Statistics 1 Unit 4 Team 8

Nikolaos Kornilakis Rodrigo Viale Jakub Trnan Aleksandra Daneva
Luis Diego Pena Monge

2024-01-05

Exercises 66-80

Exercise 66

In summary, this code is a practical demonstration of how to simulate a χ^2 distribution with a specific degree of freedom (8 in this case), and how to compare the simulated data with theoretical expectations in terms of mean and variance.

```
chi_sq <- function(n, deg){
  nrm <- matrix(rnorm(deg*n), nrow = n)

  apply(nrm, 1, function(x) sum(x^2))
}


sim <- chi_sq(10000, 8)

real <- c(8,16)
sample <- c(mean(sim), var(sim))

data.frame(Real = real,
           Sample = sample,
           Abs.Diff = abs(real - sample),
           Rel.Diff = abs(real-sample)/real,
           row.names = c("Mean", "Variance"))
```

```
##          Real    Sample   Abs.Diff    Rel.Diff
## Mean        8  8.032604 0.03260421 0.004075526
## Variance   16 16.313155 0.31315497 0.019572186
```

chi_sq <- function(n, deg): This line defines a function named chi_sq that takes two arguments: n (the number of simulations) and deg (the degrees of freedom, which is 8 in this case).
Inside the function:
nrm <- matrix(rnorm(deg*n), nrow = n): Generates a matrix nrm of random numbers drawn from a standard normal distribution (rnorm). The matrix has n rows and deg columns. The total number of elements in the matrix is deg*n.
apply(nrm, 1, function(x) sum(x^2)): Applies a function to each row of the matrix nrm. The function takes each row x, squares its elements (x^2), and then sums these squares. This operation simulates a single χ^2 variable with deg degrees of freedom.

sim <- chi_sq(10000, 8): Calls the chi_sq function with 10,000 simulations (n = 10000) and 8 degrees of freedom (deg = 8). The result is stored in sim.
real <- c(8,16): Creates a vector real containing the theoretical mean (8) and variance (16) of a χ^2 distribution with 8 degrees of freedom.
sample <- c(mean(sim), var(sim)): Calculates the sample mean and variance of thesimulated χ^2 values stored in sim.Data Frame Creation and Output:data.frame(...): Creates a data frame to neatly display the results. It includes:
Real: The theoretical values (mean and variance).
Sample: The estimated values from the simulation.
Abs.Diff: The absolute difference between the theoretical and estimated values.
Rel.Diff: The relative difference between the theoretical and estimated values, calculated as the absolute difference divided by the theoretical value.
The row names are set to "Mean" and "Variance" to indicate which row corresponds to which statistical measure.

Exercise 67

```
rannorm <- function(n, mean = 0, sd = 1){
  singlenumber <- function() {
    repeat {
      U <- runif(1)
      U2 <- sign(runif(1, min = -1)) # value is +/- 1.
      Y <- rexp(1) * U2 # Y is a double exponental r.v.
      if (U < dnorm(Y) / exp(-abs(Y))) break
    }
    Y
  }
  replicate(n, singlenumber()) * sd + mean
}
```

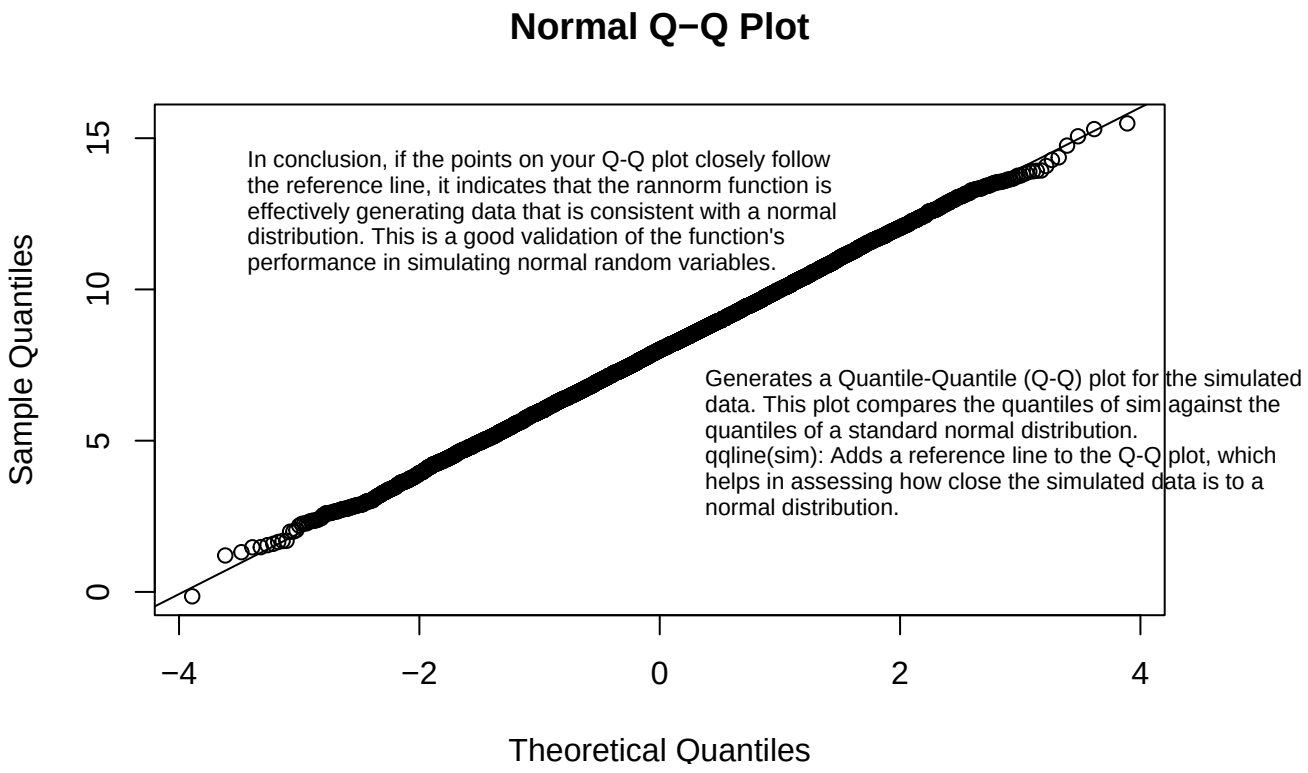
a) Generates 10,000 normal pseudorandom numbers with a mean of 8 and a standard deviation of 2. The result is stored in sim

```
sim <- rannorm(10000, mean = 8, sd = 2)
head(sim)
```

```
## [1] 10.244021  9.631736  7.908165  2.267752  9.980382  8.742088
```

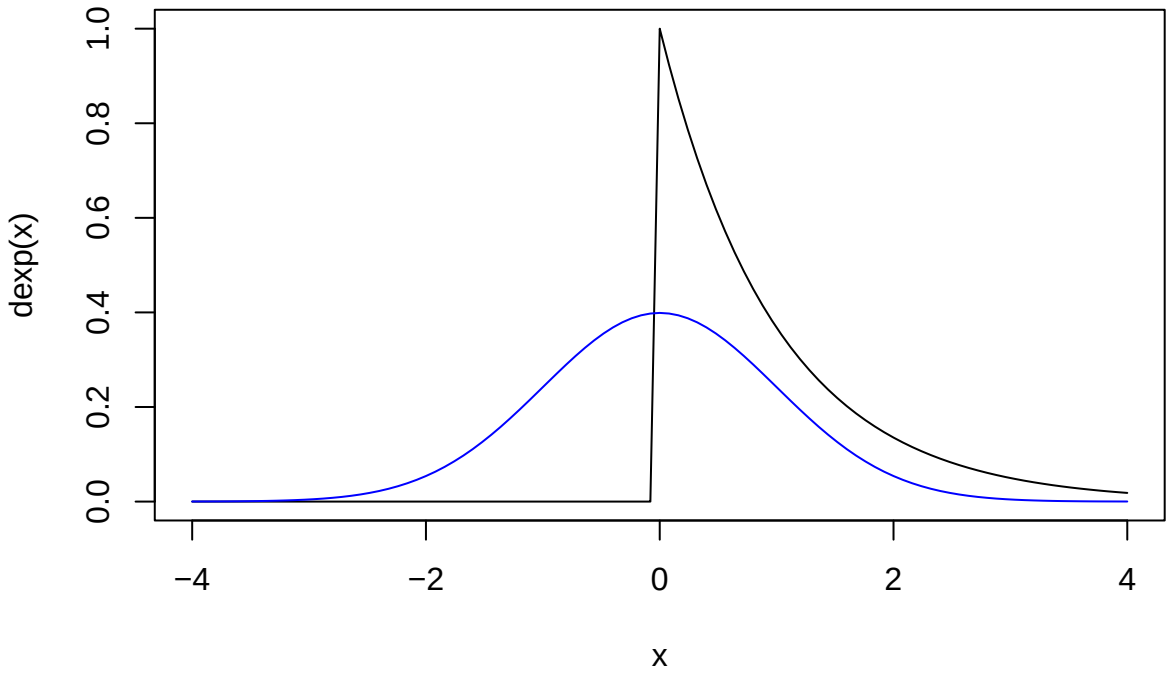
b)

```
qqnorm(sim)
qqline(sim)
```



c)

```
curve(dexp, from = -4, to = 4)
curve(dnorm, from = -4, to = 4, add = T, col = "blue")
```

