Markov Chain Monte Carlo

BEE 6940 Lecture 8

March 13, 2023

Table of Contents

Review of Markov Chains

Markov Chains

A Markov chain is a stochastic process based on memoryless probabilistic transitions between states.

Key properties:

- Detailed Balance (reversibility) $\Rightarrow$ Existence of stationary distribution
- Ergodicity (irreducible and aperiodic) $\Rightarrow$ Stationary distribution is unique and can be obtained as a limiting distribution.

Idea of Sampling Algorithm

If we construct an ergodic Markov chain with the appropriate transition probabilities for detailed balance to hold with respect to the target distribution, we can use the chain realizations as samples from the target distribution.

- No need for closed-form representation of the posterior
- Can use these samples as normal (with some caveats) for means, quantiles, simulations, etc.

This category of sampling algorithms is called **Markov chain Monte Carlo (MCMC)**.

Convergence to the Target Distribution

Given a Markov chain $\{X_t\}_{t=1,...,T}$ returned from this procedure, sampling from distribution π:

- $\mathbb{P}(X_t = y) \to \pi(y)$ as $t \to \infty$
- This means the chain can be considered a *dependent* sample approximately distributed from π.
- The first values (the *transient portion*) of the chain are highly dependent on the initial value.

The Metropolis-Hastings Algorithm

Metropolis-Hastings Algorithm

The **Metropolis-Hastings** algorithm is the foundational MCMC algorithm (and was named one of the top ten algorithms of the 20th century).

It builds a Markov chain based on transitions by accepting/rejecting proposals for new samples from a *conditional proposal distribution* $q(y|x)$ and accepting or rejecting those proposals.

Metropolis-Hastings Algorithm

Given $X_t = x_t$:

1. Generate $Y_t \sim q(y|x_t)$;
2. Set $X_{t+1} = Y_t$ with probability $\rho(x_t, Y_t)$, where

$$\rho(x, y) = \min\left\{\frac{\pi(y)}{\pi(x)}\frac{q(x|y)}{q(y|x)}, 1\right\},$$

   else set $X_{t+1} = x_t$.

Metropolis-Hastings Algorithm

That almost seems too simple!

But the devil is in the details: performance and efficiency are highly dependent on the choice of q.

Metropolis-Hastings Algorithm

That almost seems too simple!

But the devil is in the details: performance and efficiency are highly dependent on the choice of q.

Key: There is a tradeoff between exploration and acceptance.

- Wide proposal: Can make bigger jumps, may be more likely to reject proposals.
- Narrow proposal: More likely to accept proposals, may not "mix" efficiently.

Proposal Distributions

The original Metropolis et al (1953) algorithm focused on symmetric distributions ($q(y|x) = q(x|y)$), which are a convenient choice as then the acceptance probability reduces to

