SIMULATION EXERCISES IN FINANCE

KARTHIK IYER

This collection of solutions is based on assignments for a graduate level course on simulation of models of equity and fixed-income markets applied to derivative security pricing and risk management that I attended. The following topics were covered in the course:

- Generating random variables: inverse transform method, composition method, acceptance-rejection method. Generating correlated random variables: correlated normal copula method
- Simulating stochastic processes;
- Simulation output analysis; Variance reduction techniques: control variates, antithetics, conditional Monte Carlo, stratified sampling, and importance sampling;
- Pricing of financial derivatives, including European, American, and exotic derivatives; Price sensitivity (Greeks) estimation;
- Risk estimation via simulation, e.g. loss probability, value at risk risk measures, as well as credit portfolio default risks;

The assignments were a mix of written and programming questions. R programming language was used for implementation of the simulation procedures. This document was typeset using LaTex with all of the R code examples embedded into the document and evaluated using the wonderful package **knitr** (written by Yihui Xie).

All errors are entirely mine. Please let me know if you spot any. Comments, suggestions and ideas are always welcome.

1. Exercises

(1) Write a Monte Carlo simulation to generate 10000 samples of a Poisson random variable. Take $\lambda = 1$, and report the mean and variance from your samples.

Solution: We use the inverse transform method to generate samples of a Poisson random variable with mean 1. (See page 128 in *Monte Carlo Methods in Financinal Engineering* by *Paul Glasserman* for the algorithm.)

```
rm(list=ls()) #Removes all objects
MonteCarloPoisson <- function(lambda,u)
{
  j <- 0
  p <- exp(-lambda)
  f <- p

  while(u > f)
  {
    p <- (lambda*p)/(j+1)
    f <- f+p
    j <- j+1
  }
  return(j)
}
set.seed(123)
```

```r
u <- runif(10000,min=0,max=1)
Poisson_sample <- sapply(u,FUN=MonteCarloPoisson,lambda=1)
mean(Poisson_sample)

## [1] 0.9924

var(Poisson_sample)

## [1] 0.9760398
```

$\blacksquare$

(2) Generate a random variable with CDF:

$$F(x) = \int_0^\infty x^y e^{-y}\, dy \quad \text{for } 0 \le x \le 1.$$

Solution:

$\boxed{\text{Step 1}}$ First generate $U \sim U(0,1)$. Let $Y = -log(1-U)$. We note that Y is a non-negative random variable taking values in $[0,\infty)$. Moreover,

$$\begin{aligned}\mathbb{P}(Y \le x) &= \mathbb{P}(log(1-U) \ge -x)\\ &= \mathbb{P}(U \le 1-e^{-x}) = 1-e^{-x}.\end{aligned}$$

Thus the probability density function of Y, is $f_Y(y) = e^{-y}$.

$\boxed{\text{Step 2}}$ Condition on $Y = y$ to simulate a random variable X_y with the cumulative distribution function x^y and set $X = X_y$. In other words, $\mathbb{P}(X_y \le x) = \mathbb{P}(X \le x|Y = y) = x^y$. The random variable X_y can be simulated by inverse transform method similar to as done in Step 1.

$\boxed{\text{Step 3}}$ We now show that X has the distribution $F(x)$. Let us first observe that

$$\mathbb{P}(X \le x|Y = y) = x^y = \int_0^x y t^{y-1}\, dt. \tag{0.1}$$

Thus the conditional probability density of the random variable X_y is $f_{X|Y}(x|y) = yx^{y-1}$. By Step 1, the joint probability density function of (X,Y) is

$$f(x,y) = f_Y(y)\cdot f_{X|Y}(x|y) = ye^{-y}x^{y-1}, \quad (x,y) \in [0,1]\times[0,\infty)$$

Hence,

$$\begin{aligned}\mathbb{P}(X \le x) &= \int_0^\infty \int_0^x ye^{-y} t^{y-1}\, dt\, dy \text{ for } 0 \le x \le 1\\ &= \int_0^\infty x^y e^{-y}\, dy\\ &= F(x).\end{aligned}$$

$\blacksquare$

(3) Give an acceptance-rejection algorithm to generate a random variable with pdf:

$$f(x) = 2(1-x) \quad \text{for } 0 \le x \le 1. \tag{0.2}$$

Describe your choice of the density function $g(x)$ and constant a. Run your simulation code to generate 10000 samples, and compute the average number of iterations that were required until each sample was accepted.

Solution: We choose $g(x) = 1$ on $[0,1]$. Hence $max_{x\in[0,1]}\left(\frac{f(x)}{g(x)}\right) = 2$. We choose $a = 2$ and run the acceptance-rejection algorithm.

```r
rm(list=ls())
AcceptanceRejection<-function(U1)
{
  #R.V corresponding to g=1
  U2 <- runif(1,min=0,max=1)

  #count the number of attempts to generate
  #the desired R.V with pdf f(x)=2(1-x)
  i <- 1

  while(U2 >1-U1)
  {
    U2 <- runif(1,min=0,max=1)
    U1 <- runif(1,min=0,max=1)
    i <- i+1
  }
  return(c(U1,i))
}
#----------------------------
U1 <- runif(10000,min=0,max=1)
results <- sapply(U1,FUN=AcceptanceRejection)

cat("Average number of iterations till a sample with
pdf f(x)=2(1-x) was accepted:", mean(results[2,]))

## Average number of iterations till a sample with
## pdf f(x)=2(1-x) was accepted: 1.9749

#mean(results[1,])
#mean close to 1/3 which is mean of R.V with pdf f
```

This is correct, because on an average we require a tries before we accept a sample. (See page 59 in *Monte Carlo Methods in Financinal Engineering* by *Paul Glasserman* for a reference to this fact.) ■

(4) Let (V_1, V_2) in Cartesian co-ordinates, be uniformly distributed in a circle about the origin. Show that $\frac{V_1}{V_2}$ has Cauchy distribution, i.e

$$\mathbb{P}\left(\frac{V_1}{V_2} \le x\right) = \frac{1}{\pi}arctan(x) + \frac{1}{2}. \tag{0.3}$$

Use this fact to construct an algorithm to generate Cauchy variables.

Proof. We first generate a random uniform variable $U \sim U(-\pi,\pi)$. Define $(V_1,V_2) = (\sin(U),\cos(U))$. It is easy to see that (V_1,V_2) is uniformly distributed on the circle. Let S denote an arc within the circle and parametrize S by

$S = \{\theta \in [-\pi, \pi] : \theta_1 \leq \theta \leq \theta_2\}$. Then

$$\mathbb{P}[(X,Y) \in S] = \frac{1}{2\pi} \int_{\theta_1}^{\theta_2} \sqrt{\cos^2(\theta) + \sin^2(\theta)} d\theta = \frac{1}{2\pi}(\theta_2 - \theta_1)$$

We now note that by symmetry of V_2 about the origin,

$$\begin{aligned} \mathbb{P}[\frac{V_1}{V_2} \leq x] &= 2\mathbb{P}[\frac{V_1}{V_2} \leq x \cap V_2 > 0] \\ &= 2\mathbb{P}[(\tan(U) \leq x \cap U \in [-\frac{\pi}{2}, \frac{\pi}{2}]] \end{aligned} \tag{0.4}$$

The last equality follows as the region $V_2 > 0$ corresponds to $\cos(U) > 0$ which happens precisely when $U \in [-\frac{\pi}{2}, \frac{\pi}{2}]$. We thus have,

$$\begin{aligned} & 2\mathbb{P}[(\tan(U) \leq x \cap U \in [-\frac{\pi}{2}, \frac{\pi}{2}]] \\ &= 2\mathbb{P}[(U \leq \arctan(x) \cap U \in [-\frac{\pi}{2}, \frac{\pi}{2}]] \\ &= \frac{2}{2\pi}(\arctan(x) - (-\frac{\pi}{2})) \\ &= \frac{1}{\pi}\arctan(x) + \frac{1}{2}. \end{aligned} \tag{0.5}$$

We now propose the following algorithm to generate a random variable V with Cauchy distribution.

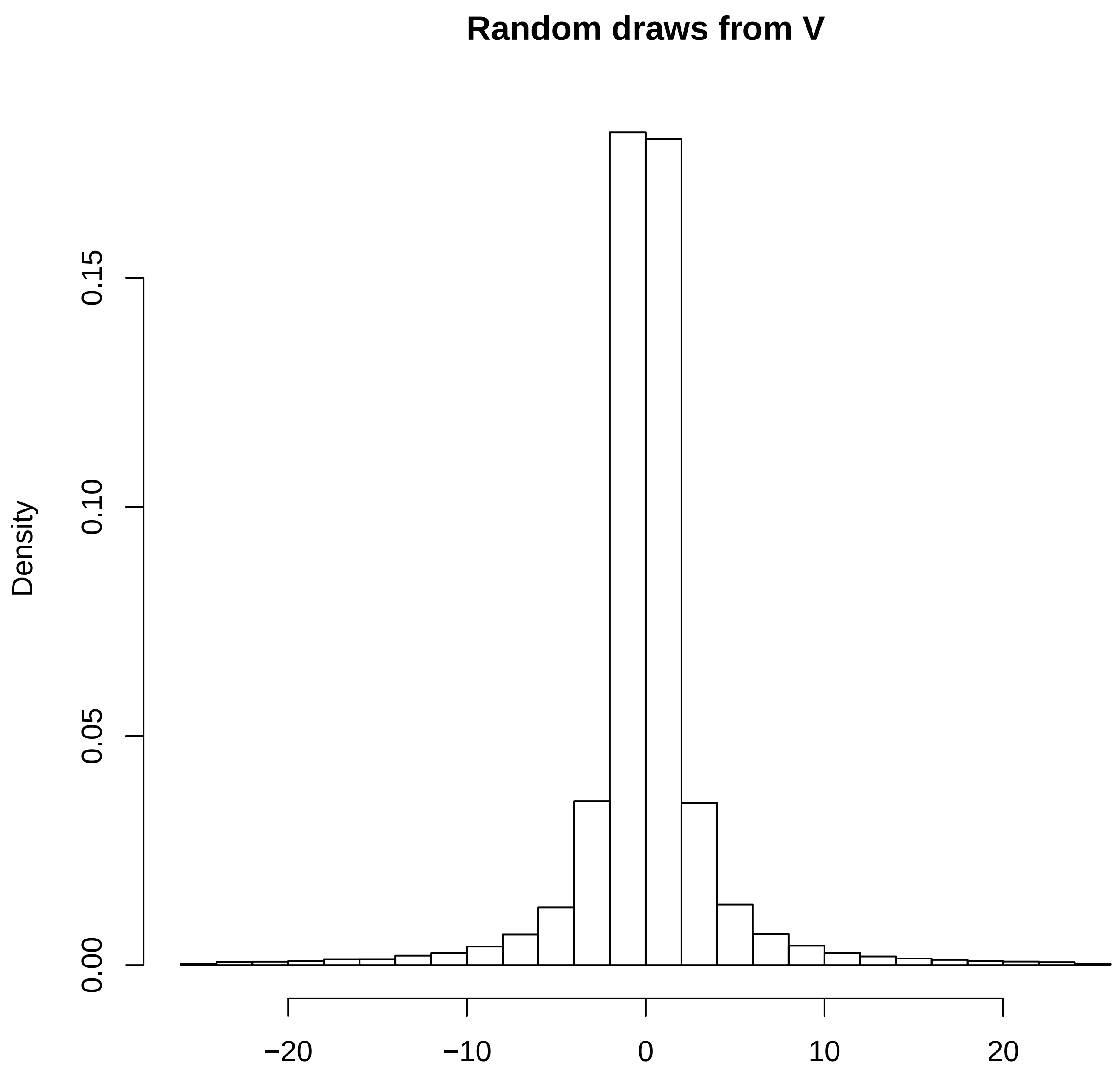
$\boxed{\text{Step 1}}$ Generate uniform random $U \sim U(-\pi, \pi)$

$\boxed{\text{Step 2}}$ Let $(X, Y) = (\sin(U), \cos(U))$.

$\boxed{\text{Step 3}}$ Set $V = \frac{X}{Y}$.

