Homework 3: Monte Carlo Sampling and the Metropolis Algorithm

Md Masudul Islam

March 8, 2025

Problem 1: Prove the Rejection Sampling Theorem

1. State and prove the main theorem in the rejection sampling algorithm.

Theorem 1 (Rejection Sampling Theorem)**.** *Let $f(x)$ be a probability density function (pdf) on a space $\mathcal{X}$, and let $g(x)$ be another pdf on the same space from which sampling is easy. Suppose there exists a constant $M > 0$ such that*

$$f(x) \;\leq\; M\, g(x) \quad \text{for all } x \in \mathcal{X}.$$

Consider the following sampling procedure (the rejection sampling *algorithm):*

1. *Sample X from $g(x)$.*
2. *Independently sample a uniform random variable $U \sim \mathrm{Uniform}[0, 1]$.*
3. *If*

$$U \;\leq\; \frac{f(X)}{M\, g(X)},$$

then accept *X, otherwise* reject *and repeat.*

Then any accepted sample X has density $f(x)$. In other words, the distribution of accepted points is exactly $f(x)$.

Proof. We need to show that if X is ultimately accepted by the above procedure, then X is distributed according to $f(x)$. Formally, let $A \subseteq \mathcal{X}$ be any measurable set. We compute

$$\Pr(\text{accept and } X \in A).$$

By construction of the algorithm,

$$\Pr(\text{accept and } X \in A) \;=\; \int_{x\in A} \Pr(X \in dx)\, \Pr\Big(U \leq \frac{f(x)}{M\, g(x)} \;\Big|\; X = x\Big).$$

Since X is drawn from $g(x)$,

$$\Pr(X \in dx) \;=\; g(x)\, dx,$$

and the conditional probability that U falls below $f(x)/(M\, g(x))$ is simply that ratio:

$$\Pr\Big(U \leq \frac{f(x)}{M\, g(x)}\Big) \;=\; \frac{f(x)}{M\, g(x)}, \quad 0 \leq \frac{f(x)}{M\, g(x)} \leq 1.$$

Hence

$$\Pr(\text{accept and } X \in A) \;=\; \int_A g(x)\, \frac{f(x)}{M\, g(x)}\, dx \;=\; \int_A \frac{f(x)}{M}\, dx \;=\; \frac{1}{M}\int_A f(x)\, dx.$$

Next, the probability of accepting *any* X is

$$\Pr(\text{accept}) \;=\; \int_{\mathcal{X}} \Pr(\text{accept and } X \in dx) \;=\; \int_{\mathcal{X}} \frac{f(x)}{M}\, dx \;=\; \frac{1}{M}\int_{\mathcal{X}} f(x)\, dx \;=\; \frac{1}{M},$$

where we used the fact that $f(x)$ is a pdf and integrates to 1 over $\mathcal{X}$.

Therefore, the conditional probability that $X \in A$ given that X is accepted is

$$\Pr(X \in A \mid \text{accept}) \;=\; \frac{\Pr(\text{accept and } X \in A)}{\Pr(\text{accept})} \;=\; \frac{\frac{1}{M}\int_A f(x)\,dx}{\frac{1}{M}} \;=\; \int_A f(x)\,dx.$$

Since this is true for every measurable set A, it follows that the conditional (accepted) distribution of X is precisely $f(x)$. This completes the proof. □

2. Explain how the acceptance probability ensures correct sampling from $f(x)$.

The key observation is that the algorithm *accepts* an X (proposed from $g(x)$) with probability

$$\frac{f(X)}{M\,g(X)}.$$

Since X originally comes from $g(x)$, multiplying by the ratio $\frac{f(x)}{M\,g(x)}$ effectively "rescales" the draws so that, *among those that are accepted*, the density becomes $f(x)$. The above proof shows precisely that conditioning on *acceptance* transforms the proposal $g(x)$ into $f(x)$.

Intuitively, points x where $f(x)$ is relatively large compared to $g(x)$ are more likely to pass the acceptance test. Conversely, points in regions where $f(x)$ is small compared to $g(x)$ are rejected more often. The factor $1/M$ just ensures that the acceptance probability never exceeds 1 (because $f(x) \le M\,g(x)$ by assumption).