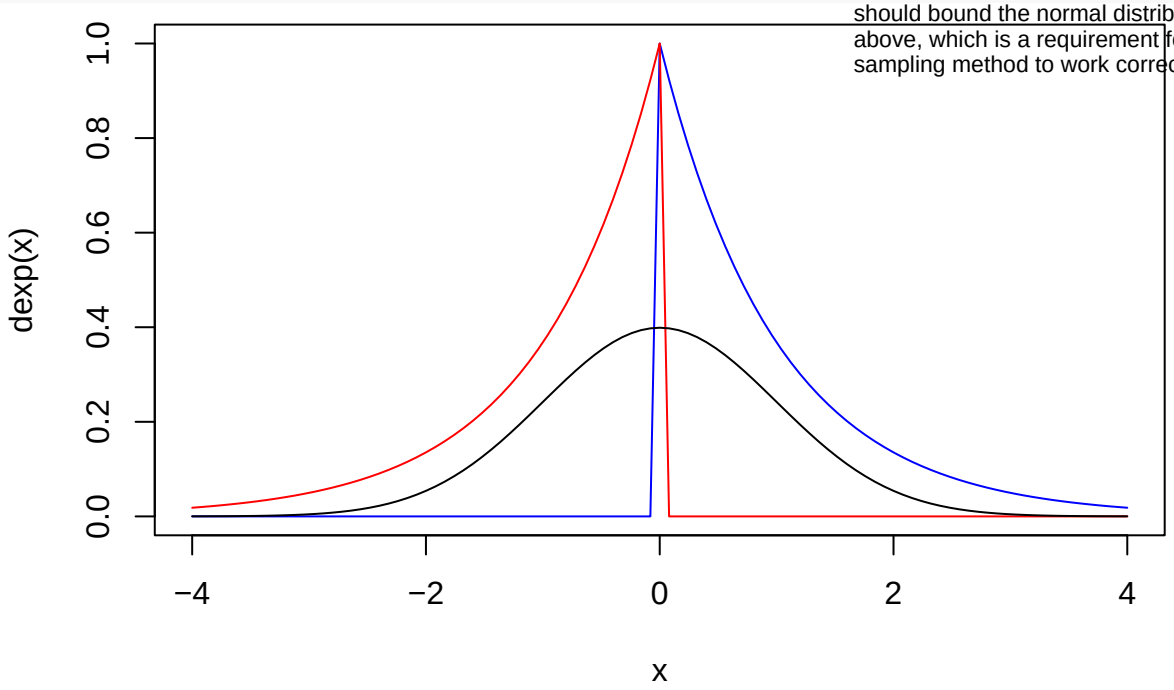


Note that the exponential distribution works adequately for the rejection method as in bounds the normal distribution above. In the previous graph, it might seem like it does not work given the negative part of the x axis, however it is possible to observe in the code that this is not in fact an issue as we take also negative values into account by multiplying the realization of the exponential by the sign of a random uniform in $[-1, 1]$. In other words, what we have is the following:

```r
neg_dexp <- function(x) dexp(-x)

curve(dexp, from = -4, to = 4, col = "blue")
curve(neg_dexp, from = -4, to = 4, col = "red", add = T)
curve(dnorm, from = -4, to = 4, add = T)
```

The purpose is to visually verify that the rejection method (used in rannorm) is appropriate. The exponential distribution should bound the normal distribution from above, which is a requirement for the rejection sampling method to work correctly.



The code is an implementation of a normal random variable generator using the rejection sampling method. The Q-Q plot and the density plots are used to validate the accuracy and correctness of the generated random variables. The exponential distribution is used as the proposal distribution in the rejection sampling, and the plots demonstrate that it bounds the normal distribution, ensuring the effectiveness of the method.

Exercise 68

```r
# Probabilities
probs <- c(0.2, 0.3, 0.1, 0.15, 0.05, 0.2)

# Inversion
randiscrete1 <- function(n, probs) {
  cumprobs <- cumsum(probs)
  singlenumber <- function() {
    x <- runif(1)
    sum(x > cumprobs)
  }
  replicate(n, singlenumber())
}

# Rejection
randiscrete2 <- function(n, probs) {
  singlenumber <- function() {
    repeat {
      U <- runif(2,
                 min = c(-0.5, 0),
                 max = c(length(probs) - 0.5, max(probs)))
      if(U[2] < probs[round(U[1]) + 1]) break
    }
    return(round(U[1]))
  }
  replicate(n, singlenumber())
}

# Timing execution
n <- 100
t1_100 <- system.time(randiscrete1(n = n, probs = probs))[3]
t2_100 <- system.time(randiscrete2(n = n, probs = probs))[3]

n <- 1000
t1_1000 <- system.time(randiscrete1(n = n, probs = probs))[3]
t2_1000 <- system.time(randiscrete2(n = n, probs = probs))[3]

n <- 10000
t1_10000 <- system.time(randiscrete1(n = n, probs = probs))[3]
t2_10000 <- system.time(randiscrete2(n = n, probs = probs))[3]

data.frame(`n=100` = c(t1_100, t2_100),
           `n=1000` = c(t1_1000, t2_1000),
           `n=10000` = c(t1_10000, t2_10000),
           row.names = c("Inversion", "Rejection"))
```

Results
The data frame shows the execution times for both methods across different sample sizes.
Observation: The inversion method (randiscrete1) is consistently faster than the rejection method (randiscrete2), especially as the number of simulations increases.
Implication: While the difference in execution time may not be significant for small sample sizes, it becomes more pronounced for larger numbers of simulations. This suggests that the inversion method is more computationally efficient and preferable for large-scale simulations.
Conclusion
The inversion method is more time-efficient for simulating values from this particular discrete distribution, making it a better choice for large-scale or computationally intensive applications. The rejection method, while conceptually straightforward, incurs more computational overhead due to the repeated sampling and checking process.

```
##           n.100 n.1000 n.10000
## Inversion 0.001  0.015   0.034
## Rejection 0.012  0.012   0.096
```

It is possible to observe that the inversion method is more time efficient than the rejection method. While for this amounts of simulations the difference might not seem that meaningful, when dealing with larger amounts of needed realizations the additional time generated from the rejection method can stack up and make more

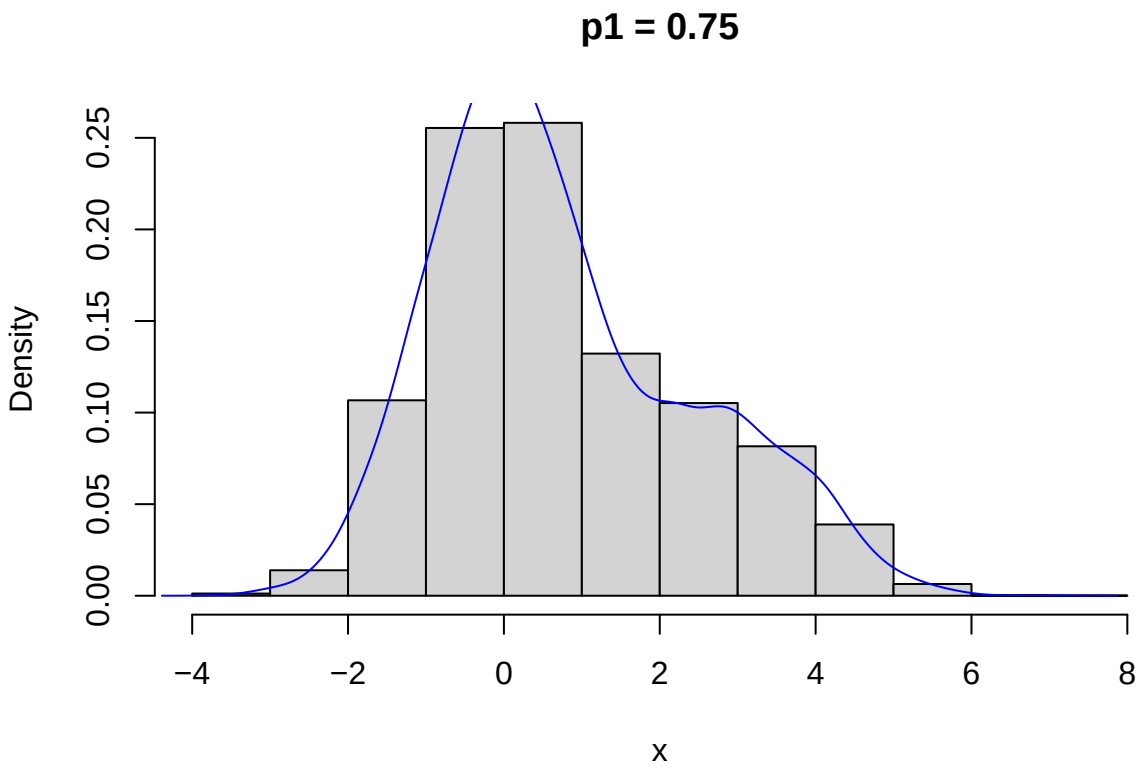
complex implementations not as computationally viable.

