Tutorial 11

ST2137-2320

Question 1

Estimate $\int_0^1 e^{x^2} dx$ by generating random numbers. Use 1000 uniform random variables and report a confidence interval for the estimate.

Solution

The pseudo-code for this integral would be:

1. Generate 1000 $Unif(0, 1)$ random variables.
2. Compute the sample mean. That would be our point estimate of the integral.
3. Compute the 95% CI.

R code

```
N <- 10000
system.time({
  set.seed(2134)
  U <- runif(N)
  hU <- exp(U^2)
  hUbar <- mean(hU)
  s <- sd(hU)
  lower_ci <- hUbar - qnorm(0.975)*s/sqrt(N)
  upper_ci <- hUbar + qnorm(0.975)*s/sqrt(N)
  print(c(lower_ci, upper_ci))
})
```

```
[1] 1.450262 1.468692
```

```
   user  system elapsed
  0.001   0.000   0.001
```

Python code

```
import numpy as np
import scipy.stats as stats
from scipy.stats import uniform, norm
import matplotlib.pyplot as plt

rng = np.random.default_rng(2137)
```

```
#%%timeit -n 100 -r 5

U = uniform.rvs(0, 1, size=1000, random_state=rng)
hU = np.exp(U**2)
hUbar = np.mean(hU)
s = np.std(hU)
lower_ci = hUbar - stats.norm.ppf(0.975)*s/np.sqrt(1000)
```

```
upper_ci = hUbar + stats.norm.ppf(0.975)*s/np.sqrt(1000)
print(lower_ci, upper_ci)
```

```
1.4315739546003494 1.4902879979167218
```

In R and Python, we can also use numerical integration. Since this is a course on statistical computing and programming, I think it is worthwhile to include and demonstrate

R code

```
integrate(function(x) exp(x^2), 0, 1)
```

```
1.462652 with absolute error < 1.6e-14
```

Python code

```
import scipy.integrate as integrate

integrate.quad(lambda x: np.exp(x**2), 0, 1)
```

```
(1.4626517459071815, 1.623869645314337e-14)
```

Question 2

It can be shown that if we add random numbers until their sum exceeds 1, then the expected number of random numbers added is equal to e. That is, if

$$N = \min\left\{n : \sum_{i=1}^{n} U_i > 1\right\}$$

then $E(N) = e$. (We do not need to prove the result for this question.)

(a) Use this method to estimate e, using 1000 simulation runs.
(b) Give a 95% confidence interval estimate of e.

Solution

R code

```
set.seed(77)
oneSampleN <- function() {
  N <- 0
  sum <- 0
  while(sum <= 1) {
    N <- N+1
    sum <- sum + runif(1)
  }
  N
}

system.time({
  N <- 100000
  N_sample <- sapply(1:N, function(x) oneSampleN())
  e_estimate <- mean(N_sample)
  upper_lim <- e_estimate + qnorm(.975)*sd(N_sample)/sqrt(N)
  lower_lim <- e_estimate - qnorm(.975)*sd(N_sample)/sqrt(N)
  c(lower_lim, upper_lim)
})
```

```
   user  system elapsed
  0.184   0.000   0.184
```

Python code

```python
def oneSampleN():
    N = 0
    sum = 0
    while sum <= 1:
        N += 1
        sum += uniform.rvs(0, 1, size=1, random_state=rng)

    return N

N_sample = np.array([oneSampleN() for i in range(1000)])
e_estimate = np.mean(N_sample)
s_estimate = np.std(N_sample)
lower_ci = e_estimate - stats.norm.ppf(0.975)*s_estimate/np.sqrt(1000)
upper_ci = e_estimate + stats.norm.ppf(0.975)*s_estimate/np.sqrt(1000)
print(f"Estimate: {e_estimate:.3f}, CI: ({lower_ci:.3f}, {upper_ci:.3f})")
```

```
Estimate: 2.736, CI: (2.682, 2.790)
```

Question 3

When preparing to perform the independent samples t-test, it is important to have some indication of what sample size to collect. In this question, we shall estimate the sample size required sample size for a t-test. Take note that, the test statistic is of the form

$$T_1 = \frac{(\bar{X} - \bar{Y}) - 0}{s_p\sqrt{1/n_1 + 1/n_2}}$$

The underlying assumption in this test is that

$$X_i \sim N(\mu_1, \sigma^2),\ i = 1, \ldots, n \quad (1)$$
$$Y_j \sim N(\mu_2, \sigma^2),\ j = 1, \ldots, n \quad (2)$$

Suppose that it is in our control to set $n_1 = n_2 = n$. We are going to estimate the required size n, for which we can detect a difference of $\mu_1 - \mu_2 = 1.5$ with probability 0.8, assuming that $\sigma = 1$, when testing at significance level 0.05.