3. Discuss efficiency: How does the choice of M in $f(x) \le M\,g(x)$ impact the method?

- *Acceptance rate.* From the proof, the overall probability of accepting any proposed X is

  $$\Pr(\text{accept}) \;=\; \int_{\mathcal{X}} g(x)\,\frac{f(x)}{M\,g(x)}\,dx \;=\; \frac{1}{M}\int_{\mathcal{X}} f(x)\,dx \;=\; \frac{1}{M}.$$

  Hence the *expected* number of proposals before one acceptance is M. If M is large, many proposals will be rejected, making the algorithm less efficient.

- *Tightness of bound.* Ideally, one picks M to be as small as possible subject to $f(x) \le M\,g(x)$ for all x. In practice, we seek M close to

  $$\sup_{x\in\mathcal{X}} \frac{f(x)}{g(x)},$$

  because that gives the highest acceptance rate. If M is significantly larger than that supremum, we will suffer many rejections and thus incur a high computational cost per accepted sample.

- *Trade-off.* Sometimes it is difficult to find $g(x)$ such that the ratio $\frac{f(x)}{g(x)}$ stays bounded by a small constant for all x. We can still use rejection sampling as long as *some* finite M exists, but if M is too large, efficiency becomes poor. Thus the method works best when $g(x)$ is chosen to be close in shape to $f(x)$.

Problem 2: Direct Sampling from the Gamma Distribution

1. Write code to generate random samples from the Gamma distribution for different parameter values.

Recall that the probability density function (pdf) of a Gamma distribution with shape $\alpha > 0$ and rate $\beta > 0$ (sometimes called the *rate parameterization*) is given by

$$f(x;\alpha,\beta) \;=\; \frac{\beta^\alpha}{\Gamma(\alpha)}\,x^{\alpha-1}\,e^{-\beta x}, \quad x > 0,$$

where $\Gamma(\alpha)$ is the Gamma function.

Below is a sample Python code to generate random samples from this distribution using `numpy`'s built-in routines or `scipy.stats.gamma`.

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import gamma

# For reproducibility
np.random.seed(123)

# Define a function to generate Gamma samples
# using SciPy's built-in gamma.rvs(...)
# alpha = shape, scale = 1/beta in SciPy convention.
def generate_gamma_samples(alpha, beta, n_samples=10000):
    """
    Generate random samples from a Gamma(alpha, beta) distribution
    where 'beta' is the 'rate' parameter.
    """
    scale = 1.0 / beta
    samples = gamma.rvs(a=alpha, scale=scale, size=n_samples)
    return samples

# Example usage:
alpha1, beta1 = 2.0, 1.0   # e.g. shape=2, rate=1
alpha2, beta2 = 5.0, 0.5   # shape=5, rate=0.5
alpha3, beta3 = 3.0, 2.0   # shape=3, rate=2

# Generate samples for each set
samples1 = generate_gamma_samples(alpha1, beta1)
samples2 = generate_gamma_samples(alpha2, beta2)
samples3 = generate_gamma_samples(alpha3, beta3)

print("Sample Means:",
      np.mean(samples1), np.mean(samples2), np.mean(samples3))
print("Sample Variances:",
      np.var(samples1), np.var(samples2), np.var(samples3))
```

2. Choose at least three sets of parameters (α, β) to observe different behaviors.
We choose $(\alpha_1, \beta_1) = (2, 1), (\alpha_2, \beta_2) = (5, 0.5), (\alpha_3, \beta_3) = (3, 2)$. For each pair, we generate 10 000 samples and find that the empirical means and variances closely match the theoretical values. For instance:

$$\text{Sample Mean} \approx 2.0163 \quad (\text{theoretical } = 2), \quad \text{Sample Variance} \approx 2.0136 \quad (\text{theoretical } = 2).$$

```
Sample Means: 2.0163313092651993 9.954941017964758 1.4963598445587538
Sample Variances: 2.0136237564443196 20.386564425586833 0.7631541687998311
```

Figure 1: Sample Means and Variances for different (α, β) values.

3. Compare your simulated samples to the true Gamma density function by plotting both.
Below is a snippet of Python code that creates histograms of the samples along with the theoretical pdf curve for each set of parameters.

