Markov Chain Monte Carlo

Dustin Lang
Perimeter Institute for Theoretical Physics

PSI Numerical Methods 2026

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

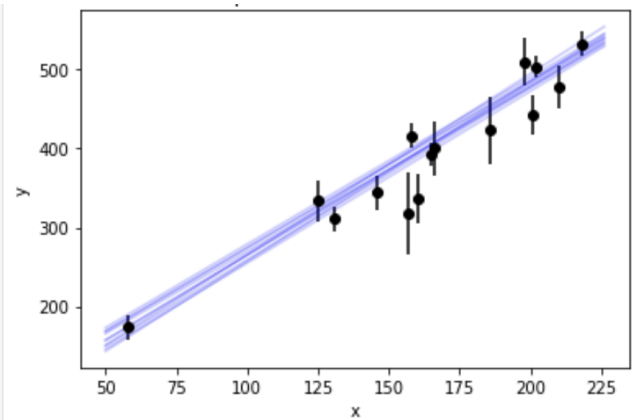
These slides are available at
https://github.com/dstndstn/MCMC-talk

Context - Generative data analysis

- In Astrophysics, often have a *generative model* of a phenomenon of interest:
- We can *simulate* or *predict* what we will observe with a *model*
- The model has *parameters*, and we want to put *constraints* on these parameters:
- *What range of parameters are consistent with our observations?*

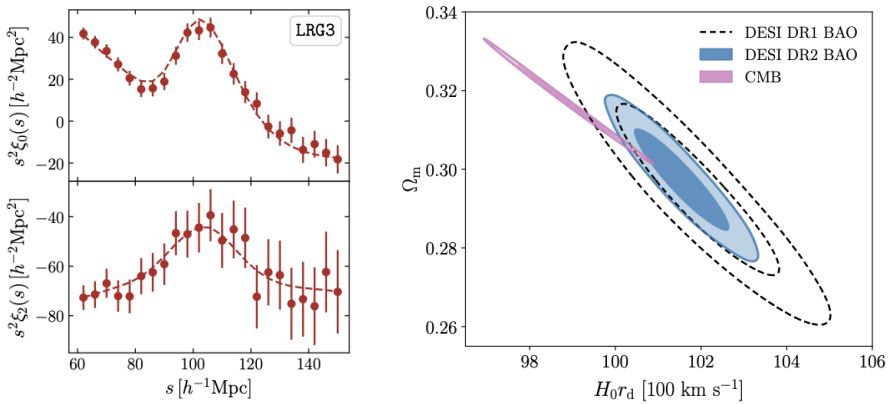
Context - Generative data analysis

- We can *simulate* or *predict* what we will observe with a *parameterized model*: $\hat{y}_i = f(\theta, x_i)$
- We can write down a *likelihood* of getting the observations we got: $\mathcal{L}(y_i|x_i, \theta) = g(y_i, x_i, \theta)$
- Eg, Gaussian likelihoods:
  $\mathcal{L}(y_i|x_i, \theta) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(y_i - f(\theta, x_i))^2}{2\sigma_i^2}\right)$
- *Note:* after we have made our observations, y_i are fixed, as are x_i, σ_i (and f and g); only parameters θ are free
- We want to know about constraints on our *parameters* θ!
- *Bayes' theorem* lets us do this: $p(\theta|y_i, x_i) = \frac{\mathcal{L}(y_i|x_i, \theta)p(\theta)}{p(y_i)}$



Context - Generative data analysis

- *Bayes' theorem* lets us do this:
  $p(\theta|\{y_i\},\{x_i\}) = \frac{\mathcal{L}(\{y_i\}|\{x_i\},\theta)p(\theta)}{p(\{y_i\})}$
- *Usually* we don't care about the $p(\{y_i\})$ ("evidence") term
- *Often* we want the $p(\theta)$ ("prior") term to be "uninformative" (that's not trivial)
- So, we usually want to *draw samples* from the *posterior distribution* $p(\theta|\{y_i\},\{x_i\})$; that's what we report in our paper
- (given samples, we can draw *contours* of the probability distribution):



From https://arxiv.org/abs/2503.14738

MCMC

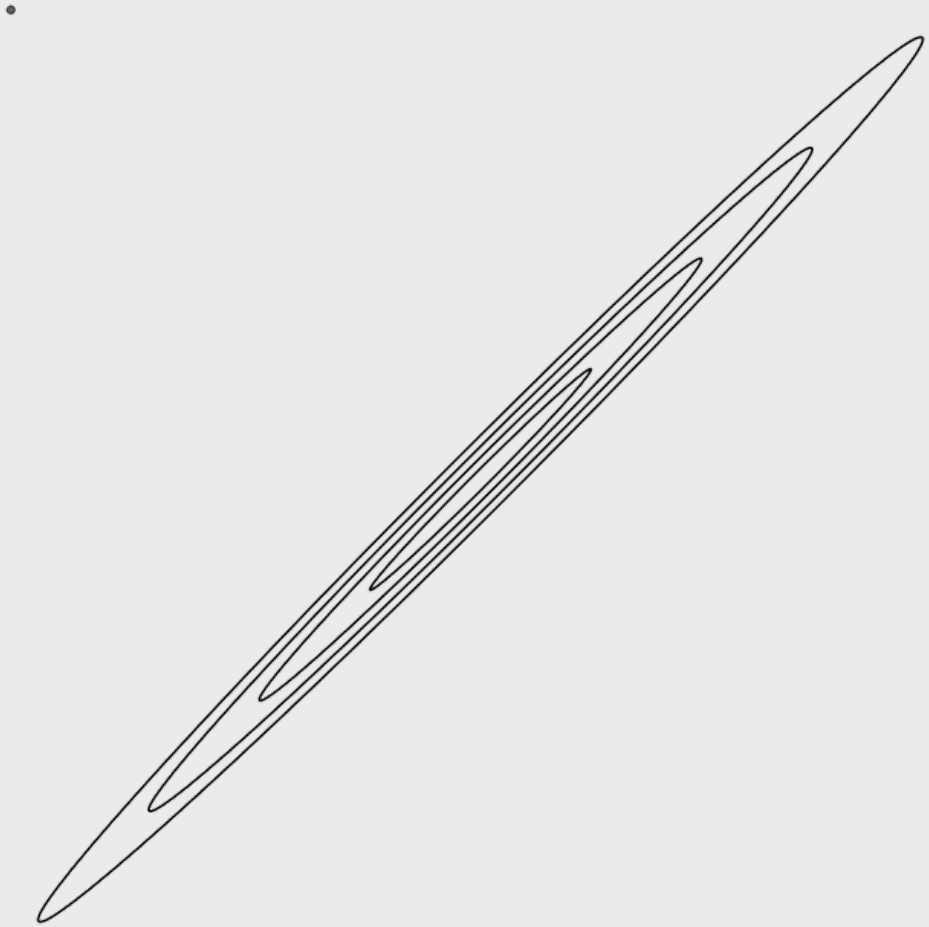
draws samples from a probability function

and all you need to be able to do is

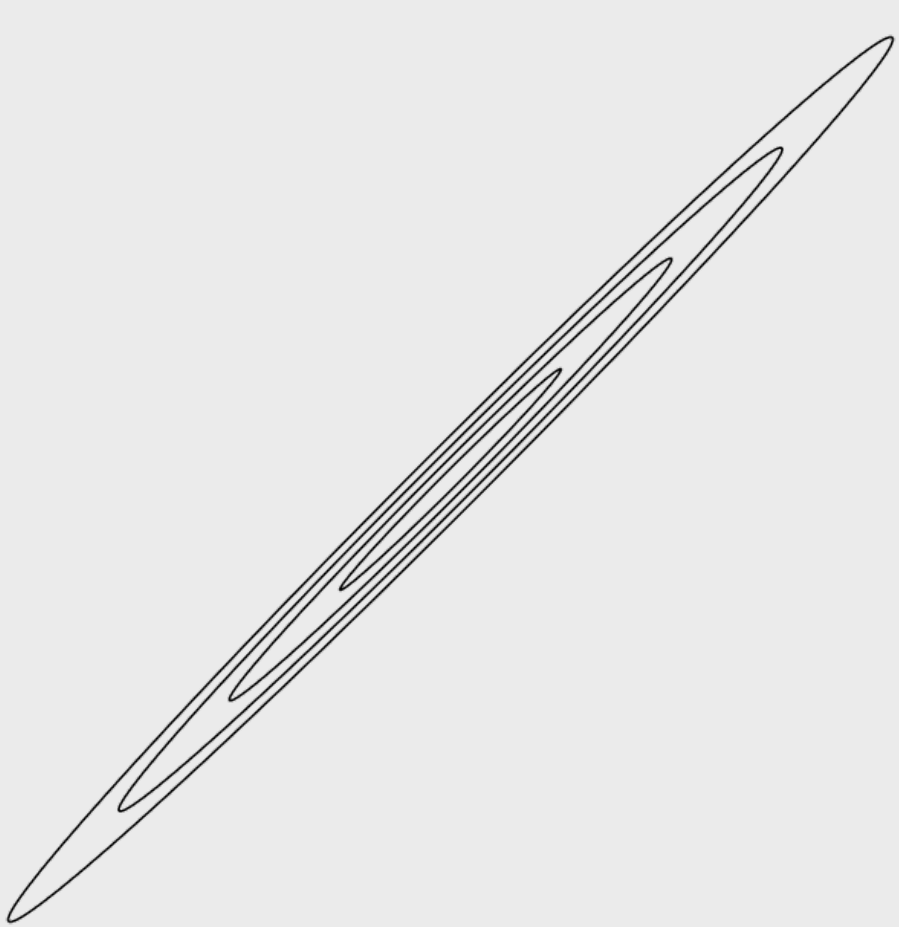
evaluate

the function

(up to a constant)



