Tutorial 6

Statistical Computation and Analysis

Spring 2025

Tutorial Outline

- Grid Method
- MCMC
  - Metropolis-Hastings
- Diagnostics
  - Convergence
  - Divergence

Grid Method

- $P(\text{model}|\text{data}) = \frac{P(\text{data}|\text{model})P(\text{model})}{\sum_{\text{models}} P(\text{data}|\text{model})P(\text{model})}$
- The marginal likelihood is often a challenging and computationally expensive integral to calculate.
- If we know that prior and the likelihood at a certain point
  - We can multiply them and get something proportional to the posterior
- If we know the prior and likelihood at a bunch of points
  - We can multiply them, and then normalize, and get an estimation of the posterior

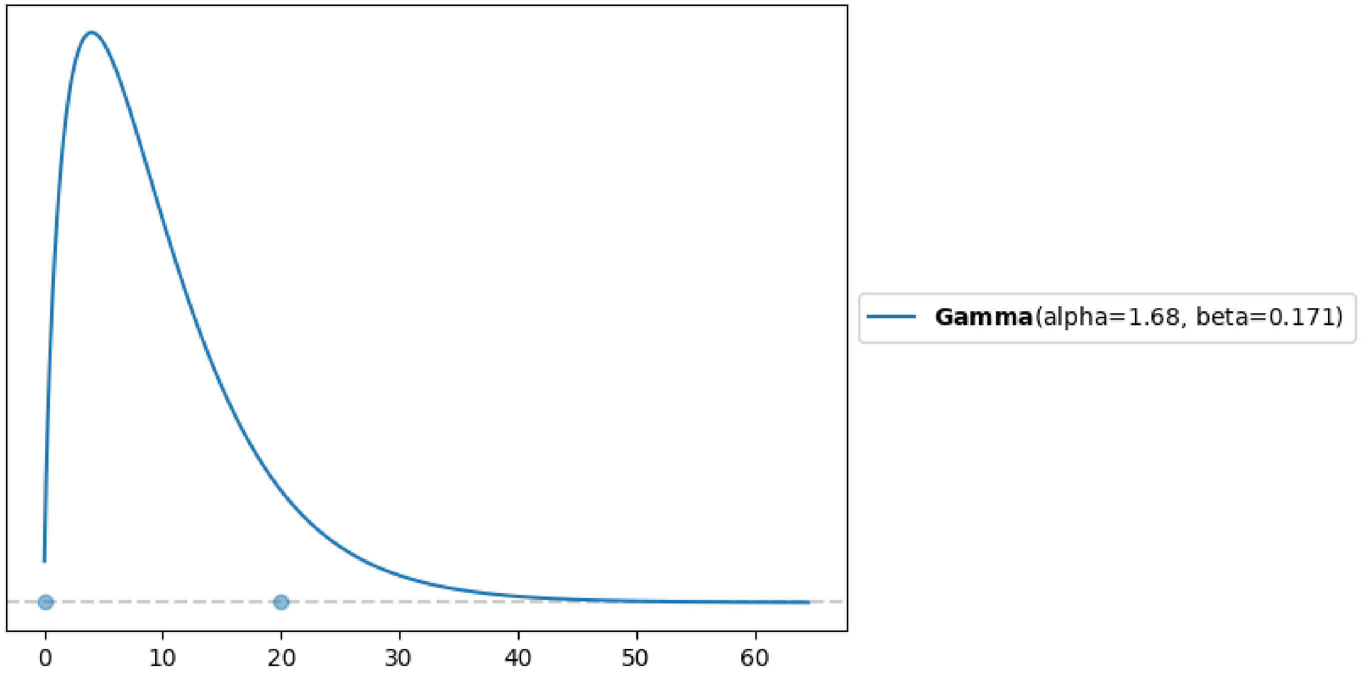
Grid Method

- In the past we looked at the question: How many emails do you receive each day?
- Likelihood:
- For n iid observations:
  - $P(data|\lambda) = \prod_{i=1}^{n} \frac{\lambda^{k_i} e^{-\lambda}}{k_i!} = \frac{\lambda^{\sum_{i=1}^{n} ki} e^{-n\lambda}}{\prod_{i=1}^{n} k_i!}$
  - $S = \sum_{i=1}^{n} ki$ (total number of observed events)
  - $P(data|\lambda) = \prod_{i=1}^{n} \frac{\lambda^{k_i} e^{-\lambda}}{k_i!} = \frac{\lambda^{S} e^{-n\lambda}}{\prod_{i=1}^{n} k_i!}$

Grid Method

- Prior:

- $P(\lambda) \sim Gamma(\alpha, \beta) = \frac{\beta^{\alpha} \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)}$



Grid Method

- Posterior:
- $P(\lambda|\text{data}) \propto P(\text{data}|\lambda)P(\lambda)$
- $P(\lambda|\text{data}) \sim Gamma(\alpha + S, n + \beta)$
  - S = total number of emails received
  - n = Number of students
- We showed this analytically.
- Let’s compare this to the results using the grid method.

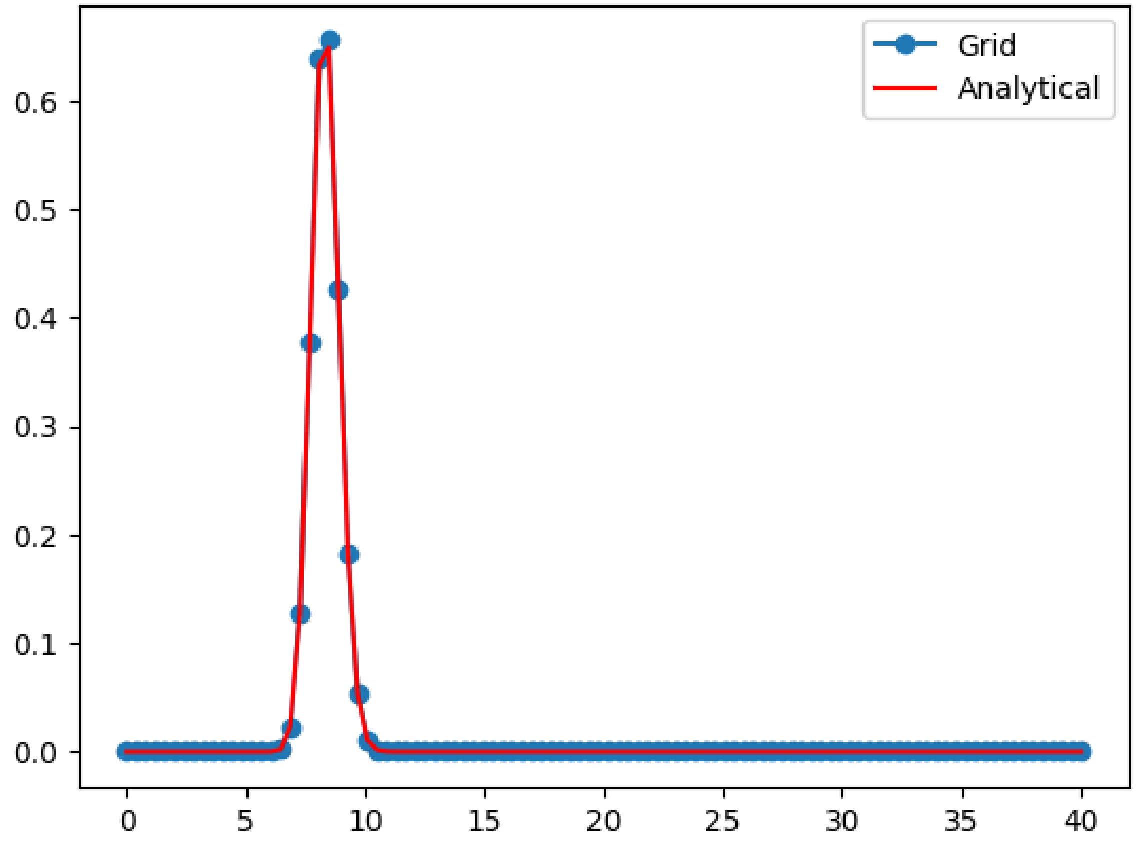
Grid Method

Steps:

1. Define a reasonable interval for the parameter (the prior should give you a hint).
   - Let’s place 100 points between 0-40.
2. Place a grid of points (generally equidistant) on that interval.
3. For each point in the grid, multiply the likelihood and the prior.

Grid Method

- We can see the overlap between the grid method and the analytical results.
- We wasted a lot of our grid sampling values below 5 and above 10.



Grid Method

- Disadvantages:
  - We focus equally on all areas of the grid, and not on high probability areas.
  - Curse of dimensionality

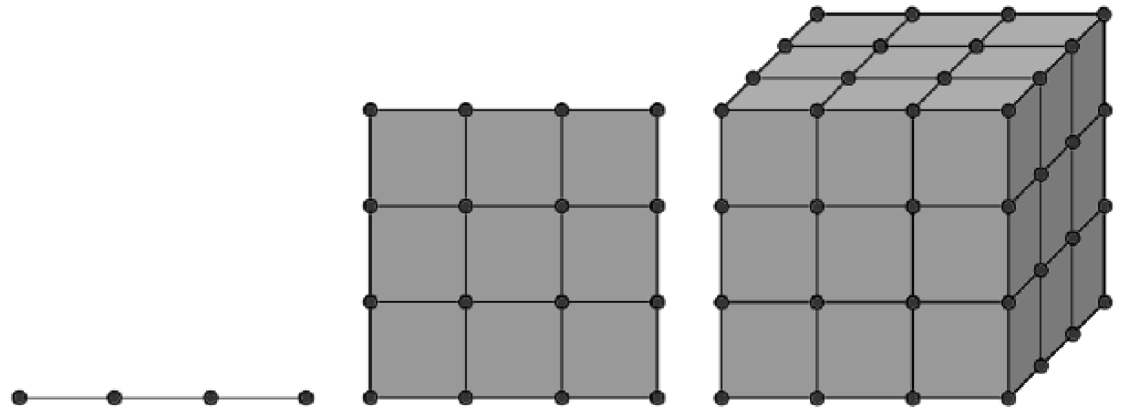


Figure 10.2: A grid with the same resolution in 1, 2, and 3 dimensions

Markovian Chain Monte Carlo Methods (MCMC)

- Allow us to get samples from the posterior distribution as long as we can compute the likelihood and the prior pointwise.
- Same condition we needed for the grid method.
- MCMC methods can efficiently sample from higher-probability regions in very high dimensions.
- MCMC methods visit each region of the parameter space following their relative probabilities.
  - If the probability of region A is twice that of region B, we will obtain twice as many samples from A as we will from B.
- Even if we are not capable of computing the whole posterior analytically, we could use MCMC methods to take samples from it.
- MCMC will give us samples from the correct distribution asymptotically (for an infinite number of samples).

Markovian Chain Monte Carlo Methods (MCMC)

- **Monte Carlo methods:**
  - A broad family of algorithms that use random sampling to compute or simulate a given process.

- **Markovian methods**:
  - Techniques or models based on the Markov property, which is the idea that the future state of a system depends only on its current state, and not on its previous history.
  - Future outcomes are independent of past events given the present state.

Markovian Chain Monte Carlo Methods (MCMC)

- **Markov Chain:**
  - A mathematical object that consists of a sequence of states and a set of transition probabilities that describe how to move among the states.
  - Simple example: flip a coin and if you get heads take a step to the right, otherwise step to the left.
  - A chain is Markovian if the probability of moving to any other state depends only on the current state.

Metropolis-Hastings

- Enables us to obtain samples from any probability distribution given that we can compute at least a value proportional to it.
  - Thus ignoring the normalization factor.
  - Useful since often the harder part is to compute the normalization factor.

Metropolis-Hastings

- We want to estimate the depth of the bottom of the lake, and find the deepest point.
  - Lake is muddy so we can’t see.

1. Take a boat and a stick.
2. Choose a random place in the lake and move the boat there.
3. Use the stick to measure the depth of the lake.
4. Move the boat to another point and take a new measurement.

Metropolis-Hastings

5. Compare the two measurements:
   - If the new spot is deeper than the old one -> write it down.
   - If the old spot was deeper -> either accept or reject the new spot.
     - Accept = write down the new spot.
     - Reject = go back to the old spot and write it down again.

6. Move to another point and repeat again and again.

Metropolis-Hastings

- What does this give us?
  - An estimate of the curvature of the bottom of the lake.
    - An analogy for the posterior.
  - The deepest point.
    - An analogy for the mode of the posterior.

Metropolis-Hastings

1. Choose an initial value for our parameter, x_i.

2. Write it down.

3. Suggest a new value for our parameter, x_{i+1}.

How do we choose x_{i+1}?

- We use a **proposal distribution q**, such as a normal or uniform distribution.

