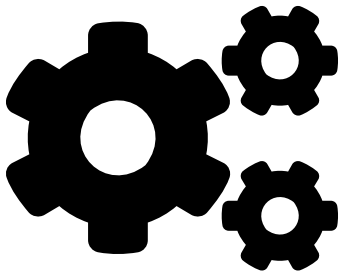
Financial engineering.



Dr. Martín Lozano.

<martin.lozano@udem.edu>

https://sites.google.com/site/mlozanoqf/

https://github.com/mlozanoqf/

Last compiled on: 10/07/2021, 06:31:45.

Abstract

This material relies on John C. Hull [Hull, 2015]. Some mathematical background is skipped to emphasize the data analysis, model logic, discussion, graphical approach and R coding. As in the philosophy of Donald Knuth [Knuth, 1984], the objective of this document is to explain to human beings what we want a computer to do as literate programming. This is a work in progress and it is under revision.

Index.

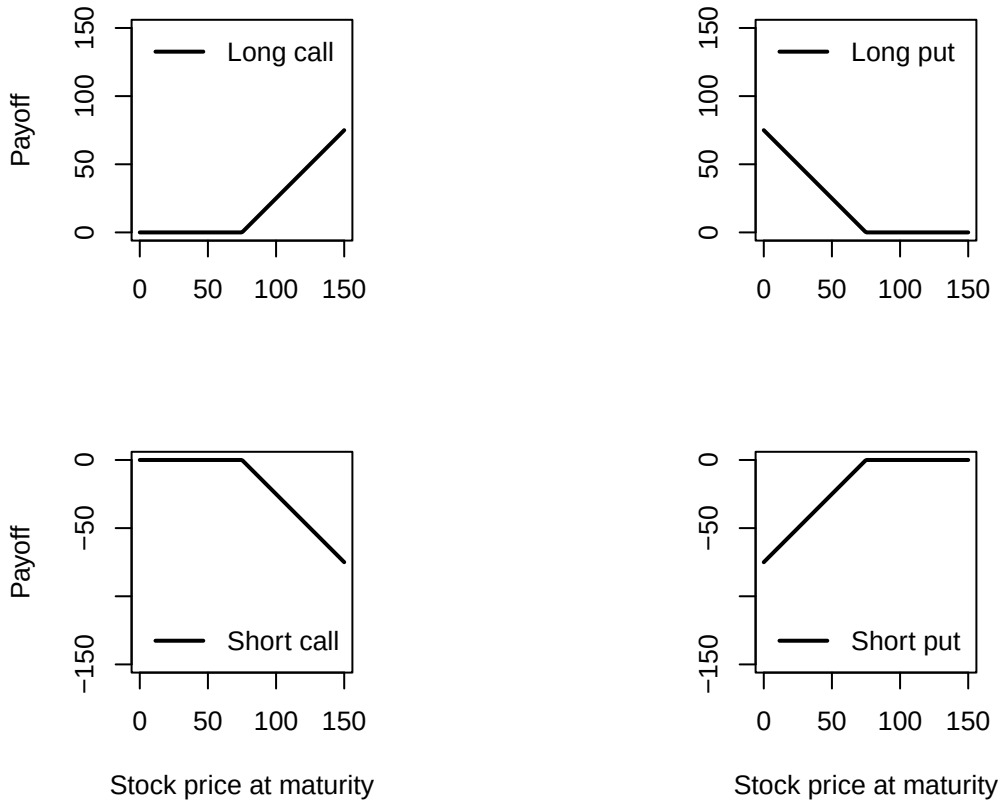
1 Options.

The basic payoff option functions are the following.

```
ST.seq <- seq(from = 0, to = 150, length.out = 150)
K = 75
cl <- pmax(ST.seq - K, 0) # long call payoff.
pl <- pmax(K - ST.seq, 0) # long put payoff.
cs <- pmin(K - ST.seq, 0) # short call payoff.
ps <- pmin(ST.seq - K, 0) # short put payoff.
```

Graphically:

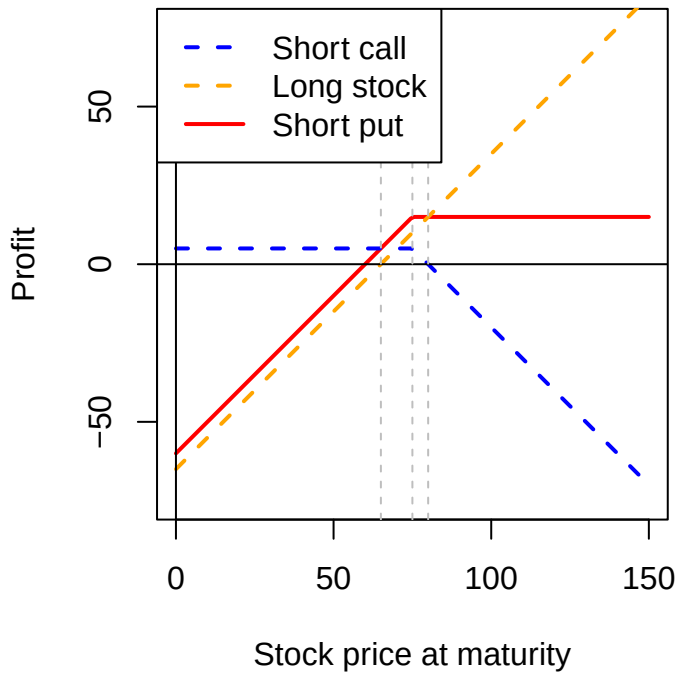
```
par(mfrow = c(2, 2), mai = c(0.7, 0.4, 0.4, 0.4))
par(pty = "s")
plot(ST.seq, cl, type = "l", ylim = c(0, 150), lwd = 2,
     ylab = "Payoff",
     xlab = "")
legend("topleft", legend = c("Long call"),
       lwd = 2, bg = "white", bty = "n")
par(pty = "s")
plot(ST.seq, pl, type = "l", ylim = c(0, 150), lwd = 2,
     ylab = "",
     xlab = "")
legend("topleft", legend = c("Long put"),
       lwd = 2, bg = "white", bty = "n")
par(pty = "s")
plot(ST.seq, cs, type = "l", ylim = c(-150, 0), lwd = 2,
     ylab = "Payoff",
     xlab = "Stock price at maturity")
legend("bottomleft", legend = c("Short call"),
       lwd = 2, bg = "white", bty = "n")
par(pty = "s")
plot(ST.seq, ps, type = "l", ylim = c(-150, 0), lwd = 2,
     ylab = "",
     xlab = "Stock price at maturity")
legend("bottomleft", legend = c("Short put"),
       lwd = 2, bg = "white", bty = "n")
```




```
# Here I assume the stock price is currently 65.
S0 <- ST.seq - K + 10
# I assume the option price is 5.
cs.p <- 5
# A short put profit is replicated by taking a short call and a long stock.
ans <- cs.p + cs + S0
par(pty = "s")
plot(ST.seq, ans, type ="l", ylim = c(-75, 75), lwd = 2, col = "red",
     main = "Figure 12.1a", xlab = "Stock price at maturity",
     ylab = "Profit")
lines(ST.seq, S0, lty = 2, lwd = 2, col = "orange")
lines(ST.seq, cs + cs.p, lty = 2, lwd = 2, col = "blue")
abline(v = 0)
abline(h = 0)
abline(v = 65, lty = 2, col = "grey")
abline(v = 75, lty = 2, col = "grey")
abline(v = 80, lty = 2, col = "grey")
```

```
legend("topleft", legend = c("Short call", "Long stock", "Short put"),
       col = c("blue", "orange", "red"), lty = c(2, 2, 1),
       lwd = 2, bg = "white")
```

Figure 12.1a

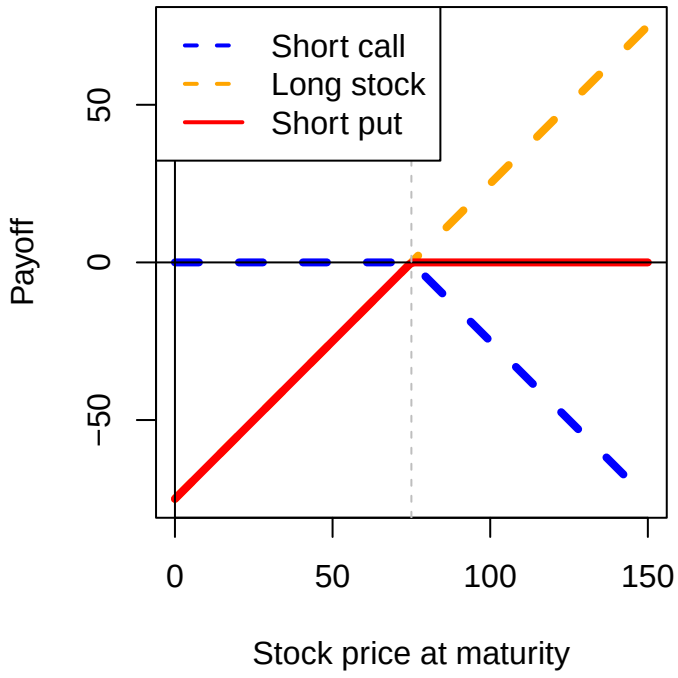


Nice. Now see how it looks if we plot a payoff diagram instead.

```
S0 <- ST.seq - K
ans <- cs + S0
par(pty = "s")
plot(ST.seq, S0, type ="l", ylim = c(-75, 75), lwd = 4, col = "orange",
     main = "Figure 12.1a", xlab = "Stock price at maturity",
     ylab = "Payoff", lty = 2)
lines(ST.seq, ans, lwd = 4, col = "red")
lines(ST.seq, cs, lty = 2, lwd = 4, col = "blue")
abline(v = 0)
abline(h = 0)
abline(v = 75, lty = 2, col = "grey")
legend("topleft", legend = c("Short call", "Long stock", "Short put"),
```

```
       col = c("blue", "orange", "red"), lty = c(2, 2, 1),
       lwd = 2, bg = "white")
```

Figure 12.1a



2 Options properties.

Financial options are one of the most flexible, popular and versatile financial instruments. They are relevant by themselves and important as building blocks to create other financial products and financial strategies. In order to fully understand options we need to understand their properties first. This is, what are the determinant of financial options, how these determinants change the option prices, what are the limits of financial options, what happens when these limits are violated, methods for determining the theoretical option prices and the relationship between call and put options.

2.1 Determinants.

The main determinants of option prices are those parameters of the Black-Scholes formula:

- The price of the underlying stock at time zero: S_0.

- The option strike price: K.
- The risk-free rate: rf.
- The option maturity: T.
- The volatility of the underlying stock σ.

This is, $BS = f(S_0, K, rf, T, \sigma)$. Here we set up the BS function and then evaluate it at $BS = f(S_0 = 50, K = 50, rf = 0.05, T = 1, \sigma = 0.3)$.

```
library(knitr)
library(kableExtra)
# Black-Scholes call.
c.bs <- function(S, K, rf, TT, sigma) {
  d1 <- (log(S / K) + (rf + sigma^2 / 2) * TT) / (sigma * sqrt(TT))
  d2 <- d1 - sigma * sqrt(TT)
  c <- S * pnorm(d1) - K * exp(-rf * TT) * pnorm(d2)
}
# Black-Scholes put.
p.bs <- function(S, K, rf, TT, sigma) {
  d1 <- (log(S / K) + (rf + sigma^2 / 2) * TT) / (sigma * sqrt(TT))
  d2 <- d1 - sigma * sqrt(TT)
  p <- K * exp(-rf * TT) * pnorm(-d2) - S * pnorm(-d1)
}
# Calculate and report results.
c <- c.bs(50, 50, 0.05, 1, 0.3)
p <- p.bs(50, 50, 0.05, 1, 0.3)
option.prices <- data.frame("call" = c, "put" = p)
row.names(option.prices) <- c("Black-Scholes option prices")
kable(option.prices, caption = "Theoretical option prices.") |>
kable_styling(latex_options = "HOLD_position")
```

Table 2.1.1: Theoretical option prices.

	call	put
Black-Scholes option prices	7.115627	4.677099

Now, let's see how BS changes (call and put) when we leave S_0 as a variable and the rest of the parameters as fixed: $BS = f(S_0, K = 50, rf = 0.05, T = 1, \sigma = 0.3)$.

```r
# Parameters.
K <- 50
rf <- 0.05
TT <- 1
sigma <- 0.3
S0 <- 50
# Sequence of S0.
S0.seq <- seq(from = 0, to = 100, length.out = 50)
# Evaluation of the c.bs and p.bs function.
c.S0 <- mapply(c.bs, S0.seq, K, rf, TT, sigma)
p.S0 <- mapply(p.bs, S0.seq, K, rf, TT, sigma)
```

We produce as many `c.S0` as values for `S0.seq`. Then, we can plot.

```r
# See the results.
plot(S0.seq, c.S0, type = "l", lwd = 2, xlab = "Stock price, S0",
     ylab = "Call option price, c")
```

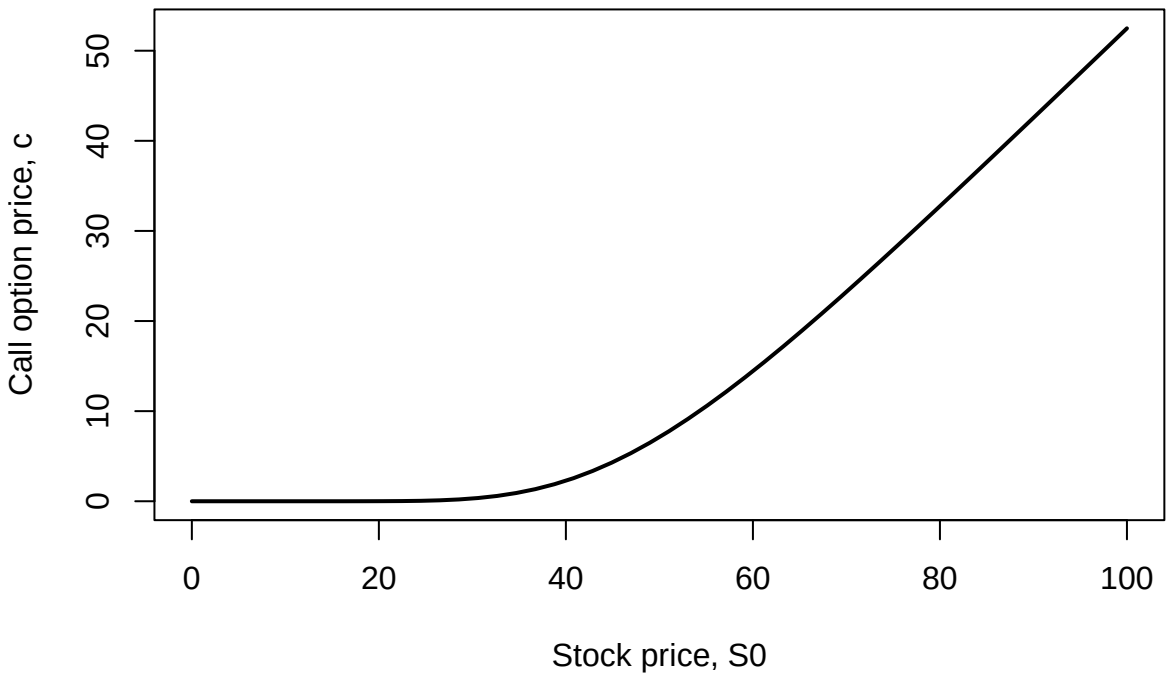


Figure 2.1.1: Call price as a function of the stock price at time zero.

The call option price is an increasing function of S_0. This makes sense because a right to buy the stock at a fix price in the future becomes more valuable as the current value of the stock increases. In other words, it is more expensive to lock the price of an expensive asset for a potential buyer.

See the case of the put option.

```
plot(S0.seq, p.S0, type = "l", lwd = 2, xlab = "Stock price, S0",
     ylab = "Put option price, p")
```

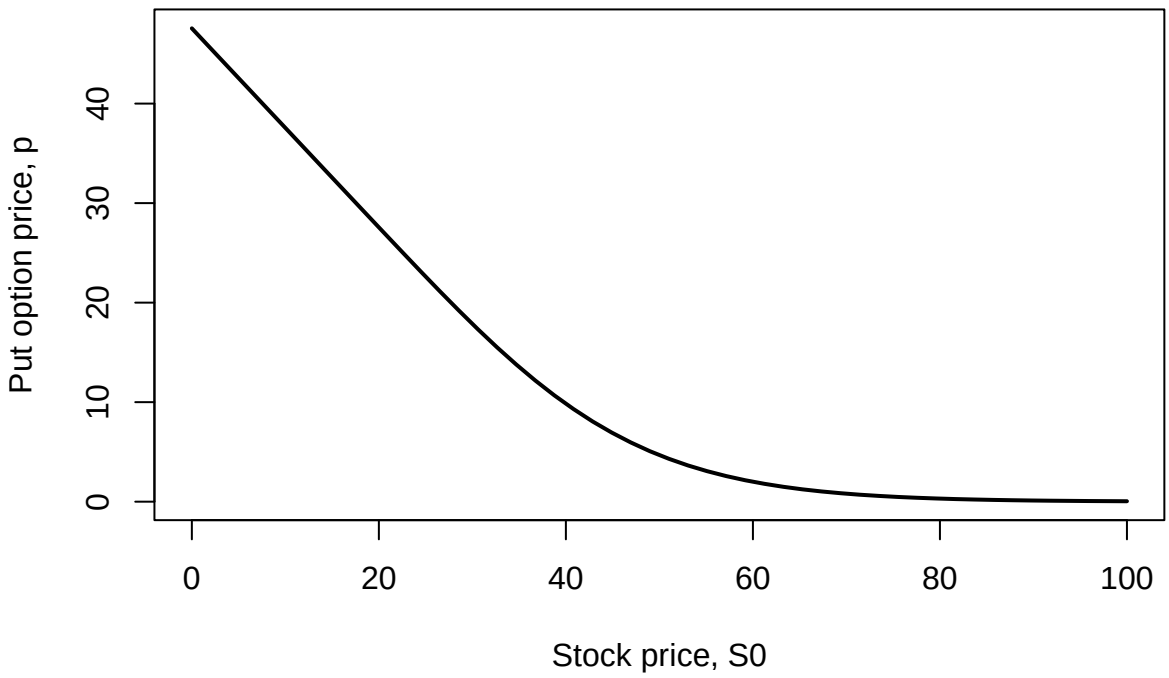


Figure 2.1.2: Put price as a function of the stock price at time zero.

Here, it is cheaper to lock the price of an expensive asset for a potential seller. This makes sense because it looks unnecessary to lock the selling price of an expensive stock.

Let's put both plots together.

```
# What is the interpretation of S0.star, and how did I found this value?
S0.star <- K * exp(-rf * TT)
c.S0.star <- mapply(c.bs, S0.star, K, rf, TT, sigma)
# Plot.
plot(S0.seq, c.S0, type = "l", lwd = 3, col = "blue", ylim = c(4, 8),
     xlab = "Stock price, S0", ylab = "Option theoretical price",
     xlim = c(40, 55))
lines(S0.seq, p.S0, lwd = 3, col = "red")
abline(v = S0.star, lty = 2)
abline(h = c.S0.star, lty = 2)
```

```
abline(h = c, lty = 2, col = "blue")
abline(h = p, lty = 2, col = "red")
abline(v = S0, lty = 2)
points(S0, c, pch = 19, col = "blue", cex = 2)
points(S0, p, pch = 19, col = "red", cex = 2)
points(S0.star, c.S0.star, pch = 19, cex = 2)
legend("topleft", legend = c("Call", "Put"),
       col = c("blue", "red"), lwd = 3, bg = "white")
```

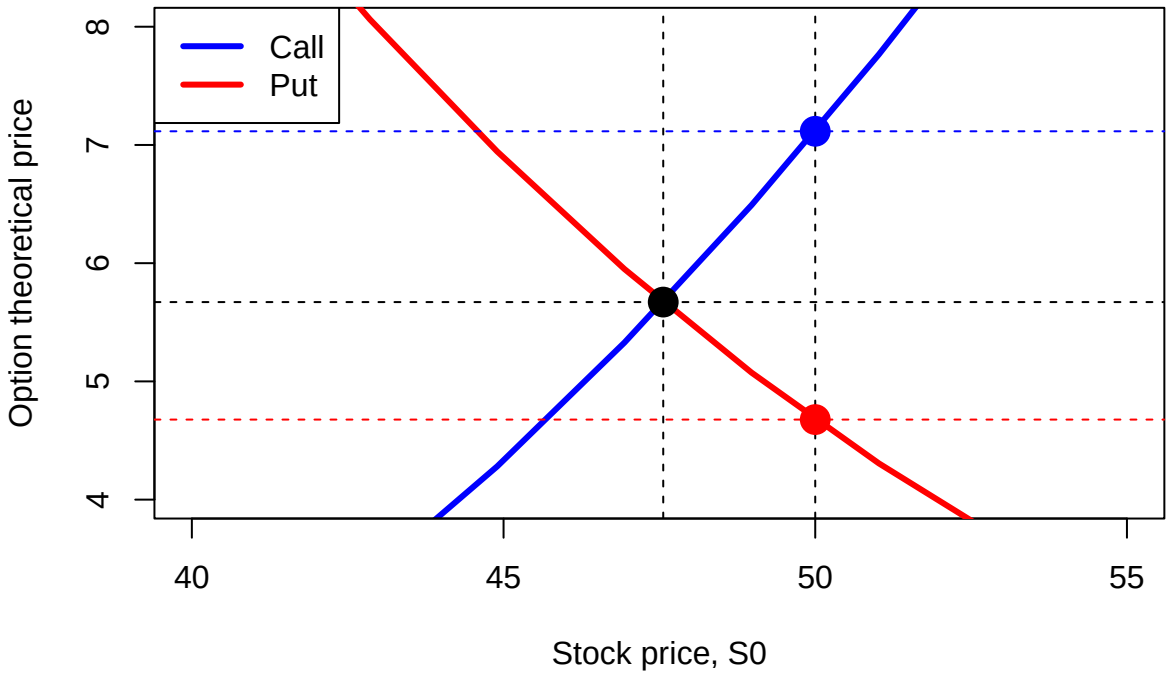


Figure 2.1.3: Call and put prices as a function of the stock price at time zero (zoom).

Apparently, there is a single case in which both options (call and put) are the same. It is interesting to find out the value of S_0 that makes $c = p$. Clearly, this value is lower than $S_0 = 50$, the point here is to find the mathematical expression of S_0 that makes $c = p$.

2.2 Option bounds.

Option values (or prices) are subject or constrained to upper and lower bounds. This basically mean that in theory option prices cannot exceed upper bounds and cannot decrease below lower bounds. If they do, then arbitrage opportunities arise.

We can extend our analysis by incorporating the corresponding call and put option bounds.

```
# Put lower bound.
lb.put <- pmax(K * exp(-rf * TT) - S0.seq, 0)
# Call lower bound.
lb.call <- pmax(S0.seq - K * exp(-rf * TT), 0)
# Again the S0.star. Where did this value comes from?
line.star <- S0.star * 2
# Plot.
par(pty = "s")
plot(S0.seq, c.S0, type = "l", lwd = 3, col = "blue", ylim = c(0, 100),
     xlab = "Stock price, S0", ylab = "Call (blue), put (red)")
lines(S0.seq, p.S0, lwd = 3, col = "red")
abline(v = S0.star, lty = 2)
abline(v = 0, lty = 2)
abline(v = line.star, lty = 2, col = "green")
abline(h = line.star, lty = 2, col = "green")
# Put upper bound.
abline(h = K * exp(-rf * TT), lwd = 2, col = "red", lty = 2)
lines(S0.seq, lb.put, lwd = 2, col = "red", lty = 2)
# Call upper bound.
lines(S0.seq, S0.seq, lwd = 2, col = "blue", lty = 2)
lines(S0.seq, lb.call, lwd = 2, col = "blue", lty = 2)
```

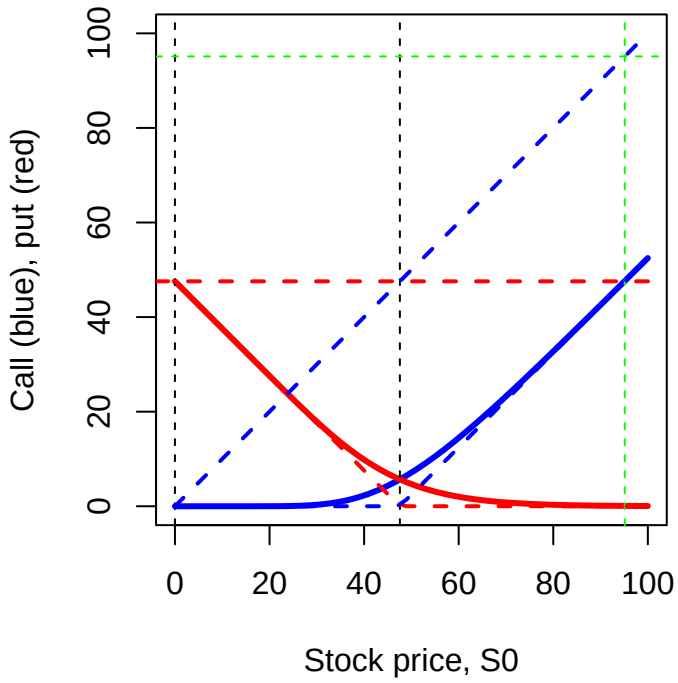


Figure 2.2.1: Option values as a function of the stock price at time zero including option bounds.

Note that the stock price range of values is from zero to positive infinity. However, this is not the case for the options. The option values depend on these bounds. We can illustrate the same plot in a cleaner way, to make emphasis on a geometric approach.

```
par(pty = "s")
plot(S0.seq, lb.put, type = "l", lwd = 2, col = "red",  ylim = c(0, 100),
     xlab = "Stock price, S0", ylab = "Call (blue), put (red)")
abline(v = S0.star, lty = 2)
abline(v = 0, lty = 2)
abline(v = line.star, lty = 2)
abline(h = line.star, lty = 2)
abline(h = S0.star, lwd = 2, col = "red")
lines(S0.seq, S0.seq, lwd = 2, col = "blue")
lines(S0.seq, lb.call, lwd = 2, col = "blue")
```

```
# Enumerate the areas.
text(S0.star / 2, (S0.star + (S0.star / 2)), "1")
text(60, 80, "2")
text(80, 60, "3")
text(10, 20, "4")
text(25, 35, "5")
text(25, 10, "6")
text(40, 20, "7")
text(60, 35, "8")
text(80, 20, "9")
```

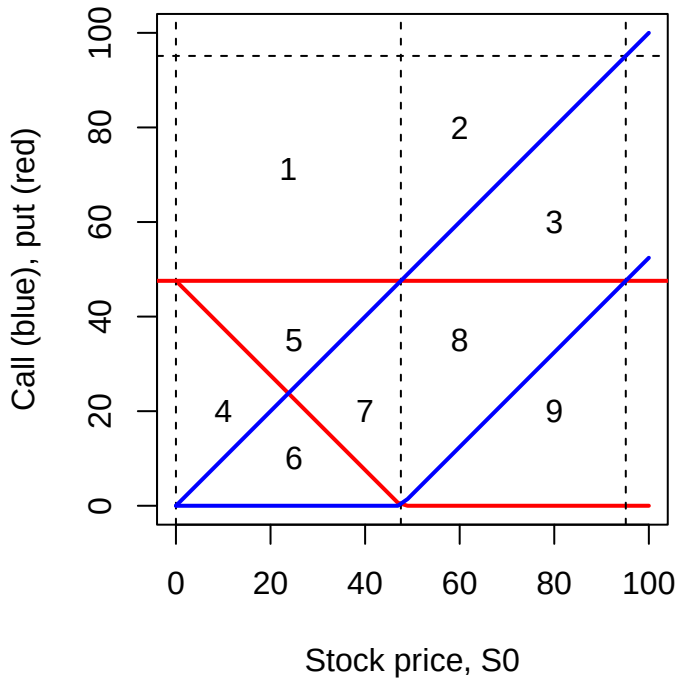


Figure 2.2.2: Call and put option bounds: geometric inspection (or 'the puzzle').

Every number from 1 to 9 is assigned to a specific geometric shape in the plot above. This is, a square as in 1, and triangles of two different shapes in the rest of the cases (2 to 9). It is interesting to analyze whether each area represents a possible price range for call and/or put

options.

Now, let's see how Black-Scholes option prices changes (call and put) when we leave S_0 as a variable, three levels of σ and the rest of the parameters as fixed: $BS = f(S_0, K = 50, rf = 0.05, T = 1, \sigma)$. We can show this in a 2 dimension plot, including the correspondent bounds.

```
# Evaluate the function.
c.S0.s1 <- mapply(c.bs, S0.seq, K, rf, TT, sigma)
c.S0.s5 <- mapply(c.bs, S0.seq, K, rf, TT, sigma * 5)
c.S0.s15 <- mapply(c.bs, S0.seq, K, rf, TT, sigma * 15)
# Plot.
plot(S0.seq, c.S0.s1, type= "l", lwd = 3,  xlab = "Stock price, S0",
     ylab = "Call option price, c")
lines(S0.seq, c.S0.s5, lwd = 3, col = "blue") # call with higher sigma
lines(S0.seq, c.S0.s15, lwd = 3, col = "red") # call with even higher sigma
lines(S0.seq, lb.call, lwd = 3, lty = 2) # lower bound
lines(S0.seq, S0.seq, lwd = 3, lty = 2) # upper bound
legend("topleft", legend = c("Call sigma=0.3", "Call sigma*5",
                             "Call sigma*15", "Call bounds"),
       col = c("black", "blue", "red", "black"), lwd = 2,
       lty = c(1, 1, 1, 2), bg = "white", cex = 0.8)
```

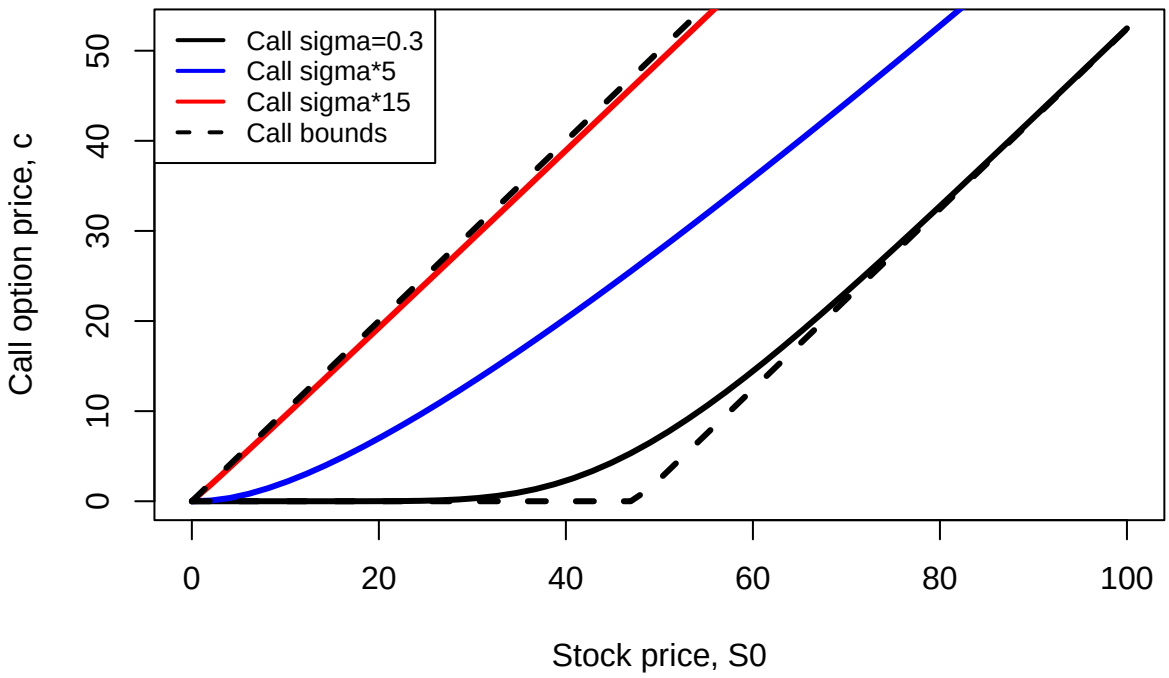


Figure 2.2.3: Call option changes as the stock price at time zero and sigma increases: the role of bounds.