Exercise 69

This function generates a random sample from a mixture of two normal distributions: N(mean1, sd1) and N(mean2, sd2).
p1 <- rbinom(n, size = 1, prob = p): Generates a binary random variable for each of the n samples, where p is the probability of choosing the first distribution.
The output is a mixture of values from the two normal distributions, weighted by p1 and 1-p1.

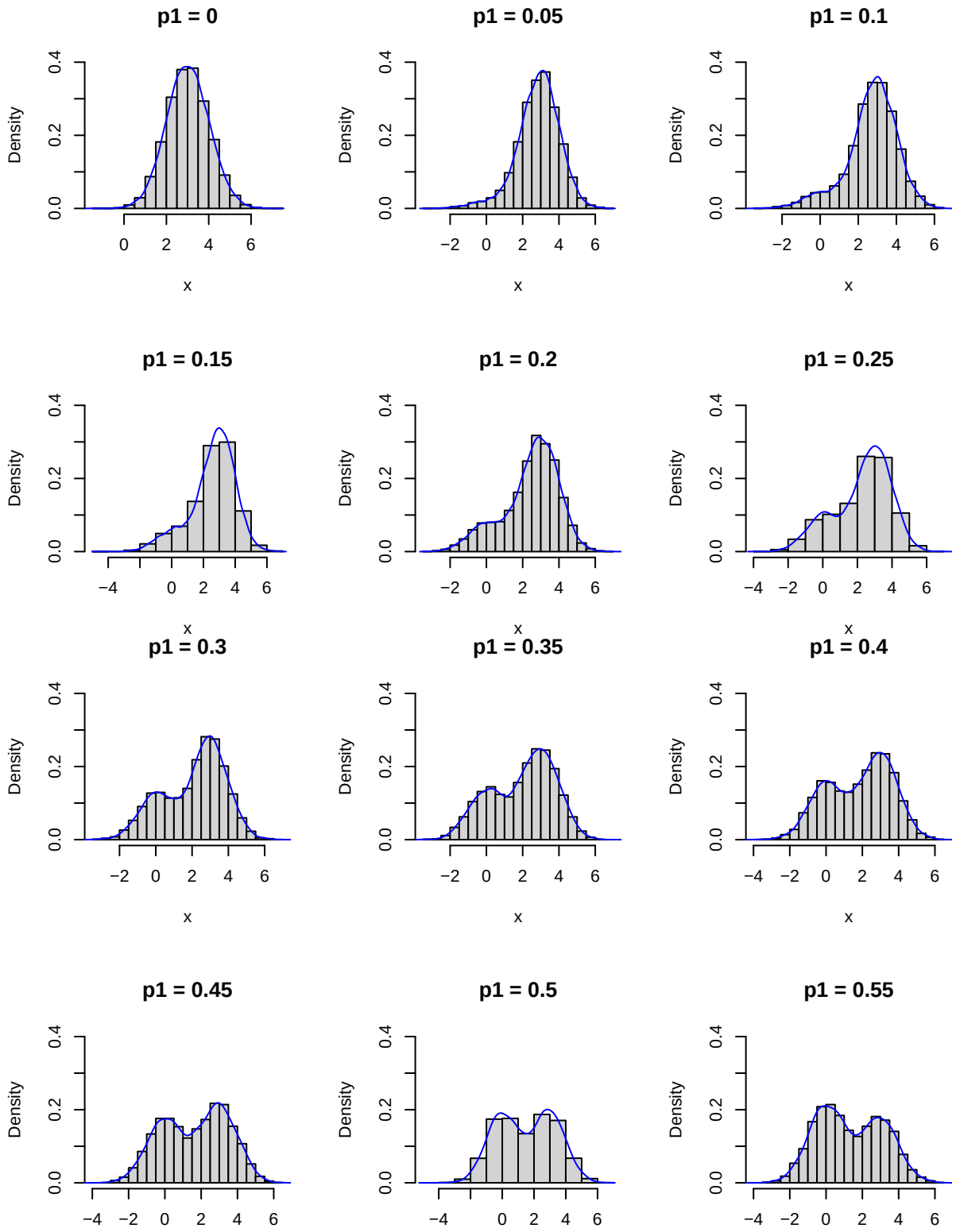
```
normal_mixture <- function(n, mean1, sd1, mean2, sd2, p){
  p1 <- rbinom(n, size = 1, prob = p)
  p1*rnorm(n, mean = mean1, sd = sd1) +
    (1-p1)*rnorm(n, mean = mean2, sd = sd2)
}

sim <- normal_mixture(10000, 0, 1, 3, 1, p = 0.75)
hist(sim, probability = T, main = "p1 = 0.75", xlab = "x")
lines(density(sim), col = "blue")
```



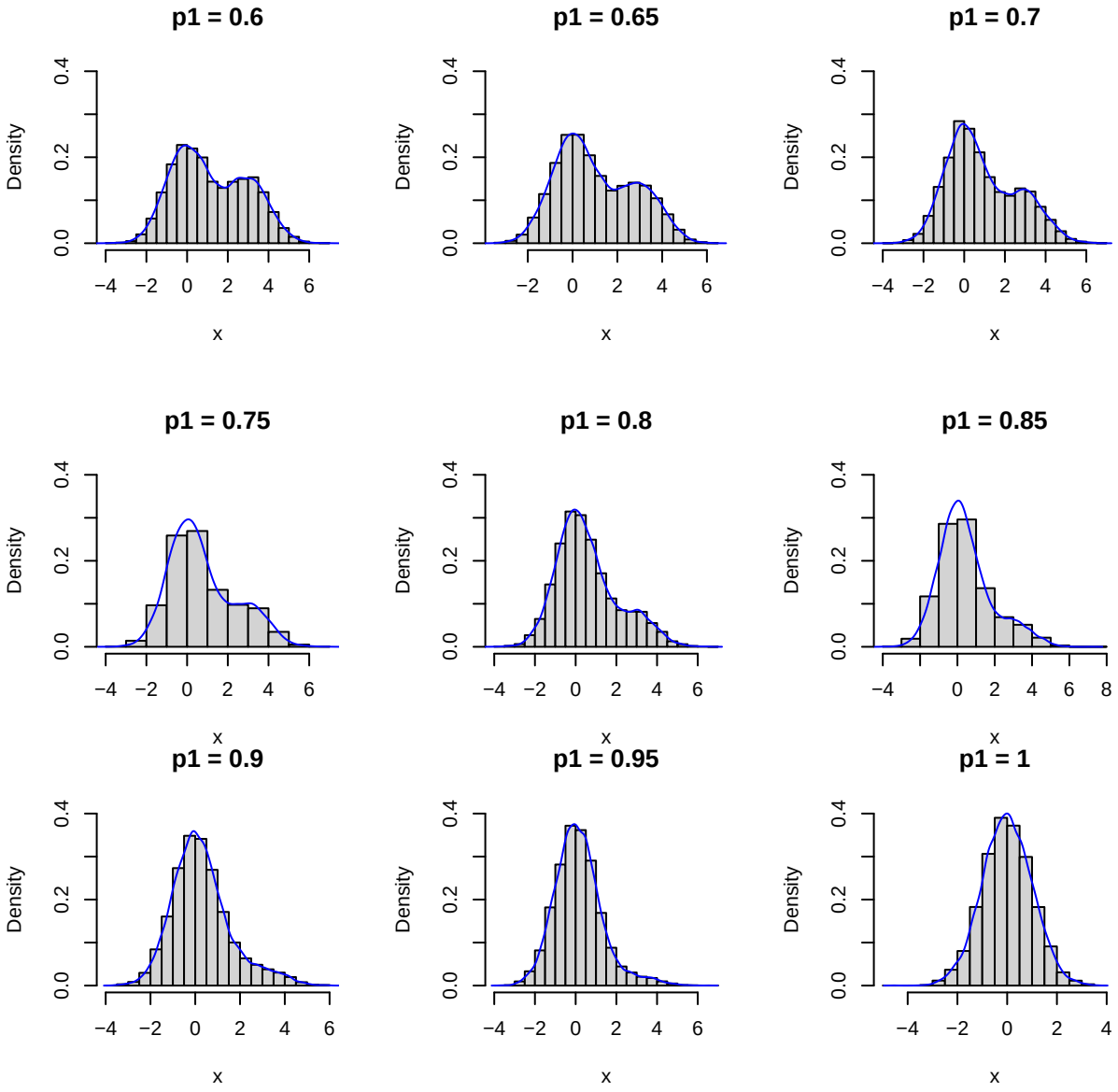

```
par(mfrow = c(2, 3))
p <- seq(from = 0, to = 1, by = 0.05)
for(p1 in p){
  sim <- normal_mixture(10000, 0, 1, 3, 1, p = p1)
  hist(sim,
       probability = T,
       main = paste0("p1 = ", p1),
       xlab = "x",
       ylim = c(0, 0.4))
  lines(density(sim), col = "blue")
}
```