Here is the pseudo-code for the simulation:

For a particular sample size n,

1. Generate n random variables from $N(0, 1)$ and $N(1.5, 1)$.
2. Compute the p-value of the t-test.
3. Set the output variable to be 1 if the p-value is less than 0.05, and 0 otherwise.
4. Repeat steps 1-3 for 1000 times. Estimate the probability of rejecting the null hypothesis for this sample size.
5. Repeat steps 1 -4 for various values of n from 5 to 10.

Create a plot (of rejection probability (y-axis) versus n (x-axis)) to aid in identifying the minimum sample size needed.

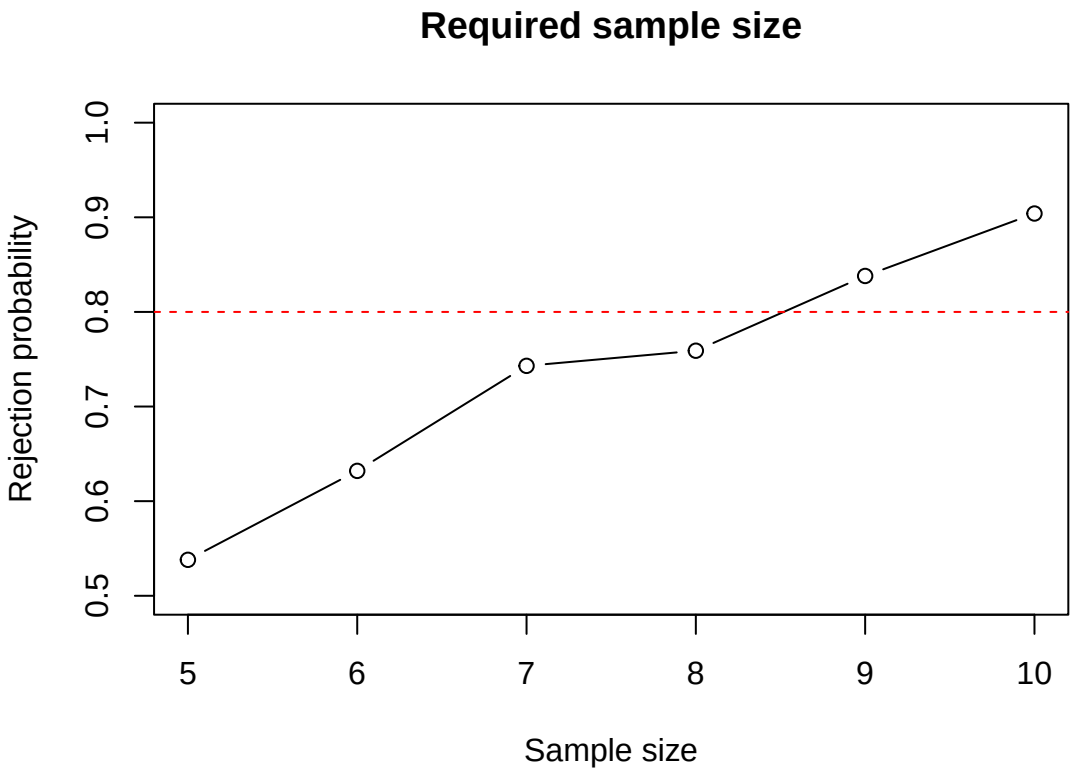
Solution

R code

```r
set.seed(7312)

onetest_outcome <- function(n=5, delta=1.5, sd= 1, alpha=0.05) {
  X <- rnorm(n, sd=sd)
  Y <- rnorm(n, delta, sd=sd)
  p_value <- t.test(X, Y)$p.value
  if(p_value < alpha) {
    return(1)
  } else {
    return(0)
  }
}
n_values <- 5:10
Nsim <- 1000
sim_output <- rep(0, length(n_values))
for (i in 1:length(n_values)){
  tmp_output <- vapply(1:Nsim,
                       function(x) onetest_outcome(n_values[i]),
                       numeric(1))
  sim_output[i] <- mean(tmp_output)
}

plot(x=n_values, y=sim_output, ylim=c(0.5, 1),
     type='b', xlab='Sample size', ylab='Rejection probability',
     main="Required sample size")
abline(h=0.8, col="red", lty=2)
```



