Math 660

Nov 2023

Name: Krisha Shah

1. Write code that would allow you to perform a simulation study to examine the CLT property.

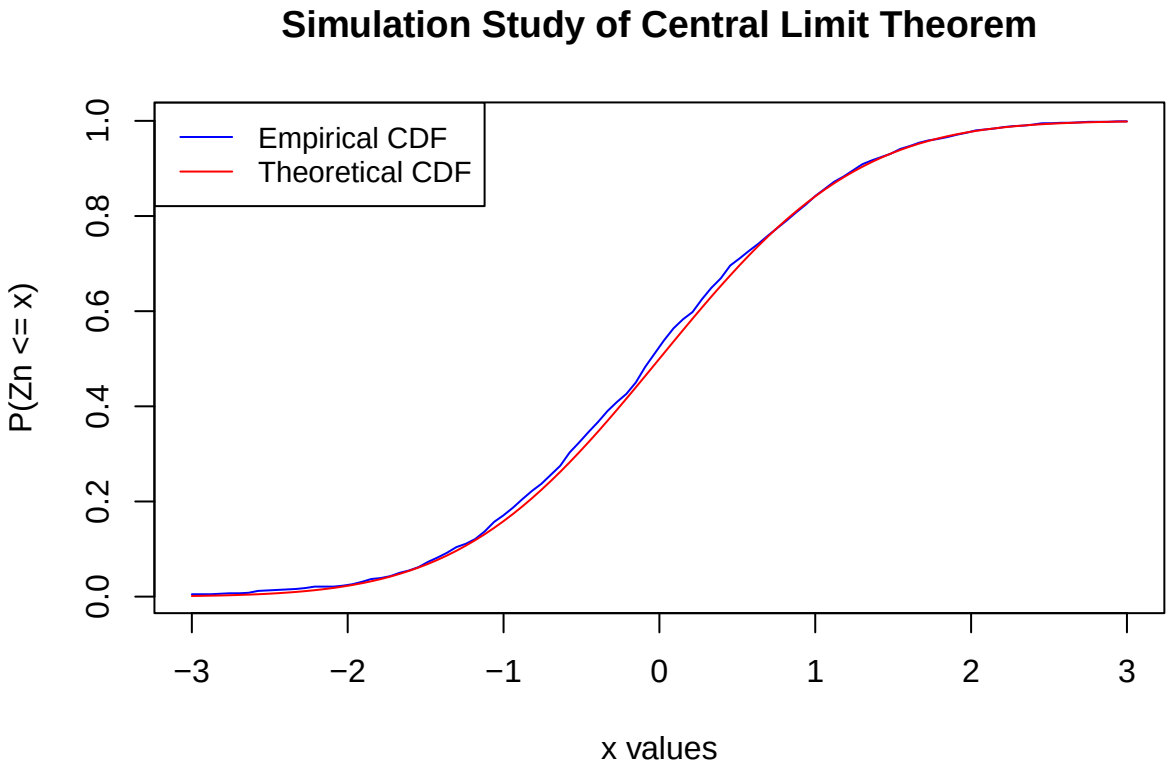
```
n <- 2000
lambda <- 10
std.X.bar <- NULL
for(r in 1:1000){
X <- rpois(n,lambda)
std.X.bar[r] <- sqrt(n)*(mean(X)-lambda)/sqrt(lambda)
}

# Compute empirical CDF
empirical_cdf <- ecdf(std.X.bar)

x<-seq(-3,3,length=100)

# Compute theoretical CDF
theoretical_cdf <- pnorm(x)


plot(x, empirical_cdf(x), type = "l", col = "blue", xlab = "x values", ylab = "P(Zn <= x)", main = "Simu
  lines(x, theoretical_cdf, type = "l", col = "red")
  legend("topleft", legend = c("Empirical CDF", "Theoretical CDF"), col = c("blue", "red"), lty = 1:1, c
```



Comment: In this code, we are trying to conduct a simulation study to illustrate the Central Limit Theorem. It generates 1000 samples, each containing 2000 observations, from a Poisson distribution with a mean of 10. The code then computes the standardized sample means and plots the empirical cumulative distribution function (ECDF) against the theoretical standard normal cumulative distribution function. Thus, the resulting visualization gives us a confirmation of the anticipated convergence to a standard normal distribution as outlined by the Central Limit Theorem.

2. In 1693 Samuel Pepys and Isaac Newton corresponded over a problem posed by Pepys in relation to a wager he planned to make. The problem was:

Which of the following three propositions has the greatest chance of success?

