R Code: A Short Note on Generating a Random Sample from Finite Mixture Distributions

```
#################################################################
# Example 1: Finite Mixtures of 3 Normal Distributions      #
#################################################################
n = 10^6
x = numeric(n)
pi = c(9/20,9/20,1/10)
mean.exact = c(-6/5,6/5,0)
sd.exact = c(3/5,3/5,1/4)

for (i in 1:n){
  u=runif(1)
  if (0<=u & u<pi[1]) {
    x[i]=rnorm(1, mean=mean.exact[1], sd=sd.exact[1])
  } else  if (pi[1] <= u & u< sum(pi[1:2])) {
    x[i]=rnorm(1, mean=mean.exact[2], sd=sd.exact[2])
  } else
    x[i]=rnorm(1, mean=mean.exact[3], sd=sd.exact[3])
}
samples = x



hist(samples, breaks = "scott", prob=TRUE, xlab="X", ylab="Relative Frequency"
    , main="Mixture of Normal Distribution", col = '#cbcbcb', border = "white"
    , xlim = c(-4,4))
curve(pi[1]*dnorm(x,mean.exact[1],sd.exact[1])+ pi[2]*dnorm(x,mean.exact[2],sd
    .exact[2])+ pi[3]*dnorm(x,mean.exact[3],sd.exact[3]),add=TRUE,col="#15157e
    ",lwd=3)
legend(1.7,0.27, c("True Density"), lwd=3, col=c('#15157e')
, bty = "n", lty = c(1))
```

```
#################################################################
# Example 3: Comparing Empirical and True Mixed Distributions #
#################################################################

#Case 1: Mixture of 3 t-distributions

# Define fixed parameters
pi <- c(9/20, 9/20, 1/10)
df <- c(5, 10, 15)

# Function to generate samples for the proposed approach
generate_samples_proposed <- function(n) {
  samples <- numeric(n)
  for (i in 1:n) {
    u <- runif(1)
    if (0 <= u & u < pi[1]) {
      samples[i] <- rt(1, df = df[1])
    } else if (pi[1] <= u & u < sum(pi[1:2])) {
      samples[i] <- rt(1, df = df[2])
    } else {
      samples[i] <- rt(1, df = df[3])
    }
  }
  return(samples)
}

# Function to generate samples for the composition algorithm
generate_samples_composition <- function(n) {
  samples <- numeric(n)
  for (i in 1:n) {
    k <- sample(1:3, size = 1, replace = TRUE, prob = pi)
    samples[i] <- rt(1, df = df[k])
  }
```

```
  return(samples)
}

# Function to compute F_x
F_x <- function(x) {
  pi[1] * pt(x, df = df[1]) + pi[2] * pt(x, df = df[2]) + pi[3] * pt(x, df =
      df[3])
}

# Number of Monte Carlo simulations
n_simulations <- 10000
n <- 10

# Vector to store the results
D <- numeric(n_simulations)
D1 <- numeric(n_simulations)

# Generate samples and compute D for each simulation
for (j in 1:n_simulations) {
  # Generate samples for the proposed approach
  samples <- generate_samples_proposed(n)

  # Generate samples for the composition algorithm
  samples1 <- generate_samples_composition(n)

  # Compute the difference between empirical CDF and true CDF, square it, and
      take the mean
  D[j] <- mean((ecdf(samples)(samples) - F_x(samples))^2)
  D1[j] <- mean((ecdf(samples1)(samples1) - F_x(samples1))^2)
}

# Compute mean and standard deviation of D for the proposed approach
mean_D <- mean(D)
sd_D <- sd(D)
```

```
# Compute mean and standard deviation of D for the composition algorithm
mean_D1 <- mean(D1)
sd_D1 <- sd(D1)

# Print results
cat("Results for the proposed approach:\n")
cat("Mean of D:", mean_D, "\n")
cat("Standard deviation of D:", sd_D, "\n\n")

cat("Results for the composition algorithm:\n")
cat("Mean of D using alternative:", mean_D1, "\n")
cat("Standard deviation of D using alternative:", sd_D1, "\n")

#Case 2: Mixture of 3 beta distributions

# Define fixed parameters
pi <- c(9/20, 9/20, 1/10)
alpha <- c(2, 2, 2)  # alpha parameter for each component
beta <- c(5, 10, 15)  # beta parameter for each component

# Function to generate samples for the proposed approach
generate_samples_proposed <- function(n) {
  samples <- numeric(n)
  for (i in 1:n) {
    u <- runif(1)
    if (0 <= u & u < pi[1]) {
      samples[i] <- rbeta(1, shape1 = alpha[1], shape2 = beta[1])
    } else if (pi[1] <= u & u < sum(pi[1:2])) {
      samples[i] <- rbeta(1, shape1 = alpha[2], shape2 = beta[2])
    } else {
      samples[i] <- rbeta(1, shape1 = alpha[3], shape2 = beta[3])
    }
  }
  return(samples)
}
```

```
# Function to generate samples for the composition algorithm
generate_samples_composition <- function(n) {