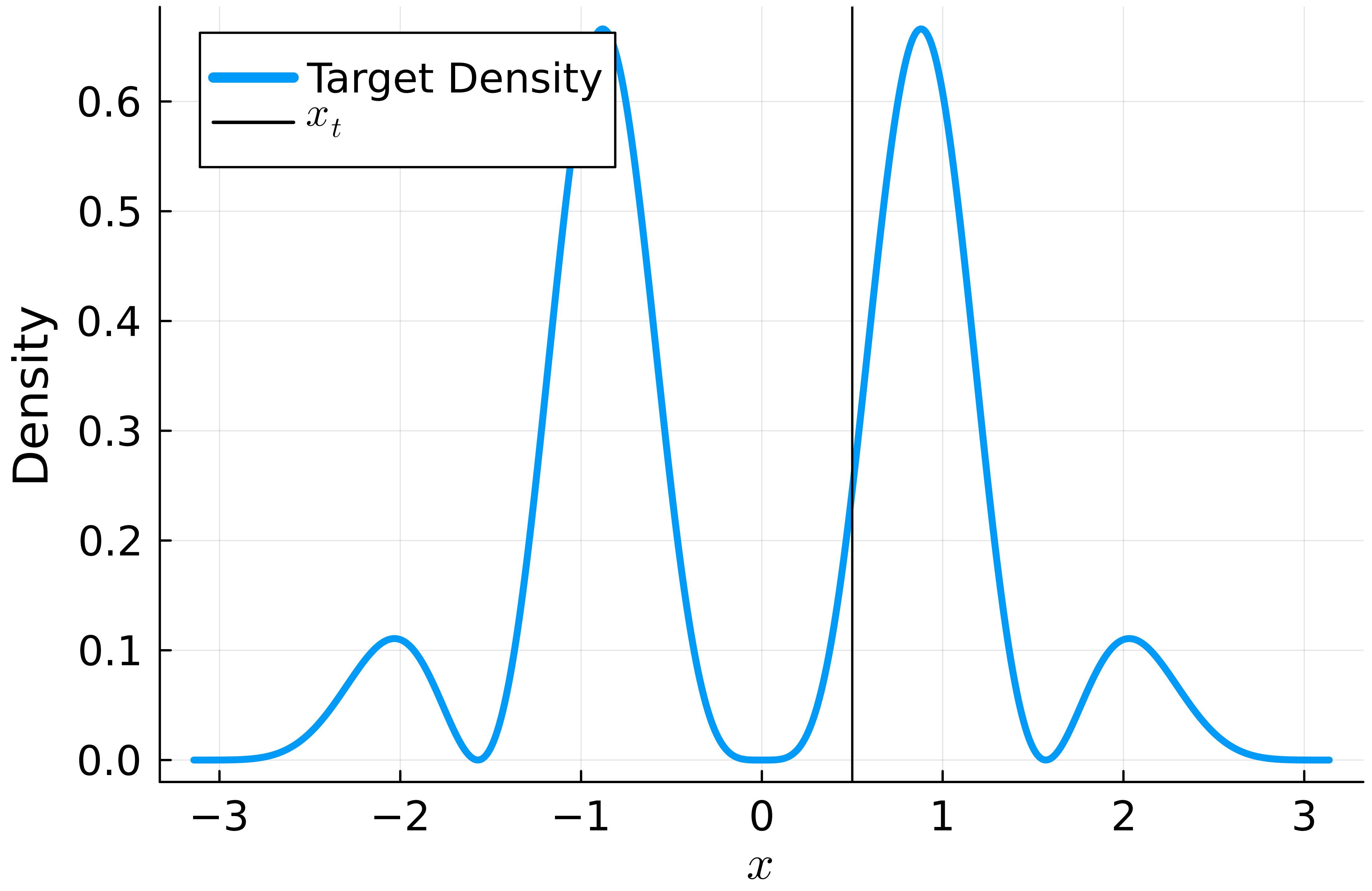
$$\rho = \min\left\{\frac{\pi(y)}{\pi(x)}, 1\right\}.$$

Proposal Distributions

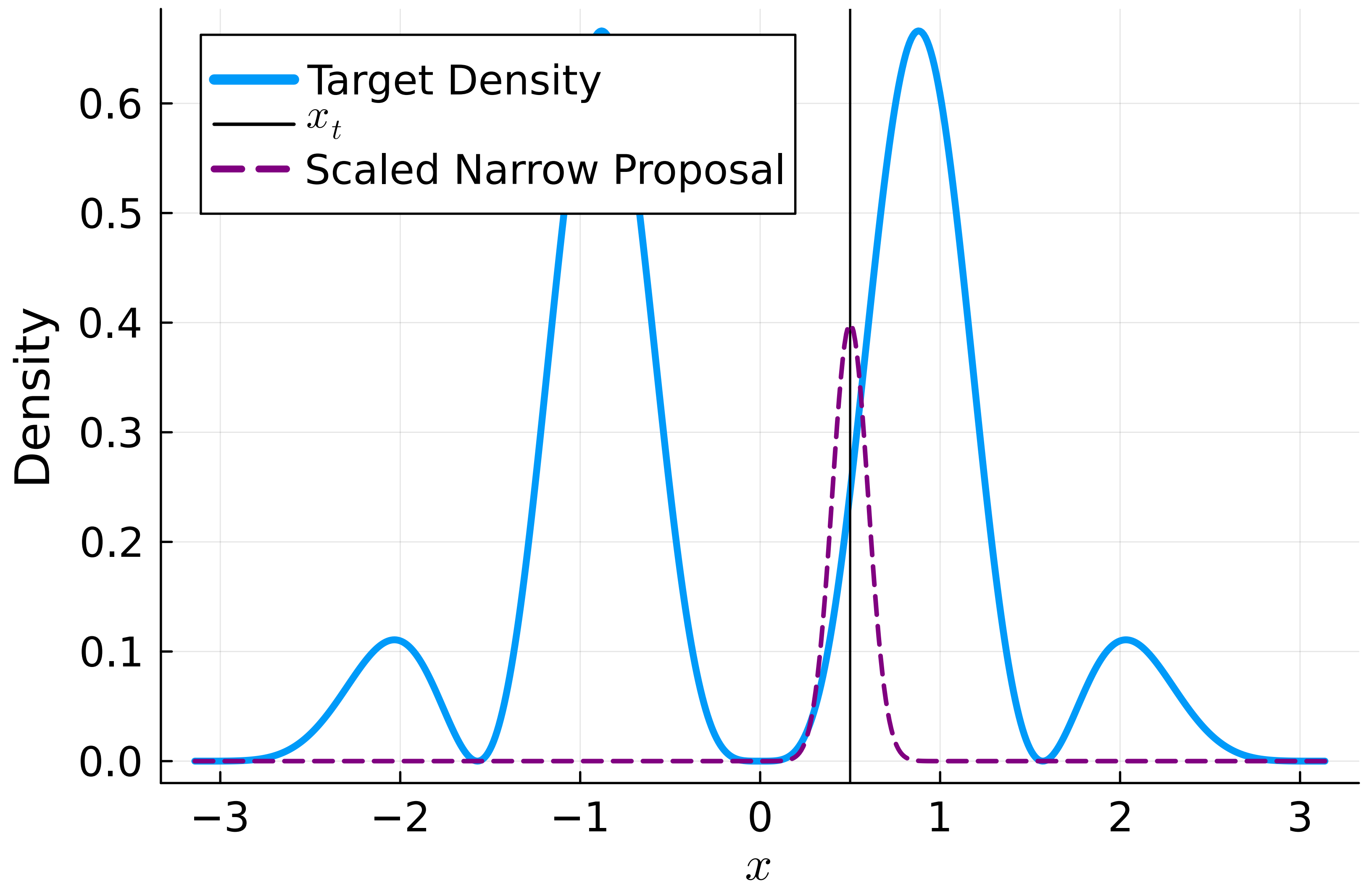
For example, a common choice is a normal density $y \sim \mathrm{Normal}(x_t, \sigma^2)$ centered around the current point.