(a) Six fair dice are tossed independently and at least one 6 appears.

(b) Twelve fair dice are tossed independently and at least two 6's appear.

(c) Eighteen fair dice are tossed independently and at least three 6's appear.

One can solve this using binomial probabilities, but for this question, you will use simulation in R to figure out the answer. That is, write a function to simulate tossing six fair die, and use it to estimate the probability for (a). Then write functions to estimate the probabilities for (b) and (c). What does your simulation say about your answer to the problem posed by Pepys?

```r
generateSimulationData <- function(num_of_dice, num_of_tosses, target)
{
outcomes <- matrix(sample(1:6, num_of_dice * num_of_tosses, replace = TRUE), ncol = num_of_dice)
success_counts <- rowSums(outcomes == 6)
success <- success_counts >= target
prob <- mean(success)

return(prob)
}

prob_of_a <- generateSimulationData(6, 1000, 1)
prob_of_b <- generateSimulationData(12, 1000, 2)
prob_of_c <- generateSimulationData(18, 1000, 3)

cat("Probability for a:", prob_of_a, "\n")
```

```
## Probability for a: 0.683
```

```r
cat("Probability for b:", prob_of_b, "\n")
```

```
## Probability for b: 0.612
```

```r
cat("Probability for c:", prob_of_c, "\n")
```

```
## Probability for c: 0.605
```

```r
max_prob = max(prob_of_a, prob_of_b, prob_of_c)
cat("\nProposition (a) with Probability->", max_prob, "has the greatest chance of success")
```

```
##
## Proposition (a) with Probability-> 0.683 has the greatest chance of success
```

Comment: The probability for a,b and c indicate the change of getting atleast a target number of 6 when tossing a certain number of dice independently for 1000 iterations. The proposition with the greatest chance of success, based on these simulation estimates, is proposition (a): Six fair dice are tossed independently, and at least one 6 appears.

The ideal value for num_of_tosses depends on the level of precision and reliability you want in estimating the probabilities for each proposition. Generally, a larger value for num_of_tosses will result in more accurate probability estimates but will also require more computational resources. For a balance between accuracy and computational efficiency, I am using a value of 1000 tosses.

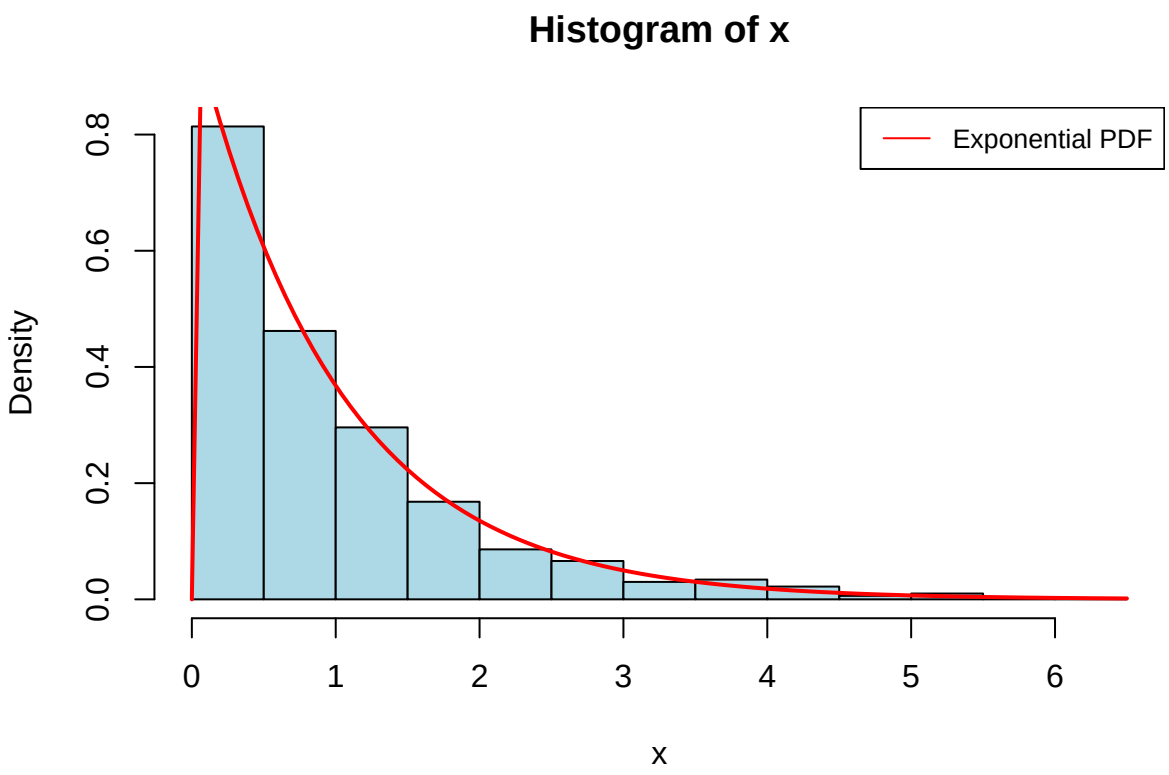
Reference: https://search.r-project.org/CRAN/refmans/dice/html/dice-package.html

