# 498818_SDS_475_Summary3

2024-10-16

## 498818 SDS 475 Summary 3

### Monte Carlo Integration

Monte Carlo Integration is a numerical method to estimate the value of an integral by taking random samples from a probability distribution. The idea is to approximate an integral by averaging realizations of a random variable $X$, based on the law of large numbers (LLN). Monte Carlo Integration turns an integration problem into an averaging problem.

#### Error Estimation

The error for Monte Carlo Integration decreases as $\frac{1}{\sqrt{N}}$, where $N$ is the number of samples. Although Monte Carlo Integration has slow convergence compared to other methods, its performance does not degrade with dimensionality, making it ideal for high-dimensional integrals (i.e., the curse of dimensionality).

### Curse of Dimensionality

In higher dimensions, traditional numerical integration methods like quadrature struggle because the number of required samples increases exponentially with the number of dimensions. This is referred to as the **Curse of Dimensionality**. Monte Carlo methods are less affected by this because the error rate $\frac{1}{\sqrt{N}}$ is independent of dimensionality.

- **Key Idea**: The error decreases at the same rate, regardless of dimensionality, making Monte Carlo methods particularly efficient for high-dimensional problems.
- **Benefit**: Traditional methods scale poorly in high dimensions, whereas Monte Carlo scales linearly with the number of samples, making it useful in cases where the number of dimensions is very large.

### Variance Reduction

To improve accuracy, variance reduction techniques can be applied:

- **Importance Sampling**: A method to reduce variance by sampling from a distribution that better approximates the target distribution.
- **Rao-Blackwellization**: A technique to reduce variance by conditioning on sufficient statistics, thus creating a more efficient estimator.
- **Acceleration Methods**:
  - **Antithetic Variables**: Generate pairs of negatively correlated variables to reduce variance.
  - **Control Variables**: Use known values from similar integrals to adjust the estimate of the unknown integral.

## Importance Sampling

**Importance Sampling** is a technique used to improve the accuracy of Monte Carlo estimates by drawing samples from a proposal distribution that is more informative for the target function. The weights are adjusted to account for the discrepancy between the proposal and target distributions.

- **Why is this useful?** In Monte Carlo, if the target distribution has regions with low probability but high contribution to the integral, standard random sampling will miss these regions. By shifting the sampling focus to these areas (via a new proposal distribution), Importance Sampling ensures that more relevant regions are sampled.
- **Key Idea**: Instead of sampling from $p(x)$, we sample from $q(x)$, which is easier to sample from, and then re-weight the estimates by $\frac{p(x)}{q(x)}$.
- **Goal**: Reduce variance by focusing on important regions of the probability space.
- **Caveat**: It can increase variance if $q(x)$ poorly approximates $p(x)$.

## Rao-Blackwellization

**Rao-Blackwellization** transforms a rough estimator into a more accurate one by conditioning on sufficient statistics. It exploits the relationship between different variables in a distribution to produce more efficient estimates.

- **Key Idea**: Use conditional expectations to reduce variance. If an estimator can be conditioned on additional information, the result is usually a lower-variance estimator.
- **Example**: In Gibbs Sampling, Rao-Blackwellization can be applied to the intermediate results to improve the efficiency of the algorithm.
- **When is this applicable?**
  - Importance sampling: when using a mixture distribution.
  - Gibbs sampling: the conditional distributions allow Rao-Blackwellization.

## Antithetic and Control Variables

### Antithetic Variables

This method works by generating two negatively correlated variables whose mean is used for the estimate. By constructing variables that "cancel each other out" in terms of their variance, the overall variance of the estimate is reduced.

- **How it works**: Generate pairs of samples, one from $X$ and the other from $1 - X$. This introduces symmetry and tends to stabilize the estimate.
- **Effect**: The method reduces variance by introducing negative correlation between estimates.

### Control Variables

Control variables are used when you know the expected value of a related integral. The known value can be used to reduce the variance in the estimate of the unknown integral.

- **Key Idea**: Use the difference between the known and estimated values of a related integral to "correct" the unknown estimate, thus reducing variance.
- **Additional Detail**: This approach can be particularly effective when the control variable is strongly correlated with the quantity of interest, resulting in significant variance reduction.

## Connection to Pseudo-random Generation Techniques

Earlier methods like inverse CDF, acceptance-rejection, and transformation methods are critical for generating i.i.d random samples from arbitrary distributions, which are used in traditional Monte Carlo methods.