The function is first used to generate 10,000 samples with p1 = 0.75.
A histogram of the sample is plotted with the density superimposed.
The process is repeated for different values of p1 (from 0 to 1 in steps of 0.05), and histograms are plotted for each.



Bimodality: The empirical distribution of the mixture appears to be bimodal in certain ranges of p1.
Pure Bimodality: This is most evident when p1 = p2 = 0.5, where both components of the mixture contribute equally, resulting in two distinct peaks (modes) in the distribution.
Emergence of Bimodality: Local maxima start to become clear when p1 is in the range of [0.2, 0.8]. In this range, both components of the mixture are sufficiently represented to create a bimodal appearance.
Dominance of One Component: When p1 is close to 0 or 1, the mixture appears less bimodal, as one of the normal distributions dominates the mixture's density.
Conjecture
The conjecture is that bimodal behavior is present when p1 is in the range [0.2, 0.8]. This is because both components of the mixture are represented enough to influence the overall shape of the distribution.
The bimodal nature is most pronounced at p1 = 0.5 due to the equal contribution of both distributions.
The bimodality would not be as clear if the standard deviations of the two normal distributions were different, as this would affect the scaling of the distributions.

Bimodality refers to a distribution that has two different modes. In the context of probability and statistics, a mode is the value that appears most frequently in a data set. Therefore, a bimodal distribution is one that has two distinct peaks or high points in its frequency plot or probability density function. These peaks represent the two most common values or ranges of values in the distribution.



While pure bimodality (i.e. both maxima share the same density) seems to be present only in the case when $p_1 = p_2 = 0.5$, it is possible to start appreciating the emergence of local maxima clearly starting from when $p_1 = 0.2$ until $p_1 = 0.8$. Therefore we conjecture the presence of bimodal behavior whenever $p_1 \in [0.2, 0.8]$. When p_1 gets closer to 0 or 1, then the mixture appears less bimodal as only one of the two normal distributions takes up practically all of the mixture's density. This same intuition justifies the first sentence, given that when $p_1 = 0.5$ then we have an equal participation of both distributions in the mixture, resulting in each one's modes to be equally important. Note that this is true since because only the location parameter changes between the individual normal distributions, what we have is a mixture between a distribution and a shift of itself along the x-axis. This pure bimodal nature would not be attained at $p_1 = 0.5$ if the standard deviations were different among both individual normals, as this would scale the distribution.

The code effectively demonstrates how the mixing probabilities in a normal location mixture distribution influence its shape, particularly in terms of bimodality. The histograms for different values of p1 visually illustrate how the balance between the two components of the mixture affects the overall distribution, providing insights into the conditions under which a bimodal distribution is observed.

Exercise 70

```
exp_gamma_mix <- function(n, r, beta){
  rexp(n, rgamma(n, r, beta))
}
```

The density function of the Lomax (Pareto) distribution is derived by differentiating its cumulative distribution function (CDF).
dpareto <- function(x, r = 4, beta = 2) { (r*beta^r)/(beta+x)^(r+1) }: This function calculates the density of the Lomax distribution for given x, r, and beta.

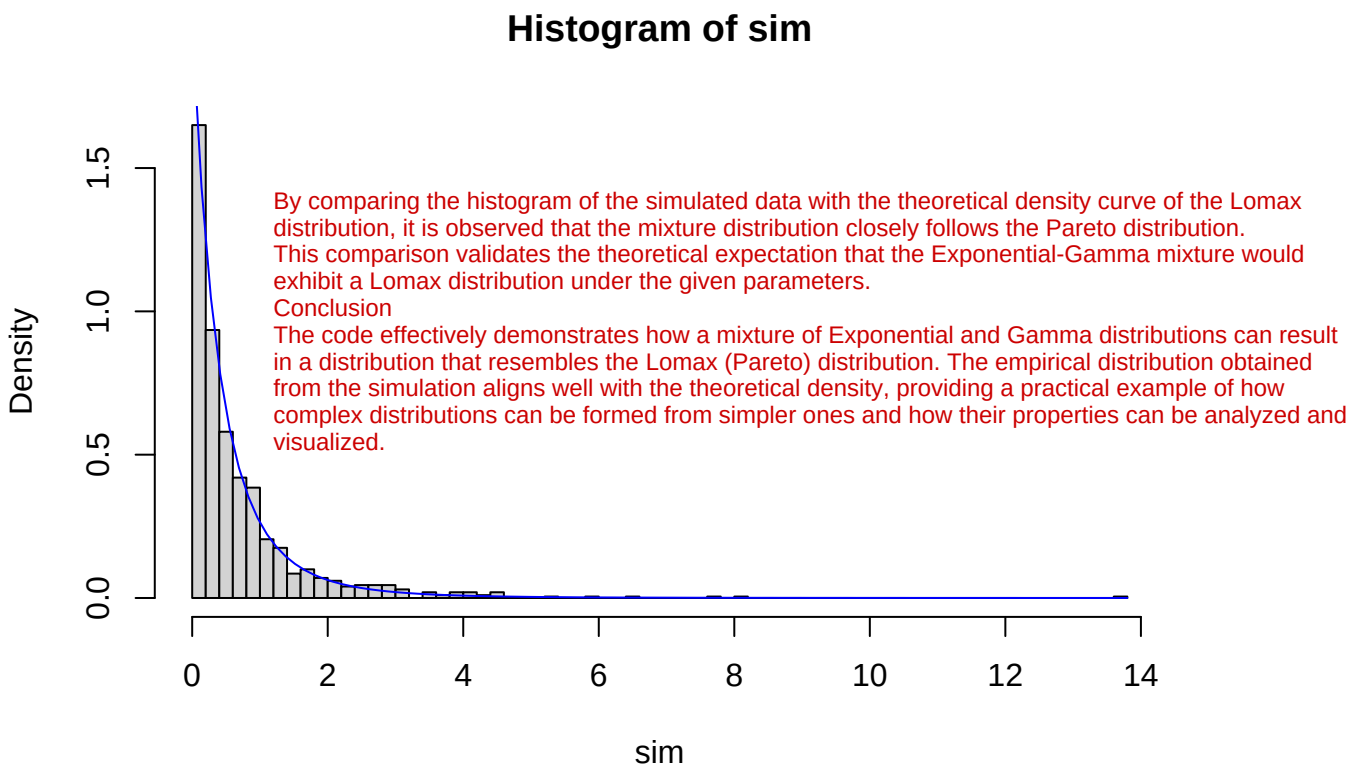
```r
sim <- exp_gamma_mix(1000, 4, 2)
```

Now, before plotting we need the Pareto density, which we can find by differentiating the distribution's cdf. For $x \geq 0$ we have:

$$\begin{aligned}
\frac{d}{dx}F(x) &= \frac{d}{dx}\left[1 - \left(\frac{\beta}{\beta + x}\right)^r\right] \\
&= -\frac{d}{dx}\left(\frac{\beta + x}{\beta}\right)^{-r} \\
&= r \cdot \left(\frac{\beta + x}{\beta}\right)^{-r-1} \cdot \frac{1}{\beta} \\
&= \frac{r \cdot \beta^r}{(\beta + x)^{r+1}}
\end{aligned}$$

```r
dpareto <- function(x, r = 4, beta = 2){
  (r*beta^r)/(beta+x)^(r+1)
}

hist(sim, probability = T, breaks = 50)
curve(dpareto, add = T, col = "blue")
```



By plotting the density histogram of the continuous mixture and the density function of the Pareto distribution with the corresponding parameters, it is possible to observe that the mixture does seem to follow the Pareto distribution.