Metropolis–Hastings

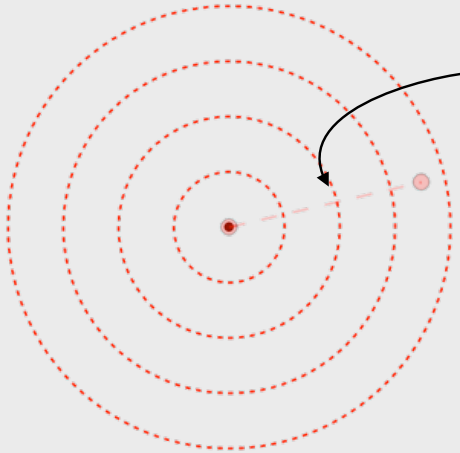


Metropolis–Hastings

in an ideal world

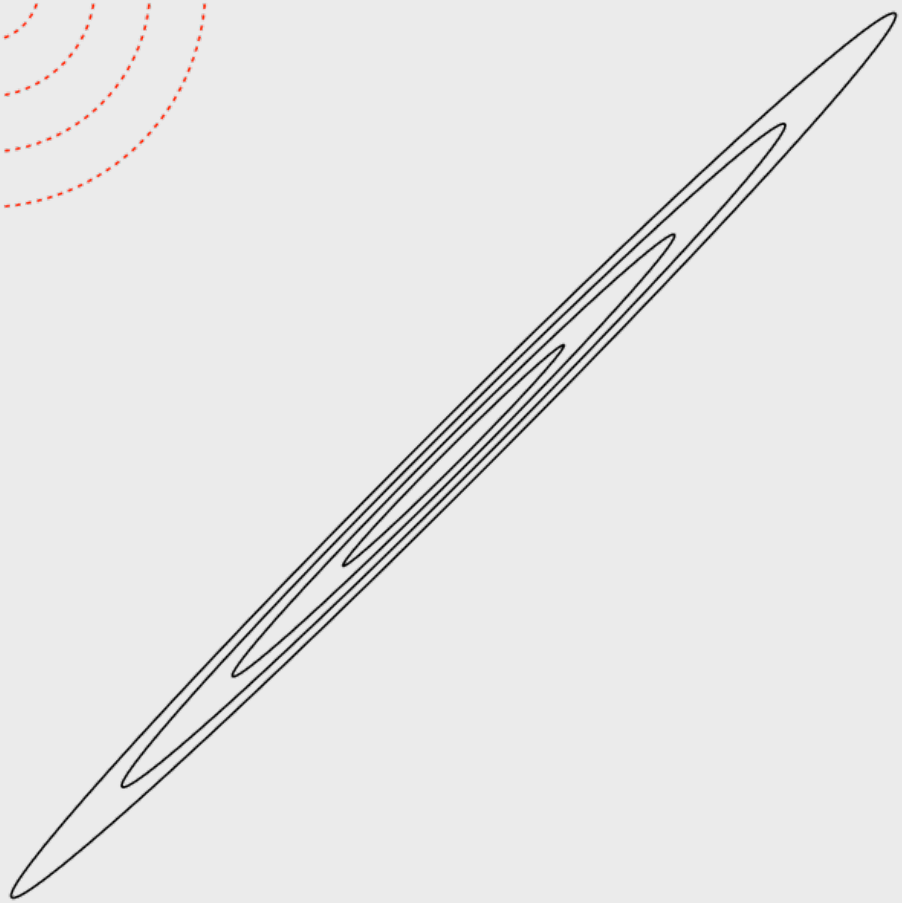
Metropolis–Hastings

in an ideal world

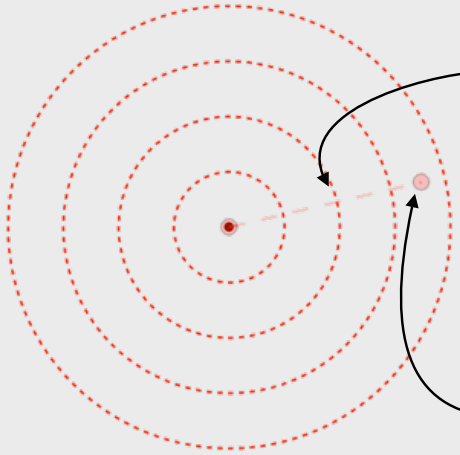


propose
a new position

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



Metropolis–Hastings
in an ideal world



propose
a new position

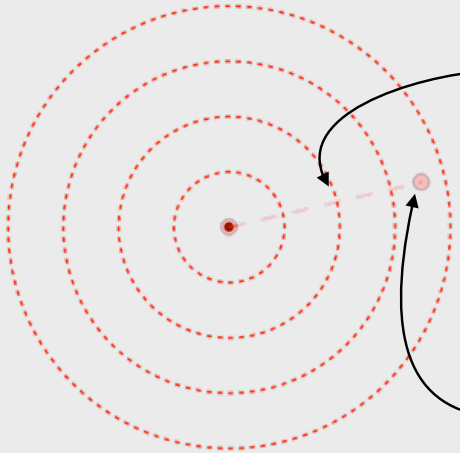
$$\mathbf{x}' \sim q(\mathbf{x}'; \mathbf{x})$$

accept?

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

Metropolis–Hastings

in an ideal world



propose
a new position

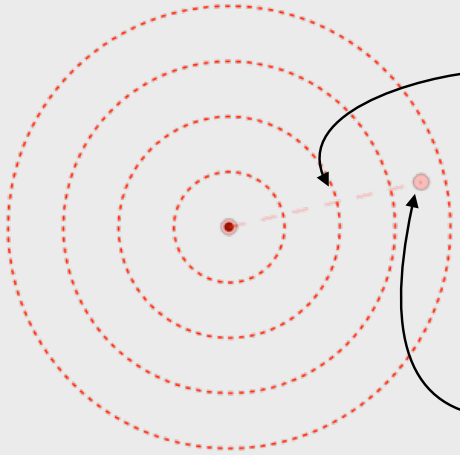
$$\mathbf{x}' \sim q(\mathbf{x}'; \mathbf{x})$$

accept?

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

definitely.

Metropolis–Hastings
in an ideal world



propose
a new position

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

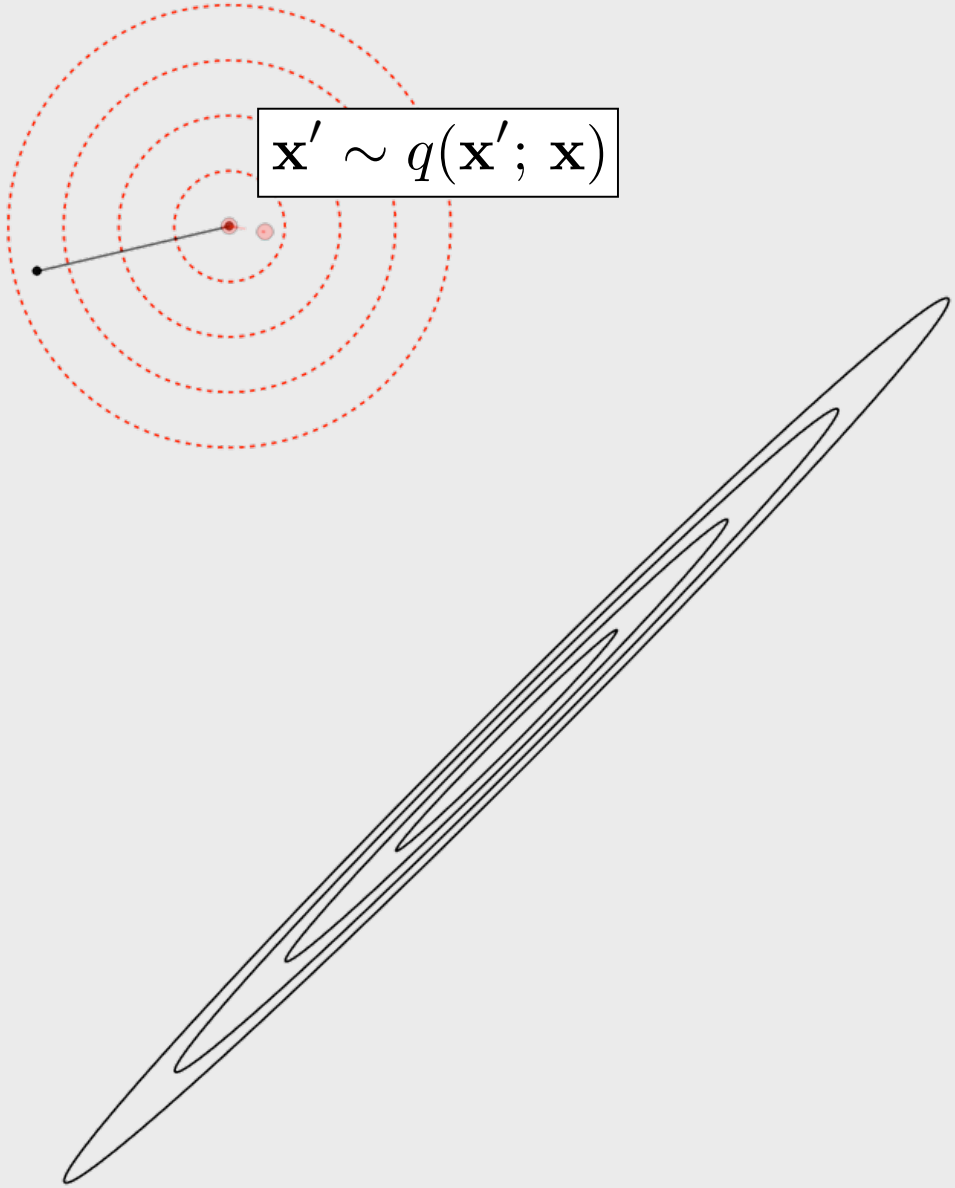
accept?