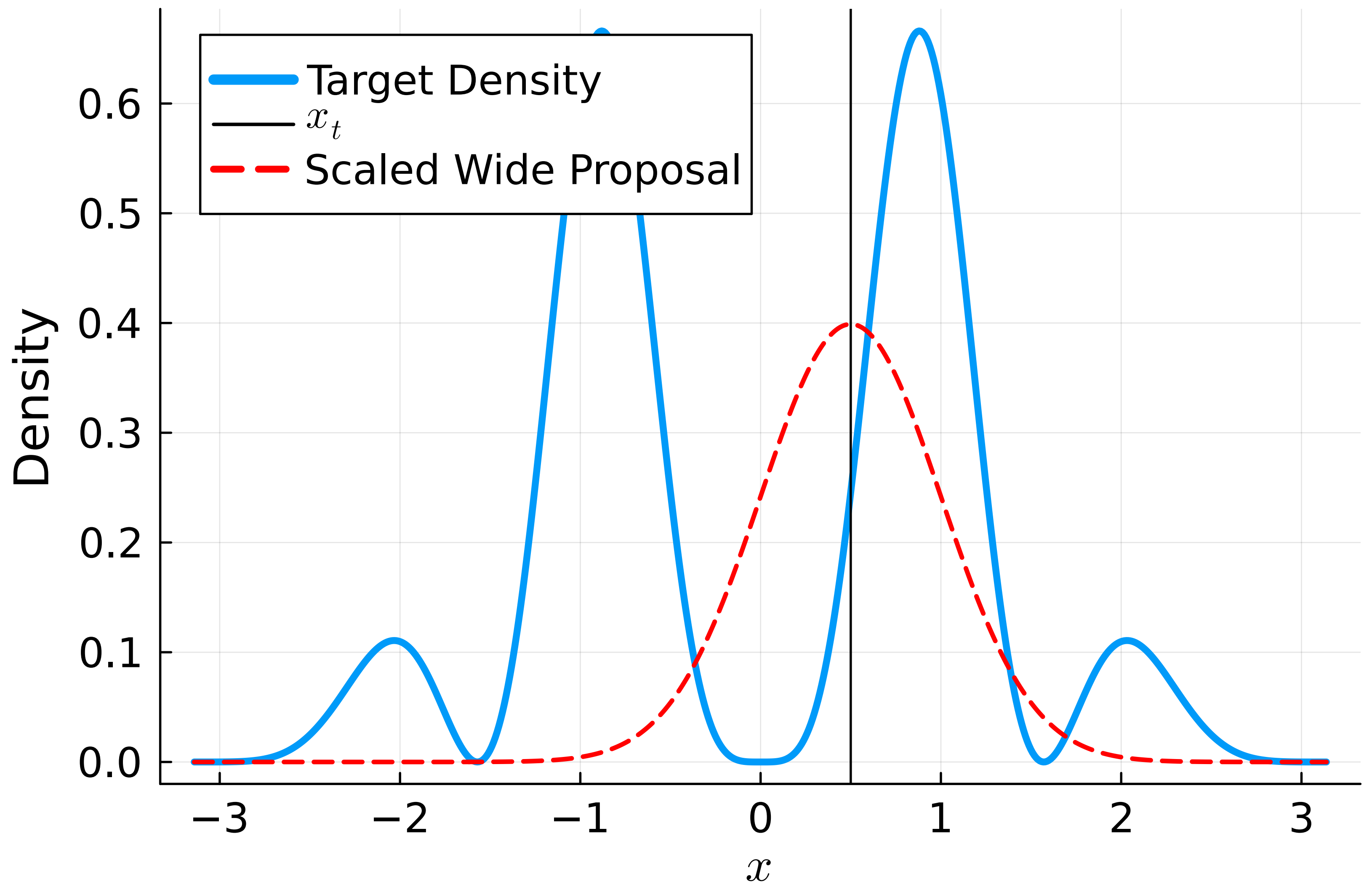
This turns the "exploration" part of the M-H algorithm into a random walk.