And the same for the put option.

```
# Evaluate the function.
p.S0.s1 <- mapply(p.bs, S0.seq, K, rf, TT, sigma)
p.S0.s5 <- mapply(p.bs, S0.seq, K, rf, TT, sigma * 5)
p.S0.s15 <- mapply(p.bs, S0.seq, K, rf, TT, sigma * 15)
# Plot.
plot(S0.seq, p.S0.s1, type= "l", lwd = 3,  xlab = "Stock price, S0",
     ylab = "Pall option price, p")
lines(S0.seq, p.S0.s5, lwd = 3, col = "blue") # put with higher sigma
lines(S0.seq, p.S0.s15, lwd = 3, col = "red") # put with even higher sigma
lines(S0.seq, lb.put, lwd = 3, lty = 2) # lower bound
lines(S0.seq, rep(S0.star, 50) , lwd = 3, lty = 2) # upper bound
legend("bottomleft", legend = c("Put sigma=0.3", "Put sigma*5",
```

```
                              "Put sigma*15", "Put bounds"),
       col = c("black", "blue", "red", "black"), lwd = 2,
       lty = c(1, 1, 1, 2), bg = "white", cex = 0.8)
```

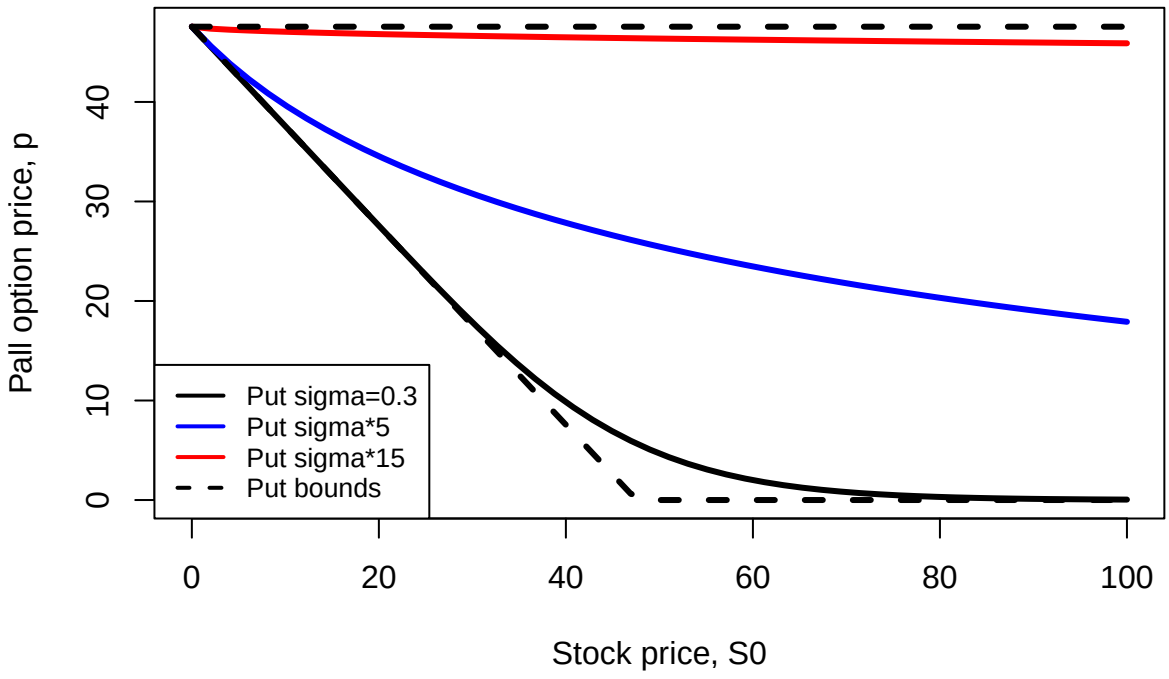


Figure 2.2.4: Put option changes as the stock price at time zero and sigma increases: the role of bounds.

Then, regardless of the extreme value of parameters, option bounds represent the maximum and minimum values for the call and put options. These option prices are theoretical as they are calculated by implementing the Black-Scholes formula. We normally evaluate whether the option *market* prices violates these bounds because this will flag an opportunity to implement an arbitrage strategy to generate a risk-free profit.

A different way to illustrate the option properties is by looking at a plane. Here, we let S_0 and K as a free variables and remain the rest as fixed: $BS = f(S_0, K, rf = 0.05, T = 1, \sigma = 0.3)$.

```
# K as a variable.
K.seq <- S0.seq
# Create the empty matrix.
c.S0.s <- matrix(0, nrow = 50, ncol = 50)
# Fill the empty matrix.
for(i in 1:50){ # Is there an easier way to do this?
  for(j in 1:50){
    c.S0.s[i, j] <- c.bs(S0.seq[i], K.seq[j], rf, TT, sigma) } }
# Plot.
c.S0.s.plot <- persp(S0.seq, K.seq, c.S0.s, zlab = "Call", xlab = "S0",
      ylab = "K", theta = -60, phi = 10, expand =  0.5, col = "orange",
      shade = 0.2, ticktype = "detailed")
points(trans3d(S0, K, c, c.S0.s.plot), cex = 2, pch = 19, col = "blue")
```

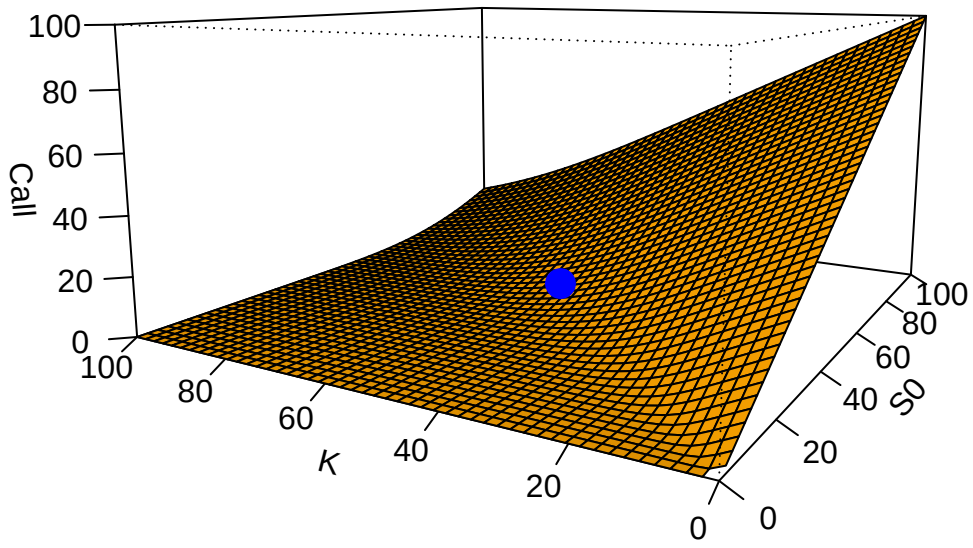


Figure 2.2.5: Call value as the stock price at time zero and the strike price change: a plane view.

An alternative view is by showing a contour plot. A contour plot is a graphical technique for representing a 3-dimensional surface by plotting constant z slices (option prices), called contours, on a 2-dimensional format. That is, given a value for z, lines are drawn for connecting the (x, y) coordinates (S_0, K) where that z value occurs.

```
contour(S0.seq, K.seq, c.S0.s, xlab = "S0", ylab = "K", lwd = 2,
        nlevels = 20)
points(S0, K, pch = 19, col = "blue", cex = 2)
```

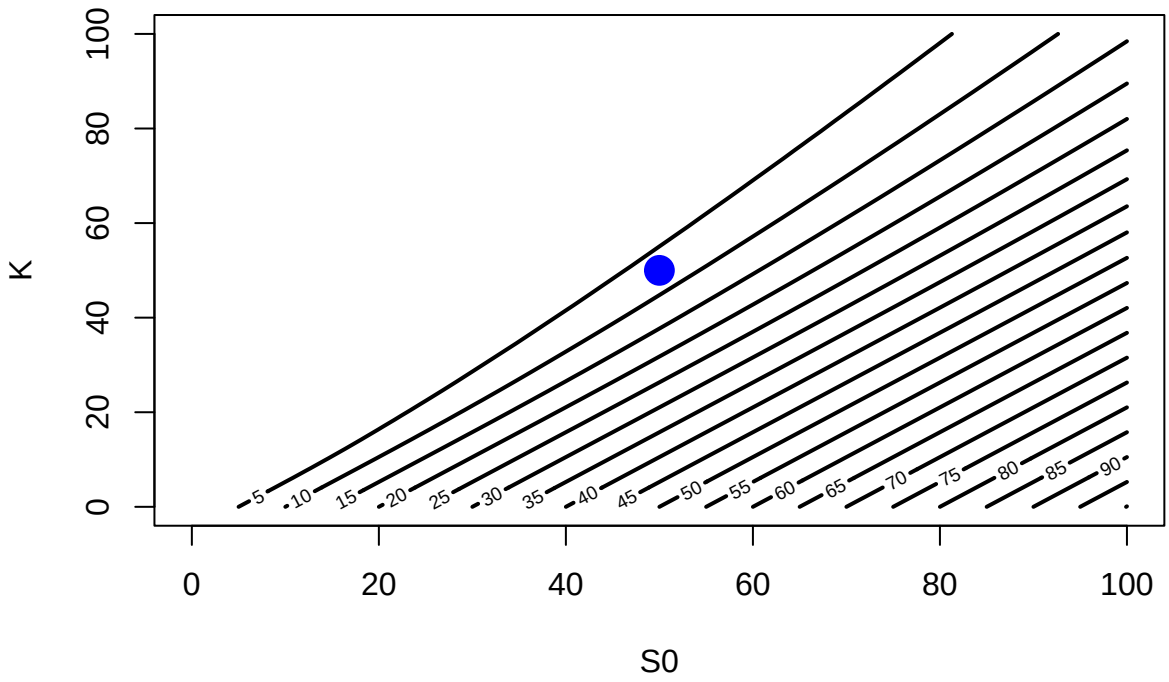


Figure 2.2.6: Call value as the stock price at time zero and the strike price change: a contour view.

Note that the blue circle $(S_0 = 50, K = 50)$ is between the contour line 5 and 10, this makes sense because the call option at $call(S_0 = 50, K = 50, rf = 0.05, T = 1, \sigma = 0.3) = 7.115627$.

An interesting case is the value of the put as a function of the time to maturity. Let's see the simplest case.

```
TT.seq <- seq(from = 1, to = 25, length.out = 50)
p.TT <- mapply(p.bs, S0, K, rf, TT.seq, sigma)
plot(TT.seq, p.TT, type = "l", lwd = 2, ylab = "Put value", xlab = "T")
points(TT, p, col = "red", cex = 2, pch = 19)
```

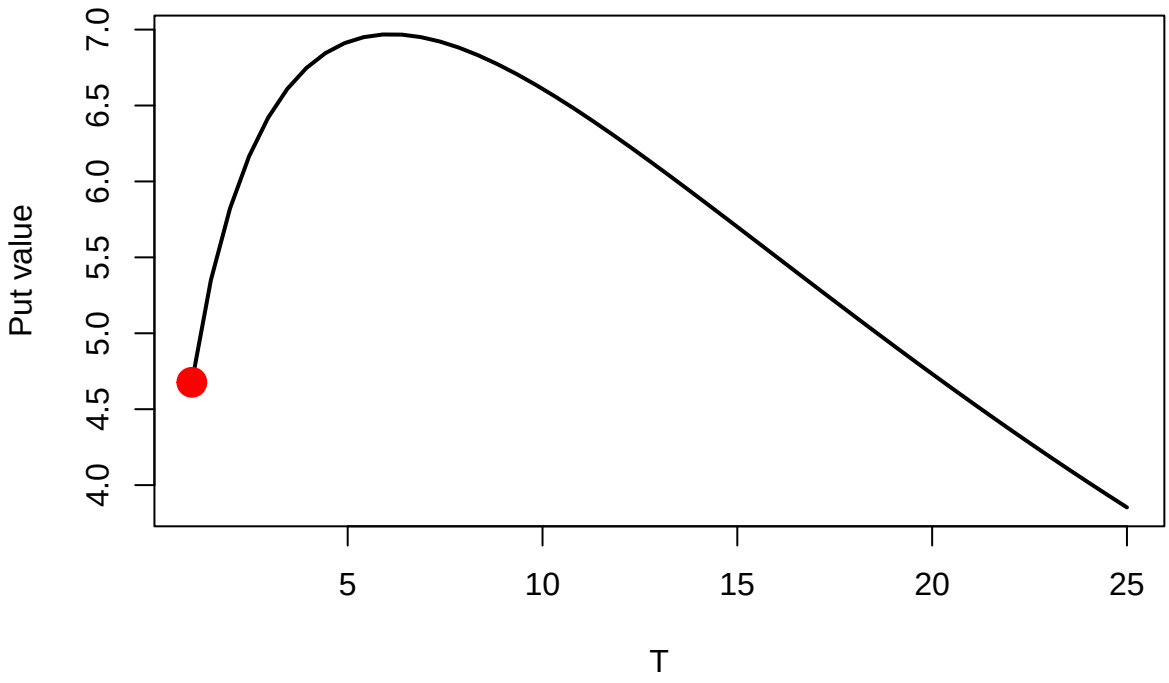


Figure 2.2.7: Put value as time to maturity increases: the mysterious hump.

The put option price first increases as the time to maturity increases and after reaching a maximum, the put option price decreases. What is the mathematical expression for the time to maturity which makes the put option price maximum? What is the reason or logic behind this mysterious hump? Those are interesting questions to address.

Let's illustrate the case when time to maturity and the stock price at time zero changes.

```
p.S0.T <- matrix(0, nrow = 50, ncol = 50)
for(i in 1:50) { # Is there an easier way to do this?
  for(j in 1:50) {
```

```
    p.S0.T[i, j] <- p.bs(S0.seq[i], K, rf, TT.seq[j], sigma) } }
p.S0.T.plot <- persp(S0.seq, TT.seq, p.S0.T, zlab = "Put", xlab = "S0",
                ylab = "T", theta = 330, phi = 10, expand =  0.5,
                col = "orange", shade = 0.2, ticktype = "detailed")
points(trans3d(S0, TT, p, p.S0.T.plot), cex = 2, pch = 19, col = "blue")
```

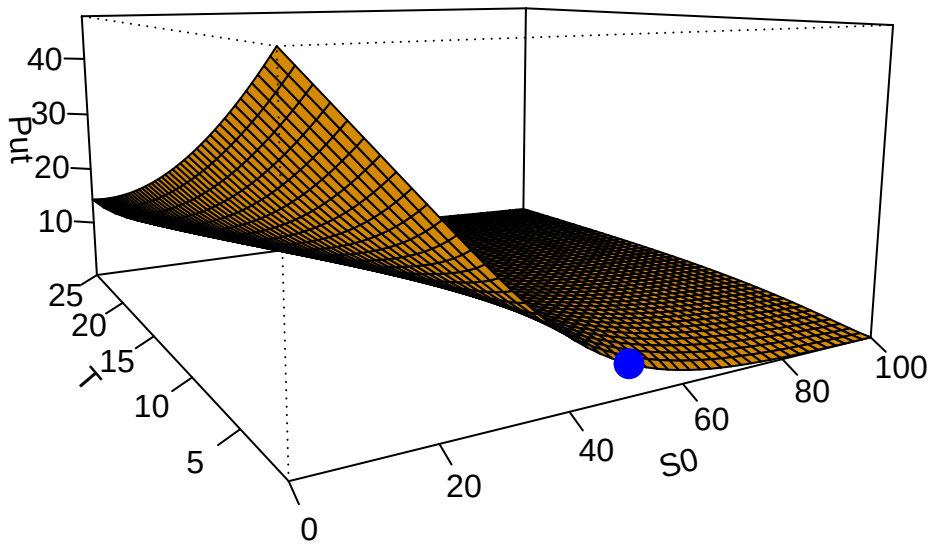


Figure 2.2.8: Put value as the stock price at time zero and time to maturity change: a plane.

That is not quite clear. So, here is the contour view.

```
contour(S0.seq, TT.seq, p.S0.T, xlab = "S0", ylab = "T", lwd = 2,
        nlevels = 40, drawlabels = TRUE)
points(S0, TT, pch = 19, col = "blue", cex = 2)
```

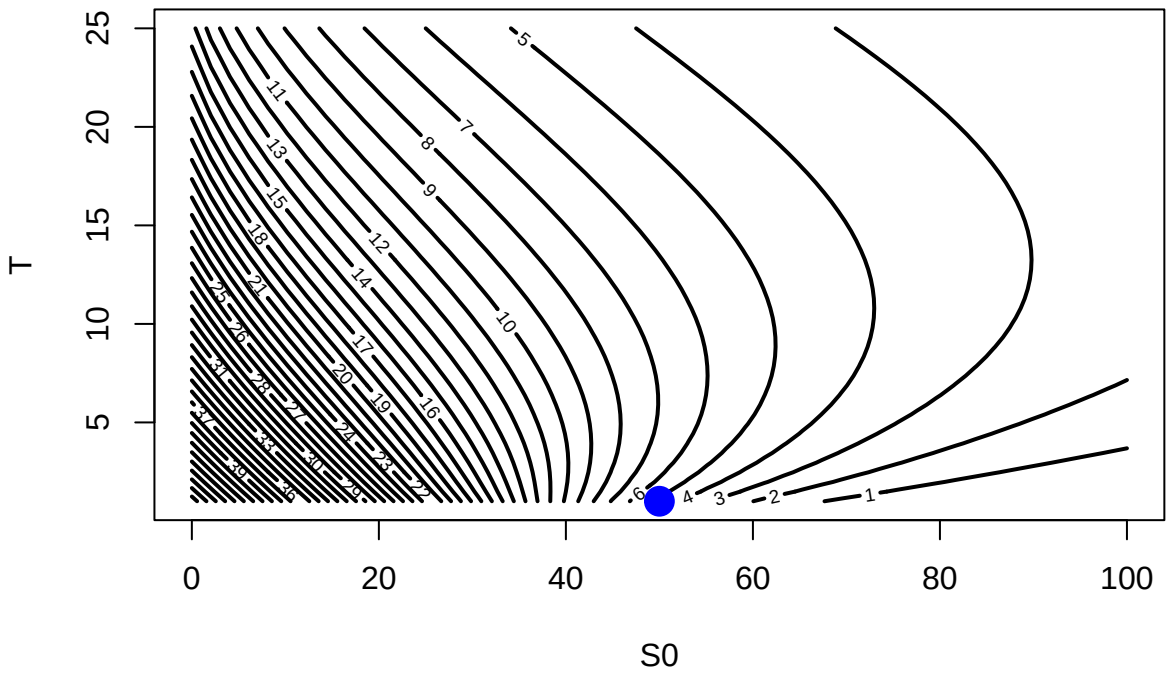


Figure 2.2.9: Put value as the stock price at time zero and time to maturity change: a contour view, or the hypnotic plot.

Now the mysterious hump is clearer than before. Remember the value of $put(S_0 = 50, K = 50, rf = 0.05, T = 1, \sigma = 0.3) = 4.677099$.

2.3 Put-call parity.

The main idea behind the put-call parity is to understand how call and put option prices are related as today, at $t = 0$. In Hull, the procedure to derive the put-call parity starts with the definition of two portfolios: (1) a call and a bond; (2) a put and a stock. Then, derive the corresponding payoff of each portfolio at maturity. Doing this is relatively easy because we do not need a valuation method as we know the payoff functions for options, bonds and stocks. Given that we can demonstrate that the value of two different portfolios are the same at T, then we can conclude that these two portfolios are worth the same at $t = 0$ as well. The put-call parity is then: $c + Ke^{-rT} = p + S_0$.

We can verify that this equation holds.

$p = c + Ke^{-rT} - S_0 \rightarrow p = 7.115627 + 50e^{-0.05\times1} - 50 \rightarrow p = 4.677098.$

$c = p + S_0 - Ke^{-rT} \rightarrow c = 4.677098 + 50 - 50e^{-0.05\times1} \rightarrow c = 7.115627.$

Let's take one step back and demonstrate that both portfolios are worth the same at maturity. We first define the assets payoffs.

```
ST.seq <- seq(from = 0, to = 150, length.out = 50)
cT <- pmax(ST.seq - K, 0) # call payoff.
pT <- pmax(K - ST.seq, 0) # put payoff.
BT <- rep(K, 50) # bond payoff (at maturity).
pc <- pmax(ST.seq, K) # this will be important.
```

Then we plot.

```
par(pty = "s")
par(mfrow = c(1, 2), oma = c(0, 0, 2, 0))
par(pty = "s")
# Portfolio A.
plot(ST.seq, cT, type = "l", ylab = "Payoff", xlab = "ST", lwd = 2,
     lty = 2, col = "blue", ylim = c(0, 150))
lines(ST.seq, BT, lwd = 2, lty = 2, col = "red")
lines(ST.seq, (cT + BT), lwd = 2)
legend("topleft", legend = c("Call option", "Zero coupon bond",
                             "Total (Portfolio A)"),
       col = c("blue", "red", "black"), lwd = c(2, 2, 2), lty = c(2, 2, 1),
       bg = "white", cex = 0.7)
par(pty = "s")
# Portfolio C.
plot(ST.seq, ST.seq, type = "l", ylab = "Payoff", xlab = "ST",
     lwd = 2, lty = 2, col = "orange", ylim = c(0, 150))
lines(ST.seq, pT, lwd = 2, lty = 2, col = "purple")
lines(ST.seq, (pT + ST.seq), lwd = 2)
legend("topleft", legend = c("Put option", "Share", "Total (Portfolio C)"),
       col = c("purple", "orange", "black"), lwd = 2,
       lty = c(2, 2, 1), bg = "white", cex = 0.7)
```

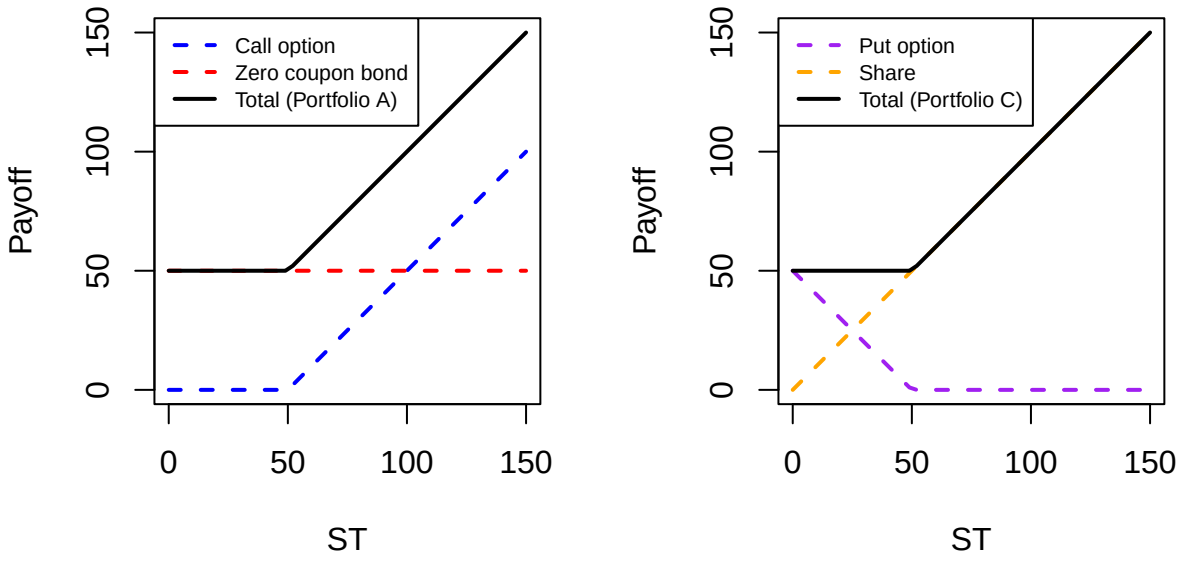


Figure 2.3.1: Portfolio A and C payoffs: the put-call parity.

The point here is that the black line (total) is the same in both cases. This is why we argue that we payoffs of both portfolios are worth the same. If this is so, then they have to value the same in time $t = 0$.

In sum, this is the payoff of portfolios A and C: $max(S_T, K)$.

```
par(pty = "s")
plot(ST.seq, pc, type = "l", ylab = "Payoff", xlab = "ST", lwd = 3,
     ylim = c(0, 150))
```

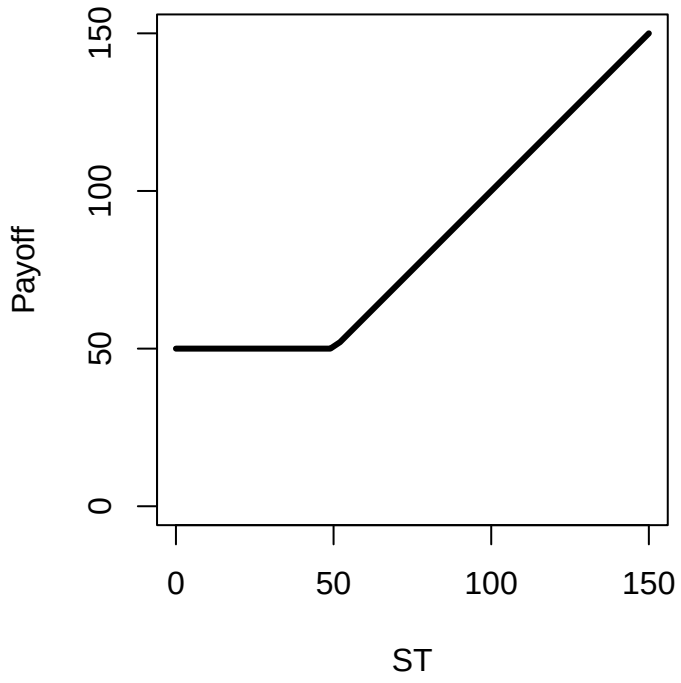


Figure 2.3.2: Portfolio A and C payoffs: max(ST,K).

Nice. We can manipulate the put-call parity to create a synthetic stock. This is:

$c + Ke^{-rT} = p + S_0 \rightarrow S_0 = c + Ke^{-rT} - p.$

Graphically:

```
par(pty = "s")
plot(ST.seq, (cT + BT - pT), type = "l", ylab = "Payoff", xlab = "ST",
     lwd = 3, ylim = c(-50, 150), xlim = c(0, 200), col = "orange")
lines(ST.seq, cT, lwd = 2, lty = 2, col = "blue") # positive call.
lines(ST.seq, BT, lwd = 2, lty = 2, col = "red") # positive bond.
lines(ST.seq, -pT, lwd = 2, lty = 2, col = "purple") # negative put.
legend("bottomright", legend = c("Call option", "Zero coupon bond",
                                 "Put option", "Total (stock)"),
       col = c("blue", "red", "purple", "orange"), lwd = 2,
       lty = c(2, 2, 2, 1), bg = "white", cex = 0.65)
```

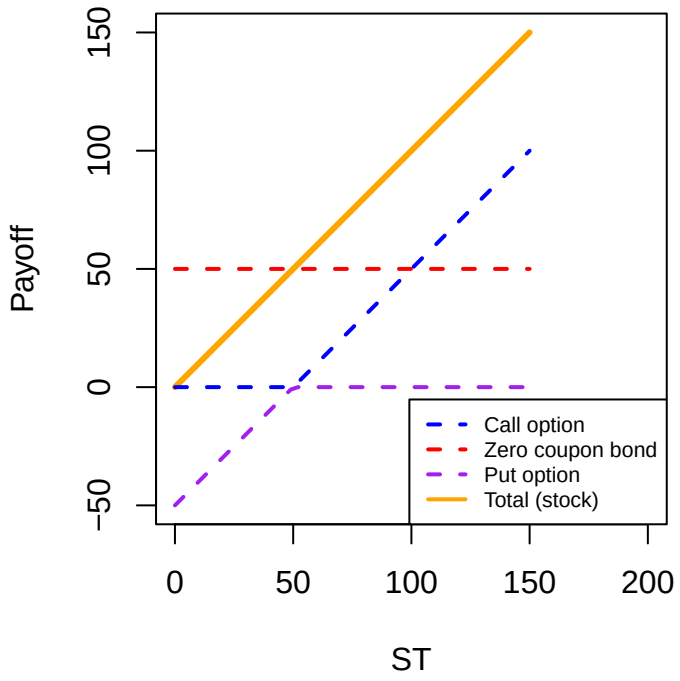


Figure 2.3.3: A synthetic stock.

So, we created a stock that did not exist with a call, a bond and a put.

We can manipulate the put-call parity to create a synthetic bond. This is:

$c + Ke^{-rT} = p + S_0 \rightarrow Ke^{-rT} = p + S_0 - c.$

Graphically:

```
par(pty = "s")
plot(ST.seq, (-cT + ST.seq + pT), type = "l", ylab = "Payoff", xlab = "ST",
     lwd = 3, ylim = c(-50, 150), xlim = c(0, 200), col = "orange",
     main = "A synthetic bond.")
lines(ST.seq, -cT, lwd = 2, lty = 2, col = "blue")
lines(ST.seq, ST.seq, lwd = 2, lty = 2, col = "red")
lines(ST.seq, pT, lwd = 2, lty = 2, col = "purple")
```

```
legend("bottomright", legend = c("Call option (short)", "Stock",
                                 "Put option", "Total (bond)"),
       col = c("blue", "red", "purple", "orange"), lwd = 2,
       lty = c(2, 2, 2, 1), bg = "white")
```

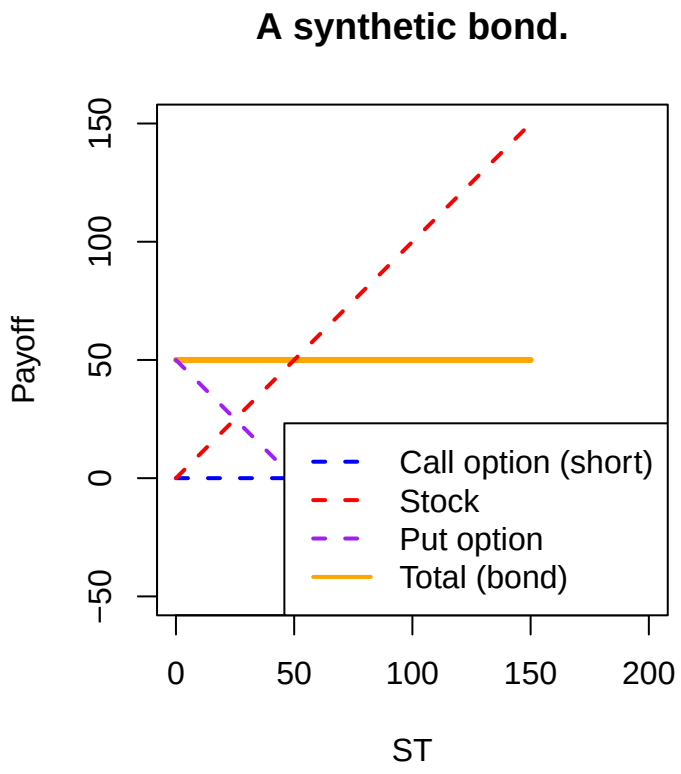


Figure 2.3.4: A synthetic bond.

There is an alternative to the Black-Scholes formula we indirectly review before. Next section introduces the binomial trees.

3 Binomial trees.

Binomial trees are a flexible valuation method for options because we can use them to value not only European but also American options.

