UCLA STATS 102C Example Code

Kelsey Lin

2024-01-21

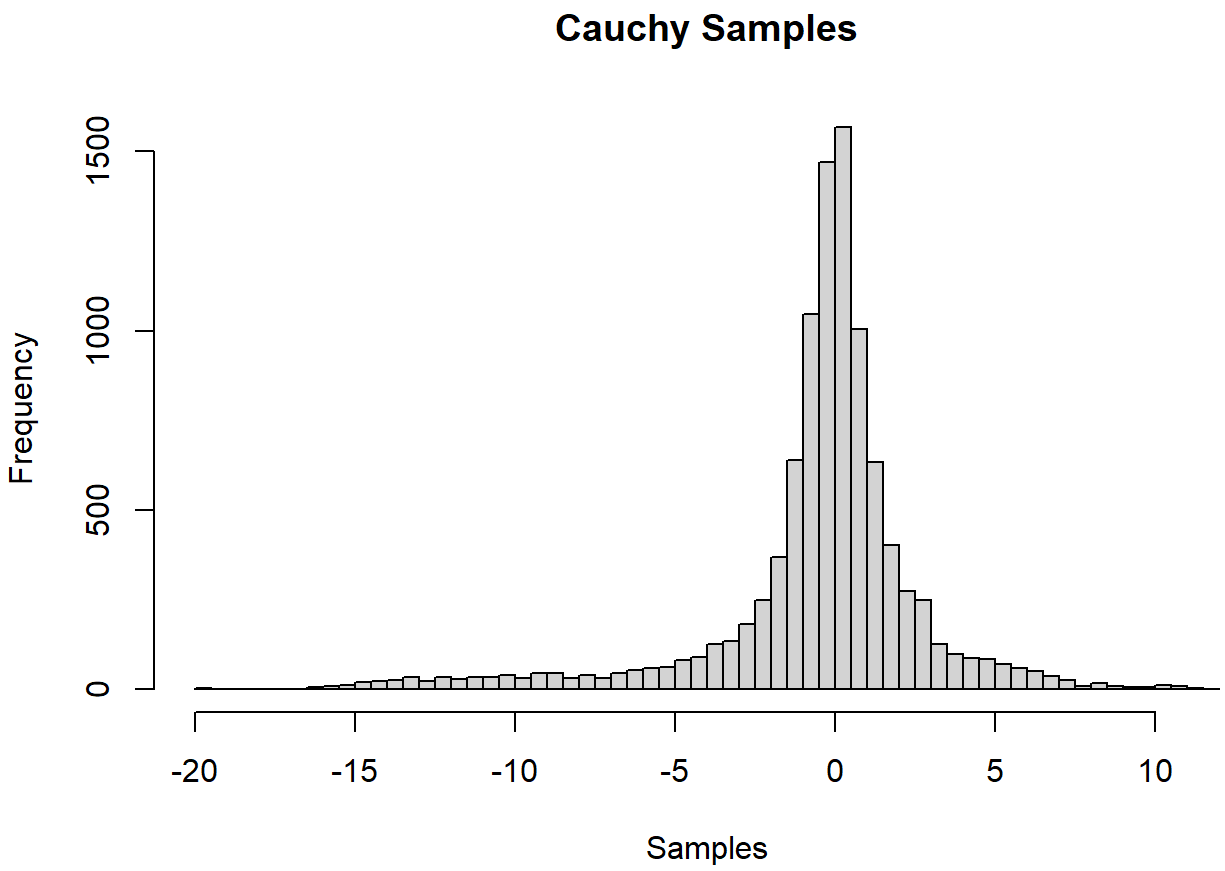
MCMC: Metropolis-Hastings Algorithm

Generating samples from the Cauchy distribution with $\gamma = 1$ and $\eta = 0$ using $N(\mu, \sigma)$ as proposal distribution:

```
#  Samples
m <- 10000

# Store Samples
x <- numeric(m)
x[1] <- rnorm(1)  # Random initial value

# Uniform Variables
u <- runif(m)

# Cauchy Function
fr <- function(x, gamma, eta) {
  1 / (pi * gamma * (1 + ((x - eta) / gamma)^2))