only **relative** probabilities

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

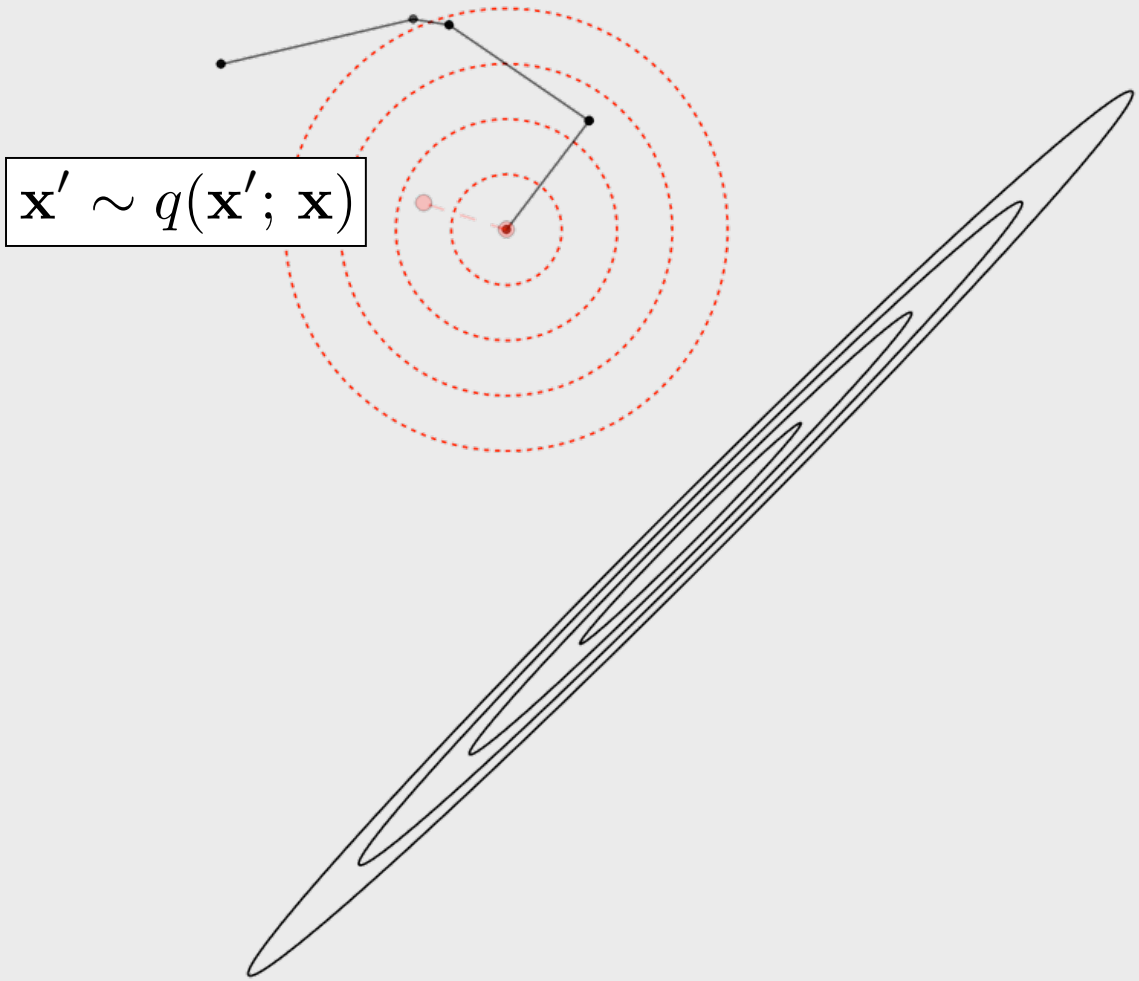
definitely.