3.1 Implementation.

The function is the following.

```
bin <- function(S0, K, sigma, TM, r, steps) {
# the parameters
dt <- TM / steps
u <- exp(sigma * sqrt(dt))
d <- exp(-sigma * sqrt(dt))
a <- exp(r * dt)
p <- (a - d) / (u - d)
S <- matrix(0, steps + 1, steps)
pam <- S
peu <- S
cam <- S
ceu <- S
# the stock price process
  for(i in 1:steps) {
    j <- i + 1
    do <- seq(0, i)
    up <- rev(do)
    S[(1:j), i] <- S0 * (u ^ up) * (d ^ do) }
# the option prices at maturity.
peu[(1:(steps + 1)), steps] <- pmax(K - S[, i], 0)
pam[(1:(steps + 1)), steps] <- pmax(K - S[, i], 0)
ceu[(1:(steps + 1)), steps] <- pmax(S[, i] - K, 0)
cam[(1:(steps + 1)), steps] <- pmax(S[, i] - K, 0)
# the binomial method to price stock options.
for(j in steps:1) { # this is a reverse loop from steps to 1.
  cd <- (seq(steps:1)) # every round we compute less option prices.
  for(i in 1:cd[j]) { # option prices per step.
    peu[i, (j - 1)] <- exp(-r * dt) * (p * peu[i, j] +
                                       (1 - p) * peu[(i + 1), j])
    ceu[i, (j - 1)] <- exp(-r * dt) * (p * ceu[i, j] +
                                       (1 - p) * ceu[(i + 1), j])
    pam[i, (j - 1)] <- max((K - S[i, (j - 1)]), exp(-r * dt) *
                           (p * pam[i, j] + (1 - p) * pam[(i + 1), j]))
```

```
    cam[i, (j - 1)] <- max((S[i, (j - 1)] - K), exp(-r * dt) *
                           (p * cam[i, j] + (1 - p) * cam[(i + 1), j])) } }
# This is the final step in the binomial tree.
p.eu <- exp(-r * dt) * (p * peu[1, 1] + (1 - p) * peu[2, 1])
c.eu <- exp(-r * dt) * (p * ceu[1, 1] + (1 - p) * ceu[2, 1])
p.am <- exp(-r * dt) * (p * pam[1, 1] + (1 - p) * pam[2, 1])
c.am <- exp(-r * dt) * (p * cam[1, 1] + (1 - p) * cam[2, 1])
# Results.
option <- data.frame(c.eu, p.eu, c.am, p.am)
option
}
```

We can evaluate the function to see how the price change depending on the number of steps in the binomial tree.

```
b1t <- bin(S0, K, sigma, TT, rf, 1)
b4t <- bin(S0, K, sigma, TT, rf, 4)
b20t <- bin(S0, K, sigma, TT, rf, 20)
b50t <- bin(S0, K, sigma, TT, rf, 50)
b200t <- bin(S0, K, sigma, TT, rf, 200)
b500t <- bin(S0, K, sigma, TT, rf, 500)
```

See the results.

```
Black.Scholes <- data.frame(c.eu = c, p.eu = p, c.am = NA, p.am = NA)
bin.bs <- rbind("Binomial (1 step)" = b1t, "Binomial (4 steps)" = b4t,
                "Binomial (20 steps)" = b20t, "Binomial (50 steps)" = b50t,
                "Binomial (200 steps)" = b200t,
                "Binomial (500 steps)" = b500t,
                "Black-Scholes" = Black.Scholes)
kable(bin.bs, caption = "Binomial and Black-Scholes comparison.") |>
kable_styling(latex_options = "HOLD_position")
```

Table 3.1.1: Binomial and Black-Scholes comparison.

	c.eu	p.eu	c.am	p.am
Binomial (1 step)	8.481986	6.043457	8.481986	6.043457
Binomial (4 steps)	6.762001	4.323472	6.762001	4.767526
Binomial (20 steps)	7.042462	4.603934	7.042462	4.898985
Binomial (50 steps)	7.086241	4.647713	7.086241	4.921038
Binomial (200 steps)	7.108267	4.669738	7.108267	4.931581
Binomial (500 steps)	7.112682	4.674153	7.112682	4.933664
Black-Scholes	7.115627	4.677099	NA	NA

In the extreme, the binomial method converges to the Black-Scholes method. Let's explore these differences.

```
b1 <- mapply(bin, S0.seq, K, sigma, TT, rf, 1)
b2 <- mapply(bin, S0.seq, K, sigma, TT, rf, 2)
b3 <- mapply(bin, S0.seq, K, sigma, TT, rf, 3)
b4 <- mapply(bin, S0.seq, K, sigma, TT, rf, 4)
b20 <- mapply(bin, S0.seq, K, sigma, TT, rf, 20)
```

Here we compare the value of the binomial method and the Black-Scholes as a function of S_0.

```
plot(S0.seq, b1[1, ], type = "l", col = "red", lwd = 2, xlab = "S0",
     ylab = "Call")
lines(S0.seq, c.S0.s1, col = "black", lwd = 2)
legend("topleft", legend = c("1 step binomial", "Black-Scholes"),
       col = c("red", "black"), lwd = 2, bg = "white")
```

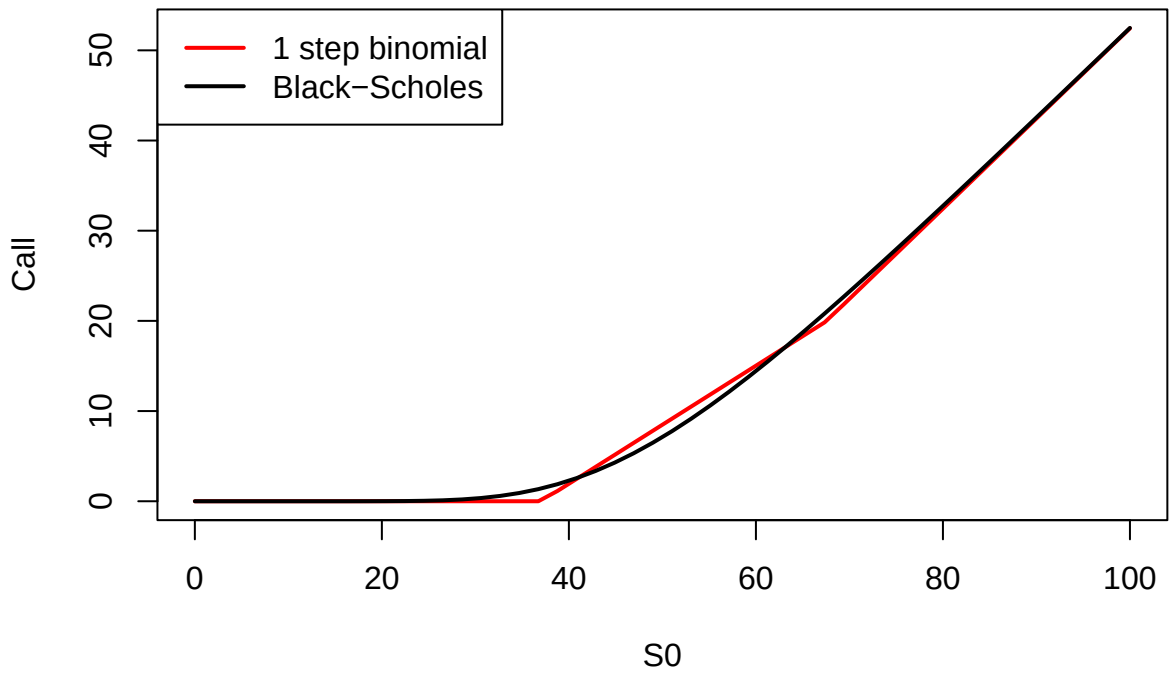


Figure 3.1.1: Binomial and Black-Scholes comparison.

There are some differences between the binomial and the Black-Scholes. Let's zoom to see the differences clearer.

```
plot(S0.seq, b1[1,], type="l", ylim = c(0, 4), xlim = c(30, 45),
     col = "green", lwd = 2, xlab = "S0", ylab = "Call")
lines(S0.seq, b2[1,], col = "purple", lwd = 2)
lines(S0.seq, b3[1,], col = "orange", lwd = 2)
lines(S0.seq, b4[1,], col = "red", lwd = 2)
lines(S0.seq, b20[1,], col = "black", lwd = 2)
lines(S0.seq, c.S0.s1, col = "black", lwd = 2)
legend("topleft", legend = c("1 step binomial", "2 steps binomial",
                             "3 steps binomial", "4 steps binomial",
                             "10 steps binomial", "Black-Scholes"),
       col = c("green", "purple", "orange", "red", "black", "black"),
```

```
       lwd = 2, bg = "white")
```

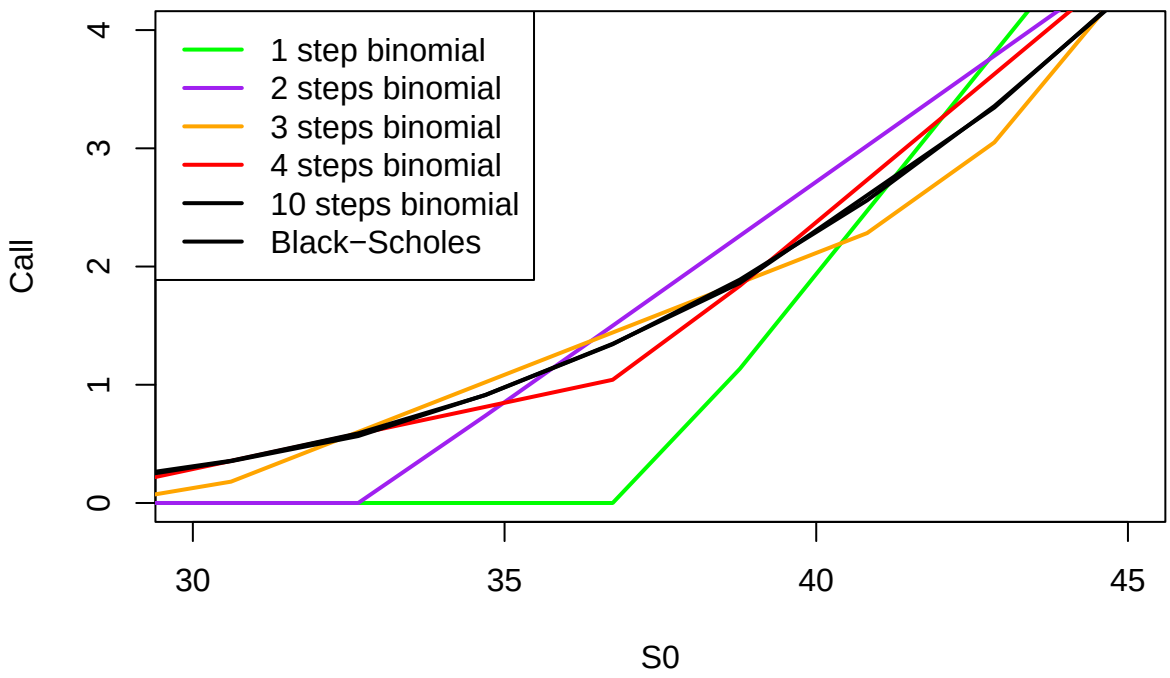


Figure 3.1.2: Binomial and Black-Scholes convergence: a zoom view.

As stated above, the binomial method converges to the Black-Scholes. We can also create a function to visualize the price path of the stock given the assumptions of the binomial method.

3.2 Stock price paths.

In order to value option prices, we first need to understand the evolution of the underlying (in this case the stock price) first. The binomial method assumes that the price can increase or decrease with a certain probability in each time step. Here, we can show how the binomial tree method assumes this stock price evolution.

```r
# Function to generate stock prices paths given the binomial method.
S.paths <- function(S0, sigma, TM, steps) {
  dt <- TM / steps
  u <- exp(sigma * dt^0.5) # Here we set u and d as a function of sigma.
  d <- exp(-sigma * dt^0.5)
  S <- matrix(0, (steps + 1), (steps + 1))
  S[1, 1] <- S0
  for(i in 2:(steps + 1)) {
    do = seq(0, i - 1)
    up = rev(do) # rev provides a reversed version of its argument.
    S[(1:i), i] = S0 * (u ^ up) * (d ^ do) }
  S }
# Evaluate the function.
Spaths <- S.paths(50, 0.3, 1, 10)
# A table.
colnames(Spaths) <- c(0, cumsum(rep(1/10, 10)))
kable(Spaths, caption = "Stock price paths given to the binomial model.",
      digits = 2) |>
kable_styling(latex_options = "HOLD_position")
```

Table 3.2.1: Stock price paths given to the binomial model.

0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
50	54.98	60.45	66.46	73.08	80.35	88.34	97.13	106.80	117.43	129.12
0	45.47	50.00	54.98	60.45	66.46	73.08	80.35	88.34	97.13	106.80
0	0.00	41.36	45.47	50.00	54.98	60.45	66.46	73.08	80.35	88.34
0	0.00	0.00	37.62	41.36	45.47	50.00	54.98	60.45	66.46	73.08
0	0.00	0.00	0.00	34.21	37.62	41.36	45.47	50.00	54.98	60.45
0	0.00	0.00	0.00	0.00	31.11	34.21	37.62	41.36	45.47	50.00
0	0.00	0.00	0.00	0.00	0.00	28.30	31.11	34.21	37.62	41.36
0	0.00	0.00	0.00	0.00	0.00	0.00	25.74	28.30	31.11	34.21
0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	23.41	25.74	28.30
0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	21.29	23.41
0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	19.36

This does not look like a typical binomial tree. In fact, it is not very clear whether a given

price corresponds to an increase or decrease. We can make a few arrangements to visualize this as a tree.

```
S.paths <- function(S0, sigma, TM, steps) {
  dt <- TM / steps
  u <- exp(sigma * dt^0.5)
  d <- exp(-sigma * dt^0.5)
  S <- matrix(0, 2 * ((steps + 1)), (steps + 1))
  S2 <- matrix(NA, ((2 * steps) + 2), (steps + 1))
  S2[(steps + 1), 1] <- S0
  for(i in 2:(steps + 1)) {
    do = seq(0, i - 1)
    up = rev(do) # rev provides a reversed version of its argument.
    S[(1:i), i] = S0 * (u ^ up) * (d ^ do)
    x = rep(NA, i) # These are the NA between stock prices.
    r = rev(c(seq(0, (steps - 1)), 0)) # These creates the blank spaces.
    # Here we combine NA and stock prices for each column.
    S2[(1 + r[i]):((2 * i) + r[i]), i] = as.numeric(rbind(S[(1:i), i], x))
    }
  S2 }
# Evaluate the function.
Spaths <- S.paths(50, 0.3, 1, 10)
# A table.
colnames(Spaths) <- round(c(0, cumsum(rep(1/10, 10))), 2)
kable(Spaths, caption = "Stock price paths given to the binomial model.",
      digits = 2) |>
kable_styling(latex_options = "HOLD_position")
```

Table 3.2.2: Stock price paths given to the binomial model.

0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	129.12
NA	NA	NA	NA	NA	NA	NA	NA	NA	117.43	NA
NA	NA	NA	NA	NA	NA	NA	NA	106.80	NA	106.80
NA	NA	NA	NA	NA	NA	NA	97.13	NA	97.13	NA
NA	NA	NA	NA	NA	NA	88.34	NA	88.34	NA	88.34
NA	NA	NA	NA	NA	80.35	NA	80.35	NA	80.35	NA
NA	NA	NA	NA	73.08	NA	73.08	NA	73.08	NA	73.08
NA	NA	NA	66.46	NA	66.46	NA	66.46	NA	66.46	NA
NA	NA	60.45	NA	60.45	NA	60.45	NA	60.45	NA	60.45
NA	54.98	NA	54.98	NA	54.98	NA	54.98	NA	54.98	NA
50	NA	50.00	NA	50.00	NA	50.00	NA	50.00	NA	50.00
NA	45.47	NA	45.47	NA	45.47	NA	45.47	NA	45.47	NA
NA	NA	41.36	NA	41.36	NA	41.36	NA	41.36	NA	41.36
NA	NA	NA	37.62	NA	37.62	NA	37.62	NA	37.62	NA
NA	NA	NA	NA	34.21	NA	34.21	NA	34.21	NA	34.21
NA	NA	NA	NA	NA	31.11	NA	31.11	NA	31.11	NA
NA	NA	NA	NA	NA	NA	28.30	NA	28.30	NA	28.30
NA	NA	NA	NA	NA	NA	NA	25.74	NA	25.74	NA
NA	NA	NA	NA	NA	NA	NA	NA	23.41	NA	23.41
NA	NA	NA	NA	NA	NA	NA	NA	NA	21.29	NA
NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	19.36
NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

Nice.

```
# Evaluate the function.
# bin <- function(S0, K, sigma, TM, r, steps) {

b2 <- mapply(bin, 50, 52, 0.3, 2, 0.05, 2)
b2
```

```
##      [,1]
## c.eu 9.194163
```

```
## p.eu 6.245708
## c.am 9.194163
## p.am 7.428402
```

```
# Risk neutral probability, note that r is the risk-free rate.
# This is the substitution of equation 13.3.
p.12 <- (exp(0.12*(3/12))-0.9)/(1.1-0.9)
p.12
```

```
## [1] 0.6522727
```

```
# Option expected value ($) at time T
option_T <- p.12*1 + (1-p.12)*0
option_T
```

```
## [1] 0.6522727
```

```
# Option value ($) at time 0.
# This option value is valid for the real world, not only the risk-neutral world.
option_0 <- option_T * exp(-0.12*(3/12))
option_0
```

```
## [1] 0.6329951
```

```
# Real world probability.
# Suppose that, in the real world, the expected return on the stock is 16%
p.real <- (exp(0.16*(3/12))-0.9)/(1.1-0.9)
p.real
```

```
## [1] 0.7040539
```

```
# The expected payoff ($) from the option in the real world is then:
option_T_real <- p.real*1 + (1-p.real)*0
option_T_real
```

```
## [1] 0.7040539
```

Can we calculate the present value of \$0.7040539? We only need the option discount factor. The problem is that we do not know this rate. It is not easy to know the correct discount rate of the option to apply to the expected payoff in the real world. Using risk-neutral valuation solves this problem because we know that in a risk-neutral world the expected return on all assets (and therefore the discount rate to use for all expected payoffs) is the risk-free rate.

What can we do to find out the option discount factor? Since we know the correct value of the option is 0.6329951, we can deduce that the correct real-world discount rate is 42.55688%. This is because $0.6329951 = 0.704053e^{-0.4255688 \times 3/12}$.

To solve for the real-world option discount rate we do the following: $r = -log(0.6329951/0.704053) \times (12/3)$.

```
# Real world option discount factor:
option.df <- -log(option_0/option_T_real)*(12/3)
option.df
```

```
## [1] 0.4255688
```

The correct real-world discount rate for the option is 42.55688%.

```
# Put everything in a function.
option.df.fun <- function(r) {
  p.real <- (exp(r*(3/12))-0.9)/(1.1-0.9)
  option_T_real <- p.real*1 + (1-p.real)*0
  option.df <- -log(option_0/option_T_real)*(12/3)
option.df
}
```

```
# Evaluate for known values:
option.df.fun(0.12)
```

```
## [1] 0.12
```

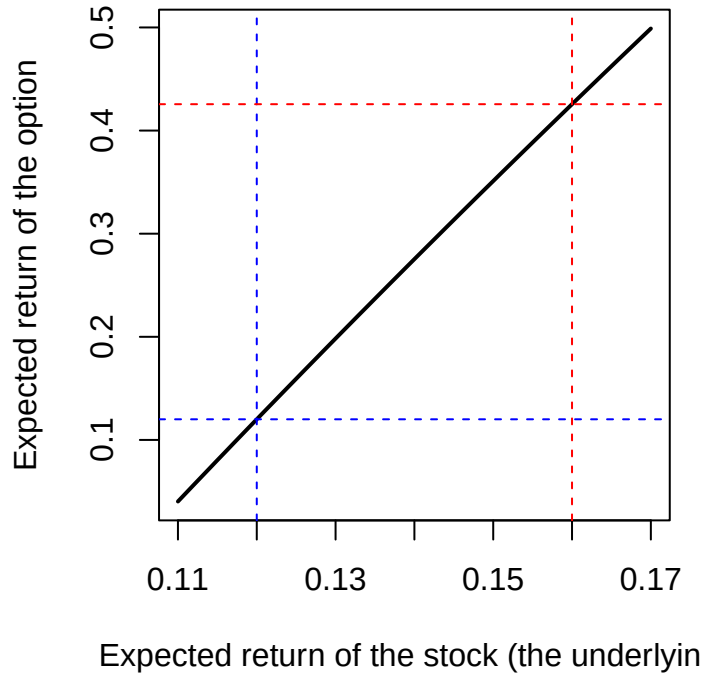
```
option.df.fun(0.16)
```

```
## [1] 0.4255688
```

Everything looks correct.

```
# Evaluate the function in a range of values.
r.seq <- seq(0.11, 0.17, 0.001)
df <- mapply(option.df.fun, r.seq)
par(pty = "s")
plot(r.seq, df, type = "l", lwd = 2,
     main = "Options are in general much risker compared with the underlying stock.",
     xlab = "Expected return of the stock (the underlying)",
     ylab = "Expected return of the option")
```

```
abline(v = 0.12, lty = 2, col = "blue")
abline(h = option.df.fun(0.12), lty = 2, col = "blue")
abline(v = 0.16, lty = 2, col = "red")
abline(h = option.df.fun(0.16), lty = 2, col = "red")
```



3.3 Parrondo's Paradox: Combine two losing investments into a winner.

Consider asset A which has 6% gross return. In the context of a binomial tree: $p \times (1+u) + (1-p) \times (1-d) = 1.06$. Let's assume $p = 0.5$, so we have: $0.5(1+u) + 0.5(1-d) = 1.06$. Asset A has 40% volatility, so $0.4 = \sqrt{0.5(1+u)^2 + 0.5(1-d)^2 - 1.06^2}$. Solving for $(1+u)$ and $(1-d)$ leads to $(1+u) = 1.46$ and $(1-d) = 0.66$.

In a period of 5 years, we would have a random path of ups (1.46) and downs (0.66).

```
u = 0.46
d = 0.34
set.seed(2, sample.kind="Rounding")
```

```r
a <- sample(c(1+u, 1-d), 5, replace = TRUE)
a
```

```
## [1] 1.46 0.66 0.66 1.46 0.66
```

The evolution of $1 dollar invested in this 5 year period would look like this:

```r
cumprod(a)
```

```
## [1] 1.4600000 0.9636000 0.6359760 0.9285250 0.6128265
```

Even simpler, a $1 dollar invested at $t = 0$ would lead to $0.6128265 at $T = 5$.

```r
prod(a)
```

```
## [1] 0.6128265
```

Let's use a function now.

```r
binomial_tree <- function(u, d, n) {
  prod(sample(c(1+u, 1-d), n, replace = TRUE))
}
```

See if it works.

```r
set.seed(2, sample.kind="Rounding")
binomial_tree(u = 0.46, d = 0.34, n = 5)
```

```
## [1] 0.6128265
```

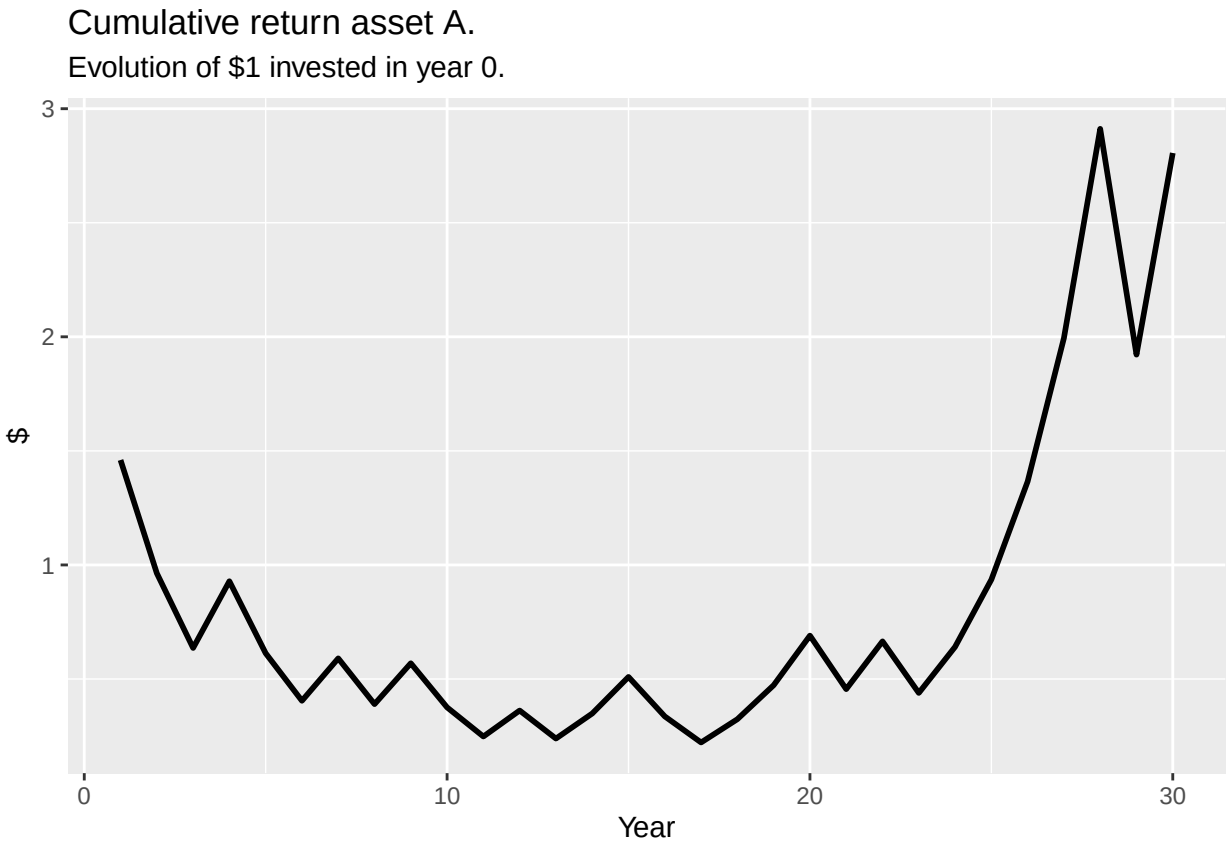
```r
set.seed(2, sample.kind="Rounding")
binomial_tree(u = 0.46, d = 0.34, n = 30)
```

```
## [1] 2.805884
```

It works, a $1 dollar invested at $t = 0$ would lead to $0.6128265 at $T = 5$. And now we know that a $1 dollar invested at $t = 0$ would lead at $T = 30$. Let's view the 30 year case:

```r
library(ggplot2)
library(tidyr)
library(dplyr)
u = 0.46
d = 0.34
n = 30
```

```r
set.seed(2, sample.kind="Rounding")
a.30 <- cumprod(sample(c(1+u, 1-d), n, replace = TRUE))
a.30 <- as.data.frame(cbind(year = c(1:30), ret = a.30))
  ggplot(a.30, aes(x = year, y = ret)) +
  geom_line(size = 1) +
      labs(y = "$",
       x = "Year",
       title = "Cumulative return asset A.",
       subtitle = "Evolution of $1 invested in year 0.")
```



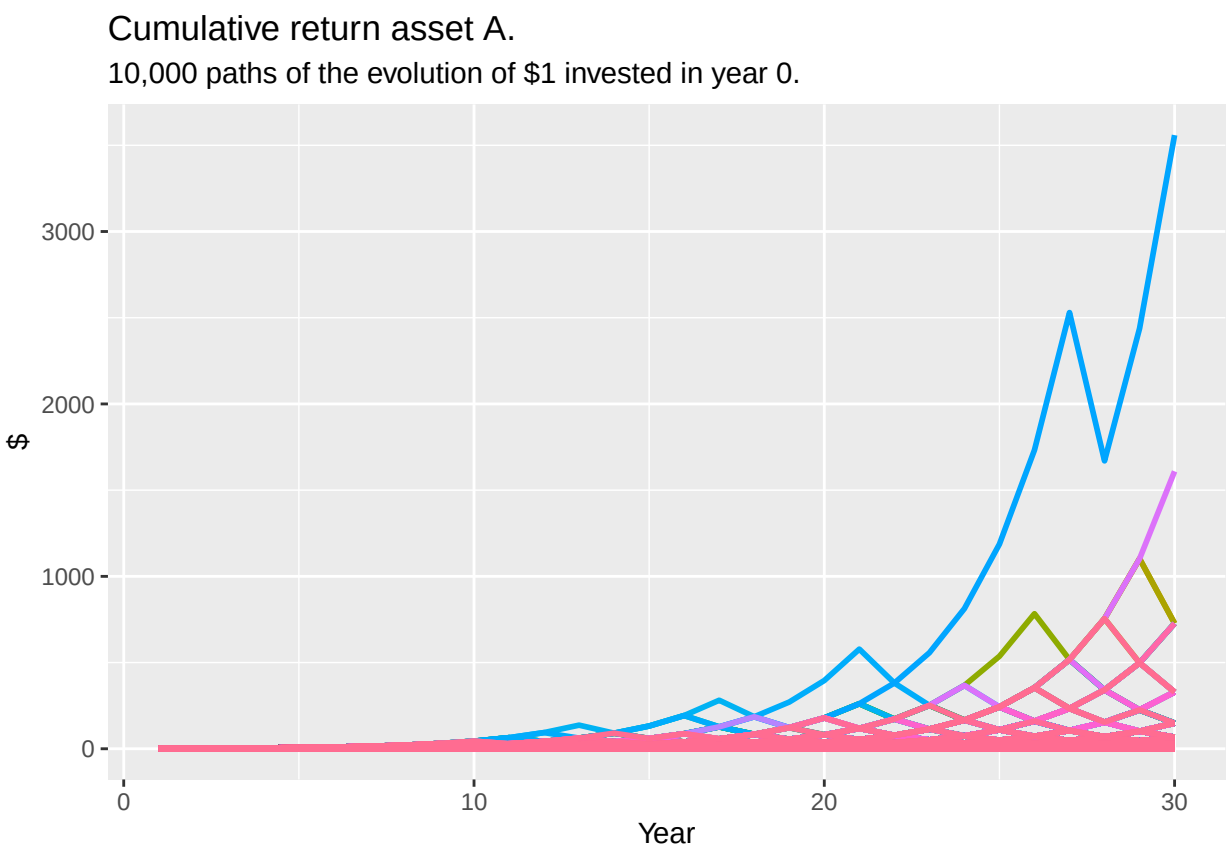
These results represent one single path. So, if we are interested in the most likely value of our investment we need to simulate many paths and then estimate the median. Let's consider we simulate 10000 paths and see the most likely value of our investment.

```r
u = 0.46
d = 0.34
n = 30
set.seed(2, sample.kind="Rounding")
```

```
a.30x10k <- replicate(10000,
                      cumprod(sample(c(1+u, 1-d), n, replace = TRUE))) |>
  as.data.frame() |>
  mutate(year = c(1:30)) |>
  gather(V1:V10000, key = name, value = c.ret)
```

See the results.