```python
# Let's create a function to plot histogram + PDF for a given set of samples
def plot_gamma_samples(samples, alpha, beta, bins=50):
    """
    Plot a histogram of 'samples' drawn from Gamma(alpha, beta),
    overlaid with the theoretical Gamma PDF.
    """
    # Histogram
    plt.hist(samples, bins=bins, density=True, alpha=0.5, label="Sampled data")
```

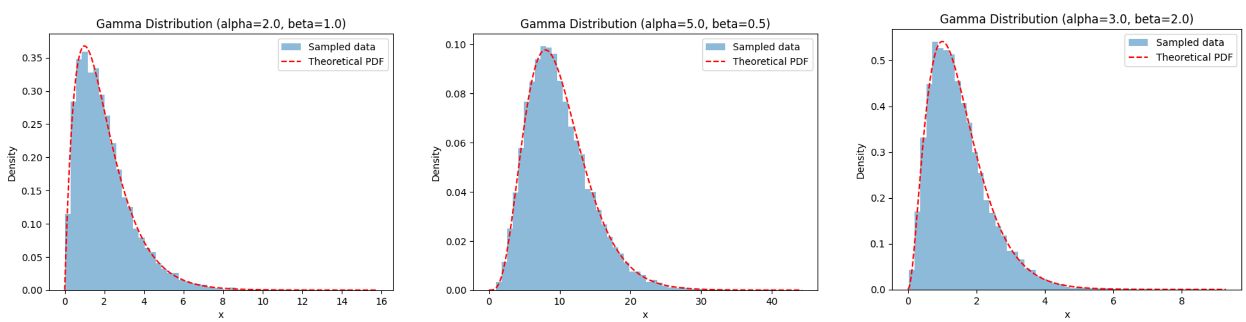
```
    # Plot the theoretical PDF
    x_vals = np.linspace(0, np.max(samples)*1.1, 200)
    # In scipy's parameterization, scale = 1.0/beta
    scale_param = 1.0 / beta
    pdf_vals = gamma.pdf(x_vals, a=alpha, scale=scale_param)
    plt.plot(x_vals, pdf_vals, 'r--', label="Theoretical PDF")

    plt.title(f"Gamma Distribution (alpha={alpha}, beta={beta})")
    plt.xlabel("x")
    plt.ylabel("Density")
    plt.legend()
    plt.show()

