.

Variance reduction techniques and MCMC

Outline

1. Antithetic variables
2. Control variates
3. Stratified sampling
4. Importance sampling
5. Markov chains
6. Metropolis-Hastings
7. Gibbs sampling

Antithetic Variables

Simulation techniques are often employed in statistics and computational mathematics to estimate a parameter or a characteristic of a distribution. In this context, we focus on the estimation of a parameter θ, which represents the expected value (mean) of a random variable. This expected value is denoted as $\theta = E[X]$, where X is a random variable.

To estimate θ, we generate a sequence of observations of X, denoted as $X_1, X_2, \dots, X_n$.

The estimate of θ is then the sample average given by

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i.$$

If the estimator $\overline{X}$ is unbiased, which means that its expected value equals the true mean θ, the Mean Square Error (MSE) of the estimate is equal to its variance.

The MSE is defined as

$$MSE\left[\overline{X}\right] = E\left[\left(\overline{X} - \theta\right)^2\right] = \frac{1}{n^2} Var\left[\sum_{i=1}^{n} X_i\right].$$

If the observations X_i are independent and identically distributed (i.i.d.), the MSE simplifies to $\frac{Var[X]}{n}$.

Now consider a situation where we simulate $n/2$ bivariate random variables (X_i, Y_i), each with the same distribution. The distribution of all the marginals X and Y is the same, and $E[X] = \theta$. In this case, $Cov(X_i, Y_i)$ is constant for all i, but can differ from zero, while $Cov(X_i, X_j) = Cov(Y_i, Y_j) = 0$ for $i \neq j$.

The variance of the average of these bivariate random variables is

$$\frac{1}{(n/2)^2} Var\left(\sum_{i=1}^{n/2} \frac{X_i + Y_i}{2}\right) = \frac{Var(X) + 2Cov(X,Y) + Var(Y)}{2n} = \frac{Var(X) + Cov(X,Y)}{n}.$$

If $Cov(X, Y) < 0$, indicating negative correlation, the variance of this estimator is less than the variance of the estimator derived from n independent random variables. This property can be exploited to reduce variance in simulation studies, a technique known as the use of *antithetic variables.*

Consider X as a function of k independent random numbers $U_1, U_2, \ldots, U_k$, each uniformly distributed in the interval $(0, 1)$. That is,

$$X = h\left(U_1, U_2, \ldots, U_k\right),$$

where h is some transformation function. Since each U_i is uniformly distributed, $1 - U_i$ is also uniformly distributed in $(0, 1)$.

Thus, we can define a new variable Y as

$$Y = h\left(1 - U_1, 1 - U_2, \ldots, 1 - U_k\right),$$

which follows the same distribution as X.

Furthermore, under certain conditions such as monotonicity of the transformation function h, the random variables X and Y are negatively correlated. This negative correlation can be beneficial in reducing the variance of the estimator in simulation studies.

Example Antithetic variables

$$\int_0^1 e^x dx = e - 1 = 1.718282$$

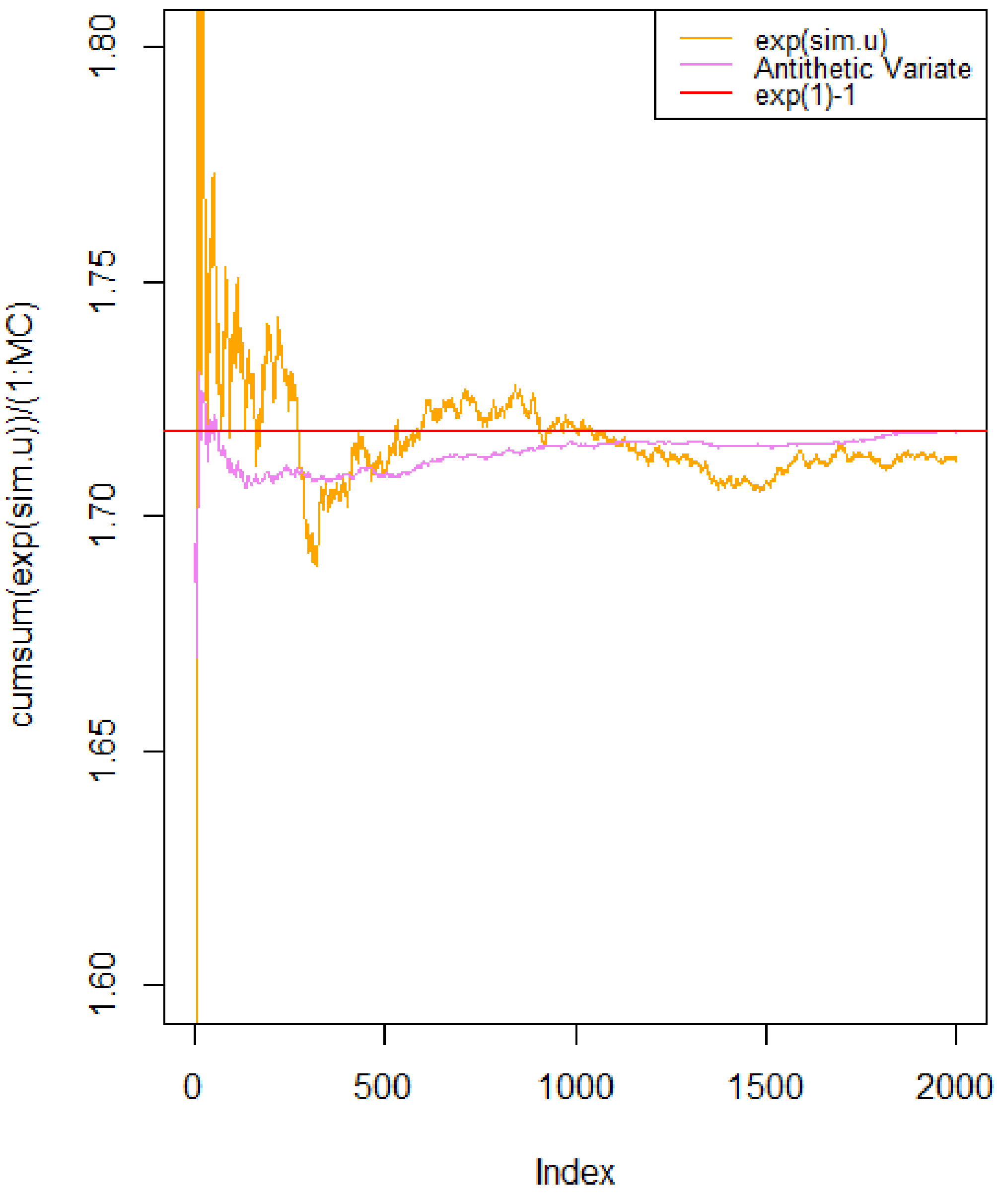
```r
set.seed(1)
MC = 2000

sim.u = runif(MC)
sim.exp = exp(sim.u)
sim.ant = (exp(sim.u[1:(MC/2)]) + exp(1-sim.u[1:(MC/2)]))/2

mean(sim.exp); sd(sim.exp)/sqrt(MC)
```

```
[1] 1.711716
[1] 0.01114951
```

```r
mean(sim.ant); sd(sim.ant)/sqrt(MC/2)
```

```
[1] 1.718079
[1] 0.001992196
```

```r
plot(cumsum(exp(sim.u))/(1:MC),type="l",ylim=c(1.6,1.8), col="orange")

bytwo = seq(2,MC,by=2)

points(bytwo,cumsum(sim.ant)/(1:(MC/2)), type="l", col="violet")
abline(h=exp(1)-1,col="red")

legend("topright", legend=c("exp(sim.u)", "Antithetic Variate", "exp(1)-1"),
col=c("orange", "violet", "red"), lty=1, cex=0.8)
```



Antithetic variables

- If the basic ingredient of our simulation algorithm is $U \sim U(0,1)$, use the pair of negatively correlated and identically distributed rvs $(U,\ 1-U)$.
- If the basic ingredient of our simulation algorithm is $X \sim N(\mu,\sigma)$, use the pair of negatively correlated and identically distributed variables $(X,\ 2\mu - X)$.
- If the basic ingredient of our simulation algorithm has cdf F, use the pair of negatively correlated

and identically distributed rvs $(F^{-1}(U),\ F^{-1}(1-U))$.

Control Variates

In statistical simulation, our primary goal is often to estimate a certain parameter.

Consider $\theta = E[X]$, where X is a random variable that we can simulate. The expected value of X, denoted as $E[X]$, is a measure of the central tendency or the average value of rv X.

However, a direct simulation of X might result in a high variance in the estimation of θ. To address this, the method of control variates is employed. This method involves using an additional rv Y, which is known to be (negatively) correlated with X. The mean of Y is given by $E[Y] = \mu_Y$.

The key idea of control variates is to construct a new estimator that has a lower variance than the original estimator. For any constant c, consider the new estimator $X + c(Y - \mu_Y)$. The mean of this new estimator remains the same as the original, which is θ, but its variance is altered.

The variance of the new estimator is given by:

$$\mathrm{Var}\left[X + c(Y - \mu_Y)\right] = \mathrm{Var}\left[X\right] + c^2\mathrm{Var}\left[Y\right] + 2c\mathrm{Cov}(X,Y),$$

where $\mathrm{Cov}(X,Y)$ represents the covariance between X and Y. This variance is minimized for a specific value of c, denoted as c^*, which is calculated as:

$$c^* = \frac{|\mathrm{Cov}(X,Y)|}{\mathrm{Var}\left[Y\right]}.$$

By substituting c^* in the variance formula, the resulting minimized variance of the new estimator becomes:

$$\mathrm{Var}\left[X + c^*(Y - \mu_Y)\right] = \mathrm{Var}\left[X\right] - \frac{\mathrm{Cov}(X,Y)^2}{\mathrm{Var}\left[Y\right]}.$$

Control Variates (Example)

Consider the problem of estimating the integral

$$\int_0^1 e^x dx = e - 1 \approx 1.718282,$$

This integral can be estimated by simulating e^U, where U is uniformly distributed over the interval $[0,1]$. As a control variate, we can use the integral

$$\int_0^1 (1-x)dx,$$

which equals 1/2.