Exercise 71

To solve this exercise, we first need to be able to generate samples from the two-dimensional t-Student distribution. We use the function from exercise 43 (previous homework) to do this:

```
rmvnorm <- function(n, m, Sigma){
  X <- matrix(rnorm(n*nrow(Sigma)), nrow = nrow(Sigma), ncol = n)

  A <- t(chol(Sigma))

  t(A %*% X + m)
}

rmvt <- function(n, nu, m, Sigma){
  AZ <- rmvnorm(n, 0, Sigma)

  W <- nu/rchisq(n, df = nu)

  sim <- t( apply(sqrt(W)*AZ, 1, function(Xi) Xi+m))

  sim
}

B <- 3000
nu <- 3
n <- 90
m <- c(0,0)
Sigma <- matrix(c(1,0.5,0.5,1), nrow = 2)

samples <- lapply(1:B, function(i) rmvt(n, nu, m, Sigma))
```

Function rmvnorm:

Generates multivariate normal random variables.
X <- matrix(rnorm(n*nrow(Sigma)), nrow = nrow(Sigma), ncol = n): Generates a matrix of standard normal random variables.
A <- t(chol(Sigma)): Computes the Cholesky decomposition of the covariance matrix Sigma.
t(A %*% X + m): Applies the linear transformation to get multivariate normal variables with mean m and covariance Sigma.

Function rmvt:

Generates random variables from a multivariate t-distribution.
AZ <- rmvnorm(n, 0, Sigma): Generates multivariate normal variables.
W <- nu/rchisq(n, df = nu): Generates weights from a chi-squared distribution.
sim <- t(apply(sqrt(W)*AZ, 1, function(Xi) Xi+m)): Applies the weights to get t-distributed variables.

Sampling:

samples <- lapply(1:B, function(i) rmvt(n, nu, m, Sigma)): Generates B = 3000 samples of size n = 90 from a bivariate t-distribution with = 3 degrees of freedom and linear correlation = 0.5

We now calculate the estimates for ρ using both methods:

```
rhos <- do.call(rbind, lapply(samples, function(X){
  c(DIRECT = cor(X[,1], X[,2]),
    KENDALL = sin(pi/2 * cor(X[,1], X[,2], method = "kendall")))
}))

head(rhos)
```

```
##         DIRECT   KENDALL
## [1,] 0.8759968 0.4993205
## [2,] 0.5543451 0.4870378
## [3,] 0.3995728 0.4235067
## [4,] 0.5580510 0.3804176
## [5,] 0.6060562 0.6526483
## [6,] 0.5146216 0.5013580
```

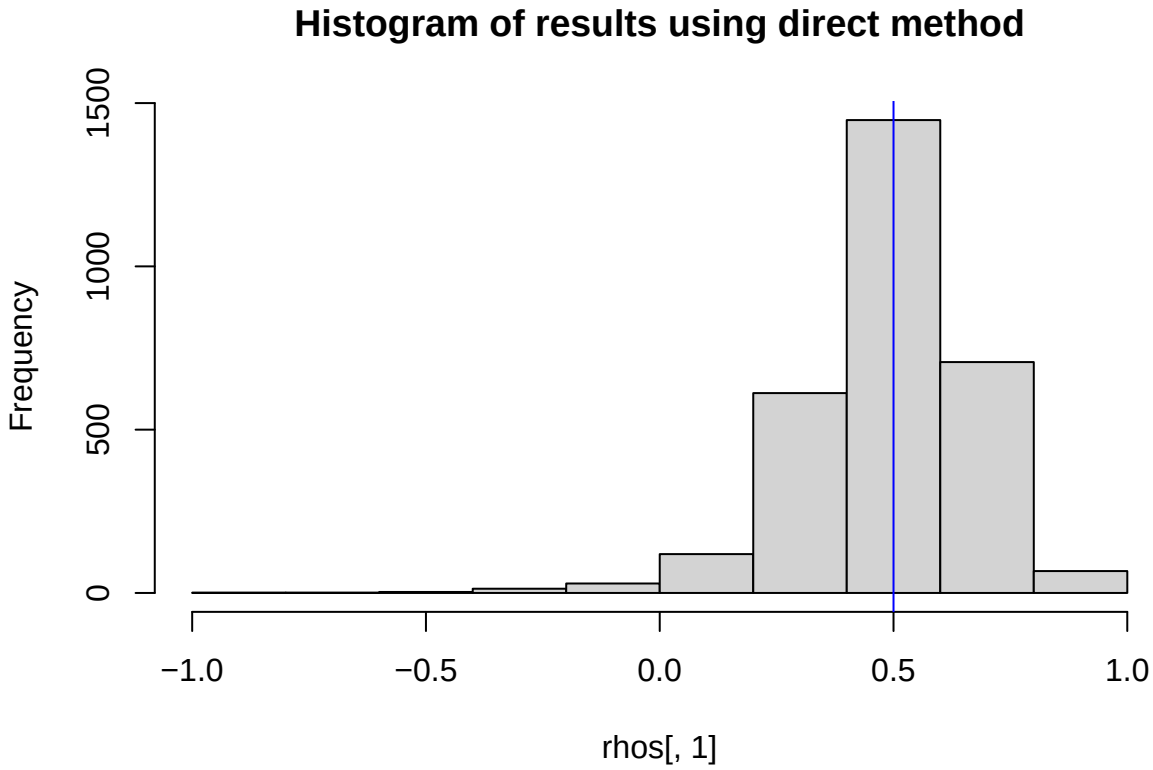
In summary, the bivariate t-distribution is a useful tool in statistics for modeling the joint behavior of two variables, especially in situations where data may not follow a normal distribution and where the sample size is small. Its ability to capture tail dependence is particularly valuable in risk assessment and financial modeling.

```
rbind(apply(rhos, 2, summary), sd = apply(rhos, 2, sd))
```

```
##             DIRECT   KENDALL
## Min.    -0.8434684 0.1154436
## 1st Qu.  0.3963591 0.4297144
## Median   0.5044006 0.5006792
```

```
## Mean     0.4877251 0.4952065
## 3rd Qu.  0.6036371 0.5650317
## Max.     0.9813804 0.7992252
## sd       0.1802387 0.1004536
```

```
hist(rhos[,1], main = "Histogram of results using direct method")
abline(v = 0.5, col = "blue")
```



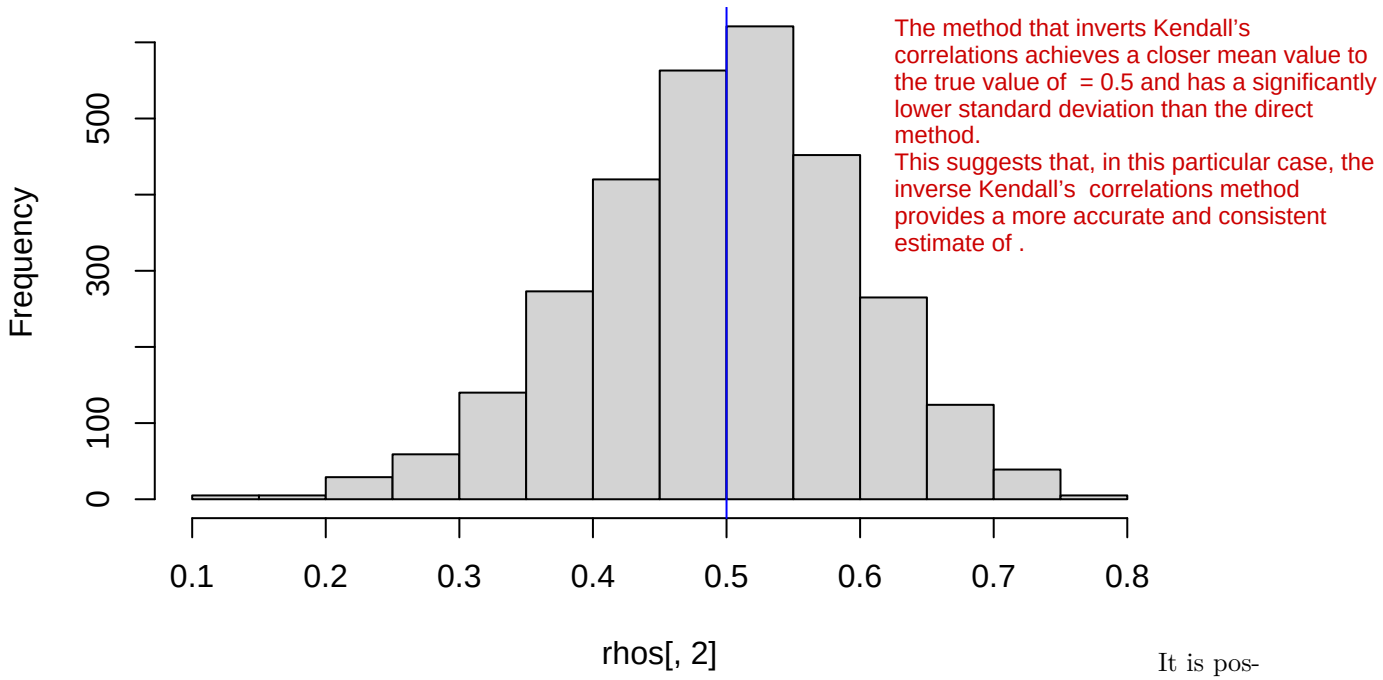

```
hist(rhos[,2], main = "Histogram of results inverting Kendall's correlations")
abline(v = 0.5, col = "blue")
```