}
gamma <- 1
eta <- 0

# M-H Algorithm
for (i in 2:m) {
  xt <- x[i - 1]

  # Proposal Distr. (sd = 1 chosen)
  y <- rnorm(1, mean = xt, sd = 1)

  # Acceptance Ratio
  r <- fr(y, gamma, eta) / fr(xt, gamma, eta)

  # Accept/Reject
  if (u[i] <= r) {
    x[i] <- y
  } else {
    x[i] <- xt
  }
}
```

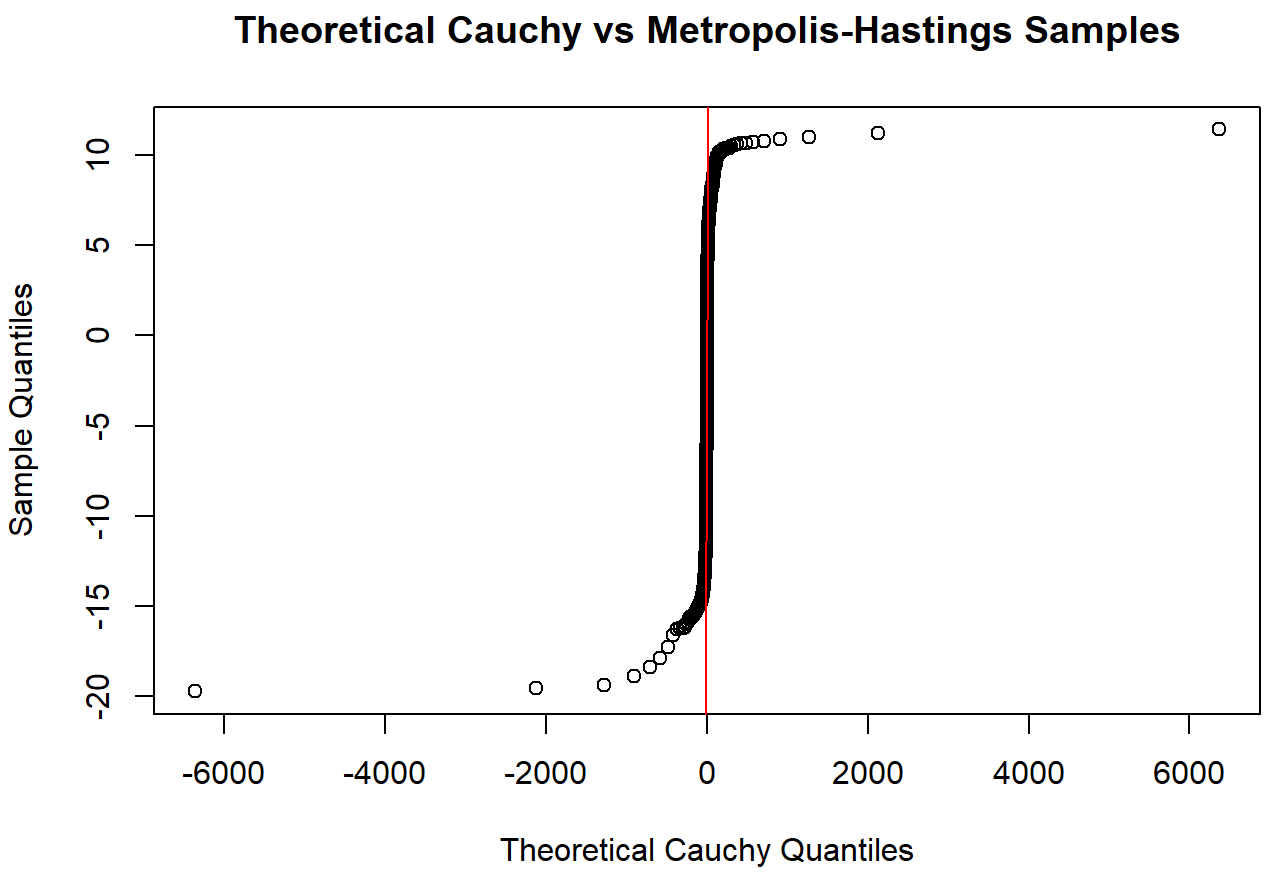
Histogram for 10,000 generated samples:

```
hist(x, breaks = 50, main = "Cauchy Samples", xlab = "Samples")
```