Metropolis–Hastings
in an ideal world



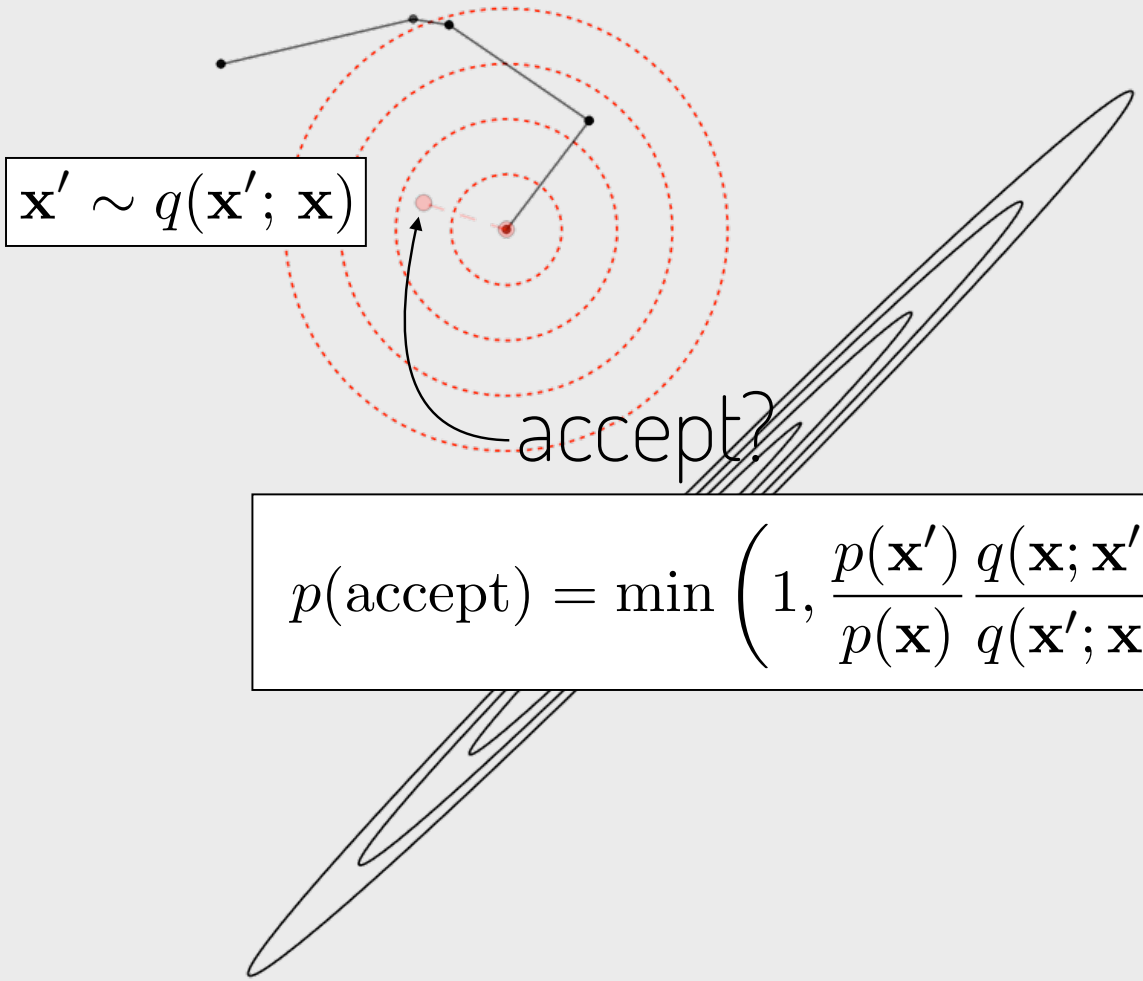
Metropolis–Hastings

in an ideal world



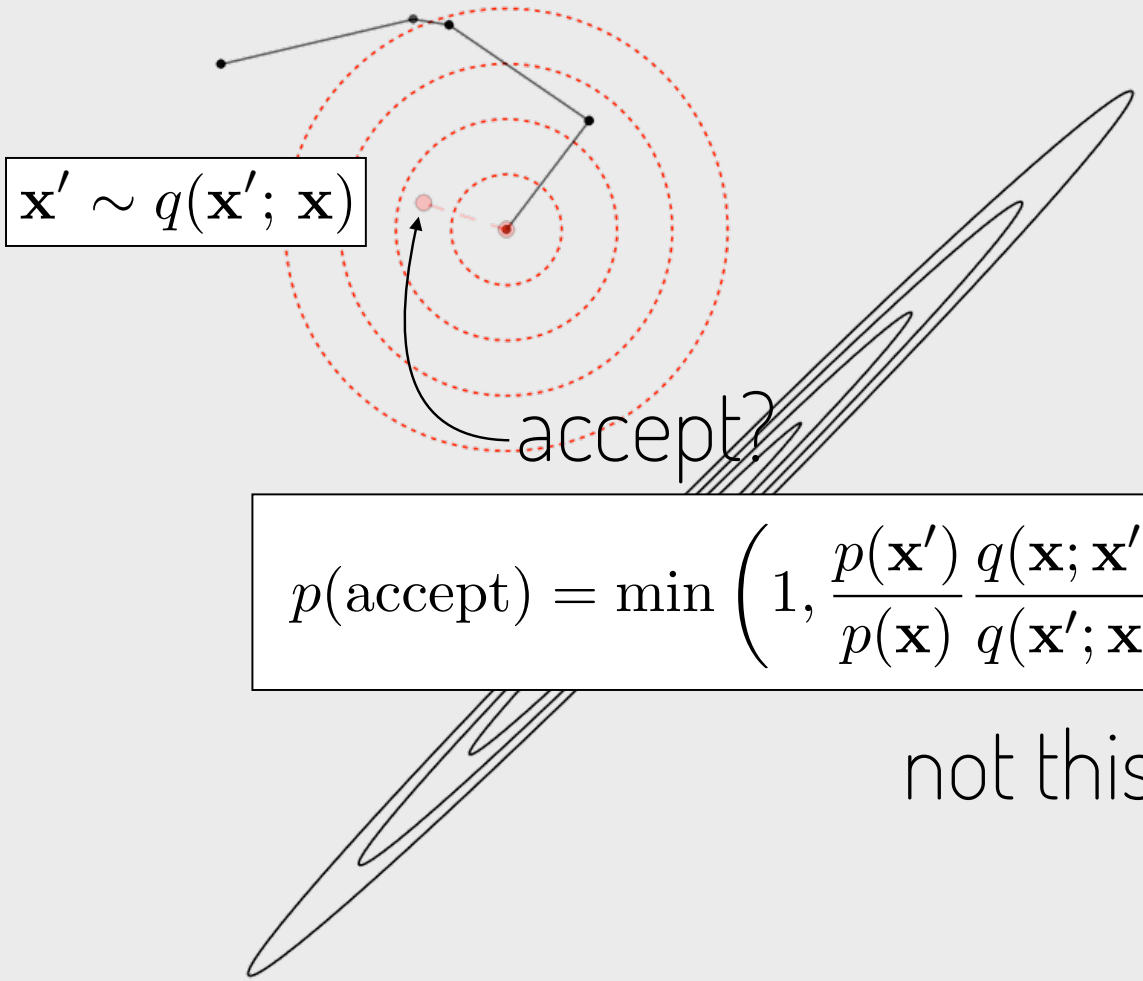
Metropolis–Hastings

in an ideal world



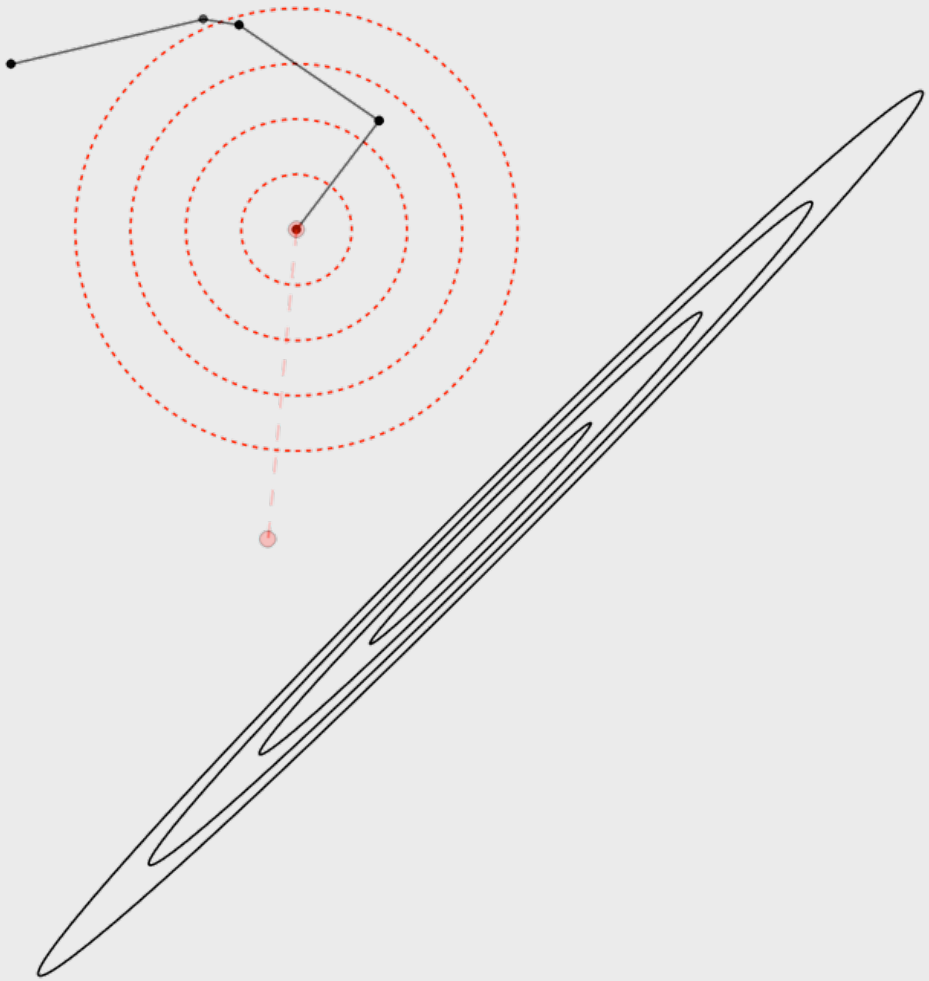
Metropolis–Hastings

in an ideal world



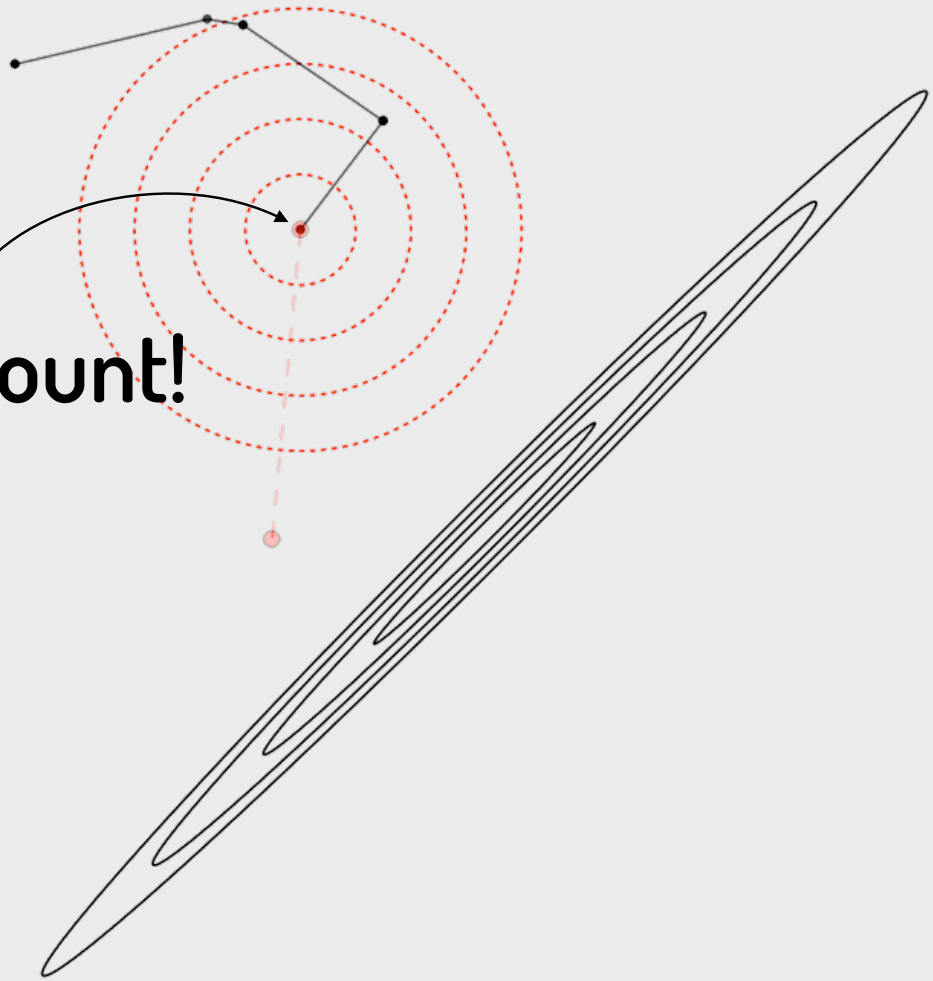
Metropolis–Hastings

in an ideal world



Metropolis–Hastings

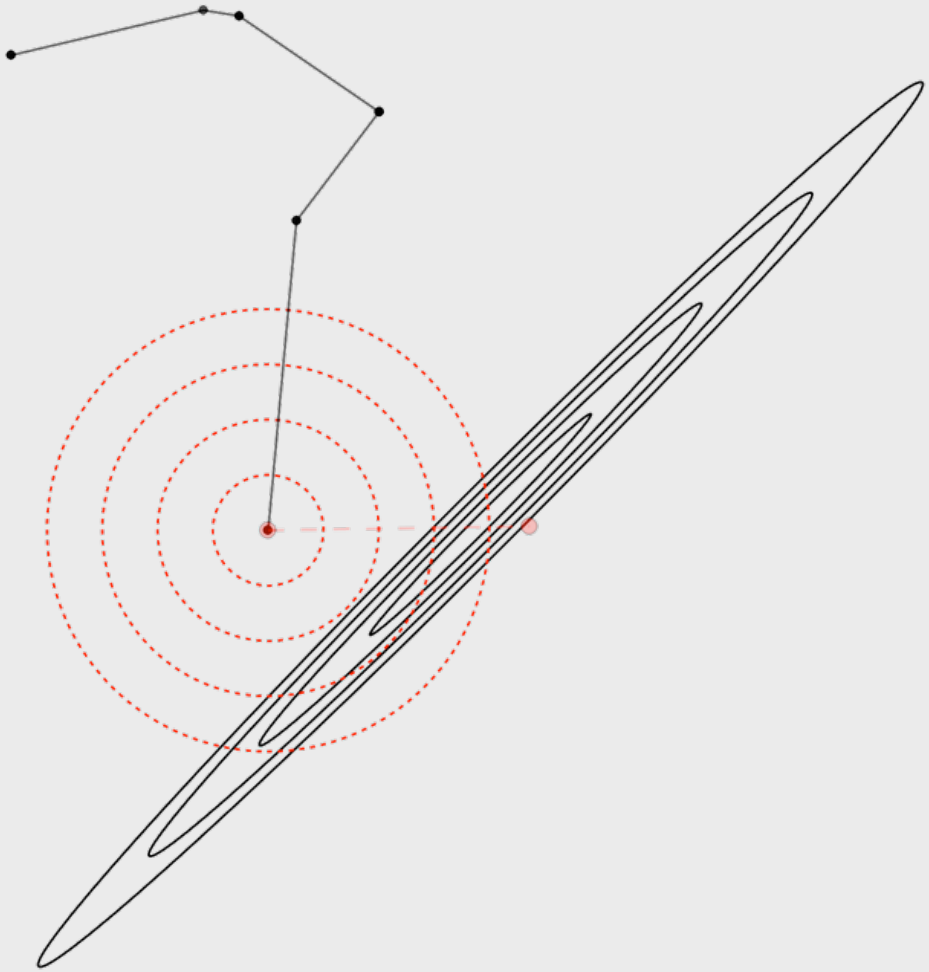
in an ideal world



double count!