However, when dealing with **dependent samples (as in MCMC)**, these methods are no longer feasible. That's where **Markov Chains** and **MCMC methods** come in, allowing sampling from complex, dependent distributions.

The transition from arbitrary distribution generation to MCMC methods is motivated by the need to handle dependent samples. MCMC allows us to sample from distributions when independence assumptions are violated, thus providing a powerful tool for sampling in more complex statistical models.

## Monte Carlo with Markov Chains (MCMC)

**Markov Chain Monte Carlo (MCMC)** is used when direct sampling from a complex distribution is impossible. Instead, MCMC constructs a Markov Chain whose stationary distribution is the target distribution, allowing us to sample from this chain.

- **Why MCMC?** Traditional Monte Carlo methods assume independent samples. However, MCMC allows us to work with dependent samples while still converging to the target distribution (ergodicity).
- **Ergodic Theorem**: If the Markov Chain is aperiodic, irreducible, and positive recurrent, the chain will converge to the stationary distribution, and the Monte Carlo estimator will be valid.

## Common MCMC Algorithms

### Metropolis-Hastings (MH)

The **Metropolis-Hastings algorithm** generates new states by proposing a move and accepting or rejecting it based on the target distribution.

**Metropolis Algorithm**: In the case of a symmetric proposal distribution where $k(y|x) = k(x|y)$, the acceptance ratio simplifies, making it computationally more efficient.

**Steps:**

1. **Propose a new state**.
2. **Calculate acceptance ratio**.
3. **Accept/reject based on the ratio**.

### Selecting a Kernel

The **kernel** (proposal distribution) in MH determines the efficiency of sampling.

- **It must ensure the chain is irreducible, positive recurrent, and aperiodic**, allowing it to explore the full space and capture the slight dependencies between states.

**Requirements:**

- Ability to explore the entire state space.
- Balance between large exploratory steps and local refinements.

### Random-Walk Metropolis (RWM)

A special case of MH where the proposal is a random walk.

**Acceptance rate:**

The step size impacts the acceptance rate. A small step size leads to high acceptance but slow exploration; a large step size leads to low acceptance.

#### Metropolized Independence Sampler (MIS)

A variation of MH where the proposal distribution is independent of the current state. It uses a fixed distribution designed to approximate the target.

## Convergence and Transition Kernel

The **transition kernel** $k(y|x)$ describes how the algorithm moves from state $x$ to state $y$. The kernel is critical for determining the efficiency and speed of convergence in MCMC.

If the conditions for ergodicity are met (aperiodic, irreducible, and positive recurrent), these methods will always converge to the target distribution. However, the convergence can be slow due to long burn-in phases or inefficiencies in exploring the state space.

## Putting it All Together

Now that we've explored pseudo-random generation, Monte Carlo methods, and variance reduction techniques, the key takeaway is the transition to **Markov Chain Monte Carlo (MCMC)** for sampling complex distributions:

- **Pseudo-random generation techniques** (inverse CDF, acceptance-rejection, etc.) work well for generating i.i.d. samples but break down when dependencies are introduced.
- **Monte Carlo methods** help us estimate integrals by averaging over random samples.
- **Variance Reduction Techniques** (Importance Sampling, Rao-Blackwellization, Antithetic Variables, Control Variables, etc.) improve the efficiency and accuracy of Monte Carlo estimates. **These techniques can be applied in both traditional Monte Carlo methods (for i.i.d. samples) and MCMC methods (for dependent samples).**
- **Curse of Dimensionality**: Monte Carlo methods remain efficient in higher dimensions because the error rate is independent of dimensionality, which makes them ideal for high-dimensional problems. Traditional quadrature methods do not scale well with increasing dimensions, but Monte Carlo (including MCMC) does, with error decreasing at $\frac{1}{\sqrt{N}}$.
- **MCMC** allows us to handle dependencies between samples, making it essential for more complex, real-world problems.