Attached is the code for the above algorithm along with a histogram.

```r
rm(list=ls())
Cauchy <- function()
{
U <- runif(100000,min=0,max=1)
X <- cos(2*pi*U)
Y <- sin(2*pi*U)
v <- X/Y
}
V <- Cauchy()
hist(V[abs(V)<25], freq=F, xlab="", main="Random draws from V")
```



```
#Try out V[abs(V)<c] for different values of c to see that
#V indeed has a Cauchy distribution.
```

■

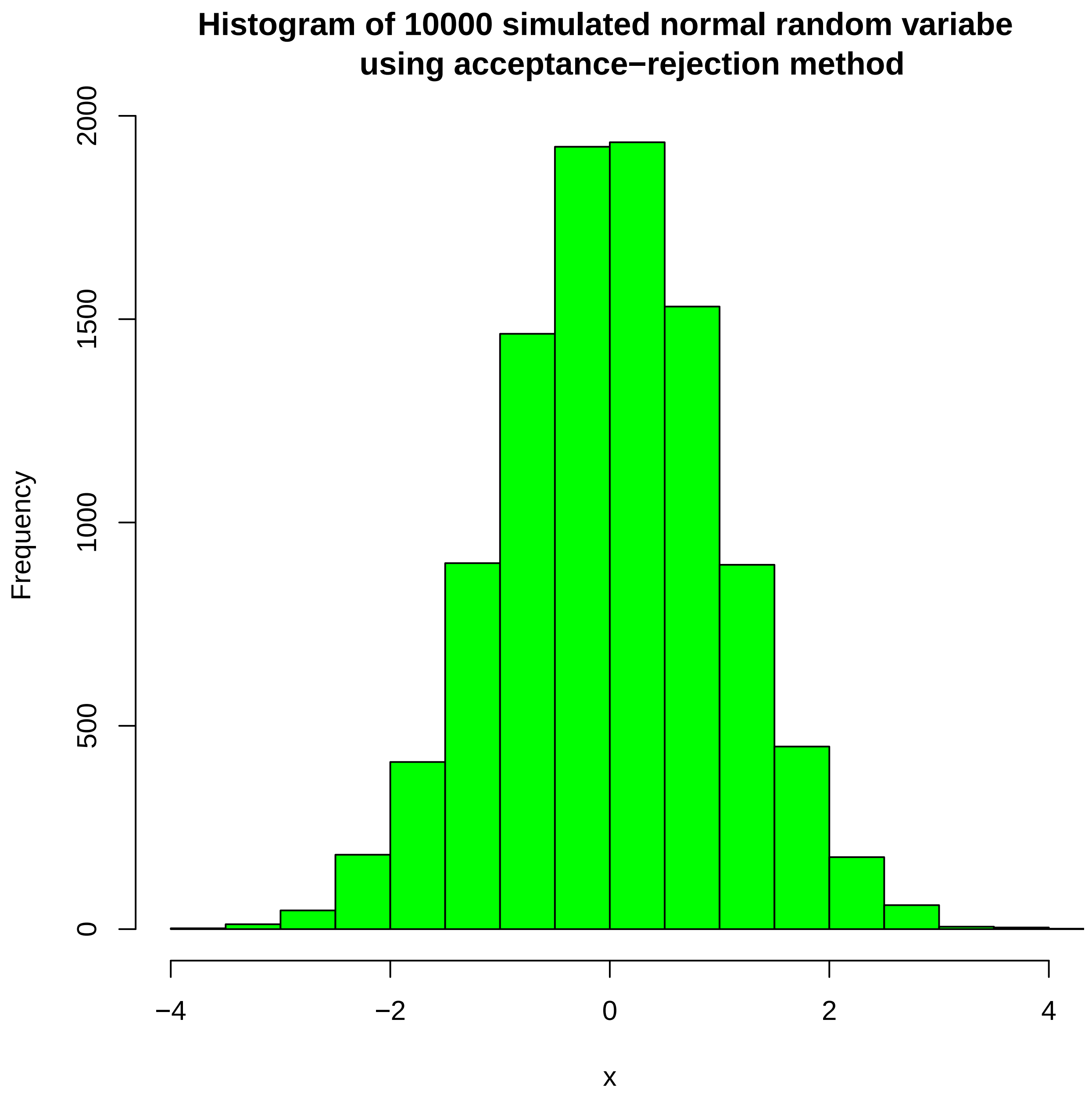
(5) Recall the acceptance-rejection algorithm for simulating $X \sim N(0,1)$ that utilizes

$$g(x) = 0.5e^{x}\mathbb{1}_{(-\infty,0]}(x) + 0.5e^{-x}\mathbb{1}_{(0,\infty)}(x) \tag{0.6}$$

Write a simulation code to implement this algorithm to generate 10000 samples of X. Show the histogram of the samples. Estimate $\mathbb{E}[X]$ and $Var(X)$. How many uniforms are used (on average) to generate one sample of X?

Solution: The code below uses acceptance-rejection algorithm to generate standard normal random variable using double exponential random variable. (See page 61 in *Monte Carlo Methods in Financinal Engineering* by *Paul Glasserman* for the acceptance rejection method of generating normal random variable from a double exponential.)

```r
rm(list=ls())
#--------------------------------------------------
#Acceptance-Rejection algorithm
AcceptanceRejectionNormal <- function(U1)
{
  a = 1.315 # max(f/g)
  count <- 1

  #Generate Y using composition method.
  #Requires one more uniform apart from U1
  U2 <- runif(1, min=0, max=1)
  if (U1 >= 0.5)
  {
    Y <- log(U2)
  }
  else
  {
    Y <- -log(U2)
  }

  U3 <- runif(1, min=0, max=1)
  while(U3 > (f(Y)/(a*g(Y))))
  {
    #Generate Y again
    U1 <- runif(1, min=0, max=1)
    U2 <- runif(1, min=0, max=1)
    if (U1 >= 0.5)
    {
      Y <- log(U2)
    }
    else
    {
      Y <- -log(U2)
    }

    #Generate unifrom U3
    U3 <- runif(1, min=0, max=1)

    #increment counter
    count <- count + 1
  }
  return(c(Y,count))
}

#-----------------------------------
f <- function(x)
{
  fx <- (1/(sqrt(2*pi)))*exp(-0.5*x*x)
}
#----------------------------------
g <- function(x)
{
  if(x <= 0)
  {
    gx <- 0.5*exp(x)
  }
  else
  {
    gx <- 0.5*exp(-x)
  }
  return(gx)
}
#----------------------------------
U1 <- runif(10000, min=0, max=1)
results <- sapply(U1, FUN=AcceptanceRejectionNormal)

hist(results[1,],
     col = "green",
     xlim = c(-4,4),
     xlab="x",
     main="Histogram of 10000 simulated normal random variabe
     using acceptance-rejection method")
```



```
mean(results[2, ]) #Mean of the number of loops required to

## [1] 1.3082

                   #generate one standard normal.

mean(results[1, ]) #Mean of the simulated standard normal rv.

## [1] 0.01578049
```

```
var(results[1, ])  #Variance of the simulated standard normal rv.

## [1] 1.005786
```

To count the average number of uniforms we need to generate 1 sample of $N(0,1)$, we multiply the expected no. of loops by 3 (which is the number of uniforms used in each loop): $3E[N] = 3 \times mean(numberofloops) \approx 4$. ■

(6) Write a simulation code to implement the Box-Muller algorithm for generating two rv's (X,Y) that are IID $N(0,1)$. Use this to estimate $\mathbb{E}[X]$, $\mathbb{E}[Y]$, $Var(X)$, $Var(Y)$, and $cov(X,Y)$. You should check the estimates with the theoretical values (which are obvious without computation).

Solution: We implement the Box-Muller algorithm to generate a pair of independent standard normal random variables. (See page 66 in *Monte Carlo methods in Financial engineering* by *Paul Glasserman* for the Box-Muller algorithm.)

```
options(digits=5)
rm(list=ls())
BoxMuller <- function(){
  uniform1 <- runif(10000, min=0, max=1)
  uniform2 <- runif(10000, min=0, max=1)

  X <- sqrt(-2*log(uniform1))* cos(2*pi*uniform2)
  Y <- sqrt(-2*log(uniform1))*sin(2*pi*uniform2)

  parameters <- c(mean(X),
                  mean(Y),
                  var(X),
                  var(Y),
                  cov(X,Y))

  names(parameters) <- c("E[X]","E[Y]","var(X)","var(Y)",
                         "Cov(X,Y)")
  parameters
}

BoxMuller()

##       E[X]       E[Y]     var(X)     var(Y)   Cov(X,Y)
## -0.0069333 -0.0058652  1.0122846  1.0188959  0.0055517

#Results match pretty well with theoretical values.
```

■

(7) Consider the CIR process

$$dX_t = (\mu - X_t)dt + \sigma\sqrt{X_t}dW_t, \quad t \geq 0 \tag{1.1}$$

with positive constants μ, σ and initial value $X_0 > 0$. Define a new process $V(X_t) = \sqrt{X_t}, t \geq 0$. There are two ways to simulate this.

(a) Write down the SDE for $V(X_t)$, and then use this SDE to simulate V_t over time using Euler's method.

Proof. Using Itô's lemma, we get the following SDE for $Y_t = \sqrt{X_t}$

$$dY_t = [\frac{\mu}{2Y_t} - \frac{Y_t}{2} - \frac{\sigma^2}{8Y_t}]dt + \frac{\sigma}{2}dW_t, \quad t \geq 0. \tag{1.2}$$

with $Y_0 = \sqrt{X_0}$. ■

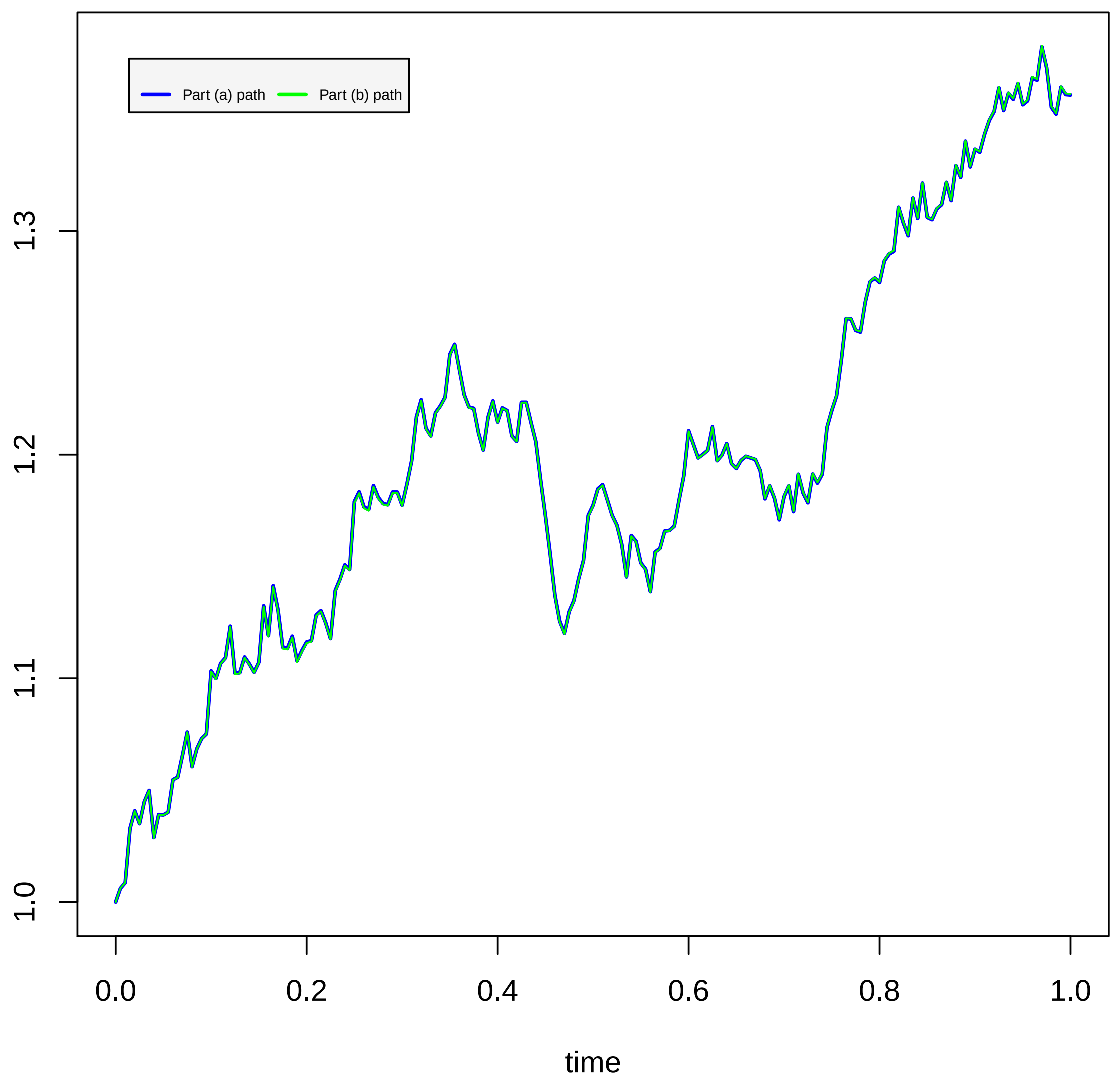
(b) You can simulate X first and then apply the function $V(x) = \sqrt{X}$.

Write the simulation codes for both methods. Take $\mu = 2, \sigma = 0.3, T = 1, N = 200, X_0 = 1$. Compare the two simulated paths (based on the same generated Brownian motion increments) in one figure. To understand their difference, compute the maximum absolute difference between the two path (over all time points).

Solution. We have simulated $V(X_t)$ with both the methods outlined in part (a) and part (b) above. For simulating $V(X_t)$ in part (a), we used the SDE (1.2).

Attached is the R code for one simulated path using the algorithm in part (a) and part (b).

```r
rm(list=ls())
#Intialization of variables
N <- 200
mu <- 2
sigma <- 0.3
T <- 1
dt <- T/N
t <- seq(0,T,by = T/N)

Y0 <- 1
Y <- rep(Y0,N) #Simulated path vector for part (a) algorithm initialized all to 1

X0 <- 1
X <- rep(X0,N) #Simulated path vector for part (b) algorithm initialized all to 1

#------------------------------------------------------------------
#Generate path according to either algorithm (a) or algorithm (b)

for(i in (1:N))
{
  temp <- rnorm(1, mean=0, sd=1)
  Y[i+1] = Y[i] + ((mu/(2*Y[i])) - Y[i]/2 -
                     sigma^2/(8*Y[i]))*dt +
    (sigma/2)*sqrt(dt)*temp
  X[i+1] = X[i] + (mu - X[i])*dt +
    sigma*sqrt(X[i])*sqrt(dt)*temp
}
#----------------------------------------------
plot(t, Y, type="l", lwd=2, col="blue", xlab="time",
     ylab="")
lines(t, sqrt(X), col="green")

legend("topleft",
       inset=.05,
       cex = 0.5,
       title="",
       c("Part (a) path","Part (b) path"),
       horiz=TRUE,
       lty=c(1,1),
       lwd=c(2,2),
       col=c("blue","green"),
       bg="grey96")
```



```
cat("Max difference is ", abs(max(sqrt(X) - Y)))

## Max difference is  0.000497556
```

■

(8) Suppose a store opens for business between $t = 0$ and $t = 10$ and that arrivals to the store during $[0, 10]$ constitute a non-homogeneous Poisson process with intensity function $\lambda(t) = \frac{(2+t+t^2)}{100}$. (The fact that the intensity function is increasing might reflect the fact that rush hour occurs at the end of the time period.) Each arrival is equally likely to spend \$100, \$400 or \$900.

(a,c) Use the thinning algorithm to simulate arrivals to the store and to estimate the average amount of money and the variance of the amount of money that is spent in $[0, 10]$. (Simulate at least 10000 samples for your estimate.)