Comparison of Methods

The exercise compares the two methods by calculating the estimates for for each sample and then analyzing the distribution of these estimates.
Histograms are plotted for the estimates from both methods, and the mean and standard deviation of the estimates are computed.

Conclusion

In this specific scenario, the inverse Kendall's correlations method is more effective for estimating the correlation coefficient in a bivariate t-distribution. However, for dimensions greater than 2 (d > 2), the computational complexity and memory requirements increase significantly, making the direct method potentially more practical for higher-dimensional data. The exercise demonstrates the importance of choosing appropriate statistical methods based on the data structure and the desired accuracy of the results.



It is possible to see that in this case, the method that inverts Kendall's τ correlations achieves a closer mean value to the real value of $\rho = 0.5$ with a significantly lower standard deviation than the direct method. This means that in this particular case, inverse Kendall's τ correlations method is better.

Now in the case of $d > 2$, it should still be possible to use the inverse Kendall's τ correlations method by taking pairwise combinations of each dimension of the realized t-Student. However, this means that for any $d > 2$, we would need to run the same method used here $\frac{d(d-1)}{2}$ times. This would already be computationally slower and potentially more memory demanding than R's implementation of cor used in the direct method.

Exercise 72

Let's start by proving the hint. Before that, notice that since $N(t) \sim Poi(\lambda t)$, then $E[N(t)] = \lambda t$ and $Var[N(t)] = \lambda t$. Also, since $Y_i \sim Gamma(\alpha, \beta)$ then $E[Y_i] = \frac{\alpha}{\beta}$ and $Var[Y_i] = \frac{\alpha}{\beta^2}$.

Now, using the law of total expectation, we have that:

$$
\begin{aligned}
E[X(t)] &= E\left[\sum_{i=1}^{N(t)} Y_i\right] \\
&= E\left[E\left[\sum_{i=1}^{N(t)} Y_i | N(t) = N\right]\right] \\
&\text{Because of iid of } Y_i \\
&= E[N(t)E[Y_1]] \\
&\text{Because of independence between } Y_i, N \\
&= E[N(t)]E[Y_1] \\
&= \lambda t \frac{\alpha}{\beta}
\end{aligned}
$$

Similarly, using the law of total variance:

$$\begin{aligned}
Var[X(t)] &= Var\left[\sum_{i=1}^{N(t)} Y_i\right] \\
&= E\left[Var\left[\sum_{i=1}^{N(t)} Y_i | N(t) = N\right]\right] + Var\left[E\left[\sum_{i=1}^{N(t)} Y_i | N(t) = N\right]\right] \\
&= E\left[N(t)Var[Y_i]\right] + Var[N(t)E[Y_i]] \\
&= E[N(t)]Var[Y_i] + E[Y_i]^2 Var[N(t)] \\
&= \lambda t[E[Y_i^2] - E[Y_i]^2] + \lambda t E[Y_i]^2 \\
&= \lambda t E[Y_i^2]
\end{aligned}$$

Now

$$Var[Y_i] = E[Y_i^2] - E[Y_i]^2 = \frac{\alpha}{\beta}$$

Therefore, recalling that $E[Y_i] = \frac{\alpha}{\beta}$ we have that $E[Y_i^2] = \frac{\alpha^2+\alpha}{\beta^2}$

Finally, we get that:

$$E[X(t)] = \lambda t \frac{\alpha}{\beta}$$

$$Var[X(t)] = \lambda t \frac{\alpha^2 + \alpha}{\beta^2}$$

We use this to simulate the process:

```
compPoisson <- function(n, t, lambda, alpha, beta){

  single_process <- function(t, lambda, alpha, beta){
    sum(rgamma(rpois(1, lambda*t), alpha, beta))
  }

  sim <- replicate(n, single_process(t, lambda, alpha, beta))

  sample <- c(mean(sim), var(sim))
  real <- c(lambda*t*alpha/beta, lambda*t*(alpha^2 + alpha)/beta^2)

  return( list(PARAMS = c(n = n, t = t, lambda = lambda, alpha = alpha, beta = beta),
               RESULTS = data.frame(SAMPLE = sample,
                                    REAL = real,
                                    ABS.DIFF = abs(sample - real),
                                    REL.DIFF = abs(sample - real)/real)) )
}
```

We proceed now to test with different parameters:

```
compPoisson(n = 10000, t = 10, lambda = 0.5, alpha = 1, beta = 1)
```

```
## $PARAMS
##      n      t lambda  alpha   beta
##  1e+04  1e+01  5e-01  1e+00  1e+00
##
```

Why This Result is Important
Model Validation: The close match between the theoretical and simulated values confirms the correctness of the model and the simulation algorithm.
Understanding Stochastic Processes: It demonstrates the behavior of compound Poisson processes, which are important in various fields like insurance, finance, and queueing theory.
Practical Applications: Such simulations are crucial for risk assessment, pricing complex financial instruments, and modeling various real-world phenomena where events occur randomly over time.

```
## $RESULTS
##     SAMPLE REAL   ABS.DIFF    REL.DIFF
## 1 5.032222    5 0.03222159 0.006444319
## 2 9.833376   10 0.16662405 0.016662405
```

```
compPoisson(n = 10000, t = 10, lambda = 2, alpha = 15, beta = 0.2)
```

```
## $PARAMS
##       n       t  lambda   alpha    beta
## 10000.0    10.0     2.0    15.0     0.2
##
## $RESULTS
##       SAMPLE   REAL   ABS.DIFF    REL.DIFF
## 1   1504.391   1500   4.391206 0.002927471
## 2 120441.730 120000 441.729843 0.003681082
```

```
compPoisson(n = 10000, t = 10, lambda = 0.7, alpha = 3, beta = 4)
```

```
## $PARAMS
##      n      t lambda  alpha   beta
##  1e+04  1e+01  7e-01  3e+00  4e+00
##
## $RESULTS
##     SAMPLE REAL   ABS.DIFF    REL.DIFF
## 1 5.223480 5.25 0.02652000 0.005051429
## 2 5.206022 5.25 0.04397835 0.008376828
```

```
compPoisson(n = 10000, t = 10, lambda = 3, alpha = 5, beta = 0.01)
```

```
## $PARAMS
##      n      t lambda  alpha   beta
##  1e+04  1e+01  3e+00  5e+00  1e-02
##
## $RESULTS
##       SAMPLE    REAL    ABS.DIFF    REL.DIFF
## 1   14955.08   15000    44.92431 0.002994954
## 2 9075104.23 9000000 75104.22563 0.008344914
```

```
compPoisson(n = 10000, t = 10, lambda = 0.85, alpha = 8, beta = 3)
```

```
## $PARAMS
##       n       t  lambda   alpha    beta
## 1.0e+04 1.0e+01 8.5e-01 8.0e+00 3.0e+00
##
## $RESULTS
##     SAMPLE     REAL   ABS.DIFF    REL.DIFF
## 1 22.61811 22.66667 0.04855333 0.002142058
## 2 66.17521 68.00000 1.82478599 0.026835088
```

We observe that in all cases the estimated sample values of mean and variance are close to the theoretical ones calculated beforehand.