This integral can be estimated by simulating $1 - U$, which is also uniformly distributed on $[0, 1]$. In particular, the random variables e^U and $1 - U$ are negatively correlated, making $1 - U$ a suitable control variate to reduce variance in estimation of $\int_0^1 e^x dx$.

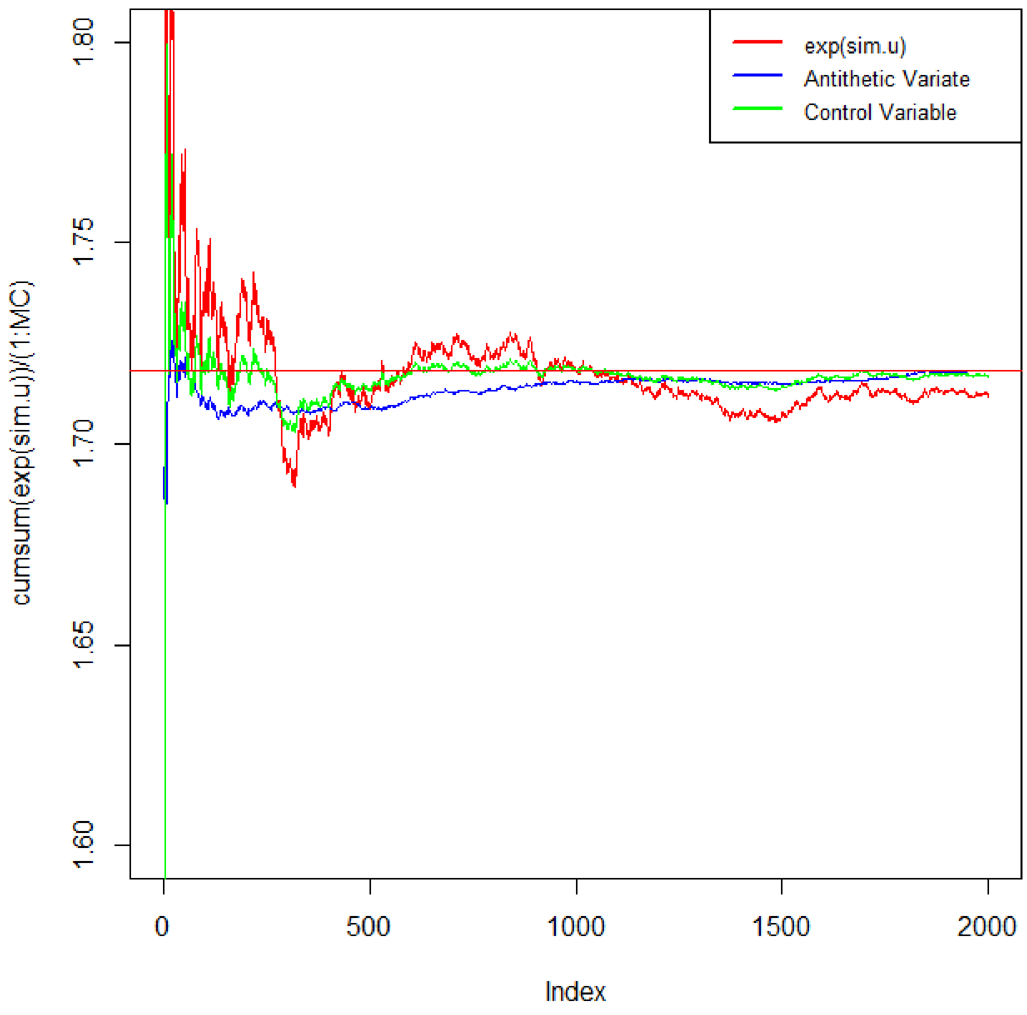
```
sim.cv = exp(sim.u) + (1 - sim.u - 0.5)
mean(sim.cv)
```

```
[1] 1.716722
```

```
sd(sim.cv)/sqrt(MC)
```

```
[1] 0.004738436
```

```
plot(cumsum(exp(sim.u))/(1:MC),type="l", ylim=c(1.6,1.8), col="red")
points(bytwo,cumsum(sim.ant)/(1:(MC/2)), type="l", col="blue")
points(cumsum(sim.cv)/(1:MC),type="l",col="green")
abline(h=exp(1)-1, col="red")

legend("topright", legend=c("exp(sim.u)", "Antithetic Variate",
      "Control Variable"), col=c("red", "blue", "green"), lwd=2, lty=1, cex=0.8)
```



exp(sim.u)
Antithetic Variate
Control Variable
cumsum(exp(sim.u))/(1:MC)
1.60
1.65
1.70
1.75
1.80
0
500
1000
1500
2000
Index

Stratified Sampling

Stratified sampling is a statistical method used to estimate a population parameter, such as the mean, by dividing the population into different subgroups or *strata* that share similar characteristics. This method is particularly useful when certain strata are expected to differ significantly with respect to the variable being measured.

Our goal is to estimate $\theta = E[X]$, the expected value of a variable X.

Consider the following set-up for our stratified sampling approach:

- There is a discrete variable Y with support $\{y_1, y_2, \ldots, y_k\}$, representing the different strata. Each y_i is a possible value of Y, and the distribution of Y is known, where $P(Y = y_i) = p_i$ is the probability that the stratum i occurs.

- For each stratum $i = 1, \ldots, k$, we can simulate or observe the conditional distribution of X given $Y = y_i$, denoted as $X|Y = y_i$.

The **stratified sampling estimator** of θ is constructed as follows:

- For each stratum $i = 1, \ldots, k$, take np_i observations of $X|Y = y_i$. Here, n represents the total number of observations in all strata.

- The estimator for θ is then given by
$$\hat{\theta} = \sum_{i=1}^{k} \overline{X}_i p_i,$$
where $\overline{X}_i$ is the mean of the sample of X over the np_i instances where $Y = y_i$.

This estimator leverages stratification to potentially reduce variance in the estimation of θ compared to simple random sampling.

Justification of the Estimator:

The stratified sampling estimator is unbiased. This can be seen through the iterated expectation formula, as follows:
$$E\left[\sum_{i=1}^{k} \overline{X}_i p_i\right] = \sum_{i=1}^{k} E\left[\overline{X}_i\right] p_i = \sum_{i=1}^{k} E\left[X|Y = y_i\right] p_i = E[X].$$
This equality shows that the expected value of our estimator is equal to the expected value of X, which is our target θ.

Variance of the Stratified Sampling Estimator

To understand the efficiency of stratified sampling, we examine the variance of the estimator:

$$\begin{aligned} Var\left[\sum_{i=1}^{k} \overline{X}_i p_i\right] &= \sum_{i=1}^{k} p_i^2 Var\left[\overline{X}_i\right] \\ &= \sum_{i=1}^{k} p_i^2 \frac{1}{n_i} Var\left[X|Y=y_i\right] \\ &= \sum_{i=1}^{k} p_i^2 \frac{1}{np_i} Var\left[X|Y=y_i\right] \\ &= \frac{1}{n} E\left[Var\left[X|Y\right]\right]. \end{aligned}$$

This formula indicates that the variance of our estimator depends on the variances within the strata of X, scaled by the probabilities of the strata.

Intuitively, this expression appears because within each stratum $Y = y_i$, we sample n_i points, and the total sample size is n. The key idea is that stratified sampling removes part of the variance by ensuring each sub-population (stratum) is sampled proportionally to its size.

The conditional variance formula is:

$$Var[X] = E[Var[X \mid Y]] + Var[E[X \mid Y]].$$

If $\overline{X}$ is the mean of a simple random sample of size n from the entire population, we have

$$Var[\overline{X}] = \frac{1}{n} Var[X],$$

where X is a generic random draw from the population.

Putting these together, we will compare the two variances.

Applying the conditional variance decomposition to $Var[X]$:

$$Var[X] = E[Var[X \mid Y]] + Var[E[X \mid Y]].$$

Hence,

$$Var[\overline{X}] = \frac{1}{n}(E[Var[X \mid Y]] + Var[E[X \mid Y]]).$$

We have already arrived at

$$Var\left[\sum_{i=1}^{k} \overline{X}_i p_i\right] = \frac{1}{n} E[Var[X \mid Y]].$$

Hence,

$$\begin{aligned} Var[\overline{X}] - Var\left[\sum_{i=1}^{k} \overline{X}_i p_i\right] &= \frac{1}{n}\left(E[Var[X \mid Y]] + Var[E[X \mid Y]]\right) - \frac{1}{n} E[Var[X \mid Y]] \\ &= \frac{1}{n}\left(E[Var[X \mid Y]] + Var[E[X \mid Y]] - E[Var[X \mid Y]]\right) \\ &= \frac{1}{n} Var[E[X \mid Y]]. \end{aligned}$$

Because $Var[E[X \mid Y]] \geq 0$, it follows that

$$Var[\overline{X}] \geq Var\left[\sum_{i=1}^{k} \overline{X}_i p_i\right],$$

with equality if and only if the conditional means $E[X \mid Y]$ do not actually vary across strata (i.e., if $E[X \mid Y]$ is essentially constant).