Solution. We use the thinning algorithm and estimate the average amount of money spent and the variance of the amount of money spent by customers in the store.

```r
rm(list=ls())
Poissonnonhomo <- function(U1, final_t)
{
  t <- 0
  I <- 0 #counts the number of arrivals
  lambda_max <- 1.12 #max value of lambda(t) on [0,final_t]
  t <- t - log(U1)/lambda_max
  amount_spent <- 0

  while(t < final_t)
  {
    U2 <- runif(1, min=0, max=1)
    if (U2 <= lambda(t)/lambda_max)
    {
      I <- I+1

      #We now introduce another uniform rv U3
      #to take care of the amount spent by each customer
      U3 <- runif(1, min=0, max=1)

      if (U3 <= 1/3)
      {
        amount_spent <- amount_spent + 100
      }
      else if (1/3 <= U3 && U3 <= 2/3)
      {
        amount_spent <- amount_spent + 400
      }

      else
      {
        amount_spent <- amount_spent + 900
      }

    }
    U1 <- runif(1, min=0, max=1)
    t <- t - log(U1)/lambda_max
  }
  return(c(I, amount_spent))
}
#-----------------------------
#intensity function
lambda <- function(t)
{
  return (0.01*(2 + t + t*t))
}
```

```
#-----------------------------
U1 <- runif(10000, min=0, max=1)
results <- sapply(U1, FUN=Poissonnonhomo, final_t=10)

cat("Average number of customers in [0,10] is ", mean(results[1,]))

## Average number of customers in [0,10] is  4.0561

cat("Average amount of money spent in store in [0,10]
    is $", mean(results[2,]))

## Average amount of money spent in store in [0,10]
##     is $ 1897.25

cat("Variance of amount of money spent in store in [0,10]
    is $", var(results[2,]))

## Variance of amount of money spent in store in [0,10]
##     is $ 1352607
```

■

(b) Now compute analytically the expected amount of money that is spent in $[0,10]$ and compare it to your estimate.

Proof. Let X denote the number of arrivals to the store during $[0,10]$. We know that X is a non-homogeneous Poisson random variable with intensity $\lambda(t) = \frac{2+t+t^2}{100}$.

Let Y denote amount each arrival spends in the store. We know Y is a discrete random variable taking values $100, 400, 900$ with equal probability $1/3$.

We need to find $\mathbb{E}[XY]$. Becuase X and Y are independent, $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$. It is easy to see that $\mathbb{E}[Y] = \frac{100+400+900}{3} = \frac{1400}{3}$.

Since X is a non-homogeneous Poisson process, we know that $\mathbb{E}[X] = \int_0^{10} \frac{2+t+t^2}{100} = \frac{121}{30}$. Hence, $\mathbb{E}[XY] = \frac{121}{30}\frac{1400}{3} = 1882.22$. This analytical value is in close agreement with the 'simulated' value found in part (a).

■

(9) Let T be the investment horizon. You can invest in a stock S and the riskless money market account. Denote by S_t and B_t the prices at time t of the stock and the money market account, respectively. Assume that $S_t \sim GBM(\mu, \sigma)$ and that $B_t = exp(rt)$, $t \in [0, T]$. Suppose you can trade at the m equally spaced time points: $\{t_i = iT/m : i = 0, ..., m-1\}$.

Our trading strategy is a constant proportion trading strategy. That is, at each trading point t_i, we re-balance our portfolio so that $\alpha\%$ of our wealth W_{t_i} is invested in the stock, and $(1-\alpha)\%$ is invested in the money market account.

You may assume that $T = 1$ years and that $m = 12$ so that we re-balance our portfolio every month. Assume $W_0 = \$100,000, S0 = \$100, B0 = \$1, r = 5\%, \mu = 15\%, \sigma = 20\%$ and $\alpha = 60\%$.

Write a simulation algorithm to (1) show the distribution of W_T (say, in a histogram) and estimate its mean and variance; (2) estimate the probability that $W_T/W_0 \leq p$ where p is some fixed constant. Run your program for values of $p = .8, .9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5$. You should use at least $20,000$ samples and plot $\mathbb{P}(W_T/W_0 \leq p)$ against p.

Solution. We run the simulation with the code provided below and have produced the desired plot and histogram. As instructed, we have used the "constant proportion strategy". Let us outline in words what our code essentially does.

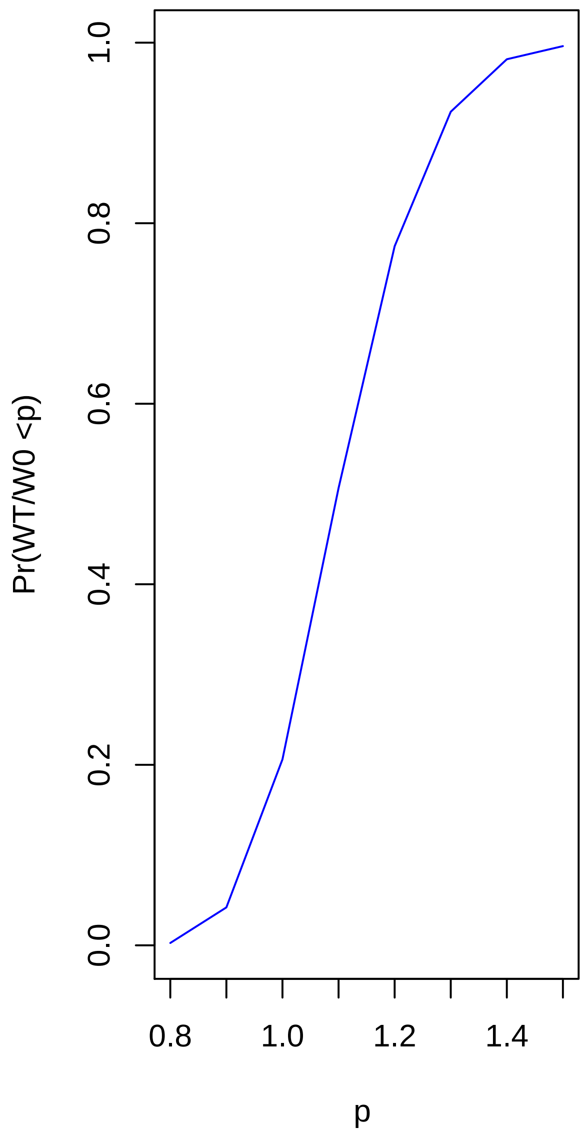
To wit, at time $t = 0$, we know 60% of the initial wealth is invested in the stocks, which means \$6000 is initially invested in the stocks, which means at time $t = 0$, the number of shares we have in the stock is $n_a = 60$ since $S_0 = 100$. Similary, at time $t = 0$, we own $n_m = 4000$ shares in the money market account. We now simulate the stock price using GBM and estimate S_{t1}. Similarly we evaluate B_{t_1}. Using the values of n_a and n_b calculated at time $t = 0$, we compute W_{t_1}.

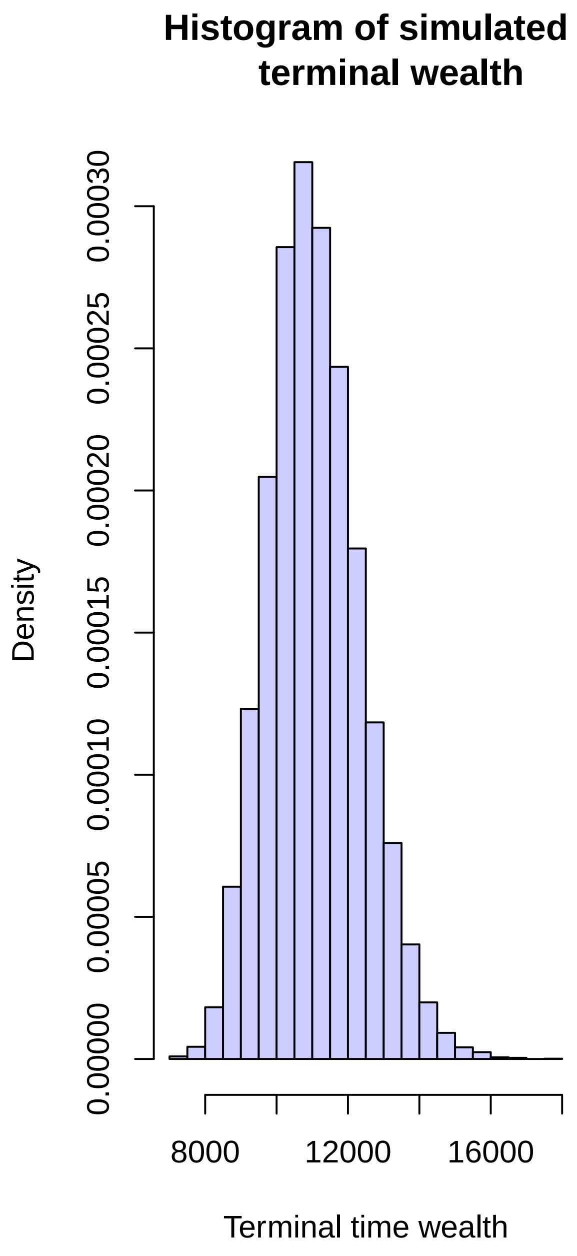
The next (and crucial) step is to rebalance at time t_1 so that proportion of money in stocks is α% and proportion of money in money market account is $(1-\alpha)$%. This amounts to *updating* n_a and n_m.

We continue like this for times $t_2, .., t_m = T$ and thus estimate W_T.

```
rm(list=ls())
S0 <- 100
B0 <- 1
r <- 0.05
mu <- 0.15
sigma <- 0.2
final_t <- 1
m <- 12 #number of times when portfolio is re-balanced
W0 <- 10000 #initial wealth
alpha <- 0.6 #weight in stock

na <- alpha*W0/S0 #initial number of shares in stock

nm <- (1- alpha)*W0/B0 #initial number of shares in money market

P <- seq(0.8, 1.5, by=0.1) #vector of proportions
                         #of Terminal wealth/initial wealth

number_of_simulations <- 20000
#------------------------------------------------------------
#Simulate stock price path

stock_price_simulation <- function(S0, mu, sigma, final_t, m)
{
  S <- rep(S0, m)
  dt <- final_t/m

  for (i in 2:m)
  {
    S[i] <- S[i-1]*exp((mu - 0.5*sigma*sigma)*dt
                     + sigma*sqrt(dt)*rnorm(1))
  }
  return(S)
}
#-----------------------------------------------------------------
#Simulate money market account. This is completely deterministic.
```

```r
money_market_account <- function(B0, r, final_t, m)
{
  B <- rep(B0, m)
  dt <- final_t/m

  for(i in 2:m)
  {
    B[i] <- B[i-1]*exp(r*dt)
  }
  return(B)
}
#-----------------------------------------------------------
#Compute portfolio wealth at m discrete times.

portfolio_wealth <- function(W0, S0, B0, mu, r, sigma, final_t, m, na, nm)
{
  wealth_vector <- rep(W0,m)
  stock_price <- stock_price_simulation(S0, mu, sigma, final_t, m)
  money_market <- money_market_account(B0, r,final_t,m)
  for (i in 2:m)
  {
    wealth_vector[i] <- na*stock_price[i] + nm*money_market[i]

    na <- alpha*wealth_vector[i]/stock_price[i]       #rebalancing
    nm <- (1- alpha)*wealth_vector[i]/money_market[i] #rebalancing
    wealth_vector[i] <- na*stock_price[i] + nm*money_market[i]
  }
  return(wealth_vector[m])
}
#-----------------------------------------------------------------
#Compute terminal wealth for 20000 simulations

simulated_wealth_terminal_time <- as.vector(rep(W0, number_of_simulations))
simulated_wealth_terminal_time <- sapply(simulated_wealth_terminal_time,
                                         FUN=portfolio_wealth,
                                         S0=S0,
                                         B0=B0,
                                         mu=mu,
                                         r=r,
                                         sigma=sigma,
                                         final_t=final_t,
                                         m=m,
                                         na=na,
                                         nm=nm)
#-----------------------------------------------------------------
#Count number of times (WT/W0 < P)
probability_estimation <- function(P)
{
  indicator_count <- 0
  indicator_count <-
    length(simulated_wealth_terminal_time
           [simulated_wealth_terminal_time <= P*W0])
  return(indicator_count)
}
#--------------------------------------------------------------
#Pr(WT/WO <P)calculation

portfolio_wealth_ratio <- rep(1, length(P))
portfolio_wealth_ratio <- sapply(P, FUN=probability_estimation)

portfolio_wealth_ratio<-
portfolio_wealth_ratio/length(simulated_wealth_terminal_time)
#-------------------------------------------------------------
#Plotting

old.par <- par(mfrow=c(1, 2))

plot(P, portfolio_wealth_ratio, type="l",
     col="blue", xlab="p ", ylab="Pr(WT/WO <p)")

hist(simulated_wealth_terminal_time, col="#CCCCFF",
     freq=F, xlab="Terminal time wealth",
     main ="Histogram of simulated
     terminal wealth")
```





```
par(old.par)
#----------------------------------------------------------
#Displaying mean and variance of simulated terminal wealth

cat("Mean of terminal wealth of portfolio is $ ",
    mean(simulated_wealth_terminal_time))

## Mean of terminal wealth of portfolio is $  11066.92

cat("Variance of terminal wealth of portfolio is $ ",
    var(simulated_wealth_terminal_time))

## Variance of terminal wealth of portfolio is $  1660351
```

∎

(10) Consider a price process $S_t, t \geq 0$. The running maximum of S at time t is defined by $\overline{S_t} := max_{0 \leq u \leq t} S_u$. To measure the distance between the current price and its running maximum, we consider the difference

$$D_t = \frac{\overline{S_t}}{S_t} - 1.$$

which is called the relative drawdown of S at time t. This is an essential concept used in evaluating investment managers and their strategies. For many hedge funds, the managers would try to avoid a large drawdown.

Suppose S is a GBM, with parameters $T = 1, \mu = 5\%, \sigma = 15\%, S_0 = 1$. Simulate a path of S together with the associated path of $\overline{S}$ over the period $[0, T]$.

Next, plot the path of the relative drawdown D (in percentage) over time. This is typically called the "underwater chart".

