Financial Risk Forecasting
Seminar Week 9: Simulation-based Risk Measurement

Jon Danielsson
London School of Economics

Version 4.0

9 Simulation-based Risk Measurement

9.1 Why simulation-based risk measurement matters for financial risk

While the analytical VaR methods from Week 8 work well for linear instruments like stocks and bonds, many financial instruments have non-linear payoffs that require simulation-based approaches. Simulation methods become essential when:

- Non-linear instruments: Options and other derivatives have payoffs that change non-linearly with underlying asset prices
- Portfolio complexity: Multi-asset portfolios with derivatives cannot be evaluated using simple analytical formulas
- Tail risk assessment: Simulation reveals the full distribution of potential losses, not just the normal distribution assumptions
- Regulatory requirements: Basel framework requires sophisticated risk measurement for complex portfolios
- Stress testing: Regulators and risk managers need to understand portfolio behaviour under extreme scenarios

Building on Week 5's introduction to simulation methods and Week 8's analytical VaR, we now combine these approaches to handle complex instruments. The Monte Carlo techniques from Week 5 provide the foundation, while Week 8's VaR framework provides the risk measurement structure.

For practitioners, simulation-based VaR is often the only feasible approach when portfolios contain derivatives, structured products or when correlations become important under stress conditions.

For more detail, see Simulation methods for risk in the R notebook.

9.2 The plan for this week

1. Review simulation fundamentals from Week 5
2. Extend Week 8 VaR methods to simulation-based approaches
3. Black-Scholes option pricing using simulation
4. Single-asset VaR using Monte Carlo methods
5. Multi-asset simulation for portfolio risk

9.3 Loading data and libraries

```
library(mvtnorm)
```

We will use the `Prices.RData` and `Returns.RData` files we created in Seminar 2.

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

9.4 Black-Scholes option pricing

While we do not need simulations for Black-Scholes pricing (the analytical formula is exact), we simulate option prices here for three important reasons. First, as an introduction to simulation pricing methods. Second, to evaluate how many simulations are needed for accuracy. Third, because simulation methods become essential for more complex options where analytical formulas do not exist.

9.4.1 Basic Black-Scholes formula implementation

We will be using the Black-Scholes function to evaluate option prices and assess 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))
}
```

9.4.2 Calculating option prices for a single case

First, we need to select a stock and get its price and returns data:

```
# Select a stock for option pricing - using JPM consistently throughout
stock = "JPM"
y = Returns[[stock]]
price = tail(Prices[[stock]], 1)
```

```r
# Set standard parameters for option pricing
K = price*0.7
r = 0.09
sigma = sd(y)  # daily volatility
sigmaAnnual = sqrt(255)*sigma  # annual volatility for Black-Scholes
Maturity = 1

# Calculate option prices using Black-Scholes
bs(K=K, P=price, r=r, sigma=sigmaAnnual, T=Maturity)
```

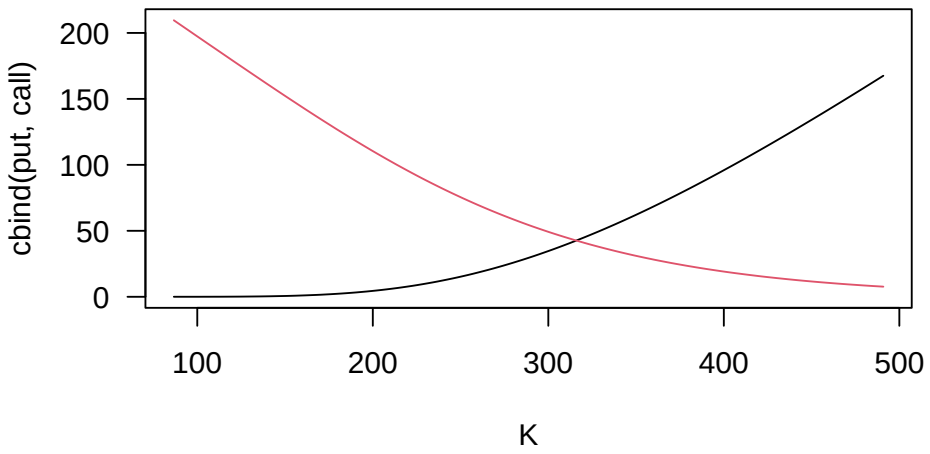
```
$Call
[1] 108.7899

$Put
[1] 4.764651
```

9.4.3 Plotting option prices across strike prices

We can plot the put and call prices over a range of strike prices to see how option values change:

```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=sigmaAnnual, T=Maturity)
    put[i]= res$Put
    call[i]= res$Call
}
matplot(K,cbind(put,call) ,type='l',lty=1,las=1)
```



9.5 Monte Carlo simulation of option prices

9.5.1 Setting up the simulation parameters

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.