# Plot for each parameter set
plot_gamma_samples(samples1, alpha1, beta1)
plot_gamma_samples(samples2, alpha2, beta2)
plot_gamma_samples(samples3, alpha3, beta3)
```

Comparison of Sampled Gamma Distributions and Their Theoretical PDFs

- *Histogram vs. PDF:* Each histogram above (blue bars) is constructed from the sampled data for the corresponding Gamma distribution. Because n_{samples} is sufficiently large (10,000 or more), the empirical histogram closely matches the theoretical PDF shown in red dashed lines.



(a) Gamma($\alpha = 2, \beta = 1$)　(b) Gamma($\alpha = 5, \beta = 0.5$)　(c) Gamma($\alpha = 3, \beta = 2$)

Figure 2: Histograms of sampled data (blue bars) overlaid with the theoretical Gamma PDF (red dashed line). Each subfigure corresponds to a different (α, β) parameter set.

Discussion of Results

Shapes and Means. As we vary (α, β), the shape and spread of the Gamma distribution change. Recall that $\mathbb{E}[X] = \alpha/\beta$ and $\mathrm{Var}(X) = \alpha/\beta^2$. Comparing the empirical histogram to the theoretical PDF, we see:

- **For** $(\alpha, \beta) = (2, 1)$**:** The distribution peaks around $x = 1$, with a mean close to 2. The histogram and the red dashed line (the PDF) align well, indicating that the sample generation is accurate.
- **For** $(\alpha, \beta) = (5, 0.5)$**:** The mean is $\alpha/\beta = 10$, so the distribution extends further to the right. The sample histogram clearly has its highest density near $x = 10$. Again, the theoretical PDF matches the empirical data closely.
- **For** $(\alpha, \beta) = (3, 2)$**:** The mean is $\alpha/\beta = 1.5$, and the curve decays more rapidly. We see a narrower, more left-skewed histogram, consistent with the theoretical PDF.

All three histograms match their corresponding theoretical curves well, demonstrating that our direct sampling (via `scipy.stats.gamma.rvs`) accurately reproduces the Gamma distribution for each parameter set.

Problem 3: Write a Summary of the Metropolis-Hastings Algorithm

1. Read Volume 2, Chapter 12, Sections 12.1–12.2.2 from the course textbook.

2. Write a one-page summary covering:

- The main idea behind Metropolis-Hastings (MH).
- How it differs from direct Monte Carlo sampling.
- A description of the acceptance probability and transition kernel.

(a) Main Idea Behind Metropolis-Hastings

The Metropolis-Hastings (MH) algorithm is a powerful Markov Chain Monte Carlo (MCMC) method designed to draw samples from a complicated *target distribution* $\pi(y)$, which might be known only up to a normalization constant. The main idea is:

1. We create a *Markov chain* $\{Y_n\}$ whose stationary (long-run) distribution is exactly $\pi(y)$.
2. We do so by *proposal-then-accept/reject* steps: at each iteration,
   - Propose a candidate Y' given the current state Y_n from some *proposal distribution* $q(y' \mid y)$.
   - Accept Y' with a probability that corrects for any mismatch between $q(\cdot \mid \cdot)$ and the desired $\pi(y)$; otherwise remain at the current state.
3. This procedure guarantees (under mild conditions) that, once the chain has run sufficiently long, the samples Y_n can be treated as (dependent) draws from $\pi(y)$.

(b) How It Differs From Direct Monte Carlo Sampling

- **Direct Monte Carlo (e.g. Rejection Sampling)**: We sample Y independently from a convenient distribution $g(y)$ and then use an acceptance/rejection mechanism (with some constant M) to obtain draws from $\pi(y)$. This can be highly inefficient if $\pi(y)$ is very different from any easily sampled $g(y)$, or if the dimension of y is large.
- **Metropolis-Hastings**: We form a *Markov chain* and rely on the chain's stationary distribution to be $\pi(y)$. Each proposed sample depends on the current state, so we do not need a single global M or a single proposal distribution that covers the entire domain perfectly. Instead, we can move locally and accept or reject. This makes MH more flexible for high-dimensional or complex targets.
