Financial Risk Forecasting
Seminar Week 5: Simulations and Functions

Jon Danielsson
London School of Economics

Version 4.0

5 Simulations and functions

5.1 Why simulations and functions matter for financial risk

Simulations enable risk managers to model complex scenarios that cannot be solved analytically, such as portfolio behaviour under extreme market conditions or derivative pricing with path dependency. Functions allow us to create reusable, testable code that ensures consistency across risk calculations and reduces errors in production systems.

These programming techniques become essential for Monte Carlo VaR, stress testing and building robust risk management systems that can handle real-world complexity.

For more detail, see Simulations in the R notebook.

5.2 The plan for this week

1. Make random numbers
2. Compare distributions
3. Make functions
4. Advanced simulation techniques

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:

```r
rnorm(1)
```

```
[1] 0.2393173
```

```r
rnorm(10, mean = 10, sd = 0.5)
```

```
 [1]  9.512647  9.900175  9.891371 10.538170  9.210489  9.778042 10.168406
 [8]  9.922595 10.331867  9.731417
```

```r
rt(1, df = 5)
```

```
[1] 0.7610775
```

Every time we run a function that outputs random numbers, it produces a different value. Usually, to allow for replication, we set a `seed`:

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

```
[1] -1.9513433 -1.5443662  0.7298327 -0.2775818 -1.6562843
```

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

```
[1] -1.9513433 -1.5443662  0.7298327 -0.2775818 -1.6562843
```

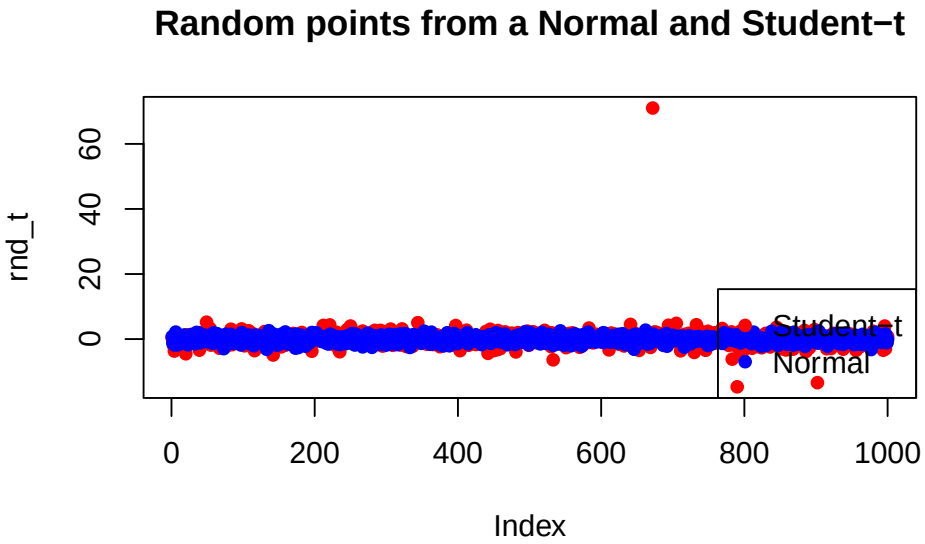
```r
set.seed(666)
rnorm(5)
```

```
[1]  0.7533110  2.0143547 -0.3551345  2.0281678 -2.2168745
```

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:

```r
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 Discrete random sampling in credit risk

The concepts we have learned about random number generation become important in financial applications, particularly in backtesting risk models. When testing how well our risk models perform, we often need to randomly sample from discrete sets of data or create bootstrap samples from historical observations.

To illustrate discrete sampling in a financial context, suppose we want to simulate credit rating assignments for a bond portfolio. Credit ratings are discrete categories and we need to sample from these categories based on historical probabilities. Drawing numbers from continuous distributions will not produce the discrete categories we need, so we use `sample()` which can handle discrete outcomes.

`sample(x, size, prob)` takes a vector `x`, a `size` and optional probabilities, returning a vector of random draws:

```
# Credit ratings from best to worst
ratings = c("AAA", "AA", "A", "BBB", "BB", "B", "CCC", "D")

# Historical probabilities for new bond issues
probs = c(0.02, 0.05, 0.15, 0.35, 0.25, 0.12, 0.05, 0.01)

# Simulate ratings for 100 bonds in a portfolio
bond_ratings = sample(ratings, 100, replace = TRUE, prob = probs)
table(bond_ratings)
```

```
bond_ratings
  A  AA AAA   B  BB BBB CCC   D
```

```
 21   6   2  15  26  24   4   2
```

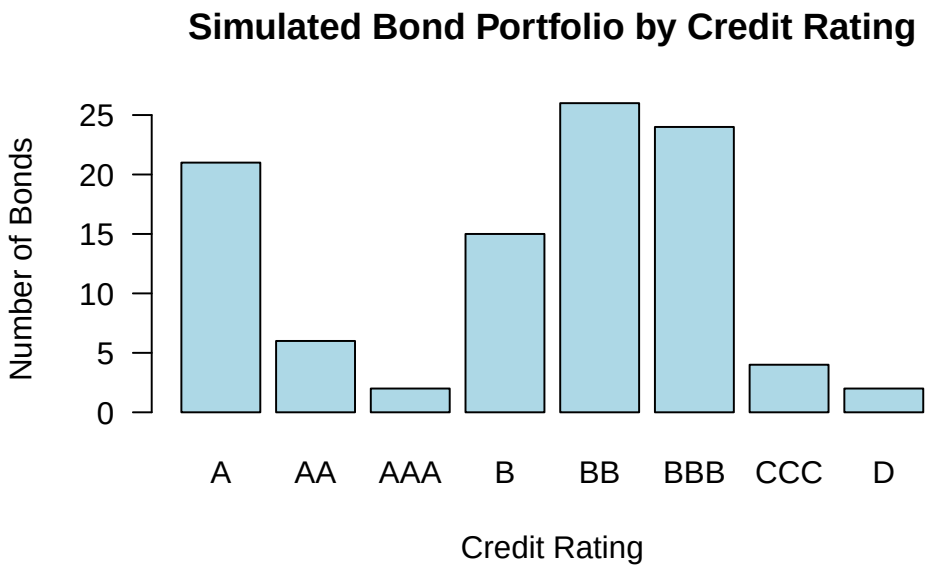
How many bonds are investment grade (BBB or better)?

```
investment_grade = sum(bond_ratings %in% c("AAA", "AA", "A", "BBB"))
cat("Investment grade bonds:", investment_grade, "out of", length(bond_ratings))
```

```
Investment grade bonds: 53 out of 100
```

We can visualize the distribution of ratings in our simulated portfolio:

```
barplot(table(bond_ratings),
        main = "Simulated Bond Portfolio by Credit Rating",
        ylab = "Number of Bonds",
        xlab = "Credit Rating",
        col = "lightblue",
        las = 1)
```



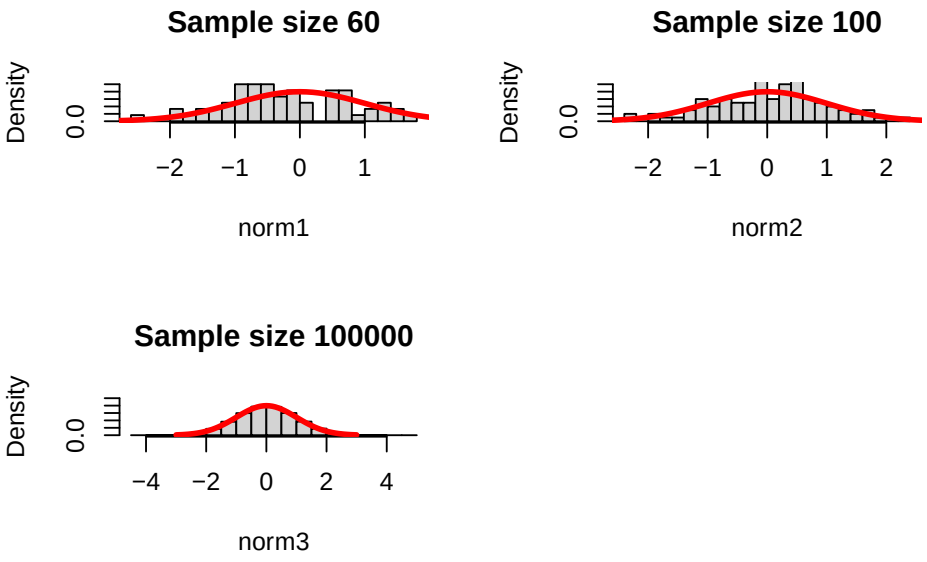
This approach is used in credit risk modelling to simulate portfolio compositions and assess concentration risk.

5.7 Small and large sample properties

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 60",
```

```
  col = "lightgrey",
  ylim = c(0, 0.5)
)
lines(x,dnorm(x), lwd = 3, col = "red")
hist(norm2,
  freq = FALSE,
  breaks = 20,
  main = "Sample size 100",
  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 calculate volatility repeatedly for different return series.

```
test_returns = c(0.01, -0.02, 0.015, -0.008, 0.003)
vol = sd(test_returns, na.rm = TRUE) * sqrt(252)
```

We can create a reusable function for this:

```
calculate_volatility = function(returns, annualize = TRUE){
  vol = sd(returns, na.rm = TRUE)
  if(annualize){
```

```
    vol = vol * sqrt(252)  # Annualize using trading days
  }
  return(vol)
}
```

```
calculate_volatility(test_returns)
```

```
[1] 0.2242186
```

```
calculate_volatility(test_returns, annualize = FALSE)
```

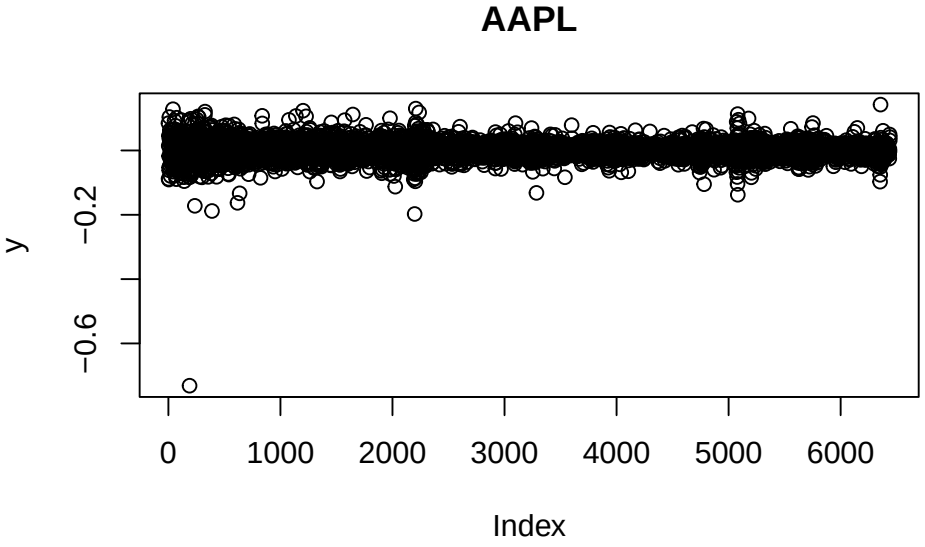
```
[1] 0.01412445
```

And for our returns

