Financial Risk Forecasting
Seminar 5

Jon Danielsson
London School of Economics

Version 4.0 August 2024

©Jon Danielsson. All rights reserved

5 Simulations and functions

5.1 The plan for this week

1. Make random numbers

2. Compare distributions
3. Make functions
4. Price derivarives with simulations

5.2 Links from the R notebook

Simulations

5.3 Loading data and libraries

```
load('Returns.RData')
load('Prices.RData')
```

5.4 Random numbers

We can get random numbers drawn from a specific distribution by using the prefix `r` before the distribution name:

```
rnorm(1)
rnorm(10, mean = 10, sd = 0.5)
rt(1, df = 5)
```

Everytime we run a function that outputs random numbers, we will get a different value. Usually, to allow for replication, we set a `seed`:

```
set.seed(888)
rnorm(5)
set.seed(888)
```

```
rnorm(5)
set.seed(666)
rnorm(5)
```

5.5 Comparing distributions using random numbers

To show the fat tails of the Student-t compared to the normal distribution, we will draw 1000 points from each distribution:

```
rnd_normal <- rnorm(1000)
rnd_t <- rt(1000, df = 3)
plot(rnd_t,
  col = "red",
  pch = 16,
  main = "Random points from a Normal and Student-t"
)
points(rnd_normal, col = "blue",pch = 16)
legend("bottomright",
  legend = c("Student-t", "Normal"),
  pch = 16,
  col = c("red", "blue")
)
```

Repeat these commands a few times.

We see that the points from a Student-t distribution take on more extreme values compared to the Normal. This is a consequence of fat tails.

5.6 Assigning your grades with a random number generator

Let's say I want to assign your grades using a random number generator. Grades go from 0 to 100 and have to be integers. Drawing numbers from a distribution like the normal or the Uniform will give us non-integers, so we can either:

- Round the number to the nearest integer
- Use a different function, like `sample()`, which only outputs integers

We will use the latter. `sample(x,size)` takes a vector `x`, and a `size`, and returns a vector of the given size of random draws from `x`:

Let's say there are 60 students in the class, we can get the grades:

```
sample(1:100, 1)
sample(1:100, 3)
sample(1:100, 5)
grades <- sample(1:100, 60)
hist(grades, col = "lightgray",breaks=10)
```

Repeat this a few times.

How many people got a Distinction?

```
length(grades[grades >= 70])
```

As exercise, make a similar bar chart, except showing the LSE mark classifications in a horizontal plot, further printing the percentage getting each grade at the end of the bar.

5.7 Small and large sample properties

Let's explore the distribution of random samples of different sizes, drawn from a standard normal distribution, compared to the distribution:

```
par(mfrow=c(2,2))
norm1 <- rnorm(60)
norm2 <- rnorm(100)
norm3 <- rnorm(100000)
x <- seq(-3,3,0.1)
hist(norm1,
  freq = FALSE,
  breaks = 20,
  main = "Sample size 100",
  col = "lightgrey",
  ylim = c(0, 0.5)
)
lines(x,dnorm(x), lwd = 3, col = "red")
hist(norm2,
  freq = FALSE,
  breaks = 20,
  main = "Sample size 1000",
  col = "lightgrey",
  ylim = c(0, 0.5)
)
lines(x,dnorm(x), lwd = 3, col = "red")
hist(norm3,
  freq = FALSE,
  breaks = 20,
  main = "Sample size 100000",
  col = "lightgrey",
  ylim = c(0, 0.5)
)
lines(x,dnorm(x), lwd = 3, col = "red")
```

5.8 Make our own functions

Suppose we want to repeatedly do a calculation

```
a=10
result=a^0.5 + sin(a)
```

We can also do

```
MyFunction = function(a){
  result=a^0.5 + sin(a)
  return(result)
}
```

```
MyFunction(a=10)
MyFunction(a=3)
```

And for our returns

```
load('Returns.RData')
ReturnAnalysis = function(name){
  load('Returns.RData')
  if(! name  %in% names(Returns)) {
    cat("Don't make mistakes,",name,"is not in the data set.\n\n")
    stop("Put the correct name in!")
  }
  y=Returns[[name]]
  cat(name,"mean",mean(y),"sd",sd(y),"\n")
  plot(y,main=name)
  return(mean(y))
}
```

```
ReturnAnalysis(name="LSE")
ReturnAnalysis(name="GE")
```