Compare samples with qcauchy and determine if samples agree with output of qcauchy.

```
cauchy_quantiles <- qcauchy(ppoints(length(x)))
sample_quantiles <- quantile(x, ppoints(length(x)))

# QQ plot
qqplot(cauchy_quantiles, sample_quantiles,
       main = "Theoretical Cauchy vs Metropolis-Hastings Samples",
       xlab = "Theoretical Cauchy Quantiles", ylab = "Sample Quantiles")

abline(0, 1, col = "red")
```



Overall, the exploratory analysis suggests the Metropolis-Hastings algorithm successfully captures the general behavior of the Cauchy distribution, particularly around its median, but there are more noticeable differences in the extreme values. This distribution is known for heavy tails, and the tail deviation could also be due to the sample size of 1000 vs if we used a larger sample size. All in all, the central part of the QQ plot represents the majority of the data, and it shows that the samples generated by the M-H algorithm are still in good agreement with the theoretical quantiles of the Cauchy distribution. Thus, the M-H algorithm still captures the characteristics of the Cauchy distribution.

Monte Carlo Algorithms for Parameter Estimation

Given $\theta = \int_0^2 xe^{-3x}dx$

Compute Monte Carlo estimate of theta without means of variance reduction

```
N <- 50 # Replicates
m <- 1000

# MC Estimate: theta_hat = mean(g(xi)) * (b-a), which approaches theta
mc_estimate <- function() {
    x <- runif(m, min = 0, max = 2)
    mean(x * exp(-3 * x)) * 2
}

theta_hat_mc <- replicate(N, mc_estimate())
var_mc <- var(theta_hat_mc)

# Actual
theta <- (1/9) - (7*exp(-6)/9)
theta
```

```
## [1] 0.1091832
```

Antithetic variate approach

```
R <- 10000

func <- function(x) {
  # Given Function:
  x * exp(-3 * x)
}

antithetic_estimate <- function() {
    u <- runif(R / 2)
    v <- 1 - u
    x <- u * 2
    x_prime <- v * 2
    Y <- sapply(x, func)
    Y_prime <- sapply(x_prime, func)
    mean(c(Y, Y_prime)) * 2
}

theta_hat_a <- replicate(N, antithetic_estimate())
var_antithetic <- var(theta_hat_a)
```

Control variate approach

```
# Function g(x)
g <- function(x) {
  x * exp(-3 * x)
}

# Control Variate f(x)
f <- function(x) {
  exp(-x)
}

# f(x) mu
mu <- (1 - exp(-2))

control_variate_estimate <- function() {
    u <- runif(m, min = 0, max = 2)
    g_u <- sapply(u, g)
    f_u <- sapply(u, f)
    c_star <- -cov(g_u, f_u) / var(f_u)
    mean(g_u + c_star * (f_u - mu))
}

theta_hat_c <- replicate(N, control_variate_estimate())
var_cv <- var(theta_hat_c)
```

Stratified sampling approach

```
k <- 10
r <- R/k

stratified_estimate <- function() {
    stratified_estimates <- numeric(k)
    for (j in 1:k) {
        samples <- runif(r, min = (j - 1) * 2 / k, max = j * 2 / k)
        stratified_estimates[j] <- mean(sapply(samples, g))
    }
    mean(stratified_estimates) * 2
}

theta_hat_s <- replicate(N, stratified_estimate())
var_stratified <- var(theta_hat_s)
```

