Markov Chain Monte Carlo
Theory and Implementation from Scratch

Corrado Botta, PhD

Bocconi University

October 26, 2025

Outline

What is MCMC?

Markov Chain Monte Carlo (MCMC) is a class of algorithms for sampling from probability distributions:

- **Markov Chain**: A sequence of random samples where each depends only on the previous one
- **Monte Carlo**: Using random sampling to solve computational problems

MCMC solves a fundamental problem in statistics:

- How to sample from complex, high-dimensional distributions
- When direct sampling is impossible or computationally infeasible
- Essential for Bayesian inference where posterior distributions are often intractable

Key Insight: Construct a Markov chain whose stationary distribution is the target distribution we want to sample from.

Markov Chains

A Markov chain is a sequence of random variables $X_0, X_1, X_2, \ldots$ with the Markov property:

$$P(X_{t+1}|X_t, X_{t-1}, \ldots, X_0) = P(X_{t+1}|X_t) \tag{1}$$

Key Properties for MCMC

- **Ergodicity**: The chain eventually visits all states with positive probability
- **Detailed Balance**: $\pi(x)P(y|x) = \pi(y)P(x|y)$ ensures π is stationary
- **Convergence**: Under certain conditions, the chain converges to a unique stationary distribution π
- **Irreducibility**: Any state can be reached from any other state

For MCMC, we design the transition kernel $P(y|x)$ so that π equals our target distribution.

Metropolis-Hastings Algorithm

Algorithm Steps:

1. Start with initial value x_0
2. For $t = 0, 1, 2, \ldots$:
   - Propose new state $y \sim q(y|x_t)$ from proposal distribution
   - Calculate acceptance ratio:

$$\alpha(x_t, y) = \min\left(1, \frac{\pi(y)q(x_t|y)}{\pi(x_t)q(y|x_t)}\right) \tag{2}$$

   - Accept $x_{t+1} = y$ with probability α
   - Otherwise, reject and set $x_{t+1} = x_t$

Special Cases

- **Random Walk Metropolis**: Symmetric proposal $q(y|x) = q(x|y)$
- **Independence Sampler**: Proposal independent of current state
- **Gibbs Sampling**: Always accepts (acceptance ratio $= 1$)

Gibbs Sampling

Gibbs sampling is a special case of Metropolis-Hastings for multivariate distributions:

For a d-dimensional distribution $\pi(x_1, x_2, \ldots, x_d)$:

1. Initialize $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, \ldots, x_d^{(0)})$
2. For iteration t:

$$\begin{aligned} x_1^{(t+1)} &\sim \pi(x_1 | x_2^{(t)}, x_3^{(t)}, \ldots, x_d^{(t)}) \\ x_2^{(t+1)} &\sim \pi(x_2 | x_1^{(t+1)}, x_3^{(t)}, \ldots, x_d^{(t)}) \\ &\vdots \\ x_d^{(t+1)} &\sim \pi(x_d | x_1^{(t+1)}, x_2^{(t+1)}, \ldots, x_{d-1}^{(t+1)}) \end{aligned} \tag{3}$$

Advantages: No tuning required, always accepts proposals
Disadvantages: Requires knowledge of conditional distributions, can be slow for highly correlated variables

R Implementation - Metropolis-Hastings

```r
# Metropolis-Hastings sampler from scratch
metropolis_hastings <- function(target_log_density, initial,
                                n_samples, proposal_sd = 1) {
  # Initialize storage for samples
  samples <- matrix(0, nrow = n_samples, ncol = length(initial))
  samples[1,] <- initial
  current <- initial
  n_accept <- 0

  # Main MCMC loop
  for (i in 2:n_samples) {
    # Propose new state (random walk)
    proposal <- current + rnorm(length(current), 0, proposal_sd)

    # Calculate log acceptance ratio
    log_alpha <- target_log_density(proposal) -
                 target_log_density(current)

    # Accept or reject
    if (log(runif(1)) < log_alpha) {
      current <- proposal
      n_accept <- n_accept + 1
    }
    samples[i,] <- current
  }

  list(samples = samples, accept_rate = n_accept / n_samples)
}
```