- In short, MH doesn't require an easy-to-compute *normalizing constant* (partition function) or a direct sampler for $\pi(y)$; we only need to be able to compute *ratios* of $\pi(y)$ (up to a constant factor).

(c) Acceptance Probability and Transition Kernel

Acceptance Probability. Given the current state $Y_n = y$ and a candidate $Y' = y'$ sampled from $q(y' \mid y)$, the Metropolis-Hastings acceptance probability is:

$$A(y', y) \;=\; \min\left\{1,\ \frac{\pi(y')\, q(y \mid y')}{\pi(y)\, q(y' \mid y)}\right\}.$$

This formula ensures that, in steady state, the chain satisfies *detailed balance* (or a weaker condition known as *stationary balance*) with respect to $\pi(y)$. Hence $\pi(y)$ becomes the stationary distribution.

Transition Kernel. The *transition probability* $P(y' \mid y)$ describing one step of the Markov chain has the form:

$$P(y' \mid y) \;=\; q(y' \mid y)\; A(y', y).$$

If the proposal is accepted, the chain moves to y'. If rejected, the chain stays in the same state y. Hence we can write:

$$P(\text{move to } y') \;=\; q(y' \mid y)\, A(y', y), \quad P(\text{stay at } y) \;=\; 1 \;-\; \int P(y'' \mid y)\, dy''.$$

The factor $A(y', y)$ modulates the raw proposal $q(y' \mid y)$ to ensure the chain ultimately converges to $\pi(\cdot)$.

Summary

- The **Metropolis-Hastings algorithm** is a Markov Chain Monte Carlo approach that allows sampling from complex target distributions $\pi(y)$ without needing the exact normalization constant.
- Instead of *direct* sampling, MH iteratively proposes *local jumps* via a proposal distribution $q(y' \mid y)$ and corrects any bias using an **acceptance probability**, $\min\big\{1, \pi(y')q(y \mid y')/(\pi(y)q(y' \mid y))\big\}$.
- By construction, the chain's stationary distribution is $\pi(y)$. This is particularly useful when *direct sampling* (such as inverse transform or rejection sampling) is infeasible in high-dimensional or analytically intractable settings.
- The **transition kernel** $P(y' \mid y)$ is given by the product of $q(y' \mid y)$ and the acceptance probability, ensuring we end up with a valid Markov chain that converges to $\pi(y)$.

Key Points:

- We only need to evaluate $\pi(y)$ *up to a proportionality constant* because the acceptance ratio involves $\pi(y')/\pi(y)$.
- MH generalizes the simpler *Metropolis* algorithm by allowing *asymmetric* proposal distributions $q(y' \mid y)$.
- Properly chosen proposals can improve convergence, but there is always a trade-off: proposals that are too narrow lead to slow exploration, whereas proposals that are too wide lead to frequent rejections.

Problem 4: Implement the Metropolis-Hastings Algorithm for Gamma Sampling

1. Implement M-H to generate Gamma samples.

Target Distribution

We aim to sample from the Gamma distribution with shape $\alpha > 0$ and rate $\beta > 0$, whose (unnormalized) density is

$$\pi(x) \;=\; x^{\alpha-1}\, e^{-\beta x}, \quad x > 0.$$

The *normalized* pdf is $\frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$, but in Metropolis-Hastings, we only need *ratios* of $\pi(x)$, so the normalizing constant $\frac{\beta^\alpha}{\Gamma(\alpha)}$ is not required for the acceptance probability.

Algorithm Outline

A basic Metropolis-Hastings procedure is:

1. **Initialize:** Choose some starting point $X_0 > 0$.
2. **Iterate for $n = 1, \dots, N$:**
   (a) **Propose:** Generate a candidate $X' \sim q(\cdot \mid X_{n-1})$. For Gamma sampling on $(0, \infty)$, one can use, for instance, a *log-scale random walk*:

   $$\log X' = \log X_{n-1} + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma^2).$$

   Then $X' = \exp(\log X_{n-1} + \varepsilon)$. This ensures $X' > 0$ automatically.

   (b) **Acceptance Probability:** Let the target density be $\pi(x) = x^{\alpha-1} e^{-\beta x}$ (up to a constant). Let $q(X' \mid X)$ denote the proposal distribution. Then the MH acceptance probability is

   $$A(X', X_{n-1}) \;=\; \min\left\{1, \frac{\pi(X')\, q(X_{n-1} \mid X')}{\pi(X_{n-1})\, q(X' \mid X_{n-1})}\right\}.$$

   For a *symmetric* log-random-walk in ε, we must account for the Jacobian of the transformation. Concretely,