This result implies that the variance of the stratified sampling estimator is always less than or equal to the variance of the simple random sampling estimator. This demonstrates the efficiency gain of stratified sampling over simple random sampling, especially when the conditional means of X given Y vary significantly between strata.

The essence of stratified sampling's efficiency lies in its ability to minimize variance within each stratum. Since each stratum is more homogeneous than the population at large, the variance within each stratum $Var\left[X|Y = y_i\right])$ is lower than the variance in the population.

By averaging these variances $E\left[Var\left[X|Y\right]\right]$ and adjusting for the total sample size, the formula captures the overall reduction in variance achieved through stratified sampling.

Example

In the book by Hossack, Pollard, and Zehnwirth (Cambridge University Press, 2003), the table presented below illustrates both the counts and amounts of claims.

This R program models insurance claim amounts based on a predetermined probability distribution, simulates a large number of claims according to this model, and calculates the mean and the variance of these simulated claims.

Claim amount (€)	**Counts**
0-50	1728
50-100	1346
100-200	1869
200-400	1822
400-800	1056
Total	**7821**

```r
p = c(1728,1346,1869,1822,1056)/7821

sim.claim = function(n) {
   claim = runif(n,min=400,max=800)
   sim.u = runif(n)
   bin1 = which(sim.u<p[1])

   claim[bin1] = runif(length(bin1), min=0, max=50)
   bin2 = which(sim.u>p[1] & sim.u<sum(p[1:2]))
   claim[bin2] = runif(length(bin2),min=50,max=100)
   bin3 = which(sim.u>sum(p[1:2]) & sim.u<sum(p[1:3]))
   claim[bin3] = runif(length(bin3),min=100,max=200)
   bin4 = which(sim.u>sum(p[1:3]) & sim.u<sum(p[1:4]))
   claim[bin4] = runif(length(bin4),min=200,max=400)

return(claim)
}

sclaim = sim.claim(7821)

c(mean(sclaim),var(sclaim)/7821)
```

```
[1] 203.902862   4.736561
```

Now, we continue the simulation of insurance claims but the approach is different, calculating the expected mean and variance of the claims using a more direct method and then comparing it to a sampling approach.

The second part uses a large sample from a simplified model of the distribution to calculate variance, serving as a comparison or validation for the detailed simulation.

```r
# We create a list, sclaim, consisting of five separate vectors.

# Each vector simulates claim amounts within specific ranges based on
# the probabilities defined in p.

# The length of each vector corresponds to the expected number of claims
# in that range, determined by multiplying each probability in p by the
# total number of simulations (7821).

# The first vector simulates claims between 0 and 50, the second between
# 50 and 100, and so on, up to the fifth vector simulating claims
# between 400 and 800.

sclaim = list(runif(p[1]*7821,min=0,max=50),
   runif(p[2]*7821,min=50,max=100),
   runif(p[3]*7821,min=100,max=200),
   runif(p[4]*7821,min=200,max=400),
   runif(p[5]*7821,min=400,max=800)
   )

# The mean of each vector within sclaim is calculated and then
# weighted by its corresponding probability in p.

# This weighted sum of means is a way to calculate the overall expected
# mean claim amount considering the distribution of claim amounts
# across the different ranges

(p[1]*mean(sclaim[[1]])+p[2]*mean(sclaim[[2]])+p[3]*mean(sclaim[[3]]) +
p[4]*mean(sclaim[[4]])+p[5]*mean(sclaim[[5]]))
```

```
[1] 205.3114
```

```r
# The variance of each vector within sclaim is calculated and then weighted
# by its corresponding probability in p.

# This weighted sum of variances is then divided by the total number of
# simulations (7821) to standardize it.

# This approach calculates the overall expected variance of claim amounts,
# taking into account the variability within each claim amount range.

(p[1]*var(sclaim[[1]]) + p[2]*var(sclaim[[2]]) + p[3]*var(sclaim[[3]]) +
p[4]*var(sclaim[[4]]) + p[5]*var(sclaim[[5]]))/7821
```

```
[1] 0.370737
```

Alternative:

```r
calculateClaimsStatistics = function(p, n) {
  # Define the ranges for the claim amounts
  ranges = list(c(0, 50), c(50, 100), c(100, 200), c(200, 400), c(400, 800))

  # Generate sclaim list based on probabilities and ranges
  sclaim = lapply(1:5, function(i) runif(p[i]*n, min=ranges[[i]][1],
  max=ranges[[i]][2]))

  # Calculate weighted mean of claim amounts
  weightedMean = sum(sapply(1:5, function(i) p[i] * mean(sclaim[[i]])))

  # Calculate standardized weighted sum of variances
  standardizedVariance = sum(sapply(1:5, function(i) p[i]*var(sclaim[[i]])))/n

  # Return a list containing both statistics
  return(list(weightedMean = weightedMean,
  standardizedVariance = standardizedVariance))
}

# Example usage:
p = c(0.1, 0.15, 0.2, 0.25, 0.3) # Example probabilities vector
n = 7821 # Total number of simulations
result = calculateClaimsStatistics(p, n)

# Accessing the results
result$weightedMean
result$standardizedVariance
```

```
> result$weightedMean
[1] 298.8201
> result$standardizedVariance
[1] 0.6734312
```

```
# Now we simulate the variance of a large sample (100000 data points) drawn
# from a discrete distribution with specified values (c(25,75,150,300,600))
# that likely represent the average claim amounts for each range.

# The probabilities for each of these values are the same as in p, and
# sampling is done with replacement. The variance of this sample is then divided
# by 7821 for standardization.

var(sample(100000, x=c(25,75,150,300,600), prob=p, replace=T))/7821
```

```
[1] 4.320348
```

This approach provides a way to compare the variance obtained from a large, discrete sample with the expected variance calculated from the individual ranges.

Example

$$\int_0^1 e^x dx = e - 1 = 1.718282$$

```
set.seed(1)
# Set the number of Monte Carlo simulations and the number of strata
numSimulations = 2000
numStrata = 10

# Generate uniformly distributed random numbers
uniformRandomNumbers = runif(numSimulations)

# Stratify the uniformly distributed data with a shift and scale transformation
stratifiedData = exp(uniformRandomNumbers / numStrata + seq(0, 0.9, by = 0.1))

# Initialize vectors to store means and variances for each stratum
```

```r
meansPerStratum = numeric(length = numStrata)
variancesPerStratum = numeric(length = numStrata)

# Loop through each stratum to calculate mean and variance
for (stratumIndex in 1:numStrata) {

  # Create a logical vector to select elements from the current stratum
  stratumSelector = rep(0:(numStrata - 1) == (stratumIndex - 1),
  (numSimulations/numStrata) )
  # generates a logical vector that repeats (numSimulations/numStrata)
  # which is used for indexing to select elements belonging to the ith stratum.

  # Calculate and store the mean and variance for the current stratum
  meansPerStratum[stratumIndex] = mean(stratifiedData[stratumSelector])
  variancesPerStratum[stratumIndex] = var(stratifiedData[stratumSelector])
}

# Calculate the overall mean of the stratum means
overallMean = mean(meansPerStratum)

# Calculate the square root of the mean of the stratum variances divided
# by the number of simulations
overallStandardError = sqrt(mean(variancesPerStratum) / numSimulations)

print(overallMean)
print(overallStandardError)
```

```
[1] 1.717646
[1] 0.001166053
```

Importance Sampling

Importance Sampling is a statistical technique used for variance reduction in Monte Carlo simulations. It is particularly useful when dealing with rare events or in scenarios where the quantity of interest has a low probability of occurrence.

Consider a scenario where we aim to estimate a small probability p. The relative standard error of the estimator $\widehat{p}_n$ is given by

$$\frac{\sigma(\widehat{p}_n)}{p} = \sqrt{\frac{p(1-p)}{n}} \cdot p^{-1} \approx \frac{1}{\sqrt{np}}.$$

For example, if $p = 10^{-6}$, then the relative standard error becomes $\frac{10^3}{\sqrt{n}}$, which indicates a significant error for small sample sizes.

To address such issues, we introduce the concept of *importance sampling.*

Consider a random variable X with a distribution that has a density (or mass) function $f(x)$. Our goal is to estimate the expected value

$$\theta = E[h(X)] = \int h(x)f(x)dx.$$

To achieve this using importance sampling, we choose an instrumental random variable Y with density function $g(y)$ satisfying the following condition: $f(x) = 0$ whenever $g(x) = 0$, such that g has a form similar to that of the product $h \cdot f$.

This ensures that the support of $f(x)$ is contained within the support of $g(x)$.

Let X be a random variable with density $f(x)$. We want to compute $\theta = E_f[h(X)]$. Choose another density $g(x)$ (with $g > 0$ wherever $f > 0$) and set $\omega(x) = \frac{f(x)}{g(x)}$.

Then

$$\theta = \int h(x)f(x)\mathrm{d}x = \int h(x)\omega(x)g(x)\mathrm{d}x = E_g\big[\omega(Y)h(Y)\big].$$

Thus, if $Y_1, \ldots, Y_n$ are i.i.d. from g, an unbiased estimator is

$$\widehat{\theta}_{IS} = \frac{1}{n}\sum_{i=1}^{n} \omega(Y_i)h(Y_i).$$

This is called an **importance sampling** estimator.

Variance:

$$\mathrm{Var}\big[\widehat{\theta}_{IS}\big] = \frac{1}{n}\mathrm{Var}_g\big[\omega(Y)h(Y)\big] = \frac{1}{n}\Big(E_g\big[\omega(Y)^2h(Y)^2\big] - \big(E_g[\omega(Y)h(Y)]\big)^2\Big).$$

Because $E_g[\omega(Y)h(Y)] = \theta$, we get

$$\mathrm{Var}\big[\widehat{\theta}_{IS}\big] = \frac{1}{n}\Big(\int h(x)^2\frac{f(x)^2}{g(x)}\mathrm{d}x - \theta^2\Big).$$

For comparison, sampling directly from f yields a variance of $\frac{1}{n}\big(E_f[h(X)^2] - \theta^2\big)$.