```
seed=888
set.seed(seed=seed)
S=1e5
K=price*0.7
r=0.09
sigmaAnnual=sqrt(255)*sd(y)
Maturity=2
```

9.5.2 Method 1: Using normal distribution simulation

Refer to the book and slides on the necessary mathematics for simulating an option. We simulate log returns from a normal distribution and then transform to get stock prices:

```
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)
simPut = pmax(K-Psim,0)
simCall = pmax(Psim-K,0)
simCall_normal = mean(simCall) * exp(-Maturity*r)
simPut_normal = mean(simPut) * exp(-Maturity*r)
```

9.5.3 Method 2: Using lognormal distribution simulation

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.

```
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_lognormal = mean(simCall) * exp(-Maturity*r)
simPut_lognormal = mean(simPut) * exp(-Maturity*r)
```

9.5.4 Comparing simulation results with Black-Scholes

We use the R command `pmax` because we need to apply the `max` to each element in a vector.

```
cat("Comparison of option pricing methods:\n")
```

```
Comparison of option pricing methods:
```

```
cat("Method", "\t\t", "Put", "\t", "Call", "\n")
```

```
Method       Put     Call
```

```
cat("Black-Scholes", "\t", round(true$Put,3), "\t", round(true$Call,3), "\n")
```

```
Black-Scholes    9.122   129.047
```

```
cat("Normal sim", "\t\t", round(simPut_normal,3), "\t", round(simCall_normal,3), "\n")
```

```
Normal sim       9.092   129.226
```

```
cat("Lognormal sim", "\t", round(simPut_lognormal,3), "\t", round(simCall_lognormal,3), "\
```

```
Lognormal sim    9.196   128.953
```

9.6 Simulation-based VaR Framework

Building on Week 8's analytical VaR methods, we now develop simulation-based approaches that can handle non-linear instruments. The general framework follows these steps:

1. Set up simulation parameters and risk calculation setup
2. Simulate future asset prices or returns
3. Calculate portfolio values under each simulation
4. Compute profit/loss for each scenario
5. Extract VaR from the simulated distribution

9.6.1 Risk calculation setup

We establish consistent parameters for all VaR calculations:

```
# Risk calculation parameters (building on option pricing setup above)
seed = 888
p = 0.01
mean = 0  # assume zero drift for risk calculations
# Reuse: y, price, r from Black-Scholes section above
sigma = sd(y)  # daily volatility for simulation
sigmaAnnual = sqrt(255)*sigma  # annual volatility for Black-Scholes
```

9.7 Single Asset VaR Using Simulation

Before applying simulation to derivatives, we verify our approach works for stocks by comparing to Week 8's analytical VaR:

```
set.seed(seed=seed)
S = 1e5
```

```r
# Simulate future stock prices
Psim = price * rnorm(S, mean=1+mean, sd=sigma)
# Calculate profit/loss
Q = Psim - price
# Simulation-based VaR
SimVaR = -sort(Q)[S * p]
# Analytical VaR from Week 8
AnalyticalVaR = -price * qnorm(p, sd=sigma)
cat("Simulation VaR:", SimVaR, "Analytical VaR:", AnalyticalVaR, "\n")
```