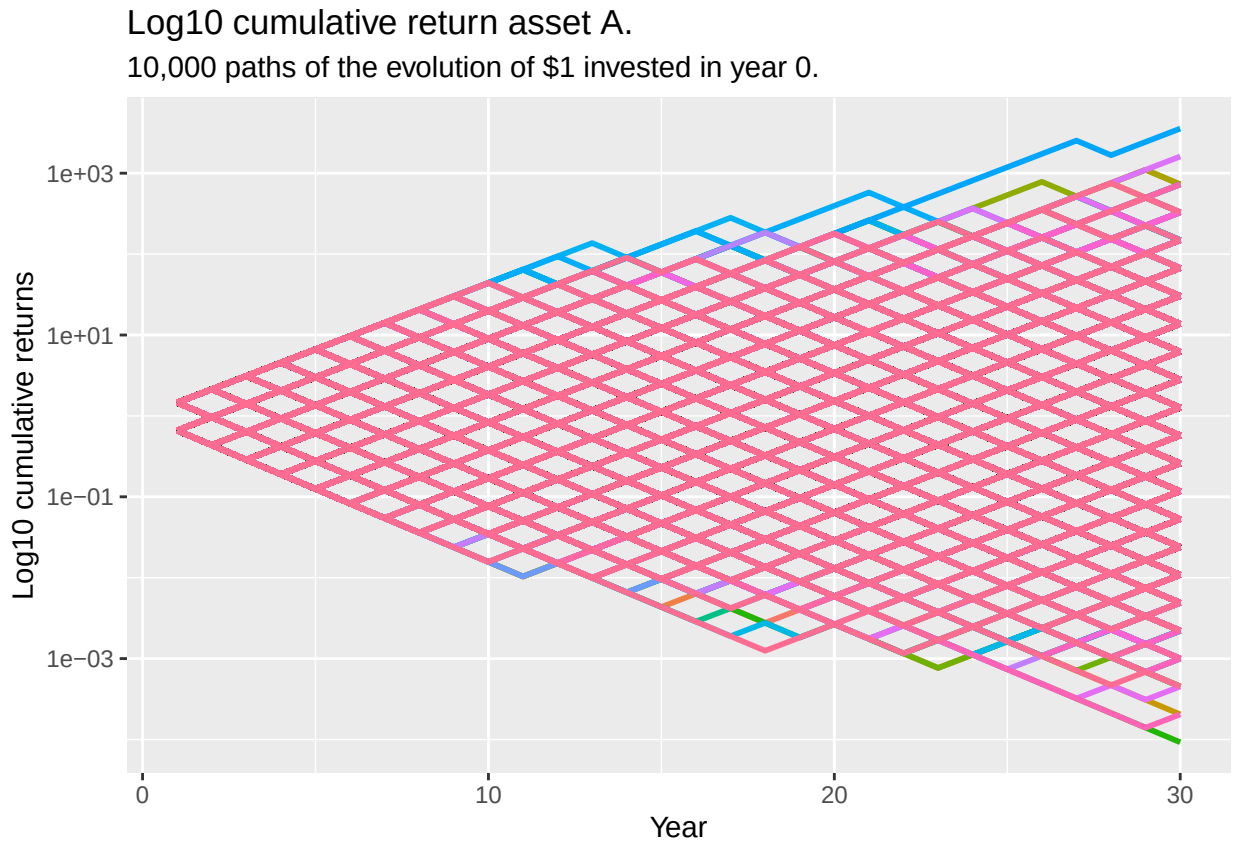
```
a.30x10k |>
ggplot(aes(x = year, y = c.ret, color = name)) +
  geom_line(size = 1) +
  theme(legend.position = "none", legend.title = element_blank()) +
  labs(y = "$",
       x = "Year",
       title = "Cumulative return asset A.",
       subtitle = "10,000 paths of the evolution of $1 invested in year 0.")
```



Note the plot is highly concentrated at low values. It is easier to show the results in logarithm

form.

```
a.30x10k |>
ggplot(aes(x = year, y = c.ret, color = name)) +
  geom_line(size = 1) +
  theme(legend.position = "none", legend.title = element_blank()) +
  labs(y = "Log10 cumulative returns",
       x = "Year",
       title = "Log10 cumulative return asset A.",
       subtitle = "10,000 paths of the evolution of $1 invested in year 0.") +
  scale_y_log10()
```



Here it is easier to see that the paths actually follow a binomial structure. Now, let's see the summary statistics for the 10,000 possible investment value at year 30.

```
set.seed(2, sample.kind="Rounding")
summary(replicate(10000, binomial_tree(u = 0.46, d = 0.34, 30)))
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
```

```
##     0.000     0.117     0.573     5.816     2.806 3559.180
```

Not good news as our 30 year investment of \$1 in this asset A with 6% gross return and 40% volatility leads to \$0.5733923. Even if we drop the set.seed, we end with the same result.

```
median(replicate(10000, binomial_tree(u = 0.46, d = 0.34, 30)))
```

```
## [1] 0.5733923
```

It would be a mistake to consider the mean as the maximum value is very high and not likely to happen. The asset A is then a loosing investment, \$1 would most likely lead to \$0.5733923.

Now, let's consider an asset B. Asset B has an excess return of -0.1%. In the context of a binomial tree: $0.5(1+u)+0.5(1-d) = 0.999$. Asset B has no volatility, imagine is a Treasury Bill or a similar risk-free asset. Solving for $(1+u)$ and $(1-d)$ leads to $(1+u) = 0.999$ and $(1-d) = 0.999$. Asset B is for sure a loosing investment as well.

In a period of 5 years, we would have a random path of ups and downs in the context of a binomial path. Although in this case the result is the same as we have no risk.

```
b <- sample(c(1+(-0.001), 1-(0.001)), 5, replace = TRUE)
b
```

```
## [1] 0.999 0.999 0.999 0.999 0.999
```

The evolution of \$1 dollar invested in this 5 year period would look like this:

```
cumprod(b)
```

```
## [1] 0.999000 0.998001 0.997003 0.996006 0.995010
```

Or simply:

```
prod(b)
```

```
## [1] 0.99501
```

As stated earlier, this asset B is a loosing investment. Let's verify this for 5 and 30 years as we did before for the case of asset A:

```
binomial_tree <- function(u, d, n) {
  prod(sample(c(1+u, 1-d), n, replace = TRUE))
}
set.seed(2, sample.kind="Rounding")
binomial_tree(u = (-0.001), d = (0.001), 5)
```

```
## [1] 0.99501
```

```
binomial_tree(u = (-0.001), d = (0.001), 30)
```

```
## [1] 0.970431
```

It works, a \$1 dollar invested at $t = 0$ would lead to \$0.99501 at $T = 5$. And now we know that a \$1 dollar invested at $t = 0$ would lead to \$0.970431 at $T = 30$. These results represent one single path, but we do not need more as there is no risk. Let's confirm:

```
set.seed(2, sample.kind="Rounding")
median(replicate(10000, binomial_tree(u = -0.001, d = 0.001, 30)))
```

```
## [1] 0.970431
```

The result is the same. Asset B is a loosing investment as well. But, what if we combine both assets A and B in an equally weighted portfolio C?

```
set.seed(2, sample.kind="Rounding")
median(replicate(1, binomial_tree(u = (0.46-0.001)/2,
                                  d = (0.34+0.001)/2, 30)))
```

```
## [1] 2.951197
```

This is how we combine two losing investments into a winner.

The new asset C or portfolio C has a return of: $0.06/2 - 0.001/2 = 0.0295$ or 2.95%. The volatility of C is $\sqrt{0.5^2(0.4)^2 + 0.5^2(0)^2 - 0} = 0.2$ or 20%. Therefore, portfolio C has a lower return and risk than A. Given that A and B are uncorrelated, the diversification gain is high so we can combine two losing investments into a winner.

Let's visualize how these assets behave. I have to set the seed to have nice results. But as we show before this works well when simulating many paths and computing the median to evaluate the expected cumulative return of each asset.

```
u = 0.46
d = 0.34
xx=8
set.seed(xx, sample.kind="Rounding")
a30 <- sample(c(1+u, 1-d), 30, replace = TRUE)
u = - 0.001
d = 0.001
b30 <- sample(c(1+u, 1-d), 30, replace = TRUE)
```

```
u = (0.46 - 0.001)/2
d = (0.34 + 0.001)/2
set.seed(xx, sample.kind="Rounding")
c30 <- sample(c(1+u, 1-d), 30, replace = TRUE)
```

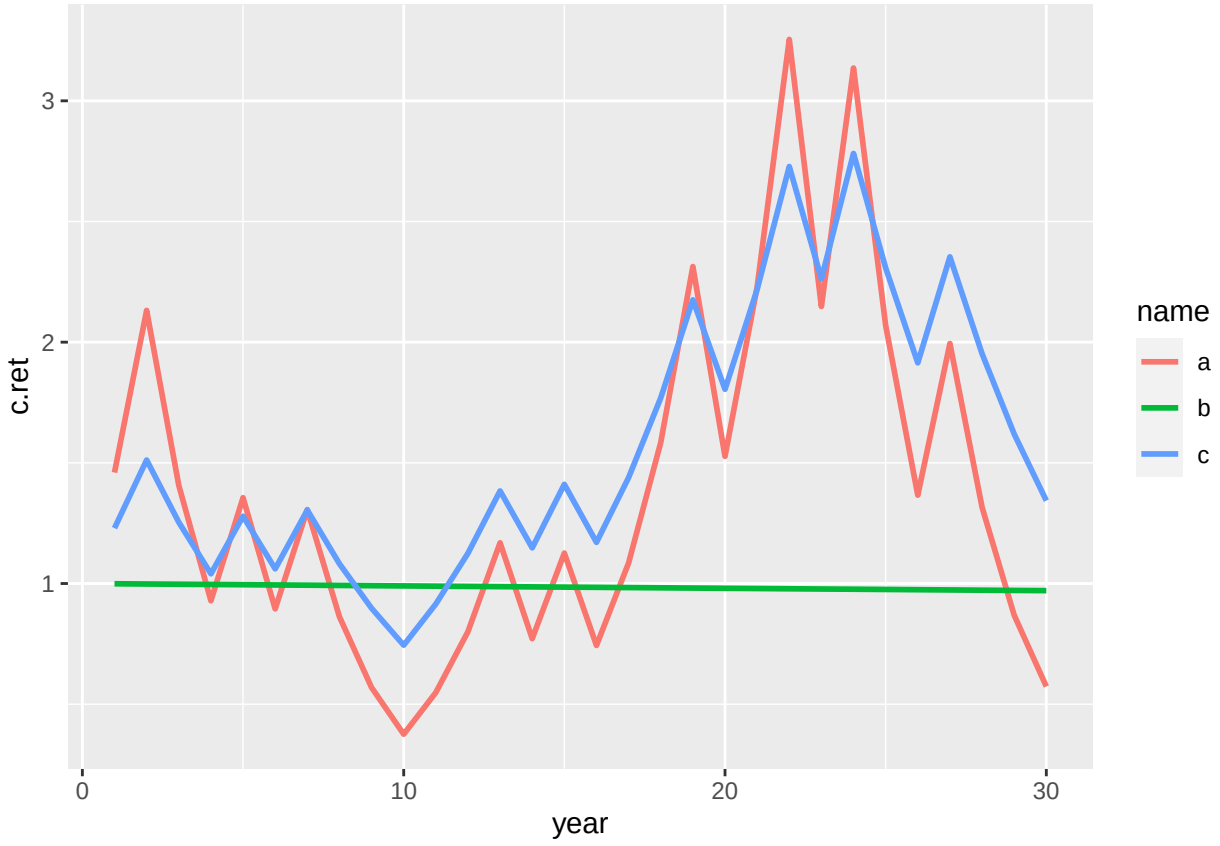
Below, it is clear that asset C wins less when asset A wins. Similarly, asset C looses less when asset A losses. This is because asset C is less risky than asset A.

```
abc <- as.data.frame(cbind(
  year = c(1:30), a = cumprod(a30), b = cumprod(b30), c = cumprod(c30)))
abc
```

```
##    year         a         b         c
## 1     1 1.4600000 0.9990000 1.2295000
## 2     2 2.1316000 0.9980010 1.5116703
## 3     3 1.4068560 0.9970030 1.2539305
## 4     4 0.9285250 0.9960060 1.0401353
## 5     5 1.3556464 0.9950100 1.2788464
## 6     6 0.8947267 0.9940150 1.0608031
## 7     7 1.3063009 0.9930210 1.3042574
## 8     8 0.8621586 0.9920279 1.0818815
## 9     9 0.5690247 0.9910359 0.8974207
## 10   10 0.3755563 0.9900449 0.7444105
## 11   11 0.5483122 0.9890548 0.9152527
## 12   12 0.8005358 0.9880658 1.1253032
## 13   13 1.1687822 0.9870777 1.3835602
## 14   14 0.7713963 0.9860906 1.1476632
## 15   15 1.1262386 0.9851045 1.4110519
## 16   16 0.7433175 0.9841194 1.1704676
## 17   17 1.0852435 0.9831353 1.4390899
## 18   18 1.5844555 0.9821522 1.7693610
## 19   19 2.3133050 0.9811700 2.1754294
## 20   20 1.5267813 0.9801889 1.8045187
## 21   21 2.2291007 0.9792087 2.2186557
## 22   22 3.2544870 0.9782295 2.7278372
## 23   23 2.1479614 0.9772512 2.2627409
## 24   24 3.1360237 0.9762740 2.7820400
```

```
## 25   25 2.0697756 0.9752977 2.3077022
## 26   26 1.3660519 0.9743224 1.9142389
## 27   27 1.9944358 0.9733481 2.3535568
## 28   28 1.3163276 0.9723747 1.9522754
## 29   29 0.8687762 0.9714024 1.6194124
## 30   30 0.5733923 0.9704310 1.3433026
```

```
abc |>
  gather(a:c, key = name, value = c.ret) |>
  ggplot(aes(x = year, y = c.ret, color = name)) +
  geom_line(size = 1)
```

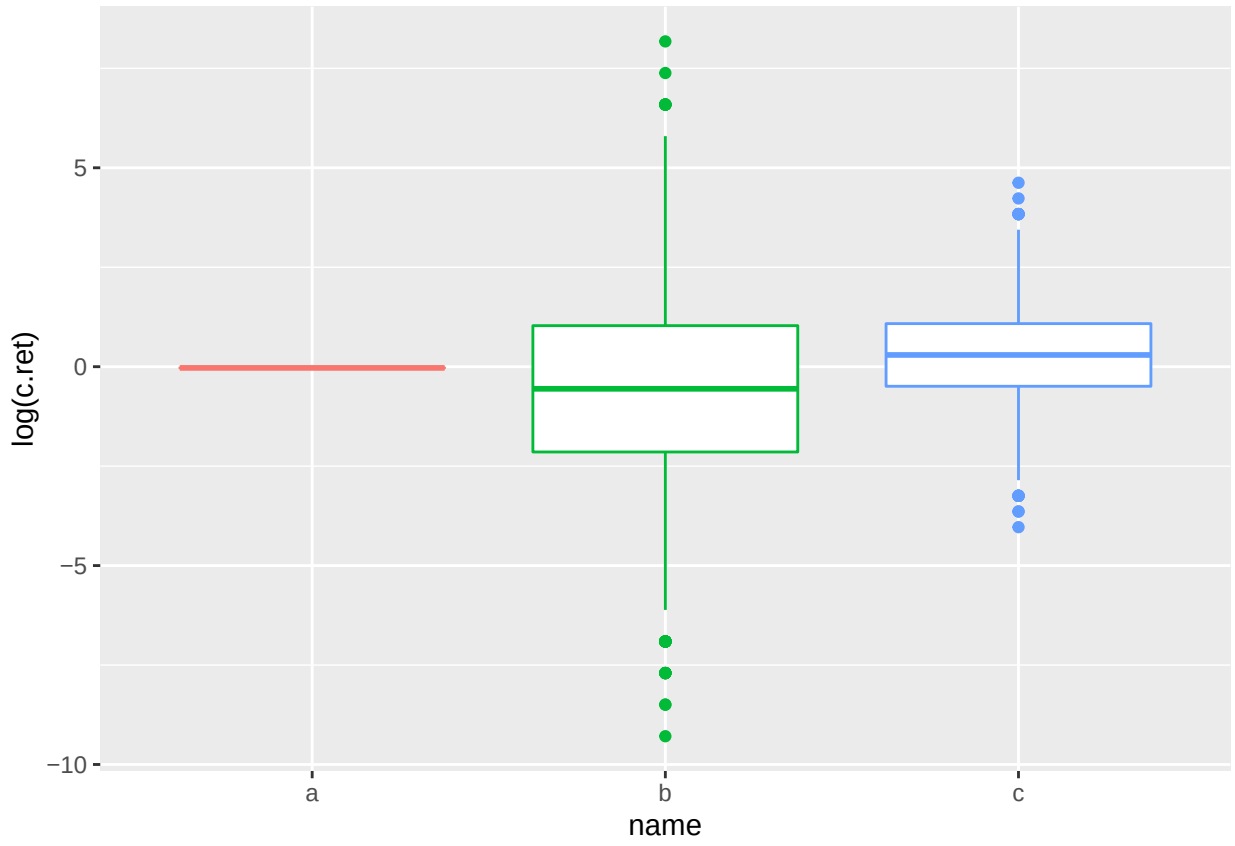



```
set.seed(2, sample.kind="Rounding")
ax <- (replicate(10000, binomial_tree(u = -0.001, d = 0.001, 30)))
set.seed(2, sample.kind="Rounding")
cx <- (replicate(10000, binomial_tree(u = (0.46-0.001)/2,
                                      d = (0.34+0.001)/2, 30)))
set.seed(2, sample.kind="Rounding")
```

```
bx <-(replicate(10000, binomial_tree(u = 0.46, d = 0.34, 30)))

abcx <- as.data.frame(cbind(
  year = c(1:30), a = ax, b = bx, c = cx))

abcx |>
  gather(a:c, key = name, value = c.ret) |>
  ggplot(aes(x = name, y = log(c.ret), color = name)) +
  geom_boxplot() +
  theme(legend.position = "none")
```




```
  #geom_hline(yintercept = 0, linetype = "longdash")
```

Stock price paths are useful to value option stocks. Let's review some stochastic processes in the next section.

4 Wiener processes.

Here we review stochastic process that are useful in finance, especially in the Black-Scholes context.

4.1 The basic process.

A simple Wiener process.

```
rm(list=ls()) #Removes all items in Environment!
# A typical Wiener process.
TT <- 5
N <- 10000
dt <- TT / N
Time <- seq(from = dt, to = TT, by = dt)
set.seed(560746) # for reproducibility purposes.
delta_z <- rnorm(N, 0, 1) * dt^0.5
z <- 25 + cumsum(delta_z) # The initial price is 25.
```

Graphically:

```
plot(Time, z, type = "l",
     ylab = "z in the texbook, it could be a stock price",
     xlab = "Time, this could be 5 days or even shorter periods")
lines(Time, seq(25, (25 + sum(delta_z)), length.out = N), lty = 2,
      col = "purple")
points(Time[1], 25, col = "blue", cex = 2, pch = 16)
points(Time[N], z[N], col = "red", cex = 2, pch = 16)
legend("topleft", legend = c(round(z[N], 2)), bg = "white",
       text.col = "red")
```

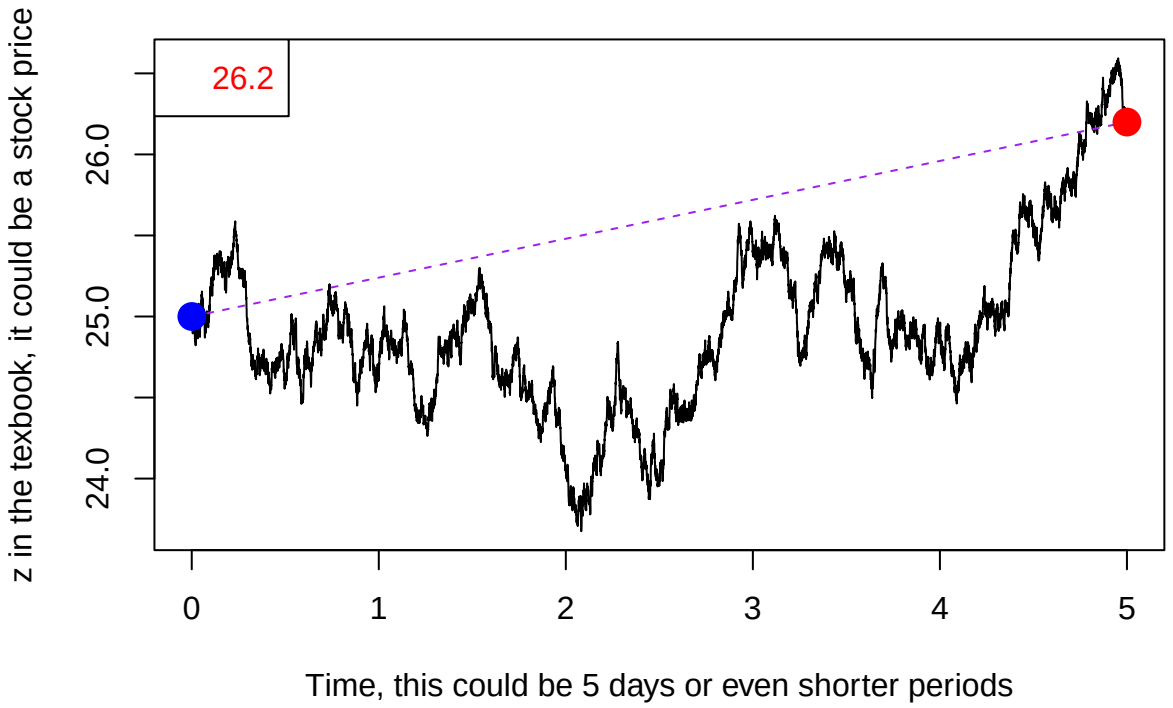


Figure 4.1.1: Wiener process 10,000 steps from blue z(0)=25 to z(T)=26.2 in red.

This is supposed to mimic the evolution of a stock price. Now, we propose to produce not only one but 100 processes.

```
# 100 Wiener processes.
set.seed(365633)
mat.delta_z <- matrix(rnorm(100 * N, mean = 0, sd = 1), 100, N)
mat.z1 <- 25 + t(apply(mat.delta_z, 1, cumsum))
mat.z <- t(cbind(matrix(25, 100), mat.z1))
Time2 <- c(0, Time)
pred <- mean(mat.z[(N + 1), ])
# 95% range.
rangetop <- pred + qnorm(0.975) * var(mat.z[(N + 1), ])^0.5
rangedown <- pred - qnorm(0.975) * var(mat.z[(N + 1), ])^0.5
```

Graphically.

```
matplot(Time2, mat.z, type = "l", lty = 1, col = rgb(0, 0, 1, 0.3),
     ylab = "z in the texbook, it could be a stock price",
     xlab = "Time")
abline(h = pred, lty = 2, col = "red")
abline(v = 0, lty = 2)
lines(Time2, seq(25, pred, length.out = N + 1), lty = 2)
points(Time2[N + 1], pred, col = "red", cex = 2, pch = 16)
points(0, 25, col = "black", cex = 2, pch = 16)
legend("topleft", legend = c(round(pred, 2)), bg = "white",
       text.col = "red")
abline(h = rangetop, lty = 2)
abline(h = rangedown, lty = 2)
```

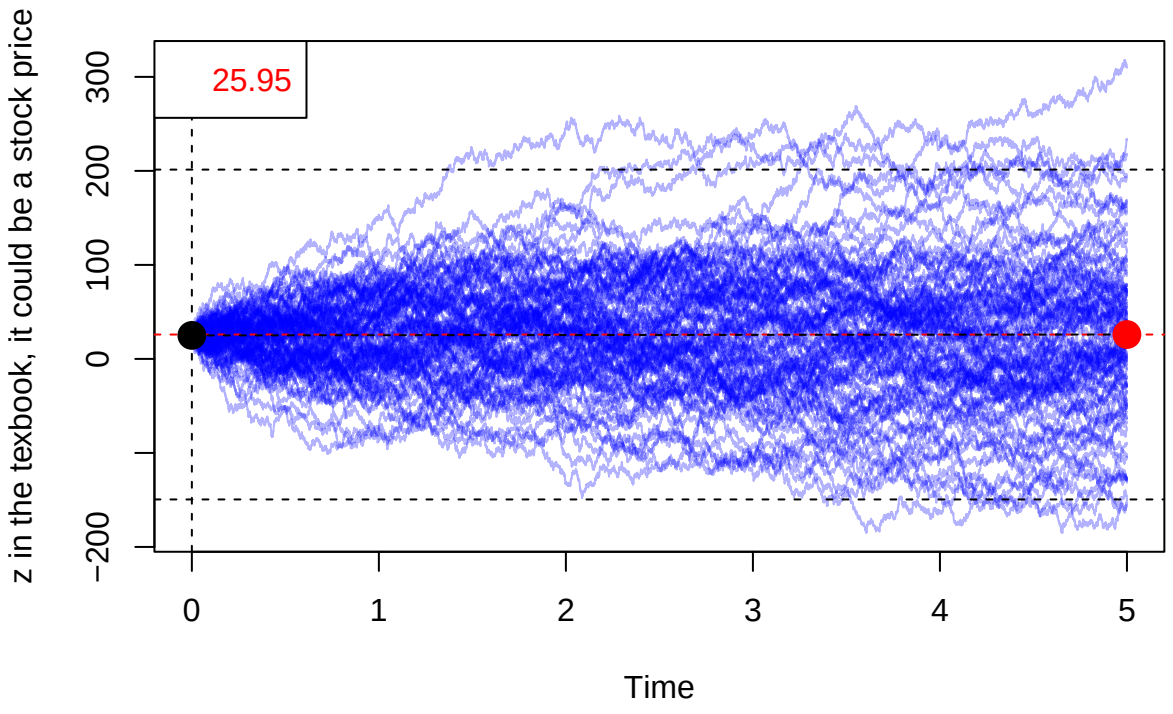


Figure 4.1.2: 100 Wiener process 10,000 steps, initial price at 25, red is the mean at T.

Let's confirm this 95% confidence interval.

```r
sum(mat.z[(N + 1), ] > rangetop)
```

```
## [1] 3
```

```r
sum(mat.z[(N + 1), ] < rangedown)
```

```
## [1] 2
```

This means that 5 values are outside this 95% confidence interval. Confidence intervals are then correct.

```r
summary(mat.z[(N + 1), ])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -155.23  -29.14   23.33   25.95   72.21  310.30
```

A potential problem is that the values can be negative. This is problematic when we are interested to model stock prices.

Let's explore deeper the stochastic process below.

```r
set.seed(1)
delta_z <- rnorm(N, 0, 1) * dt^0.5
z <- 25 + cumsum(delta_z)
### A stochastic process
par(mfrow = c(2, 2), mai = c(0.4, 0.4, 0.4, 0.4))
par(pty = "s")
plot(Time, z, type = "l", xlim = c(0, TT), ylim = c(22.5, 26),
     main = "z", ylab = "")
plot(Time, delta_z, type = "h", xlim = c(0, TT),
     main = "delta_z", ylab = "")
abline(h = 0, lty = 2, col = "orange", lwd = 2)
plot(Time, z, type = "l", xlim = c(0, 1/12), ylim = c(24.9, 25.35),
     main = "z (first month only)", ylab = "")
plot(Time, delta_z, type = "h", xlim = c(0, 1/12),
     col = ifelse(delta_z < 0, "red", "blue"), ylim = c(-0.055, 0.055),
     main = "delta_z (first month only)", ylab = "")
```

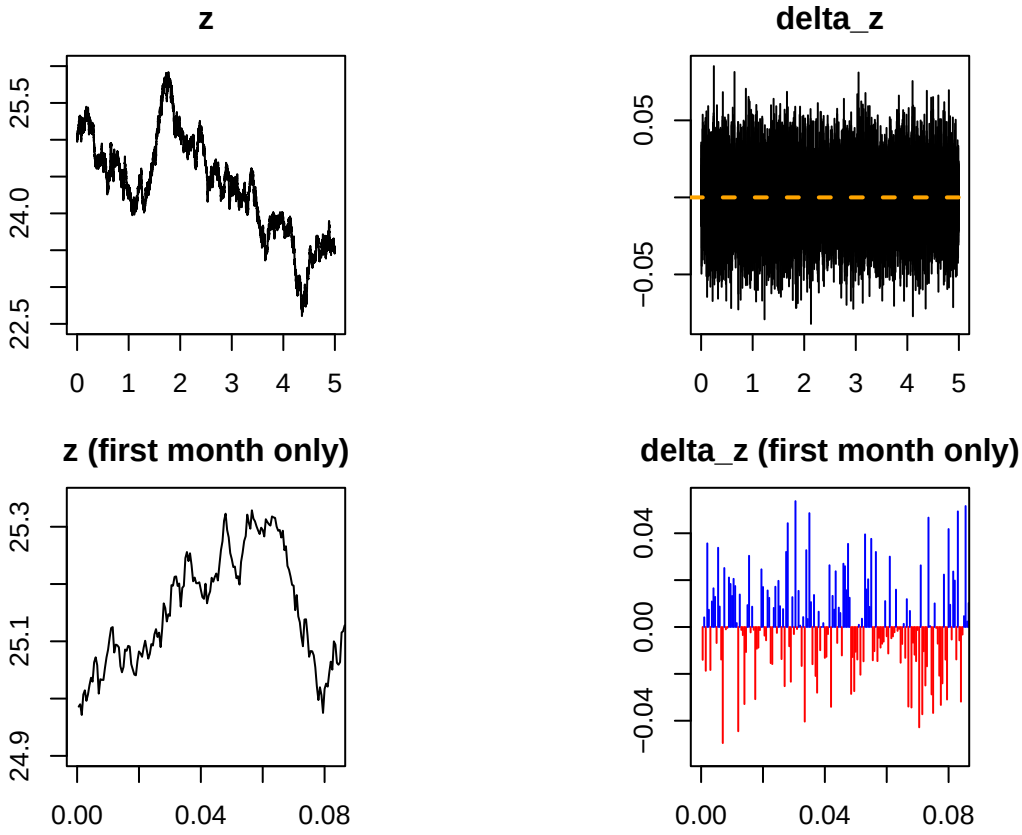


Figure 4.1.3: A stochastic process.

These processes are driven by a random component. The lower panel is revealing because we can see how the random component goes from positive to negative without any pattern, it is entirely random. Now, let's explore the role of Δt.

```
par(mfrow=c(2, 2), mai = c(0.4, 0.4, 0.4, 0.4))
par(pty = "s")
plot(Time[(seq(from = 1, to = N, length.out = 10))],
     z[(seq(from = 1, to = N, length.out = 10))], type = "l",
     ylim = c(22.5, 26), main = "Relatively large value of delta_t",
     ylab = "")
abline(v = 0, lty = 2)
abline(v = 5, lty = 2)
plot(Time[(seq(from = 1, to = N, length.out = 100))],
     z[(seq(from = 1, to = N, length.out = 100))], type = "l",
```

```
     ylim = c(22.5, 26), ylab = "z", main = "Smaller value of delta_t")
abline(v = 0, lty = 2)
abline(v = 5 , lty = 2)
plot(Time[(seq(from = 1, to = N, length.out = 1000))],
     z[(seq(from = 1, to = N, length.out = 1000))], type = "l",
     ylim = c(22.5, 26), ylab = "z", main = "Smaller value of delta_t")
abline(v = 0, lty = 2)
abline(v = 5 , lty = 2)
plot(Time, z, type = "l", ylim = c(22.5, 26), # length 10,000
     main = "True process as delta_t tends to 0")
abline(v = 0, lty = 2)
abline(v = 5, lty = 2)
```

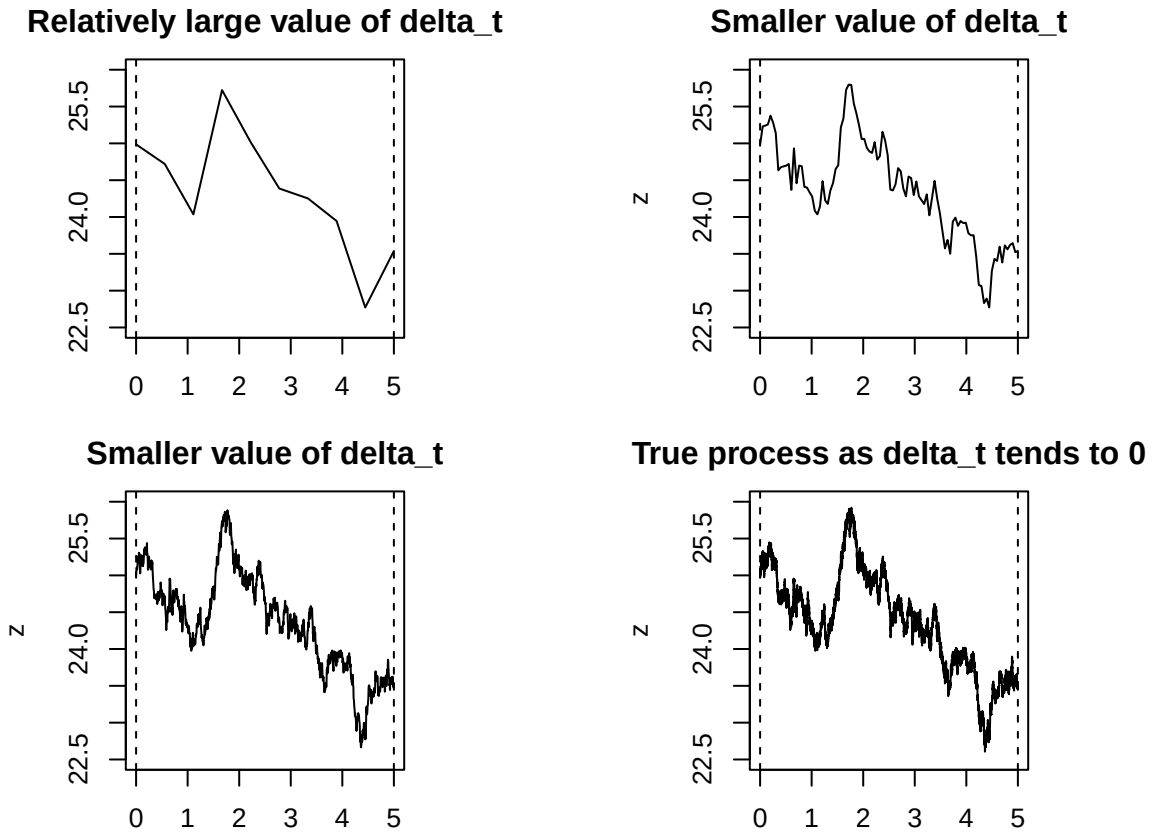


Figure 4.1.4: Wiener process.