  samples <- numeric(n)
  for (i in 1:n) {
    k <- sample(1:3, size = 1, replace = TRUE, prob = pi)
    samples[i] <- rbeta(1, shape1 = alpha[k], shape2 = beta[k])
  }
  return(samples)
}

# Function to compute F_x
F_x <- function(x) {
  pi[1] * pbeta(x, shape1 = alpha[1], shape2 = beta[1]) +
    pi[2] * pbeta(x, shape1 = alpha[2], shape2 = beta[2]) +
    pi[3] * pbeta(x, shape1 = alpha[3], shape2 = beta[3])
}

# Number of Monte Carlo simulations
n_simulations <- 10000
n <- 100

# Vector to store the results
D <- numeric(n_simulations)
D1 <- numeric(n_simulations)

# Generate samples and compute D for each simulation
for (j in 1:n_simulations) {
  # Generate samples for the proposed approach
  samples <- generate_samples_proposed(n)

  # Generate samples for the composition algorithm
  samples1 <- generate_samples_composition(n)

  # Compute the difference between empirical CDF and true CDF, square it, and
```

```
    take the mean
  D[j] <- mean((ecdf(samples)(samples) - F_x(samples))^2)
  D1[j] <- mean((ecdf(samples1)(samples1) - F_x(samples1))^2)
}

# Compute mean and standard deviation of D for the proposed approach
mean_D <- mean(D)
sd_D <- sd(D)

# Compute mean and standard deviation of D for the composition algorithm
mean_D1 <- mean(D1)
sd_D1 <- sd(D1)

# Print results rounded to six decimal places
cat("Results for the proposed approach:\n")
cat("Mean of D:", round(mean_D, 6), "\n")
cat("Standard deviation of D:", round(sd_D, 6), "\n\n")

cat("Results for the composition algorithm:\n")
cat("Mean of D using alternative:", round(mean_D1, 6), "\n")
cat("Standard deviation of D using alternative:", round(sd_D1, 6), "\n")

#Case 3: Mixture of 3 Pareto distributions

# Load required package
library(VGAM)

# Define fixed parameters
pi <- c(9/20, 9/20, 1/10)
xmin <- c(1, 2, 3)  # Minimum value parameter for each component
alpha <- c(1, 1.5, 2)  # Shape parameter for each component

# Function to generate samples for the proposed approach
generate_samples_proposed <- function(n) {
  samples <- numeric(n)
```

```
  for (i in 1:n) {
    u <- runif(1)
    if (0 <= u & u < pi[1]) {
      samples[i] <- rpareto(1, scale = xmin[1], shape = alpha[1])
    } else if (pi[1] <= u & u < sum(pi[1:2])) {
      samples[i] <- rpareto(1, scale = xmin[2], shape = alpha[2])
    } else {
      samples[i] <- rpareto(1, scale = xmin[3], shape = alpha[3])
    }
  }
  return(samples)
}

# Function to generate samples for the composition algorithm
generate_samples_composition <- function(n) {
  samples <- numeric(n)
  for (i in 1:n) {
    k <- sample(1:3, size = 1, replace = TRUE, prob = pi)
    samples[i] <- rpareto(1, scale = xmin[k], shape = alpha[k])
  }
  return(samples)
}

# Function to compute F_x
F_x <- function(x) {
  pi[1] * ppareto(x, scale = xmin[1], shape = alpha[1]) +
    pi[2] * ppareto(x, scale = xmin[2], shape = alpha[2]) +
    pi[3] * ppareto(x, scale = xmin[3], shape = alpha[3])
}

# Number of Monte Carlo simulations
n_simulations <- 10000
n <- 20

# Vector to store the results
```

```
D <- numeric(n_simulations)
D1 <- numeric(n_simulations)

# Generate samples and compute D for each simulation
for (j in 1:n_simulations) {
  # Generate samples for the proposed approach
  samples <- generate_samples_proposed(n)

  # Generate samples for the composition algorithm
  samples1 <- generate_samples_composition(n)

  # Compute the difference between empirical CDF and true CDF, square it, and
      take the mean
  D[j] <- mean((ecdf(samples)(samples) - F_x(samples))^2)
  D1[j] <- mean((ecdf(samples1)(samples1) - F_x(samples1))^2)
}

# Compute mean and standard deviation of D for the proposed approach
mean_D <- mean(D)
sd_D <- sd(D)

# Compute mean and standard deviation of D for the composition algorithm
mean_D1 <- mean(D1)
sd_D1 <- sd(D1)

# Print results rounded to six decimal places
cat("Results for the proposed approach:\n")
cat("Mean of D:", round(mean_D, 6), "\n")
cat("Standard deviation of D:", round(sd_D, 6), "\n\n")

cat("Results for the composition algorithm:\n")
cat("Mean of D using alternative:", round(mean_D1, 6), "\n")
cat("Standard deviation of D using alternative:", round(sd_D1, 6), "\n")
```