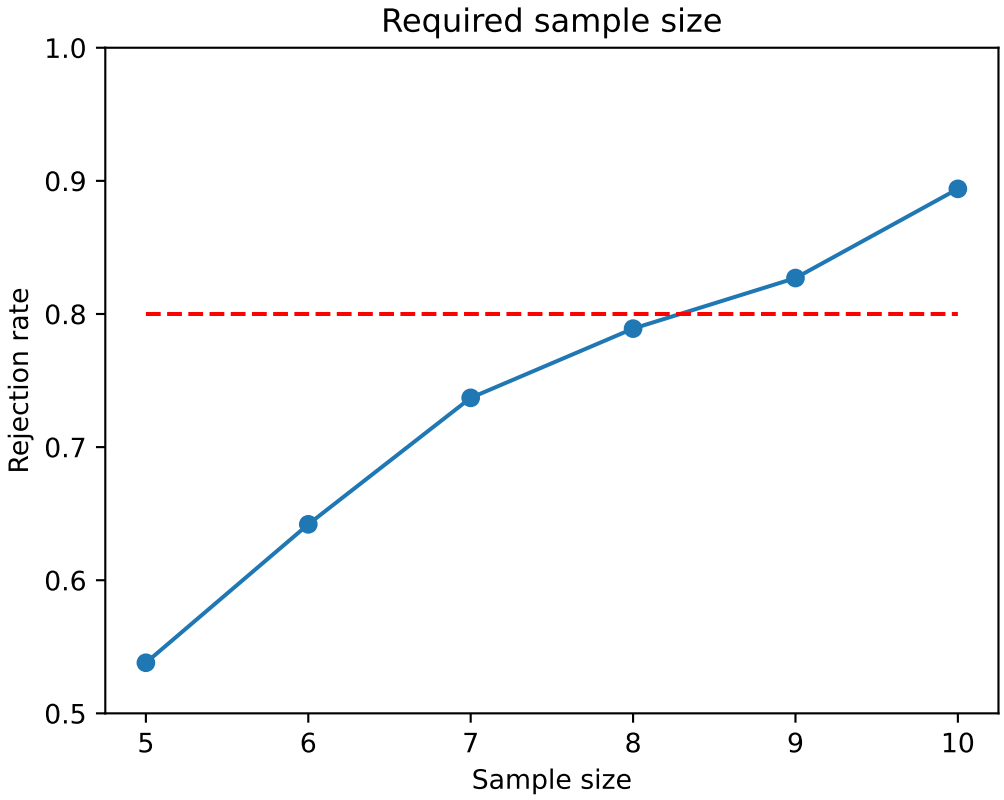
Python code

```python
def onetest_outcome(n=5, delta=1.5, sd=1, alpha=0.05):
    X = norm.rvs(0, sd, size=n, random_state=rng)
    Y = norm.rvs(delta, sd, size=n, random_state=rng)
```

```
    p_value = (stats.ttest_ind(X, Y)).pvalue
    if p_value < alpha:
        return 1
    else:
        return 0

n_values = np.arange(5,11)
N_sim = 1000
sim_output = np.zeros_like(n_values, dtype=float)
for i, n in enumerate(n_values):
    sim_output[i] = np.mean([onetest_outcome(n) for j in range(N_sim)])

plt.plot(n_values, sim_output)
plt.scatter(n_values, sim_output)
plt.ylim(0.5, 1);
plt.hlines(0.8, 5, 10, color="red", linestyles="dashed")
plt.title("Required sample size")
plt.xlabel("Sample size")
plt.ylabel("Rejection rate");
```