In addition, the maximum relative drawdown over $[0, T]$ is defined by

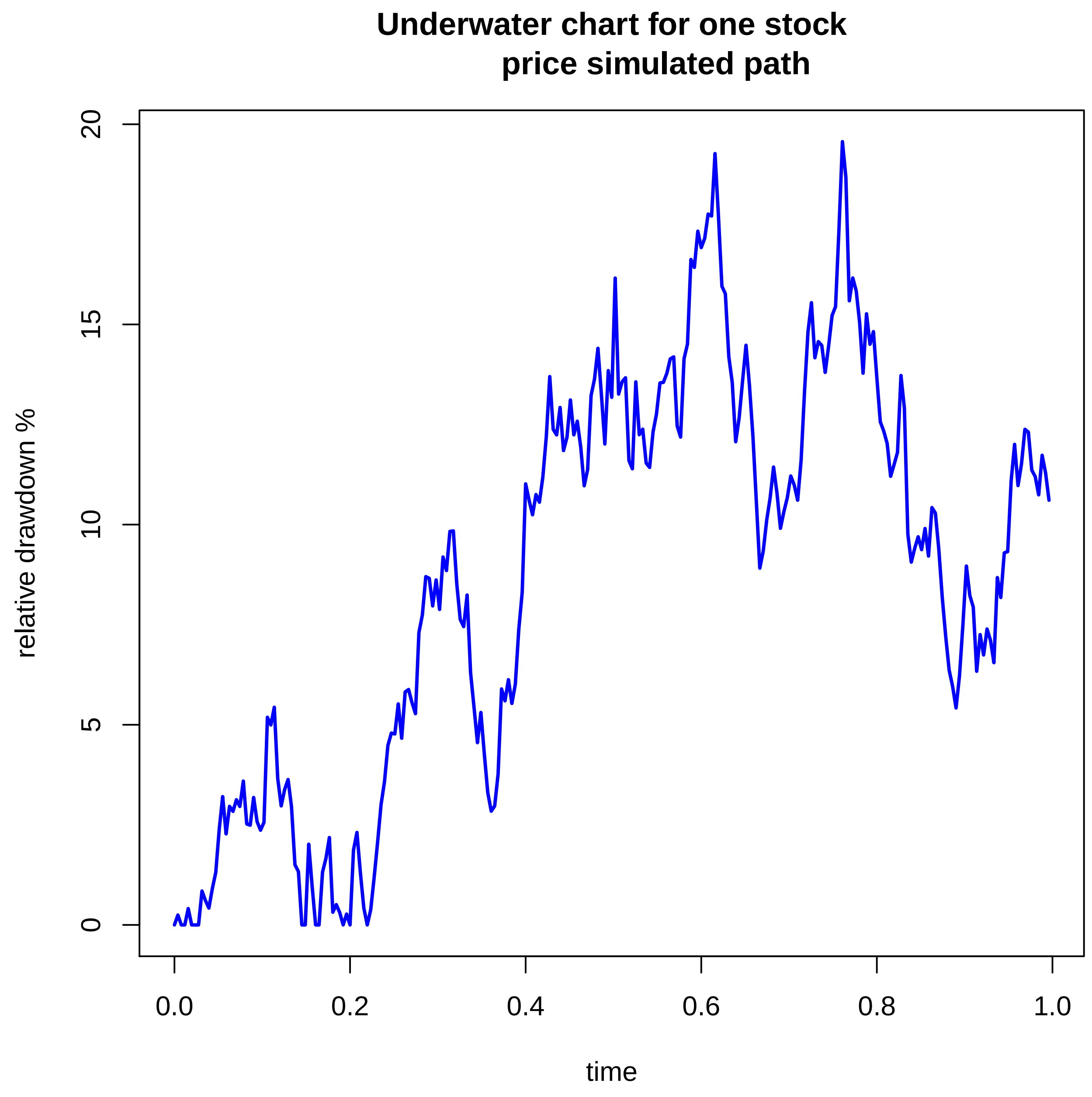
$$D_T^* = max_{0 \leq t \leq T} D_t$$

For D_T^* , plot its (estimated) expected value when $\sigma = 10\%, 12\%, \ldots, 40\%$, using at least 10000 samples each and holding other parameters fixed. How do the expected values vary with the volatility parameter σ?

Solution. Attached is the R code along with the "underwater chart" for $\sigma = 0.15$. Here we have used $m = 255$ independent Brownian motions to simulate the stock price path for $t \in [0, 1]$.

```
rm(list=ls())

S0 <- 1
mu <- 0.05
sigma <- 0.15
final_t <- 1
m <- 255
#-------------------------------------------------------
#Stock price simulation

stock_price_simulation <- function(S0, mu, sigma, final_t, m)
{
  S <- rep(S0, m)
  dt <- final_t/m

  for (i in 2:m)
  {
    S[i] <- S[i-1]*exp((mu - 0.5*sigma*sigma)*dt
                    + sigma*sqrt(dt)*rnorm(1))
  }
  return(S)
}
#---------------------------------------------------
#Running maximum calculation
```

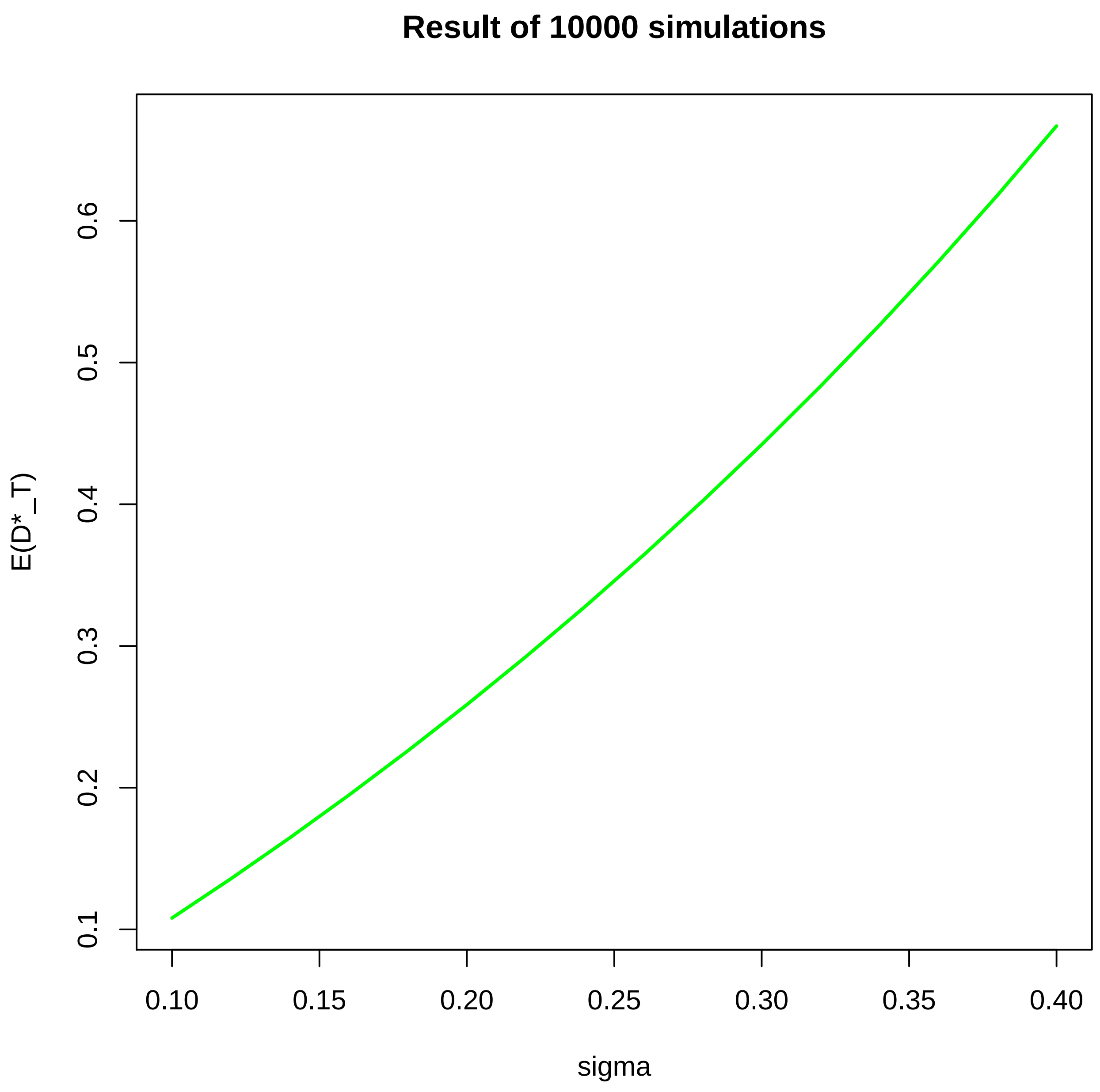
```
running_maximum_simulation <- function(stock_price_path)
{
  running_max <- rep(1, S0)
  for (i in 2:m)
  {running_max[i] = max(stock_price_path[1:i])}
  return(running_max)
}
#-------------------------------------------------------
stock_price_path <- stock_price_simulation(S0, mu, sigma, final_t, m)

running_max <- running_maximum_simulation(stock_price_path)

relative_drawdown <- running_max/stock_price_path - 1

time <- seq(0, final_t-final_t/m, by=final_t/m)

#-----------------------------------------------------
#Plotting

plot(time, relative_drawdown*100, type="l", lwd=2,
     col="blue",
     xlab="time",
     ylab="relative drawdown %",
     main="Underwater chart for one stock
          price simulated path")
```



Next is the R code and the plot for maximum relative drawdown vs the standard deviation σ keeping other paramters fixed. As we can see, the maximum relative drawdown increases linearly with σ. For this code, we chose 10000 sample stock price path with each path having $m = 255$ Brownian motions.

```
rm(list=ls())
S0 <- 1
mu <- 0.05
final_t <- 1
m <- 255
sigmavector <- seq(0.1, 0.4, by=0.02)
number_of_simulations <- 10000
```

```r
#---------------------------------------------------------------------------
#This function calculates a stock path with m Brownian motion increments
#and computes the maximum relative drawdown
drawdown_sim <- function(S0, mu, sigma, final_t, m, normal_vector)
{
  stock_price_vector <- rep(S0, m+1)
  running_max <- rep(S0, m+1)

  dt <- (final_t/m)

  for (i in 2:m+1)
  {
    temp <- exp((mu - 0.5*sigma*sigma)*dt+ sigma*sqrt(dt)*normal_vector[i-1])
    stock_price_vector[i] <- stock_price_vector[i-1]*temp

    running_max[i] <- max(stock_price_vector[1:i])
  }
  drawdown <- (running_max/stock_price_vector) - 1
  return(max(drawdown))

}
#------------------------------------------------------------------------------
#10000*255 matrix that stores the max drawdowns
mymat <- matrix(0, nrow=number_of_simulations, ncol=length(sigmavector))
for (j in 1:nrow(mymat))
{
  normal_vector <- rnorm(m, mean=0, sd=1)
   mymat[j,] <- sapply(sigmavector,
                       FUN=drawdown_sim,
                       S0=S0,
                       mu=mu,
                       final_t=final_t,
                       m=m,
                       normal_vector=normal_vector)
}
#-------------------------------------------------------------------------------
plot(sigmavector,
     apply(mymat, 2, mean),
     type="l", lwd=2,
     col="green", xlab="sigma", ylab="E(D*_T)")
     title(main="Result of 10000 simulations")
```


Result of 10000 simulations
E(D*_T)
sigma
0.1
0.2
0.3
0.4
0.5
0.6
0.10
0.15
0.20
0.25
0.30
0.35
0.40

■

(11) Consider the random variable:

$$V = min\{X_1 + X_4, X_1 + X_3, X_2 + X_3, X_2 + X_4\},$$

where $X_i's$ are independent and $X_i \sim exp(\lambda_i), i = 1,2,3,4$. The objective is to estimate the probability $\mathbb{P}\{V > v\}$ for a large constant v. Describe a simulation algorithm that uses importance sampling with a new distribution $X_i \sim exp(\nu_i), i = 1,2,3,4$ (all independent).

Proof. We are given that $X_i \sim exp(\lambda_i)$ for $i = 1,..,4$ with probability density function $f_i(y) = \lambda_i e^{-\lambda_i y}$ for $y \geq 0$ and $f_i(y) = 0$ otherwise. Moreover we note that

$$\mathbb{P}(V > v) = \mathbb{P}(X_1 + X_4 > v, X_1 + X_3 > v, X_2 + X_3 > v, X_2 + X_4 > v)$$

We will use the method of tilted densities to obtain the sampling distribution. Let $t > 0$ be such that $t < min\{\lambda_i : i = 1,,,4\}$. Let $M_{t_i} = \mathbb{E}_{f_i}[e^{tX_i}]$ and define for $x \geq 0$,

$$f_{t,i}(x) = \frac{e^{tx} f_i(x)}{M_x(t)}$$

It is easy to see that

$$f_{t,i}(x) = (\lambda_i - t)e^{-(\lambda_i - t)x}$$

We may then write

$$\begin{aligned}\theta &= \mathbb{P}(X_1 + X_4 > v, X_1 + X_3 > v, X_2 + X_3 > v, X_2 + X_4 > v) \\ &= \mathbb{E}_t[\mathbb{1}_{(X_1+X_4>v, X_1+X_3>v, X_2+X_3>v, X_2+X_4>v)} \prod_{i=1}^{4} \frac{f_i(X_i)}{f_{t,i}(X_i)}]\end{aligned}$$

where $E_t[.]$ denotes expectation with respect to the $X_i's$ under the tilted densities, $f_{i,t}()$, and $M_i(t)$ is the moment generating function of X_i. Our likelihood ratio will thus be $\prod_{i=1}^{4} \frac{f_i(x_i)}{f_{t,i}(X_i)}$.

Let $i = 1,..,r$. Since v is large, we need to sample more often from the regions where X_i is large and hence we use tilted densitites $f_{i,t}$ where $t > 0$. As mentioned before, we choose $f_{i,t}(x_i) = (\lambda_i - t)e^{-(\lambda_i - t)x_i}$ for $x_i \geq 0$. We will hence sample from a new distribution $Y_i \sim exp(\nu_i)$ where $\nu_i = \lambda_i - t$ where Y_i are independent. To ensure, $\nu_i > 0$, we choose t to be small enough. For instance, $t < min\{\lambda_i : i = 1,2,3,4\}$ or $t = \frac{v}{2}$. ■

(12) *Black-Scholes Gammas and Deltas*

Estimate the Delta and Gamma of Black-Scholes Puts. For Deltas, use the pathwise (PW) and likelihood ratio (LR) methods. For Gammas, use the pure LR method, the combined LR-PW method, as well as the combined PW-LR method.

