1. Simulation of Brownian Sample Paths in R/S-Plus:

(i) Simulation of standard Brownian Motion

Assume $Y_t$ follows the process: $dY_t = dB_t$, write a program to generate 1000 paths for $Y_t$, on the time interval $[0, 1]$ with $\Delta t = \frac{1}{260}$ and $Y_0 = 0$.

To simulate the above process, we use the discrete model:

$$Y_{i+1} = Y_i + \varepsilon \sqrt{\Delta t} \quad \varepsilon \sim N(0,1)$$

Simulation algorithm:

Step 1: Set $Y_0 = 0$

Step 2: Set $Y_{(i+1)\Delta} = Y_{i\Delta} + \varepsilon_i \sqrt{\Delta} \quad \varepsilon_i \sim iid N(0,1), \ i = 0, 1, \ldots, 259$

Step 3: Repeat Step 1 - Step 2 for 1000 times

```r
# a simulation program for Brownian Motion
deltat<-1/260  # input delta t
N<-1000       # number of sample paths
T<-1   # length of each path
Y0<-0  # input initial value of Y
n<-T/deltat # number of time steps
Y<-matrix(rep(0, (n+1)*N), N, (n+1) )
# setup a matrix to store up the value of Ys

for (j in 1:N) {  # for loop to simulate 10000 sample paths
  Y[j,1]<-Y0
  for (i in 1:n) {  # for loop to generate 1 sample path
    z<-rnorm(1)    # generate 1 standard Normal r.v.
    Y[j,i+1]<-Y[j,i]+z*sqrt(deltat)
  }
}
```

The above program can generate the sample path one by one with using two for loops, which requires long time for implementation. So we develop an alternative method. Generate the sample path as a vector sense.
Simulation algorithm:

Step 1: \[ \bar{Y}_0 = 0 \]

Step 2: \[ \bar{Y}_{(i+1)} = \bar{Y}_i + \bar{\epsilon}_i \sqrt{\Delta t} \]

where \( \bar{\epsilon}_i \sim iid N(0, I) \) which is a vector of dimension \( 1000 \times 1 \), 
\( i = 0, 1, \ldots, 259 \)

---

# a simulation program for Brownian Motion

deltat<-1/260 #input delta t
N<-1000    #number of sample paths
T<-1       #length of each path
Y0<-0      #input initial value of Y
n<-T/deltat #number of time steps

Y<-matrix(rep(0, (n+1)*N), N, (n+1) ) #setup a matrix to store up the value of Ys

#for loop to generate a vector of sample path with dim
Y[,1]<-Y0
for (i in 1:n) {
  z<-rnorm(N)           #generate a vector of standard Normal r.v.
  Y[,i+1]<-Y[,i]+z*sqrt(deltat)
}

B<-t(Y)                      #take transpose for the matrix Y
a<-seq(0,1,1/260)   #setup values for the time line
ub<-max(Y)*1.1
lb<-min(Y)*1.1

matplot(a,B,type="l",col=1,xlim=c(0,1),ylim=c(lb,ub),xlab="t",ylab="B")
(ii) Simulation of standard Brownian Motion with drift and non-unit variance

Assume $X_t$ follows the process: $dX_t = \mu t + \sigma dB_t$, write a program to generate 10000 paths for $X_t$ on the time interval $[0, 1]$ with $\mu = 0.1, \sigma = 0.3$

\[ \Delta t = \frac{1}{260} \quad \text{and} \quad X_0 = 50 \]

To simulate the above process, we use the discrete model:

\[ X_t - X_{t-\Delta t} = \mu \Delta t + \sigma \epsilon \sqrt{\Delta t} \]

where $\epsilon \sim N(0,1)$

Simulation algorithm:

Step 1: Set $X_0 = 0$

Step 2: Set $\bar{X}_{(i+1)\Delta t} = \bar{X}_{i\Delta t} + \mu \Delta t + \sigma \bar{\epsilon}_i \sqrt{\Delta t}$

---

```r
# a simulation program for Brownian Motion with drift and volatility(Method 2)
deltat<-1/260    #input delta t
sigma<-0.3       #input volatility
drift<-0.1       #input drift
N<-30000         #number of sample paths
T<-1             #length of each path
X0<-50           #input initial value
n<-T/deltat      #number of time steps

X<-matrix(rep(0,(n+1)*N),N,(n+1))
X[,1]<-X0
for (i in 1:n)  {
z<-rnorm(N)
X[,i+1]<-X[,i]+drift*deltat+sigma*sqrt(deltat)*z
}
```

For simulating the stock price, it is just similar; you can try it yourself in the assignment 2. If you find any difficulties, you can discuss with the tutor.

Useful function for your assignment:

For a vector with standard normal random variables, if you want to assign values 0 to those cells, which have negative values, you can:

\[ z <- \text{rnorm}(1000) \]
\[ z [z<0] <- 0 \quad \text{#compute max}[z,0] \]
If we even don’t need to plot the simulation paths out, and just require the terminal value of the paths, we can just store the terminal values instead of the whole paths. Let see the following examples:
Simulate the same dynamics as above examples:
Using updating algorithm without storage of intermediate values

```r
# a simulation program for Brownian Motion with drift and volatility (Method 3)
deltat<-1/260    #input delta t
sigma<-0.3     #input volatility
drift<-0.1       #input drift
N<-30000      #number of sample paths
T<-1   #length of each path
X0<-50  #input initial value
n<-T/deltat #number of time steps
X<-matrix(rep(0,N),N,1)
X<-X0
for (i in 1:n)  {
    z<-rnorm(N)
    X<-X+drift*deltat+sigma*sqrt(deltat)*z
}
```

We can use the most tedious method 1, by using two for loops:

```r
# a simulation program for Brownian Motion with drift and volatility (Method 1)
deltat<-1/260    #input delta t
sigma<-0.3     #input volatility
drift<-0.1       #input drift
N<-30000      #number of sample paths
T<-1   #length of each path
X0<-50  #input initial value
n<-T/deltat #number of time steps
X<-matrix(rep(0,(n+1)*N),N,(n+1))
for (j in 1:N) {    #for loop to simulate 30000 sample paths
    X[j,1]<-X0
    for (i in 1:n)  {  #for loop to generate 1 sample path
        z<-rnorm(1) #generate 1 standard Normal r.v.
        X[j,i+1]<-X[j,i]+drift*deltat+sigma*sqrt(deltat)*z
    }
}
```
From these three programs, we have the following results:

<table>
<thead>
<tr>
<th>Method</th>
<th>Algorithm</th>
<th>Speed for simulation (Within a same computer)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method 1</td>
<td>Two for loops</td>
<td>About 10 minutes</td>
</tr>
<tr>
<td>Method 2</td>
<td>Whole path storage</td>
<td>35 seconds</td>
</tr>
<tr>
<td>Method 3</td>
<td>Terminal value storage</td>
<td>15 seconds</td>
</tr>
</tbody>
</table>