Comparison of estimates

```
# Theoretical
theta
```

```
## [1] 0.1091832
```

```
# MC and Variance
mean(theta_hat_mc)
```

```
## [1] 0.1096502
```

```
var_mc
```

```
## [1] 6.557706e-06
```

```
# Antithetic Variate and Variance
mean(theta_hat_a)
```

```
## [1] 0.1092144
```

```
var_antithetic
```

```
## [1] 1.34603e-07
```

```
# Control Variate and Variance
mean(theta_hat_c)
```

```
## [1] 0.1104126
```

```
var_cv
```

```
## [1] 9.75054e-06
```

```
# Stratified Sampling and Variance
mean(theta_hat_s)
```

```
## [1] 0.1092056
```

```
var_stratified
```

```
## [1] 3.721988e-08
```

Note: Based on the structure of my functions, I obtained 50 replicates per function/method to calculate the variance. Hence, I used mean() for the theta_hat estimates for a more robust estimation per method for comparisons. We can see the estimated thetas are very close to the actual. Variance was reduced for antithetic

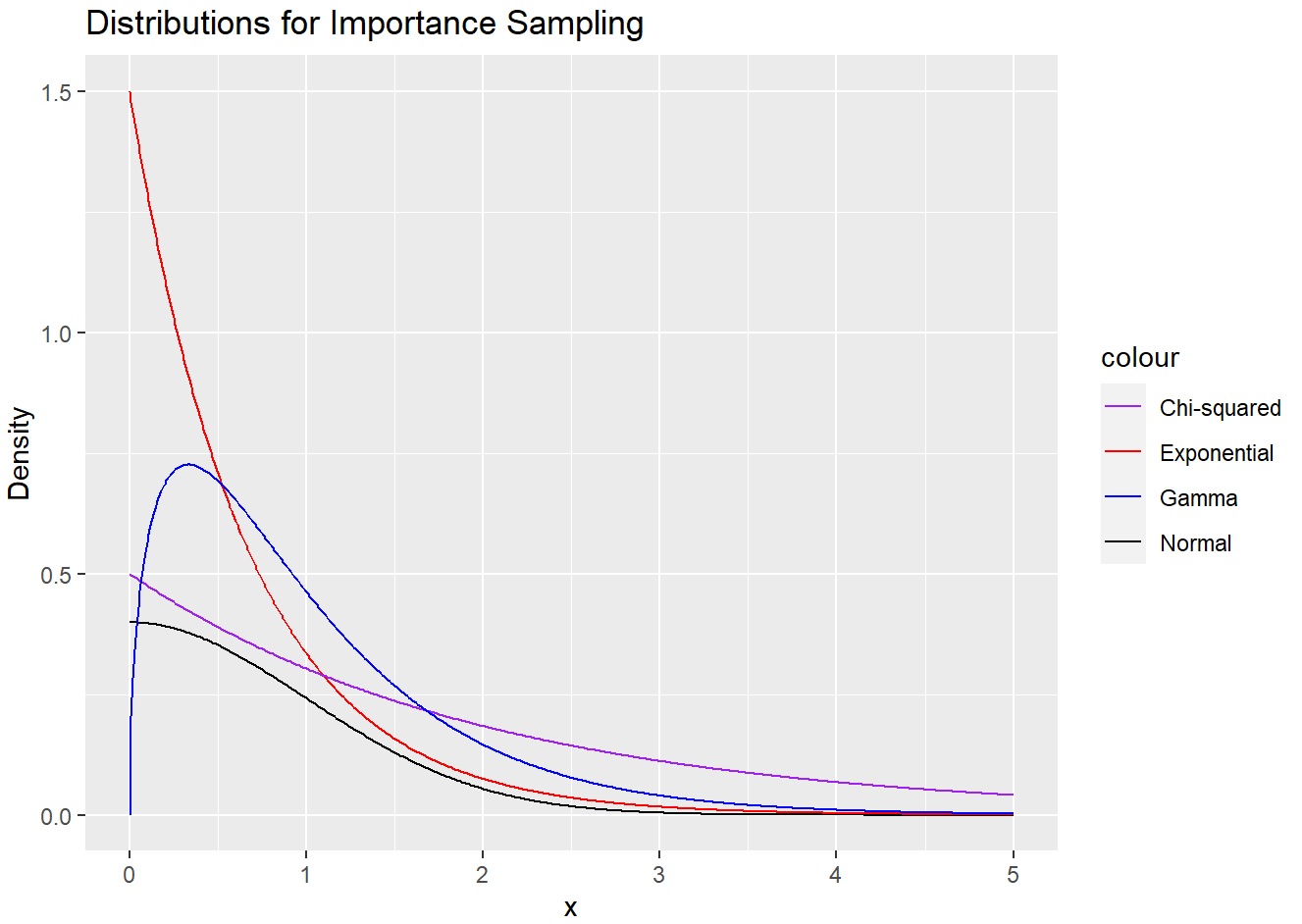
and stratified. The variance for control variate can be harder to control because it is so dependent on the choice of f(x). The control variate approach is more sensitive due to its dependence on choosing the right function that reduces variance without introducing significant bias.