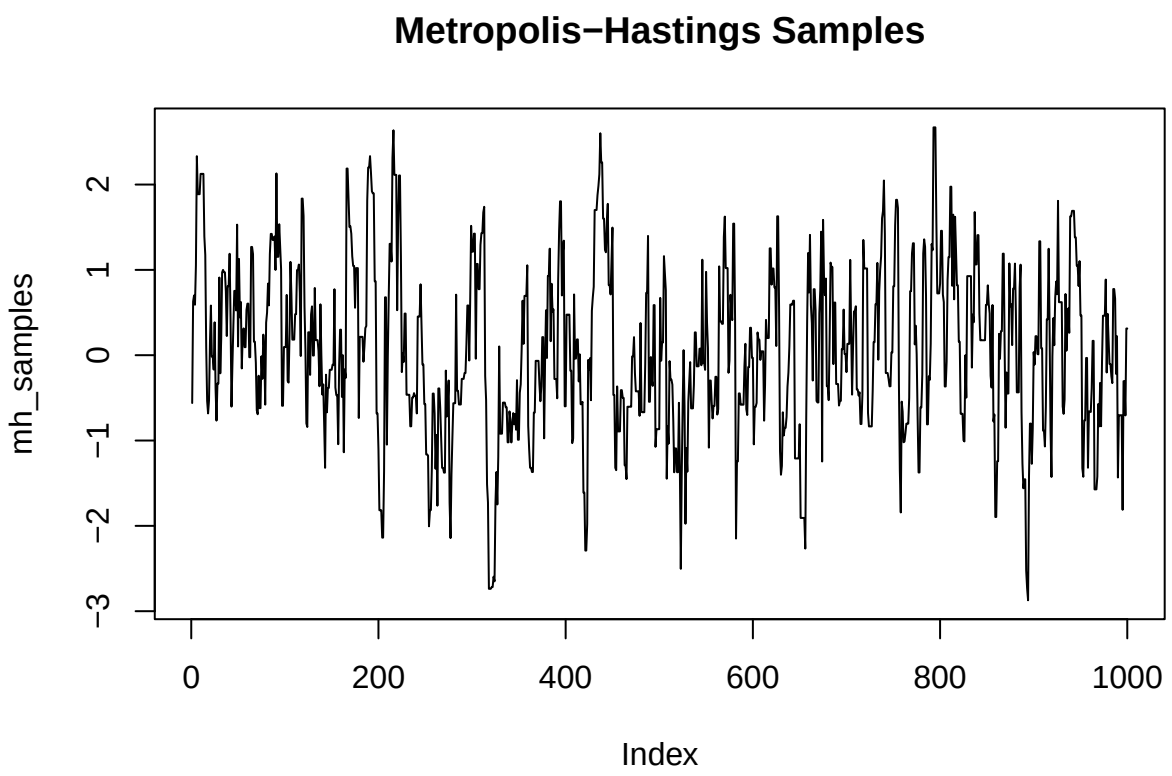
Additionally, bias reduction techniques like **Cross-Validation (CV)**, **Jackknife**, and **Bootstrap** can be applied to both **traditional Monte Carlo methods** and **MCMC** to help reduce bias in estimates, further enhancing model performance.

**MCMC algorithms** combine everything we've learned to create a framework for sampling from difficult-to-analyze distributions. It unifies random generation, Monte Carlo techniques, variance reduction, bias reduction, and advanced statistical estimation into a single powerful tool that applies to both independent and dependent sampling scenarios.