The integrand has the factor $\frac{f(x)^2}{g(x)}$. Intuitively:

- Where $h(x)^2f(x)^2$ is large, we want $g(x)$ to be large as well, so that $\frac{f(x)^2}{g(x)}$ stays small and does not blow up.

- Where $h(x)^2f(x)^2$ is small, it is less harmful for $g(x)$ to be smaller.

Hence putting more mass in g where the integrand is most significant tends to reduce the overall variance. This is often described by the rule of making $\omega(y)$ small in regions where $h(y)^2$ (or more precisely $|h(y)f(y)|$) is large.

Importance sampling (Example)

The goal is to calculate

$$P(X > 4.5) \quad \text{where} \quad X \sim N(0,1).$$

```
# Calculate P(X > 4.5) where X ~ N(0,1)
1 - pnorm(4.5)
```

```
[1] 3.397673e-06
```

Next, a direct Monte Carlo simulation is attempted to estimate the same probability:

```
# Direct Monte Carlo simulation for estimating P(X > 4.5) where X ~ N(0,1)
set.seed(1)
MC = 10000

s.norm = rnorm(MC) # Sample from N(0,1)

c(mean(s.norm > 4.5), var(s.norm > 4.5)/MC)
```

```
[1] 0 0
```

The result is 0 0, meaning that none of the 10000 samples exceeded 4.5. It fails because $P(X > 4.5)$ is very small, a sample size of 10000 is typically insufficient to observe even a single occurrence, leading to an estimate of zero.

Importance sampling is introduced as an alternative to direct simulation when dealing with rare events.

Instead of sampling from the original density $f(x) = \phi(x)$ (the standard normal density), you sample from an auxiliary density $g(x)$ that gives more weight to the region $x > 4.5$.

An exponential distribution with parameter $\lambda = 1$ is used, but it is shifted (truncated) so that it only takes values for $x > t$, where $t = 4.5$:

$$g(x) = \lambda \exp\left[-\lambda(x - 4.5)\right], \quad \text{for } x > 4.5.$$

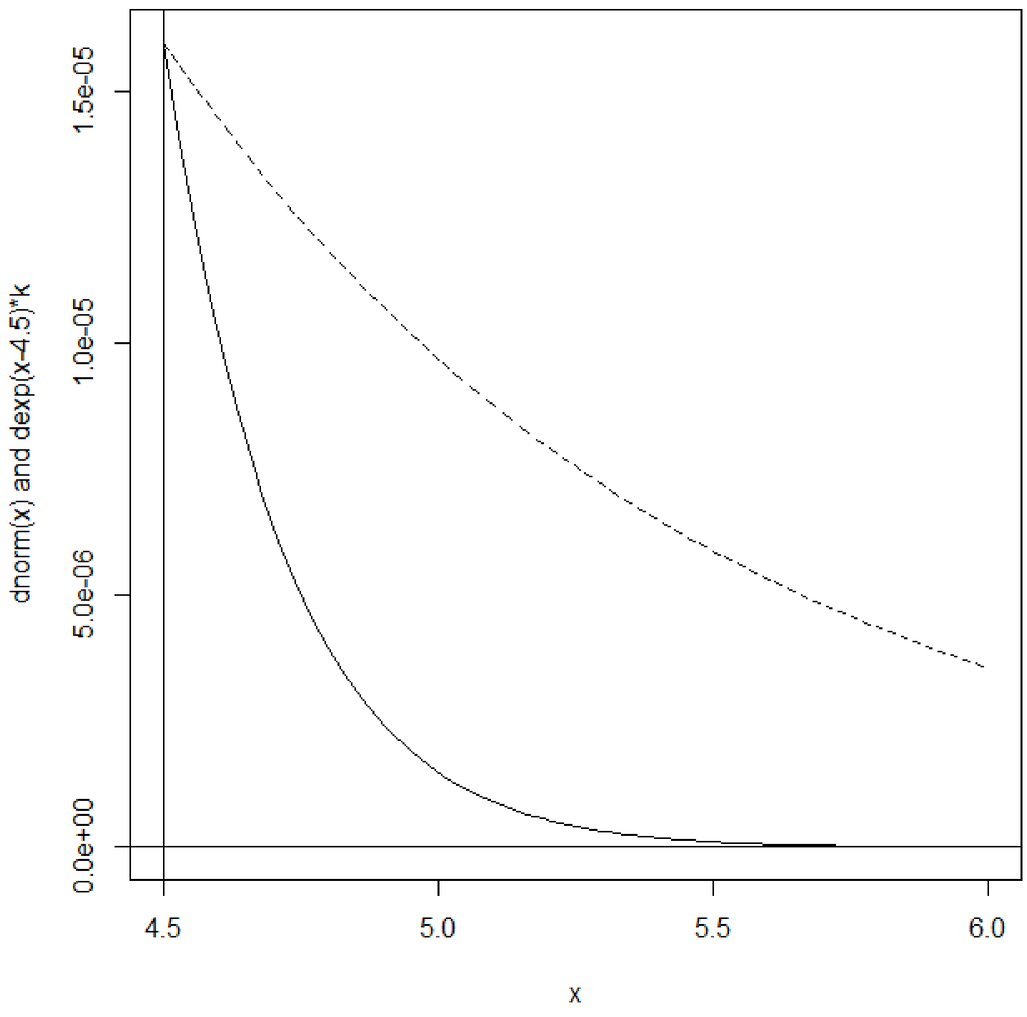
`dexp(x - 4.5)` evaluates the exponential density at x after shifting by 4.5.

`rexp(n) + 4.5` generates samples from this shifted exponential density.

We first compare $h(x)f(x)$ with the rescaled auxiliary density, $g(x)f(4.5)$, to check if it is a good choice.

A plot is made to compare the standard normal density and the rescaled auxiliary density

```r
curve(dnorm(x), 4.5, 6, ylab = "dnorm(x) and dexp(x-4.5)*k")
abline(v = 4.5)
abline(h = 0)
# The scaling factor scal is the value of the normal density at 4.5.
scal = dnorm(4.5)

# Rescale for comparison
# This second curve plots the auxiliary density multiplied by this scaling factor,
# making it easier to visually compare with the tail of the normal density
curve(dexp(x - 4.5) * scal, add = TRUE, lty = 2)
```



The comparison confirms that the shifted exponential density (when appropriately scaled) is a reasonable approximation for the tail of the normal distribution in the region of interest.

We generate auxiliary density values and calculate the weights.

```r
set.seed(1)
nsim = 10000
```

```
y = rexp(nsim) + 4.5     # Y ~ g

w =  dnorm(y)/dexp(y - 4.5)

# The simulation approximation would be mean(w * h(y)):
# h(x) = function(x) x > 4.5
# (1 if x > 4.5 => h(y) = 1)

mean(w)     # mean(w*h(y))
```

```
[1] 3.457241e-06
```

The mean of the weights is approximately $3.457241e - 06$, which is very close to the exact value computed earlier.

The weight w for each sample is computed as:

$$w = \frac{f(y)}{g(y)} = \frac{\text{dnorm}(y)}{\text{dexp}(y - 4.5)}.$$

This ratio adjusts for the fact that the samples were drawn from g instead of f.

Since $h(x)$ is an indicator function that equals 1 for $x > 4.5$ (and all y generated satisfy this condition), the estimate of $P(X > 4.5)$ is simply the mean of the weights.

The standard error of the approximation is `sqrt(var(w * h(y))/nsim)`:

```
sqrt(var(w)/nsim)             # sd(w*h(y))/sqrt(nsim)
```

```
[1] 4.474429e-08
```

The SE is approximately $4.474429e - 08$, which is very low, indicating that the importance sampling estimate is quite precise.

By applying these weights, the variance of the estimate obtained through importance sampling can be significantly reduced, particularly in scenarios where the event of interest has a low probability of occurrence.

Resampling (of sampling) by importance

When the densities f (target) or g (auxiliary) are only known up to a constant (quasi-densities), explicitly normalizing them can be challenging. Instead, we approximate a parameter θ (or an expectation) using weighted samples:

$$\theta \approx \frac{\sum_{i=1}^{n} \omega(y_i) h(y_i)}{\sum_{i=1}^{n} \omega(y_i)}$$

- $y_1, y_2, \ldots, y_n$: Samples drawn from an auxiliary distribution g.
- $\omega(y_i)$: The importance weights, usually defined as $\omega(y) = \frac{f(y)}{g(y)}$.
- $h(y_i)$: A function of interest (often an indicator or another function for estimating expectations).

This estimator is often preferred because (even though it might be biased) it tends to be more efficient (i.e., lower variance) when $\omega(y)$ and $\omega(y)h(y)$ are highly correlated.

Additionally, if you weight the sample values by

$$\frac{\omega(y_i)}{\sum_{i=1}^{n} \omega(y_i)},$$

the resampled set approximates a sample from the target density f.

Example: Simulation of a standard normal from a Cauchy distribution

The example demonstrates how to simulate samples from a standard normal distribution $N(0, 1)$ (target density) using an auxiliary distribution (a Cauchy distribution).