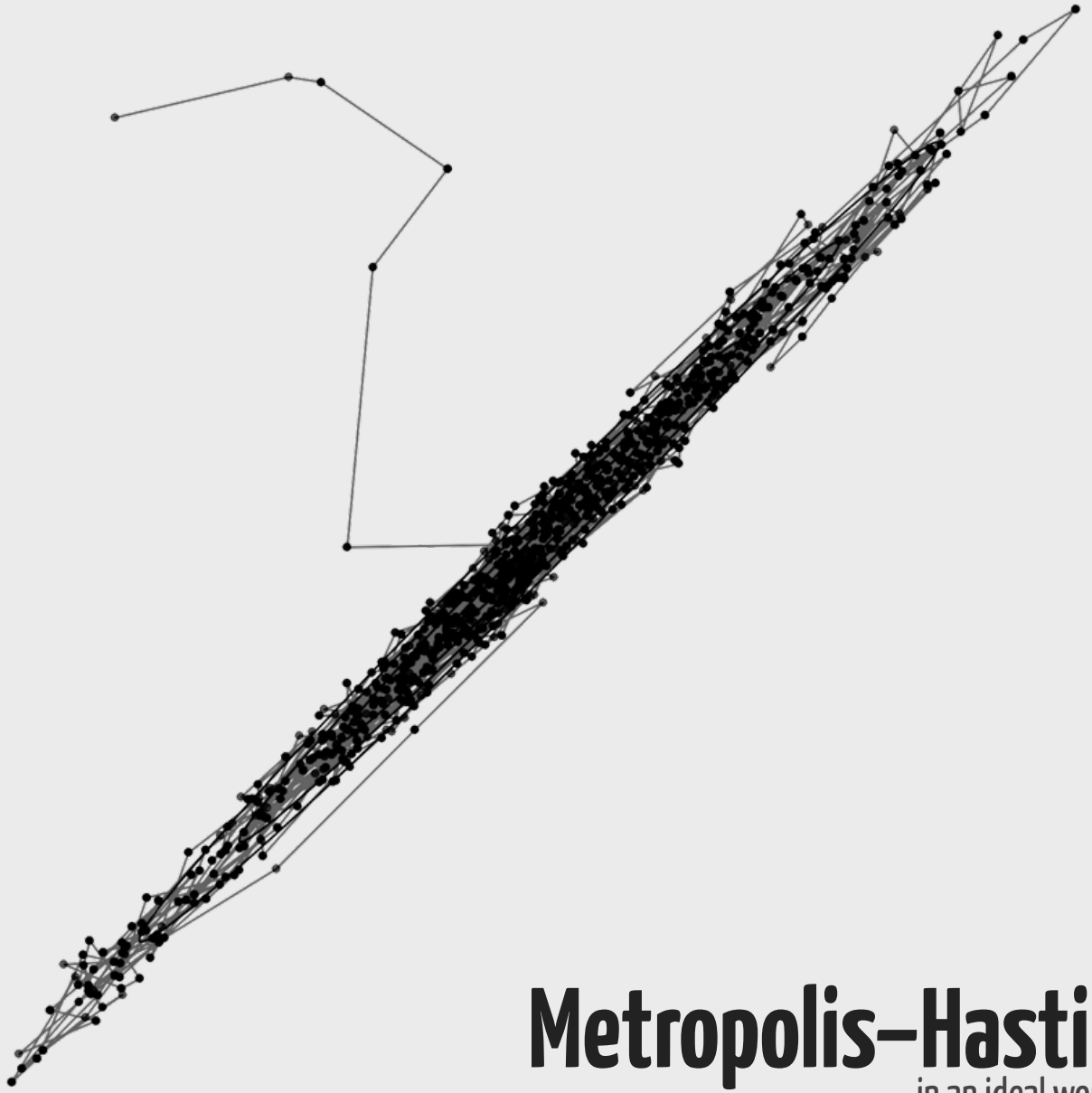
Metropolis–Hastings

in an ideal world



Metropolis–Hastings

in an ideal world



Metropolis–Hastings

in an ideal world

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-centure 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
        ratio = prob_new / prob
        if ratio > 1 or ratio > uniform_random()
            p = p_new
            prob = prob_new
        end
        # save the position
        append!(chain, p)
    end
    return chain
end
```

The Algorithm (3)

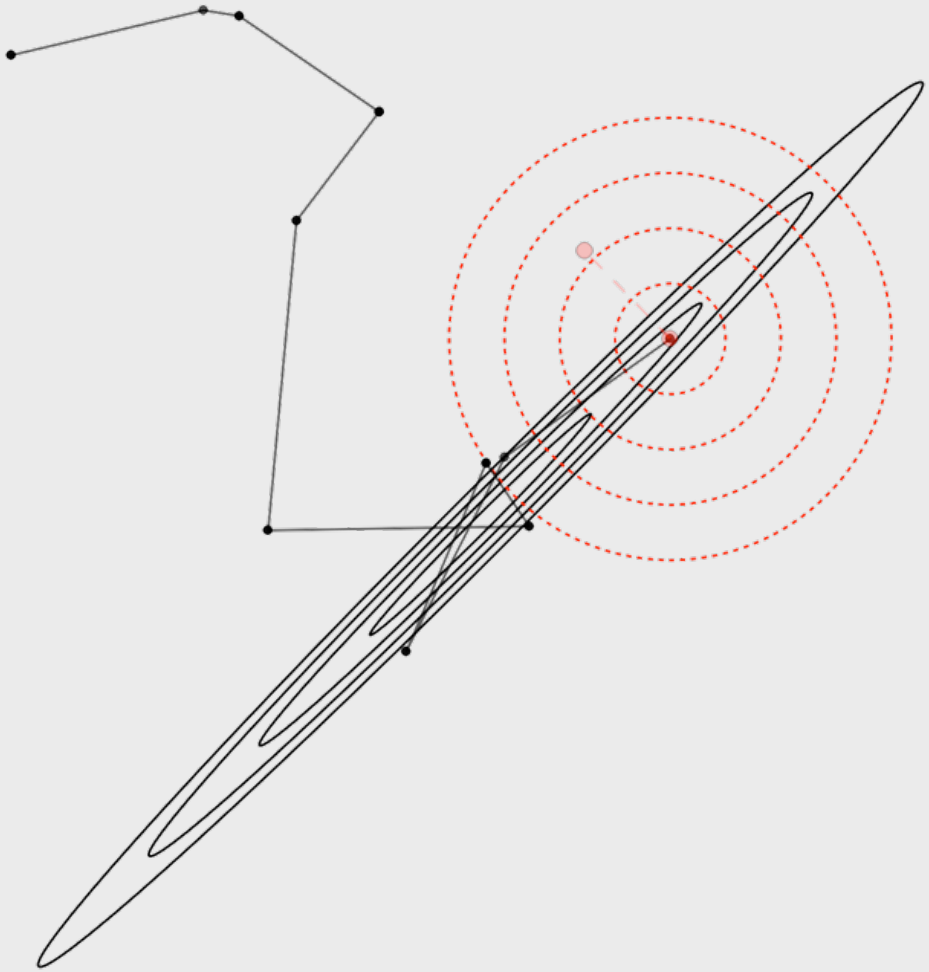
```
function mcmc(logprob_func, propose_func, initial_pos, nsteps)
    p = initial_pos
    logprob = logprob_func(p)
    chain = []
    for i in 1:nsteps
        # propose a new position in parameter space
        p_new = propose_func(p)
        # compute probability at new position
        logprob_new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