```
Simulation VaR: 15.61417 Analytical VaR: 15.65606
```

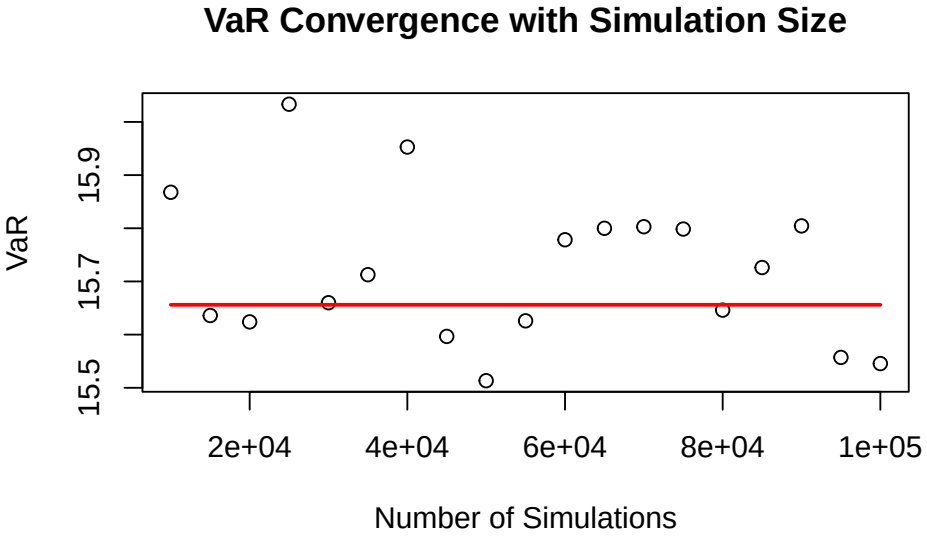
9.7.1 Simulation convergence analysis

A key question in simulation-based VaR is how many simulations are needed for accuracy. The hundred thousand simulations we use provide reasonable accuracy for most applications, but increasing simulation size improves precision at the cost of computation time and memory usage:

```r
# Test different simulation sizes
sim_sizes = seq(10000, 1e5, by=5000)
SimVaR_convergence = vector(length=length(sim_sizes))
TrueVaR = -price * qnorm(p, sd=sd(y))

for(i in 1:length(sim_sizes)){
  Psim = price * rnorm(sim_sizes[i], mean=1+mean, sd=sigma)
  Q = Psim - price
  SimVaR_convergence[i] = -sort(Q)[sim_sizes[i] * p]
}

plot(sim_sizes, SimVaR_convergence,
     main="VaR Convergence with Simulation Size",
     xlab="Number of Simulations", ylab="VaR")
segments(min(sim_sizes), TrueVaR, max(sim_sizes), TrueVaR, col="red", lwd=2)
```




```
# Even with infinite simulations, we would not get identical results to analytical VaR
# due to numerical precision differences between simulation and analytical methods
cat("Simulation accuracy improves with size, but computational trade-offs apply\n")
```

```
Simulation accuracy improves with size, but computational trade-offs apply
```

9.8 VaR for Derivatives

Options require simulation because their payoffs are non-linear. We calculate VaR for individual derivatives and then portfolio combinations.

9.8.1 Single option VaR

For a call option, we simulate underlying prices and calculate the change in option value:

```
K=price*0.7
Maturity=2
call_today = bs(K=K, P=price, r=r, sigma=sigmaAnnual, T=Maturity)$Call
S=1e5
Psim = price * rnorm(S, mean=1+mean, sd=sigma)
call_tomorrow = bs(K=K, P=Psim, r=r, sigma=sigmaAnnual, T=Maturity-1/365)$Call
Q = call_tomorrow - call_today
VaR_call = -sort(Q)[S * p]
cat("Call option VaR:", VaR_call, "\n")
```

```
Call option VaR: 13.97831
```

9.8.2 Portfolio VaR with mixed positions

We can combine stocks and options in a single portfolio VaR calculation:

```
# Define portfolio positions
K.call = price*0.7
Maturity.call = 2
```

```
call_today = bs(K=K.call, P=price, r=r, sigma=sigmaAnnual, T=Maturity.call)$Call

K.put = price*1.1
Maturity.put = 0.7
put_today = bs(K=K.put, P=price, r=r, sigma=sigmaAnnual, T=Maturity.put)$Put

# Calculate portfolio value tomorrow under each simulation
portfolio_today = price + call_today + put_today

portfolio_tomorrow = Psim +
  bs(K=K.call, P=Psim, r=r, sigma=sigmaAnnual, T=Maturity.call-1/365)$Call +
  bs(K=K.put, P=Psim, r=r, sigma=sigmaAnnual, T=Maturity.put-1/365)$Put

Q = portfolio_tomorrow - portfolio_today
VaR_portfolio = -sort(Q)[S * p]
cat("Portfolio VaR:", VaR_portfolio, "\n")
```

```
Portfolio VaR: 21.58579
```

9.9 Multi-Asset Portfolio VaR

If we hold a portfolio of two stocks and options on each, we need to simulate from the bivariate normal distribution. In this case, we call `rmvnorm()`, but it would be straightforward to do it manually with a Cholesky decomposition.