Here, the functions are:

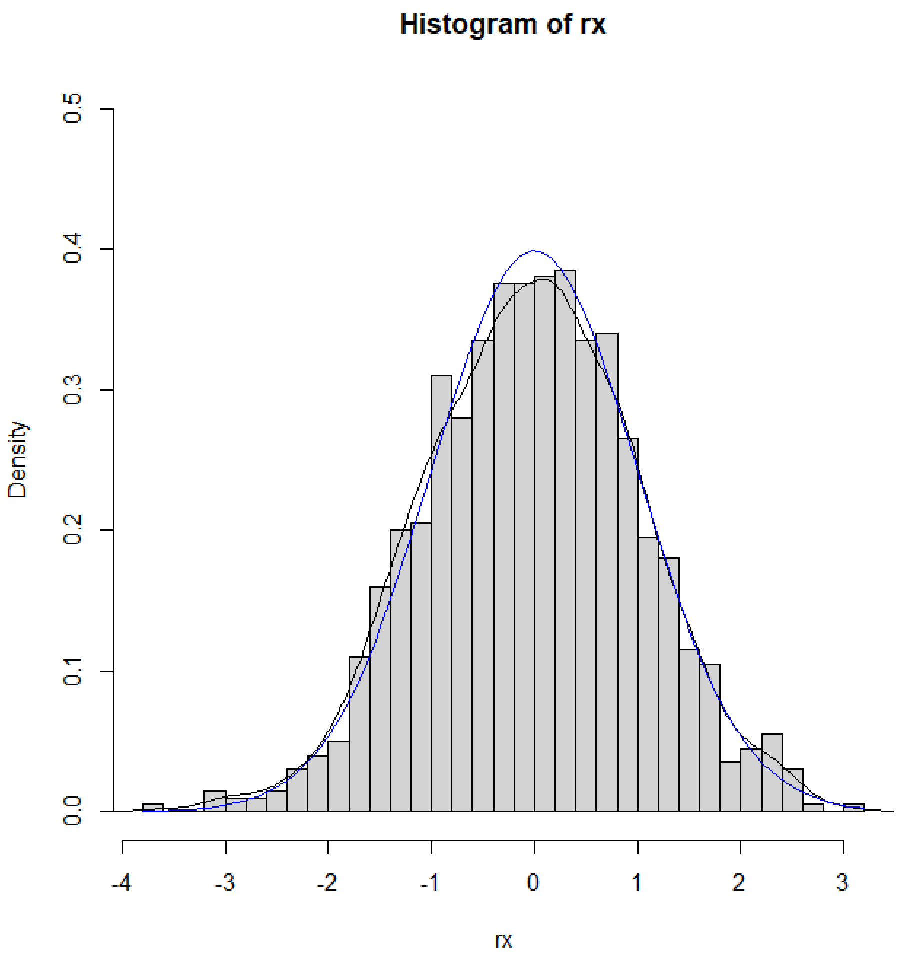
- Target density: $f(y) =$ `dnorm(y)`
- Auxiliary density: $g(y) =$ `dcauchy(y)`
- `nsim` (1000): Number of samples we want from the target density.
- `nsim2` (100,000): Number of samples drawn from the auxiliary Cauchy distribution. A larger sample from g improves the approximation.

The ratio $\omega(y) = \frac{f(y)}{g(y)}$ adjusts for the fact that the samples were drawn from g rather than directly from f.

We resample 1000 values from the auxiliary sample y, with probabilities proportional to the normalized weights $w/\sum(w)$

```
# Target density
# f = dnorm # f = function(x) ....
nsim = 10^3
# The number of simulations of the auxiliary density must be much higher:
nsim2 = 10^5
set.seed(4321)
```

```r
y = rcauchy(nsim2)
w = dnorm(y)/dcauchy(y) # w = w/sum(w)

# Sampling Importance Resampling:
rx = sample(y, nsim, replace = TRUE, prob = w/sum(w))
# This produces an approximate sample from the target density N(0, 1)

hist(rx, freq = FALSE, breaks = "FD", ylim = c(0, 0.5))
lines(density(rx))
curve(dnorm, col = "blue", add = TRUE)
```



Importance Sampling (Tilting)

Importance sampling is a statistical technique that is used to estimate the properties of a particular distribution, while only having samples from a different distribution. A common approach to importance sampling is *tilting*, which involves modifying a probability density function to emphasize certain regions of the sample space more heavily.

Given a probability density function (pdf) $f(x)$, the *tilted density* is defined as follows:

$$f_t(x) = \frac{e^{tx} f(x)}{M(t)}$$

where t is the tilting parameter, and $M(t)$ is the **moment-generating function** of $f(x)$ evaluated at t. This definition holds for any real value of t (i.e. $-\infty < t < \infty$).

NOTE:

A moment-generating function (MGF) is a valuable tool in probability and statistics, offering a compact way to analyze and understand the behavior of random variables. It essentially encodes information about a random variable's distribution through an expectation (average) under an exponential function.

- For a discrete random variable X with probability mass function $p(x)$:

$$M_X(s) = E[e^{sX}] = \sum_x p(x) e^{sx}$$

- For a continuous random variable X with probability density function $f(x)$:

$$M_X(s) = E[e^{sX}] = \int_{-\infty}^{\infty} f(x) e^{sx} dx$$

Properties:

1. $M_X(0) = 1$ always.
2. Higher derivatives of $M_X(s)$ evaluated at $s = 0$ give moments of the distribution (e.g., the first derivative gives the mean, the second gives the variance).
3. The existence of the MGF depends on the specific distribution and its moments.
4. Facilitates the study of sums of independent random variables, thanks to the property: $M_{X+Y}(s) = M_X(s) M_Y(s)$.
5. However, MGFs are not unique identifiers for distributions (different distributions can have the same MGF).

For example:

- MGF of normal distribution: $e^{\mu s + \sigma^2 s^2/2}$

- MGF of a binomial distribution: $(1+pe^s)^n$

The tilt adjusts the original density function $f(x)$ to obtain a new density function $f_t(x)$ that emphasizes different regions of the sample space. This is particularly useful in importance sampling, where the goal is to reduce variance in the estimation process by choosing an *instrumental* distribution.

A random variable with density f_t tends to be larger than a variable with density f when $t > 0$ and tends to be smaller when $t < 0$. In certain cases, the tilted densities f_t have the same parametric form as f.

Example

Consider a random variable X that follows a normal distribution $N(\mu, \sigma^2)$, where μ is the mean and σ^2 is the variance. By applying the tilting process with parameter t, the distribution of X changes to a new normal distribution:

$$X \sim N(\mu, \sigma^2) \xrightarrow{\text{tilting}} N(\mu + t\sigma^2, \sigma^2)$$

Here, tilting shifts the mean of the distribution from μ to $\mu + t\sigma^2$ while keeping the variance σ^2 unchanged.

The expression for the MGF of a normal random variable $X \sim N(\mu, \sigma^2)$ is given by:

$$M_X(t) = \exp\left(\mu t + \frac{\sigma^2 t^2}{2}\right)$$

Now, we are interested in the expression:

$$\frac{e^{tx} f(x)}{MGF(t)}$$

where $f(x)$ is the density function of X.

Substituting the expressions for $f(x)$ and $MGF(t)$, we get the following.

$$\frac{e^{tx}\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}}{\exp\left(\mu t + \frac{\sigma^2 t^2}{2}\right)} = \frac{e^{tx-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}}{\sqrt{2\pi}\sigma \exp\left(\mu t + \frac{\sigma^2 t^2}{2}\right)}$$

Now, notice that the expression inside the exponential function in the numerator and the denominator can be written as:

$$tx - \frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2 - \mu t - \frac{\sigma^2 t^2}{2} = -\frac{1}{2\sigma^2}\left(t\sigma^2 - x + \mu\right)^2$$

Therefore, the expression becomes:

$$\frac{\exp\left(-\frac{1}{2\sigma^2}\left(t\sigma^2 - x + \mu\right)^2\right)}{\sqrt{2\pi}\sigma} = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2\sigma^2}(x-(\mu+t\sigma^2))^2\right)$$

This is the density function of a normal random variable with mean $\mu + t\sigma^2$ and variance σ^2.

The effect of tilting in the context of normal distributions shows how the probability mass can be shifted towards or away from certain values. This property is crucial in importance sampling for efficiently estimating tail probabilities or expectations of rare events in a distribution.

Discrete-Time Markov Chains

A discrete-time Markov chain is a stochastic process that models a sequence of events where the probability of each event depends only on the state attained in the previous event.

Formally, a discrete-time Markov chain $\{X_k\}_{k=0}^{\infty}$ is a sequence of random variables satisfying the Markov property. This property states that the distribution of a future state X_k depends only on the current state X_{k-1}, and not on the sequence of events that preceded it. It is is expressed as:

$$P(X_k = x | X_0 = x_0, X_1 = x_1, \ldots, X_{k-1} = x_{k-1}) = P(X_k = x | X_{k-1} = x_{k-1}).$$

Homogeneity and Transition Probabilities

A Markov chain is said to be *homogeneous* if the conditional probabilities (transition probabilities) do not depend on time k. For a homogeneous Markov chain, the transition probabilities are defined as:

$$P_{i,j} = P(X_k = j | X_{k-1} = i),$$

where $P_{i,j}$ represents the probability of transitioning from state i to state j.

Stationary Distribution

A probability mass function (pmf) π is said to be a *stationary distribution* of a Markov chain if it remains unchanged over time under the dynamics of the chain. Formally, π is a stationary distribution if it satisfies the following condition for every state j:

$$\pi(j) = \sum_i P_{i,j} \pi(i).$$

This equation means that the distribution π is invariant under the transition probabilities of the Markov chain.

Ergodicity and Limiting Distribution

A Markov chain is *ergodic* if it is irreducible, aperiodic, and positive recurrent. Ergodic Markov chains have a unique stationary distribution. Moreover, for an ergodic Markov chain, the limiting distribution of the chain, as time goes to infinity, is the stationary distribution, regardless of the initial distribution of the chain.