Importance Sampling

Given $X \sim N(0, 1)$, we want to compute $\theta = P(X > C)$ where C is a positive constant.

Find 3 importance functions supported on (0, infinity), and explain which of your importance functions should produce smaller Var[thetahat].

```
library(ggplot2)

# Sequence of x for Distribution Functions
x <- seq(0, 5, by = 0.01)

# Given
normal <- dnorm(x)

# Distributions Chosen:
exp <- dexp(x, rate = 1.5)
chi_sq <- dchisq(x, df = 2)
gamma <- dgamma(x, shape = 1.5, rate = 1.5)

# Visualizing Distributions
df <- data.frame(x, normal, exp, chi_sq, gamma)
ggplot(df, aes(x)) +
  geom_line(aes(y = normal, color = "Normal")) +
  geom_line(aes(y = exp, color = "Exponential")) +
  geom_line(aes(y = chi_sq, color = "Chi-squared")) +
  geom_line(aes(y = gamma, color = "Gamma")) +
  labs(title = "Distributions for Importance Sampling", x = "x", y = "Density") +
  scale_color_manual(values = c("purple", "red", "blue", "black"))
```



I chose the chi-squared, exponential, and gamma distributions as my importance functions because they are somewhat similar to the given X which follows $N(0,1)$. These distributions have similar tail behaviors especially. When it came down to parameters, I tried various rates for tuning until they seemed to be closest to X. The importance function that should produce the smallest $Var[\hat{\theta}]$ would be the one closest in shape to the given X. From the plots, we can see that the exponential distribution at rate = 1.5 is a close match, especially when it comes to tail behavior. Examining the tails is crucial because we want to choose one that also can closely follow the decay of the the standard normal right-side tail beyond just C. We can note that the gamma(1.5, 1.5) is also quite close and can quantitatively find out which is best in the next few steps beyond just a visual examination.

Function to compute Monte Carlo estimate of theta using proposed importance functions.

```r
# Define g(x)
g <- function(x, C) {
  dnorm(x) * (x > C)
}

# Function to compute Monte Carlo estimate of theta using proposed importance functions:

estimate_theta <- function(n, C, importance_func, num_replicates) {

  replicate(num_replicates, {

    if (importance_func == "exp") {
      x <- rexp(n, rate = 1.5)
      phi <- dexp(x, rate = 1.5)

    } else if (importance_func == "chi_sq") {
      x <- rchisq(n, df = 2)
      phi <- dchisq(x, df = 2)

    } else if (importance_func == "gamma") {
      x <- rgamma(n, shape = 1.5, rate = 1.5)
      phi <- dgamma(x, shape = 1.5, rate = 1.5)
    }

    # Compute the outcome for importance sampling
    outcome <- g(x, C) / phi
    sum(outcome) / n
  })
}
```