```
# For bivariate simulation, use JPM and another financial stock (C) for sector comparison
Assets=c("JPM","C")
y=Returns[,Assets]
price=tail(Prices[,Assets],1)
Sigma=cov(y)
```

9.9.1 Bivariate simulation setup

For portfolios with multiple underlying assets, we need to simulate correlated returns:

```
print(Sigma)
```

```
              JPM           C
JPM 0.0005431774 0.0005085806
C   0.0005085806 0.0008487875
```

```
set.seed(seed)
r=rmvnorm(10,sigma=Sigma)
Prices1 = as.numeric(price[1])*(1+r[,1])
Prices2 = as.numeric(price[2])*(1+r[,2])
SimPrice=cbind(Prices1,Prices2)
print(r)
```

```
             [,1]          [,2]
 [1,] -0.05688564 -0.0626260594
```

```
 [2,]  0.01219066  0.0002233776
 [3,] -0.03702474 -0.0244057486
 [4,] -0.03870619 -0.0070956606
 [5,]  0.02480580  0.0366376019
 [6,]  0.02098719  0.0401502108
 [7,]  0.02414309  0.0211599440
 [8,] -0.02254434 -0.0363703746
 [9,]  0.02246876  0.0073545978
[10,] -0.04908384 -0.0324776895
```

```
print(SimPrice)
```

```
      Prices1  Prices2
 [1,] 272.3337 86.92269
 [2,] 292.2802 92.75071
 [3,] 278.0687 90.46685
 [4,] 277.5832 92.07202
 [5,] 295.9229 96.12740
 [6,] 294.8203 96.45313
 [7,] 295.7316 94.69216
 [8,] 282.2501 89.35738
 [9,] 295.2481 93.41199
[10,] 274.5866 89.71834
```

```
print(cov(r))
```

```
            [,1]        [,2]
[1,] 0.001146843 0.001001409
[2,] 0.001001409 0.001119089
```

```
print(cor(r))
```

```
          [,1]      [,2]
[1,] 1.0000000 0.8839489
[2,] 0.8839489 1.0000000
```

```
print(price)
```

```
         JPM     C
24516 288.76 92.73
```

```
print(colMeans(SimPrice))
```

```
  Prices1   Prices2
285.88253  92.19727
```

9.9.2 Portfolio VaR calculation

Using the simulated correlated prices, we calculate portfolio VaR and compare to individual asset risks:

```
set.seed(seed)
r=rmvnorm(S,sigma=Sigma)
Prices1 = as.numeric(price[1])*(1+r[,1])
```

```
Prices2 = as.numeric(price[2])*(1+r[,2])
SimPrice=cbind(Prices1,Prices2)

today = sum(price)
tomorrow = rowSums(SimPrice)
Q = tomorrow - today
SimVaR = -sort(Q)[p*S]

w=as.numeric(price/sum(price))
s2= w %*% Sigma %*% w
s2=as.numeric(s2)
value=sum(price)
VaR=-qnorm(p) * value * sqrt(s2 )
cat("Portfolio VaR = $",VaR,", SimVaR $",SimVaR,"\n",sep="")
```

```
Portfolio VaR = $20.78488, SimVaR $20.92558
```

9.9.3 Understanding correlation effects on portfolio risk

The key insight from multivariate simulation is how correlation affects portfolio risk compared to individual asset risks:

```
# Calculate individual asset VaRs
VaR1 = -qnorm(p) * as.numeric(price[1]) * sd(y[,1])
VaR2 = -qnorm(p) * as.numeric(price[2]) * sd(y[,2])
sum_individual_VaR = VaR1 + VaR2

# Compare portfolio vs sum of individual VaRs
correlation = cor(y[,1], y[,2])
cat("Individual VaRs: $", round(VaR1,2), " + $", round(VaR2,2), " = $", round(sum_individu
```