Markov Chain Monte Carlo (MCMC) Methods

Markov Chain Monte Carlo (MCMC) methods are a class of algorithms used for sampling complex probability distributions. They do this by constructing a Markov chain that has the desired distribution as its stationary distribution. Importantly, MCMC methods generate sequences of correlated samples, not sequences of independent observations. The key idea is to use this correlation structure to efficiently explore the high-probability regions of the distribution.

Metropolis-Hastings Algorithm

The Metropolis-Hastings algorithm is a Markov chain Monte Carlo (MCMC) method used to obtain a sequence of random samples from a probability distribution for which direct sampling is difficult. This sequence can be used to approximate the distribution (denoted as f) or to compute an integral (such as an expectation).

We employ an instrumental conditional probability distribution function (pdf or pmf, depending on the context), denoted as $g(\cdot|\cdot)$, which is easier to sample from. It is important to note that the support of g, which is the set of values where g is positive, must include the support of f to ensure proper sampling.

Outline of the Algorithm The Metropolis-Hastings algorithm can be summarized in the following steps:

1. **Initialization:** Set the initial state X_0 such that $f(X_0) > 0$. This means that the initial state must be a point where the target distribution f is positive. In addition, initialize the iteration counter $k = 0$.

2. **Iteration:** Increment the counter k by 1 (that is, $k = k + 1$). Generate a candidate sample Y_k from the instrumental distribution $g(\cdot|X_{k-1})$, where X_{k-1} is the current state. Furthermore, generate a uniform random number U in the interval $(0, 1)$, independent of Y_k.

3. **Acceptance Test:** Calculate the acceptance probability $\alpha(X_{k-1}, Y_k)$, given by:

$$\alpha(x, y) = \min\left\{\frac{f(y)g(x|y)}{f(x)g(y|x)}, 1\right\}.$$

If $U \leq \alpha(X_{k-1}, Y_k)$, accept the candidate by setting $X_k = Y_k$. Otherwise, reject the candidate and maintain the current state, setting $X_k = X_{k-1}$.

4. **Termination:** If the number of iterations k is less than the desired number of samples n, repeat from Step 2. Otherwise, terminate the algorithm and return the sequence $X_0, \ldots, X_n$.

Example Metropolis-Hastings with a Poisson distribution

In this example, we consider the problem of simulating a random variable from a Poisson distribution, which is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time or space if these events occur with a known constant mean rate and independently of the time since the last event.

To implement the Metropolis-Hastings algorithm, we need to define an instrumental probability mass function (pmf). This function is used to propose moves in the state space of our Markov chain. For our Poisson rv, we define the instrumental pmf as follows:

- For $x \neq 0$: We set the probability of moving to $x-1$ as $1/2$ and the probability of moving to $x+1$ to be $1/2$.
- For $x = 0$: The probability of remaining at 0 is set to $1/2$, and the probability of moving to 1 is also set to $1/2$.

This choice of instrumental pmf ensures that the proposed moves are symmetric, which simplifies the acceptance criterion of the Metropolis-Hastings algorithm. Specifically, the acceptance probability for a proposed move from a state x to a state x' is given by the following.

$$\alpha(x, x') = \min\left(1, \frac{P(x')q(x'|x)}{P(x)q(x|x')}\right),$$

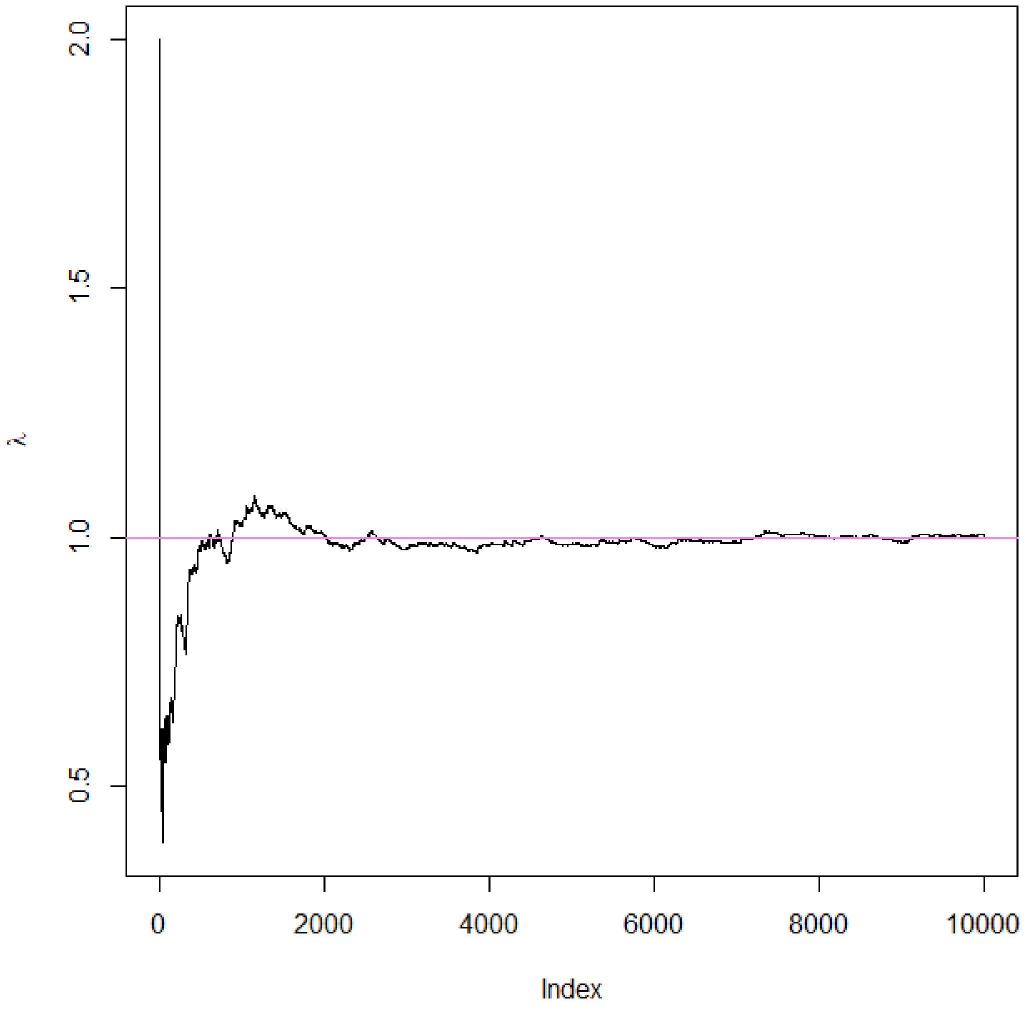
where $P(x)$ is the target Poisson distribution, and $q(x'|x)$ is the probability of proposing x' given x under the instrumental pmf. In this case, due to the symmetry of the instrumental pmf $q(x'|x) = q(x|x')$, the acceptance probability simplifies to the following:

$$\alpha(x, x') = \min\left(1, \frac{P(x')}{P(x)}\right).$$

The algorithm proceeds by iteratively proposing moves according to the instrumental pmf and accepting or rejecting these moves based on the acceptance probability. Over time, the sequence of states visited by the algorithm converges in distribution to the target Poisson distribution.

```
# Function to generate candidate values
rinst = function(x) {
  if (x == 0) {
    return(sample(c(0, 1), 1))
  } else {
    return(sample(c(x - 1, x + 1), 1))
  }
}

# Metropolis-Hastings algorithm to sample from Poisson(lambda)
poisson.metro = function(lamb, ini, n) {
  x = numeric(n)  # Preallocate memory for efficiency
  x[1] = ini
  y = rinst(ini)

  for (k in 2:n) {
    u = runif(1)

    # Compute acceptance ratio (likelihood ratio of Poisson probabilities)
    alpha = (lamb^y / factorial(y)) / (lamb^x[k-1] / factorial(x[k-1]))

    # Accept or reject proposal
    if (u <= alpha) {
      x[k] = y
    } else {
      x[k] = x[k-1]
    }

    # Generate next candidate
    y = rinst(x[k])
  }
  return(x)
}

## Example usage
# samples = poisson.metro(lamb = 1, ini = 2, n = 1000)

# Visualizing the distribution
# barplot(sample_counts / sum(sample_counts),
#         col = "lightblue",
#         main = "Poisson MCMC Samples",
#         xlab = "Value",
#         ylab = "Relative Frequency",
#         names.arg = names(sample_counts))
```

```
lmbd = 1
plot(cumsum(poisson.metro(lamb=lmbd,ini=2,n=10000))/(1:10000),
type="l", ylab = expression(lambda))
abline(h=lmbd, col="violet")
```