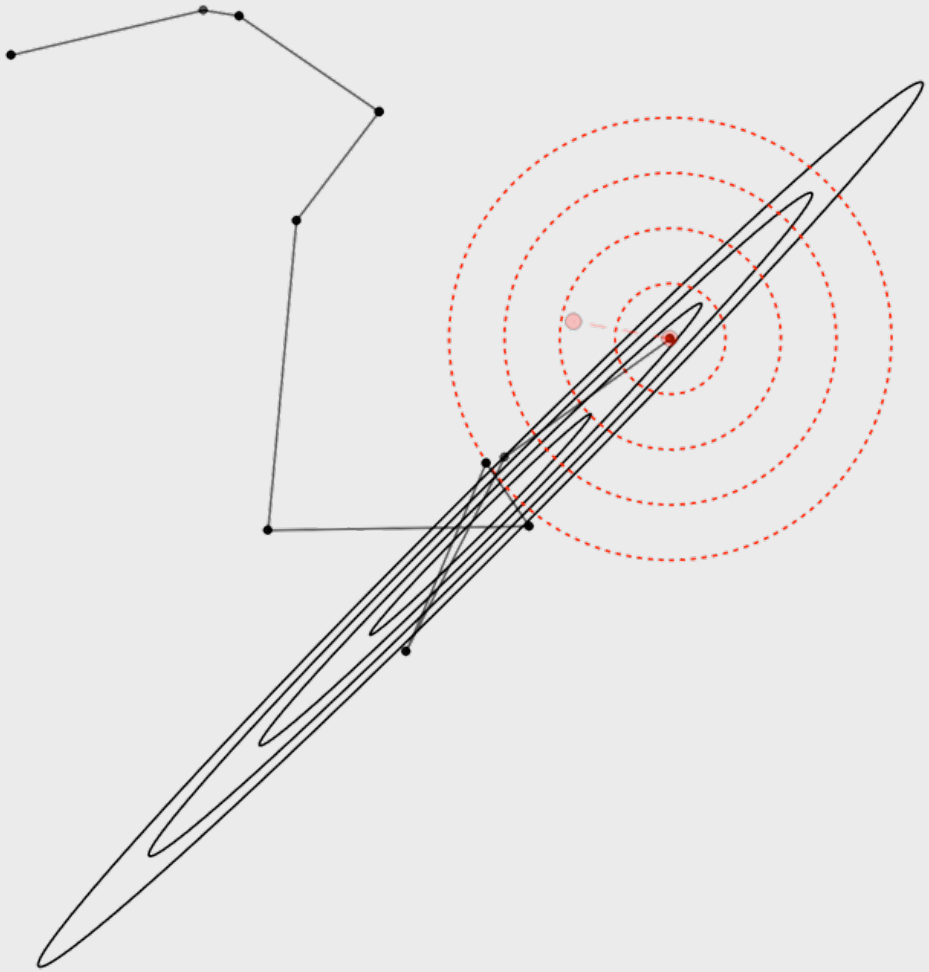
Practicalities

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



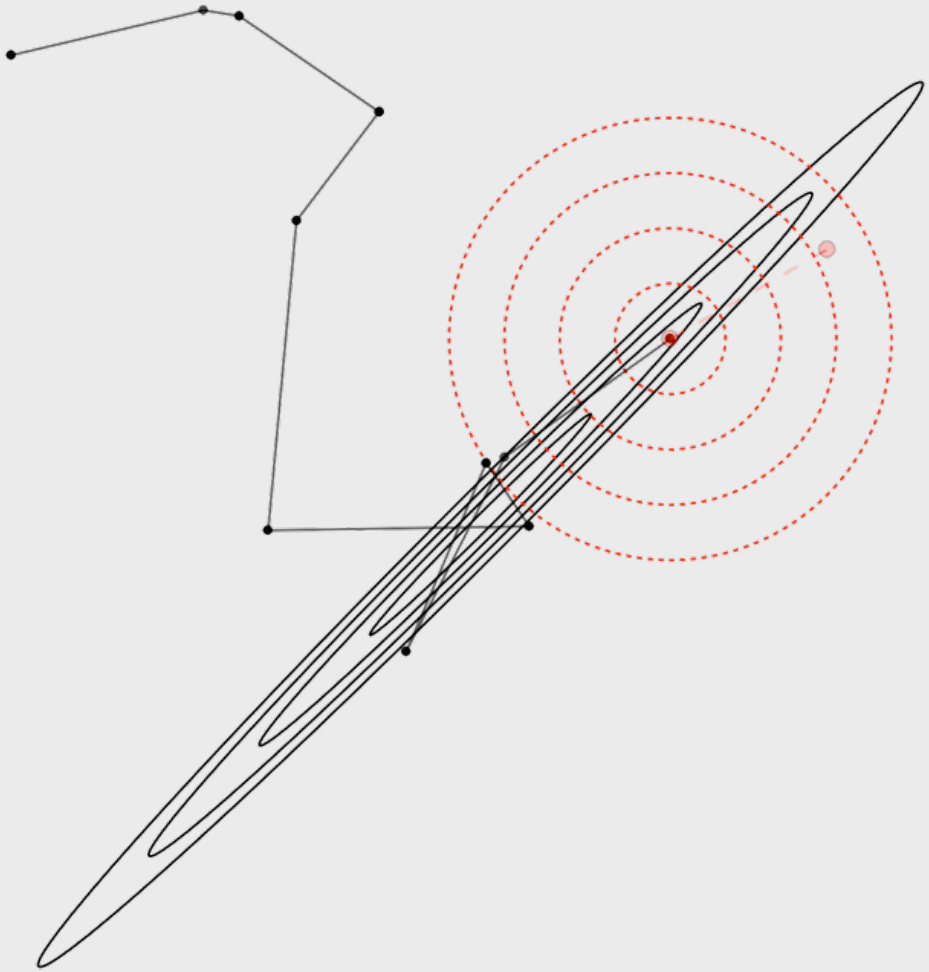
Metropolis–Hastings

in the real world



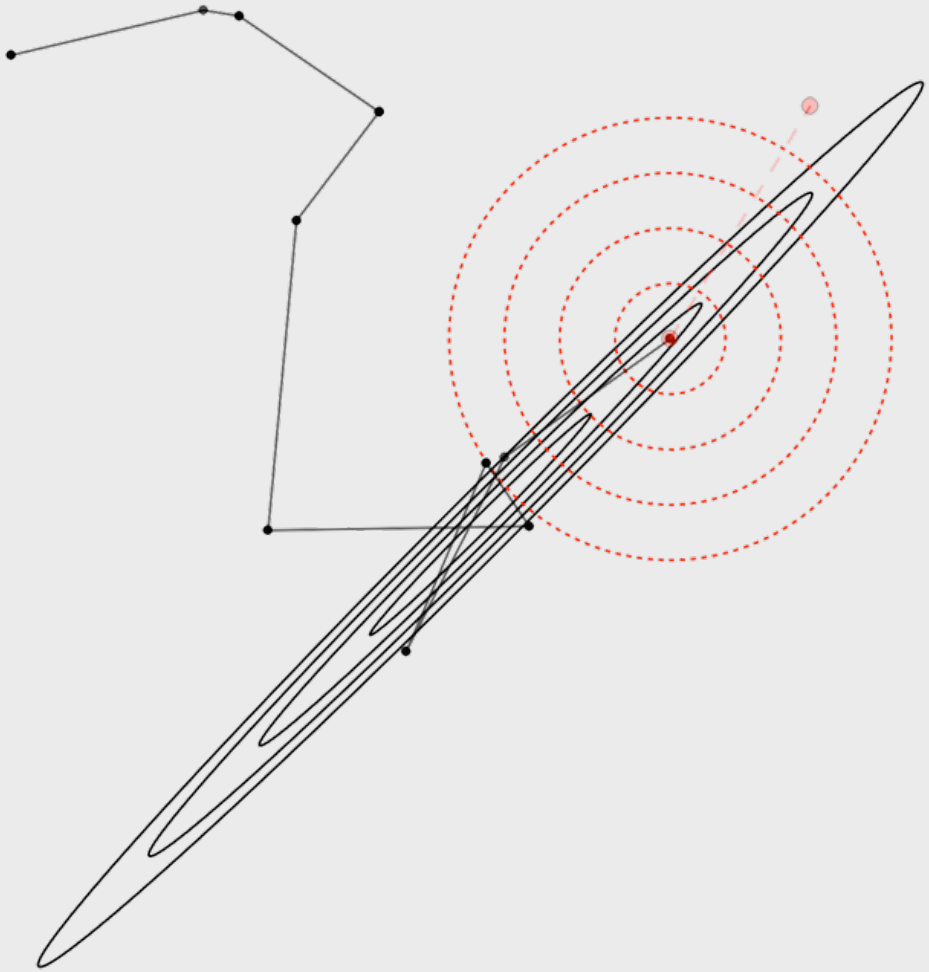
Metropolis–Hastings

in the real world



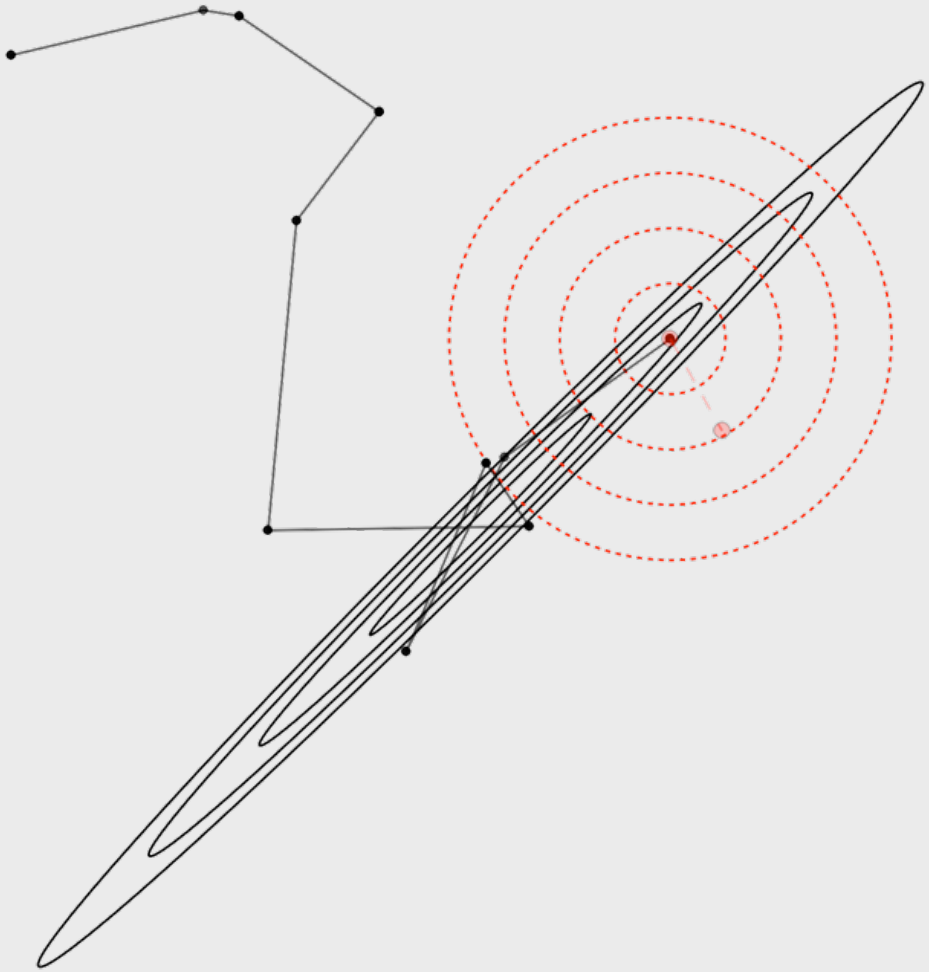
Metropolis–Hastings

in the real world



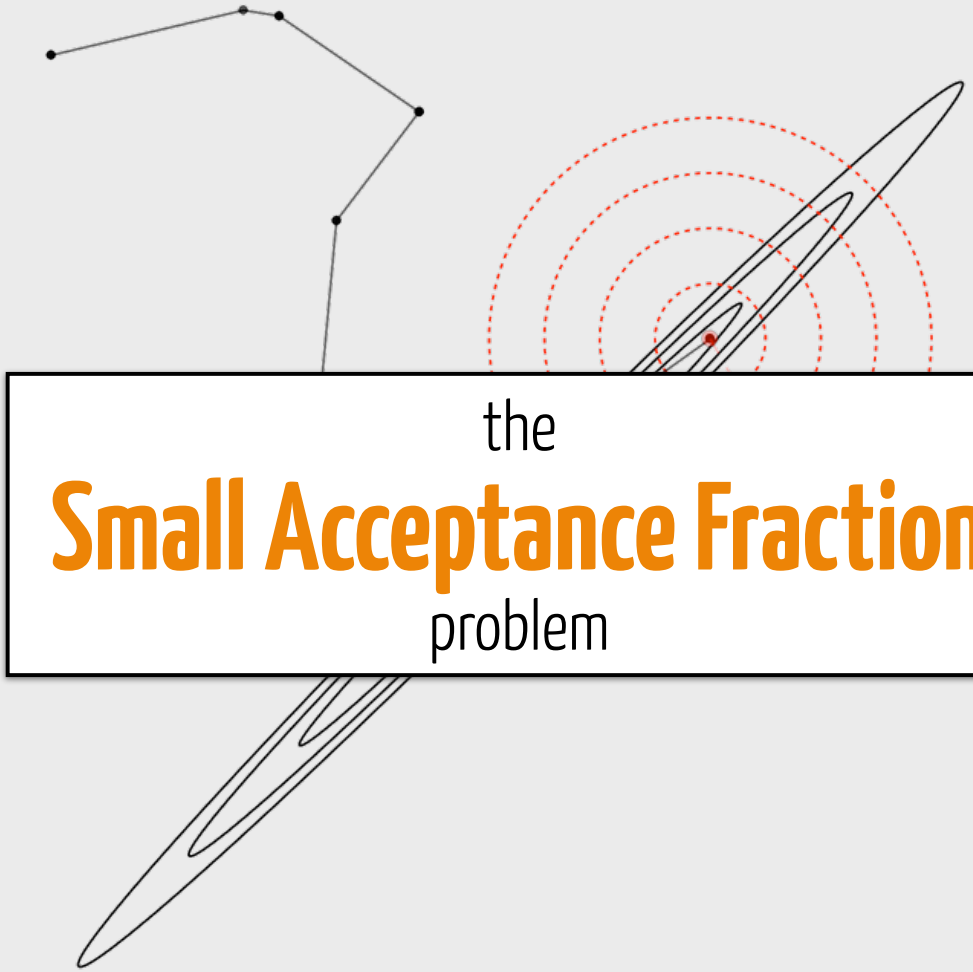