```
Individual VaRs: $15.66 + $6.28 = $21.94
```

```
cat("Portfolio VaR: $", round(VaR,2), "\n", sep="")
```

```
Portfolio VaR: $20.78
```

```
cat("Diversification benefit: $", round(sum_individual_VaR - VaR,2), "\n", sep="")
```

```
Diversification benefit: $1.16
```

```
cat("Correlation between assets:", round(correlation,3), "\n")
```

```
Correlation between assets: 0.749
```

This demonstrates the fundamental principle that portfolio risk is less than the sum of individual risks when correlation is less than 1. The diversification benefit depends on the correlation structure we studied in Week 7's multivariate volatility models.

The correlation between JPM and C of 0.749 shows why portfolio VaR is lower than the sum of individual VaRs. During crisis periods, this correlation typically increases, reducing diversification benefits precisely when they are most needed.

9.10 Recap

9.10.1 In this seminar, we have covered:

- Understanding when simulation-based risk measurement is necessary for non-linear instruments
- Building on Week 5's simulation foundations and Week 8's analytical VaR methods
- Black-Scholes option pricing using simulation methods:
  - Comparing normal vs lognormal simulation approaches
  - Understanding why we simulate when analytical formulas exist
  - Evaluating simulation accuracy and convergence
- Simulation-based VaR framework with structured methodology:
  - Single asset VaR using Monte Carlo methods
  - Comparing simulation vs analytical VaR from Week 8
  - Convergence analysis for determining simulation size
- VaR for derivatives and non-linear instruments:
  - Single option VaR calculations
  - Portfolio VaR combining stocks and derivatives
  - Understanding why simulation is essential for complex payoffs
- Multi-asset portfolio risk using correlated simulations:
  - Bivariate normal simulation with correlation structure
  - Portfolio diversification effects and correlation impact
  - Comparing portfolio VaR to sum of individual asset VaRs
  - Connection to Week 7's multivariate volatility concepts

9.10.2 Some new functions used:

- `rmvnorm()` — generate random multivariate normal variables
- `colMeans()` — calculate column means of a matrix
- `rowSums()` — calculate row sums of a matrix
- `rlnorm()` — generate random lognormal variables
- `pmax()` — element-wise maximum for vectors

9.11 Optional exercises

1. Expected Shortfall for derivatives:
   - Modify the VaR calculations to compute Expected Shortfall instead
   - Compare ES for a stock vs a call option vs a put option
   - Which has the highest ES relative to its value?
   - Create a function `ES_derivative(type, K, Maturity, S)` that returns ES for any derivative
2. Large-scale simulation analysis:
   - Write a function `option_price_sim(S, K, Maturity, n_sim)` that handles arbitrary simulation sizes
   - Test with n_sim = 10^6, 10^7, 10^8 (be careful with memory!)
   - Plot convergence of option price estimates as simulation size increases
   - Calculate the standard error of your estimates
3. Multi-asset portfolio VaR:

- Extend the bivariate simulation to three stocks (add a third stock of your choice)
- Calculate portfolio VaR for equal weights vs market-cap weights
- How does the correlation structure affect portfolio VaR?
- Create a heatmap showing pairwise correlations

4. Time-varying volatility for derivatives:
   - Fit a GARCH model to get daily conditional volatility
   - Use the GARCH volatility forecast in Black-Scholes instead of historical volatility
   - Calculate daily VaR for an option over 30 days using updated volatilities
   - Compare with constant volatility assumption - when do they differ most?
5. Option portfolio builder:
   - Create a function `build_option_portfolio()` that allows specification of multiple options
   - Include: stock positions, calls (different strikes/maturities), puts
   - Calculate portfolio Greeks (delta, gamma) using finite differences
   - Compute portfolio VaR, including all positions
6. Stress testing and scenarios:
   - Implement stressed VaR by adjusting the covariance matrix (multiply correlations by 1.5)
   - Create specific scenarios: market crash (-20%), volatility spike (vol $\times$ 2)
   - Calculate the option portfolio value under each scenario
   - Which positions provide the best hedging in stressed conditions?
7. Backtesting simulation accuracy:
   - Compare 1-day VaR forecasts using simulation vs analytical methods
   - Use a rolling window of 250 days to calculate daily VaR
   - Count VaR violations (when actual loss exceeds VaR)
   - Which method is more accurate? Does it depend on market conditions?