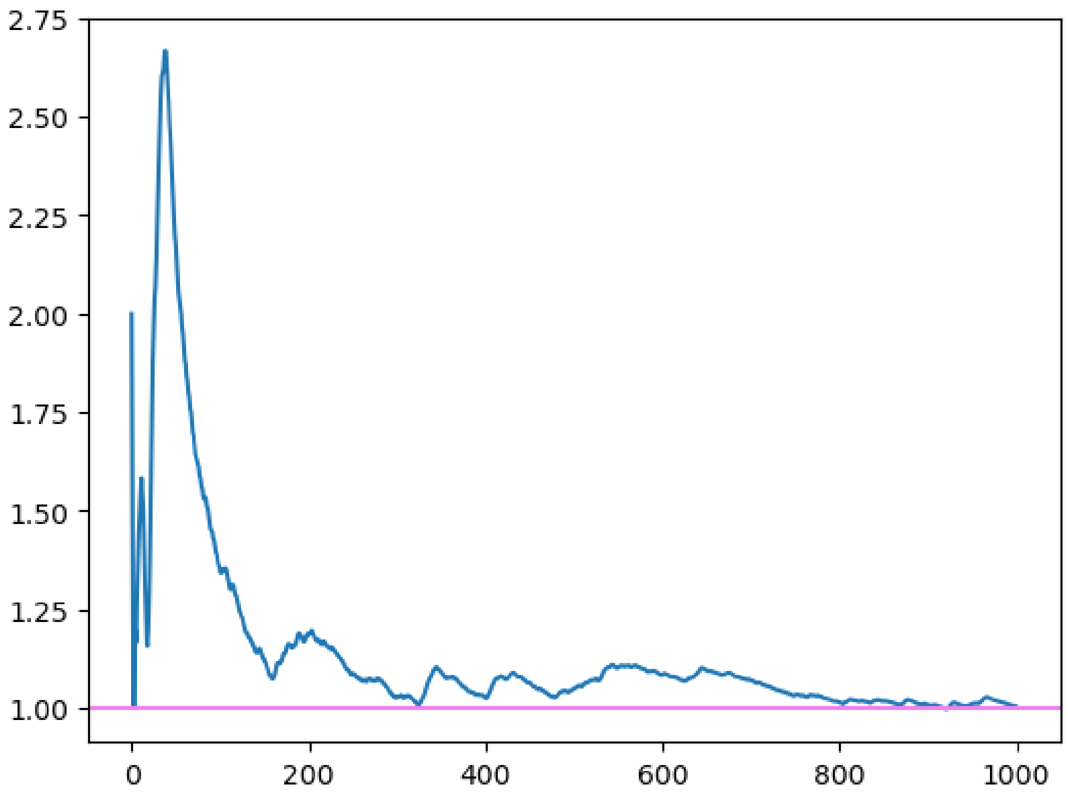


In Rcpp

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

// Helper function
int rinst(int x) {
    if (x == 0) {
        NumericVector vals = {0, 1};
        return Rcpp::sample(vals, 1)[0];
    } else {
        NumericVector vals = {x - 1, x + 1};
        return Rcpp::sample(vals, 1)[0];
    }
}

// Main function
// [[Rcpp::export]]
NumericVector poisson_metro(double lamb, int ini, int n) {
    NumericVector x(n);
    x[0] = ini;
    int y = rinst(ini);

    for (int k = 1; k < n; ++k) {
        double u = R::runif(0, 1);
        double alpha = pow(lamb, y) / R::gammafn(y + 1) /
        (pow(lamb, x[k - 1]) / R::gammafn(x[k - 1] + 1));

        if (u <= alpha) {
            x[k] = y;
        } else {
            x[k] = x[k - 1];
        }

        y = rinst(x[k]);
    }
    return x;
}
'
)

lmbd = 1
poisson_metro(lamb=lmbd ,ini=2,n=1000)
```

In Python

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

def rinst(x):
    if x == 0:
        return np.random.choice([0, 1])
    else:
        return np.random.choice([x - 1, x + 1])

def poisson_metro(lamb, ini, n):
    x = [ini]
    y = rinst(ini)

    for k in range(1, n):
        u = np.random.uniform()
        alpha = (lamb ** y / factorial(y)) / (lamb ** x[k - 1] / factorial(x[k -
                                                            1]))

        if u <= alpha:
            x.append(y)
        else:
            x.append(x[k - 1])

        y = rinst(x[k])

    return x

lmbd = 1
result = poisson_metro(lamb=lmbd, ini=2, n=1000)
cumsum_result = np.cumsum(result) / np.arange(1, 1001)

plt.plot(cumsum_result)
plt.axhline(y=lmbd, color="violet")
plt.show()
```


2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0
200
400
600
800
1000

Gibbs sampling

Consider a random vector $X = (X^{(1)}, \ldots, X^{(d)})$ with joint dmf f difficult to simulate from, but whose conditional densities of each component given the remaining components

$$f_i\left(\cdot | x_k^{(1)}, \ldots, x_k^{(i-1)}, x_k^{(i+1)}, \ldots, x_k^{(d)}\right)$$

are easy to simulate.

The **Gibbs sampler** is Metropolis-Hastings algorithm with $g(\cdot|X)$ having marginal densities as above.

In such a case, for the first component of a bivariate vector

$$f(y_1, x_2)g(x_1|x_2) = f(y_1, x_2)f_{1|2}(x_1|x_2) = f(y_1, x_2)\frac{f(x_1, x_2)}{f_2(x_2)}$$

while

$$f(x_1, x_2)g(y_1|x_2) = f(x_1, x_2)f_{1|2}(y_1|x_2) = f(x_1, x_2)\frac{f(y_1, x_2)}{f_2(x_2)}.$$

So, all candidates will be accepted because the acceptance ratio $\alpha = 1$ always.

Algorithm

1. Set $\mathbf{X}_0$ such that $f(\mathbf{X}_0) > 0$ and $k = 0$.

2. Set $k = k + 1$.

   For $i = 1, \ldots, d$, generate $X_k^{(i)}$ from

$$f_i\left(\cdot | X_k^{(1)}, \ldots, X_k^{(i-1)}, X_{k-1}^{(i+1)}, \ldots, X_{k-1}^{(d)}\right).$$

3. If $k < n$ go to Step 2, otherwise return $\mathbf{X}_0, \ldots, \mathbf{X}_n$.

Gibbs sampling (Example)

If a random vector $(X, Y)^t$ follows a bivariate normal distribution

$$(X, Y)^t \sim N\left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\right]$$

where X and Y are standard normal random variables with correlation ρ.

In general, given a bivariate normal distribution with vector mean $(\mu_X, \mu_Y)^t$ and covariance matrix

$$\Sigma = \begin{bmatrix} \sigma_X^2 & \rho\sigma_X\sigma_Y \\ \rho\sigma_X\sigma_Y & \sigma_Y^2 \end{bmatrix}$$

where σ_X and σ_Y are the standard deviations of X and Y respectively, and ρ is the correlation coefficient.

The conditional mean of Y given $X = x$ is given by

$$\mu_{Y|X} = \mu_Y + \rho\frac{\sigma_Y}{\sigma_X}(x - \mu_X)$$

And the conditional variance of Y given $X = x$ is given by

$$\sigma_{Y|X}^2 = \sigma_Y^2(1 - \rho^2)$$

In this case, $\mu_X = \mu_Y = 0$, $\sigma_X = \sigma_Y = 1$, and ρ is the correlation coefficient.

So, the conditional mean of Y given $X = x$ is

$$\mu_{Y|X} = \rho x$$

And the conditional variance of Y given $X = x$ is

$$\sigma_{Y|X}^2 = 1 - \rho^2$$

Similarly, the conditional mean of X given $Y = y$ is

$$\mu_{X|Y} = \rho y$$

And the conditional variance of X given $Y = y$ is

$$\sigma_{X|Y}^2 = 1 - \rho^2$$

So, the conditional distributions are:

$$X|Y \sim N\left(\rho Y,\ \sqrt{1-\rho^2}\right)$$

$$Y|X \sim N\left(\rho X,\ \sqrt{1-\rho^2}\right)$$

```
set.seed(1)
n = 10^4
x = c(0)
y = c(0)

rho = 0.8
sigma = sqrt(1 - rho^2)

for (i in 2:n) {
   x = c(x,rnorm(1,mean=rho*y[i-1], sd=sigma))
   y = c(y,rnorm(1,mean=rho*x[i], sd=sigma))
}

c(mean(x), mean(y))
```

```
[1] -0.01986475 -0.01208011
```

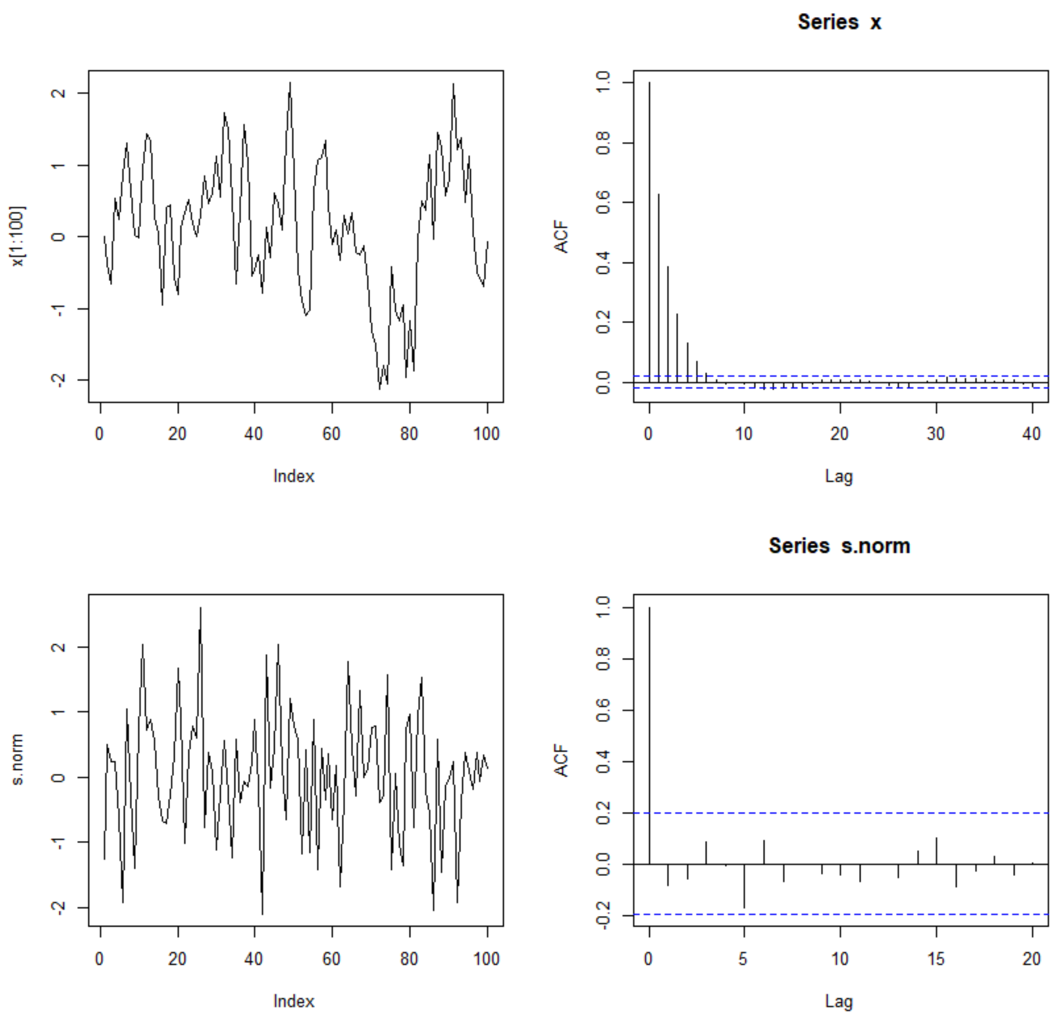
```
c(sd(x), sd(y))
```

```
[1] 0.9947670 0.9899634
```

```
cor(x,y)
```

```
[1] 0.7984191
```

```
par(mfrow=c(2,2))

plot(x[1:100], type="l")
acf(x)

s.norm = rnorm(100)
plot(s.norm,type="l")
acf(s.norm)
```