- We sample from $q(x_{i+1}|x_i)$

Metropolis-Hastings

4. Compute the probability of accepting a new parameter value by using the **Metropolis-Hastings criteria:**

$$p_a(x_{i+1}|x_i) = \min(1, \frac{p(x_{i+1})\, q(x_i|x_{i+1})}{p(x_i)\, q(x_{i+1}|x_i)})$$

5. If $p_a(x_{i+1}|x_i)$ > random number from a continuous uniform distribution ranging [0, 1]: Accept the new parameter.
6. Otherwise – reject and stay where we are.
7. Repeat again and again until we have enough samples.

Metropolis-Hastings

Metropolis-Hastings criteria: $p_a(x_{i+1}|x_i) = \min(1, \frac{p(x_{i+1})\, q(x_i|x_{i+1})}{p(x_i)\, q(x_{i+1}|x_i)})$

- The proposal distribution, q, is not part of the model. It is a component of the MCMC algorithm.

- If q is symmetric: $p_a(x_{i+1}|x_i) = \min(1, \frac{p(x_{i+1})}{p(x_i)})$

Metropolis-Hastings

- If q is symmetric: $p_a(x_{i+1}|x_i) = \min(1, \frac{p(x_{i+1})}{p(x_i)})$
- If x_{i+1} is a more probable state ($p(x_{i+1}) > p(x_i)$):
    - We for sure accept the new state.
    - Because then we get a probability of 1.
- If $p(x_{i+1}) < p(x_i)$:
    - We don't for sure move.
    - We move with a probability of $\frac{p(x_{i+1})}{p(x_i)} < 1$.

Metropolis-Hastings

- If $p(x_{i+1}) < p(x_i)$:
  - We move with a probability of $\frac{p(x_{i+1})}{p(x_i)} < 1.$

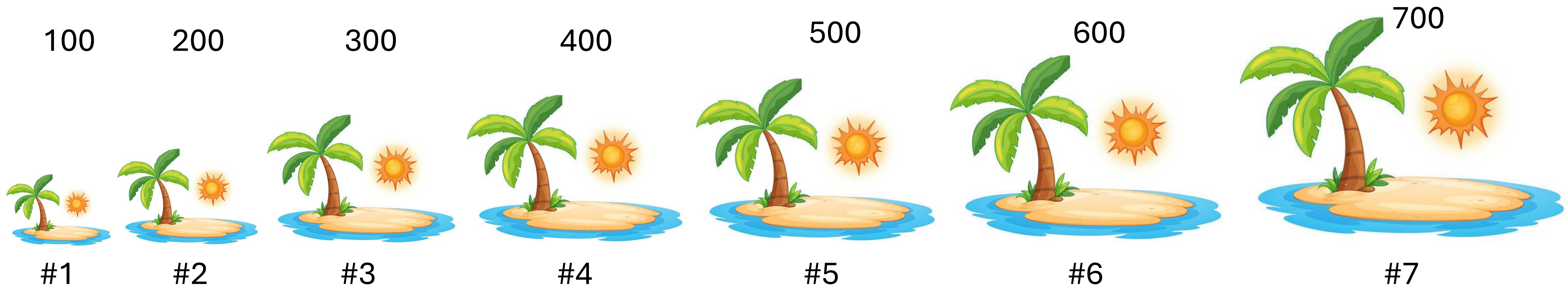
How do we quantify this?

- Choose a random number between 0-1 from a uniform distribution.

- If $\frac{p(x_{i+1})}{p(x_i)} = 0.8$
- and our random number is 0.7 (there is a 70% chance of this)
- Then we move, as we exceeded this this value (the metropolis hastings criteria).

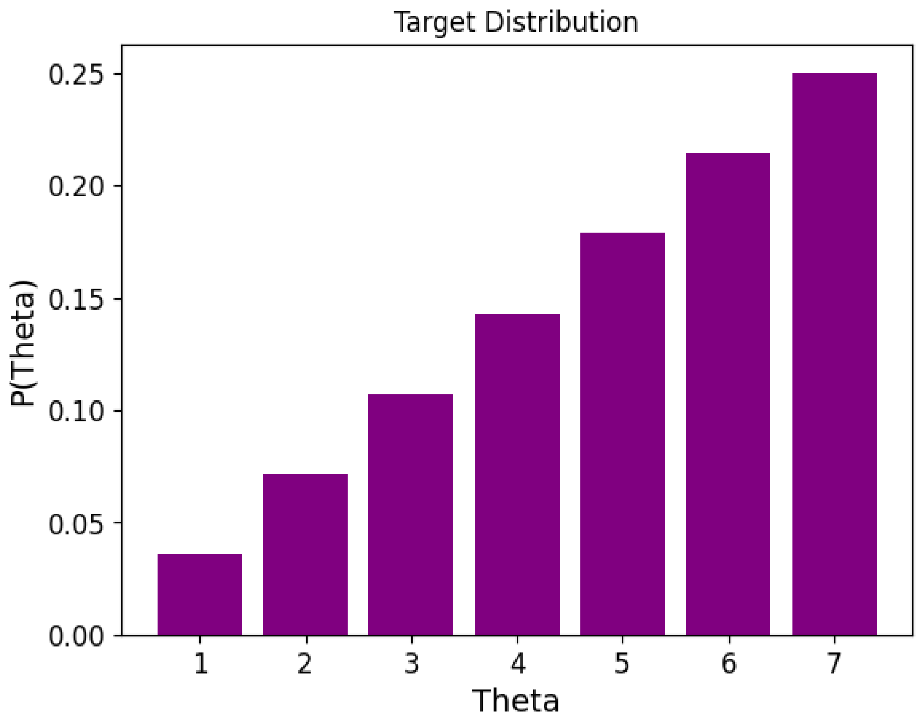
Metropolis-Hastings

- The island example.
  - We have 7 islands and we move between them.
  - The probability we are trying to sample from is $P(\theta) \propto \theta$
  - θ is the relative number of people on each island.



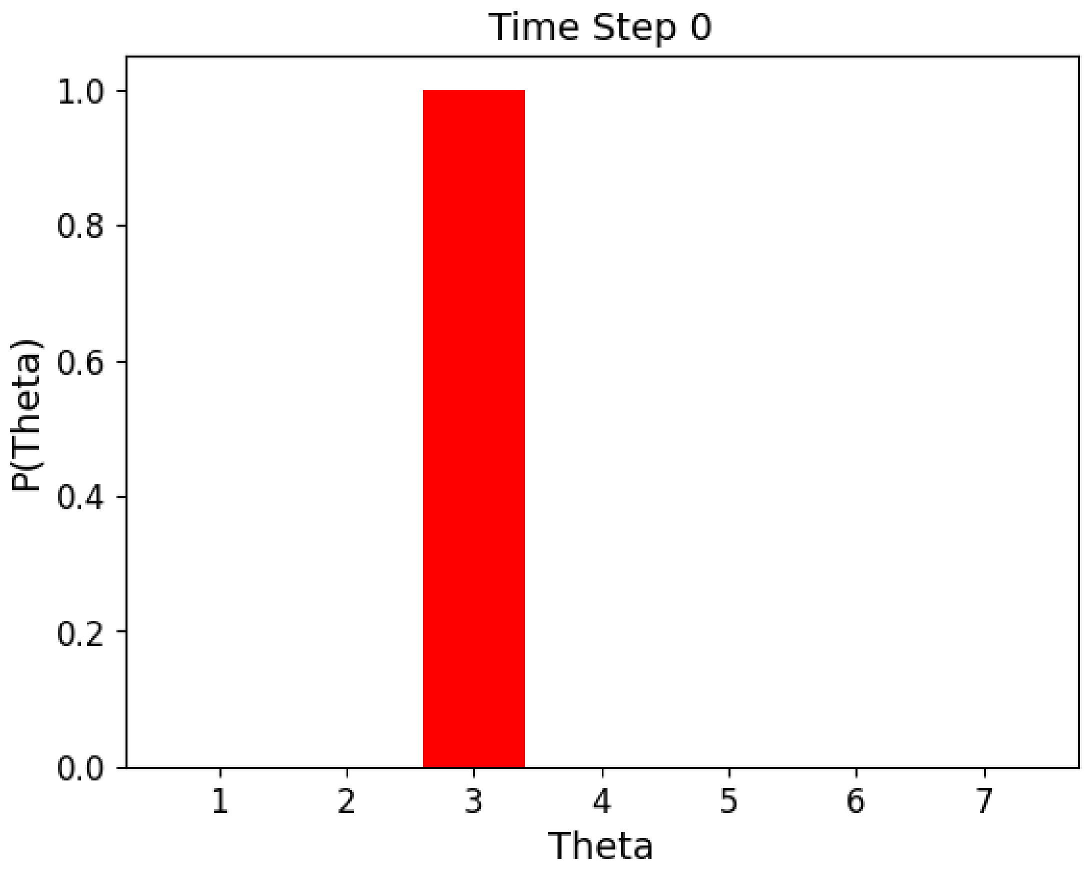
Metropolis-Hastings

- The island example.
  - The probability we are trying to sample from is $P(\theta) \propto \theta$
  - θ is the relative number of people on each island.



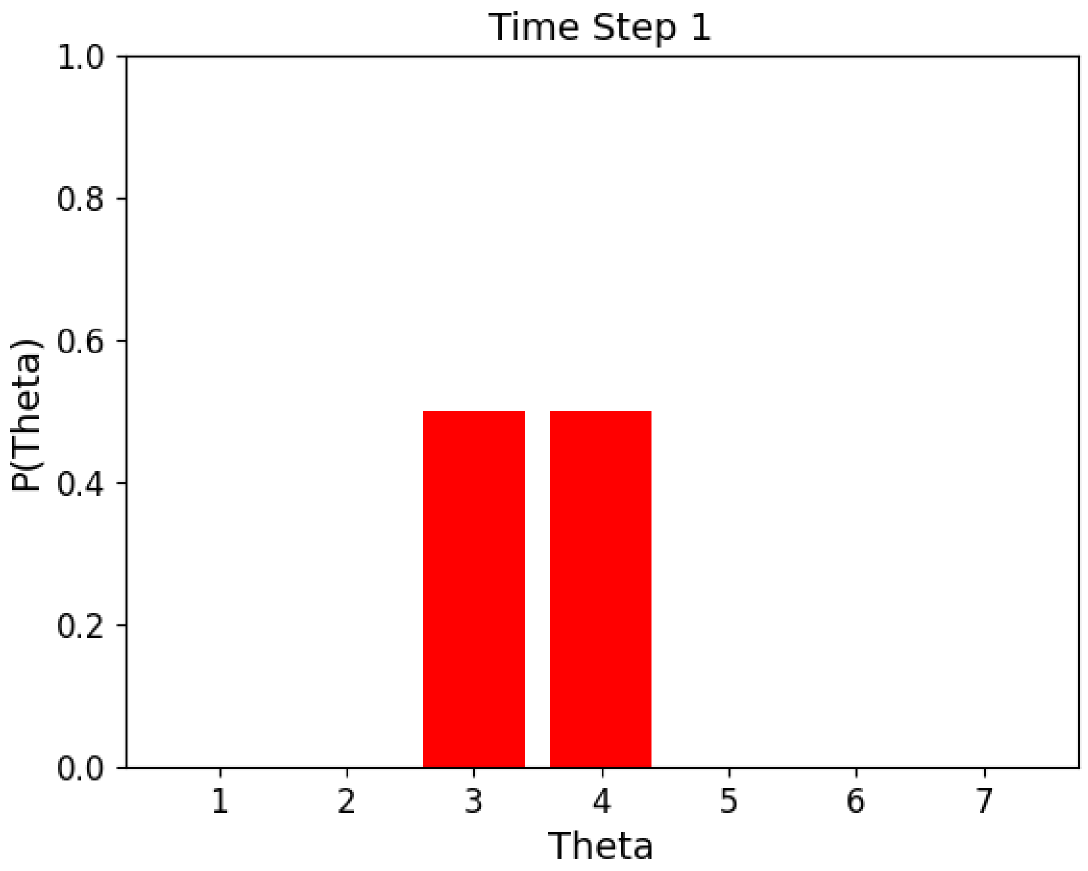
Metropolis-Hastings

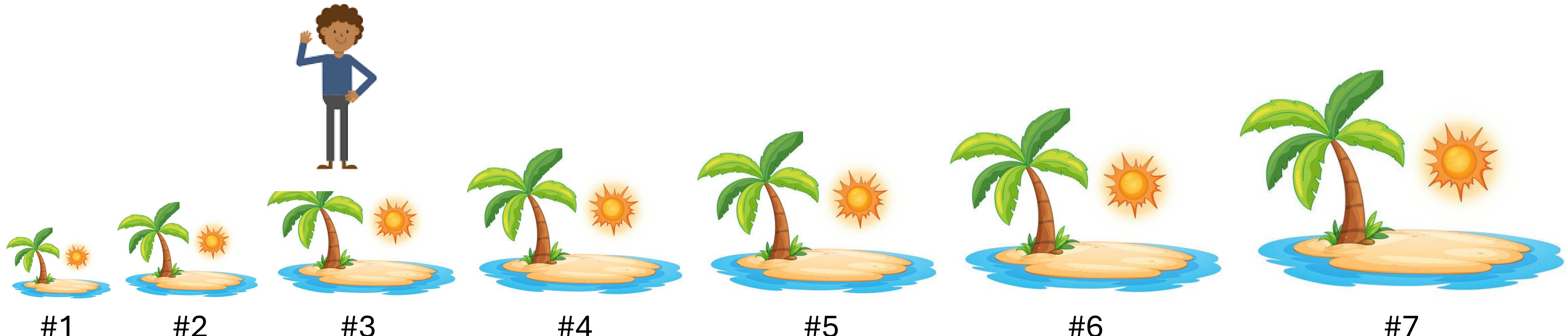
- Start at a random island
  - We chose island 3
- So we write it down.