## Key Algorithms Basic Examples in R

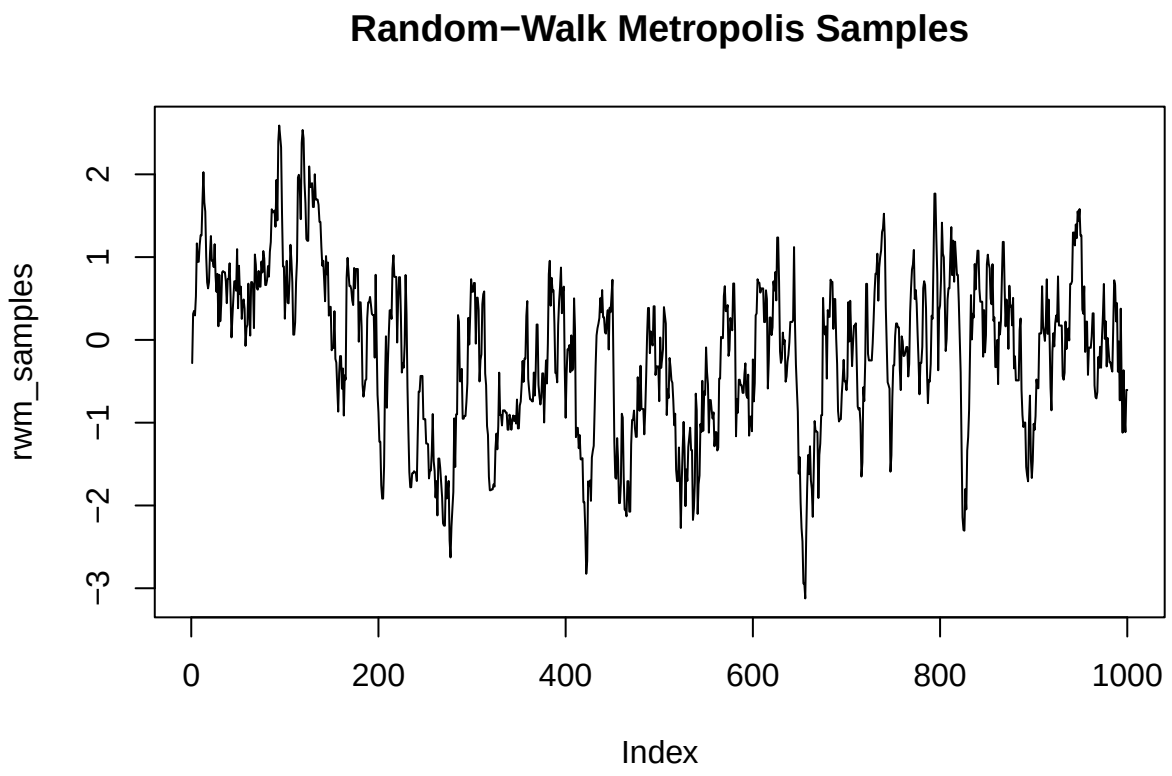
```r
set.seed(123)
# Simple Metropolis-Hastings Example in R
metropolis_hastings <- function(initial_state, n_iter, proposal_sd, target_density) {
  state <- initial_state
  samples <- numeric(n_iter)

  for (i in 1:n_iter) {
    proposal <- rnorm(1, state, proposal_sd)
    acceptance_ratio <- min(1, target_density(proposal) / target_density(state))
```

```
    if (runif(1) < acceptance_ratio) {
      state <- proposal
    }
    samples[i] <- state
  }

  return(samples)
}

# Example target density (standard normal)
target_density <- function(x) {
  dnorm(x, mean = 0, sd = 1)
}

# Run MH algorithm
mh_samples <- metropolis_hastings(initial_state = 0, n_iter = 1000, proposal_sd = 1, target_density)
plot(mh_samples, type="l", main="Metropolis-Hastings Samples")
```




```
set.seed(123)
# Random-Walk Metropolis Algorithm
random_walk_metropolis <- function(initial_state, n_iter, proposal_sd, target_density) {
  state <- initial_state
  samples <- numeric(n_iter)

  for (i in 1:n_iter) {
    proposal <- state + rnorm(1, 0, proposal_sd)
    acceptance_ratio <- min(1, target_density(proposal) / target_density(state))

    if (runif(1) < acceptance_ratio) {
```

```
      state <- proposal
    }
    samples[i] <- state
  }

  return(samples)
}

# Example target density (standard normal)
target_density <- function(x) {
  dnorm(x, mean = 0, sd = 1)
}

# Run Random-Walk Metropolis algorithm
rwm_samples <- random_walk_metropolis(initial_state = 0, n_iter = 1000, proposal_sd = 0.5, target_densi
plot(rwm_samples, type="l", main="Random-Walk Metropolis Samples")
```




```
set.seed(123)
# Metropolized Independence Sampler Example
mis_sampler <- function(initial_state, n_iter, proposal_density, target_density) {
  state <- initial_state
  samples <- numeric(n_iter)

  for (i in 1:n_iter) {
    proposal <- rnorm(1, 0, 2)  # Example fixed proposal distribution
    acceptance_ratio <- min(1, (target_density(proposal) * proposal_density(state)) /
                            (target_density(state) * proposal_density(proposal)))

    if (runif(1) < acceptance_ratio) {
```

```
      state <- proposal
    }
    samples[i] <- state
  }

  return(samples)
}

# Example target and proposal densities
target_density <- function(x) {
  dnorm(x, mean = 0, sd = 1)
}

proposal_density <- function(x) {
  dnorm(x, mean = 0, sd = 2)  # Fixed proposal distribution
}

# Run Metropolized Independence Sampler
mis_samples <- mis_sampler(initial_state = 0, n_iter = 1000, proposal_density, target_density)
plot(mis_samples, type="l", main="Metropolized Independence Sampler Samples")
```