Question 4

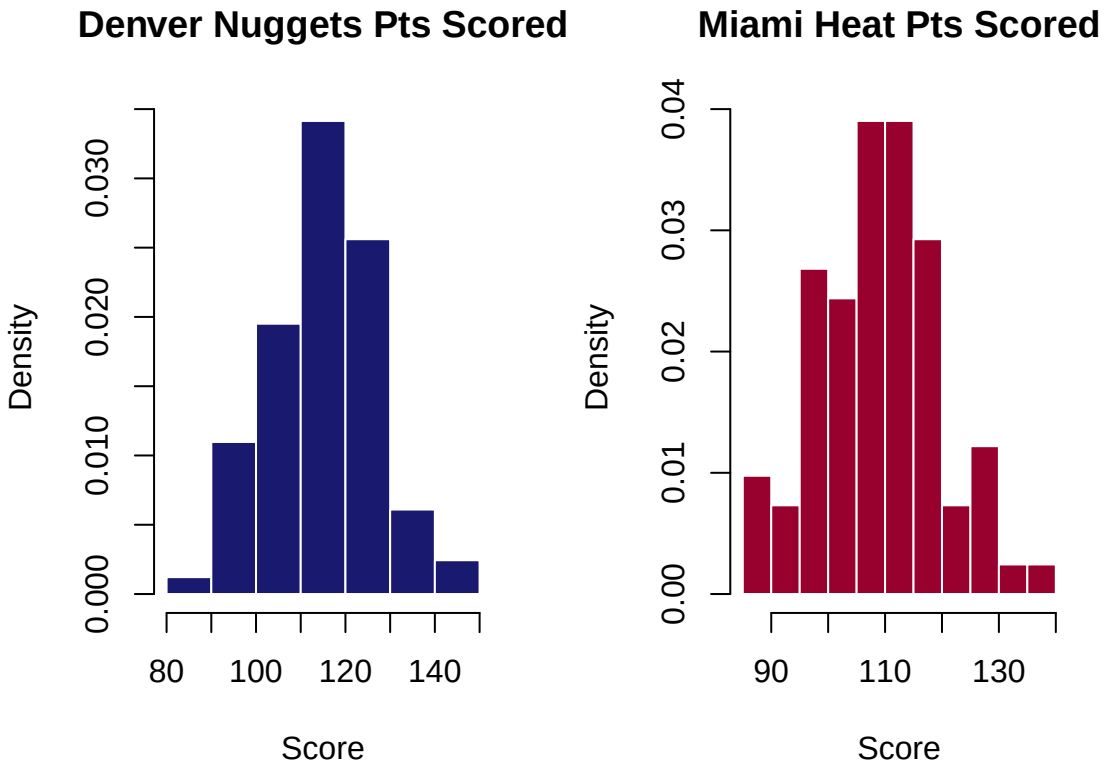
In the NBA (basketball) finals of 2023, Denver Nuggets beat Miami Heat 4-1. In this question, we shall simulate the outcome of that game in order to estimate:

1. The probability that Denver Nuggets wins the game.
2. The probability that the series would have gone to 7 games.

It is an academic exercise, since we already know the outcome of the game. However, one can certainly

do the same *before* the 2024 finals is played.

To simulate the outcome of the series, we need to model the number of points scored by each team in a game. The histograms below depict the number of points that each of those two teams scored during the regular season of 2023.