Metropolis-Hastings

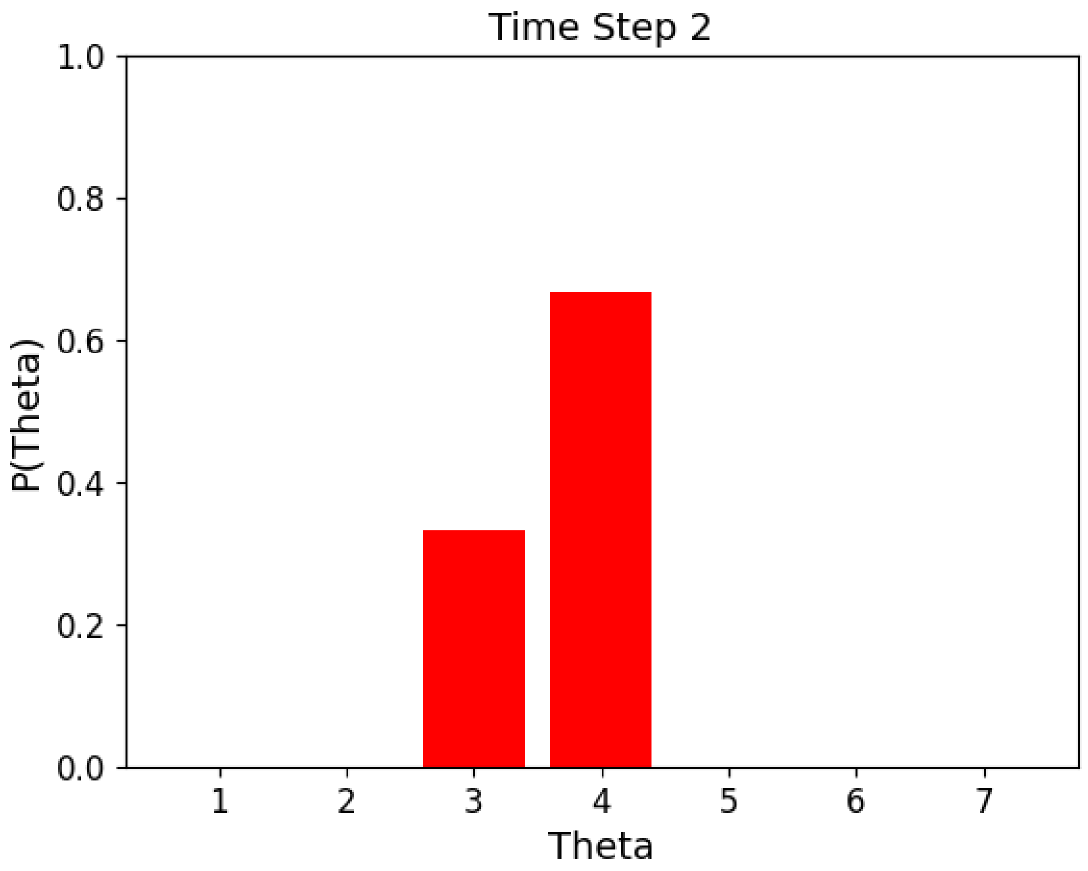
- Let's try and move an island.
- We choose randomly and get island 4.
- $p_a(\theta_{i+1}|\theta_i) = \min\left(1, \frac{p(\theta_{i+1})}{p(\theta_i)}\right) = \min\left(1, \frac{4}{3}\right) = 1$



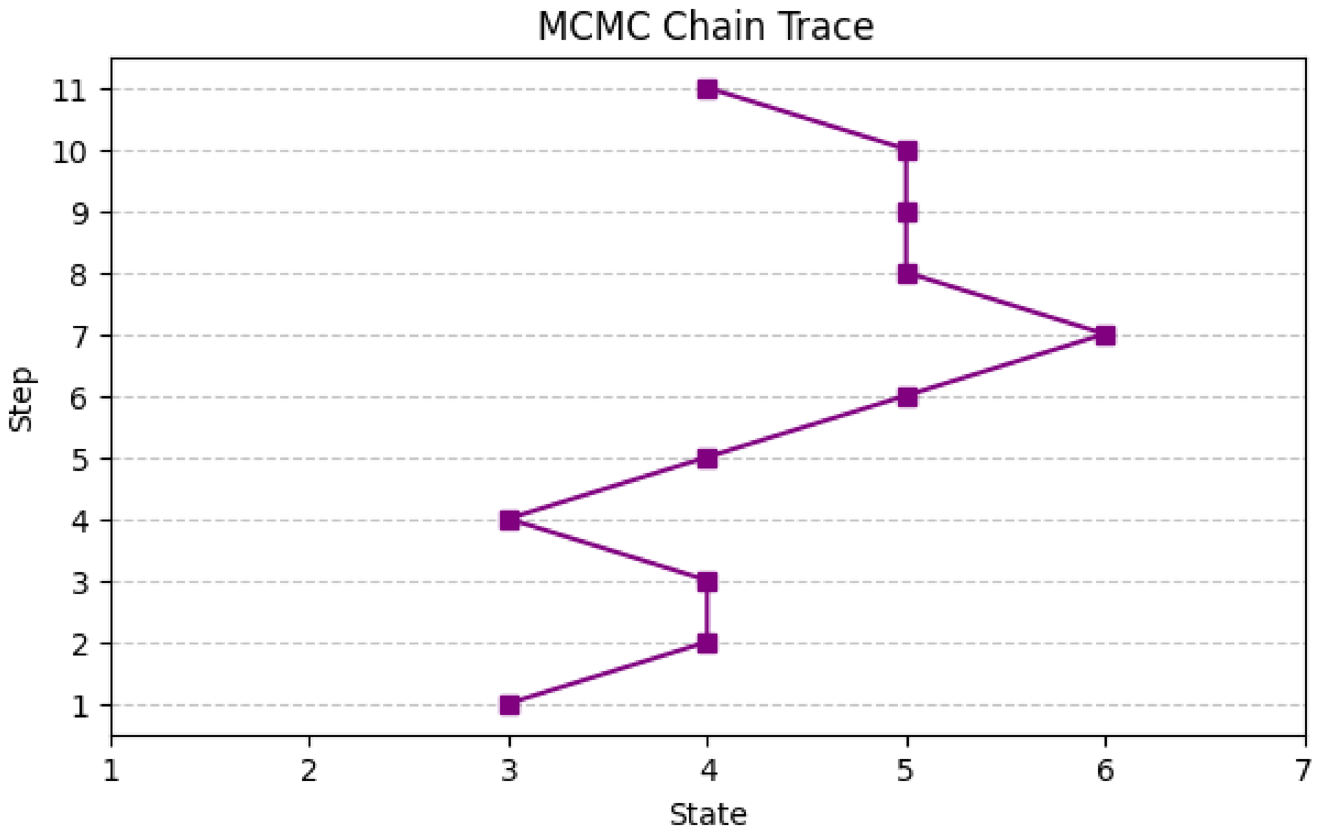


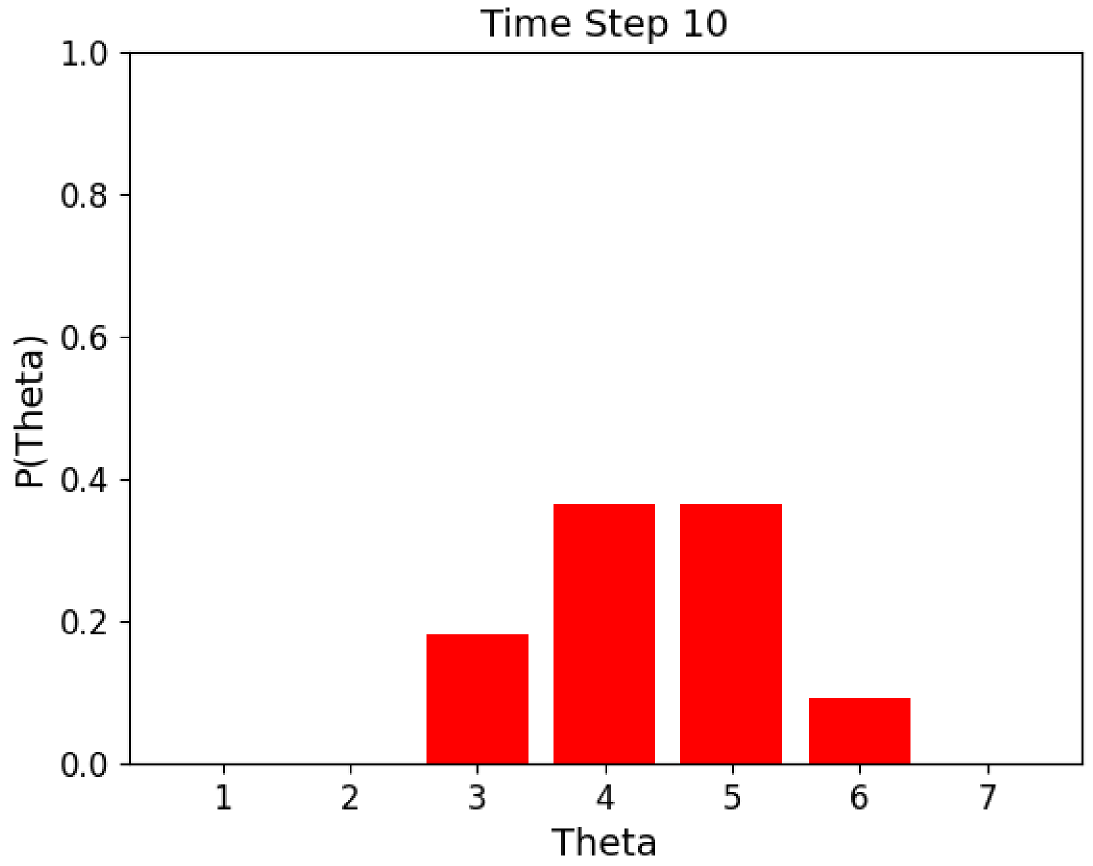
Metropolis-Hastings

- Try to move back to 3
- $p_a(\theta_{i+1}|\theta_i) = \min\left(1, \frac{p(\theta_{i+1})}{p(\theta_i)}\right) \min\left(1, \frac{3}{4}\right) = \frac{3}{4}$
- Choose a random number (0.91)
- 0.75<0.91 – don't move back to 3.
- Write 4 again.