5.9 Black-Scholes function

We will be using the Black-Scholes function to evaluate the one day ahead risk.

```
bs = function(K, P, r, sigma, T){
  d1 = (log(P/K) +
    (r + 0.5*sigma^2)*(T))/(sigma*sqrt(T))
  d2 = d1 - sigma*sqrt(T)
  Call = P*pnorm(d1, mean = 0, sd = 1) -
    K*exp(-r*(T))*pnorm(d2, mean = 0, sd = 1)
  Put = K*exp(-r*(T))*
    pnorm(-d2, mean = 0, sd = 1) -
    P*pnorm(-d1, mean = 0, sd = 1)
  return(list(Call=Call,Put=Put))
}
```

5.10 Simulate options

```
K=price*0.8
r=0.09
sigma=sqrt(255)*sd(y)
Maturity=1
bs(K=K, P=price, r=r, sigma=sigma, T=Maturity)
```

We can also plot the put and call prices over a range of strike prices.

```r
K=price*seq(0.3,1.7,by=0.01)
put=rep(NA,length(K))
call=rep(NA,length(K))
for(i in 1:length(K)){
  res=bs(K=K[i], P=price, r=r, sigma=sigma, T=Maturity)
    put[i]= res$Put
    call[i]= res$Call
}
matplot(K,cbind(put,call) ,type='l',lty=1,las=1)
```

5.11 Simulate option prices

First do the basic setup. Always set the seed, and then pick the number of simulations, strike price, maturity in years, the annual volatility and the annual risk free rate.

```r
seed=888
set.seed(seed=seed)
S=1e5
K=price*0.6
r=0.09
sigmaAnnual=sqrt(255)*sd(y)
Maturity=2
```

Refer to the book and slides on the necessary mathematics for simulating an option. We use the R command `pmax` because we need to apply the `max` to each row in a vector.

Note we use the `rlnorm` which is the lognormal. While it is conceptually the same as simulating from a normal and taking the exponential, `exp(rnorm())`, using the lognormal is more accurate, especially in the tails.

```r
true =bs(K=K, P=price, r=r, sigma=sigmaAnnual, T=Maturity)
ysim=rnorm(S,
  mean=r*Maturity- 0.5* Maturity* (sigmaAnnual^2),
  sd=sqrt(Maturity)*sigmaAnnual
)
Psim=price  * exp(ysim)
ysim=rlnorm(S,
  meanlog=r*Maturity-0.5* Maturity * (sigmaAnnual^2),
   sdlog=sqrt(Maturity)*sigmaAnnual
  )
Psim=price  * ysim
simPut=pmax(K-Psim,0)
simCall=pmax(Psim-K,0)
simCall=mean(simCall )* exp(-Maturity*r)
simPut=mean(simPut )* exp(-Maturity*r)
cat("put",
  "call","\n",
  true$Put,
  true$Call,
  "\n",
  simPut,simCall,
  "\n"
)
```

We only used hundred thousand simulations, which is not sufficient, and this is only accurate to 2 digits, at most. As we increase the number of simulations, the accuracy increases. The problem is that the computation time also increases, and you run out of memory in your computer.

Even if you did an infinite number of simulations, you would not get the a value identical to the Black-Scholes price because each method of calculation is subject to numerical error, and there is more scope for numerical error in the simulation. You can see that in action using alternative ways of doing the same simulation, and noting how the last digits change, for apparent no reason.

5.12 Recap

In this seminar we have covered:

- Drawing random numbers from specified distributions
- Using seeds for replicability
- Small and large sample properties of random numbers drawn from a distribution

Some new functions used:

- `rnorm()`
- `rt()`
- `if()`
- `sample()`
- `hist()`
- `set.seed()`
- `function()`

5.13 Optional exercises

1. Make a bar chart or random marks, showing the LSE mark classifications in a horizontal plot, further printing the percentage getting each grade at the end of the bar.
2. Make a plot of the number of simulations against the option price.
3. Write code that allows you to put arbitrary number of simulations for pricing an option, and try it with 10 billion simulations.
4. Explore techniques for reducing the number of required simulations for the pricing of the option, perhaps antithetic and important sampling.