```
load('Returns.RData')
ReturnAnalysis = function(name){
  load('Returns.RData')
  if(! name  %in% names(Returns)) {
    cat("Do not 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="AAPL")
```

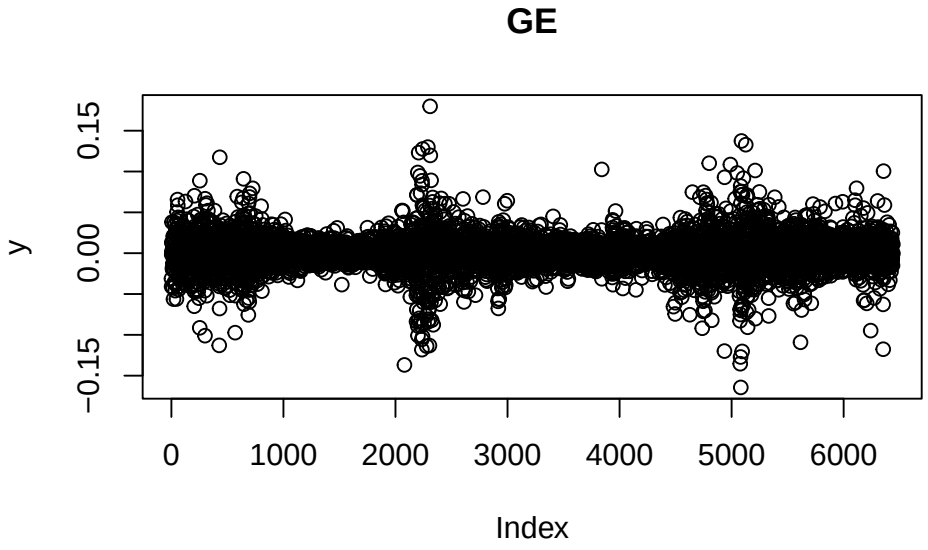
```
AAPL mean 0.0008842013 sd 0.02524584
```




```
[1] 0.0008842013
```

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

```
GE mean 0.0001114734 sd 0.02092539
```




```
[1] 0.0001114734
```

5.9 Recap

5.9.1 In this seminar, we have covered:

- Drawing random numbers from specified distributions
- Using seeds for replicability
- Comparing distributions visually using random samples
- Small and large sample properties of random numbers
- Creating custom functions in R
- Discrete sampling techniques for financial applications
- Function development for financial analysis

5.9.2 Some new functions used:

- `rnorm()` — generate random numbers from normal distribution
- `rt()` — generate random numbers from Student-t distribution
- `set.seed()` — set random seed for reproducibility
- `sample()` — random sampling from a vector with optional probabilities
- `hist()` — create histograms
- `barplot()` — create bar charts
- `table()` — create frequency tables
- `if()` — conditional statements
- `%in%` — check if element is in vector
- `stop()` — stop execution with error message
- `sum()` — calculate sum of values
- `points()` — add points to existing plot
- `sd()` — calculate standard deviation
- `sqrt()` — calculate square root

5.10 Optional exercises

1. **Credit risk analysis**:
   - Modify the credit rating simulation to use different probability distributions
   - Create a horizontal barplot showing rating percentages
   - Calculate the percentage of bonds in each rating category
2. **Advanced sampling techniques**:
   - Implement stratified sampling for credit ratings by industry sector
   - Compare simple random sampling vs stratified sampling results
   - Create visualizations showing the difference between methods
3. **Bootstrap simulation**:
   - Create a function that bootstrap samples from historical returns
   - Use bootstrap to estimate confidence intervals for volatility
   - Compare bootstrap results with theoretical confidence intervals
4. **Distribution comparison exercise**:
   - Generate 10,000 random returns using a normal distribution with mean and sd from actual stock returns
   - Generate 10,000 random returns using Student-t distribution with df=3
   - Create side-by-side QQ plots comparing each to the actual returns
   - Which distribution better captures the tail behaviour of real stock returns?