Proposal Distributions: Example



Proposal Distributions: Example





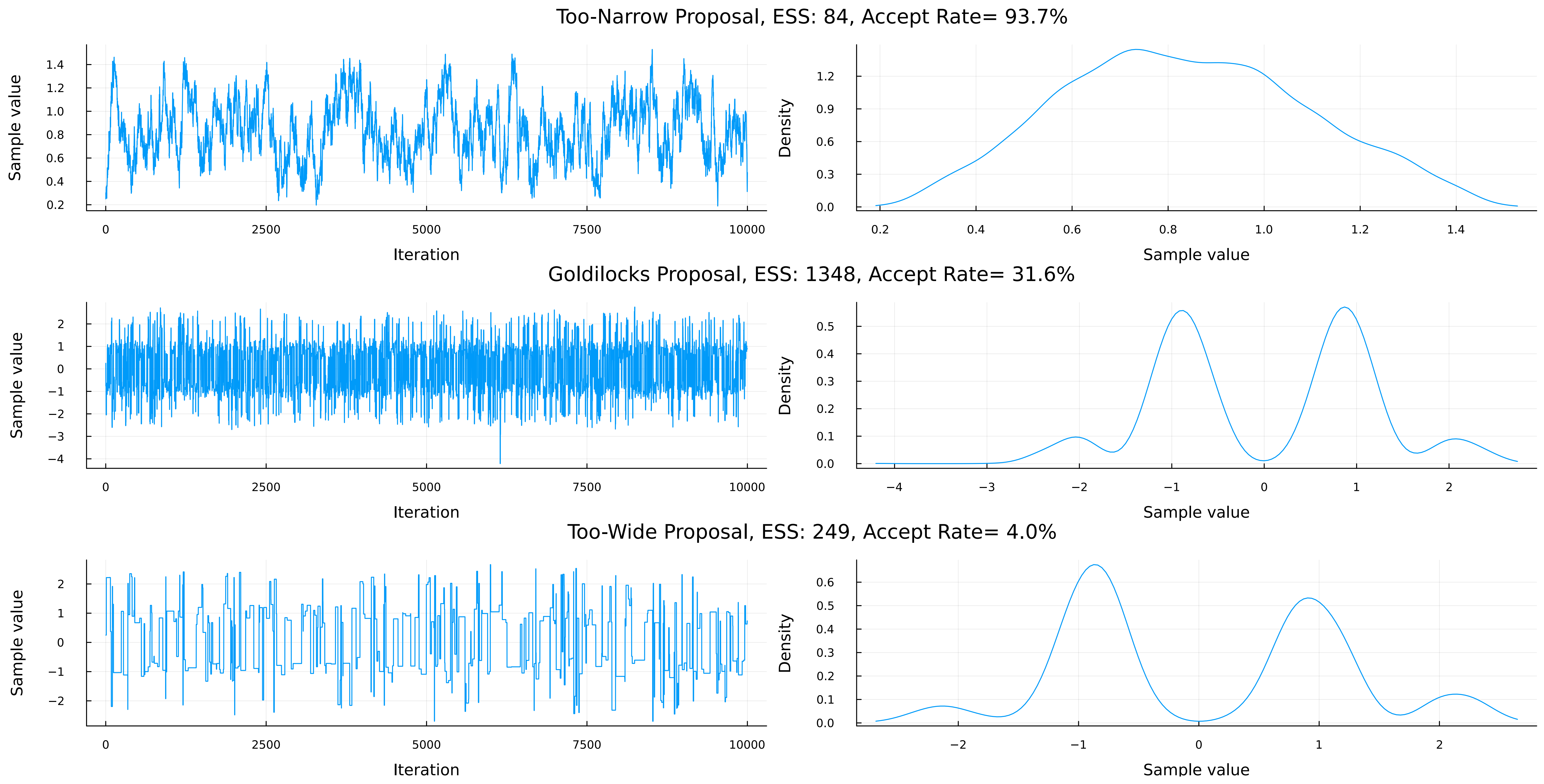
Sampling Efficiency

Two common measures of sampling efficiency:

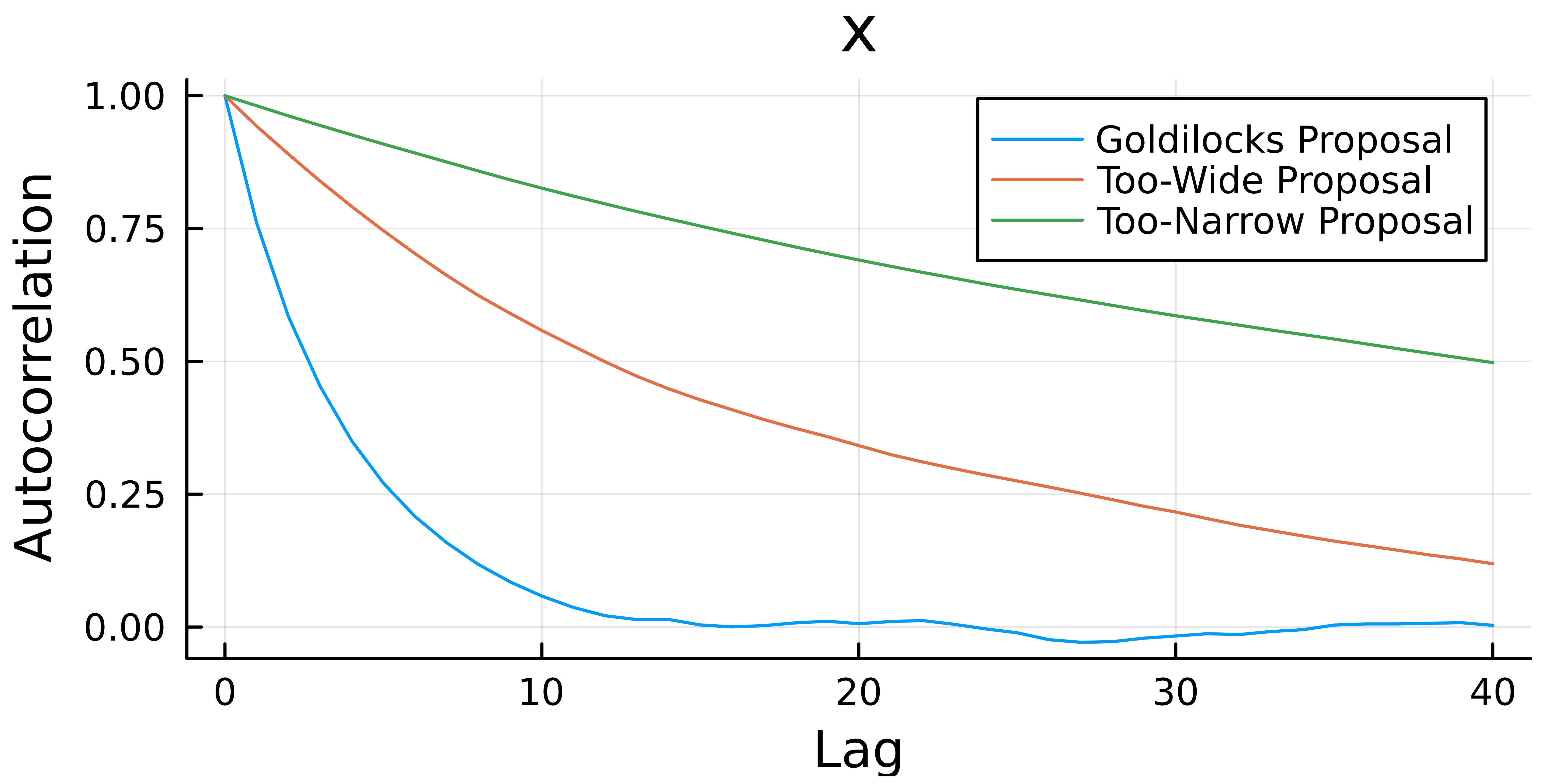
- **Acceptance Rate**: Rate at which proposals are accepted
  - "Optimally" 30-45% (depending on number of parameters)
- **Effective Sample Size (ESS)**: Accounts for autocorrelation ρ_t across samples