Metropolis–Hastings

in the real world



Metropolis–Hastings

in the real world



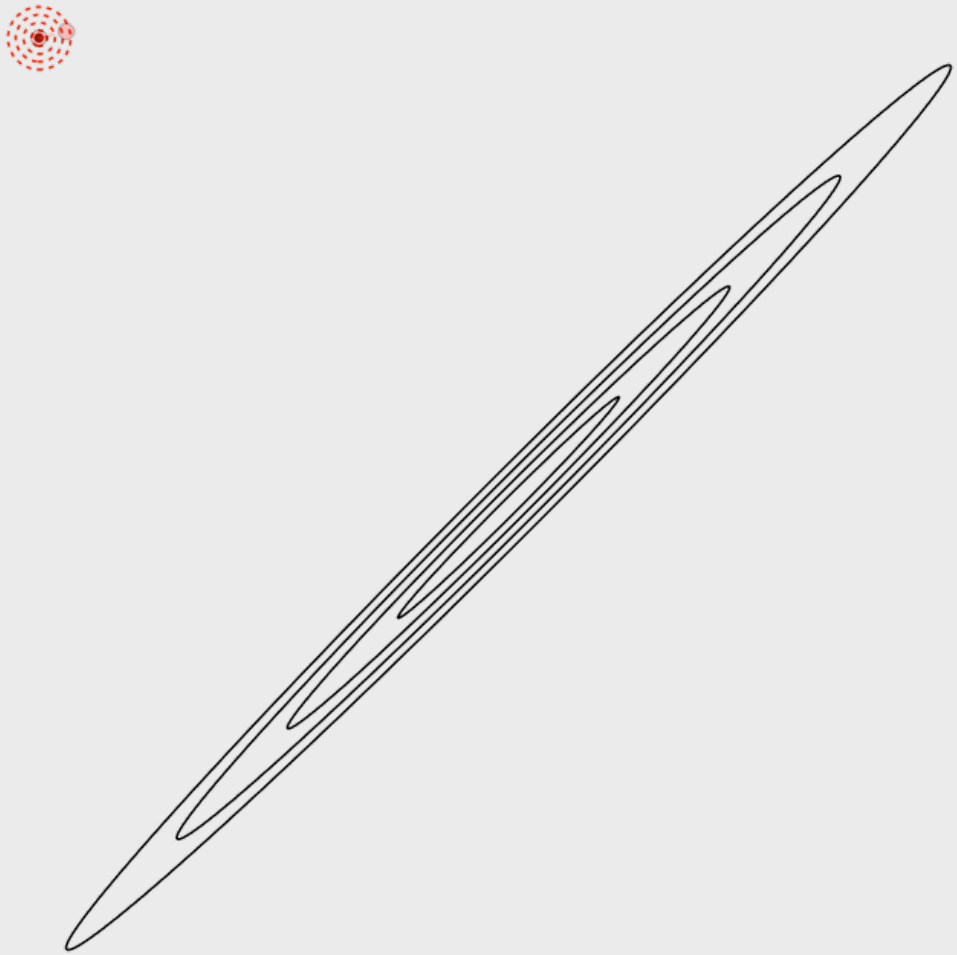
the

Small Acceptance Fraction

problem

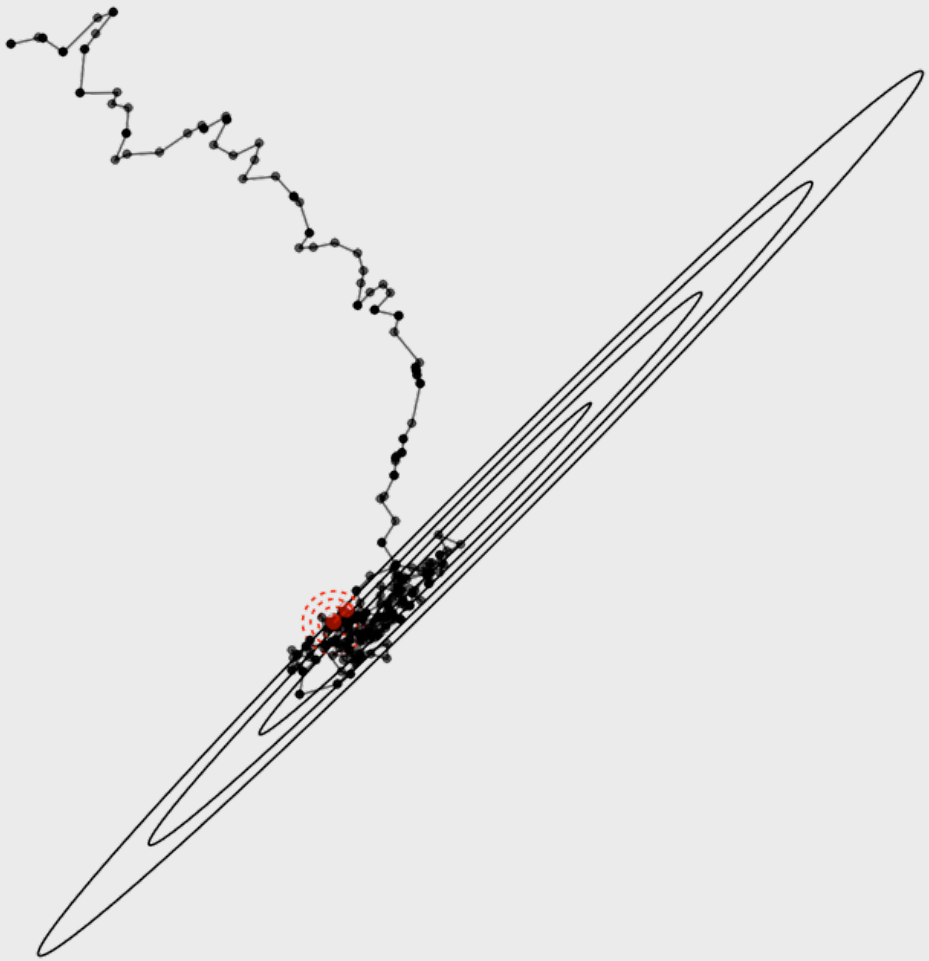
Metropolis–Hastings

in the real world



Metropolis–Hastings

in the real world



Metropolis–Hastings

in the real world

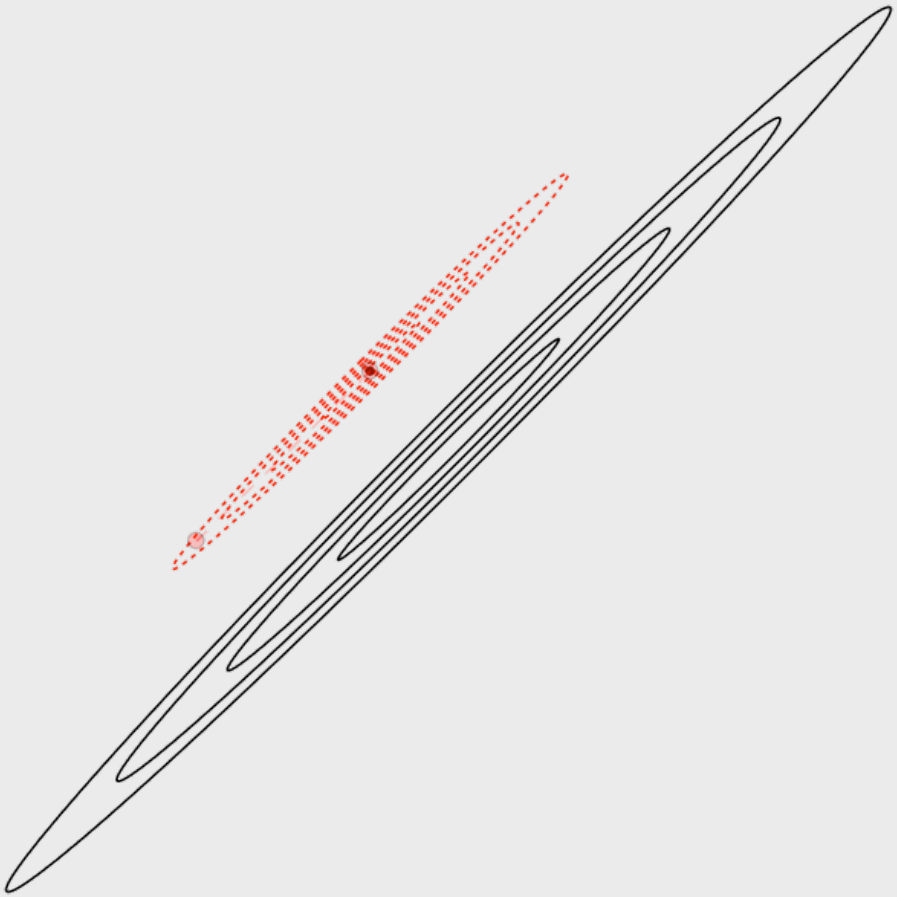
the

Huge Acceptance Fraction