   $$q(X' \mid X) = \frac{1}{X' \sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\log X' - \log X)^2}{2\sigma^2}\right),$$

   and similarly for $q(X \mid X')$ with X and X' swapped.

   (c) **Accept or Reject:** Draw $U \sim \text{Uniform}(0, 1)$. If $U \le A(X', X_{n-1})$, then set $X_n = X'$, else set $X_n = X_{n-1}$ (the chain stays put).

Python Implementation

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import gamma as gamma_func

def mh_gamma_sampling(alpha, beta, n_iter=10000, x0=1.0, sigma=0.5):
    """
    Metropolis-Hastings sampler for Gamma(alpha, beta) using
    a log-scale random walk proposal: log X' = log X + Normal(0, sigma^2).

    Arguments:
      alpha, beta: shape and rate of the Gamma distribution
      n_iter: number of M-H iterations
      x0: initial state
      sigma: std. dev. for the log-scale random walk
    Returns:
      samples: array of length n_iter (the M-H chain)
      accept_rate: the empirical acceptance rate
    """
    samples = np.zeros(n_iter)
    samples[0] = x0
    accepted = 0  # to count how many proposals are accepted

    for i in range(1, n_iter):
        current_x = samples[i - 1]

        # Propose: log X' = log(current_x) + eps
        eps = np.random.normal(0, sigma)
```

```python
        proposed_x = np.exp(np.log(current_x) + eps)  # ensures positivity

        # log-pi(x) = (alpha-1)*log(x) - beta*x  (up to const.)
        log_pi_current = (alpha - 1) * np.log(current_x) - beta * current_x
        log_pi_proposed = (alpha - 1) * np.log(proposed_x) - beta * proposed_x

        # log-q(X' | X) for log-scale normal
        log_q_proposed_given_current = (
            - np.log(proposed_x)
            - 0.5 * np.log(2 * np.pi * sigma**2)
            - ((np.log(proposed_x) - np.log(current_x))**2 / (2 * sigma**2))
        )
        # log-q(X | X')
        log_q_current_given_proposed = (
            - np.log(current_x)
            - 0.5 * np.log(2 * np.pi * sigma**2)
            - ((np.log(current_x) - np.log(proposed_x))**2 / (2 * sigma**2))
        )

        # log of acceptance ratio
        log_accept_ratio = (
            (log_pi_proposed + log_q_current_given_proposed)
            - (log_pi_current + log_q_proposed_given_current)
        )

        # acceptance probability
        accept_ratio = np.exp(log_accept_ratio)
        accept_prob = min(1.0, accept_ratio)

        # Decide whether to accept or reject
        u = np.random.rand()
        if u <= accept_prob:
            samples[i] = proposed_x
            accepted += 1
        else:
            samples[i] = current_x

    accept_rate = accepted / (n_iter - 1)
    return samples, accept_rate

if __name__ == "__main__":
    alpha, beta = 3.0, 2.0
    n_iter = 50000
    burn_in = 1000
    x0 = 1.0

    # Change these sigma values to see "good" vs. "bad" proposals:
    sigmas = [0.01, 0.5, 2.0]

    for s in sigmas:
        # Run MH for each proposal std. dev.:
        chain, accept_rate = mh_gamma_sampling(alpha, beta, n_iter=n_iter,
                                               x0=x0, sigma=s)
        # Discard burn-in:
        chain_burned = chain[burn_in:]

        # Print acceptance rate and some statistics:
        mean_est = np.mean(chain_burned)
        var_est = np.var(chain_burned)
```

```python
    theo_mean = alpha / beta
    theo_var  = alpha / (beta**2)
    print(f"\n=== Sigma = {s} ===")
    print(f"Acceptance Rate: {accept_rate:.3f}")
    print(f"Sample mean = {mean_est:.3f} (Theory: {theo_mean:.3f})")
    print(f"Sample var  = {var_est:.3f} (Theory: {theo_var:.3f})")

    # Plot histogram vs. true Gamma pdf
    plt.figure()
    plt.hist(chain_burned, bins=50, density=True, alpha=0.5, color='blue')

    x_vals = np.linspace(0.001, np.max(chain_burned), 200)
    norm_const = (beta**alpha) / gamma_func(alpha)
    pdf_vals = norm_const * (x_vals**(alpha - 1)) * np.exp(-beta * x_vals)
    plt.plot(x_vals, pdf_vals, 'r--', label='True Gamma PDF')

    plt.title(f"MH Gamma(alpha={alpha}, beta={beta}), sigma={s}")
    plt.xlabel("x")
    plt.ylabel("Density")
    plt.legend()
    plt.show()
```