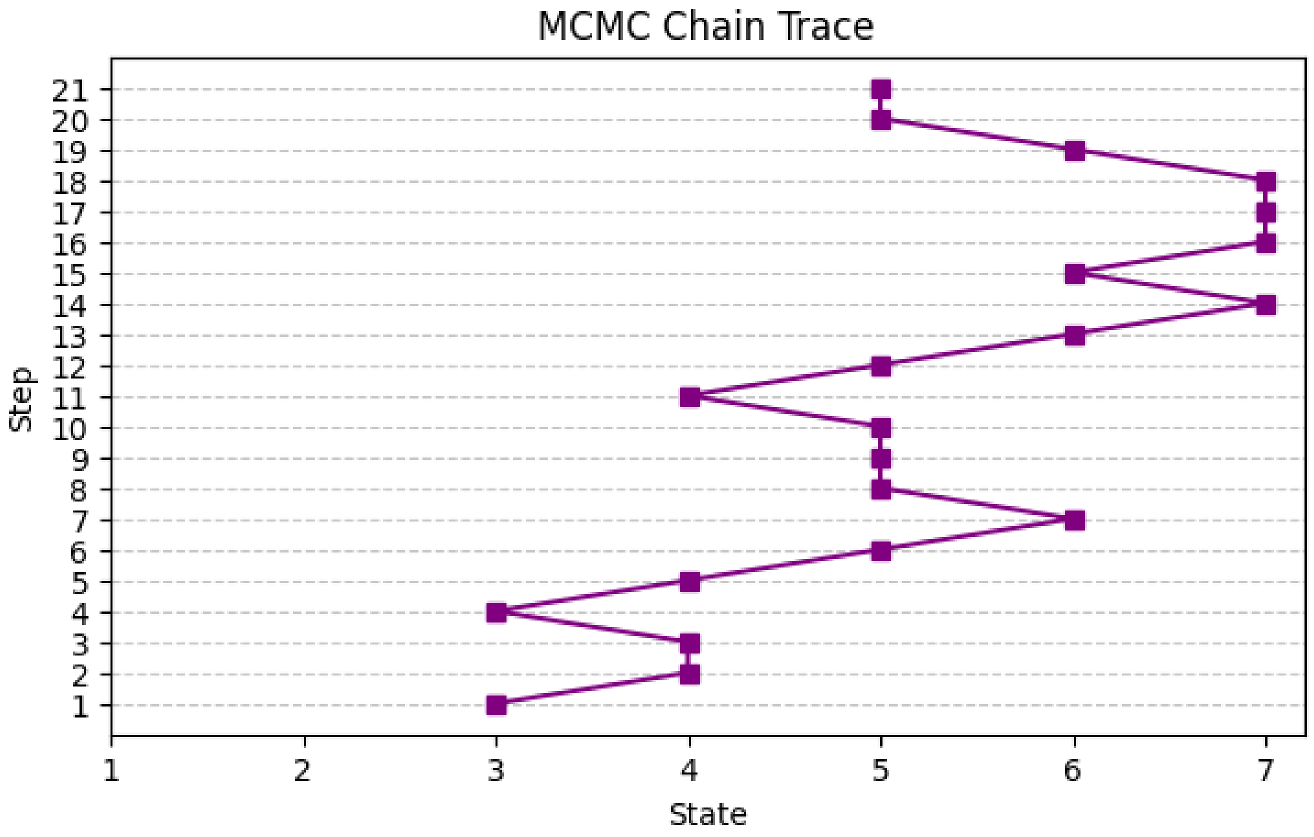


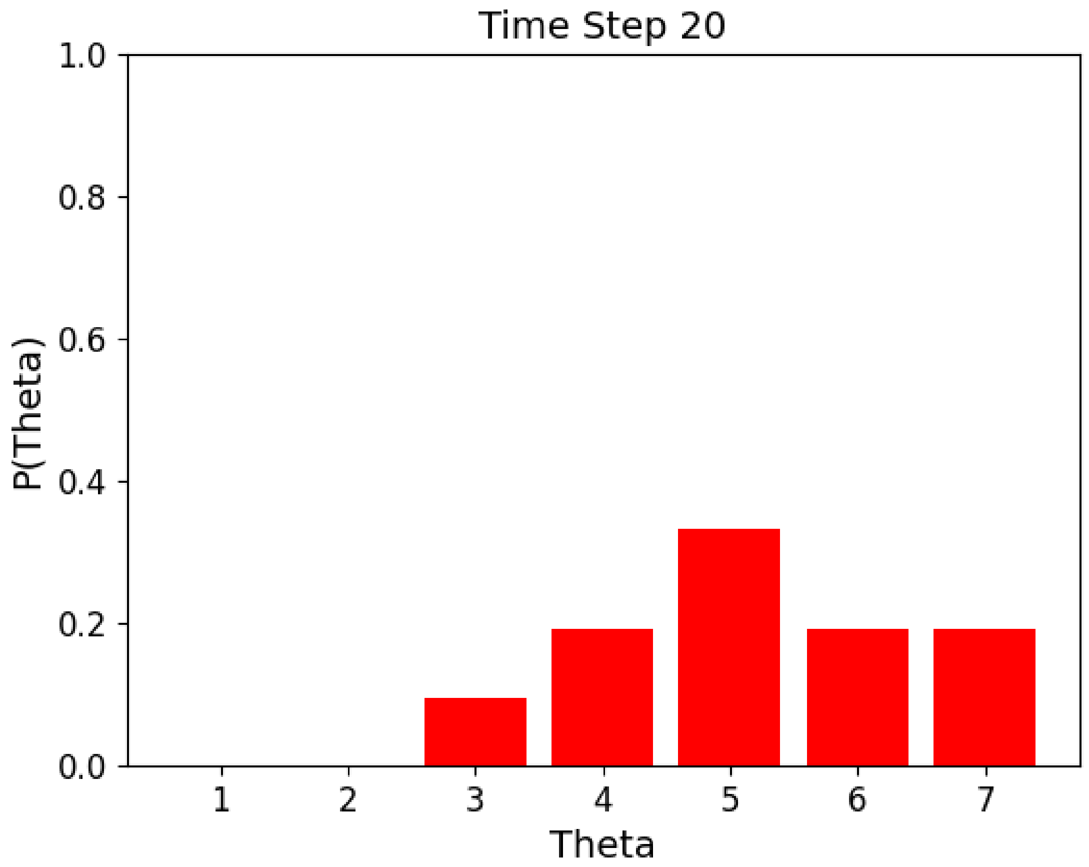
Metropolis-Hastings





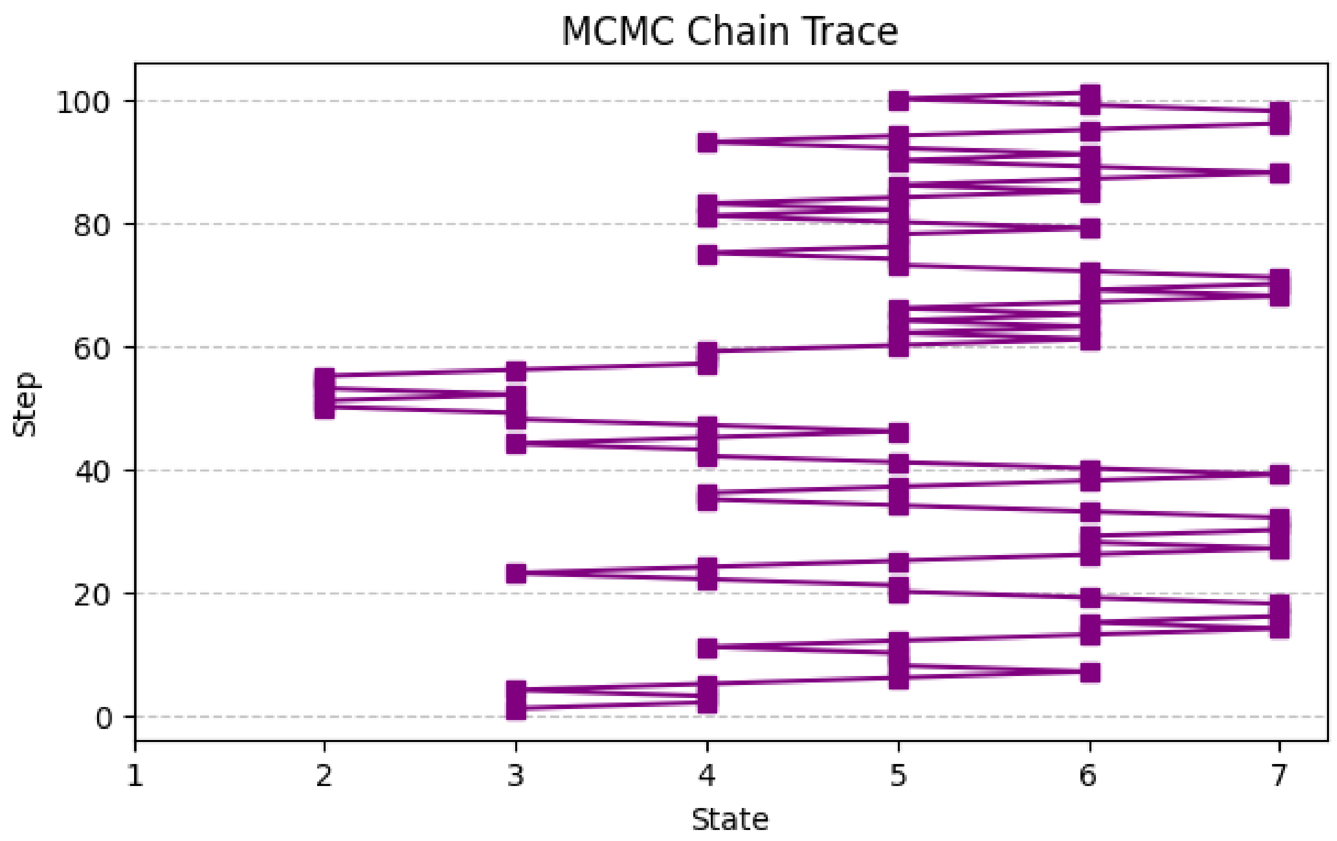
Metropolis-Hastings

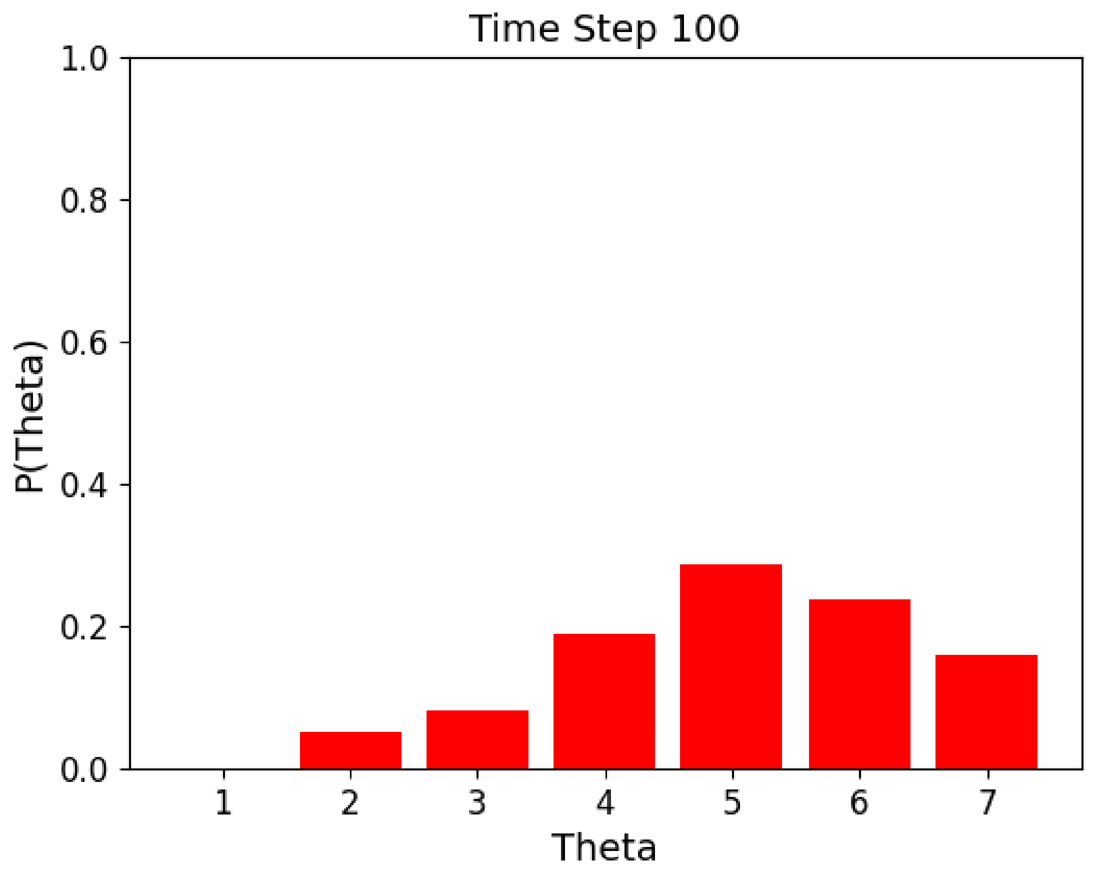




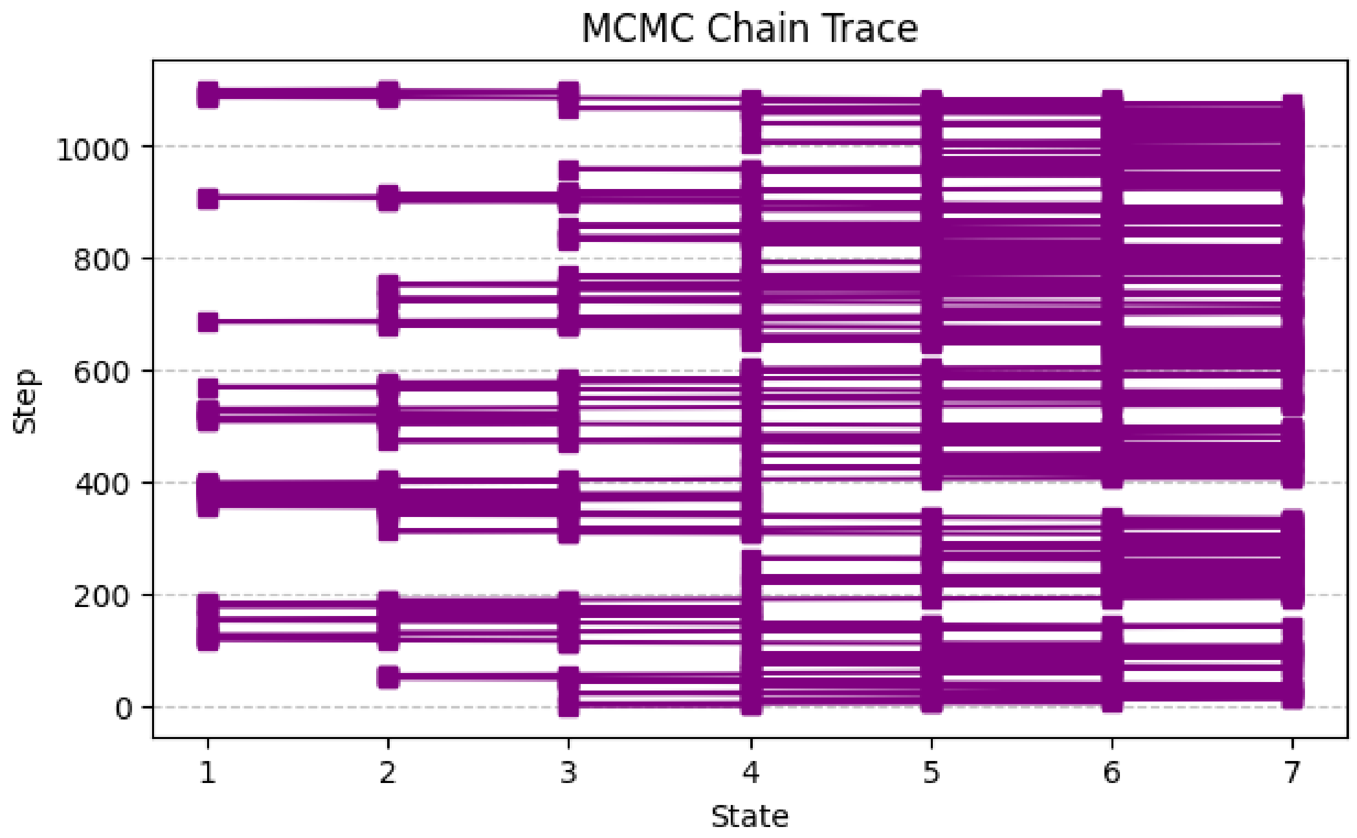
#1 #2 #3 #4 #5 #6 #7

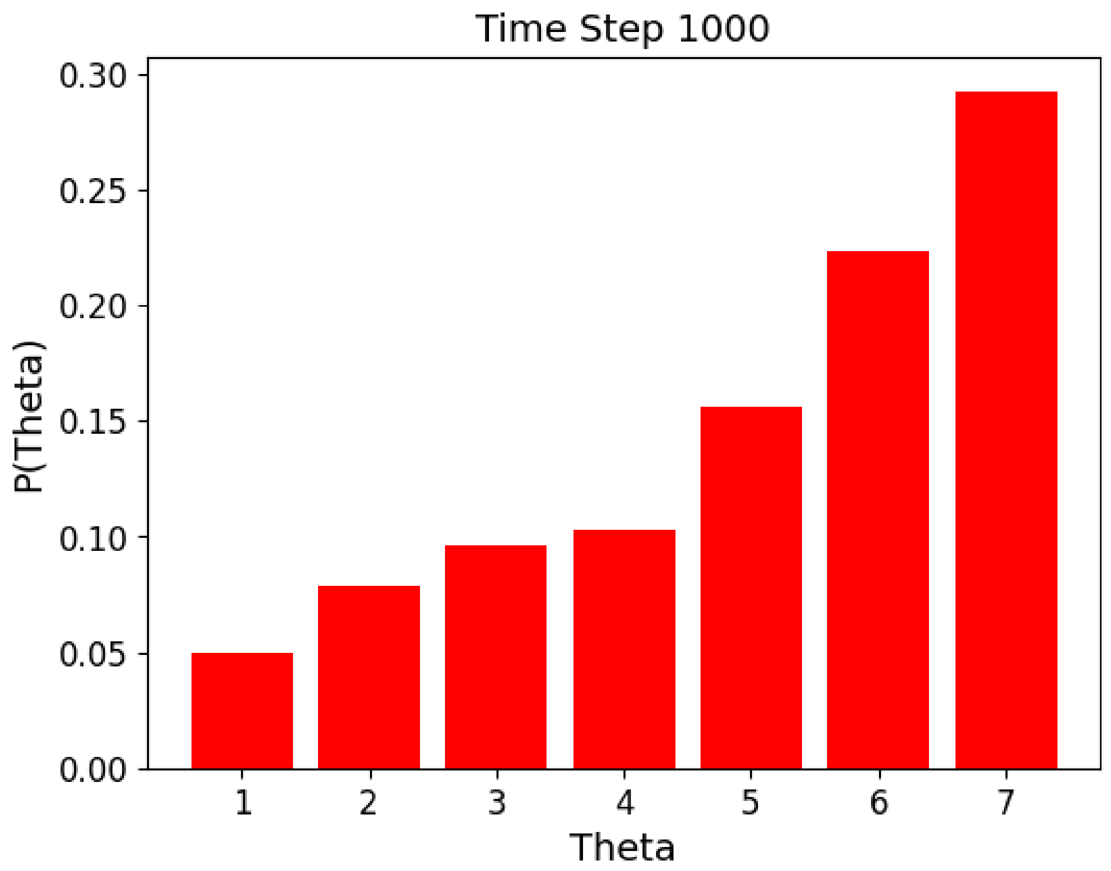
Metropolis-Hastings



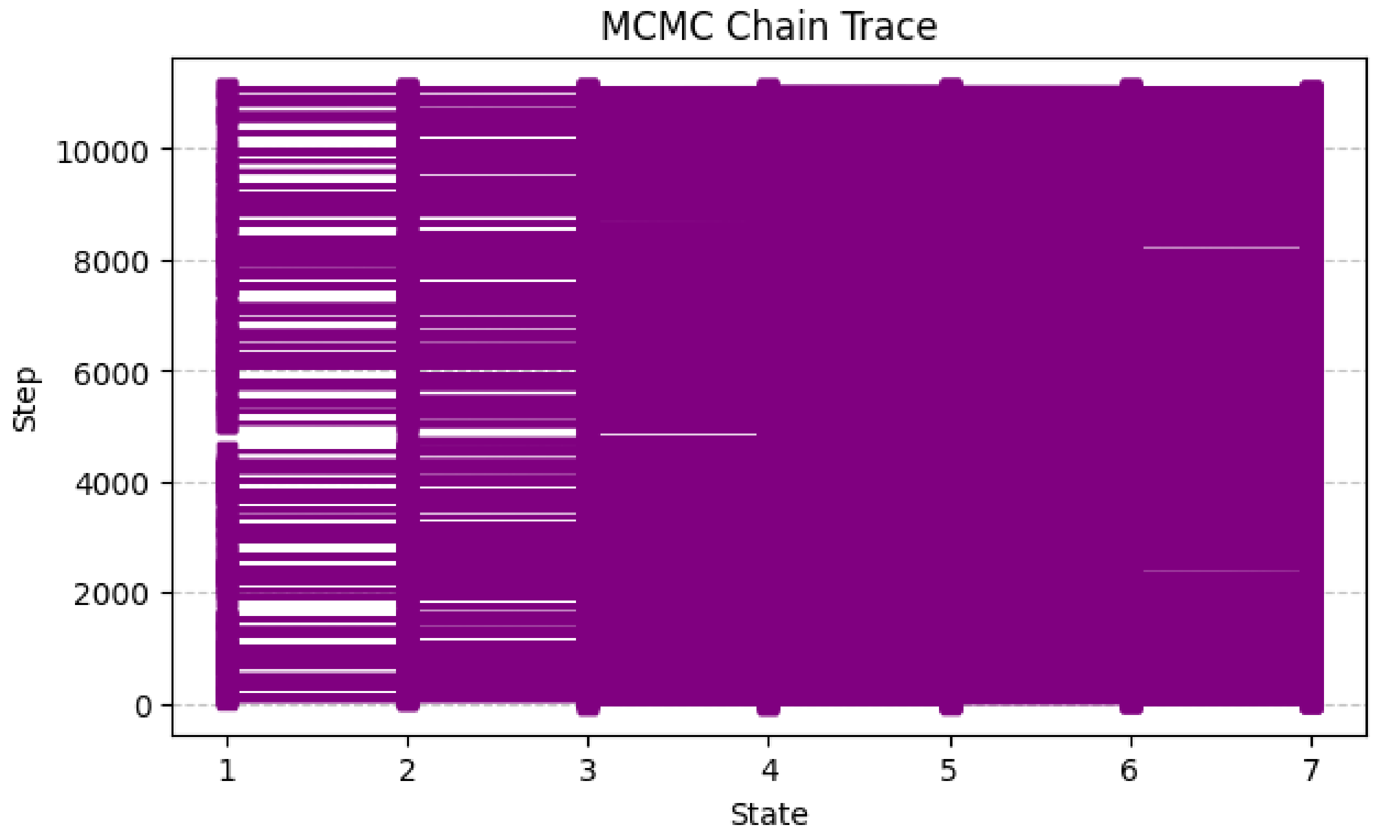


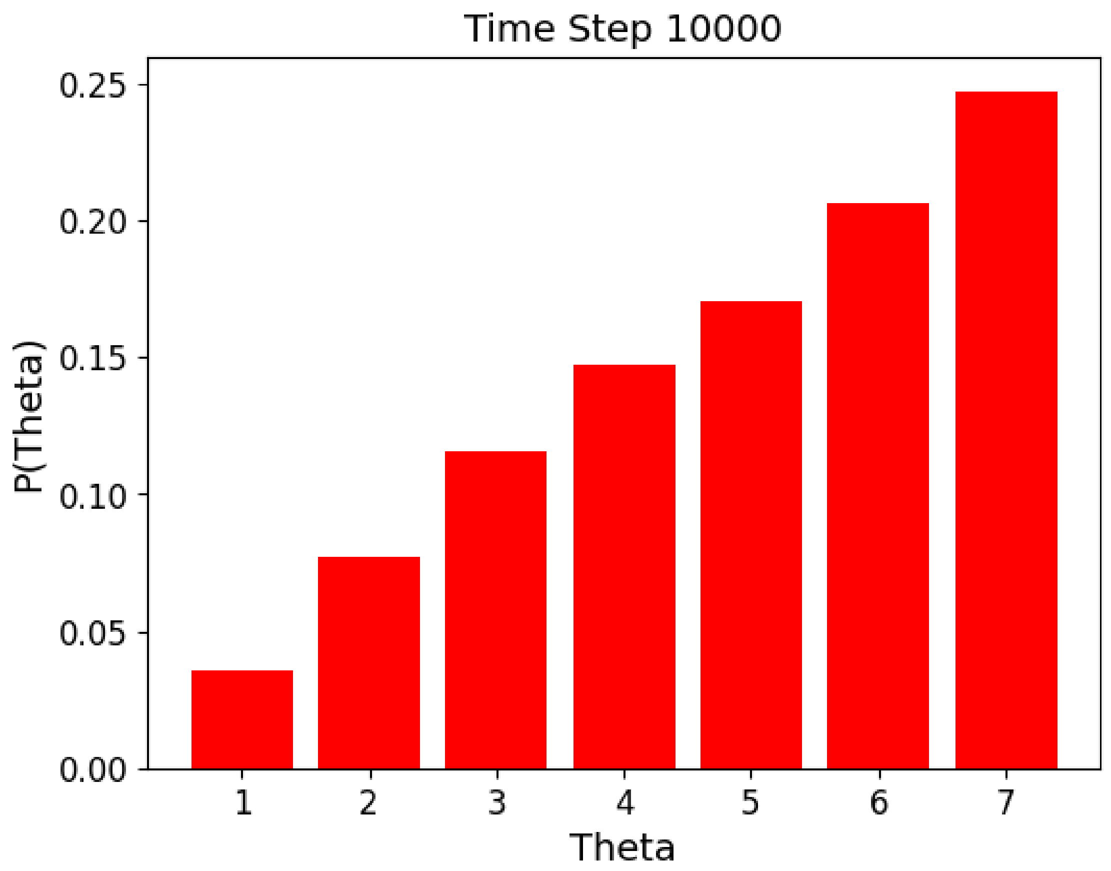
Metropolis-Hastings





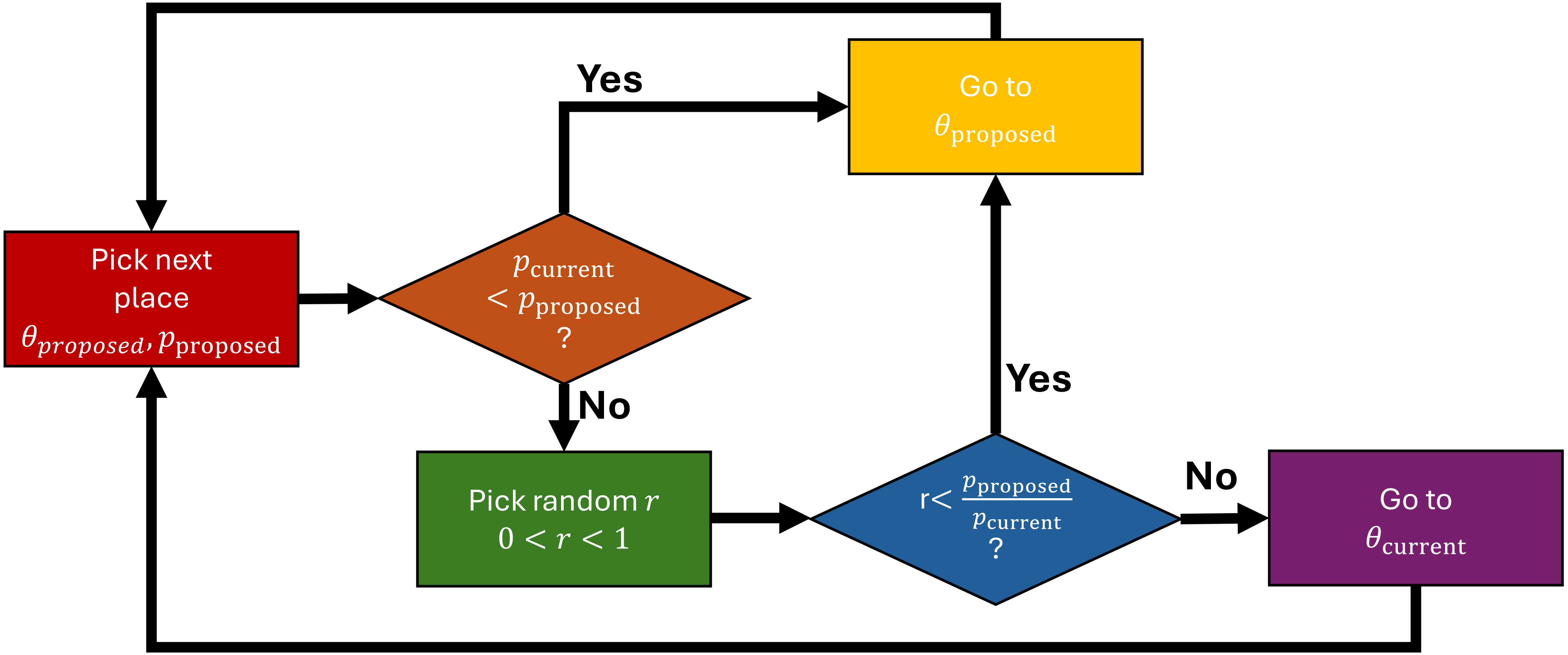
Metropolis-Hastings





#1 #2 #3 #4 #5 #6 #7

Metropolis-Hastings



Pick next place $\theta_{proposed}, p_{\text{proposed}}$
$p_{\text{current}} < p_{\text{proposed}}$?
Yes
No
Go to θ_{proposed}
Pick random r $0 < r < 1$