problem

Metropolis–Hastings

in the real world

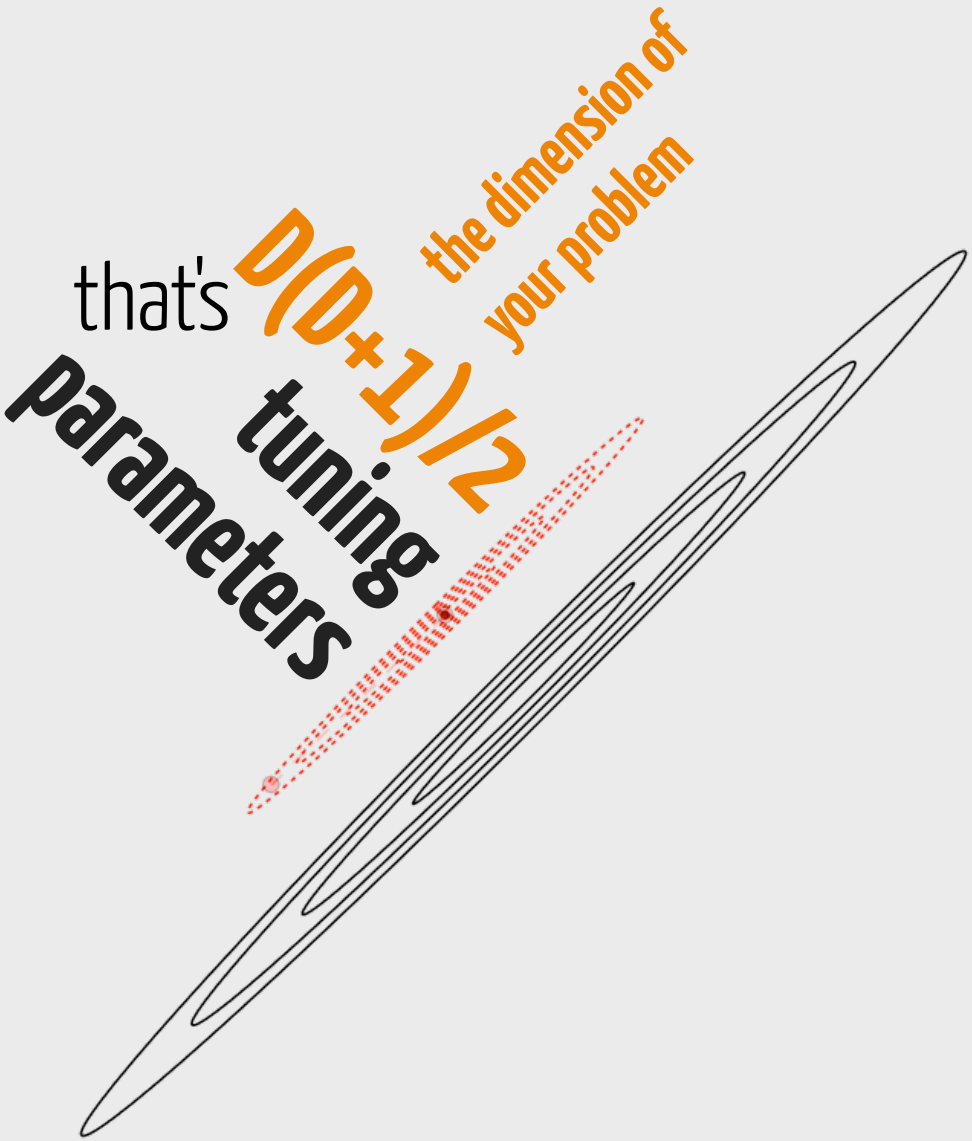


Metropolis–Hastings

in the real world

Metropolis–Hastings

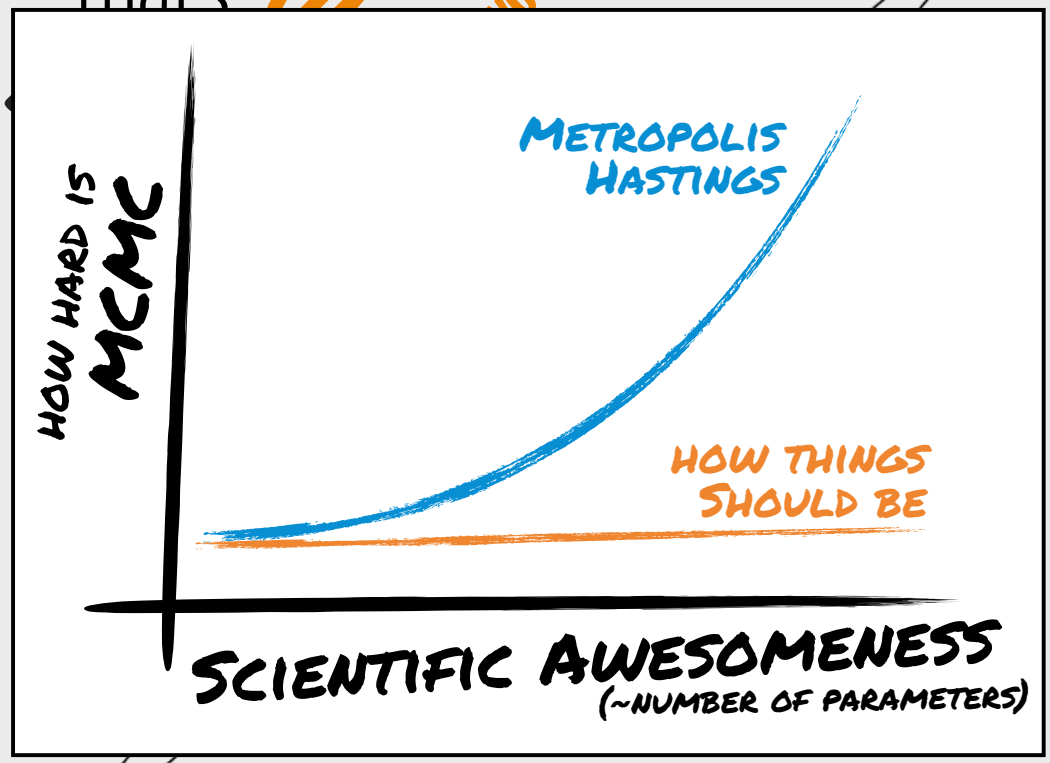
in the real world



Metropolis–Hastings

in the real world

that's D the dimension of your problem



Metropolis–Hastings
in the real world

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 $\to \infty$