Compare estimates with theoretical values for C = 0.25, 0.5, 1, 2

```r
# Define the function and number of simulations and replicates
n <- 10000
num_replicates <- 1000
C_values <- c(0.25, 0.5, 1, 2)

for (C in C_values) {
  cat("For C =", C, ":\n")

  estimates_exp <- estimate_theta(n, C, "exp", num_replicates)
  estimates_chi_sq <- estimate_theta(n, C, "chi_sq", num_replicates)
  estimates_gamma <- estimate_theta(n, C, "gamma", num_replicates)

  # Mean and Variance
  mean_exp <- mean(estimates_exp)
  var_exp <- var(estimates_exp)

  mean_chi_sq <- mean(estimates_chi_sq)
  var_chi_sq <- var(estimates_chi_sq)

  mean_gamma <- mean(estimates_gamma)
  var_gamma <- var(estimates_gamma)

  # Theoretical
  theoretical_value <- 1 - pnorm(C)

  # Results
  cat("Exponential: Mean =", mean_exp, ", Variance =", var_exp, "\n")
  cat("Chi-squared: Mean =", mean_chi_sq, ", Variance =", var_chi_sq, "\n")
  cat("Gamma: Mean =", mean_gamma, ", Variance =", var_gamma, "\n")
  cat("Theoretical: ", theoretical_value, "\n\n")
}
```

```
## For C = 0.25 :
## Exponential: Mean = 0.4010623 , Variance = 9.459613e-06
## Chi-squared: Mean = 0.4014181 , Variance = 1.307874e-05
## Gamma: Mean = 0.4014058 , Variance = 3.845294e-06
## Theoretical:  0.4012937
##
## For C = 0.5 :
## Exponential: Mean = 0.3084551 , Variance = 1.224908e-05
## Chi-squared: Mean = 0.3086756 , Variance = 1.233796e-05
## Gamma: Mean = 0.308623 , Variance = 5.60723e-06
## Theoretical:  0.3085375
##
## For C = 1 :
## Exponential: Mean = 0.1586491 , Variance = 9.792165e-06
## Chi-squared: Mean = 0.1587252 , Variance = 6.226249e-06
## Gamma: Mean = 0.158611 , Variance = 4.635283e-06
## Theoretical:  0.1586553
##
## For C = 2 :
## Exponential: Mean = 0.02273041 , Variance = 1.240628e-06
## Chi-squared: Mean = 0.02278169 , Variance = 3.458238e-07
## Gamma: Mean = 0.02275404 , Variance = 5.248476e-07
## Theoretical:  0.02275013
```

Based on these results, it appears that the distribution choices are all very close to the theoretical X following $N(0,1)$. It also appears that whichever has smallest variance depends on the value of C because C determines regions for approximation.

Inverse CDF Method

Write a function using the inverse cdf method to generate Poisson random numbers:

Generating Poisson random numbers can be slightly more complex.

We can use a recursive algorithm as mentioned in the course textbook by Rizzo.

First, we initialize a counter variable to 0 and calculate the cumulative probability for x = 0

When generating Possion(λ) variates, we can refer to the following:

$$f(x+1) = \frac{\lambda f(x)}{x+1}$$

$$F(x+1) = F(x) + f(x+1)$$

Thus, per variate, we can use the inverse transform method by generating a random uniform u.

The CDF vector is searched for the solution to

$$F(x-1) < u \leq F(x)$$

We increment x until u is within the desired range and return x as the random number for the variate.

```
poissongenerator <- function(lambda, n) {
  # Method of P(X=i-1)

  samples <- numeric(n)

  for (i in 1:n) {
    x <- 0
    u <- runif(1)
    F_x <- exp(-lambda)
    while (u > F_x) {
      x <- x + 1
      F_x <- F_x + (lambda^x) * exp(-lambda) / factorial(x)
    }
    samples[i] <- x
  }
  # Final
  samples
}
```

Generate 10,000 random numbers with lambda = 4.2 and compare results using function rpois()

```
# My function
summary(poissongenerator(lambda = 4.2, n = 10000))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   3.000   4.000   4.163   5.000  13.000
```

```
# R's built-in
summary(rpois(n = 10000, lambda = 4.2))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   3.000   4.000   4.213   6.000  14.000
```