r< $\frac{p_{\text{proposed}}}{p_{\text{current}}}$?
Yes
No
Go to θ_{current}

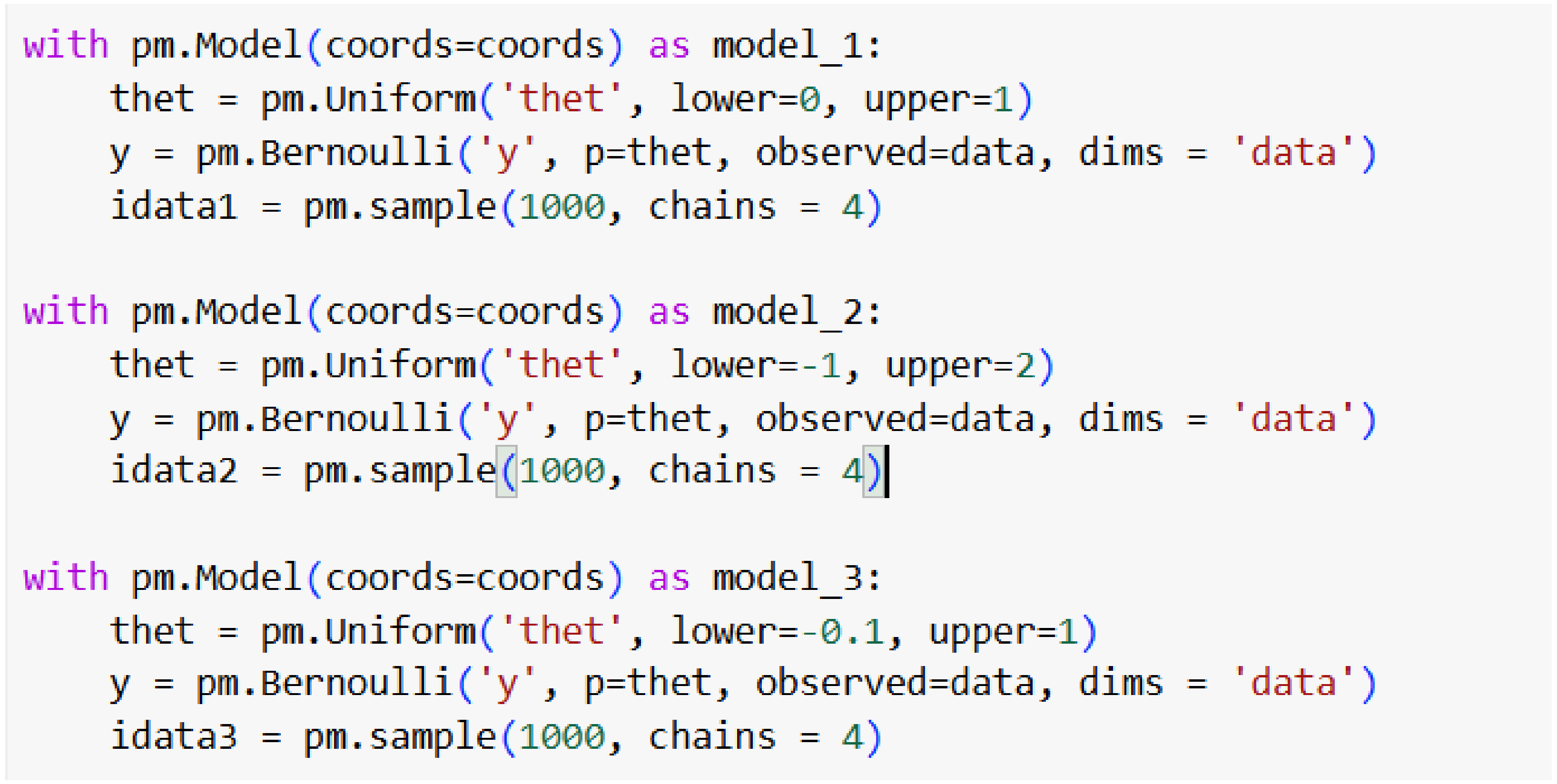
Metropolis-Hastings

- How do we choose a proposal function?
  - The proposal function has to be scaled appropriately.
  - If we have a very broad proposal for a narrow posterior, all the proposals are to a place with no probability and we never accept a new state.
  - If your proposal is very narrow, get stuck near the starting point and explore that excessively.
  - We often use a normal distribution N(0, sigma) and tune the sigma.
  - We can run the algorithm with an acceptance rate in mind (~70%-80%).
  - Look how many of the first N steps were accepted.