In summary, the exercise successfully uses theoretical properties of the compound Poisson-Gamma process to predict its behavior, and then validates these predictions through simulation. This approach is fundamental in stochastic modeling, providing a way to understand and predict the behavior of complex random processes

Exercise 73

a)

Notice that the real value of the integral is given by:

$$\int_1^3 x^2 dx = \frac{x^3}{3}\Big|_1^3 = \frac{26}{3}$$

Using Montecarlo simulation, take $g(x) = x^2$ and $f(x) = \frac{1}{3-1}$ for $1 \leq x \leq 3$ and 0 otherwise. That is, we sample from a $\mathcal{U}(1,3)$ distribution. We get:

```
real <- 26/3
sim <- (3-1)*mean(runif(10000, min = 1, max = 3)^2)

data.frame(Real = real,
           Monte.Carlo = sim,
           Abs.Diff = abs(real-sim),
           Rel.Diff = abs(real-sim)/real)
```

Absolute Difference Calculation (Abs.Diff = abs(real-sim)):
This calculates the absolute difference between the real value and the simulated value. It's the absolute value of realsim realsim.
Relative Difference Calculation (Rel.Diff = abs(real-sim)/real):
This calculates the relative difference, which is the absolute difference divided by the real value. It provides a measure of the difference relative to the size of the real value.

```
##       Real Monte.Carlo   Abs.Diff    Rel.Diff
## 1 8.666667    8.679198 0.01253091 0.001445874
```

b)

The real value of the integral is given by

$$\int_0^\pi sin(x)dx = -cos(x)|_0^\pi = 2$$

Similar to part a) take $g(x) = sin(x)$ and $f(x) = \frac{1}{\pi-0}$ for $0 \leq x \leq \pi$ and 0 otherwise. In this case we sample from a $\mathcal{U}(0,\pi)$ distribution. We get:

```
real <- 2
sim <- (pi-0)*mean(sin(runif(10000, min = 0, max = pi)))

data.frame(Real = real,
           Monte.Carlo = sim,
           Abs.Diff = abs(real-sim),
           Rel.Diff = abs(real-sim)/real)
```

```
##   Real Monte.Carlo    Abs.Diff    Rel.Diff
## 1    2     2.00388 0.003880029 0.001940015
```

c)

To see the real value of the third integral, it is enough to notice that $f(x) = e^{-x}$ is the density function of an exponential random variable with parameter 1. That means the value of the integral is 1, due to the exponential's support. However, we can also calculate it using integration by parts, taking $u = 1$ and $dv = e^{-x}$. Then $du = 0$ and $v = -e^{-x}$. Thus:

$$\int_0^\infty e^{-x} dx = -e^{-x}|_0^\infty - 0 = 0 - (-1) = 1$$

In this case, notice we cannot sample from a uniform that has infinity as upper limit. Therefore, we will instead use the fact stated above that $f(x) = e^{-x}$ is the density function of an exponential random variable with parameter 1, and sample from this distribution. In this case, $g(x) = \mathbb{1}_{x \geq 0}$. However, this will be redundant in our case given by definition the exponential random variable only generates non-negative numbers. Therefore, we can use Montecarlo as follows:

```r
real <- 1
sim <- mean(rexp(10000, 1))

data.frame(Real = real,
           Monte.Carlo = sim,
           Abs.Diff = abs(real-sim),
           Rel.Diff = abs(real-sim)/real)
```

```
##   Real Monte.Carlo    Abs.Diff    Rel.Diff
## 1    1    1.005588 0.005588177 0.005588177
```

d)

In this case, the value of the integral has simplified close form, therefore numerical approximations are used to determine its approximate value. In this case, we take WolfraAlpha's value as real, given by:

$$\int_0^3 sin(e^x)dx \approx 0.606124$$

Similar to parts a) and b), we take $g(x) = sin(e^x)$ and $f(x) = \frac{1}{3-0}$ for $0 \leq x \leq 3$ and 0 otherwise. In this case we sample from a $\mathcal{U}(0,3)$ distribution. We get:

```r
real <- 0.606124
sim <- (3-0)*mean(sin(exp(runif(10000, min = 0, max = 3))))

data.frame(Real = real,
           Monte.Carlo = sim,
           Abs.Diff = abs(real-sim),
           Rel.Diff = abs(real-sim)/real)
```

```
##       Real Monte.Carlo   Abs.Diff   Rel.Diff
## 1 0.606124   0.6414693 0.03534533 0.05831369
```

e)

For the final integral, notice that the function we wish to integrate corresponds to the density of a standard normal distribution. Also, notice that:

$$\int_0^2 \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx = \int_{-\infty}^2 \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx - \int_{-\infty}^0 \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx = \Phi(2) - \Phi(0) \approx 0.4772499$$

Similar to previous exercises, we take $g(x) = e^{-x^2/2}$ and $f(x) = \frac{1}{2-0}$ for $0 \leq x \leq 2$ and 0 otherwise. Again, we sample from a $\mathcal{U}(0,2)$ distribution. Notice that we take out the constant $\frac{1}{\sqrt{2\pi}}$

```r
real <- pnorm(2)-pnorm(0)
sim <- 1/sqrt(2*pi) * (2-0)*mean(exp( -runif(10000, min = 0, max = 2)^2 / 2 ))

data.frame(Real = real,
           Monte.Carlo = sim,
           Abs.Diff = abs(real-sim),
           Rel.Diff = abs(real-sim)/real)
```

```
##        Real Monte.Carlo     Abs.Diff    Rel.Diff
## 1 0.4772499   0.4779446 0.0006947718 0.001455782
```

Monte Carlo integration is a powerful tool for estimating the values of integrals, especially when traditional analytical methods are challenging to apply. The exercise demonstrates how random sampling can be used to approximate integrals with varying levels of complexity and how these estimates compare favorably with known or exact values.

Exercise 74

First, let's caculate the real value of the integral. Similar to the previous exercise, we can do integration by parts taking $u = 1$ and $dv = e^{-x}$. Then $du = 0$ and $v = -e^{-x}$. Thus:

$$\int_0^{0.5} e^{-x}dx = -e^{-x}\Big|_0^{0.5} - 0 = -\frac{1}{\sqrt{e}} + 1 \approx 0.3934693$$

We start by sampling from a uniform distribution. Like in the previous exercise, we take $g(x) = e^{-x}$ and $f(x) = \frac{1}{0.5-0}$ for $0 \leq x \leq 0.5$ and 0 otherwise. In this case we sample from a $\mathcal{U}(0, 0.5)$ distribution. We get:

```
real <- -1/sqrt(exp(1)) + 1
sim <- (0.5-0)*mean(exp(-runif(10000, min = 0, max = 0.5)))
mc_var <- var(exp(-runif(10000, min = 0, max = 0.5)))/10000

data.frame(Sampling = "Uniform",
           Real = real,
           Monte.Carlo = sim,
           Abs.Diff = abs(real-sim),
           Rel.Diff = abs(real-sim)/real,
           Var = mc_var)
```

```
##   Sampling      Real Monte.Carlo     Abs.Diff   Rel.Diff         Var
## 1  Uniform 0.3934693   0.3941624 0.0006930452 0.00176137 1.28624e-06
```

Next, we can use the fact that $f(x) = e^{-x}$ is the density function of an exponential random variable with parameter 1, and sample from this distribution. In this case, $g(x) = \mathbb{1}_{0.5 \geq x \geq 0}$.

```
real <- -1/sqrt(exp(1)) + 1
x <- rexp(10000, 1)
sim <- mean((x >= 0) & (x <= 0.5))
mc_var <- var((x >= 0) & (x <= 0.5))/10000

data.frame(Sampling = "Exponential",
           Real = real,
           Monte.Carlo = sim,
           Abs.Diff = abs(real-sim),
           Rel.Diff = abs(real-sim)/real,
           Var = mc_var)
```

Uniform Sampling: Every sample contributes to the estimate since they all fall within the integration limits. This results in a more efficient use of the samples and a lower variance.
Exponential Sampling: Many samples do not contribute to the estimate because they fall outside the integration limits. This "wasted" sampling leads to higher variance.

```
##      Sampling      Real Monte.Carlo     Abs.Diff    Rel.Diff          Var
## 1 Exponential 0.3934693      0.3942 0.0007306597 0.001856967 2.388302e-05
```

It is possible to see that both methods approximate the integral fairly well. However, the first method (sampling from an uniform distribution) has a lower variance and difference values than the second one. This happens because by sampling from a uniform distribution that already falls wothin the integration limits, all realizations of the random variable are used to calculate the mean. This means that if we take n realizations, all n are effectively useful and provide information to make the approximation better. However, if we sample from an exponential distribution as in the second part, we need to use an indicator to guarantee that the points fall within the integration limits. This means we are essentially throwing away information whenever a realization lies outside our desired interval, which reduces accuracy and increases variance.