(a) Give the Deltas and Gammas from the Black-Scholes formula, and use them to check/compare with your estimates.

(b) Report the variances of your estimators.

Parameters: $S_0 = 100, \sigma = 25\%, r = 3\%, T = 0.5$, at strikes $K = 90, 100, 110$. Use at least 100,000 samples (use up to 1 million if feasible). Summarize your answers in a table.

Solution. The closed form expression for Black Scholes Put Delta and Gamma are

$$\Delta = N(d_1) - 1$$

$$\Gamma = \frac{N^{'}(d_1)}{S_0\sigma\sqrt{T}}$$

where S_0 is the current stock price, σ is the volatility of the stock, T is the expiration date, $N(x)$ is the cumulative standard normal distribution and

$$d_1 = \frac{\log(S_0/K) + (r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}$$

Note that $S_T = S_0 e^{(r-\frac{1}{2}\sigma^2)T+\sigma\sqrt{T}Z}$ where $Z \sim N(0,1)$. For estimating the delta and gamma, we use the following estimators

- **Delta using PW** $-e^{-rT}\mathbb{1}_{K>S_T}\frac{S_T}{S_0}$.
- **Delta using LR** $-e^{-rT}(K-S_T)^+\frac{Z}{S_0\sigma\sqrt{T}}$.
- **Gamma using LR** $e^{-rT}(K-S_T)^+(\frac{Z^2-1}{S_0^2\sigma^2 T} - \frac{Z}{S_0^2\sigma\sqrt{T}})$.
- **Gamma using LR-PW** $-Ke^{-rT}\frac{Z}{S_0^2\sigma\sqrt{T}}\mathbb{1}_{S_T>K}$.
- **Gamma using PW-LR** $e^{-rT}\frac{S_T}{S_0^2}\mathbb{1}_{K>S_T}(-\frac{Z}{\sigma\sqrt{T}}+1)$.

(See page 404 in *Monte Carlo Methods in Financial Engineering* by *Paul Glasserman* for a derivation of the estimators.)

Attached is the R code for Delta and Gamma put using various methods (PW, LR, LR-PW and PW-LR). Note that all the estimators for Delta and Gamma that we use are unbiased.

```
# Calculation of Delta and Gamma of a BS put
# using different methods
#-------------------------------------------------------
# BS Delta theoretical value

BSDelta <- function(K)
{
  d1 <- (log(S0/K) + (r + 0.5*sigma*sigma)*Tf)/(sigma*sqrt(Tf))
  return(pnorm(d1, mean=0, sd=1)-1)
}
#-------------------------------------------------------
# BS Delta using PW method
BSDeltaPW <- function(K, z)
{
  sT <- S0*exp((r - 0.5*sigma*sigma)*Tf + sigma*sqrt(Tf)*z)
  if (K < sT)
  {
    bsdeltapw <- 0 #no pay-off
  }
  else
  {
    bsdeltapw <- -exp(-r*Tf)*sT/S0
  }

  return(bsdeltapw)
```

```r
}
#--------------------------------------------------------
# LR Delta calculation.

# Advantage of LR method is that the form of the option payoff is
# irrelevant!

BSDeltaLR <- function(K, z)
{
  sT <- S0*exp((r - 0.5*sigma*sigma)*Tf + sigma*sqrt(Tf)*z)
  if (K < sT)
  {
    bsdeltalr <- 0 #zero pay-off
  }
  else
  {
    bsdeltalr <- exp(-r*Tf)*max(K-sT, 0)*z/(S0*sigma*sqrt(Tf))
  }
  return(bsdeltalr)
}
#---------------------------------------------------------
# Theoretical Gamma calculation

BSGamma <- function(K)
{
  d1 <- (log(S0/K) + (r + 0.5*sigma*sigma)*Tf)/(sigma*sqrt(Tf))
  return(dnorm(d1)/(S0*sigma*sqrt(Tf)))
}
#---------------------------------------------------------
# Gamma using LR method

BSGammaLR <- function(K, z)
{
  sT <- S0*exp((r - 0.5*sigma*sigma)*Tf + sigma*sqrt(Tf)*z)
  if (K < sT)
  {
    bsgammalr <- 0
  }
  else
  {
    second_score <- (z*z - 1)/(S0*S0*sigma*sigma*Tf) -
                      z/(S0*S0*sigma*sqrt(Tf))
    bsgammalr <- exp(-r*Tf)*max(K-sT,0)*second_score
  }
  return(bsgammalr)
}
#-----------------------------------------------------
# Gamma using LR-PW method

BSGammaLRPW <- function(K,z)
{
  sT <- S0*exp((r - 0.5*sigma*sigma)*Tf + sigma*sqrt(Tf)*z)
  if (K < sT)
  {
    bsgammalrpw <- 0
  }
  else
  {
    bsgammalrpw <- -K*exp(-r*Tf)*z/(S0*S0*sigma*sqrt(Tf))
  }
  return(bsgammalrpw)
}
#------------------------------------------------------
# Gamma using PW-LR method

BSGammaPWLR <- function(K, z)
{
  sT <- S0*exp((r - 0.5*sigma*sigma)*Tf + sigma*sqrt(Tf)*z)
  if (K < sT)
  {
    bsgammapwlr <- 0
  }
  else
  {
    temp <- -z/(sigma*sqrt(Tf)) + 1
    bsgammapwlr <- exp(-r*Tf)*sT*temp/(S0*S0)
  }
  return(bsgammapwlr)
}
#----------------------------------------------------------
# Set parameters

S0 <- 100
sigma <- 0.25
r <- 0.03
Tf <- 0.5
strikeprice <- c(90,100,110)
Z <- rnorm(100000)

#-------------------------------------------------------------
#Define a matrix to store the results of Delta Calculation

resultsofDelta <- matrix(nrow=5, ncol=3)
colnames(resultsofDelta) <- c("K=90","K=100","K=110")
rownames(resultsofDelta) <- c("DeltaFormula","DeltaPW",
                              "DeltaLR","Var of DeltaPW",
                              "Var of DeltaLR")

# Call auxillary functions and populate the matrix
for(j in 1:length(strikeprice))
{
  bsdeltapw_vector <- sapply(Z, FUN=BSDeltaPW, K=strikeprice[j])
  bsdeltalr_vector <- sapply(Z, FUN=BSDeltaLR, K=strikeprice[j])
  resultsofDelta[1,j] <- BSDelta(strikeprice[j])
  resultsofDelta[2,j] <- mean(bsdeltapw_vector)
  resultsofDelta[3,j] <- mean(bsdeltalr_vector)
  resultsofDelta[4,j] <- var(bsdeltapw_vector)
  resultsofDelta[5,j] <- var(bsdeltalr_vector)
}
resultsofDelta

##                       K=90      K=100      K=110
## DeltaFormula   -0.2208724 -0.4312309 -0.6427856
## DeltaPW        -0.2206449 -0.4319248 -0.6429344
## DeltaLR        -0.2225793 -0.4335235 -0.6458286
## Var of DeltaPW  0.1285492  0.1887053  0.1798762
## Var of DeltaLR  0.3826645  0.8430202  1.5248074

#----------------------------------------------------------------
#Define a matrix to store the results of Gamma Calculation
resultsofGamma <- matrix(nrow=7, ncol=3)
colnames(resultsofGamma) <- c("K=90","K=100","K=110")
rownames(resultsofGamma) <- c("GammaFormula","GammaLR",
                              "GammaLRPW","GammaPWLR",
                              "Var of GammaLR",
                              "Var of GammaLRPW",
                              "Var of GammaPWLR")
# Call auxillary functions and populate the matrix
for(j in 1:length(strikeprice))
{
  bsgammalr_vector <- sapply(Z, FUN=BSGammaLR, K=strikeprice[j])
  bsgammalrpw_vector <- sapply(Z, FUN=BSGammaLRPW, K=strikeprice[j])
  bsgammapwlr_vector <- sapply(Z, FUN=BSGammaPWLR, K=strikeprice[j])

  resultsofGamma[1,j] <- BSGamma(strikeprice[j])
  resultsofGamma[2,j] <- mean(bsgammalr_vector)
  resultsofGamma[3,j] <- mean(bsgammalrpw_vector)
  resultsofGamma[4,j] <- mean(bsgammapwlr_vector)
  resultsofGamma[5,j] <- var(bsgammalr_vector)
  resultsofGamma[6,j] <- var(bsgammalrpw_vector)
  resultsofGamma[7,j] <- var(bsgammapwlr_vector)
}
cat("\n")
```

```
resultsofGamma

##                          K=90        K=100        K=110
## GammaFormula     0.0167876172 0.0222314569 0.0211062193
## GammaLR          0.0169607370 0.0224873926 0.0214273484
## GammaLRPW        0.0168365031 0.0222987520 0.0211988991
## GammaPWLR        0.0168171590 0.0222827647 0.0211699565
## Var of GammaLR   0.0056849867 0.0111751281 0.0192520653
## Var of GammaLRPW 0.0009185158 0.0010642075 0.0015127723
## Var of GammaPWLR 0.0008059532 0.0007401201 0.0008120036
```

We observe that variance of the PW-LR estimator for Gamma is the least among all considered estimators for Gamma and the variance of the PW estimator is lesser than that for the LR estimator for Delta. ■

(13) Consider the terminal payoff of an exotic option

$$h(\frac{S_T}{2}, S_T) := (S_T - K_1)^+ \mathbb{1}_{S_{\frac{T}{2}} \leq L} + (S_T - K_2)^+ \mathbb{1}_{S_{\frac{T}{2}} > L}$$

Apply the likelihood ratio method to estimate the delta and vega of this option with parameters

$S_0 = 100, \sigma = 25\%, r = 4\%, T = 0.5, K_1 = 100, K_2 = 110, L = 90$. Use at least 100,000 runs.

Solution. The value of this option is

$$\begin{aligned} C_0(S_0) &= \mathbb{E}[e^{-rT} h(\frac{S_T}{2}, S_T)] \text{ (By risk-neutral pricing)} \\ &= \mathbb{E}[e^{-rT/2} \mathbb{1}_{S_{T/2} \leq L} \mathbb{E}[e^{-r(T-T/2)} (S_T - K_1)^+ | S_{T/2}]] \\ &+ \mathbb{E}[e^{-rT/2} \mathbb{1}_{S_{T/2} > L} \mathbb{E}[e^{-r(T-T/2)} (S_T - K_2)^+ | S_{T/2}]] \\ &= \mathbb{E}[e^{-rT/2} \mathbb{1}_{S_{T/2} \leq L} C_{BS}(T/2, S_{T/2}, r, \sigma, T/2, K_1)] \\ &+ \mathbb{E}[e^{-rT/2} \mathbb{1}_{S_{T/2} > L} C_{BS}(T/2, S_{T/2}, r, \sigma, T/2, K_2)] \end{aligned}$$

where C_{BS} denotes the value of a standard European call option at time $T/2$ with expiration date at time T. Note that

$$C_{BS}(T/2, S_{T/2}, r, \sigma, T/2, K_1) = N(d_1) S_{T/2} - e^{-rT/2} N(d_2) K_1$$

where

$$d_1 = \frac{\log(S_{T/2}/K_1) + (r + \frac{\sigma^2}{2})\frac{T}{2}}{\sigma\sqrt{T/2}}$$

and

$$d_2 = d_1 - \sigma\sqrt{T/2}$$

Here, N denotes the standard cumulative normal distribution. Similarly,

$$C_{BS}(T/2, S_{T/2}, r, \sigma, T/2, K_1) = N(e_1) S_{T/2} - e^{-rT/2} N(e_2) K_2$$

where

$$e_1 = \frac{\log(S_{T/2}/K_2) + (r + \frac{\sigma^2}{2})\frac{T}{2}}{\sigma\sqrt{T/2}}$$

and

$$e_2 = e_1 - \sigma\sqrt{T/2}$$

Hence, $C_0(S_0)$ can be written succintly as $C_0(S_0) = \mathbb{E}[e^{-rT/2} f(S_{T/2})]$ where

$$f(x) = \mathbb{1}_{x \leq L}(N(d_1)x - e^{-rT/2}N(d_2)K_1) + \mathbb{1}_{x > L}(N(e_1)x - e^{-rT/2}N(e_2)K_2)$$

Using the expression for Black-Scholes Delta using LR method, the estimator for Delta for the option considered in this problem is

$$e^{-rT/2} f(S_{T/2}) \frac{Z}{S_0 \sigma \sqrt{T/2}}$$

The pseudo-code would for estimatiing Delta would thus be

– Simulate $S_{T/2}$ using Z.

– Compute $e^{-rT/2} f(S_{T/2}) \frac{Z}{S_0 \sigma \sqrt{T/2}}$

– Take expectation.

Estimating vega for this option by LR method is similar. Since the discounted payoff $e^{-rT}h(S_{T/2}, S_T)$ does not explicitly depend on σ and depends on the value of stock price at $T/2$, we only need to find the score function at $T/2$. Once the score is calculated it can be multiplied by discounted payoff to estimate the delta. (This is one advantage of using Likelihood Ratio method. The form of the estimator does not depend on details of the discounted payoff.)

The score function of estimating the vega at $T/2$ is known to be $\frac{Z^2-1}{\sigma} - Z\sqrt{\frac{T}{2}}$ (See page 404 in *Monte Carlo Methods in Financinal Engineering* by *Paul Glasserman* for a derivation of this fact.) The pseudo-code would for estimatiing vega for this option thus takes the form

- Simulate $S_{T/2}$ using Z.
- Compute $e^{-rT/2}f(S_{T/2})[\frac{Z^2-1}{\sigma} - Z\sqrt{T/2}]$
- Take expectation.