  - If the proposal is too narrow – we'll almost always accept.
    - We'll see we surpassed the acceptance rate and make the proposal wider.
  - If the proposal is too wide – we'll rarely accept.
    - We'll see we did not reach the acceptance rate and make the proposal narrower.

Diagnostics

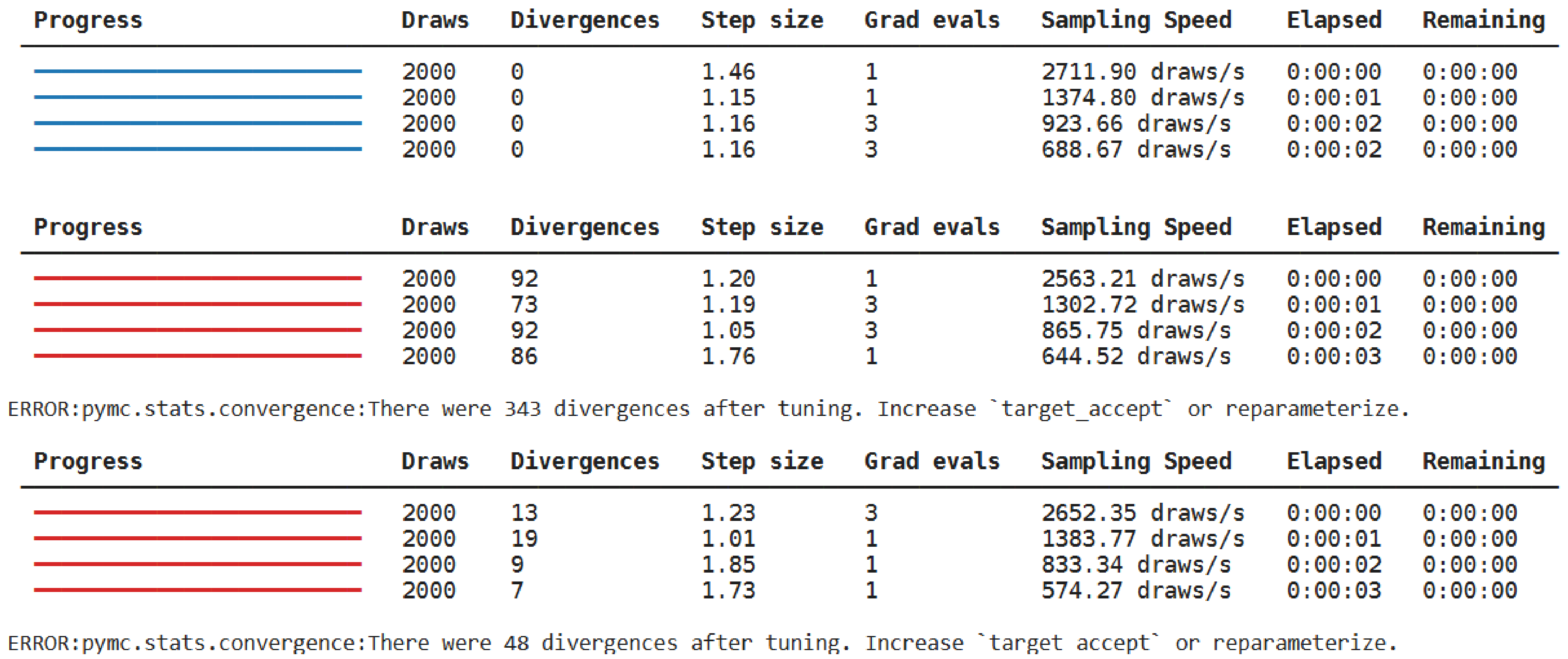
- Using numerical methods means we are approximating the posterior with a finite number of samples.
  - Do we have a valid sample?
- We will use tests to see if the sampling algorithm worked.
  - These tests can spot problems with our samples.
  - They cannot prove that we have the correct distribution.
  - They can also provide evidence that the samples seem reasonable.