The Δt value determines how frequent are the changes.

A few properties of stochastic processes.

```
mean(delta_z) # should be zero
```

```
## [1] -0.0001461726
```

```
var(delta_z)^0.5 # should be dt^.5
```

```
## [1] 0.02263698
```

```
dt^0.5
```

```
## [1] 0.02236068
```

```
var(delta_z) # should be dt
```

```
## [1] 0.0005124328
```

```
dt
```

```
## [1] 5e-04
```

4.2 The generalized Wiener process.

Now let's analyze the case of the generalized Wiener process.

```
# Figure 14.2
TT <- 50
N <- 200
dt <- TT / N
Time <- seq(from = 0, to = TT, by = dt)
set.seed(123)
delta_z <- 1.5 * rnorm(N, 0, 1) * dt^0.5
delta_x <- (0.3 * dt) + delta_z
x <- c(0, cumsum(delta_x))
z <- c(0, cumsum(delta_z))
```

Graphically:

```
plot(Time, x, type = "l", lwd = 2, ylim = c(-7, 22))
lines(Time, z, col = "red", lwd = 2)
lines(Time, 0.3 * Time, col = "blue", lwd = 2)
abline(0, 0)
```

```
abline(v = 0)
legend("topleft", legend = c("Generalized Wiener process",
                             "Drift", "Basic Wiener process"),
       col = c("black", "blue", "red"), lwd = 3, bg = "white", cex = 0.8)
```

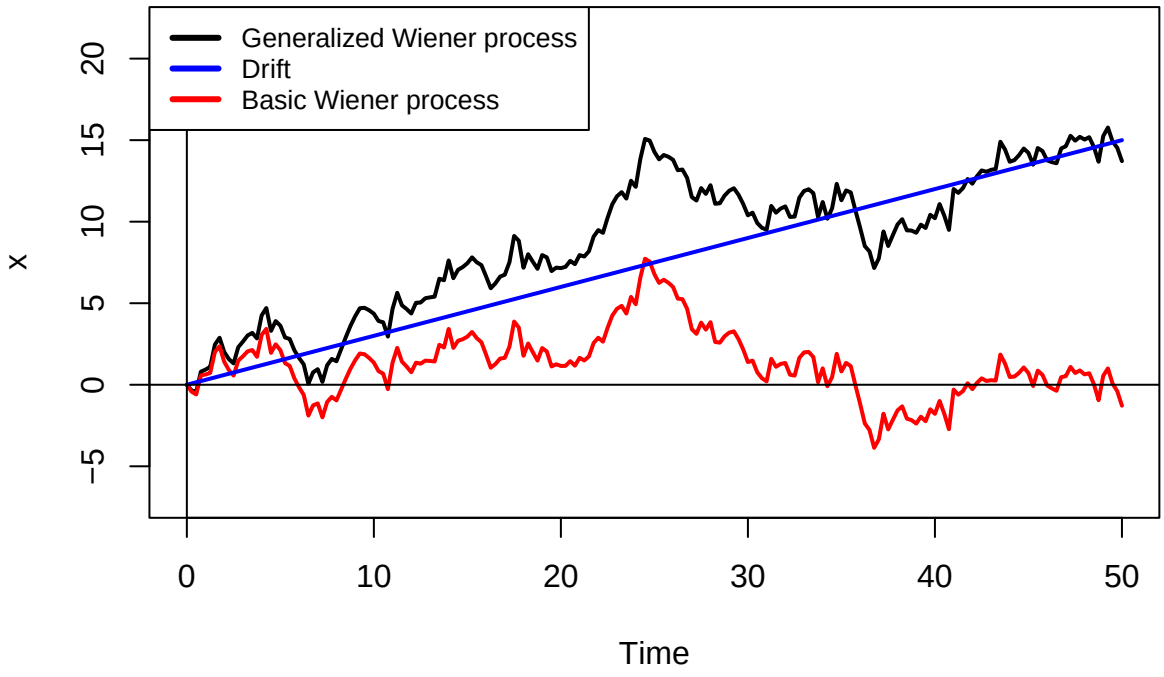


Figure 4.2.1: In Hull: Figure 14.2

Here, the generalized Wiener process is decomposed into the drift and the basic Wiener process. A zoom of the same plot.

```
# Figure 14.2 Zoom
plot(Time, x, type = "b", ylim = c(-0.6, 1), xlim = c(0, 1), lwd = 2)
lines(Time, z, col = "red", lwd = 2, type = "b")
lines(Time, 0.3 * Time, col = "blue", lwd = 2, type = "b")
abline(0, 0)
abline(v = 0)
abline(v = dt, lty = 2)
```

```
abline(v = dt * 2, lty = 2)
abline(v = dt * 3, lty = 2)
abline(v = dt * 4, lty = 2)
points(dt, 0, pch = 1, col = "blue", lwd = 2)
points(dt * 2, 0, pch = 1, col = "blue", lwd = 2)
points(dt * 3, 0, pch = 1, col = "blue", lwd = 2)
points(dt * 4, 0, pch = 1, col = "blue", lwd = 2)
legend("topleft", legend = c("Generalized Wiener process",
                             "Drift", "Basic Wiener process"),
       col = c("black", "blue", "red"), lty = 1, bg = "white", lwd = 2)
```

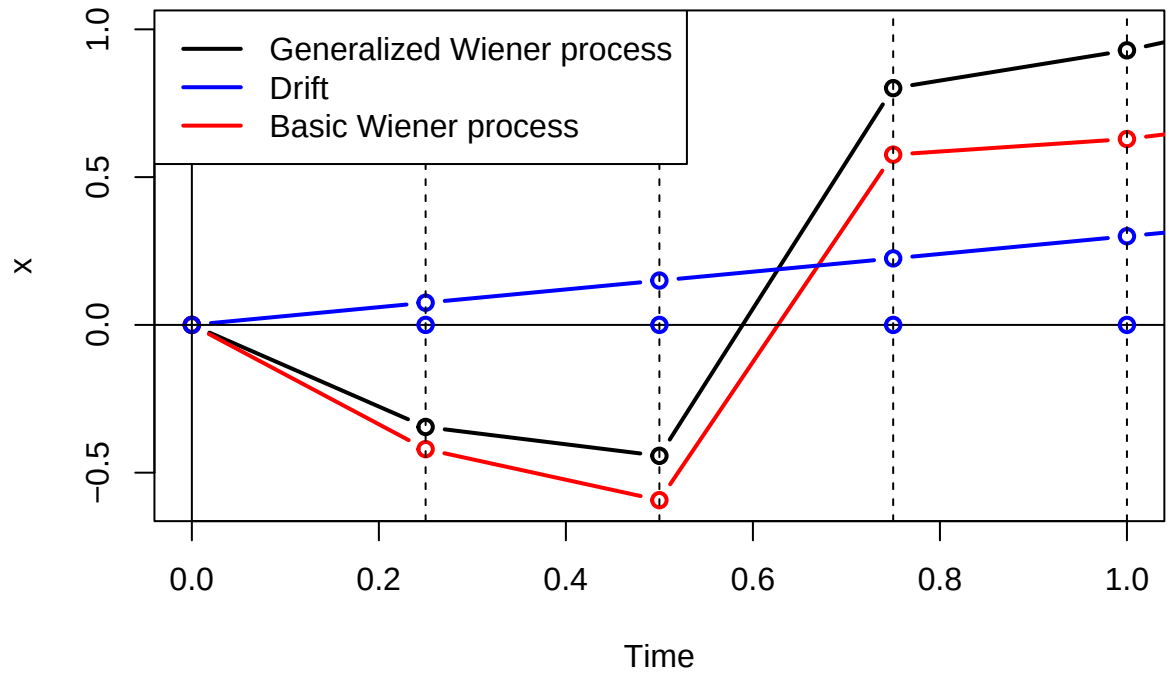


Figure 4.2.2: In Hull: Figure 14.2 (zoom)

This is a nice visual representation of: generalized = drift + Wiener.

See how the process changes when we consider a lower value of a.

```r
set.seed(123)
delta_xlow <- (0.15 * dt) + delta_z
xlow <- c(0, cumsum(delta_xlow))
# Now plot.
plot(Time, xlow, type = "l", ylim = c(-7, 22), lwd = 2, ylab = "x")
lines(Time, x, lwd = 2, col = "grey")
lines(Time, z, lwd = 2, col = "red")
lines(Time, 0.15 * Time, lty = 2, lwd = 2)
lines(Time, 0.3 * Time, lty = 2, lwd = 2, col = "grey")
abline(0, 0)
abline(v = 0)
legend("topleft", legend = c("Original generalized Wiener process",
    "Original drift", "New generalized Wiener process", "New drift",
    "Basic Wiener process"), lty = c(1, 2, 1, 2, 1), bg = "white", lwd = 2,
     col = c("grey", "grey", "black", "black", "red"), cex = 0.7)
```

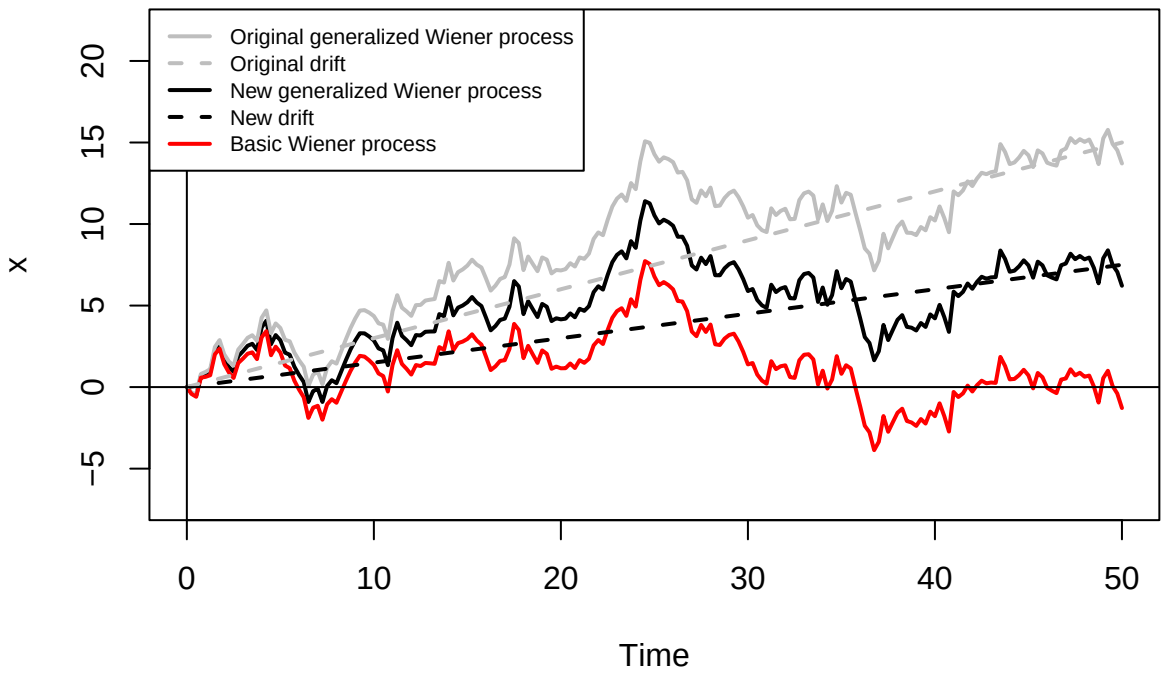


Figure 4.2.3: Generalized Wiener process: lower a

And a higher value of b.

```
set.seed(123)
delta_zhigh <- 3 * rnorm(N, 0, 1) * dt^0.5
delta_xhigh <- (0.3 * dt) + delta_zhigh
xhigh <- c(0, cumsum(delta_xhigh))
zhigh <- c(0, cumsum(delta_zhigh))
plot(Time, xhigh, type = "l", ylim = c(-7, 22), lwd = 2, ylab = "x")
lines(Time, x, lwd = 2, col = "grey")
lines(Time, zhigh, lwd = 2, col = "red")
lines(Time, z, lwd = 2, col = "grey")
lines(Time, 0.3 * Time, lty = 2, lwd = 2)
abline(0, 0)
abline(v = 0)
```

```
legend("topleft", legend = c("New generalized Wiener process",
      "Original generalized Wiener process", "Drift" ,
      "New basic Wiener process", "Original basic Wiener process"),
      lty = c(1, 1, 2, 1, 1), bg = "white", lwd = 2,
       col = c("black", "grey", "black", "red", "grey"), cex = 0.7)
```

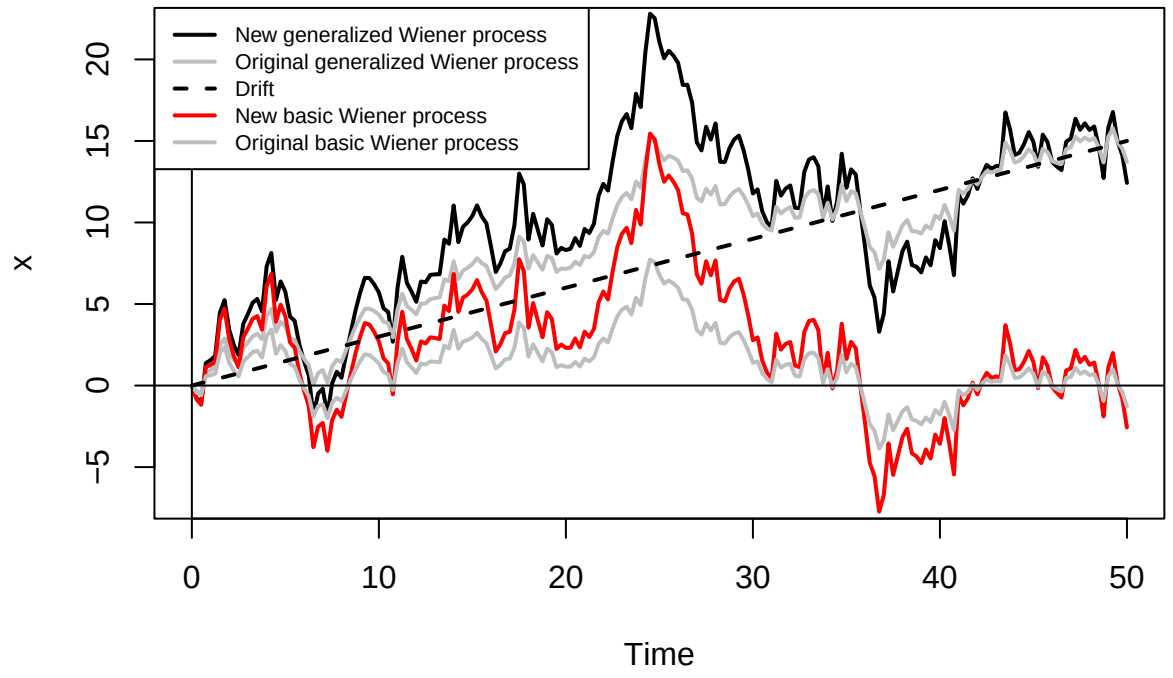


Figure 4.2.4: Generalized Wiener process: higher b

Note the changes are now more pronounced.

```
# Table 14.1
set.seed(19256)
delta_S <- rep(0, 10)
S <- rep(100, 10)
epsilon <- rep(0, 10)
for(i in 1:10){
  epsilon[i] <- rnorm(1, 0, 1)
```

```
  delta_S[i] <- 0.15 * (1 / 52) * S[i] + 0.3 * ((1 / 52)^0.5) *
    epsilon[i] * S[i]
  S[i+1] <- S[i] + delta_S[i]
}
epsilon <- c(epsilon, NA)
delta_S <- c(delta_S, NA)
results <- data.frame(S, epsilon, delta_S)
kable(results, caption = "Simulation of stock price (Table 14.1 in Hull).",
      digits = 4) |>
kable_styling(latex_options = "HOLD_position")
```

Table 4.2.1: Simulation of stock price (Table 14.1 in Hull).

S	epsilon	delta_S
100.0000	0.8269	3.7284
103.7284	0.4958	2.4388
106.1672	-0.9314	-3.8075
102.3597	-0.1496	-0.3418
102.0179	0.7310	3.3969
105.4148	-0.6111	-2.3761
103.0387	1.2491	5.6516
108.6904	-0.6243	-2.5096
106.1808	0.6995	3.3964
109.5772	0.3613	1.9629
111.5401	NA	NA

It is interesting to note that in Hull the final simulated price is 111.54 and here it is 111.5401. Is this a pure coincidence?

A correlated process.

```
# Section 14.5
rho <- 0.8
TT <- 1
steps <- 1000
dt <- TT / steps
Time <- seq(from = dt, to = TT, by = dt)
```

```
set.seed(123)
e1 <- rnorm(steps, 0, 1)
e2 <- rho * e1 + ((1 - rho^2)^0.5) * rnorm(steps, 0, 1)
delta_z1 <- 1.5 * e1 * dt^0.5
delta_z2 <- 1.5 * e2 * dt^0.5
delta_x1 <- (0.3 * dt) + delta_z1
delta_x2 <- (0.3 * dt) + delta_z2
x1 <- cumsum(delta_x1)
x2 <- cumsum(delta_x2)
x <- c(x1, x2)
```

Visually:

```
plot(Time, x1, type = "l", ylim = c(min(x), max(x)), ylab = "x1 and x2")
lines(Time, x2, col = "red")
abline(0, 0)
```

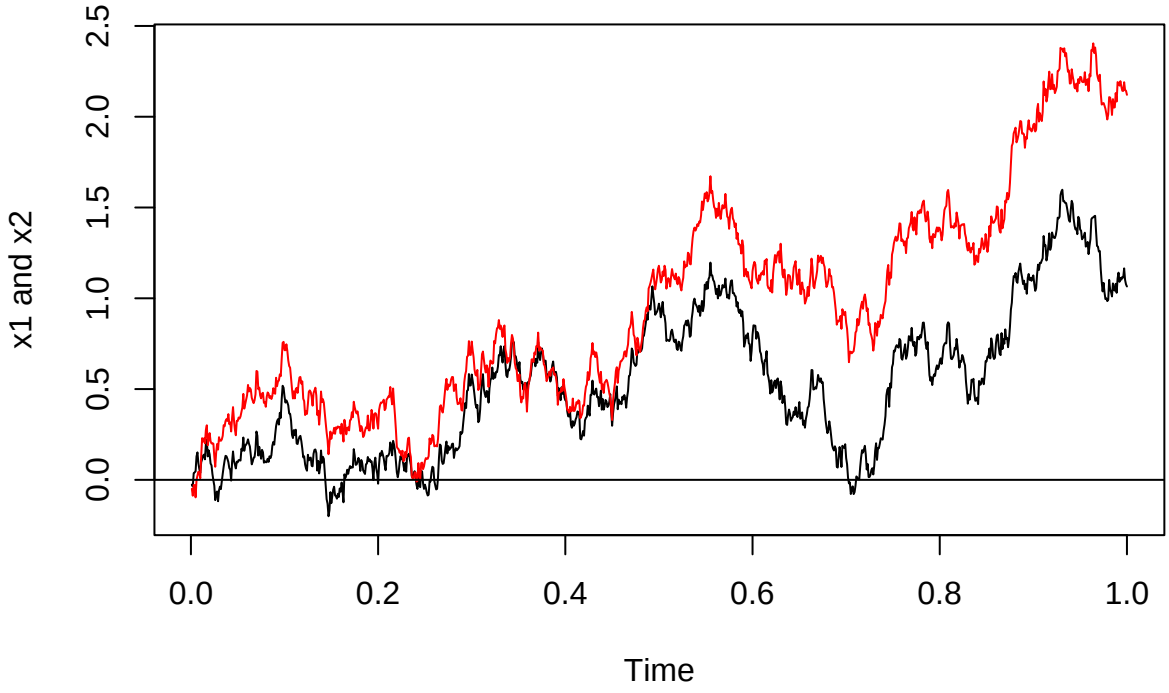


Figure 4.2.5: Correlated processes x1 and x2.

The following code needs revision.

```
# Figure 14.2 with Itô process
TT <- 50
N <- 200
dt <- TT / N
Time <- seq(from = 0, to = TT, by = dt)
set.seed(1)
axt <- 0.3 + rnorm(N + 1, 0, 1) * 0.001 * Time
bxt <- 0.001 * Time + 0.1
delta_z <- bxt * rnorm(N + 1, 0, 1) * dt^0.5
delta_x <- (axt * dt) + delta_z
xI <- cumsum(delta_x)
zI <- cumsum(delta_z)
```

```
# Now plot.
plot(Time, xI, type= "l", lwd = 2, ylim = c(0, 18), ylab = "x")
lines(Time, zI, col = "red", lwd = 2)
lines(Time, axt * Time, col = "blue", lwd = 2)
abline(0, 0)
abline(v = 0)
legend("topleft", legend = c("Itô process",
                             "Stochastic drift", "Wiener process"),
       col = c("black", "blue", "red"), lty = 1, bg = "white", lwd = 2)
```

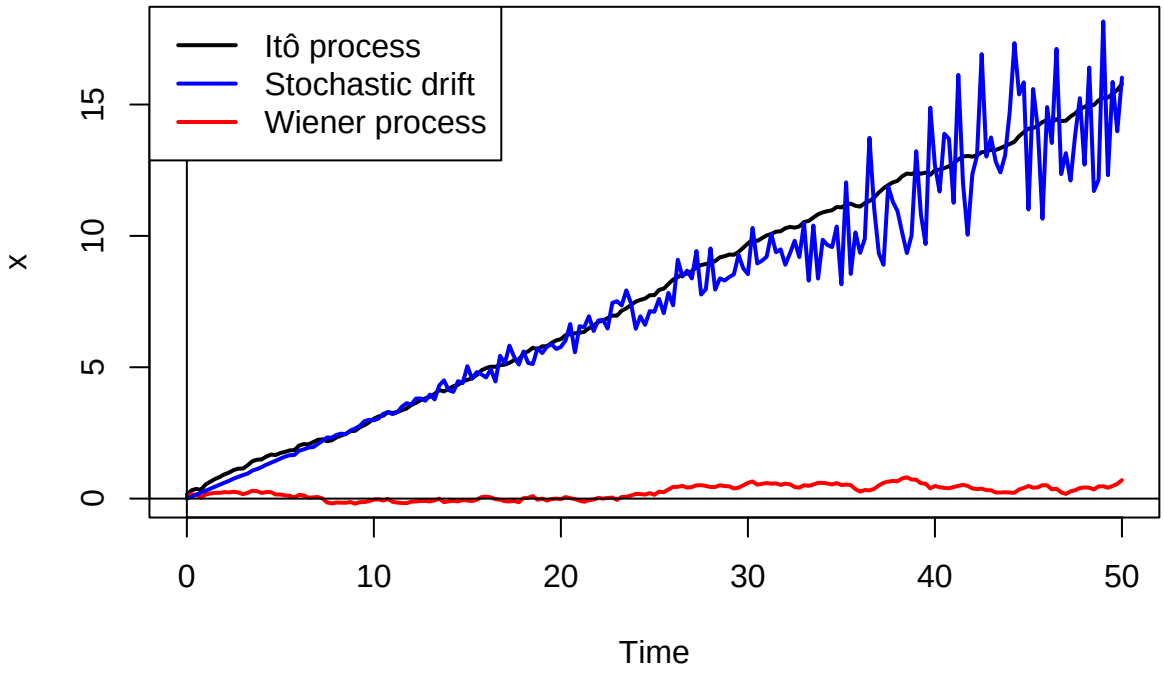


Figure 4.2.6: Itô process. This figure needs revision.

This is a clear view of a typical Itô process. Taken from Esteban Moro.

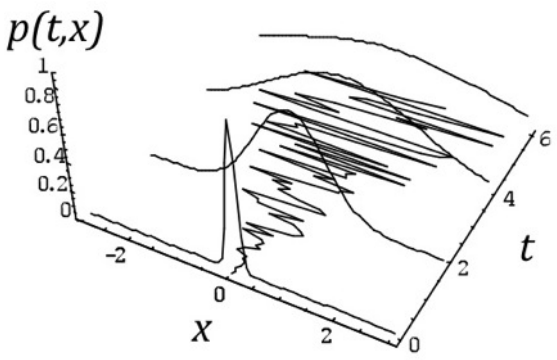


Figure 4.2.7: Itô process.

The model of stock price behavior implies that a stock's price at time T, given its price today, is lognormally distributed. The following is similar to example 15.1 in Hull.

```
set.seed(10101010)
for(j in 1:100) {
  delta_S <- NULL
  S <- rep(40, 365) # Now it is 40 for all days, the loop fill this out.
  epsilon <- NULL
  for(i in 1:365) {
    epsilon[i] <- rnorm(1, 0, 1)
    delta_S[i] <- 0.16 * (1 / 365) * S[i] + 0.2 * ((1 / 365)^0.5) *
      epsilon[i] * S[i]
    S[i+1] <- S[i] + delta_S[i]
    }
  if (j==1) {
    plot(S, type = "l", ylim = c(20, 80), ylab = "$", xlab = "Time (days)")
    abline(h = 10, col = "red", lwd = 3)
    }
  lines(S)
  }
abline(h = 32.55, lwd = 2, col = "red")
abline(h = 56.56, lwd = 2, col = "red")
abline(v = 180, lwd = 2, col = "red")
```

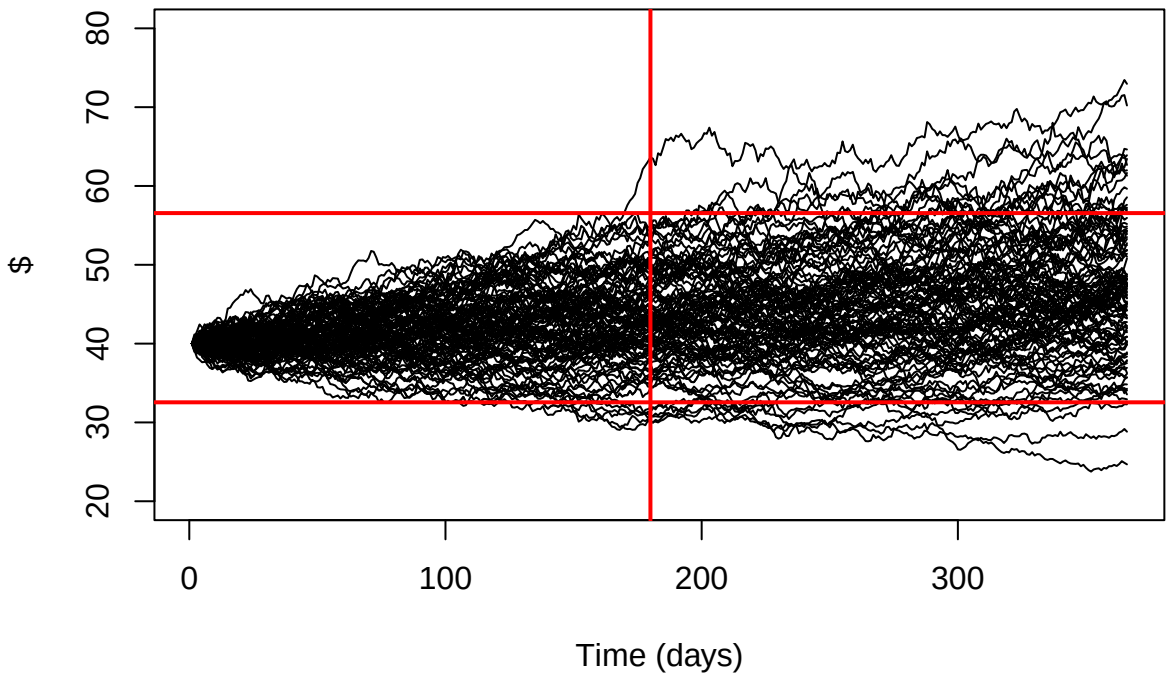


Figure 4.2.8: A geometric Brownian motion simulation: 100 stock price paths.

The code above can be problematic for a few reasons. Can you see why? An alternative code is the following. Again, this is similar to example 15.1 in Hull.

```
S0 <- 40
nDays <- 360
mu <- 0.16
sig <- 0.2
TT <- 0.5
mean.logST <- log(S0)+(mu - (sig^2 / 2)) * TT
variance.logST <- sig^2 * TT
lower.ST <- exp(mean.logST + qnorm(0.025) * variance.logST^0.5)
upper.ST <- exp(mean.logST + qnorm(0.975) * variance.logST^0.5)
set.seed(3)
nSim <- 1000
```

```
SP_sim <- matrix(0, nrow = nDays, ncol = nSim)
for(i in 1:nSim){
  SVec <- rep(0, nDays)
  SVec[1] <- S0
  for(j in 2:nDays){
    DeltaS <- mu * SVec[j - 1] * (1 / nDays) + sig * SVec[j - 1] *
      rnorm(1) * (1 / nDays)^0.5
    SVec[j] <- SVec[j - 1] + DeltaS
  }
  SP_sim[, i] <- SVec
}
```

Now, let's plot the results.

```
matplot(SP_sim, type = 'l', col = rgb(0, 0, 1 ,0.3), lty = 1,
        ylab = 'This is similar as example 15.1',
        xlab = "Time")
points(0, S0, col = "black", pch = 19)
abline(h = lower.ST)
abline(h = upper.ST)
abline(v = 180)
```

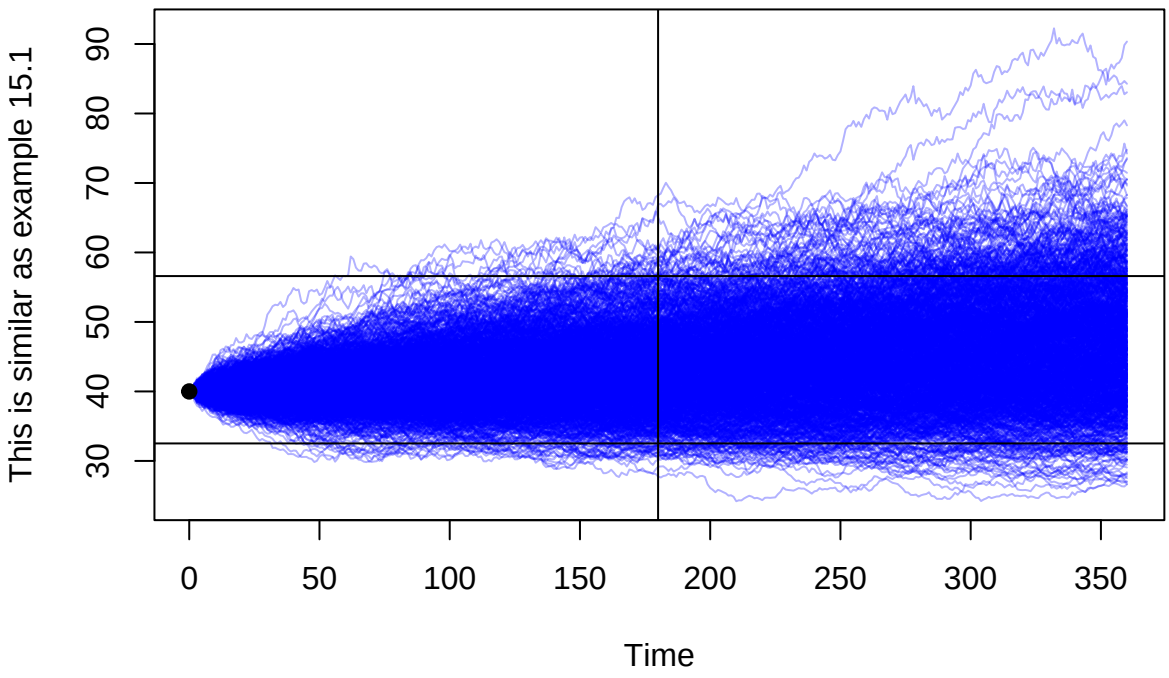


Figure 4.2.9: A geometric Brownian motion simulation: 1000 stock price paths.