$$N_{\text{eff}} = \frac{N}{1 + 2\sum_{t=1}^{\infty} \rho_t}$$

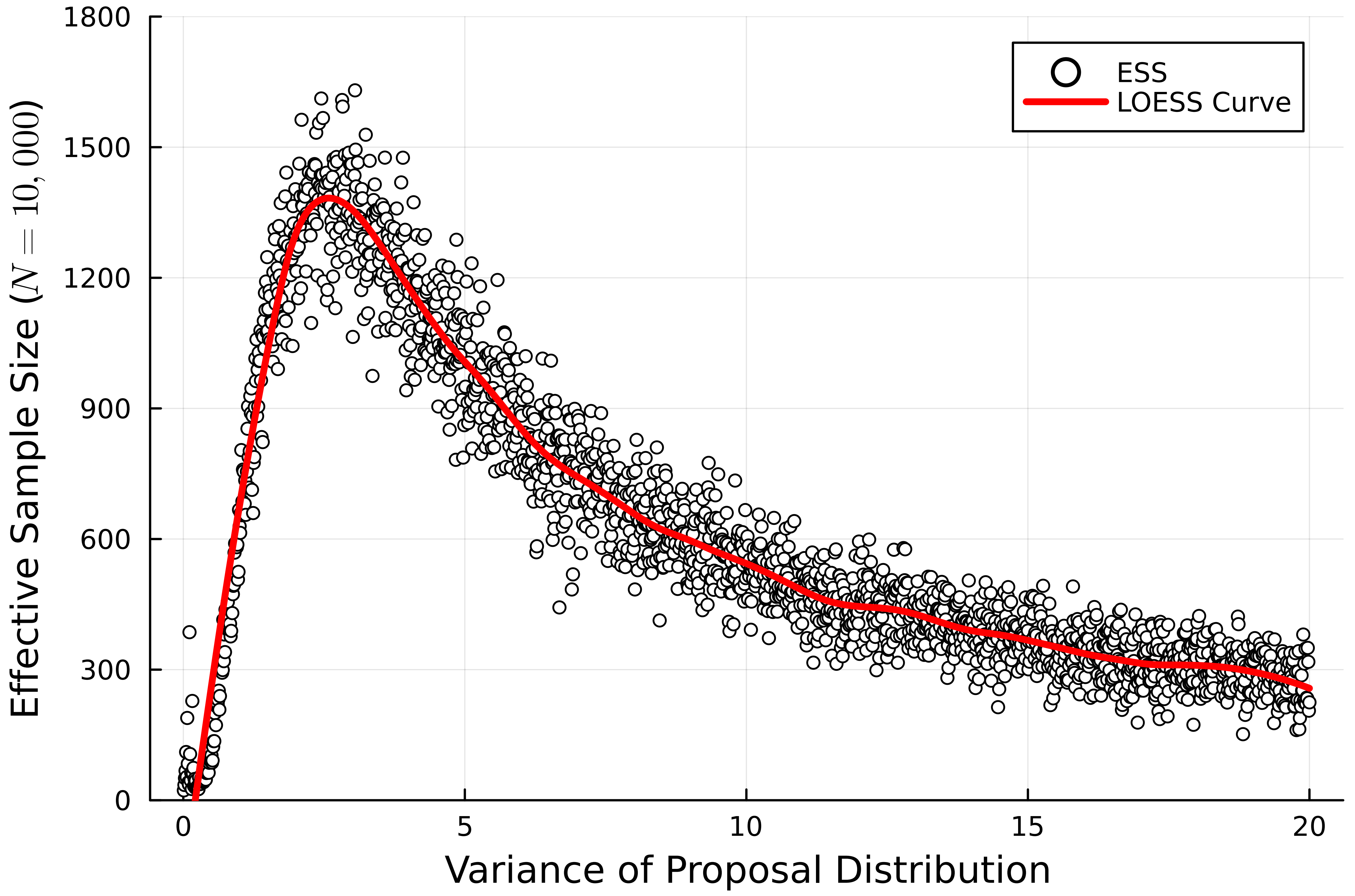
Sampling Efficiency Example



Autocorrelation Of Chains



ESS By Proposal Variance



Convergence to Stationary Distribution

Since the samples are a Markov chain, there is a *transient* portion prior to convergence to the stationary distribution.

What to do about these samples?

Convergence to Stationary Distribution

Since the samples are a Markov chain, there is a *transient* portion prior to convergence to the stationary distribution.

What to do about these samples?

- Discard as *burn-in*;
- Just run the chain longer.

How to Identify Convergence?