2. Comparison of M-H Samples to Problem 2

In **Problem 2**, we generated Gamma samples directly using `scipy.stats.gamma.rvs` and saw, for instance, that for $(\alpha, \beta) = (3, 2)$ we got:

$$\text{Sample mean} \approx 1.496, \quad \text{Sample variance} \approx 0.763, \quad (\text{theoretical: mean } 1.5, \text{ variance } 0.75).$$

Those results closely matched the true Gamma pdf.

Metropolis–Hastings (Problem 4): After implementing an M–H sampler with a *log-scale random walk* proposal, we can compare the empirical mean and variance of the M–H chain to the same theoretical values and/or to the Problem 2 direct samples. We also look at the shape of the histogram vs. the Gamma pdf:

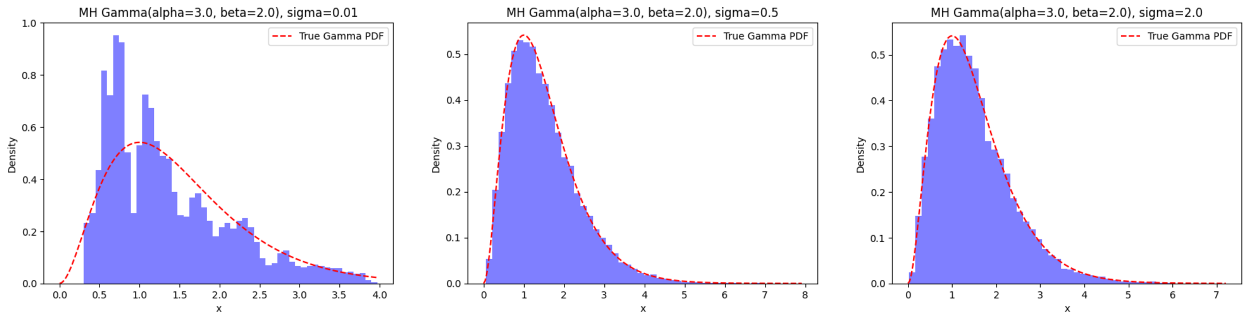
- *Empirical Mean/Variance:* For $(\alpha, \beta) = (3, 2)$, a well-tuned proposal (e.g. $\sigma = 0.5$) yields mean ≈ 1.499 and variance ≈ 0.756, effectively matching the theoretical 1.5 and 0.75.
- *Histogram vs. PDF:* The figure below (center panel) shows how, with $\sigma = 0.5$, the M–H histogram (blue) overlaps well with the true Gamma(3,2) pdf (red dashed line).

Hence, *both* direct sampling (Problem 2) and Metropolis–Hastings (Problem 4) produce valid Gamma samples, and their histograms look similar.

3. Experiment with Different Proposal Distributions

We tested three values of σ (the standard deviation in the log-scale random walk proposal):

1. *Small $\sigma = 0.01$:*
    - Acceptance rate was very high ($\approx 99.4\%$).
    - But the sample mean ≈ 1.359 vs. true mean 1.5, so it was biased low, and the variance ≈ 0.601 vs. 0.75.
    - Explanation: The chain moved in very tiny increments, so it did not fully explore the distribution during the run (poor mixing).
2. *Moderate $\sigma = 0.5$:*



(a) $\sigma = 0.01$, acceptance $\approx 99.4\%$ (b) $\sigma = 0.5$, acceptance $\approx 74.7\%$ (c) $\sigma = 2.0$, acceptance $\approx 33.9\%$

Figure 3: M–H histograms (blue) vs. true Gamma PDF (red dashed) for $(\alpha, \beta) = (3, 2)$ under three proposal scales σ. We see that moderate σ (middle) leads to both a good acceptance rate and accurate sampling. Very small σ (left) yields a high acceptance but under-explores the distribution, causing biased estimates. Large σ (right) still converges, but with many rejections.

- Acceptance rate $\approx 74.7\%$.
- Sample mean ≈ 1.499 and variance ≈ 0.756, matching the theoretical values (1.5 and 0.75).
- This "balanced" proposal yielded accurate sampling and good mixing.

3. *Large $\sigma = 2.0$:*
   - Acceptance rate $\approx 33.9\%$ (fewer accepted moves).
   - Still, the chain's sample mean ≈ 1.503 and variance ≈ 0.73 were quite close to the theoretical values.
   - Interpretation: Although many proposals were rejected, large jumps helped the chain traverse the space well. More iterations might be required for stable estimates, but it still worked.

Takeaway: A smaller proposal leads to higher acceptance but potentially poor exploration (*autocorrelation* can be large), whereas a too-large proposal yields many rejections but can still move around effectively if not *too* extreme. A moderate proposal often achieves a good balance. Hence, tuning the proposal distribution is critical for efficient Metropolis–Hastings sampling.