Let's verify that the there is a 95% probability that the stock price in 6 months will lie between 32.55 and 56.56.

```
1 - (sum(SP_sim[180, ] < lower.ST) + sum(SP_sim[180, ] > upper.ST)) / 1000
```

```
## [1] 0.952
```

```
S0 <- 40
mu <- 0.16
sig <- 0.2
TT <- 0.5
mean.logST <- log(S0)+(mu - (sig^2 / 2)) * TT
variance.logST <- sig^2 * TT
lower.ST <- exp(mean.logST + qnorm(0.025) * variance.logST^0.5)
upper.ST <- exp(mean.logST + qnorm(0.975) * variance.logST^0.5)
```

```
lower.ST
```

```
## [1] 32.51491
```

```
upper.ST
```

```
## [1] 56.6029
```

```
S0 <- 40
mu <- 0.16
sig <- 0.2
TT <- 1
mean.logST <- log(S0)+(mu - (sig^2 / 2)) * TT
variance.logST <- sig^2 * TT
lower.ST <- exp(mean.logST + qnorm(0.05) * variance.logST^0.5)
upper.ST <- exp(mean.logST + qnorm(0.95) * variance.logST^0.5)
lower.ST
```

```
## [1] 33.11243
```

```
upper.ST
```

```
## [1] 63.93393
```

Nice.

This is how a stock price evolves according to the Black-Scholes formula.

5 VaR.

Here we develop and extend the example called "Investment in Four Stock Indices".

5.1 Prepare the data.

Load the data and prepare the database.

```
rm(list=ls())
df <- read.table("dataR.txt", sep = "", header = TRUE)
df.hoy <- read.table("hoy.txt", sep = "", header = TRUE)
library(zoo)
# Easy to remember names:
```

```
DJIA <- df[1]
FTSE100 <- df[2]
USDGBP <- df[3]
CAC40 <- df[4]
EURUSD <- df[5]
Nikkei <- df[6]
YENUSD <- df[7]
# Easy to remember names:
DJIA.hoy <- df.hoy[1]
FTSE100.hoy <- df.hoy[2]
USDGBP.hoy <- df.hoy[3]
CAC40.hoy <- df.hoy[4]
EURUSD.hoy <- df.hoy[5]
Nikkei.hoy <- df.hoy[6]
YENUSD.hoy <- df.hoy[7]
```

Replicate Table 22.2 in Hull.

```
# Adjustments for exchange rate risk. See Hull for details.
# Now the variables starting with "A" stands for "adjusted".
# We do not need to adjust DJIA since we are supposed to be in the US.
AFTSE100 <- FTSE100 * USDGBP
ACAC40 <- CAC40 / EURUSD
ANikkei <- Nikkei / YENUSD
# Inspection of the original data.
# Verify this is the same as Table 22.2 in Hull.
Table22.2 <- data.frame(DJIA, AFTSE100, ACAC40, ANikkei)
kable(head(Table22.2), caption = "US dollar equivalent of stock indices for
      historical simulation (first values).", digits = 4,
      row.names = TRUE) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.1.1: US dollar equivalent of stock indices for historical simulation (first values).

	DJIA	FTSE100	CAC40	Nikkei
1	11219.38	11131.84	6373.894	131.7744
2	11173.59	11096.28	6378.162	134.3818
3	11076.18	11185.35	6474.040	135.9433
4	11124.37	11016.71	6357.486	135.4381
5	11088.02	11040.73	6364.765	134.1003
6	11097.87	11109.50	6431.668	136.1710

Now the last part of Table 22.2.

```
kable(tail(Table22.2), caption = "US dollar equivalent of stock indices for
      historical simulation (last values).", digits = 4) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.1.2: US dollar equivalent of stock indices for historical simulation (last values).

	DJIA	FTSE100	CAC40	Nikkei
496	10609.66	8820.214	5677.136	112.1698
497	11019.69	8878.184	5689.850	109.5471
498	11388.44	9734.020	6230.006	111.6185
499	11015.69	9656.261	6181.953	113.2290
500	10825.17	9438.580	6033.935	114.2604
501	11022.06	9599.898	6200.396	112.8221

We are interested to evaluate what happened on September 26, 2008. This is why we need the real values.

```
AFTSE100.hoy <- FTSE100.hoy * USDGBP.hoy
ACAC40.hoy <- CAC40.hoy / EURUSD.hoy
ANikkei.hoy <- Nikkei.hoy / YENUSD.hoy
# Table.
Table22.2hoy <- data.frame(DJIA.hoy, AFTSE100.hoy, ACAC40.hoy, ANikkei.hoy)
kable(tail(Table22.2hoy), caption = "Here we show 501 and 502 values.",
      digits = 4) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.1.3: Here we show 501 and 502 values.

DJIA	FTSE100	CAC40	Nikkei
11022.06	9599.898	6200.396	112.8221
11143.13	9379.123	6081.478	112.1679

```
# S stands for scenarios for September 26th 2008:
SDJIA <- DJIA[501, ] * DJIA[2:501, ] / DJIA[1:500, ]
SFTSE100 <- AFTSE100[501, ] * AFTSE100[2:501, ] / AFTSE100[1:500, ]
SCAC40 <- ACAC40[501, ] * ACAC40[2:501, ] / ACAC40[1:500, ]
SNikkei <- ANikkei[501, ] * ANikkei[2:501, ] / ANikkei[1:500, ]
# Real values for September 26th 2008:
SDJIA.hoy <- DJIA.hoy[2, ] * DJIA.hoy[2, ] / DJIA.hoy[1, ]
SFTSE100.hoy <- AFTSE100.hoy[2, ] * AFTSE100.hoy[2, ] / AFTSE100.hoy[1, ]
SCAC40.hoy <- ACAC40.hoy[2, ] * ACAC40.hoy[2, ] / ACAC40.hoy[1, ]
SNikkei.hoy <- ANikkei.hoy[2, ] * ANikkei.hoy[2, ] / ANikkei.hoy[1, ]
# Investment portfolio value
p <- (4000 * DJIA[2:501, ] / DJIA[1:500, ]) +
  (3000 * AFTSE100[2:501, ] / AFTSE100[1:500, ]) +
  (1000 * ACAC40[2:501, ] / ACAC40[1:500, ]) +
  (2000 * ANikkei[2:501,] / ANikkei[1:500, ])
# Evaluated according to the real values.
p.hoy <- (4000 * DJIA.hoy[2, ] / DJIA.hoy[1, ]) +
  (3000 * AFTSE100.hoy[2, ] / AFTSE100.hoy[1, ]) +
  (1000 * ACAC40.hoy[2, ] / ACAC40.hoy[1, ]) +
  (2000 * ANikkei.hoy[2, ] / ANikkei.hoy[1, ])
# Losses l
l <- 10000 - p
l.hoy <- 10000 - p.hoy
```

Replicate Table 21.3 in Hull.

```
# Verify this is the same as Table 21.3 in Hull.
Table21.3 <- data.frame(SDJIA, SFTSE100, SCAC40, SNikkei, p, l)
kable(head(Table21.3), caption = "Scenarios generated for September 26,
```

```
      2008 (first values).", digits = 4, row.names = TRUE) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.1.4: Scenarios generated for September 26, 2008 (first values).

	SDJIA	SFTSE100	SCAC40	SNikkei	p	l
1	10977.08	9569.230	6204.547	115.0545	10014.334	-14.3338
2	10925.97	9676.957	6293.602	114.1331	10027.481	-27.4813
3	11070.01	9455.160	6088.768	112.4028	9946.736	53.2641
4	10986.04	9620.831	6207.495	111.7077	9974.861	25.1394
5	11031.85	9659.698	6265.572	114.5642	10063.635	-63.6351
6	11153.55	9660.357	6305.745	112.9480	10085.834	-85.8336

The last values of Table 21.3.

```
kable(tail(Table21.3), caption = "Scenarios generated for September 26,
      2008 (last values).", digits = 4) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.1.5: Scenarios generated for September 26, 2008 (last values).

	SDJIA	SFTSE100	SCAC40	SNikkei	p	l
495	10574.20	9495.783	6099.845	114.3359	9815.550	184.4496
496	11448.03	9662.993	6214.283	110.1841	10129.781	-129.7806
497	11390.89	10525.305	6789.019	114.9555	10555.795	-555.7954
498	10661.30	9523.211	6152.571	114.4499	9866.256	133.7444
499	10831.43	9383.488	6051.936	113.8498	9857.465	142.5355
500	11222.53	9763.974	6371.450	111.4019	10126.439	-126.4390

Nice. Values are exactly the same as in Hull.

See what really happened.

```
Table21.3hoy <- data.frame(SDJIA.hoy, SFTSE100.hoy, SCAC40.hoy, SNikkei.hoy,
                           p.hoy,l.hoy)
kable(head(Table21.3hoy), caption = "What really happened.", digits = 4,
```

```
      row.names = TRUE) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.1.6: What really happened.

	SDJIA.hoy	SFTSE100.hoy	SCAC40.hoy	SNikkei.hoy	p.hoy	l.hoy
1	11265.53	9163.425	5964.841	111.5174	9944.167	55.8325

Visually.

```
plot(l, type = "h", lwd = 2,
     xlab = "500 scenarios (sorted by historical date)",
     ylab = "Gains (-) and losses (+)", col = ifelse(l < 0, "blue", "red"))
points(501, l.hoy, pch = 19, cex = 2)
abline(0, 0)
abline(v = 0)
lines(seq(0, max(l), length.out = 500), lty = 2)
lines(seq(0, min(l), length.out = 500), lty = 2)
abline(h = 253.385, lwd = 2, col = "green")
legend("bottomleft", legend = c("Gains", "Losses", "Historic VaR (253.385)"),
       col = c("blue", "red", "green"), lty = 1, bg = "white", lwd = 2)
```

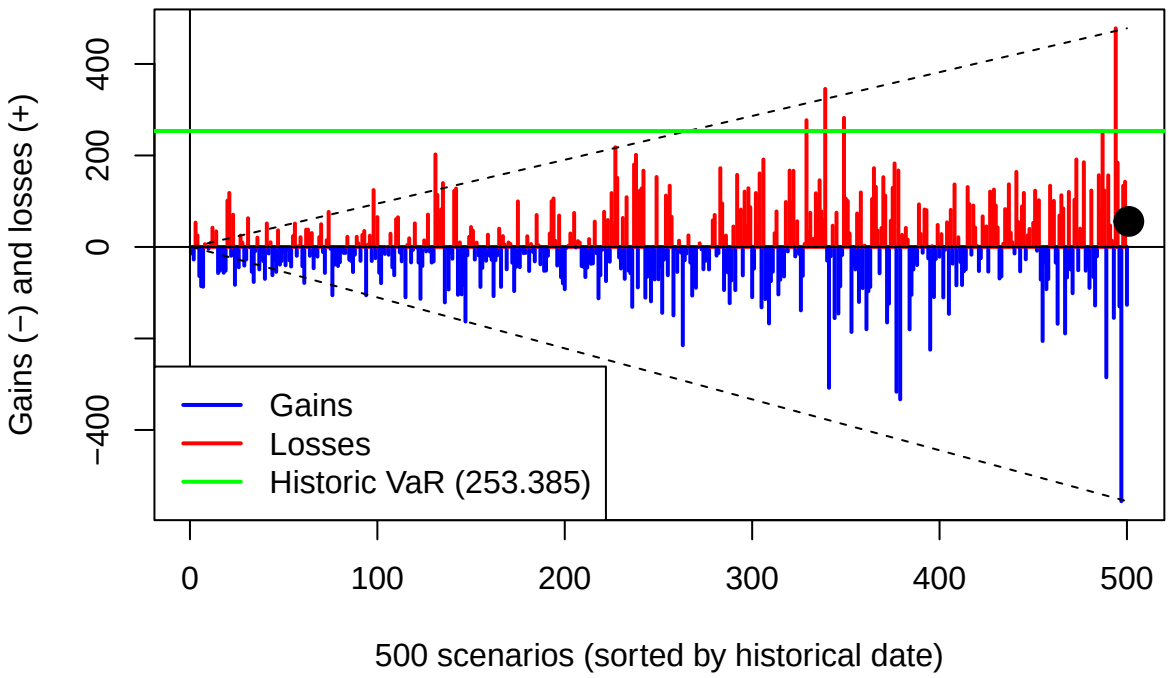


Figure 5.1.1: Today is September 25, 2008. What is the worst that can happen to my portfolio tomorrow?

Now, a histogram of losses.

```
hist(l, 15, xlab = "Losses (+) and gains (-)", main = NULL)
points(l.hoy, 1, pch = 19, cex = 2)
abline(v = 253.385, col = "red", lwd = 2)
legend("topleft", legend = c("Historic VaR (253.385)"), col = "red",
       bg = "white", lwd = 2)
```

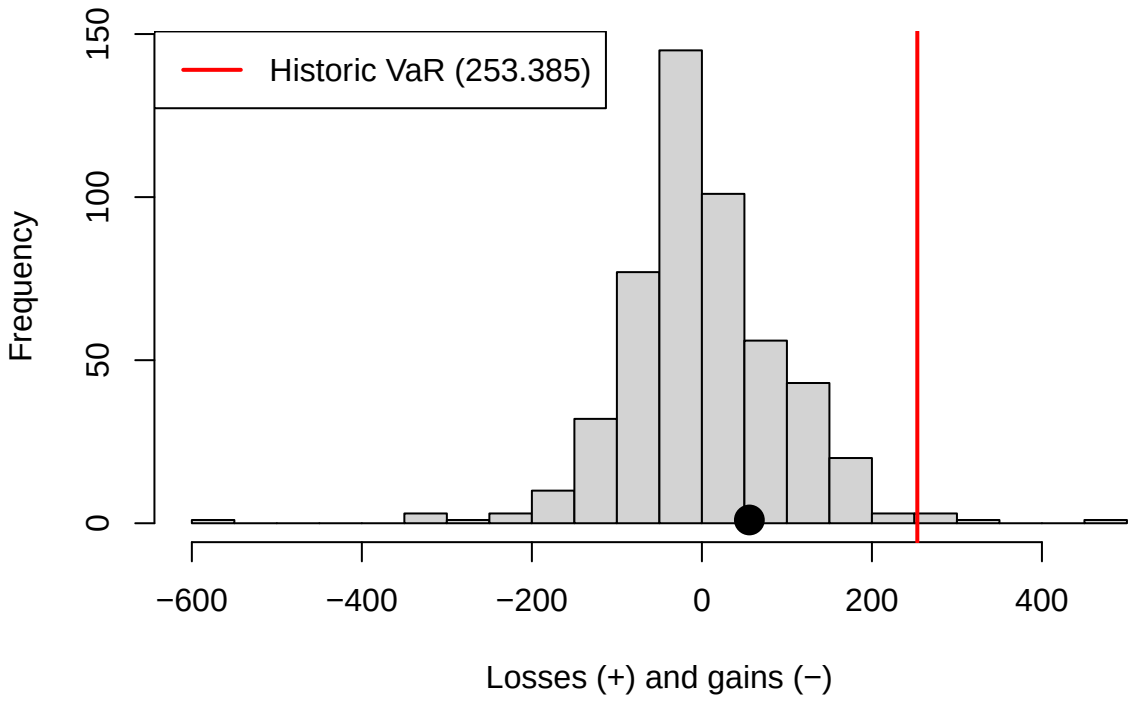


Figure 5.1.2: Histogram of losses for the scenarios considered between September 25 and 26, 2008: a histogram.

The point refers to what really happened, and the red line is the VaR following a historic approach. Now, the same information but now using a density plot.

```
densi <- density(l)
plot(densi, main = "", xlab = "Losses (+) and gains (-)")
polygon(densi, col = "grey", border = "black")
points(l.hoy, 0, pch = 19, cex = 2)
abline(v = 253.385, col = "red", lwd = 2)
legend("topleft", legend = c("Historic VaR (253.385)"), col = "red",
       bg = "white", lwd = 2)
```

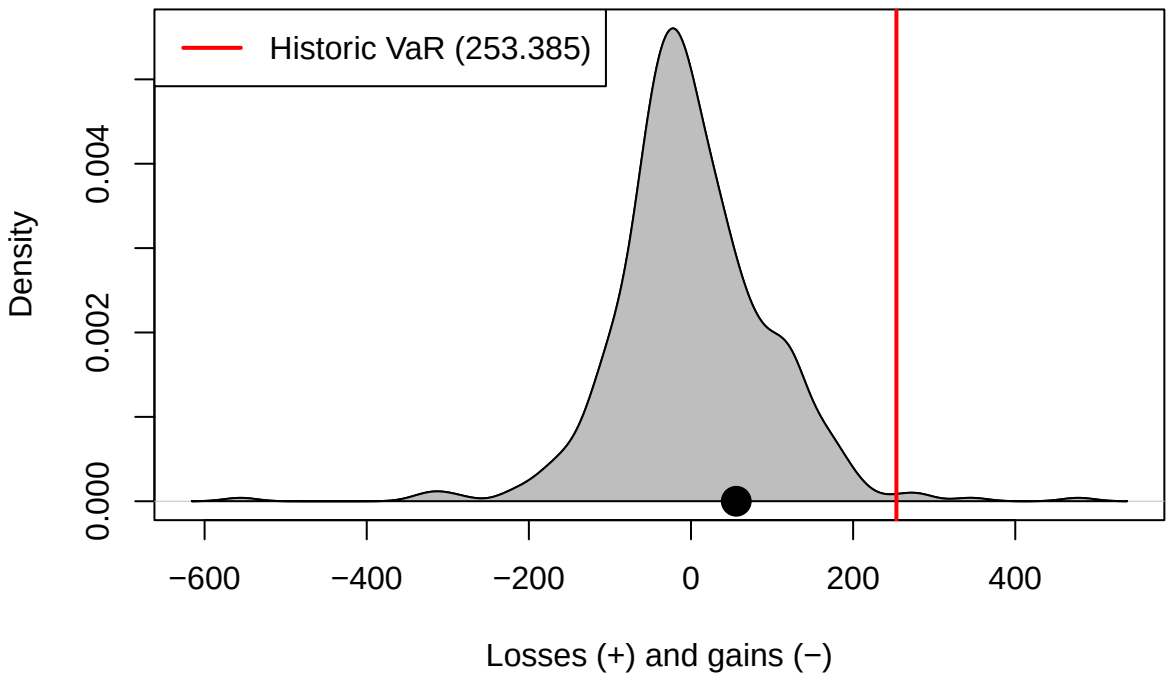


Figure 5.1.3: Histogram of losses for the scenarios considered between September 25 and 26, 2008: a density plot.

Looks nice.

Here is a histogram that looks like the one in Hull.

```
hist(l, 15, axes = F, lty = "blank", col = "grey", main = "",
     xlab = "Losses (+) and gains (-)")
axis(1, pos = 0)
axis(2, pos = 0, las = 1)
```

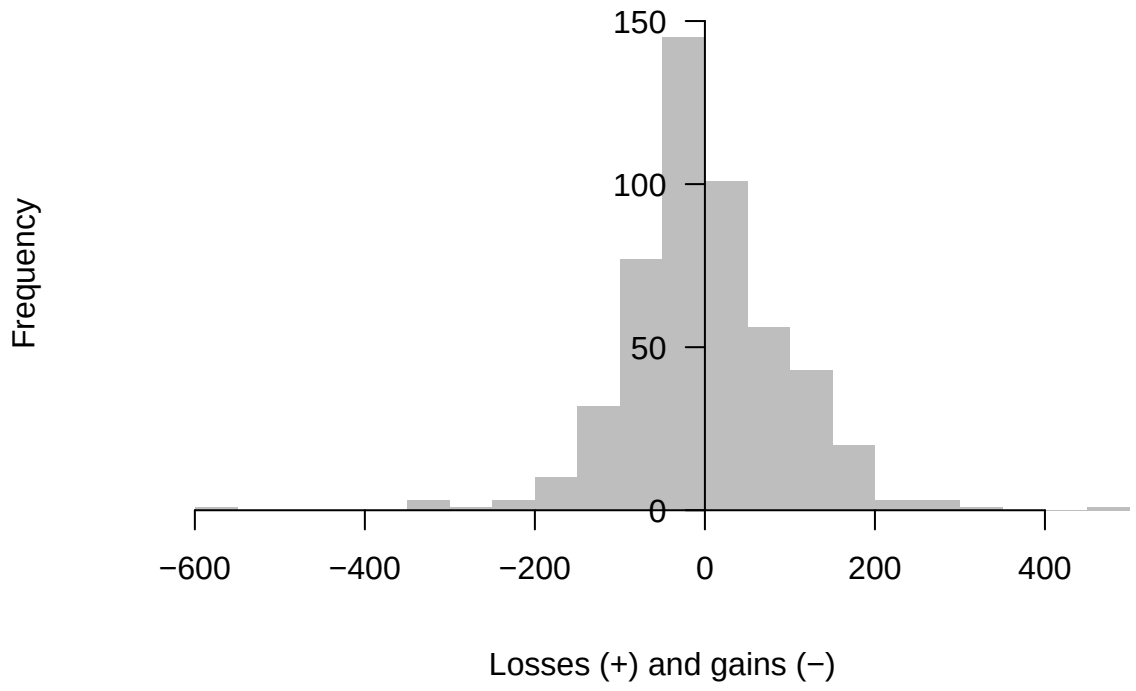


Figure 5.1.4: Figure 21.3. Histogram of losses for the scenarios considered between September 25 and 26, 2008.

A table for the losses ranked.

```
Table22.4 <- data.frame(scenario.number = order(l, decreasing = TRUE),
                        `loss in positive values` = l[order(l, decreasing = TRUE)])
kable(head(Table22.4), caption = "Losses ranked from highest to lowest for
      500 scenarios (first values).", digits = 4, row.names = TRUE) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.1.7: Losses ranked from highest to lowest for 500 scenarios (first values).

	scenario.number	loss.in.positive.values
1	494	477.8410
2	339	345.4351
3	349	282.2038
4	329	277.0413
5	487	253.3850
6	227	217.9740

And the last values (not shown in Hull).

```
kable(tail(Table22.4), caption = "Losses ranked from highest to lowest for
      500 scenarios (last values).", digits = 4) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.1.8: Losses ranked from highest to lowest for 500 scenarios (last values).

	scenario.number	loss.in.positive.values
495	395	-224.5146
496	489	-284.9247
497	341	-307.9301
498	377	-316.4893
499	379	-333.0218
500	497	-555.7954

Note that one of the highest gains happen in scenario 497, almost at the end.

```
# Results
(VaR.hist <- Table22.4[5, 2])
```

```
## [1] 253.385
```

The VaR according to the historic approach is the highest 5th loss because the top 1% of 500 scenarios is the 5th scenario. We can view this in a barplot.

```
barplot(l[order(l, decreasing = TRUE)],
        names.arg = order(l, decreasing = TRUE), ylim = c(0, 500),
        xlim = c(0, 20), las = 2, cex.names = 1, ylab = "Loss ($000s)",
        col = "red", xlab = "Scenario number (ranked)")
abline(h = 253.385, lwd = 3)
abline(h = l.hoy, lwd = 3, col = "green")
legend("topright", legend = c("Historic VaR (253.385)",
       "What really happened"), col = c("black", "green"),
       bg = "white", lwd = 2)
```

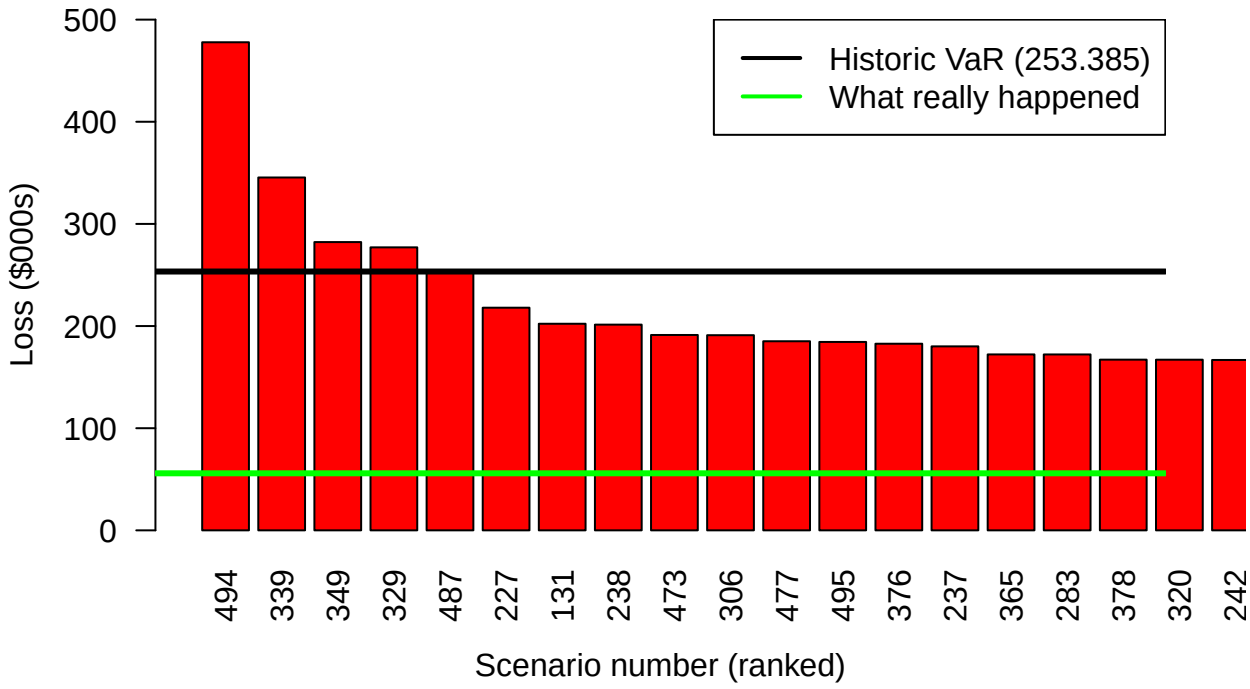


Figure 5.1.5: Extract of table 22.4 in a graph VaR historical approach 1 percent 253.385 What really happen in green.

```
quantile(l, 0.99)
```

```
##       99%
```

```
## 218.3281
```

```
barplot(quantile(l), ylim = c(min(l), max(l)),
        col = ifelse(quantile(l) < 0, "blue", "red"),
        xlab = "Quantiles", ylab = "Gains (-) and losses (+)")
```

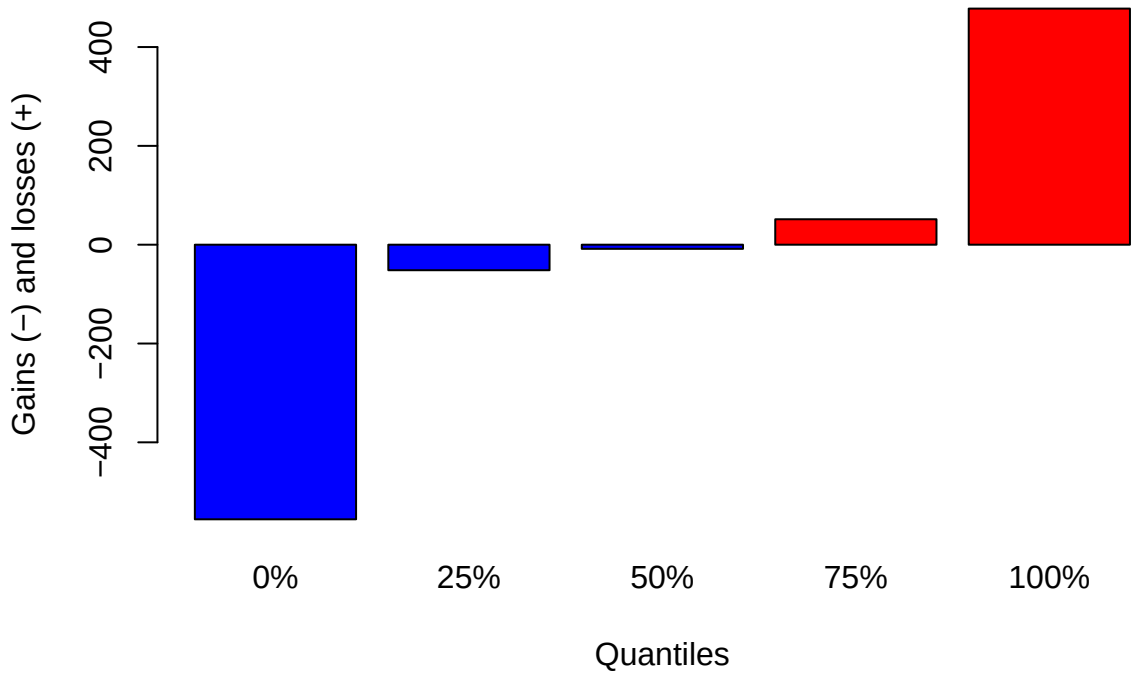


Figure 5.1.6: Ilustration of portfolio losses in quantiles.

```
# quantile(l,seq(from=0.8,to=0.99,by=0.001))
```

```
# Results
VaR.hist.quantile <- quantile(l, 0.99)
kable(VaR.hist.quantile, caption = "Quantile.", digits = 4) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.1.9: Quantile.

	x
99%	218.3281

```
plot(seq(from = 0.902, to = 1, by = 0.002),
     quantile(l, seq(from = 0.902, to = 1, by = 0.002)), cex = 1.5,
     xlab = "Confidence level (vertical line at 99%)", ylab = "VaR")
abline(v = 0.99, lty = 2, lwd = 2)
abline(h = quantile(l, 0.99), col = "blue", lty = 2, lwd = 2)
abline(h = Table22.4[5, 2], col = "red", lty = 2, lwd = 2)
legend("topleft", legend = c("99% level", "Historic VaR (253.385)",
                             "99% quantile VaR (218.3281)"),
       col = c("black", "red", "blue"), lty = 2, bg = "white", lwd = 2)
```

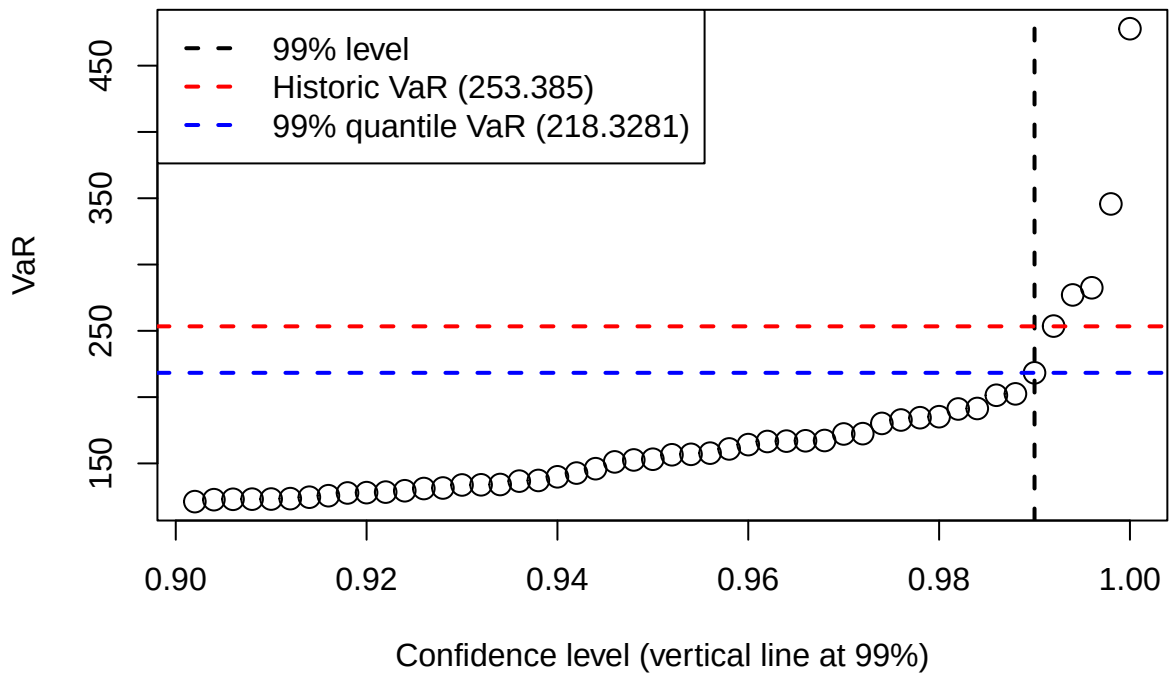


Figure 5.1.7: If we use a quantile approach.

Summary of results.