The R code for estimating Delta and Vega for this exotic option is attached below.

```
rm(list=ls())
S0 <- 100
sigma <- 0.25
r <- 0.04
Tf <- 0.5
K1 <- 100
K2 <- 110
L <- 90
Z <- rnorm(100000)

#----------------------------------------------------------------
#Given a normal random value, f() calculates the option payoff

f <- function(z)
{
  bs_k1 <- 0
  bs_k2 <- 0
  STover2 <- S0*exp((r - 0.5*sigma*sigma)*(Tf/2) + sigma*sqrt(Tf/2)*z)

  d1 <- (log(STover2/K1) + (r + 0.5*sigma*sigma)*(Tf/2))/(sigma *sqrt(0.5*Tf))
  d2 <- d1 - sigma*sqrt(0.5*Tf)

  if(STover2 <= L)
  {
    bs_k1 <- pnorm(d1)*STover2 - exp(-r*0.5*Tf)*pnorm(d2)*K1
  }

  e1 <- (log(STover2/K2) + (r + 0.5*sigma*sigma)*(Tf/2))/(sigma *sqrt(0.5*Tf))
  e2 <- e1 - sigma*sqrt(0.5*Tf)

  if(STover2 >L)
  {
    bs_k2 <- pnorm(e1)*STover2 - exp(-r*0.5*T)*pnorm(e2)*K2
  }

  return(bs_k1 + bs_k2)
```

```r
}
#-------------------------------------------------------------
# Exotic option Delta estimation using LR method

deltaestimation <- function(z)
{
  (exp(-r*0.5*Tf)*f(z)*z)/(S0*sigma*sqrt(0.5*Tf))
}
#--------------------------------------------------------------
#Exotic option Vega estimation using LR method

vegaestimation <- function(z)
{
  (exp(-r*0.5*Tf)*f(z))*((z*z - 1)/(sigma) - z * sqrt(0.5*Tf))
}
#---------------------------------------------------------------------------
cat("Estimated Delta using LR method is", mean(sapply(Z, FUN=deltaestimation)))

## Estimated Delta using LR method is 0.3800503

cat("Estimated Vega using LR method is", mean(sapply(Z, FUN=vegaestimation)))

## Estimated Vega using LR method is 13.70004
```

■

(14) (Portfolio Loss Probability) Suppose your portfolio consists of n shares of stock S and m units of a put option written on S with strike K and expiration date T. Assume the usual GBM model for the stock price S.

(a) First, write down the delta-gamma approximation Q of the portfolio loss over a small time Δt. The portfolio loss can also be computed analytically in terms of the Black-Scholes formula.

Proof. Let W_t denote the value of the given portfolio at time t. Note that $W_t = nS_t + mP_t$ where P_t denotes the value of the put option at time t and S_t denotes the value of stock at time t. By Delta-Gamma approximation,

$$\begin{aligned} \Delta W &= W_{\Delta t} - W_0 \\ &\approx n\Delta S + m\Delta P \\ &\approx n\Delta S + m\left(\frac{\partial P}{\partial t}\Delta t + \frac{\partial P}{\partial S}\Delta S + \frac{1}{2}\frac{\partial^2 P}{\partial S^2}\Delta S^2\right). \end{aligned} \tag{2.1}$$

Note that,

$$\begin{aligned} \frac{\partial P}{\partial t} &= \theta_P \text{ (θ of put option)} \\ \frac{\partial P}{\partial S} &= \delta_P \text{ (δ of put option)} \\ \frac{\partial^2 P}{\partial S^2} &= \gamma_P \text{ (γ of put option)} \end{aligned}$$

Under the *Geometric Brownian motion (GBM) model,*

$$\Delta S \approx S_0(\hat{\mu}\Delta t + \sigma\sqrt{\Delta t}Z).$$

where $\hat{\mu} = r - \frac{1}{2}\sigma^2$ and $Z \sim N(0,1)$.

Hence by Ito's lemma the last line in (2.1) is approximately equal to

$$nS_0(\hat{\mu}\Delta t + \sigma\sqrt{\Delta t}Z) + m(b_0^p + b_1^p Z + b_2^p Z^2).$$

where

$$\begin{aligned} b_0^p &= \theta_P \Delta t + \delta_P S_0 \hat{\mu}\Delta t \\ b_1^p &= \delta_P S_0 \sigma\sqrt{\Delta t} \\ b_2^p &= \frac{\gamma_P}{2}\sigma^2 S_0^2 \Delta t. \end{aligned} \tag{2.2}$$

Under the constant volatility assumption in the GBM model, the analytic expressions for the put Greeks take the following form

$$\begin{aligned} \theta_P &= -\frac{S_0\phi(d_1)\sigma}{2\sqrt{T}} + rKe^{-rT}\Phi(-d_2) \\ \delta_P &= -\Phi(-d_1) \\ \gamma_P &= Ke^{-rT}\frac{\phi(d_2)}{S^2\sigma\sqrt{T}}. \end{aligned} \tag{2.3}$$

where $d_1 = \frac{\log(S_0/K)+(r+0.5\sigma^2)T}{\sigma\sqrt{T}}$, $d_2 = d_1 - \sigma\sqrt{T}$, $\phi(x)$ is the density for $N(0,1)$ and $\Phi(x)$ is the cumulative distribution function for $N(0,1)$.

Hence our loss L over the time horizon Δt can be approximated as

$$-L \approx nS_0(\hat{\mu}\Delta t + \sigma\sqrt{\Delta t}Z) + m(b_0^p + b_1^p Z + b_2^p Z^2). \tag{2.4}$$

where $b_j^p, j = 1,2,3$ are as in (2.2) and (2.3). ■

(b) Write a script to generate samples of the portfolio loss at time Δt using (i) the analytical formula and (ii) the delta-gamma approximation. Then, let's compare (i) and (ii). For each pair of realizations, show in a scatter plot to illustrate how well/poorly the delta-gamma approximation works compared to the samples from the analytical formula. You may take parameters: $S0 = 100, \sigma = 30\%, r = 5\%$, $T = 0.5, K = 100, \Delta t = 14/365, n = 2, m = 3$. (Vary Δt or σ, and observe the performance of the delta-gamma approximation - what do you observe?)

Solution: Let us first anayltically compute the portfolio loss. Note that for any $0 \le t \le T$, we have

$$W_t = nS_t + mP_t = nS_0 e^{\hat{\mu}t+\sigma\sqrt{t}Z} + m(e^{-r(T-t)}K\Phi(-d_2) - S_0\Phi(-d_1)).$$

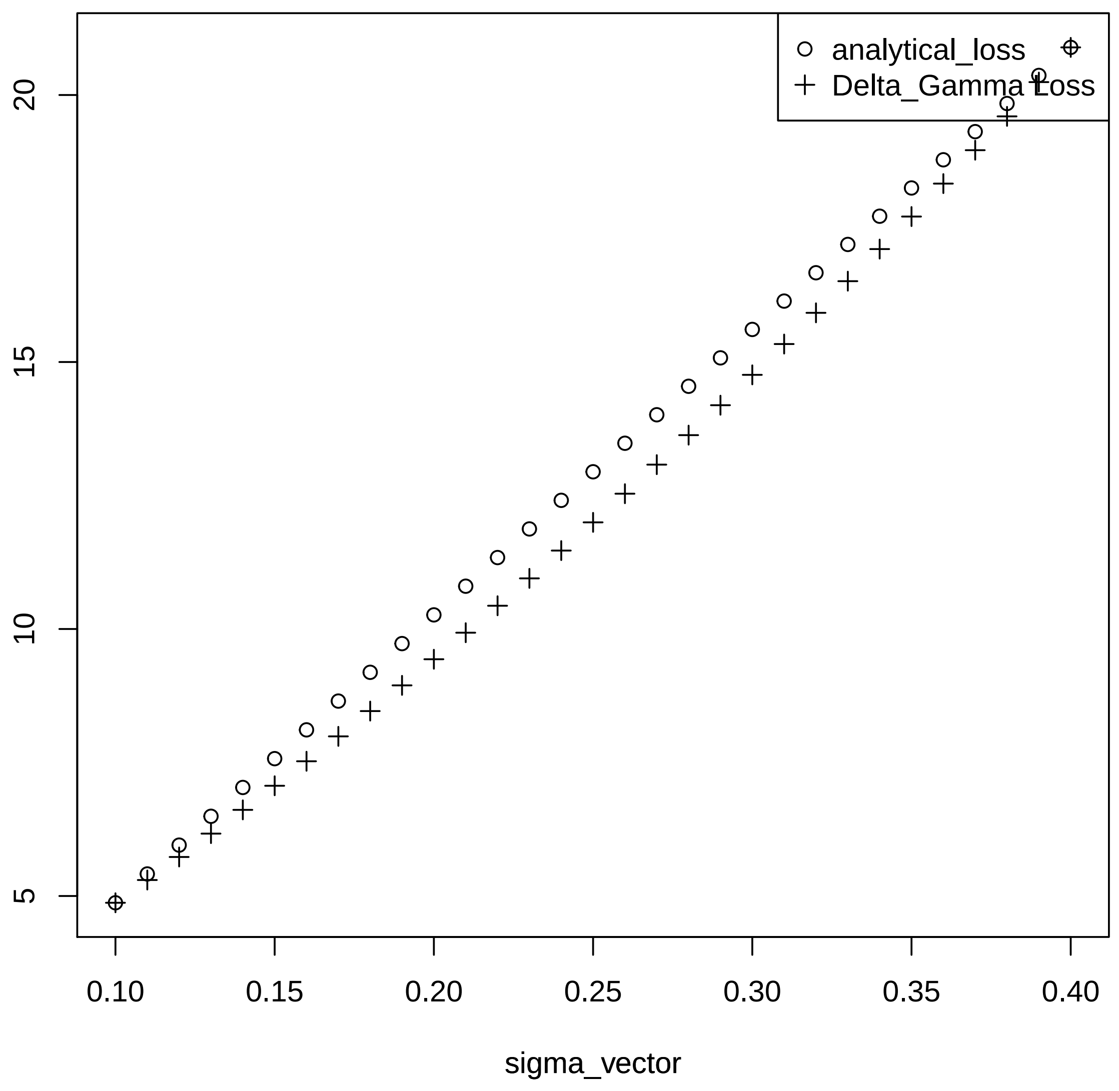
where d_1 and d_2 are evaluted at $T - t$. The portfolio loss over the time horizon Δt is thus $-L = W_{\Delta t} - W_0$.

We get the following comparison of Anayltical Loss vs Simulated Delta-Gamma Approximation loss for different values of σ and for **one** relaization of driving Brownian motion. In our code, we take σ varying from 0.01 to 0.4 in steps of 0.01.

```r
rm(list=ls())
#----------------------------------------------------------------------
#Compute Portfolio loss using Delta-Gamma approximation

delta_gamma_approx <- function(delta_t, sigma, z, terminal_time)
{

  d1 <- (log(S0/K) + (r + 0.5*sigma*sigma)*terminal_time)/(sigma*sqrt(terminal_time))
  d2 <- d1 - sigma*sqrt(terminal_time)

  hat_mu <- r - 0.5*sigma*sigma

  theta_put <- (- S0*dnorm(d1)*sigma)/(2*sqrt(terminal_time)) +
              r*K*exp(-r*terminal_time)*pnorm(-d2)

  delta_put <- -pnorm(-d1)

  gamma_put <- (K*exp(-r*terminal_time)*dnorm(d2))/(S0*S0*sigma*sqrt(terminal_time))

  bput_zero <- theta_put*delta_t + delta_put*S0*hat_mu*delta_t

  bput_one <- delta_put*S0*sigma*sqrt(delta_t)

  bput_two <- 0.5*gamma_put*sigma*sigma*S0*S0*delta_t

  stock_change <- n*S0*(hat_mu*delta_t  +sigma*sqrt(delta_t)*z)

  put_option_change <- m*(bput_zero + bput_one*z + bput_two*z*z)

  return(stock_change + put_option_change)
}
#-----------------------------------------------------------------
#Compute Portfolio value using analytical formulas
```