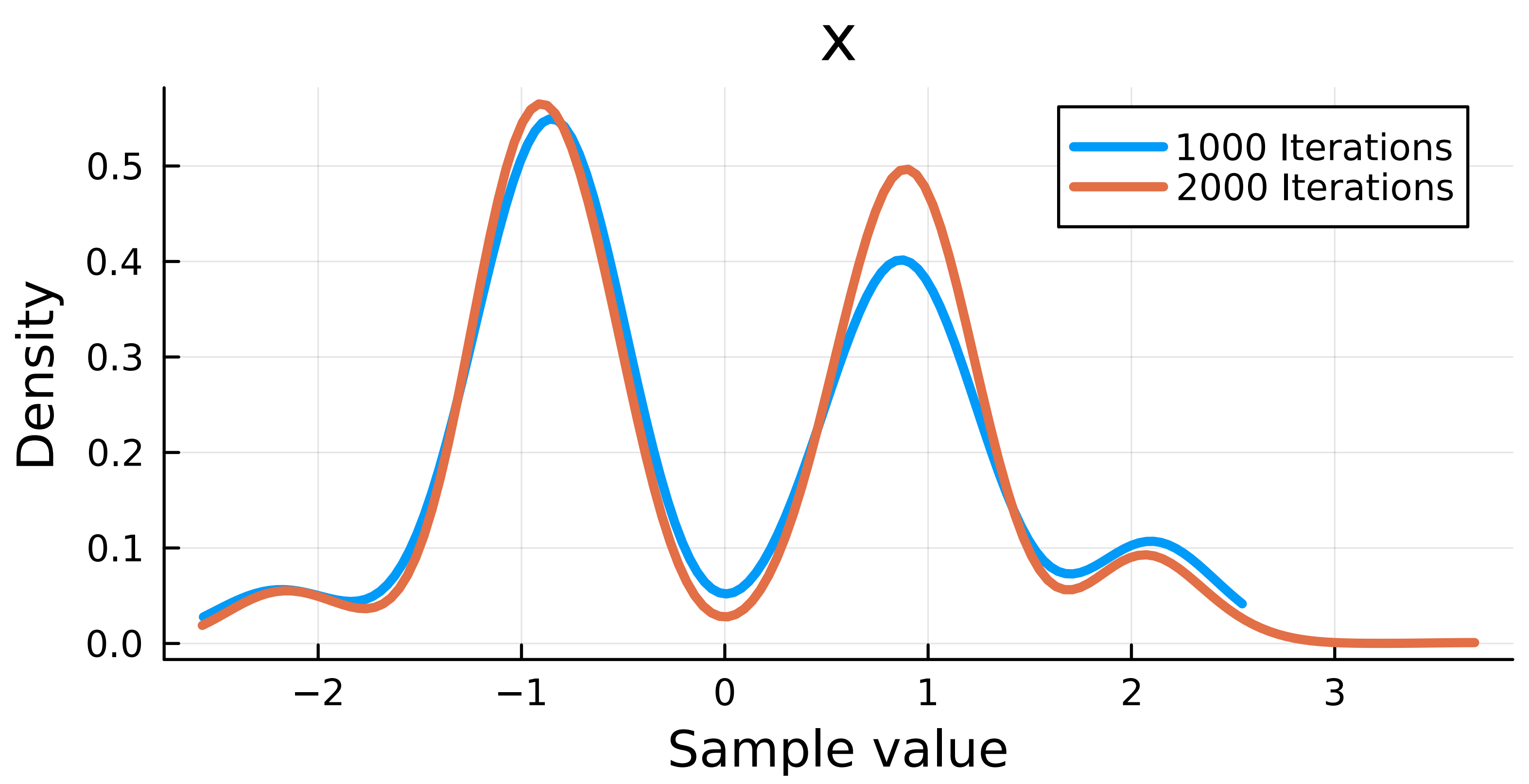
This is probably the most challenging part of MCMC, other than tuning the proposal distribution.