```
Results <- data.frame(VaR.method = c("VaR.hist","VaR.hist.quantile"),
                      value = c(VaR.hist,VaR.hist.quantile))
kable(Results, caption = "Results.", digits = 4) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.1.10: Results.

	VaR.method	value
	VaR.hist	253.3850
99%	VaR.hist.quantile	218.3281

5.2 Model building approach.

Let's calculate more VaR with different methods.

```
# Model building example -- equally weighted
# Compute returns (this code should be simplified)
RDJIA <- (DJIA[2:501, ] - DJIA[1:500, ]) / DJIA[1:500, ]
RAFTSE100 <- (AFTSE100[2:501, ] - AFTSE100[1:500, ]) / AFTSE100[1:500, ]
RACAC40 <- (ACAC40[2:501, ] - ACAC40[1:500, ]) / ACAC40[1:500, ]
RANikkei <- (ANikkei[2:501, ] - ANikkei[1:500, ]) / ANikkei[1:500, ]
# Gather returns.
R <- merge.zoo(RDJIA, RAFTSE100, RACAC40, RANikkei)
# Compute returns for "today".
RDJIA.hoy <- (DJIA.hoy[2, ] - DJIA.hoy[1, ]) / DJIA.hoy[1, ]
RAFTSE100.hoy <- (AFTSE100.hoy[2, ] - AFTSE100.hoy[1, ]) / AFTSE100.hoy[1, ]
RACAC40.hoy <- (ACAC40.hoy[2, ] - ACAC40.hoy[1, ]) / ACAC40.hoy[1, ]
RANikkei.hoy <- (ANikkei.hoy[2, ] - ANikkei.hoy[1, ]) / ANikkei.hoy[1, ]
# Gather returns.
R.hoy <- merge.zoo(RDJIA.hoy, RAFTSE100.hoy, RACAC40.hoy, RANikkei.hoy)
```

Replicate Hull's table 23.5.

```
# correlation
Rcor <- cor(R)
Table23.5 <- Rcor
kable(Table23.5, caption = "Correlation matrix on September 25, 2008,
```

```
      calculated by giving equal weight to the last 500 daily returns.",
      digits = 4) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.2.1: Correlation matrix on September 25, 2008, calculated by giving equal weight to the last 500 daily returns.

	RDJIA	RAFTSE100	RACAC40	RANikkei
RDJIA	1.0000	0.4891	0.4957	-0.0619
RAFTSE100	0.4891	1.0000	0.9181	0.2009
RACAC40	0.4957	0.9181	1.0000	0.2110
RANikkei	-0.0619	0.2009	0.2110	1.0000

Replicate Hull's table 23.6.

```
# covariance
Rcov <- cov(R)
Table23.6 <- Rcov
kable(Table23.6, caption = "Covariance matrix on September 25, 2008,
      calculated by giving equal weight to the last 500 daily returns.",
      digits = 8) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.2.2: Covariance matrix on September 25, 2008, calculated by giving equal weight to the last 500 daily returns.

	RDJIA	RAFTSE100	RACAC40	RANikkei
RDJIA	0.00012295	0.00007697	0.00007683	-0.00000949
RAFTSE100	0.00007697	0.00020140	0.00018211	0.00003944
RACAC40	0.00007683	0.00018211	0.00019535	0.00004078
RANikkei	-0.00000949	0.00003944	0.00004078	0.00019131

This covariance matrix is quite similar to the one in Hull. We are not taking round values this explains minor differences.

This covariance matrix gives the variance of the portfolio losses ($000s) as

```
# standard deviation
Rsd <- apply(R, 2, sd)
# allocation of the 10,000 usd
alpha <- c(4000, 3000, 1000, 2000)
# Answers can be compared in page 538 (Chapter 23)
(Pvar <- t(alpha) %*% Rcov %*% alpha) # equation in page 506 (chapter 22)
```

```
##          [,1]
## [1,] 8779.392
```

This is slightly different as in Hull: 8,761.833. The standard deviation is the square root of this

```
(Psd <- Pvar^0.5)
```

```
##          [,1]
## [1,] 93.69841
```

As expected, this is a bit different as in Hull (93.60). The one-day 99% VaR in is therefore

```
(VaR.ew <- qnorm(0.99, 0, 1) * Psd) # I did not use the simplification 2.33
```

```
##          [,1]
## [1,] 217.9751
```

In Hull, this value is 217,757, which compares with 253,385, calculated using the historical simulation approach in Section 22.2.

```
Results <- data.frame(VaR.method = c("VaR.hist", "VaR.hist.quantile",
                                     "VaR.ew"),
                      value = c(VaR.hist, VaR.hist.quantile, VaR.ew))
kable(Results, caption = "Summary of results.", digits = 4) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.2.3: Summary of results.

VaR.method	value
VaR.hist	253.3850
VaR.hist.quantile	218.3281
VaR.ew	217.9751

We are interested to compare the equally weighted approach with the EWMA (exponentially weighted moving average method). In fact, there are two EWMA versions here, one was taken from a different author, and the other is mine (Martín).

First the other author's approach to calculate an EWMA covariance considering $\lambda = 0.94$.

```
# Model building example -- EWMA
covEWMA <-
  function(factors, lambda = 0.96, return.cor = FALSE) {
    factor.names  = colnames(factors)
    t.factor      = nrow(factors)
    k.factor      = ncol(factors)
    factors       = as.matrix(factors)
    t.names       = rownames(factors)

    cov.f.ewma = array(,c(t.factor, k.factor, k.factor))
    cov.f = var(factors)  # unconditional variance as EWMA at time = 0
    FF = (factors[1, ] - mean(factors)) %*% t(factors[1,] - mean(factors))
    cov.f.ewma[1,,] = (1 - lambda) * FF  + lambda * cov.f
    for (i in 2:t.factor) {
    FF = (factors[i, ] - mean(factors)) %*% t(factors[i, ] - mean(factors))
    cov.f.ewma[i,,] = (1 - lambda) * FF  + lambda * cov.f.ewma[(i-1),,]
    }
    dimnames(cov.f.ewma) = list(t.names, factor.names, factor.names)

    if(return.cor) {
      cor.f.ewma = cov.f.ewma
      for (i in 1:dim(cor.f.ewma)[1]) {
        cor.f.ewma[i, , ] = cov2cor(cov.f.ewma[i, ,])
      }
      return(cor.f.ewma)
    } else {
      return(cov.f.ewma)
    }
  }
```

Let's implement this approach to our problem.

```
Rdf <- as.data.frame(R)
C <- covEWMA(Rdf, lambda = 0.94)[500,,] # Table 23.7
Table23.7 <- C
kable(Table23.7, caption = "Covariance matrix on September 25, 2008,
      calculated using the EWMA (other author's approach).", digits = 8) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.2.4: Covariance matrix on September 25, 2008, calculated using the EWMA (other author's approach).

	RDJIA	RAFTSE100	RACAC40	RANikkei
RDJIA	0.00047986	0.00043008	0.00042553	-0.00003989
RAFTSE100	0.00043008	0.00103126	0.00096296	0.00020934
RACAC40	0.00042553	0.00096296	0.00095350	0.00016796
RANikkei	-0.00003989	0.00020934	0.00016796	0.00025383

The variance of portfolio losses is:

```
(Pvar_EWMA <- t(alpha) %*% C %*% alpha)
```

```
##          [,1]
## [1,] 40977.52
```

In Hull, this value is 40,995.765. The standard deviation is the square root of this, or:

```
(Psd_EWMA <- Pvar_EWMA^0.5)
```

```
##          [,1]
## [1,] 202.4291
```

In Hull, this value is 202.474. The one-day 99% VaR is therefore:

```
(VaR.ewma <- qnorm(0.99, 0, 1) * Psd_EWMA)
```

```
##          [,1]
## [1,] 470.9204
```

In Hull, this value is 471,025, over twice as high as the value given when returns are equally weighted.

```
Results  <-  data.frame(VaR.method=c("VaR.hist", "VaR.hist.quantile",
                                     "VaR.ew","VaR.ewma"),
                    value = c(VaR.hist, VaR.hist.quantile, VaR.ew,VaR.ewma))
kable(Results, caption = "Summary of results.", digits = 4) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.2.5: Summary of results.

VaR.method	value
VaR.hist	253.3850
VaR.hist.quantile	218.3281
VaR.ew	217.9751
VaR.ewma	470.9204

Let's replicate Table 23.8.

```
#Table 23.8
Table23.8 <- data.frame(Rsd * 100, diag(C)^0.5 * 100)
colnames(Table23.8) <- c("Equal weighting", "EWMA")
rownames(Table23.8) <- c("DJIA", "FTSE 100", "CAC 40", "Nikkei 225")
kable(t(Table23.8), caption = "Volatilities % per day using equal
      weighting and EWMA (other author approach).", digits = 4) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.2.6: Volatilities weighting and EWMA (other author approach).

	DJIA	FTSE 100	CAC 40	Nikkei 225
Equal weighting	1.1088	1.4191	1.3977	1.3832
EWMA	2.1906	3.2113	3.0879	1.5932

The estimated daily standard deviations are much higher when EWMA is used than when data are equally weighted. This is because volatilities were much higher during the period immediately preceding September 25, 2008, than during the rest of the 500 days covered by the data.

Now EWMA with my own approach.

```
# Model building example -- EWMA -- INEFFICIENT BUT ACCURATE.
# My approach based on Hull.
lambda = 0.94
Var.R.EWMA <- matrix(0, 501, 4)
for(i in 1:501) {
  if(i==1) {
    Var.R.EWMA[, (1:4)] <- Rsd^2
    } else if (i > 1) {
      Var.R.EWMA[i,] <- (Var.R.EWMA[(i - 1), (1:4)] * lambda) +
        (R[(i - 1), (1:4)]^2 * (1 - lambda))}
    }

Cov.DF.EWMA <- NULL
Cov.DC.EWMA <- NULL
Cov.DN.EWMA <- NULL
Cov.FC.EWMA <- NULL
Cov.FN.EWMA <- NULL
Cov.CN.EWMA <- NULL

for(i in 1:501){
if(i==1){
  Cov.DF.EWMA[i] <- cov(RDJIA, RAFTSE100)
  Cov.DC.EWMA[i] <- cov(RDJIA, RACAC40)
  Cov.DN.EWMA[i] <- cov(RDJIA, RANikkei)
  Cov.FC.EWMA[i] <- cov(RAFTSE100, RACAC40)
  Cov.FN.EWMA[i] <- cov(RAFTSE100, RANikkei)
  Cov.CN.EWMA[i] <- cov(RACAC40, RANikkei)
  } else if (i > 1) {
    Cov.DF.EWMA[i] <- (Cov.DF.EWMA[i - 1] * lambda) +
      (RAFTSE100[i - 1] * RDJIA[i - 1] * (1 - lambda))
    Cov.DC.EWMA[i] <- (Cov.DC.EWMA[i - 1] * lambda) +
      (RACAC40[i - 1] * RDJIA[i - 1] * (1 - lambda))
    Cov.DN.EWMA[i] <- (Cov.DN.EWMA[i - 1] * lambda) +
      (RANikkei[i - 1] * RDJIA[i - 1] * (1 - lambda))
    Cov.FC.EWMA[i] <- (Cov.FC.EWMA[i - 1] * lambda) +
      (RACAC40[i - 1] * RAFTSE100[i - 1] * (1 - lambda))
```

```
    Cov.FN.EWMA[i] <- (Cov.FN.EWMA[i - 1] * lambda) +
      (RANikkei[i-1] * RAFTSE100[i-1] * (1 - lambda))
    Cov.CN.EWMA[i] <- (Cov.CN.EWMA[i - 1] * lambda) +
      (RANikkei[i - 1] * RACAC40[i - 1] * (1 - lambda))
  }
  }

C11 <- c(Var.R.EWMA[501, 1])
C22 <- c(Var.R.EWMA[501, 2])
C33 <- c(Var.R.EWMA[501, 3])
C44 <- c(Var.R.EWMA[501, 4])
C12 <- c(Cov.DF.EWMA[501])
C13 <- c(Cov.DC.EWMA[501])
C14 <- c(Cov.DN.EWMA[501])
C23 <- c(Cov.FC.EWMA[501])
C24 <- c(Cov.FN.EWMA[501])
C34 <- c(Cov.CN.EWMA[501])
VCV <- c(C11 ,C12, C13, C14, C12, C22, C23, C24,C13, C23,
         C33, C34, C14, C24, C34, C44)
VCV <- matrix(VCV, 4, 4)
```

The variance of portfolio losses is now:

```
(Pvar_EWMA1 <- t(alpha) %*% VCV %*% alpha)
```

```
##          [,1]
## [1,] 40995.76
```

Which is basically the same as in Hull: 40,995.765. The standard deviation is the square root of this, or:

```
(Psd_EWMA1 <- Pvar_EWMA1^0.5)
```

```
##          [,1]
## [1,] 202.4741
```

The same as in Hull: 202.474. The one-day 99% VaR is therefore:

```
# My approach is longer but more accurate
(VaR.ewma.martin <- qnorm(0.99, 0, 1) * Psd_EWMA1)
```

```
##           [,1]
## [1,] 471.0252
```

The same as in Hull: 471.0252. Let's report all results so far.

```
Results <- data.frame(VaR.method=
                        c("VaR.hist",
                          "VaR.hist.quantile",
                          "VaR.ew",
                          "VaR.ewma",
                          "VaR.ewma.martin"),
                      value=c(VaR.hist,
                              VaR.hist.quantile,
                              VaR.ew,
                              VaR.ewma,
                              VaR.ewma.martin))
kable(Results, caption = "Summary of results.", digits = 4) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.2.7: Summary of results.

VaR.method	value
VaR.hist	253.3850
VaR.hist.quantile	218.3281
VaR.ew	217.9751
VaR.ewma	470.9204
VaR.ewma.martin	471.0252

Let's replicate an extended Table 23.8.

```
#Table 23.8
Table23.8 <- data.frame(Rsd * 100, diag(C)^0.5 * 100, diag(VCV)^0.5 * 100)
colnames(Table23.8) <- c("Equal weighting", "EWMA", "EWMA Martín")
rownames(Table23.8) <- c("DJIA", "FTSE 100", "CAC 40", "Nikkei 225")
kable(t(Table23.8), caption = "Volatilities % per day using equal
      weighting and EWMA (other author approach).", digits = 4) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.2.8: Volatilities weighting and EWMA (other author approach).

	DJIA	FTSE 100	CAC 40	Nikkei 225
Equal weighting	1.1088	1.4191	1.3977	1.3832
EWMA	2.1906	3.2113	3.0879	1.5932
EWMA Martín	2.1911	3.2115	3.0880	1.5941

Again: The estimated daily standard deviations are much higher when EWMA is used than when data are equally weighted. This is because volatilities were much higher during the period immediately preceding September 25, 2008, than during the rest of the 500 days covered by the data.

5.3 Optimal allocation of resources and VaR.

The current portfolio weights are:

```
W_hull <- alpha / 10000
W_hullt <- data.frame(W_hull)
rownames(W_hullt) <- c("DJIA", "FTSE 100", "CAC 40", "Nikkei 225")
kable(t(W_hullt), caption = "Original (and presumible sub-optimal) Hull's
      portfolio weights.", digits = 4) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.3.1: Original (and presumible sub-optimal) Hull's portfolio weights.

	DJIA	FTSE 100	CAC 40	Nikkei 225
W_hull	0.4	0.3	0.1	0.2

Let's calculate optimal weights.

```
# Further extension: propose an optimal allocation of resources
R <- R * 100
R.hoy <- R.hoy * 100
stocks<- c("DJIA", "FTSE100", "CAC40", "Nikkei")
sigma <- var(R)
sd <- diag(sigma)^0.5
E <- colMeans(R)
```

```
E.hoy <- colMeans(R.hoy)
ones <- abs(E / E)
a <- c(t(ones) %*% solve(sigma) %*% ones)
b <- c(t(ones) %*% solve(sigma) %*% E)
c <- c(t(E) %*% solve(sigma) %*% E)
d <- c(a * c - (b^2))
g <- c(solve(sigma) %*% (c * ones - b * E) / d)
h <- c(solve(sigma) %*% (a * E - b * ones) / d)
ER <- seq(from = -0.3, to = 0.3, by = 0.0001)
S <- ER
W <- matrix(0, nrow = length(ER), ncol = length(stocks))
for(i in 1:length(ER)){
W[i,] <- g + h * ER[i]
ER[i] <- W[i,] %*% E
S[i] <- (t(W[i,]) %*% sigma %*% W[i, ])^0.5
}
W_min <- (solve(sigma) %*% ones) / a
R_min <- c(t(W_min) %*% E)
R_min.realized <- c(t(W_min) %*% E.hoy)
S_min <- c(t(W_min) %*% sigma %*% W_min)^0.5
# Allocation to mimic the return of the CAC40 but with lower risk
W.cac40 <- g + h * E[3]
R_cac40 <- c(t(W.cac40) %*% E)
R_cac40.realized <- c(t(W.cac40) %*% E.hoy)
S_cac40 <- c(t(W.cac40) %*% sigma %*% W.cac40)^0.5
R_hull.realized = c(t(W_hull) %*% E.hoy)
```

Now we have four different portfolio weights alternatives. Two of them are optimal. Can we reduce the VaR by using optimal weights?

```
W <- data.frame(W_hull, W_min, W.cac40, c(0.25,0.25,0.25,0.25))
rownames(W) <- c("DJIA", "FTSE 100", "CAC 40", "Nikkei 225")
colnames(W) <- c("Hull (original weights)", "minimum variance (Martín)", "CAC 40 target
kable(t(W), caption = "Four different weights alternatives.",
      digits = 4) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.3.2: Four different weights alternatives.

	DJIA	FTSE 100	CAC 40	Nikkei 225
Hull (original weights)	0.4000	0.3000	0.1000	0.2000
minimum variance (Martín)	0.5592	0.0379	0.0228	0.3800
CAC 40 target return	0.6075	-0.4034	0.4598	0.3361
naive (1/N)	0.2500	0.2500	0.2500	0.2500

See the results graphically.

```
W_hull <- alpha / 10000
R_hull <- c(t(W_hull) %*% E)
S_hull <- c(t(W_hull) %*% sigma %*% W_hull)^0.5

plot(S, ER, type ="l", lwd = 2, xlim = c(0.7, 1.6), ylim = c(-0.022, 0.005),
     ylab = "Expected return", xlab = "Standard deviation")
points(diag(sigma)^0.5, E)
abline(0, R_hull / S_hull, lty = 3, col = "blue")
abline(0, R_min / S_min, lty = 3, col = "red")
points(S_min, R_min, col = "red", pch = 19, cex = 2)
abline(0, 0, lty = 3)
text(S_min, R_min, "Martin's
     alpha", pos = 2, cex = 0.8)
text(sd, E, stocks, adj = -0.5, cex = 0.8)
points(S_hull, R_hull, col = "blue", pch = 19, cex = 2)
text(S_hull, R_hull, "Hull's alpha", adj = -0.3, cex = 0.8, pos = 1)
legend("bottomleft", legend = c("DJ=0.4, FTSE=0.3, CAC=0.1, Nikkei=0.2",
                                "DJ=0.56, FTSE=0.038, CAC=0.022, Nikkei=0.38"),
       col = c("blue","red"), pch = 19, bg = "white", cex = 0.8)
```

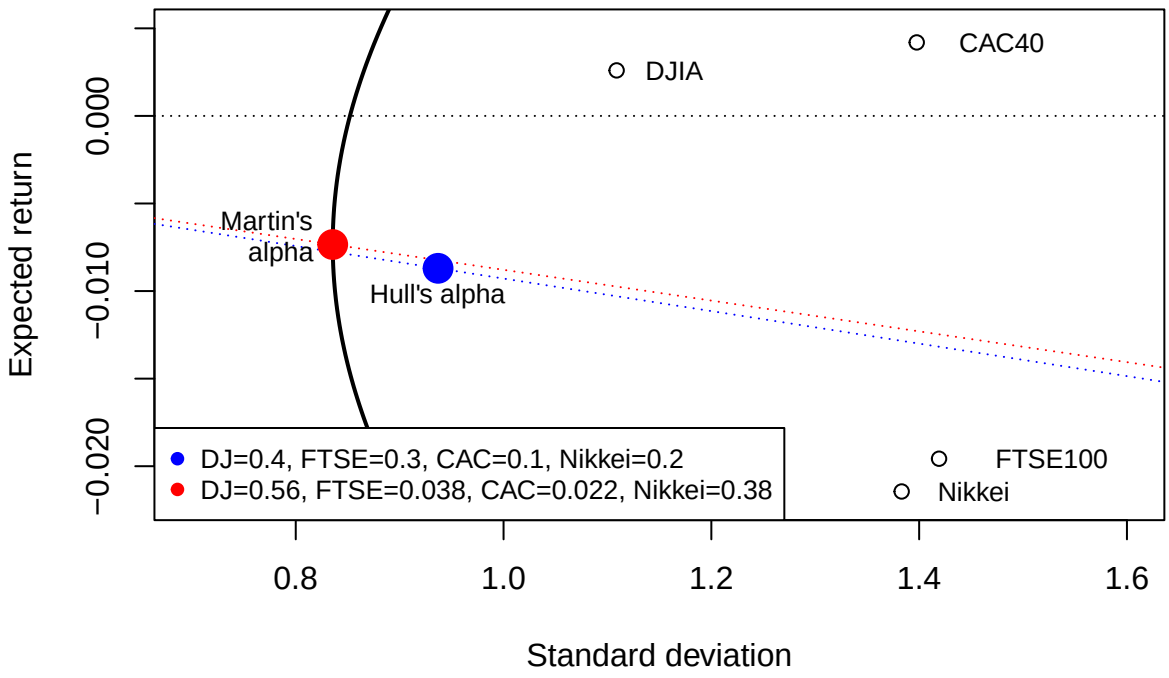


Figure 5.3.1: Hull's allocation is sub-optimal, the red one is a better alternative, although with negative expected return.

The figure suggest that the minimum variance alternative reduce the original portfolio risk. In fact, the minimum variance portfolio has a higher return and lower risk. This representation is an estimate of the future. We can evaluate what really happened by using the real returns.

```
(R_min.realized <- c(t(W_min) %*% E.hoy))
```

```
## [1] 0.2629549
```

```
(R_hull.realized = c(t(W_hull) %*% E.hoy))
```

```
## [1] -0.5583249
```

```
W_hull <- alpha / 10000
R_hull <- c(t(W_hull) %*% E)
S_hull <- c(t(W_hull) %*% sigma %*% W_hull)^0.5
```

```
W_1N <- c(0.25,0.25,0.25,0.25)
R_1N <- c(t(W_1N) %*% E)
S_1N <- c(t(W_1N) %*% sigma %*% W_1N)^0.5

plot(S, ER, type ="l", lwd = 2, xlim = c(0.7, 1.6), ylim = c(-0.022, 0.005),
     ylab = "Expected return", xlab = "Standard deviation")
points(diag(sigma)^0.5, E)
abline(0, R_hull / S_hull, lty = 3, col = "blue")
abline(0, R_min / S_min, lty = 3, col = "red")
abline(0, R_1N / S_1N, lty = 3, col = "purple")
points(S_min, R_min, col = "red", pch = 19, cex = 2)
points(S_1N, R_1N, col = "purple", pch = 19, cex = 2)
abline(0, 0, lty = 3)
text(S_min, R_min, "Martin's
     alpha", pos = 2, cex = 0.8)
text(sd, E, stocks, adj = -0.5, cex = 0.8)
points(S_hull, R_hull, col = "blue", pch = 19, cex = 2)
text(S_hull, R_hull, "Hull's alpha", adj = -0.3, cex = 0.8, pos = 1)
text(S_1N, R_1N, "1/N alpha", adj = -0.3, cex = 0.8, pos = 3)
legend("bottomleft",
       legend = c("Hull: DJ=0.4, FTSE=0.3, CAC=0.1, Nikkei=0.2",
"Martin: DJ=0.56, FTSE=0.038, CAC=0.022, Nikkei=0.38",
"1/N: DJ=0.25, FTSE=0.25, CAC=0.25, Nikkei=0.25"),
       col = c("blue","red", "purple"), pch = 19, bg = "white", cex = 0.8)
```

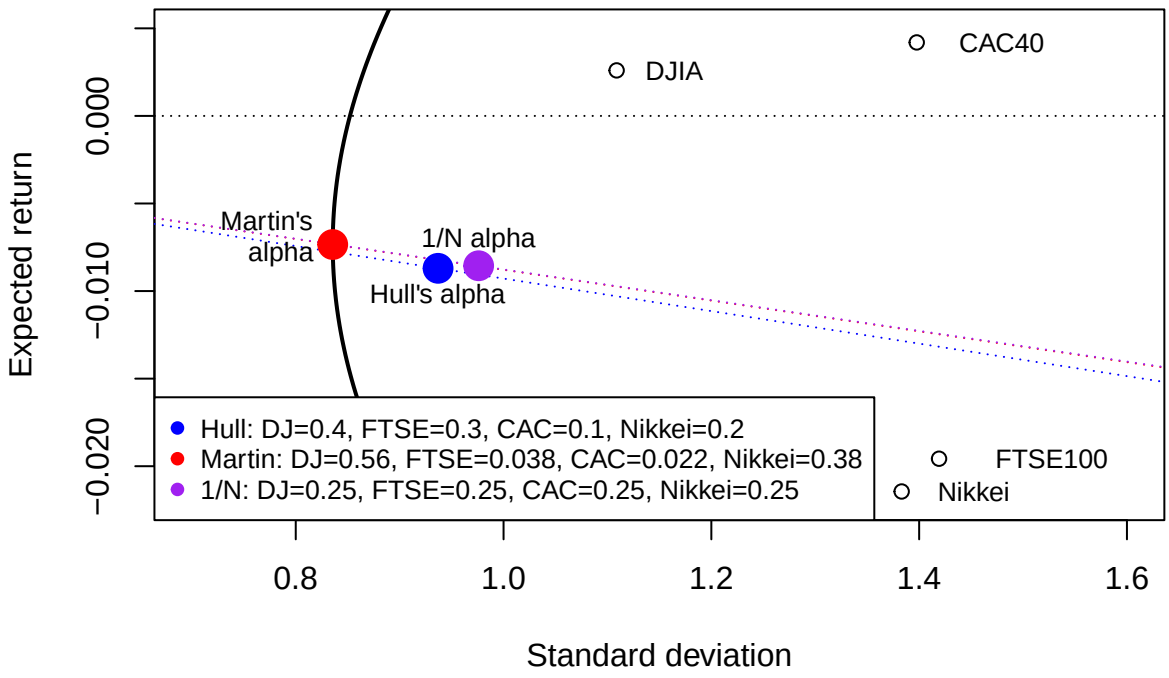


Figure 5.3.2: Hull's allocation is sub-optimal, the red one is a better alternative, although with negative expected return.

