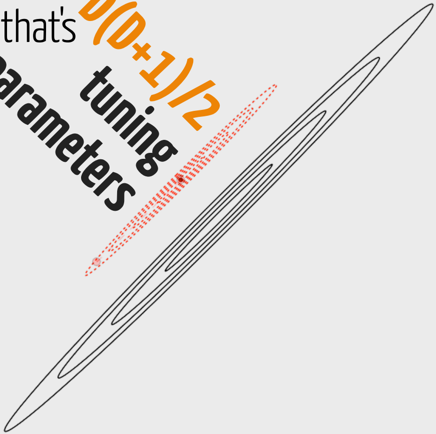
Metropolis–Hastings
in the real world



Metropolis–Hastings

in the real world

that's $D(D+1)/2$ tuning parameters

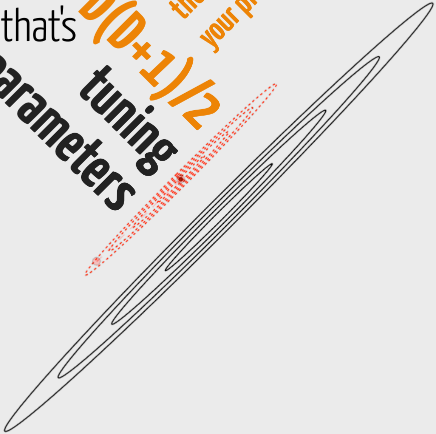


Metropolis–Hastings

in the real world

that's D(D+1)/2 tuning parameters

the dimension of your problem

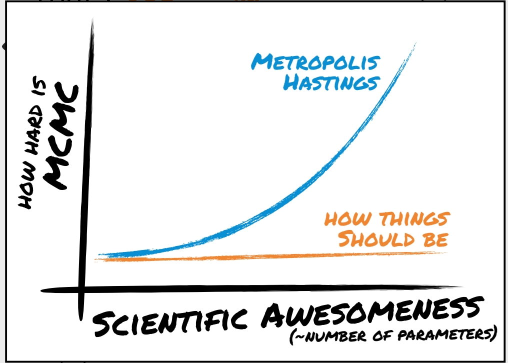


Metropolis–Hastings

in the real world

that's

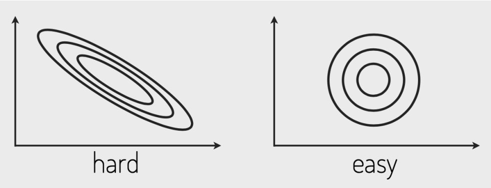
the dimension of
our problem



Metropolis–Hastings
in the real world

A connection to symmetries

- In Metropolis–Hastings MCMC, the *proposal distribution* needs tuning parameters, especially as dimensionality increases
- Can be seen as a lack of symmetry in the algorithm—the algorithm is sensitive to the parameterization of the problem
- For example, it's not invariant to an affine transformation
- Next lecture, I'll show you an alternative algorithm that does have affine invariance



How many samples do I need?

- Burn-in — skip the first N samples
- *Has my chain converged?*
- MCMC produces correlated samples, so
  - How correlated are my samples?
    - Can measure the *autocorrelation time* τ
    - Keep $1/\tau$ of the MCMC samples
    - eg *https://github.com/dfm/acor*
  - How many uncorrelated samples do I need?
    - No easy general answer to this question!
    - “How many can you afford?”

How many samples do I need?

- Burn-in — skip the first N samples
- *Has my chain converged?*
- MCMC produces correlated samples, so
  - How correlated are my samples?
    - Can measure the *autocorrelation time* τ
    - Keep $1/\tau$ of the MCMC samples
    - eg *https://github.com/dfm/acor*
  - How many uncorrelated samples do I need?
    - No easy general answer to this question!
    - "How many can you afford?"

How many samples do I need?

- Burn-in — skip the first N samples
- *Has my chain converged?*
- MCMC produces correlated samples, so
  - How correlated are my samples?
    - Can measure the *autocorrelation time* τ
    - Keep $1/\tau$ of the MCMC samples
    - eg *https://github.com/dfm/acor*
  - How many uncorrelated samples do I need?
    - No easy general answer to this question!
    - “How many can you afford?”

Conclusions

- MCMC remains an essential tool for probabilistic inference
- For science: lets us contrain model parameters based on data (Bayesian inference)
- Beguilingly simple algorithm, but difficult practicalities
- MCMC has beautiful theoretical guarantees... as compute time $\to \infty$