3. Use your function to obtain a sample of size 1000. Make a histogram of the data set (you will need to set the option for freq to FALSE), and overlay the exponential pdf on the plot.

```r
random_draws_expo_distribution <- function(lambda, n) {
u <- runif(n)
y <- (-(1/lambda)*log(1-u))
return(y)
}

x <- random_draws_expo_distribution(1, 1000)

hist(x, freq=FALSE, col = "lightblue")
curve(dexp(x, rate = 1), add = TRUE, col = "red", lwd = 2)
legend("topright", legend = c("Exponential PDF"), col = c("red"), lty = 1:1, cex = 0.8)
```



Comment: Here, we define a function for random draws from exponential distribution that uses the inversion method to generate random draws. We take lambda and sample size 'n' as input to generate uniform random variables and later transform them using inverse of the exponential CDF. I have taken the sample size to be 1000 and lambda to be 1 and generated a histogram for showing the exponential probability density function(PDF) as a red curve plot.

4. Do a simulation study to explore how accurate these two approximation formulas are for the AR(1) model, for h = 1, 5 and 10. Remember to write modular code - write functions to do specific tasks and then put these functions together.

```r
# Function 1
AR1 <- function(num, autoregressiveCoefficient, errorStdDev) {
```

```r
  set.seed(123)
  timeSeries <- numeric(num)

  timeSeries[1] <- 0
  for (t in 2:num) {
    timeSeries[t] <- autoregressiveCoefficient * timeSeries[t - 1] + rnorm(1, mean = 0, sd = errorStdDev
  }
  return(timeSeries)
}

# Function 2
Sample_ACF <- function(data, lag) {
  num <- length(data)
  dataMean <- mean(data)

  gammaH <- sum((data[(lag + 1):num] - dataMean) * (data[1:(num - lag)] - dataMean)) / num
  gamma0 <- sum((data - dataMean)^2) / num
  rho_h <- gammaH / gamma0

  return(rho_h)
}

# Function 3
simulation_AR1 <- function(num, h, errorStdDev) {
  if (h == 0) {
    return(1 / num)
  } else {
    sum_rho_l_sq <- sum((errorStdDev^seq(1, h - 1))^2)
    var_formula1 <- 1 / num * (1 + 2 * sum_rho_l_sq)
    var_formula2 <- (1 / num) * ((num - h) / (num + 2))
    return(c(var_formula1, var_formula2))
  }
}

# Set parameters
num <- 1000
autoregressiveCoefficient <- 0.7
errorStdDev <- 1

lagValues <- c(1, 5, 10)

y <- AR1(num, autoregressiveCoefficient, errorStdDev)

# Run the simulation study
for(h in lagValues){
  rho_h <- Sample_ACF(y,h)
  var_approximations <- simulation_AR1(num, h, errorStdDev)
cat("\nResults for lag =", h, "\n")
cat("Simulated Autocorrelation Values:", errorStdDev, "\n")
cat("Variance Approximation Formula 1:", var_approximations[1], "\n")
cat("Variance Approximation Formula 2:", var_approximations[2], "\n")
}
```

```
##
## Results for lag = 1
## Simulated Autocorrelation Values: 1
## Variance Approximation Formula 1: 0.005
## Variance Approximation Formula 2: 0.000997006
##
## Results for lag = 5
## Simulated Autocorrelation Values: 1
## Variance Approximation Formula 1: 0.009
## Variance Approximation Formula 2: 0.000993014
##
## Results for lag = 10
## Simulated Autocorrelation Values: 1
## Variance Approximation Formula 1: 0.019
## Variance Approximation Formula 2: 0.000988024
```