Hull's sub-optimal portfolio allocation turns out to be problematic. His recommendation would lead to a loss and the optimal minimum variance would lead to a positive return. Nice.

```
set.seed <- 1
M <- rnorm(10000, R_min, S_min)
hist(M, 100, xlim = c(-4, 4), ylim = c(0, 450), main = "")
abline(v = R_min.realized, col = "blue", lwd = 3)
abline(v = quantile(M, 0.025), lty = 2)
abline(v = quantile(M, 0.975), lty = 2)
```

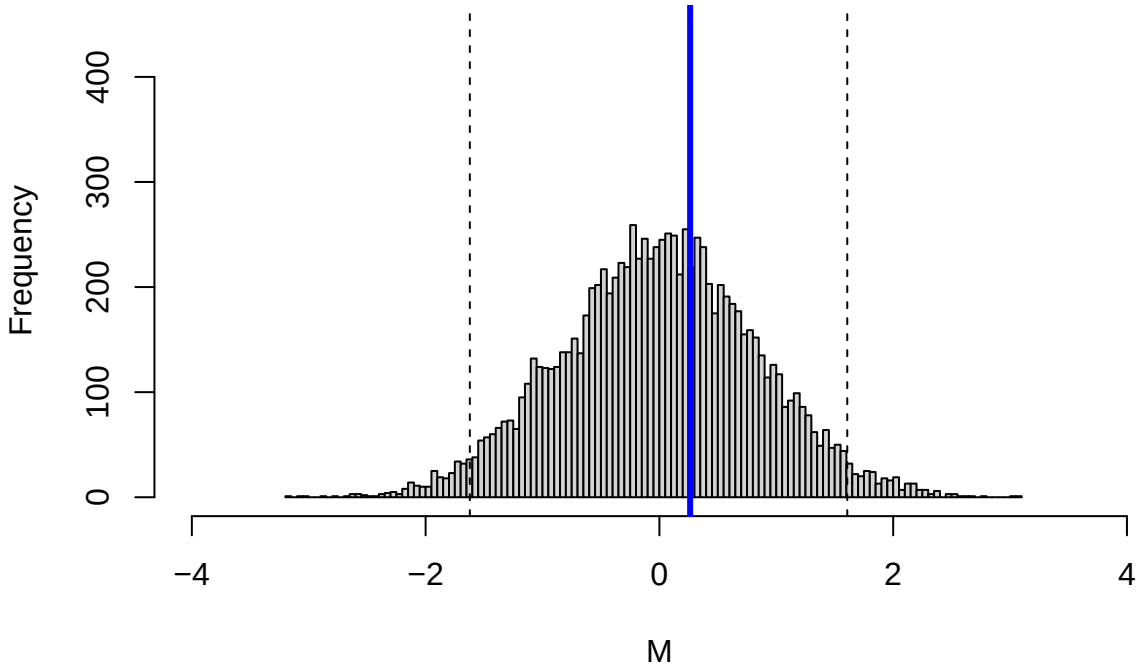


Figure 5.3.3: Martin recommendation and realized outcome in blue.

```
set.seed = 1
H = rnorm(10000, R_hull, S_hull)
hist(H,100, xlim = c(-4, 4), ylim = c(0, 450), main = "")
abline(v = R_hull.realized, col = "blue", lwd = 3)
abline(v = quantile(H, 0.025), lty = 2)
abline(v = quantile(H, 0.975), lty = 2)
```

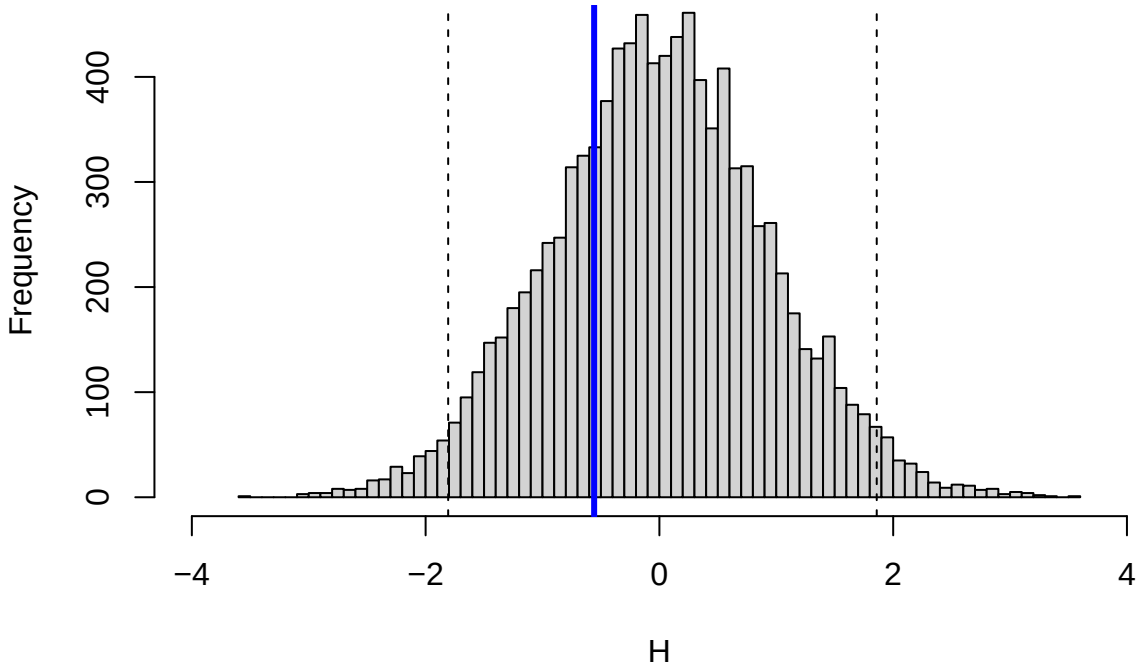


Figure 5.3.4: Hull recommendation and realized outcome.

```
set.seed <- 1
Ca <- rnorm(10000, R_cac40, S_cac40)
hist(Ca, 100, xlim = c(-4, 4), ylim = c(0, 450), main = "")
abline(v = R_cac40.realized, col = "blue", lwd = 3)
abline(v = quantile(C, 0.025), lty = 2)
abline(v = quantile(C, 0.975), lty = 2)
```

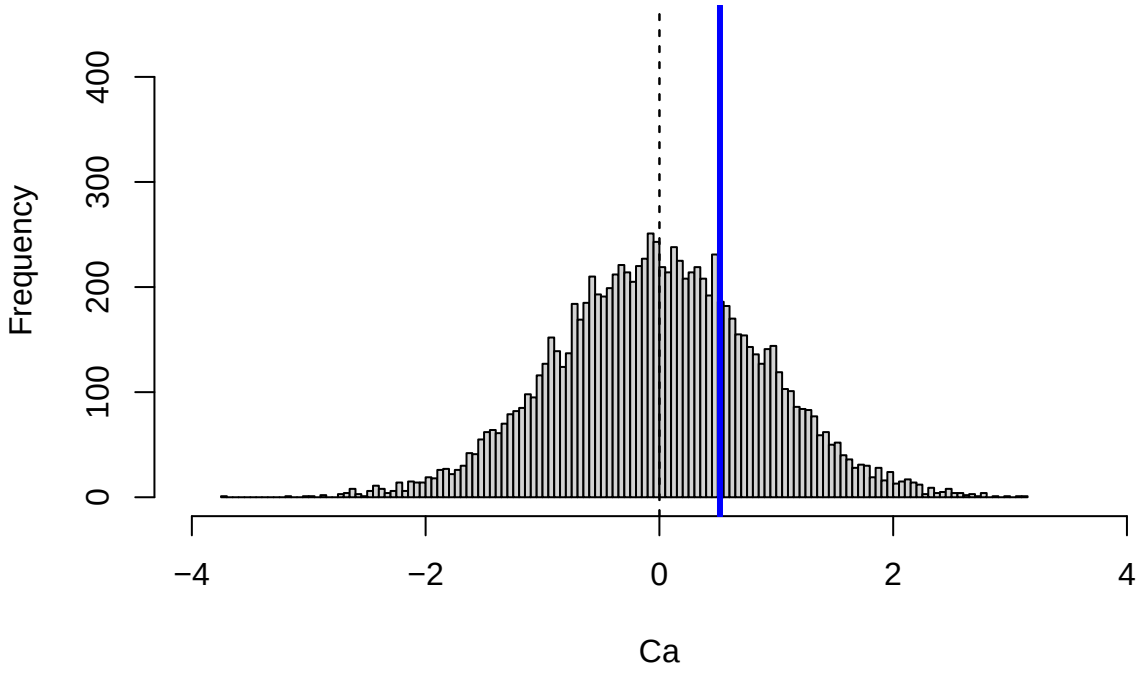


Figure 5.3.5: CAC40 recommendation and realized outcome.

Let's compare a different portfolio. This is optimal and targets the CAC 40 expected return.

```
plot(S, ER, type = "l", lwd = 2, xlim = c(0.7, 1.6),
     ylim = c(-0.022, 0.007), ylab = "Expected return",
     xlab = "Standard deviation")
points(diag(sigma)^0.5, E)
abline(0, 0, lty = 3)
points(S_cac40, R_cac40, col = "red", pch = 19, cex = 2)
points(0.68, R_cac40.realized, col = "red", pch = 15, cex = 2)
text(S_cac40, R_cac40, "CAC40
     expected return", pos = 2, cex = 0.8)
text(sd,E,stocks,adj=-0.5,cex=0.8)
W_hull = alpha / 10000
R_hull = c(t(W_hull) %*% E)
```

```
S_hull = c(t(W_hull) %*% sigma %*% W_hull)^0.5
points(S_hull, R_hull, col = "blue", pch = 19, cex = 2)
points(0.68, R_hull.realized, col = "blue", pch = 15, cex = 2)
abline(0,R_hull / S_hull, lty = 3, col = "blue")
abline(0,R_cac40 / S_cac40, lty = 3, col = "red")
abline(0,E[3] / sd[3], lty = 3)
abline(v = S_cac40, lty = 3)
text(S_hull, R_hull, "Hull's alpha", adj = -0.3, cex = 0.8, pos = 1)
legend("bottomleft", legend = c("DJ=0.4, FTSE=0.3, CAC=0.1, Nikkei=0.2",
                                "DJ=0.607, FTSE=-0.403, CAC=0.459, Nikkei=0.336"),
       col = c("blue", "red"), pch = 19, bg = "white", cex = 0.8)
```

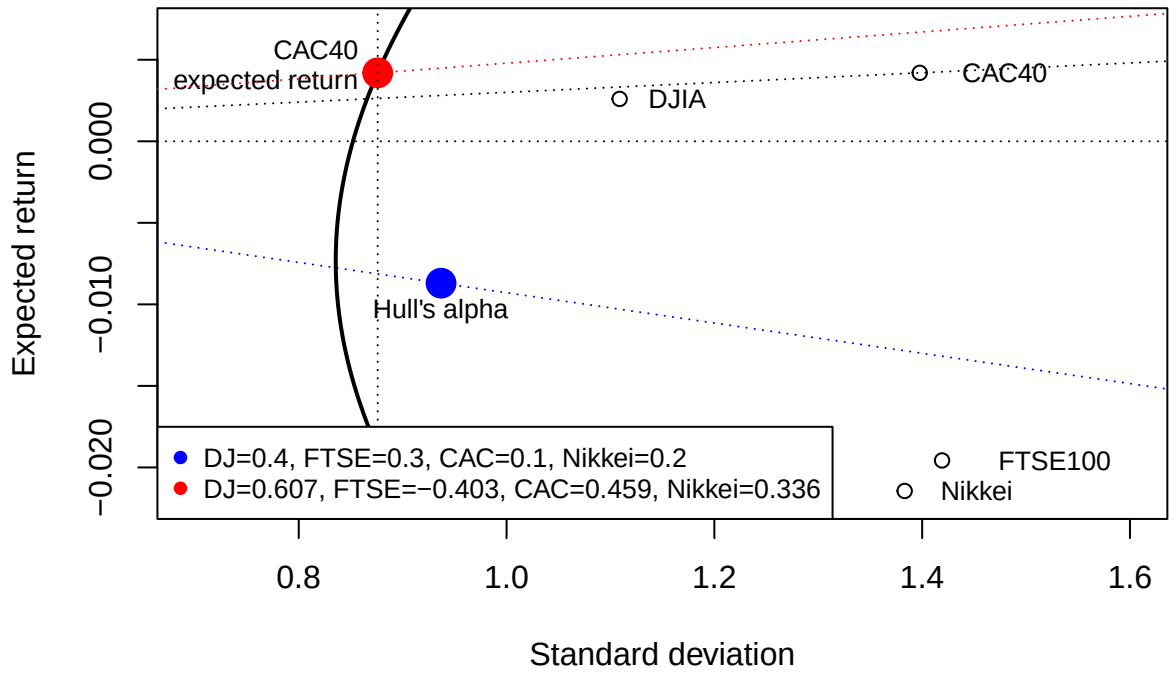


Figure 5.3.6: Estimation of a new optimal alpha: Replicate CAC40 ER.

```
(R_min.realized)
```

```
## [1] 0.2629549
```

```
(R_hull.realized)
```

```
## [1] -0.5583249
```

```
(R_cac40.realized)
```

```
## [1] 0.5183212
```

As stated before, Hull's sub-optimal portfolio allocation turns out to be problematic. His recommendation would lead to a loss (-0.558) and the optimal minimum variance would lead to a positive return (0.2629), and the optimal portfolio that targets the CAC 40 expected return leads to an even higher return (0.518). Nice.

5.4 Model building approach with optimal weights.

Now that we have two additional optimal portfolios, we can compute the corresponding VaR for the model building approach. First, the equally weighted case.

```
# Model building approach, equally weighted.
alpha2 <- c(W_min * 10000)
alpha3 <- c(W.cac40 * 10000)
# This is the minimum variance optimal portfolio.
Pvar2 <- t(alpha2) %*% Rcov %*% alpha2
Psd2 <- Pvar2^0.5
VaR.ew.optimal <- qnorm(0.99, 0, 1) * Psd2
# This is optimal portfolio targeting the CAC 40 expected return.
Pvar3 <- t(alpha3) %*% Rcov %*% alpha3
Psd3 <- Pvar3^0.5
VaR.ew.optimal.cac <- qnorm(0.99, 0, 1) * Psd3
```

Summary of results:

```
Results <- data.frame(VaR.method=c("VaR.hist",
                                   "VaR.hist.quantile",
                                   "VaR.ew",
                                   "VaR.ew.optimal",
                                   "VaR.ew.optimal.cac",
```

```
                                    "VaR.ewma",
                                    "VaR.ewma.martin"),
                          value=c(VaR.hist,
                                  VaR.hist.quantile,
                                  VaR.ew,
                                  VaR.ew.optimal,
                                  VaR.ew.optimal.cac,
                                  VaR.ewma,
                                  VaR.ewma.martin))
kable(Results, caption = "Summary of results.", digits = 4) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.4.1: Summary of results.

VaR.method	value
VaR.hist	253.3850
VaR.hist.quantile	218.3281
VaR.ew	217.9751
VaR.ew.optimal	194.3892
VaR.ew.optimal.cac	203.7821
VaR.ewma	470.9204
VaR.ewma.martin	471.0252

Second, the EWMA case.

```
Pvar2_EWMA <- t(alpha2) %*% C %*% alpha2
Psd2_EWMA <- Pvar2_EWMA^0.5
VaR.ewma.optimal <- qnorm(0.99, 0, 1) * Psd2_EWMA
VaR.ewma.optimal
```

```
##          [,1]
## [1,] 338.2937
```

```
Pvar3_EWMA <- t(alpha3) %*% C %*% alpha3
Psd3_EWMA <- Pvar3_EWMA^0.5
VaR.ewma.optimal.cac <- qnorm(0.99, 0, 1) * Psd3_EWMA
VaR.ewma.optimal.cac
```

```
##           [,1]
## [1,] 347.9537
```

Summary of results.

```
Results <- data.frame(VaR.method=c("VaR.hist",
                                   "VaR.hist.quantile",
                                   "VaR.ew",
                                   "VaR.ew.optimal",
                                   "VaR.ewma",
                                   "VaR.ewma.optimal",
                                   "VaR.ewma.optimal.cac",
                                   "VaR.ewma.martin"),
                      value=c(VaR.hist,
                              VaR.hist.quantile,
                              VaR.ew,
                              VaR.ew.optimal,
                              VaR.ewma,
                              VaR.ewma.optimal,
                              VaR.ewma.optimal.cac,
                              VaR.ewma.martin))
kable(Results, caption = "Summary of results.", digits = 4) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.4.2: Summary of results.

VaR.method	value
VaR.hist	253.3850
VaR.hist.quantile	218.3281
VaR.ew	217.9751
VaR.ew.optimal	194.3892
VaR.ewma	470.9204
VaR.ewma.optimal	338.2937
VaR.ewma.optimal.cac	347.9537
VaR.ewma.martin	471.0252

And third, my EWMA version.

```
# My EWMA approach
VaR.ewma.martin
```

```
##          [,1]
## [1,] 471.0252
```

```
Pvar_EWMA12 <- t(alpha2) %*% VCV %*% alpha2
Psd_EWMA12 <- Pvar_EWMA12^0.5
VaR.ewma.martin.optimal <- qnorm(0.99, 0, 1) * Psd_EWMA12
VaR.ewma.martin.optimal
```

```
##          [,1]
## [1,] 338.4803
```

```
Pvar_EWMA13 <- t(alpha3) %*% VCV %*% alpha3
Psd_EWMA13 <- Pvar_EWMA13^0.5
VaR.ewma.martin.optimal.cac <- qnorm(0.99, 0, 1) * Psd_EWMA13
VaR.ewma.martin.optimal.cac
```

```
##          [,1]
## [1,] 348.1087
```

Summary of results.

```
Results <- data.frame(VaR.method=c("VaR.hist",
                                   "VaR.hist.quantile",
                                   "VaR.ew",
                                   "VaR.ew.optimal",
                                   "VaR.ewma",
                                   "VaR.ewma.optimal",
                                   "VaR.ewma.martin",
                                   "VaR.ewma.martin.optimal",
                                   "VaR.ewma.martin.optimal.cac"),
                      value=c(VaR.hist,
                              VaR.hist.quantile,
                              VaR.ew,VaR.ew.optimal,
                              VaR.ewma,
                              VaR.ewma.optimal,
                              VaR.ewma.martin,
```

```
                              VaR.ewma.martin.optimal,
                              VaR.ewma.martin.optimal.cac))
kable(Results, caption = "Summary of results.", digits = 4) |>
kable_styling(latex_options = "HOLD_position")
```

Table 5.4.3: Summary of results.

VaR.method	value
VaR.hist	253.3850
VaR.hist.quantile	218.3281
VaR.ew	217.9751
VaR.ew.optimal	194.3892
VaR.ewma	470.9204
VaR.ewma.optimal	338.2937
VaR.ewma.martin	471.0252
VaR.ewma.martin.optimal	338.4803
VaR.ewma.martin.optimal.cac	348.1087

5.5 Historic approach with optimal weights.

Let's go back to the historic approach. Now we can incorporate a comparison with the optimal weights.

```
# Miniumu variance optimal weights.
p2 <- (alpha2[1] * DJIA[2:501, ] / DJIA[1:500, ]) +
  (alpha2[2] * AFTSE100[2:501, ] / AFTSE100[1:500, ]) +
  (alpha2[3] * ACAC40[2:501, ] / ACAC40[1:500, ]) +
  (alpha2[4] * ANikkei[2:501, ] / ANikkei[1:500, ])
# Optimal portfolio targeting the CAC 40 expected return.
p3 <- (alpha3[1] * DJIA[2:501, ] / DJIA[1:500, ]) +
  (alpha3[2] * AFTSE100[2:501, ] / AFTSE100[1:500, ])+
  (alpha3[3] * ACAC40[2:501, ] / ACAC40[1:500, ]) +
  (alpha3[4] * ANikkei[2:501, ] / ANikkei[1:500, ])
# Losses in both cases.
l2 <- 10000 - p2
l3 <- 10000 - p3
```

This is the Hull original case.

```
plot(l, type = "h", ylim = c(-600, 600), lwd = 2,
     xlab = "500 scenarios (sorted by historical date)",
ylab = "Gains (-) and losses (+)", col = ifelse(l < 0, "blue", "red"))
abline(0, 0)
abline(v = 0)
abline(h = max(l), lty = 2)
abline(h = min(l), lty = 2)
```

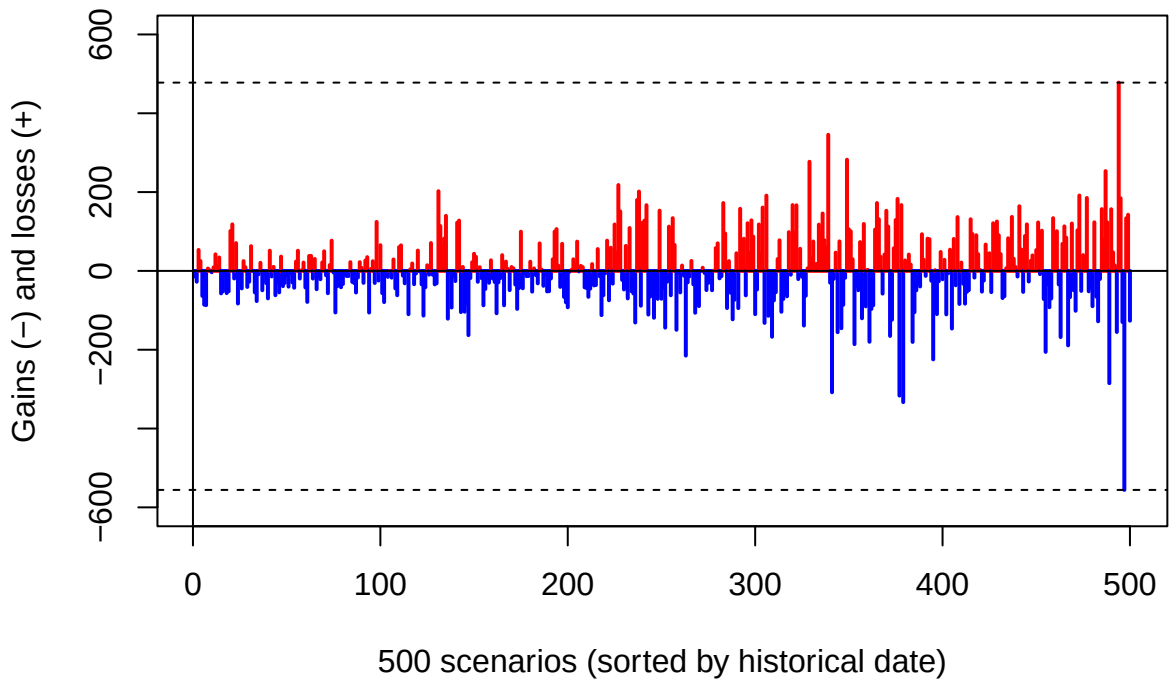


Figure 5.5.1: Losses with Hull's suboptimal alpha DJ=0.4, FTSE=0.3, CAC=0.1, Nikkei=0.2.

The one-day 99% value at risk can be estimated as the fifth-worst loss.

```
sort(l)[496]
```

```
## [1] 253.385
```

Now, let's see the minimum variance optimal portfolio.

```
plot(l2, type = "h", ylim = c(-600, 600), lwd = 2,
     xlab = "500 scenarios (sorted by historical date)",
     ylab = "Gains (-) and losses (+)", col = ifelse(l2 < 0, "blue", "red"))
abline(0, 0)
abline(v = 0)
abline(h = max(l2), lty = 2)
abline(h = min(l2), lty = 2)
```

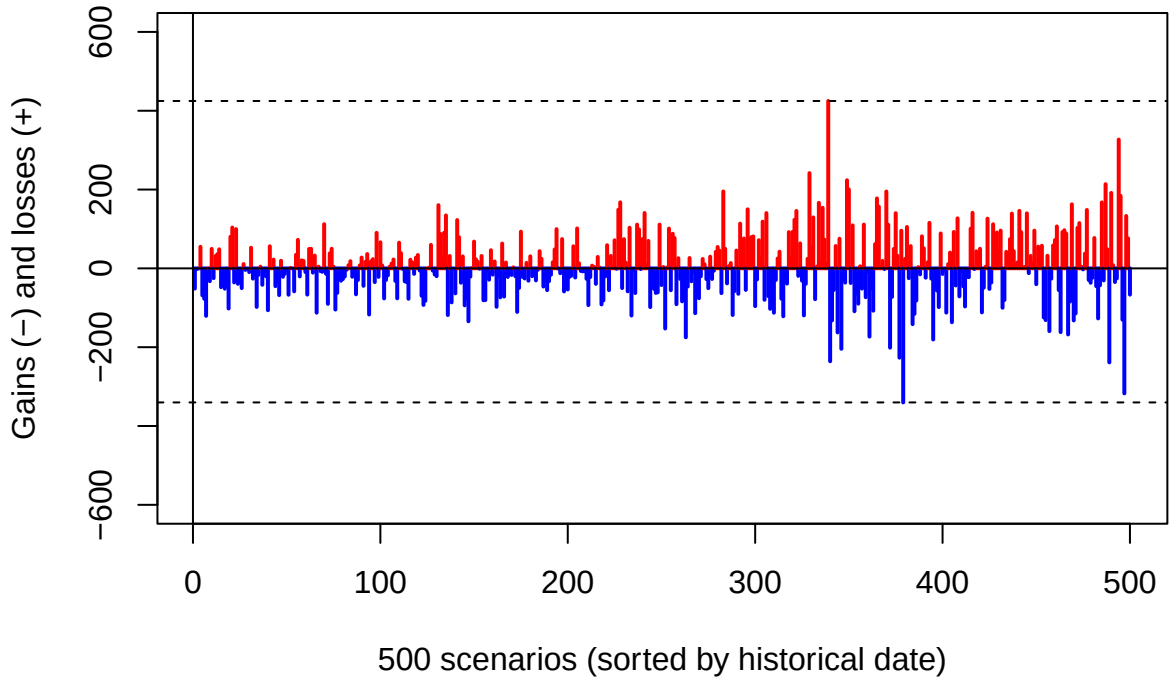


Figure 5.5.2: Losses with optimal alpha DJ=0.56, FTSE=0.038, CAC=0.022, Nikkei=0.38.

The one-day 99% value at risk can be estimated as the fifth-worst loss.

```
sort(l2)[496]
```

```
## [1] 213.6898
```

See how the range of losses and gains is now reduced.

And this is the optimal portfolio targeting the CAC 40 expected return.

```
plot(l3, type = "h", ylim = c(-600, 600), lwd = 2,
     xlab = "500 scenarios (sorted by historical date)",
     ylab = "Gains (-) and losses (+)", col = ifelse(l3 < 0, "blue", "red"))
abline(0, 0)
abline(v = 0)
abline(h = max(l3), lty = 2)
abline(h = min(l3), lty = 2)
```

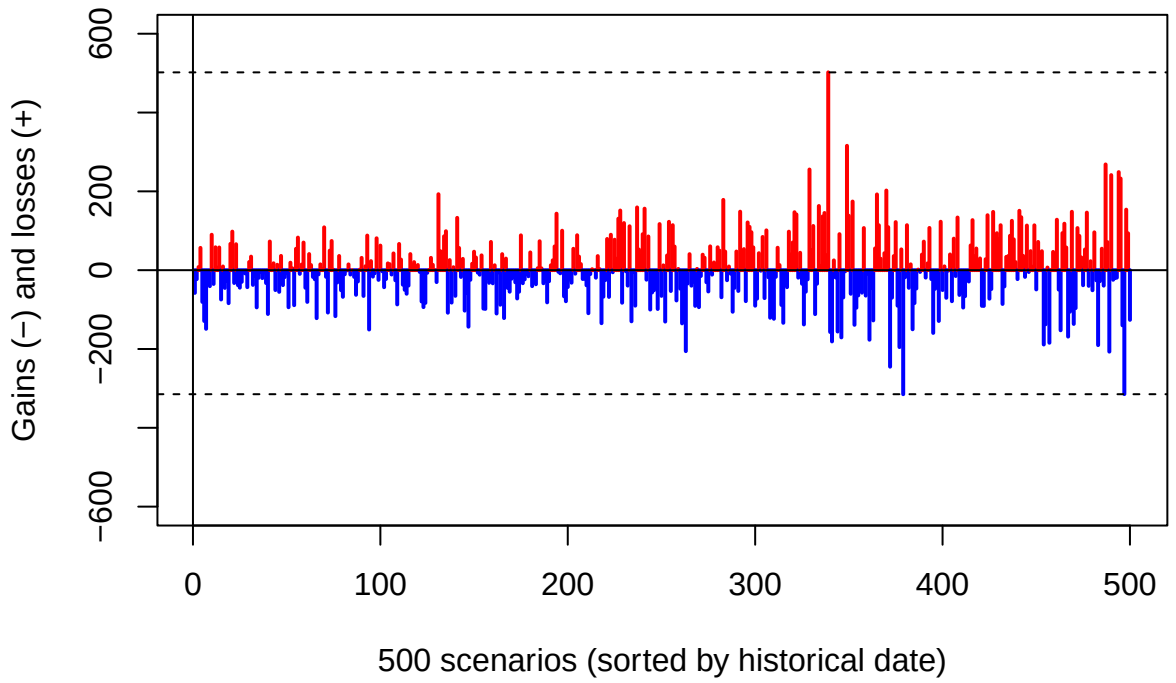


Figure 5.5.3: Losses with CAC40 optimal alpha DJ=0.607, FTSE=-0.403, CAC=0.459, Nikkei=0.336.

The one-day 99% value at risk can be estimated as the fifth-worst loss.

```
sort(l3)[496]
```

```
## [1] 248.8909
```

Note that the range is reduced but allows for a higher gains. Let's see these three together.

```
par(mfrow=c(1, 3), oma = c(0, 0, 2, 0))
par(pty = "s")
plot(l, type = "h", ylim = c(-600, 600), lwd = 2, xlab = "",
     main = "Hull original case.", ylab = "Gains (-) and losses (+)",
     col = ifelse(l < 0, "blue", "red"))
abline(0, 0)
abline(v = 0)
abline(h = max(l), lty = 2)
abline(h = min(l), lty = 2)
plot(l2, type = "h", ylim = c(-600, 600), lwd = 2,
     main = "Minimum variance.",
     xlab = "500 scenarios sorted by historical date.", ylab = "",
     col = ifelse(l2 < 0, "blue", "red"))
abline(0, 0)
abline(v = 0)
abline(h = max(l2), lty = 2)
abline(h = min(l2), lty = 2)
plot(l3, type = "h", ylim = c(-600, 600), lwd = 2, xlab = "",
     main = "Optimal targeting CAC 40.", ylab = "",
     col = ifelse(l3 < 0, "blue", "red"))
abline(0, 0)
abline(v = 0)
abline(h = max(l3), lty = 2)
abline(h = min(l3), lty = 2)
```

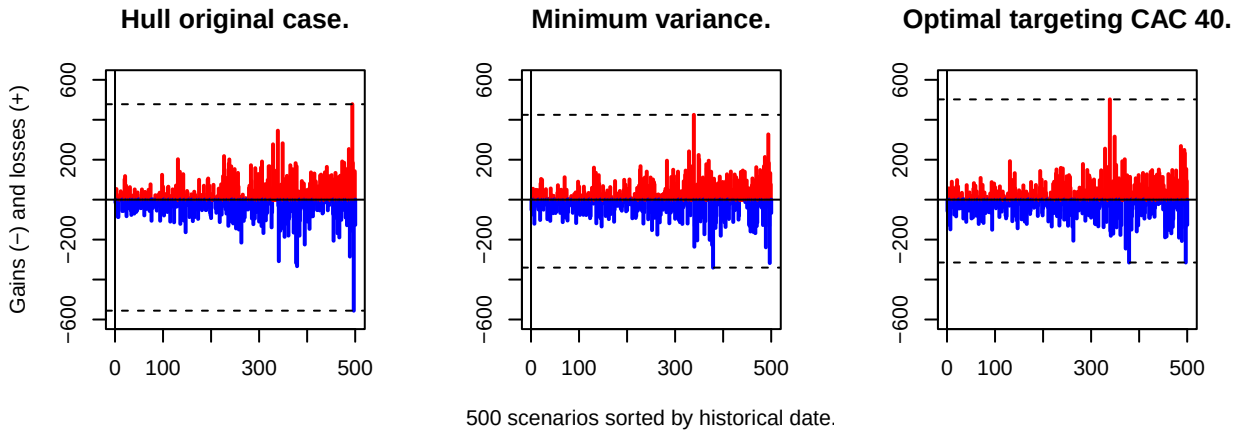


Figure 5.5.4: Hull's suboptimal alpha (left); optimal alpha (center); optimal alpha CAC40 (right).

A quantile comparison.

```
lq <- quantile(l, c(0.25, 0.5, 0.75))
lq2 <- quantile(l2, c(0.25, 0.5, 0.75))
lq3 <- quantile(l3, c(0.25, 0.5, 0.75))
ll <- data.frame(lq, lq2, lq3)
colours <- c("red", "orange", "blue")
```

```
barplot(as.matrix(t(ll)), beside = TRUE, col = colours, ylim = c(-60, 60),
        xlab = "Quantiles", ylab = "Gains (-) and Losses (+)")
legend("topleft", c("Hull original","Optimal minvar",
                    "Replicating CAC40 ER"), cex = 1.3, bty = "n",
       fill = colours)
```

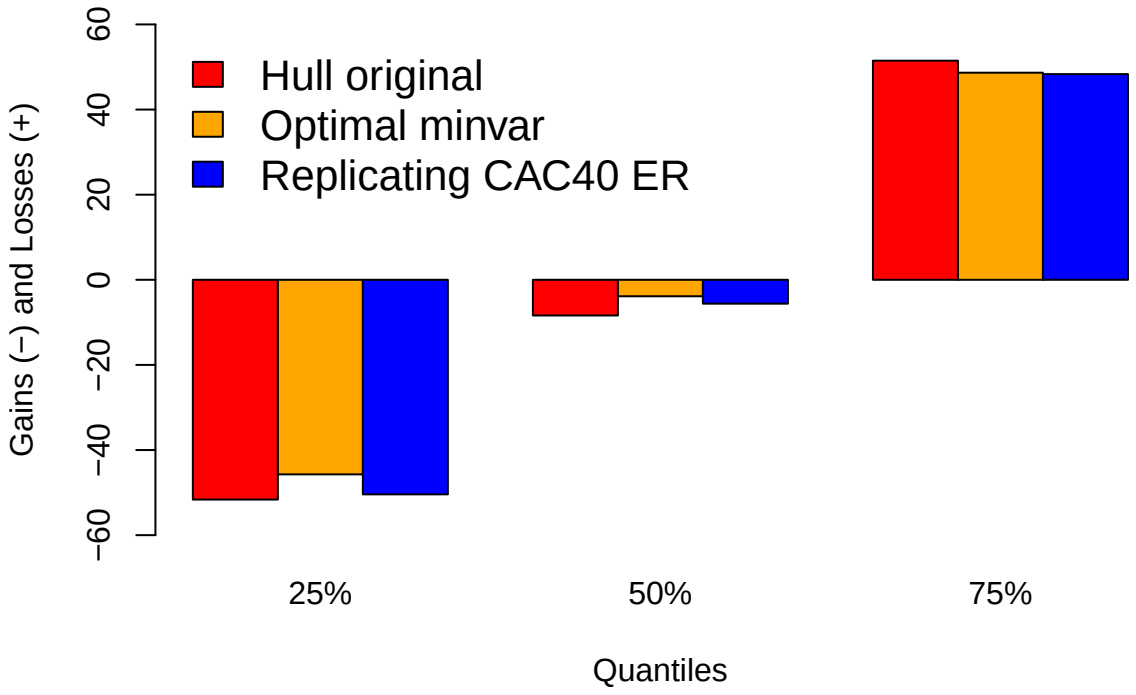


Figure 5.5.5: Quantile comparison of losses distribution.

```
a <- toc()
```

```
## 34.82 sec elapsed
```

This document took 34.82 seconds to compile in Rmarkdown.

References.

John C. Hull. *Options, futures, and other derivatives*. Prentice Hall, 9th ed. edition, 2015.

Donald Ervin Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.