Three Different Priors For the Coin Flip



```
with pm.Model(coords=coords) as model_1:
    thet = pm.Uniform('thet', lower=0, upper=1)
    y = pm.Bernoulli('y', p=thet, observed=data, dims = 'data')
    idata1 = pm.sample(1000, chains = 4)

with pm.Model(coords=coords) as model_2:
    thet = pm.Uniform('thet', lower=-1, upper=2)
    y = pm.Bernoulli('y', p=thet, observed=data, dims = 'data')
    idata2 = pm.sample(1000, chains = 4)

with pm.Model(coords=coords) as model_3:
    thet = pm.Uniform('thet', lower=-0.1, upper=1)
    y = pm.Bernoulli('y', p=thet, observed=data, dims = 'data')
    idata3 = pm.sample(1000, chains = 4)
```

Three Different Priors For the Coin Flip



```
 Progress                   Draws    Divergences   Step size   Grad evals   Sampling Speed     Elapsed   Remaining
─────────────────────────────────────────────────────────────────────────────────────────────────────────────────
 ━━━━━━━━━━━━━━━━━━━━━━━━   2000     0             1.46        1            2711.90 draws/s    0:00:00   0:00:00
 ━━━━━━━━━━━━━━━━━━━━━━━━   2000     0             1.15        1            1374.80 draws/s    0:00:01   0:00:00
 ━━━━━━━━━━━━━━━━━━━━━━━━   2000     0             1.16        3            923.66 draws/s     0:00:02   0:00:00
 ━━━━━━━━━━━━━━━━━━━━━━━━   2000     0             1.16        3            688.67 draws/s     0:00:02   0:00:00

 Progress                   Draws    Divergences   Step size   Grad evals   Sampling Speed     Elapsed   Remaining
─────────────────────────────────────────────────────────────────────────────────────────────────────────────────
 ━━━━━━━━━━━━━━━━━━━━━━━━   2000     92            1.20        1            2563.21 draws/s    0:00:00   0:00:00
 ━━━━━━━━━━━━━━━━━━━━━━━━   2000     73            1.19        3            1302.72 draws/s    0:00:01   0:00:00
 ━━━━━━━━━━━━━━━━━━━━━━━━   2000     92            1.05        3            865.75 draws/s     0:00:02   0:00:00
 ━━━━━━━━━━━━━━━━━━━━━━━━   2000     86            1.76        1            644.52 draws/s     0:00:03   0:00:00

ERROR:pymc.stats.convergence:There were 343 divergences after tuning. Increase `target_accept` or reparameterize.
 Progress                   Draws    Divergences   Step size   Grad evals   Sampling Speed     Elapsed   Remaining
─────────────────────────────────────────────────────────────────────────────────────────────────────────────────
 ━━━━━━━━━━━━━━━━━━━━━━━━   2000     13            1.23        3            2652.35 draws/s    0:00:00   0:00:00
 ━━━━━━━━━━━━━━━━━━━━━━━━   2000     19            1.01        1            1383.77 draws/s    0:00:01   0:00:00
 ━━━━━━━━━━━━━━━━━━━━━━━━   2000     9             1.85        1            833.34 draws/s     0:00:02   0:00:00
 ━━━━━━━━━━━━━━━━━━━━━━━━   2000     7             1.73        1            574.27 draws/s     0:00:03   0:00:00

ERROR:pymc.stats.convergence:There were 48 divergences after tuning. Increase `target accept` or reparameterize.
```

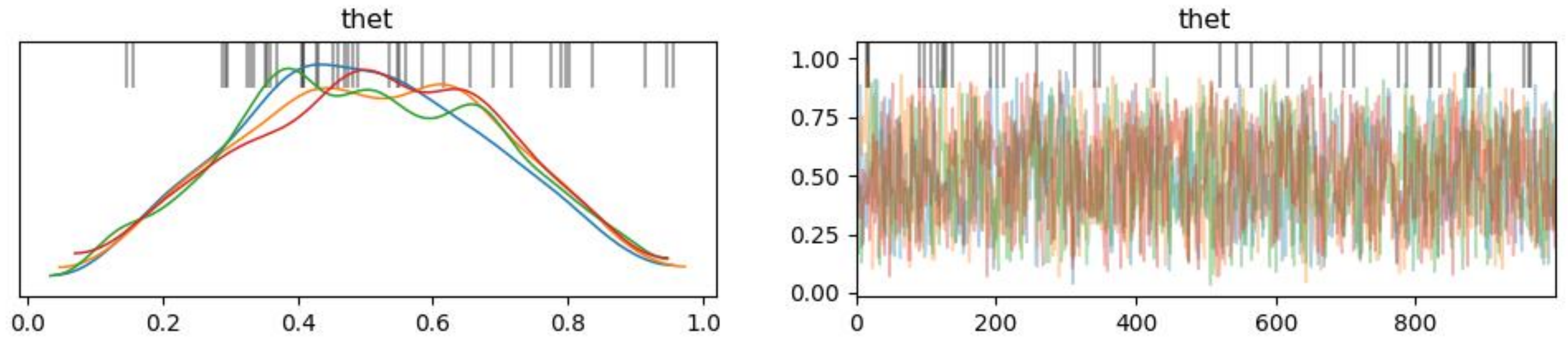
Convergence

- MCMC methods are guaranteed to converge for infinite samples.
- How do we know if we have a reasonable number of samples?
- The sampler has **converged** if the samples are stable.
    - Run the simulation multiple times and see if we get the same result every time.
    - We often used four chains.
    - This is also efficient as we can combine the samples from the different chains to have four times as many samples.

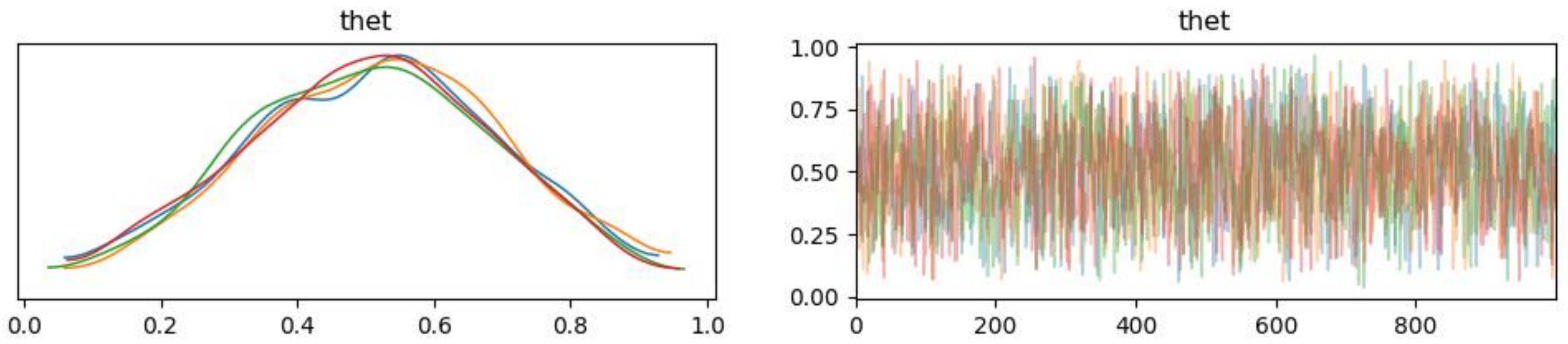
Convergence

- How do we check convergence?
    - Look at the trace plots and see if the chains look similar

Less good:

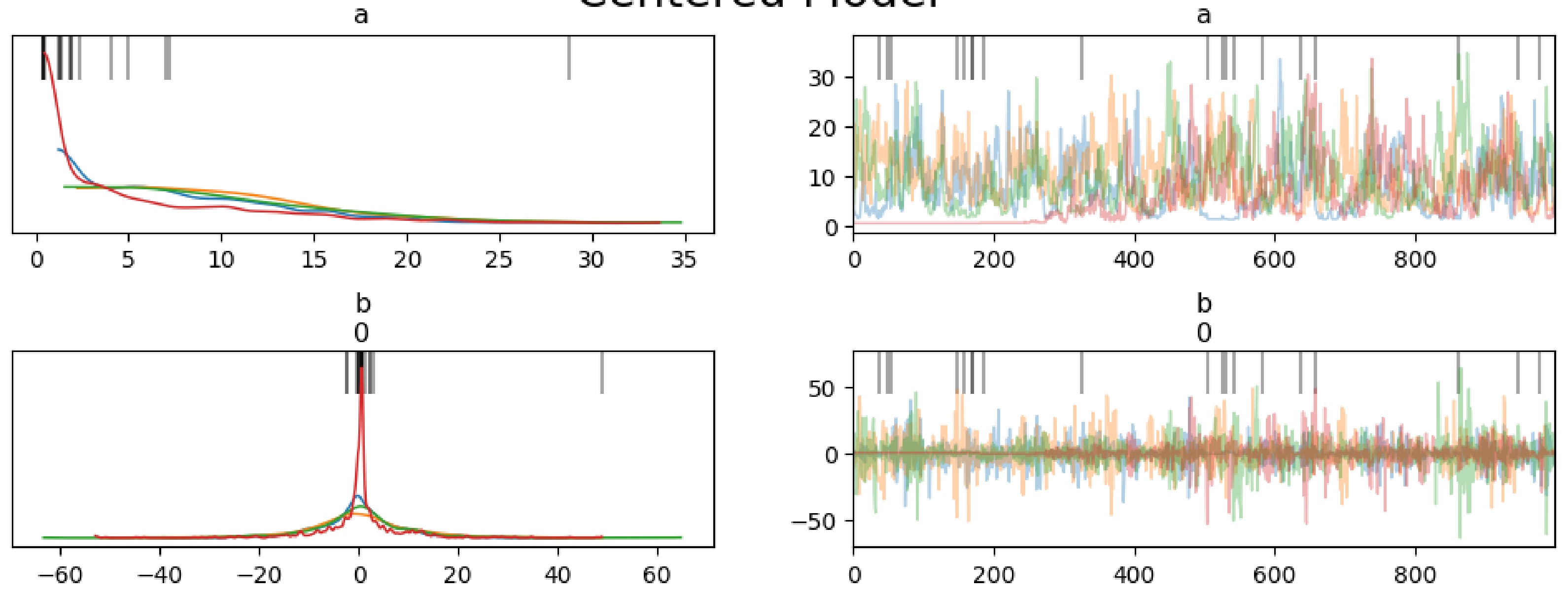


Better:



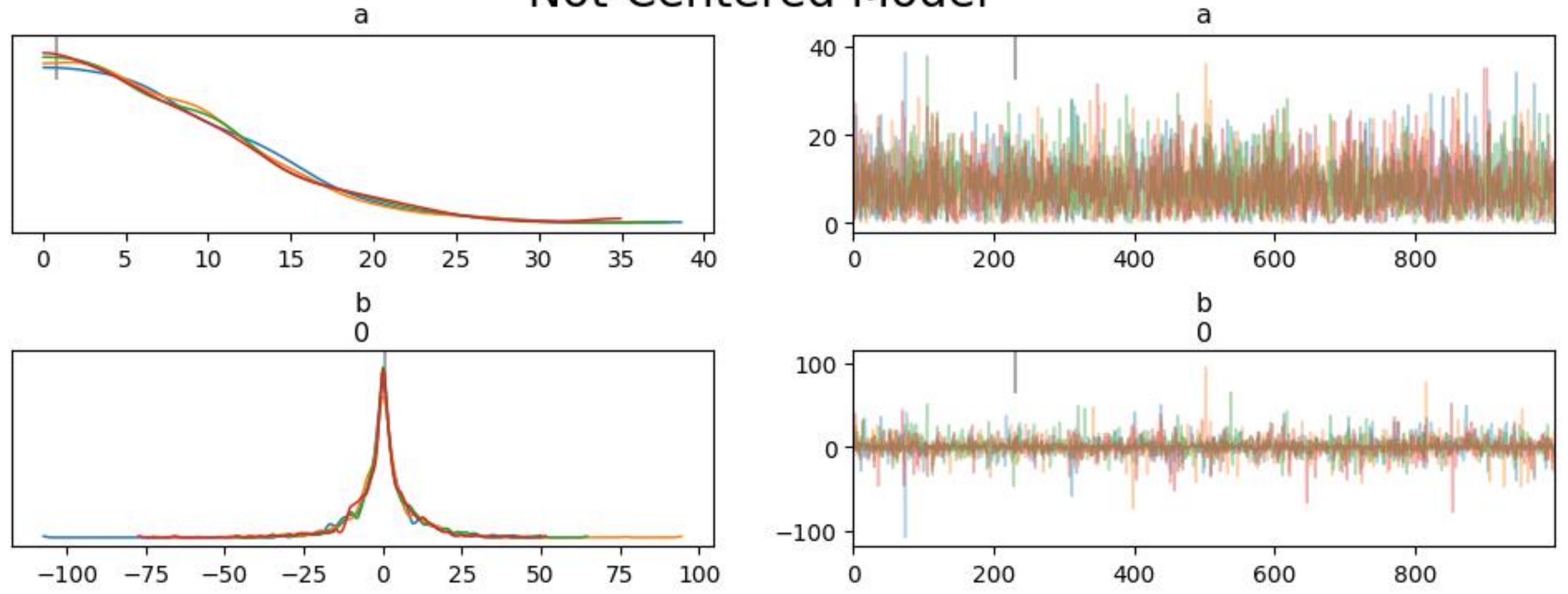
Centered Model

Less good:



Not-Centered Model

Better:



Convergence

- How do we check convergence?
    - Look at the trace plots and see if the chains look similar
        - Overlapping KDE chains
        - Trace plots that look noisy, mixed and like they explored the different values.
        - The overlapping and mixing means that even when the different chains start from different points, they all describe the same distribution.

Convergence

- How do we check convergence?
  - Rhat ($\hat{R}$)
    - Compare the variance between chains with the variance within the chains.
    - Ideally should be 1 (the variance between the chains is not more than the variance within the chains)
    - Values below 1.1.