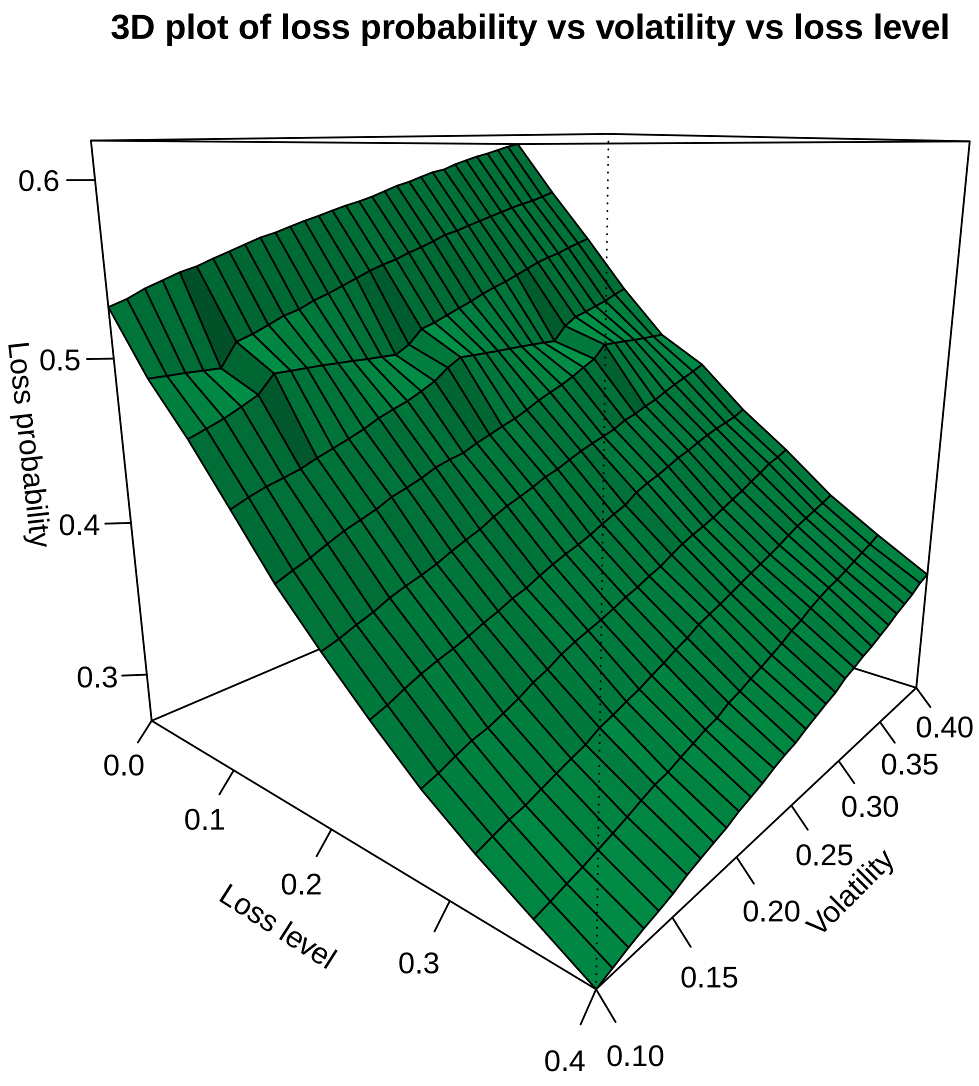
```r
#tau is time till expiry of option

analytical_portfolio_calculation <- function(delta_t, sigma, z, tau)
{
  d1 <- (log(S0/K) + (r + 0.5*sigma*sigma)*tau)/(sigma*sqrt(tau))

  d2 <- d1 - sigma*sqrt(tau)

  hat_mu <- r - 0.5*sigma*sigma

  stock_price <- n*S0*exp(hat_mu *(terminal_time-tau) +
                 sigma*sqrt(terminal_time-tau)*z)

  put_option_price <- m*(exp(-r*tau)*K*pnorm(-d2) - S0*pnorm(-d1))

  return(stock_price + put_option_price)
}
#----------------------------------------------------------------------
S0 <- 100
r <- 0.05
terminal_time <- 0.5
K <- 100
n <- 2
m <- 3
Z <- rnorm(1)
number_of_simulations <- 10000
sigma_vector <- seq(0.1, 0.4, by=0.01)
delta_t <- 14/365

W0 <- sapply(sigma_vector,
             FUN=analytical_portfolio_calculation,
             delta_t=delta_t,
             z=Z,
             tau=terminal_time)

W_deltat <- sapply(sigma_vector,
                   FUN=analytical_portfolio_calculation,
                   delta_t=delta_t,
                   z=Z,
                   tau=terminal_time-delta_t)

analytical_loss <- W0 - W_deltat

Delta_gamma_loss <- sapply(sigma_vector,
                           FUN=delta_gamma_approx,
                           delta_t=delta_t,
                           z=Z,
                           terminal_time=terminal_time)

plot(sigma_vector, analytical_loss, ylab="" )# first plot
par(new=TRUE)# second plot is going to get added to first
plot(sigma_vector, -Delta_gamma_loss, pch=3, axes=F, ylab="") # don't overwrite
```

```r
legend("topright",
       legend=c("analytical_loss","Delta_Gamma Loss"),
       pch=c(1,3))
```



(c) Estimate the probabilities of losing x% of the initial portfolio value for σ ranging from 10% to 40%. In other words, show a 3-D plot where the z-axis is the loss probability, x-axis is loss level x, and y-axis is σ.

```r
rm(list=ls())
library(plotly)

## Loading required package:  ggplot2

##
## Attaching package:  'plotly'
## The following object is masked from 'package:ggplot2':
##
##     last_plot

## The following object is masked from 'package:stats':
##
##     filter

## The following object is masked from 'package:graphics':
##
##     layout

#----------------------------------------------------------------------
#Initial set up
S0 <- 100
r <- 0.05
terminal_time <- 0.5
K <- 100
n <- 2
m <- 3
number_of_simulations <- 10000
sigma_vector <- seq(0.1, 0.4, by=0.01)
delta_t <- 14/365
Z <- rnorm(10000)
xvalues <- seq(0, 0.4, by=0.04)
#-----------------------------------------------------------------
#Compute Portfolio value using analytical formulas

#tau is time till expiry of option

analytical_portfolio_calculation <- function(delta_t, sigma, z, tau)
{
  d1 <- (log(S0/K) + (r + 0.5*sigma*sigma)*tau)/(sigma*sqrt(tau))

  d2 <- d1 - sigma*sqrt(tau)

  hat_mu <- r - 0.5*sigma*sigma

  stock_price <- n*S0*exp(hat_mu *(terminal_time-tau) +
                 sigma*sqrt(terminal_time-tau)*z)

  put_option_price <- m*(exp(-r*tau)*K*pnorm(-d2) - S0*pnorm(-d1))

  return(stock_price + put_option_price)
}
#----------------------------------------------------------------------
#Indicator function which returns 1 when
#analytical portfolio loss > x

indicator <- function(loss, x)
{
  if(loss > x)
  {return (1)}
  else
  {return (0)}
}
#-----------------------------------------------------------
#Computes the probabality that portfolio loss >x

prob_loss <- function(x, sigma, Z)
{
  W0 <- sapply(Z,
               FUN=analytical_portfolio_calculation,
               delta_t=delta_t,
               sigma=sigma,
               tau=terminal_time)

  W_deltat <- sapply(Z,
                     FUN=analytical_portfolio_calculation,
                     delta_t=delta_t,
                     sigma,
                     tau=terminal_time-delta_t)

  mean(sapply((W0-W_deltat)/(W_deltat),FUN=indicator, x=x))
}
#------------------------------------------------------------------------
mm <- matrix(0, nrow=length(xvalues), ncol=length(sigma_vector))
for( j in 1:length(xvalues))
{
  mm[j,] <- sapply(sigma_vector, FUN=prob_loss, x=xvalues[j], Z=Z)
}
persp(x=xvalues, y=sigma_vector, z=mm,
      main="3D plot of loss probability vs volatility vs loss level",
      ticktype="detailed",
      xlab="Loss level",
      ylab="Volatility",
      zlab = "Loss probability",
      theta = 40, phi = 15,
      col = "springgreen", shade = 0.5)
```



(15) (Continued) Since the loss approxmization is of the form: $L \sim a + \sum_{j=1}^{m}(b_j Z_j + \lambda_j Z_j^2) := Q$.The moment generating function of Q is $\mathbb{E}[e^{\theta Q}] = e^{a\theta}\prod_{j=1}^{m}\mathbb{E}[e^{\theta(b_j Z_j + \lambda_j Z_j^2)}], \theta \in \mathbb{R}$. Compute $\mathbb{E}[e^{\theta Q}]$. You can take $m = 1$.

Proof. For $m = 1$, the approximate loss in (2.4) takes the following form

$$-L \approx nS_0(\hat{\mu}\Delta t + \sigma\sqrt{\Delta t}Z) + m(b_0^p + b_1^p Z + b_2^p Z^2)$$
$$L \approx (-nS_0\hat{\mu}\Delta t - b_0^p) + (-b_1^p - nS_0\sigma\sqrt{\Delta t})Z - b_2^p Z^2.$$

where $b_j^p, j = 1,2,3$ are as in (2.2) and (2.3) and $Z \sim N(0,1)$

Denote $h_0 = -nS_0\hat{\mu}\Delta t - b_0^p$, $h_1 = -b_1^p - nS_0\sigma\sqrt{\Delta t}$ and $h_2 = -b_2^p$. We can succintly (and approximately) write our loss as

$$L \approx h_0 + h_1 Z + h_2 Z^2 := Q.$$

Thus for any $\theta \in \mathbb{R}$,

$$\begin{aligned}
\mathbb{E}[e^{\theta Q}] &= \frac{e^{\theta h_0}}{\sqrt{2\pi}} \int_{\mathbb{R}} e^{\theta h_1 z + (\theta h_2 - 0.5) z^2}\, dz \\
&= \frac{e^{\theta h_0 + 0.5\theta^2 h_1^2 \hat{\sigma}^2}}{\sqrt{2\pi}} \int_{\mathbb{R}} e^{-\frac{1}{2\hat{\sigma}^2}(z - \theta h_1 \hat{\sigma}^2)^2}\, dz \quad \left(\text{where } \hat{\sigma}^2 = \tfrac{1}{1-2\theta h_2}\right) \\
&= \hat{\sigma} e^{\theta h_0 + 0.5\theta^2 h_1^2 \hat{\sigma}^2} \left(\frac{1}{\sqrt{2\pi\hat{\sigma}^2}} \int_{\mathbb{R}} e^{-\frac{1}{2\hat{\sigma}^2}(z - \theta h_1 \hat{\sigma}^2)^2}\, dz \right) \\
&= \hat{\sigma} e^{\theta h_0 + 0.5\theta^2 h_1^2 \hat{\sigma}^2} \quad \text{(The expression inside the big brackets integrates to 1)}
\end{aligned}$$

■

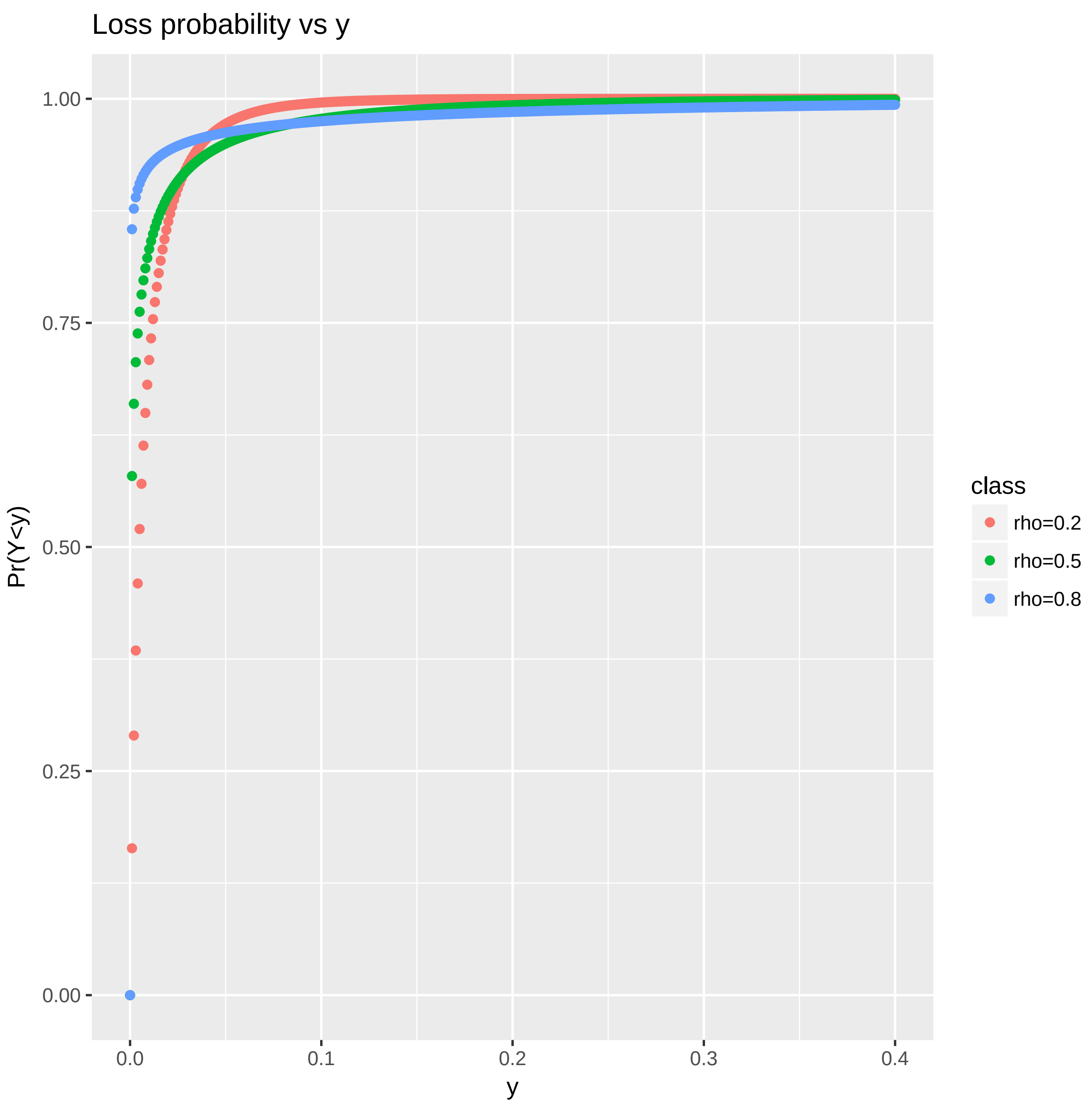
(16a) Write a script to implement the loss probability formula

$$\mathbb{P}(Y \le y) \approx \Phi\left(\frac{\Phi^{-1}(1-p) - \sqrt{1-\rho}\,\Phi^{-1}(1-y)}{\sqrt{\rho}} \right). \tag{2.5}$$

Fix $p = 1\%$. Plot $\mathbb{P}(Y \le y)$ as a function of y, for $\rho = 0.2; 0.5; 0.8$.