Exercise 75

In this case, sampling from a uniform distribution that aligns with the integration limits proves to be more efficient, resulting in a lower variance for the Monte Carlo estimate. This illustrates an important principle in Monte Carlo integration: the choice of sampling distribution can significantly impact the efficiency and accuracy of the estimation.

We first introduce the portfolio composition and default probabilities:

```r
nAssets <- c(AA = 10,
             A = 25,
             BBB = 96)

pd <- c(AA = 0.0001,
        A = 0.0005,
        BBB = 0.0025)
```

We now create a function to simulate one year of the portfolio multiple times, using bernoulli random variables, where we assume independence between defaults and that a "success" (i.e. a realization of 1 by the Bernoulli) represents a default. This way, we can generate one Bernoulli realization by asset according to its respective default probability, and repeat this multiple times.

```r
port_sim <- function(n, assets, probs){
  do.call(rbind, lapply(1:n, function(i){
    c(AA = sum(rbinom(n = assets[1], size = 1, prob = probs[1])),
      A = sum(rbinom(n = assets[2], size = 1, prob = probs[2])),
      BBB = sum(rbinom(n = assets[3], size = 1, prob = probs[3]))
    )
  }))
}
```

A function to simulate default scenarios. rbinom: Generates Bernoulli trials for each obligor. It simulates whether each obligor defaults (1) or not (0) based on their PD. The function is applied to each credit rating category separately within each simulation. do.call(rbind, ...): Aggregates the results of each simulation into a matrix where each row represents one simulation of the entire portfolio.

```r
set.seed(1802) # For reproducibility and interpretations
sim <- port_sim(1e5, nAssets, pd)

head(sim)
```

```
##      AA A BBB
## [1,]  0 0   1
## [2,]  0 0   0
## [3,]  0 0   1
## [4,]  0 0   2
## [5,]  0 0   0
## [6,]  0 0   1
```

1 succes in BBB or 2 means we loose money so thats why is risky

```r
apply(sim, 2, sum)
```

```
##    AA     A   BBB
##   105  1297 23904
```

In 100,000 simulations, it is possible to observe that there were only 105 defaults total for AA, 1,297 for A and 23,904 for BBB. If we look at it in a simulation-by-simulation basis, we get the following results:

```r
rbind(apply(sim, 2, summary), sd = apply(sim, 2, sd))
```

Provides a statistical summary (minimum, first quartile, median, mean, third quartile, maximum, standard deviation) for each credit rating category.

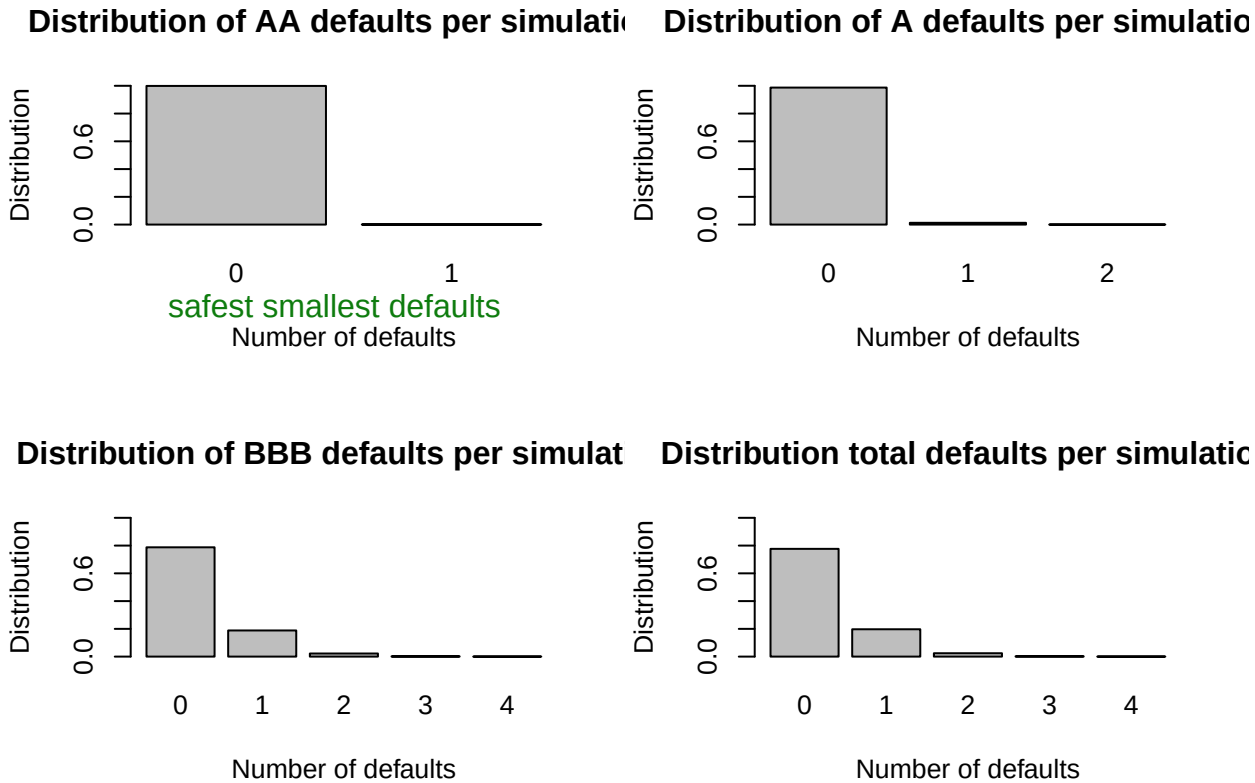
```
##                 AA         A       BBB
## Min.    0.00000000 0.0000000 0.0000000
## 1st Qu. 0.00000000 0.0000000 0.0000000
## Median  0.00000000 0.0000000 0.0000000
## Mean    0.00105000 0.0129700 0.2390400
## 3rd Qu. 0.00000000 0.0000000 0.0000000
## Max.    1.00000000 2.0000000 4.0000000
## sd      0.03238685 0.1140259 0.4893488
```

This shows that, indeed, the AA obligors are the safest, with a maximum value of 1 default per simulation in the sample, which means the 105 reported above all happened in different simulations. Following the AA come the A obligors with a maximum value of 2 defaults per simulation in the sample, and finally the

BBB with a maximum of 4. It is important to note that all of them have a mean value less than 1 default per simulation. However, there is indeed more volatility as the credit rating decreases (which makes sense intuitively but is confirmed by the numbers), as demonstrated by the increasing standard deviation with decrease of rating.

We now look at the default distributions via histograms:

```
par(mfrow = c(2,2))
barplot(table(sim[,1])/1e5,
        main = "Distribution of AA defaults per simulation",
        ylim = c(0, 1),
        xlab = "Number of defaults",
        ylab = "Distribution")
barplot(table(sim[,2])/1e5,
        main = "Distribution of A defaults per simulation",
        ylim = c(0, 1),
        xlab = "Number of defaults",
        ylab = "Distribution")
barplot(table(sim[,3])/1e5,
        main = "Distribution of BBB defaults per simulation",
        ylim = c(0, 1),
        xlab = "Number of defaults",
        ylab = "Distribution")
barplot(table(apply(sim, 1, sum))/1e5,
        main = "Distribution total defaults per simulation",
        ylim = c(0, 1),
        xlab = "Number of defaults",
        ylab = "Distribution")
```



Since some of the values are too small to be legible, we include a table of values as well. Note that the values are in percentages:

```r
data.frame( AA = c(table(sim[,1])/1e5, 0, 0, 0)*100,
            A = c(table(sim[,2])/1e5, 0, 0)*100,
            BBB = c(table(sim[,3])/1e5)*100,
            TOTAL = c(table(apply(sim, 1, sum))/1e5)*100)
```

```
##       AA      A    BBB  TOTAL
## 0 99.895 98.713 78.743 77.630
## 1  0.105  1.277 18.828 19.696
## 2  0.000  0.010  2.224  2.428
## 3  0.000  0.000  0.192  0.230
## 4  0.000  0.000  0.013  0.016
```

The code effectively simulates default scenarios for a credit portfolio with different credit ratings and their respective probabilities of default. It uses Monte Carlo methods to generate a large number of simulations, then aggregates and analyzes these simulations to understand the distribution and risk of defaults in the portfolio. This approach is valuable in risk management, providing insights into the potential default risk of a credit portfolio.