Convergence

- How do we check convergence?
  - Look at the trace plots and see if the chains look similar
    - Overlapping KDE chains
    - Trace plots that look noisy, mixed and like they explored the different values.
  - Rhat
    - Below 1.1
  - Effective Sample Size

Convergence

- Effective Sample Size
  - How many useful draws do we have in our sample?
  - Due to autocorrelation: Number of useful draws < number of samples.
  - We need an effective sample size of at least 100 per chain (400 for 4 chains).
    - We'll use at least 500
  - The effective sample size is the number of samples, corrected for the autocorrelation.

$$N_{\text{eff}} = \frac{N_{\text{MCMC}}}{\sum_{\delta=-\infty}^{\infty} \rho(\delta)}$$

Convergence

- How do we check convergence?
  - Look at the trace plots and see if the chains look similar
    - Overlapping KDE chains
    - Trace plots that look noisy, mixed and like they explored the different values.
  - Rhat
    - Below 1.1
  - Effective Sample Size
    - At least 100 per chain
  - Monte Carlo Standard Error

Convergence

- Monte Carlo Standard Error
  - Width of the posterior divided by the square root of the effective sample size.

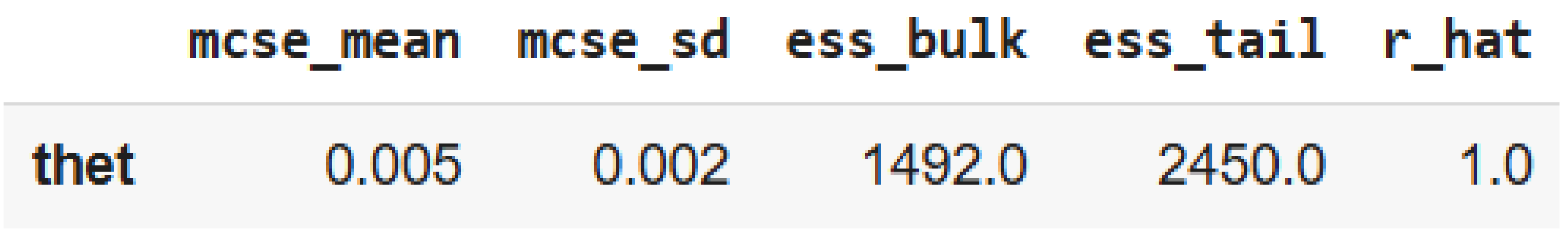
$$MCSE = \frac{\sigma_\theta}{\sqrt{N_{\text{eff}}}}$$

  - Indication of how well we are measuring the parameter.
  - Should be less than 1%-2%.

Convergence

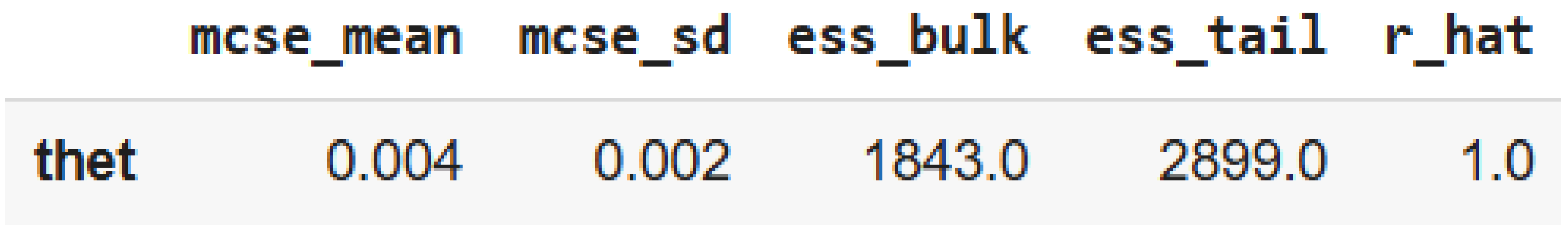
- How do we check convergence?
  - Look at the trace plots and see if the chains look similar
    - Overlapping KDE chains
    - Trace plots that look noisy, mixed and like they explored the different values.
  - Rhat
    - Below 1.1
  - Effective Sample Size
    - At least 100 per chain
  - Monte Carlo Standard Error
    - Below 1-2%.

Less good:



	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
thet	0.005	0.002	1492.0	2450.0	1.0

Better:



	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
thet	0.004	0.002	1843.0	2899.0	1.0

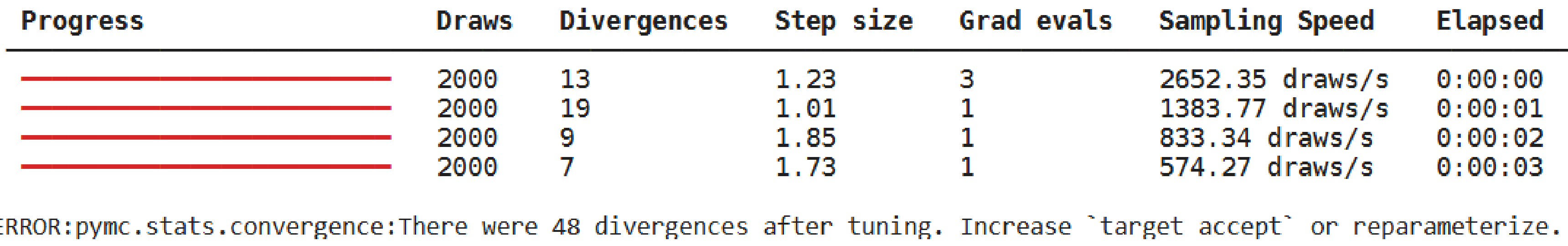
Convergence

- What do we do if there is no convergence?
    - Check our codes.
    - Increase number of samples.
    - Increase number of tuning samples.
    - Change the model
        - Try a simpler model
        - Change the priors
        - Change the parameters

Divergence

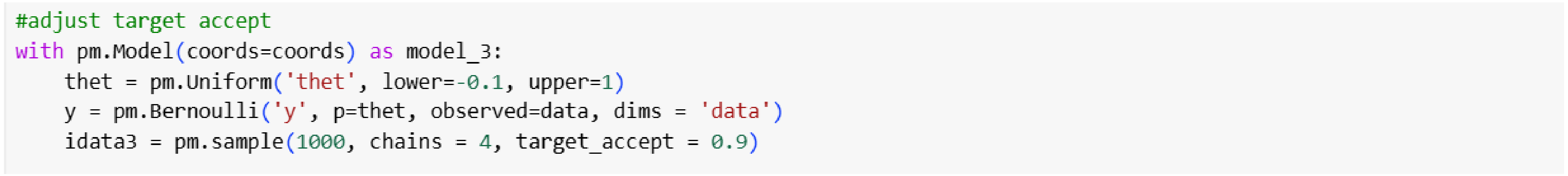
- If there are divergences, it indicates that the sampler has likely found a region of high curvature in the posterior that cannot be explored properly.
  - Only for the NUTS sampler (based on the Hamiltonian Monte Carlo method).
  - We can try and reparametrize the model.
  - We can increase target accept in pm.sample.

Divergence

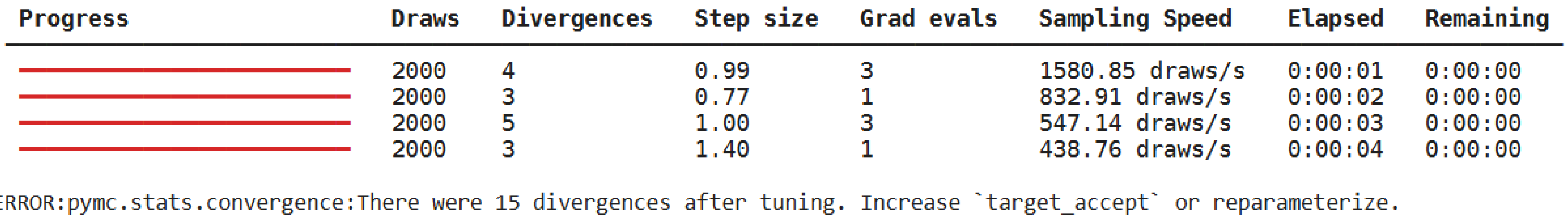


```
Progress                     Draws   Divergences   Step size   Grad evals   Sampling Speed    Elapsed   Remaining

                             2000    13            1.23        3            2652.35 draws/s   0:00:00   0:00:00
                             2000    19            1.01        1            1383.77 draws/s   0:00:01   0:00:00
                             2000    9             1.85        1            833.34 draws/s    0:00:02   0:00:00
                             2000    7             1.73        1            574.27 draws/s    0:00:03   0:00:00

ERROR:pymc.stats.convergence:There were 48 divergences after tuning. Increase `target accept` or reparameterize.
```

- That was for a default target_accept = 0.8
- Increase to 0.9



```python
#adjust target accept
with pm.Model(coords=coords) as model_3:
    thet = pm.Uniform('thet', lower=-0.1, upper=1)
    y = pm.Bernoulli('y', p=thet, observed=data, dims = 'data')
    idata3 = pm.sample(1000, chains = 4, target_accept = 0.9)
```

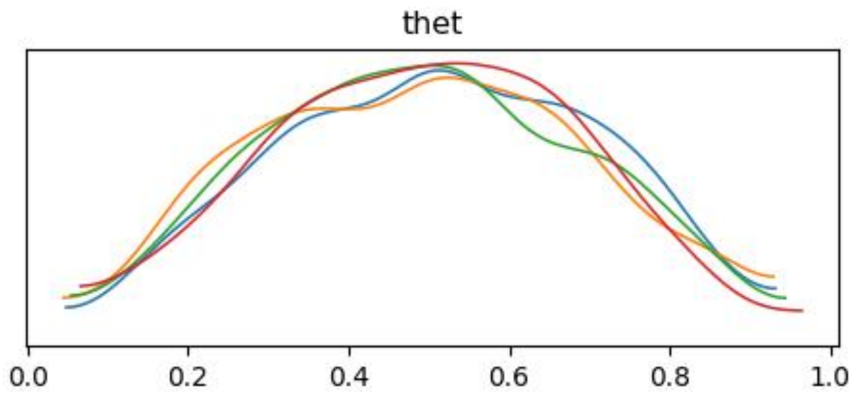


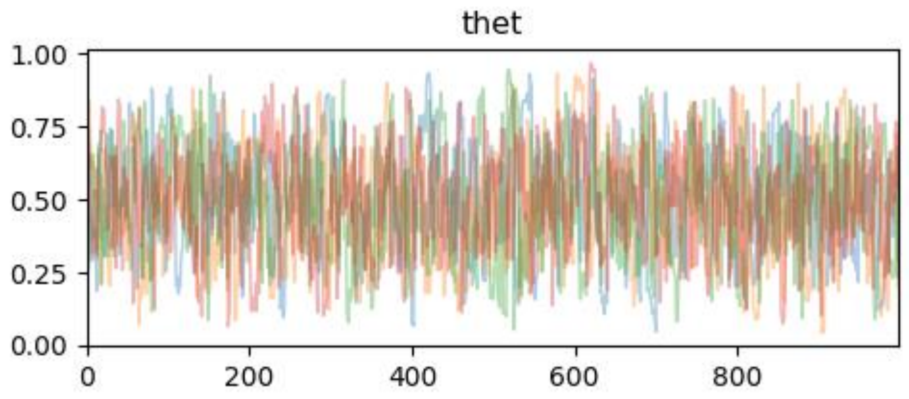
```
Progress                     Draws   Divergences   Step size   Grad evals   Sampling Speed    Elapsed   Remaining

                             2000    4             0.99        3            1580.85 draws/s   0:00:01   0:00:00
                             2000    3             0.77        1            832.91 draws/s    0:00:02   0:00:00
                             2000    5             1.00        3            547.14 draws/s    0:00:03   0:00:00
                             2000    3             1.40        1            438.76 draws/s    0:00:04   0:00:00

ERROR:pymc.stats.convergence:There were 15 divergences after tuning. Increase `target_accept` or reparameterize.
```

Divergence

```
with pm.Model(coords=coords) as model_3:
    thet = pm.Uniform('thet', lower=-0.1, upper=1)
    y = pm.Bernoulli('y', p=thet, observed=data, dims = 'data')
    idata3 = pm.sample(1000, chains = 4, target_accept = 0.99)
```

```
Progress                        Draws   Divergences   Step size   Grad evals   Sampling Speed    Elapsed   Remaining
────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━  2000    0             0.43        15           675.87 draws/s    0:00:02   0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━  2000    0             0.35        7            485.84 draws/s    0:00:04   0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━  2000    0             0.42        15           372.43 draws/s    0:00:05   0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━  2000    0             0.33        3            303.08 draws/s    0:00:06   0:00:00
```


thet


thet