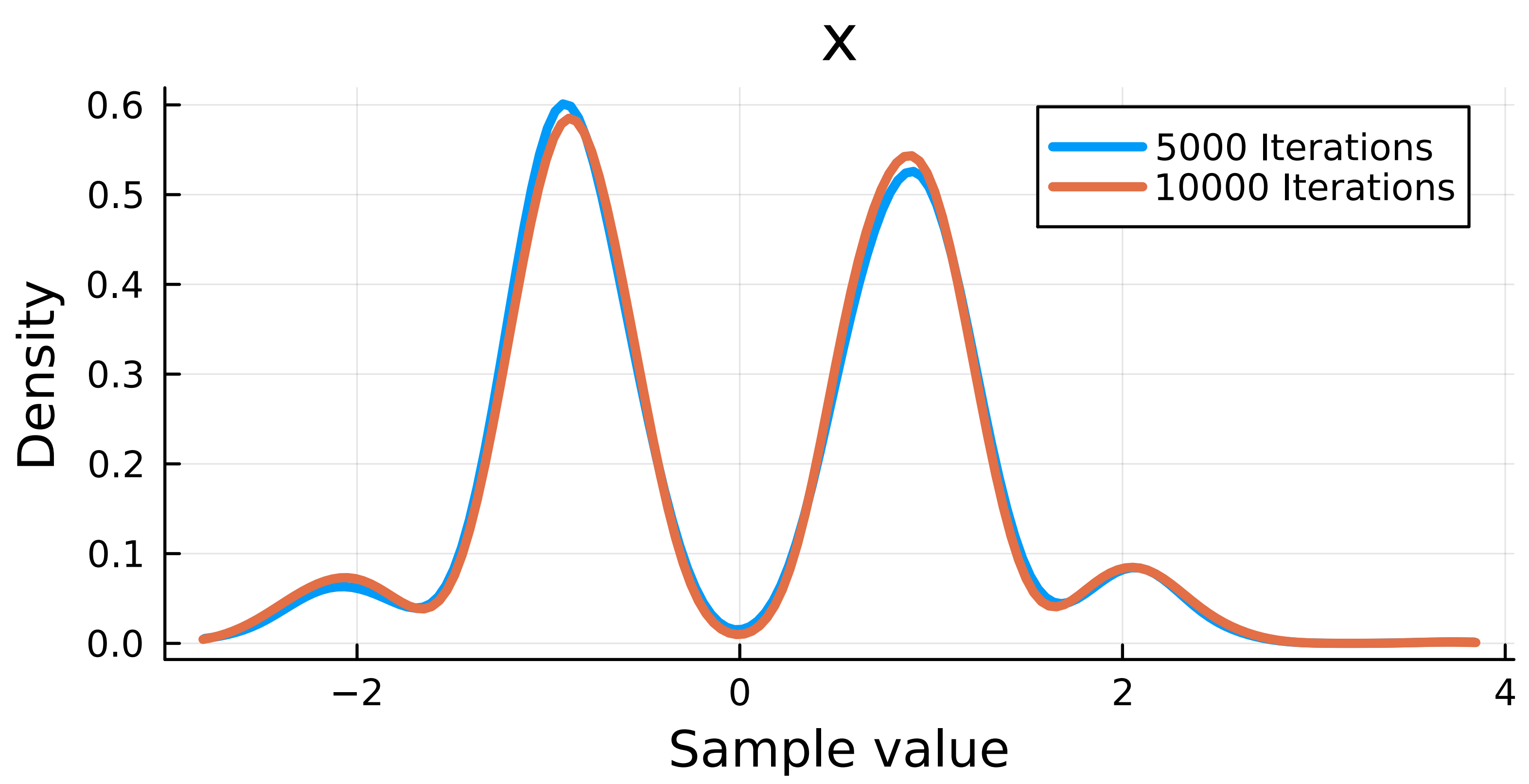
Short answer: There is no guarantee! Judgement based on an accumulation of evidence from various heuristics.

The good news — getting the precise "right" end of the transient chain doesn't matter. If a few transient iterations of the transient portion, the effect will be washed out with a large enough post-convergence chain.

Heuristics for Convergence

- Compare distribution (histogram/kernel density plot) after half of the chain to full chain.





Heuristics for Convergence

- Gelman-Rubin criterion (Gelman & Rubin (1992)):
  - Run multiple chains from "overdispersed" starting points
  - Compare intra-chain and inter-chain variances
  - Summarized as $\hat{R}$ statistic: closer to 1 implies better convergence.
- Can also check distributions across multiple chains vs. the half-chain check. This is the default in `Turing.jl` with multiple chains (will see in lab).

Aside: Multiple Chains

Unless a specific scheme is used, multiple chains are not a solution for issues of convergence, as each individual chain needs to converge and have burn-in discarded/watered-down.

This means multiple chains are more useful for diagnostics, but once they've all been run long enough, can mix samples freely.

Heuristics for Convergence

- If you're more interested in the mean estimate, can also look at the its stability by iteration or the *Monte Carlo standard error*.
- Look at traceplots; do you see sudden "jumps"?
- **When in doubt, run the chain longer.**

Approaches for Increasing Efficiency

- Adaptive M-H (*e.g.* Vihola (2012)): adjusts proposal density to hit target acceptance rate
  - Need to be cautious about detailed balance
  - Typical strategy is to adapt for a portion of the initial chain (part of the burn-in), then run longer with that proposal.

Approaches for Increasing Efficiency

- Hamiltonian Monte Carlo: proposed samples based on "energy function"
  - Can be very efficient due to potential for anti-correlated samples;
  - Becoming the standard in probabilistic programming frameworks (Stan, `Turing.jl`, `pyMC3`)
  - Requires gradient information: can be obtained through autodifferentiation, challenging for external, black-box models.

Key Takeaways: MCMC

- Construct ergodic and reversible Markov chains with posterior as stationary distribution.
- Metropolis-Hastings: conceptually simple algorithm, but implementation plays a major role.
- Must rely on "accumulation of evidence" from heuristics for determination about convergence to stationary distribution.
- Transient portion of chain: Meh. Some people worry about this too much. Discard or run the chain longer.
- Parallelizing solves few problems, but running multiple chains can be useful for diagnostics.

Upcoming Schedule

Upcoming Schedule

Wednesday: Lab on using `Turing.jl` for probabilistic programming and MCMC

Next Monday: Storm surge and modeling extreme values