R Implementation - Gibbs Sampling

```
# Gibbs sampler for bivariate normal distribution
gibbs_sampler <- function(n_samples, mu1 = 0, mu2 = 0,
                          sigma1 = 1, sigma2 = 1, rho = 0.8) {
  # Initialize storage
  samples <- matrix(0, nrow = n_samples, ncol = 2)
  samples[1,] <- c(0, 0)  # Initial values

  # Conditional distribution parameters
  for (i in 2:n_samples) {
    # Sample x1 | x2
    mu_cond1 <- mu1 + rho * sigma1/sigma2 * (samples[i-1, 2] - mu2)
    sigma_cond1 <- sqrt((1 - rho^2) * sigma1^2)
    samples[i, 1] <- rnorm(1, mu_cond1, sigma_cond1)

    # Sample x2 | x1
    mu_cond2 <- mu2 + rho * sigma2/sigma1 * (samples[i, 1] - mu1)
    sigma_cond2 <- sqrt((1 - rho^2) * sigma2^2)
    samples[i, 2] <- rnorm(1, mu_cond2, sigma_cond2)
  }

  return(samples)
}
```

R Implementation - Example Application

```r
# Example: Sampling from a mixture of Gaussians
mixture_log_density <- function(x) {
  # Log density of 0.3*N(-2,1) + 0.7*N(3,1.5)
  comp1 <- 0.3 * dnorm(x, -2, 1)
  comp2 <- 0.7 * dnorm(x, 3, 1.5)
  return(log(comp1 + comp2))
}

# Run MCMC
set.seed(123)
result <- metropolis_hastings(mixture_log_density,
                              initial = 0,
                              n_samples = 10000,
                              proposal_sd = 2)

# Print diagnostics
cat("Acceptance rate:", round(result$accept_rate, 3), "\n")
```

Acceptance rate: 0.671

```r
cat("Sample mean:", round(mean(result$samples[5001:10000]), 3), "\n")
```

Sample mean: 1.279

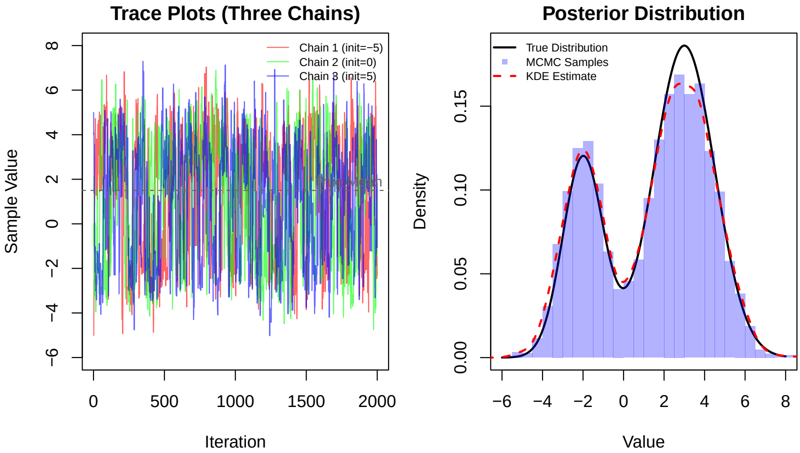
```r
cat("Sample std dev:", round(sd(result$samples[5001:10000]), 3), "\n")
```

Sample std dev: 2.791

```r
# True theoretical values
true_mean <- 0.3*(-2) + 0.7*3  # 1.5
true_var <- 0.3*(1 + 4) + 0.7*(1.5^2 + 9) - 1.5^2  # 8.475
cat("True mean:", true_mean, "\n")
```

True mean: 1.5

Visualization of MCMC Convergence



- Left: Three chains with different initial values converge to the same distribution
- Right: MCMC samples accurately approximate the true mixture distribution
- Burn-in period (1000 iterations) allows chains to reach stationarity

Applications and Takeaways

Key Applications

- **Bayesian Inference**: Sampling from posterior distributions
- **Financial Modeling**: Option pricing, risk assessment, portfolio optimization
- **Machine Learning**: Bayesian neural networks, latent variable models
- **Physics**: Statistical mechanics, quantum field theory simulations
- **Genetics**: Phylogenetic tree reconstruction, population genetics

Takeaways

- MCMC enables sampling from complex distributions where direct methods fail
- Metropolis-Hastings provides a general framework: accept/reject based on $\alpha = \min(1, \frac{\pi(y)q(x|y)}{\pi(x)q(y|x)})$
- Convergence diagnostics are crucial: check trace plots, autocorrelation, and $\hat{R}$
- Trade-off between exploration (large steps) and acceptance rate
- Modern variants (HMC, NUTS) improve efficiency for high-dimensional problems