Interpretation in terms of Bayesian statistics:

Let $\theta = (\theta_1, \ldots, \theta_k)$ be a random vector with joint distribution f

Aim: simulate a sample with distribution f

1. Choose arbitrary starting values: $\theta^{(0)} = (\theta_1^{(0)}, \ldots, \theta_k^{(0)})$

2. Sample new values for each element of θ by cycling through the following steps:

3. Sample a new value for $\theta_1^{(1)}$ from the full conditional distribution of θ_1 given the most recent values of all other elements of θ:

$$\theta_1^{(1)} \sim p\left(\theta_1 | \theta_2^{(0)}, \ldots, \theta_k^{(0)}\right)$$

4. Sample a new value $\theta_2^{(1)}$ for the 2nd component of θ, from its full conditional distribution

$$\begin{aligned}\theta_2^{(1)} &\sim p\left(\theta_2|\theta_1^{(1)}, \theta_3^{(0)}, \ldots, \theta_k^{(0)}\right)\\ &\vdots\\ \theta_k^{(1)} &\sim p\left(\theta_2|\theta_1^{(1)}, \theta_2^{(1)}, \ldots, \theta_{k-1}^{(1)}\right)\end{aligned}$$

5. This completes one iteration of the Gibbs sampler and generates a new realization of the vector of unknowns $\theta^{(1)}$.

6. Repeat Step 2 many times and obtain a sample $\theta^{(1)}, \ldots \theta^{(T)}$.

Example Posterior of the Mean and Precision of a Normal Distribution

Suppose that we had observed a sample $y_1, \ldots, y_n$ generated from a random variable $Y \sim N(\text{mean} = \mu, \text{Var} = 1/\tau)$ where μ and τ are unknown.

Suppose also that we had defined the following prior distribution

$$f(\mu, \tau) \propto 1/\tau$$

Based on a sample, we can obtain the posterior distributions of μ and τ using the Gibbs sampler. It can be proved that

Conditional posterior for the mean, given the precision, is:

$$\mu|y_1, \ldots, y_n, \tau \sim N\left(\overline{y}, \frac{1}{n\tau}\right)$$

Conditional posterior for the precision, given the mean, is:

$$\tau|y_1, \ldots, y_n, \mu \sim \text{gamma}\left(\frac{n}{2}, \frac{2}{(n-1)s^2 + n(\mu - \overline{y})^2}\right)$$

where

$$\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

is the sample mean and

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \overline{y})^2$$

is the sample variance.

Then the Gibbs sampler into R can be written as:

```r
n    = 30     # Sample size
ybar = 15     # Sample mean
s2   = 3      # Sample variance

# Gibbs sampling parameters
N     = 11000 # Total iterations
T     = 1000  # Burn-in period
mu    = numeric(N) # Preallocate memory
tau   = numeric(N)

# Initialize tau
tau[1] = 1

# Gibbs sampling loop
for(i in 2:N) {
    mu[i]  = rnorm(n = 1, mean = ybar, sd = sqrt(1 / (n * tau[i - 1])))
    tau[i] = rgamma(n=1, shape=n/2, scale=2/((n-1)*s2 + n*(mu[i] - ybar)^2))
}

# Remove burn-in period
mu  = mu[-(1:T)]
tau = tau[-(1:T)]

# Density plot for mu
plot(density(mu),
     col = "red",
     lwd = 2,
     main = "Posterior Distribution of   ",
     xlab = expression(mu))

# Density plot for tau
plot(density(tau),
     col = "blue",
     lwd = 2,
     main = "Posterior Distribution of t",
     xlab = expression(tau))
```

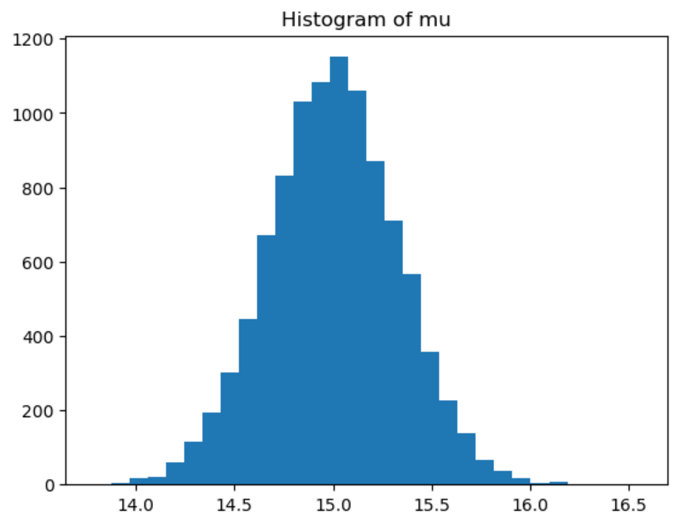
In Rcpp

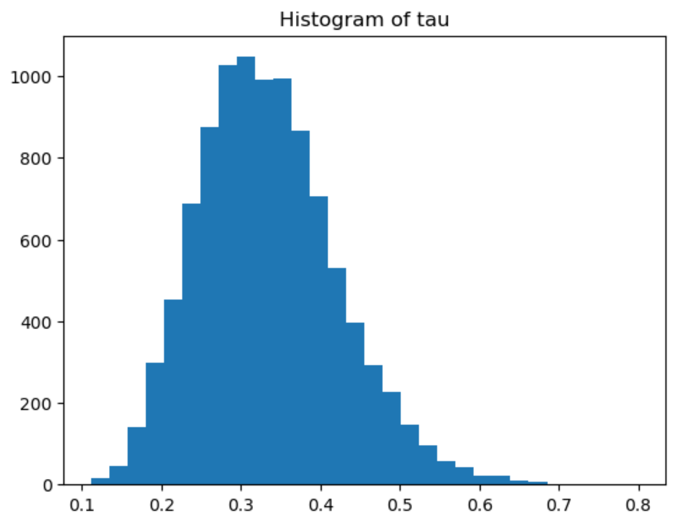
```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
List sample_posterior(int n, double ybar, double s2,
     int total_iterations, int burnin) {
    NumericVector mu(total_iterations);
    NumericVector tau(total_iterations);

    // Initialisation
    tau[0] = 1;

    // Sampling loop
    for(int i = 1; i < total_iterations; i++) {
        mu[i] = R::rnorm(ybar, sqrt(1 / (n * tau[i - 1])));
        tau[i] = R::rgamma(n / 2, 2 /
        ((n - 1) * s2 + n * pow(mu[i] - ybar, 2)));
    }

    // Remove burn-in
    mu = mu[Range(burnin, total_iterations - 1)];
    tau = tau[Range(burnin, total_iterations - 1)];

    return List::create(Named("mu") = mu, Named("tau") = tau);
}
'
)
```

```
n = 30
ybar = 15
s2 = 3
total_iterations = 11000
burnin = 1000

result = sample_posterior(n, ybar, s2, total_iterations, burnin)
mu = result$mu
tau = result$tau

# hist(mu)
# hist(tau)
```

In Python

```python
import numpy as np
import matplotlib.pyplot as plt

# Summary statistics of the sample
n = 30
ybar = 15
s2 = 3

# Sample from the joint posterior (mu, tau | data)
mu = np.empty(11000)
tau = np.empty(11000)
T = 1000  # Burnin
tau[0] = 1  # Initialization

for i in range(1, 11000):
    mu[i] = np.random.normal(loc=ybar, scale=np.sqrt(1 / (n * tau[i - 1])))
    tau[i] = np.random.gamma(shape=n/2, scale=2/((n-1)*s2 + n*(mu[i] - ybar)**2))

# Remove burnin
mu = mu[T:]
tau = tau[T:]

# Plot histograms
plt.hist(mu, bins=30)
plt.title("Histogram of mu")
plt.show()

plt.hist(tau, bins=30)
plt.title("Histogram of tau")
plt.show()
```


Histogram of mu


Histogram of tau