Solution: We first note that the approximation (2.5) holds for a portfolio where we have credit risk exposure on m diferent firms and m is very large. Y denotes the random variable $\frac{L}{m}$ where L is the number of defaults from m firms. Φ of course denotes the standard normal cumulative distribution function and ρ incorporates dependence among the defaults.

```r
rm(list=ls())
y <- seq(0,0.4, by=0.001)
p <- 0.01

prob_loss_formula <- function(y, rho)
{
  numerator <- qnorm(1-p) - sqrt(1-rho)*qnorm(1-y)
  denominator <- sqrt(rho)
  return(pnorm(numerator/denominator))
}
y1 <- sapply(y, FUN=prob_loss_formula, rho=0.2)
y2 <- sapply(y, FUN=prob_loss_formula, rho=0.5)
y3 <- sapply(y, FUN=prob_loss_formula, rho=0.8)

df <- data.frame(x=rep(y,3), y=c(y1, y2,y3),
                 class=c(rep("rho=0.2", length(y)),
                         rep("rho=0.5", length(y)),
                         rep("rho=0.8", length(y))))
library(ggplot2)

ggplot(data = df, aes(x=x, y=y, color=class)) +
  geom_point()+
  xlab("y") +
  ylab("Pr(Y<y)") +
  ggtitle("Loss probability vs y")
```



■

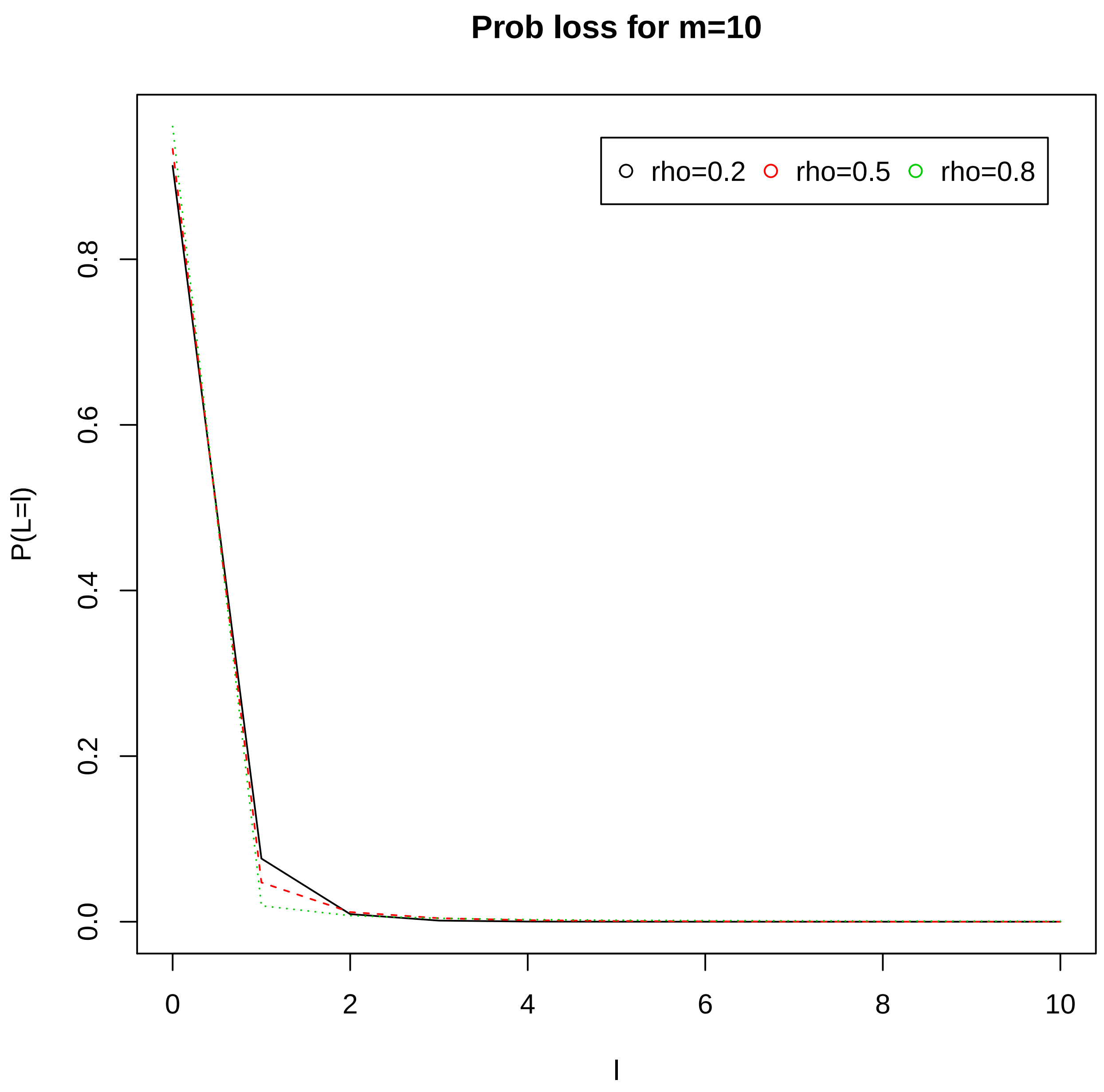
(16b) Alternatively, can numerically compute the PMF:

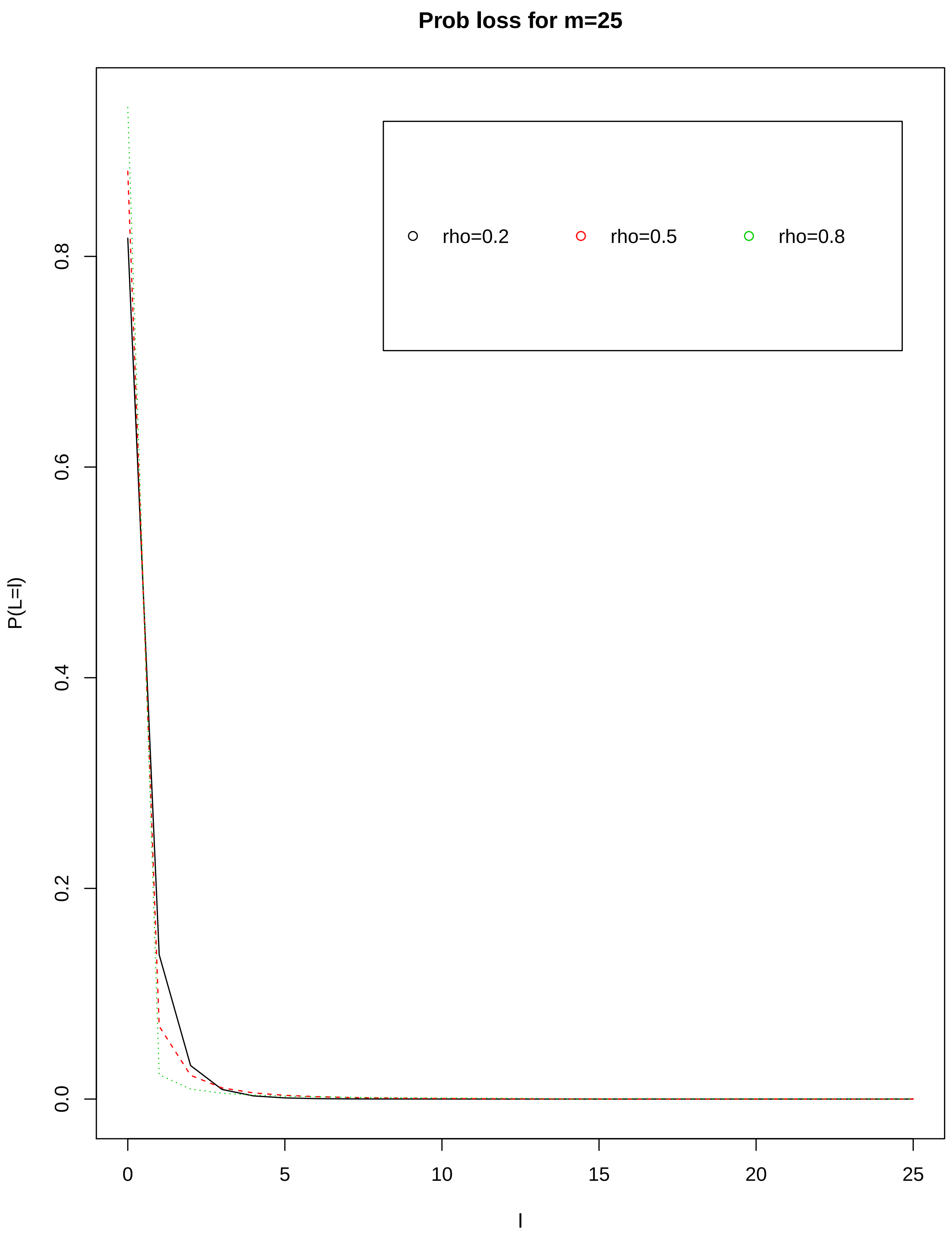
$$\mathbb{P}(L=l)=\int_{\mathbb{R}}\binom{m}{l}\tilde{p}(z)^{l}(1-\tilde{p}(z))^{m-l}f_Z(z)\,dz$$

where f_Z is the pdf of a $N(0,1)$ randdom variable and $\tilde{p}(z)$ is defined as $\tilde{p}(z)=1-\Phi(\frac{x-\sqrt{\rho}z}{\sqrt{1-\rho}})$ where $x=\Phi^{-1}(1-p)$. Take m = 10, 25, 50. It will become more computationally expensive as m increases. Again, take p = 1%. Plot the PMF $\mathbb{P}(L=l)$ as a function of l, for $\rho=0.2;0.5;0.8$.

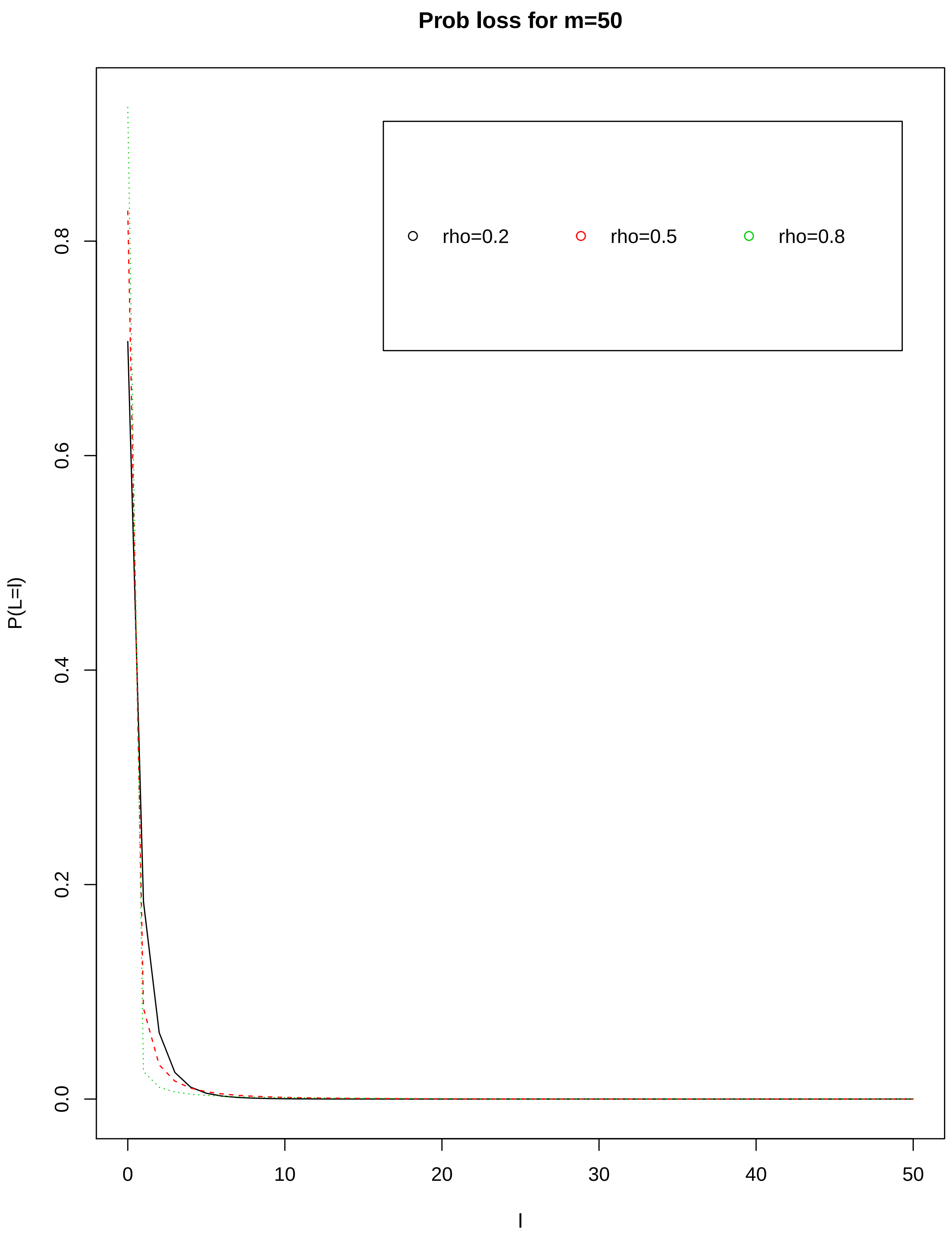
Proof. For the sake of brevity, we have only attached the R code $m = 10$.

```r
rm(list=ls())

p <- 0.01
x <- qnorm(1-p)
m <- 10
#m<-25
#m<-50

tilde_p <- function(z, rho)
{
  1- pnorm((x - sqrt(rho)*z)/sqrt(1 - rho))
}
#----------------------------------------------------------
#Computes and returns the PMF

prob_loss <- function(rho)
{
integrand <-function(z,l)
{
  factor1 <- choose(m, l)
  factor2 <- (tilde_p(z, rho))^l
  factor3 <- (1-tilde_p(z, rho))^(m-l)
  factor4 <- dnorm(z)
  return(factor1*factor2*factor3*factor4)
}

prob_loss_vector <- rep(0, m+1)
prob_loss_vector <- unlist(prob_loss_vector)

for (j in 1:(m+1))
{
  prob_loss_vector[j] <- integrate(integrand, lower=-Inf, upper=Inf, l=j-1)
}

return(prob_loss_vector)
}
#---------------------------------------------------------------
#Plotting

y1 <- unlist(prob_loss(rho=0.2))
y2 <- unlist(prob_loss(rho=0.5))
y3 <- unlist(prob_loss(rho=0.8))
vec <- c(y1,y2,y3)
dim(vec) <- c(m+1,3)
colnames(vec) <- c("rho=0.2", "rho=0.5","rho=0.8")

matplot(0:m, vec, type='l', xlab='l', ylab='P(L=l)', col=1:5)
legend('topright', inset=.05, legend=colnames(vec),
       pch=1, horiz=TRUE, col=1:5)
title('Prob loss for m=10')
```


Prob loss for m=10
rho=0.2
rho=0.5
rho=0.8
P(L=l)
l
0.0
0.2
0.4
0.6
0.8
0
2
4
6
8
10



Prob loss for m=25
P(L=l)
l
rho=0.2
rho=0.5
rho=0.8
0.0
0.2
0.4
0.6
0.8
0
5
10
15
20
25


Prob loss for m=50
P(L=l)
0.0
0.2
0.4
0.6
0.8
0
10
20
30
40
50
l
rho=0.2
rho=0.5
rho=0.8