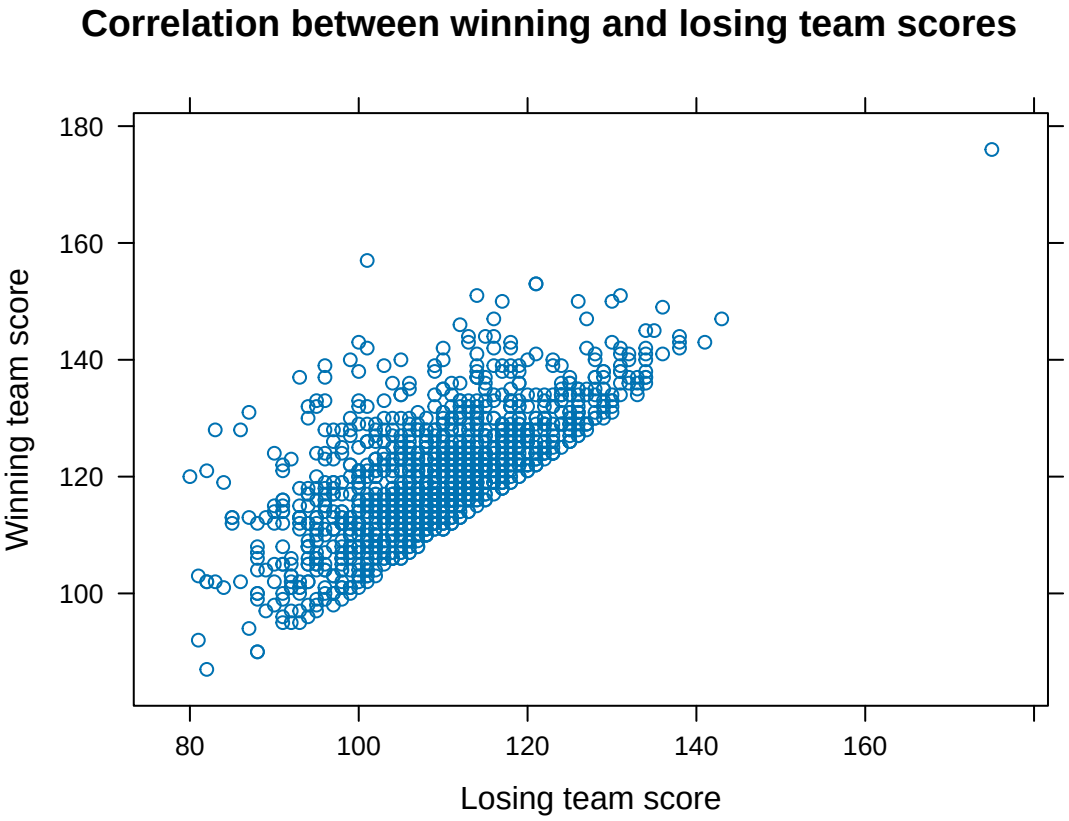


Based on the data that led to the above histograms, we shall assume that marginally,

$$\text{Denver: } X \sim N(\mu_X = 115.8,\, \sigma_X^2 = 11.2^2) \tag{3}$$

$$\text{Miami: } Y \sim N(\mu_Y = 109.5,\, \sigma_Y^2 = 10.6^2) \tag{4}$$

There is one complicating factor though - in any game, the scores of the winning and losing teams are correlated. Here is a scatterplot of the winning and losing scores for all regular season games, that shows this to be true:



The estimated correlation is $\rho = 0.7$. To take into account the correlation, we shall simulate the scores of the two teams in a **single** game as follows:

1. Simulate the score of Denver using $X \sim N(115.8, 11.2^2)$.
2. Simulate the correlated score of Miami using

$$Y|X = x \sim N\left(\mu_Y + \rho\frac{\sigma_Y}{\sigma_X}(x - \mu_X),\ \sigma_Y^2(1-\rho^2)\right)$$

Here is the pseudo-code for the simulation:

1. Generate the score of Denver using point (1) above.
2. Generate the score of Miami using the distribution in point (2) above.
3. Repeat steps 1 and 2 until one of the teams wins 4 games. (best-of-seven series)
4. Repeat steps 1 - 3 for 1000 times.

You should then be able to answer these questions:

(a) What is the probability that Denver Nuggets wins the series?
(b) What is the probability that the series would have gone to 7 games?
(c) How realistic is our simulation? What factors could we have included to make it more realistic?

Solution

R code

```
generate_one_game <- function() {
  X <- rnorm(1, 115.8, sd=11.2)
  Y <- rnorm(1, 109.5 + 0.7*(10.6/11.2)*(X - 115.8), sd=10.6*sqrt(1 - 0.7^2))
  if(X > Y) {
    return("denver")
  } else {
    return("miami")
  }
```

```r
}
generate_one_series <- function() {
  sum_d <- 0; sum_m <- 0
  while((sum_d < 4) && (sum_m < 4)){
    game_output <- generate_one_game()
    # print(game_output)
    if(game_output == "denver") {
      sum_d <- sum_d + 1
    } else {
      sum_m <- sum_m + 1
    }
  }
  series_winner <- ifelse(sum_d > sum_m, 1, 0)
  n_games <- sum_d + sum_m
  output_vec <- c(series_winner, n_games)
  names(output_vec) <- c("series_winner", "n_games")
  output_vec
}

set.seed(2317)
sim_output <- t(vapply(1:1000, function(x) generate_one_series(), c(1.0, 2.9)))
```

Python code

```python
def generate_one_game():
    X = norm.rvs(115.8, 11.2, size=1)
    Y = norm.rvs(109.5 + 0.7*(10.6/11.2)*(X - 115.8),
                 10.6*np.sqrt(1 - 0.7**2), size=1)
    if X > Y:
        return "denver"
    else:
        return "miami"
def generate_one_series():
    sum_d = 0
    sum_m = 0
    while sum_d < 4 and sum_m < 4:
        game_output = generate_one_game()
        if game_output == "denver":
            sum_d += 1
        else:
            sum_m += 1
    series_winner = 1 if sum_d > sum_m else 0
    n_games = sum_d + sum_m
    return series_winner, n_games

sim_output = np.array([generate_one_series() for j in range(1000)])
```

The estimated probability of Denver winning is 0.95.

The estimated probability of the series going to seven games is 0.111.

Knowing how to make a simulation model closer to reality requires good understanding of the real-life phenomenon. In this case, we would want to account for home-court advantage, difference between playoff and regular season, and for games that go into overtime.