Quantitative Finance

Lab - Monte Carlo Methods”

Fiona Martin, Romain Pepin, Hedi Sagar, Nathan Sanglier

Version: 10 avr. 2024

```r
library(anytime)
library(xts)
library(xtable)
library(timeDate)
library(fOptions)
#library(fExoticOptions)
library(ggplot2)
library(lubridate)
library(NFCP)
```

The objective of this lab is the pricing by simulation of options that enable several exercises during the option life. Some examples of this kind of options are :

- EDF’s “Tempo” tariff, which applies a high tariff 22 days of the year, during periods of high demand, in exchange for a low tariff the rest of the time.
- Swing” or “take-or-pay” options used in the commodities markets. In this case, the option gives the right to modulate the quantity of commodity purchased, depending on the conditions at the time. In the case of natural gas, for example, the parties, having agreed on a volume of gas to be delivered each day, also contract a ‘swing’ option, which gives the buyer the right to request delivery of an additional quantity of gas at a pre-determined price. Over a one-month period, the buyer may have around ten daily rights of this kind.

In this lab, we focus on “swing” options. In 2004, Jaillet defined a pricing algorithm for a one factor model; however this solution does not generalize for multifactor models. Thus, we’ll focus here on pricing a swing option thanks to the Longstaff-Schwartz algorithm.

Longstaff-Schwartz Algorithm

In this first part, as an introduction, we’ll price an American option with maturity 1 year, using the Longstaff-Schwartz algorithm.

This algorithm is based on Monte-Carlo simulation and dynamic programming. The main idea is to recall that at any exercise time, the holder of an American option optimally compares the payoff from immediate exercise with the expected payoff from continuation and then exercises if the immediate payoff is higher. The optimal exercise strategy is thus determined by the conditional expectation of the payoff from continuing to keep the option alive. Although this conditional expectation is not directly observable, it can be estimated via a functional form as a weighted sum of basis functions, where the weights are determined by regressing ex-post realized payoffs from continuation on this sum of basis functions of the values of the state variables.

However, we must work backwards since the path of cash flows generated by the option is defined recursively. Indeed, if it is optimal to exercise at t, we need to know if it was optimal to exercise at $t+1$, in order to compare payoff from immediate exercise at t with expected payoff from continuation at t (which depends on immediate exercise at $t+1$ versus continuation at $t+1$, and so on). This refers to dynamic programming, as in order to "solve" the problem with t timesteps, we need to "solve" it first with $t+1$ timesteps.

More specifically, let us denote N the number of paths in our Monte-Carlo simulation and T the option maturity. Then, we denote $\forall i \in 1:N$, $\forall t \in 1:T$, $V_{i,t}$ the option price for path i at time $t=0$ if the option can't be exercised before time t. Similarly, the underlying price is $S_{i,t}$ and the discounted continuation value for path i at time $t=0$ (i.e. the option price for path i at time $t=0$ if the option can't be exercised before and at time t) is :

$$C_{i,t} = \mathbb{E}\left[V_{t+1}\left(S_{i,t+1}\right) \mid S_{i,t}\right]$$

where V_{t+1} is the unobservable option price at time $t=0$ if the option can't be exercised before time t, as a function of the underlying price.

Thus, we'll estimate $C_{i,t}$ by :

$$\hat{C_{i,t}} = \sum_{r=0}^{R-1} \beta_{r,t}\phi_r\left(S_{i,t}\right)$$

where ϕ_r's are the first R functions of a basis of a function space. The coefficients $\beta_{r,t}$'s are determined by regressing ex-post realized payoffs (here we'll take payoffs discounted to $t=0$ for coding simplicity, but it changes nothing) from continuation onto the $\phi_r\left(S_{i,t}\right)$'s, only for paths that are in-the-money at time t ($h\left(S_{i,t} > 0\right)$), where h is the payoff function.

Now, the question is how to update $V_{i,t}$ based on $V_{i,t+1}$ as we use dynamic programming. We have $\forall t \in 1:T-1$ (see Longstaff, Schwartz, Valuing American Options by Simulation, 2001) :

$$V_{i,t} = \begin{cases} V_{i,t+1} & \hat{C_{i,t}} > \frac{h(S_{i,t})}{(1+r)^t} \\ \frac{h(S_{i,t})}{(1+r)^t} & \hat{C_{i,t}} \leq \frac{h(S_{i,t})}{(1+r)^t} \end{cases}$$

And when $t=T$:

$$V_{i,T} = \frac{h\left(S_{i,T}\right)}{\left(1+r\right)^T}$$

$$C_{i,T} = \hat{C_{i,T}} = 0$$

where r the constant risk-free rate.

With this formulation, notice that discounted to $t=0$ ex-post realized payoffs from continuation at time t are defined as $V_{i,t+1}$.

Then, it is possible to compute an upper bound of our option price using this Longstaff-Schwartz algorithm (see Glasserman, Monte Carlo Methods in Financial Engineering, 2004). In our case, as we did not have time to investigate this upper bound, we'll only focus on computing a lower bound, also by using this algorithm.

The lower bound can be computed using directly our Longstaff-Schwartz algorithm as, by definition, the value of an American option is based on the stopping rule that maximizes the value of the option; all other stopping rules including the stopping rule implied by the Longstaff-Schwartz algorithm (given by comparison of estimated continuation function with discounted payoff at each timestep), result in values less than or equal to that implied by the optimal stopping rule. However, since we are using the same paths for regression as for evaluation, the estimate for the option price could be high biased.

In order to ensure that it is low biased, we must use the original set for computing the coefficients $\beta_{r,t}$'s (i.e. define estimated continuation function), and a new set of paths for the valuation. In this valuation step, we work forward : as soon as $\hat{C_{i,t}} \leq \frac{h(S_{i,t})}{(1+r)^t}$, we set the exercise date to t for this path, and compute the option price on this path as $\frac{h(S_{i,t})}{(1+r)^t}$. Then, the option price lower bound is simply given by the mean of computed option prices in this forward iteration.

Let us implement this lower bound algorithm for the pricing of an American call option ($h(S_{i,t}) = \max(0, S_{i,t} - K)$) assuming the underlying follows a Geometric Brownian Motion. The paths will be generated thanks to function "spot_price_simulate" of the package "NFCP", with a timestep of 1 week and maturity of 1 year. Moreover, we'll take as first basis functions ($R = 2$) 1, X, X^2.

```r
K              = 20      # Option strike price
r              = 0.06    # Risk-free rate
delta.t        = 1/52    # timestep
maturity       = 1       # Option maturity
S.0            = 20      # Initial underlying price
sigma          = .25     # Process Volatility
N              = 10000   # Number of paths
```

```r
# Call payoff
payoff.func <- function(S) {

  pmax(S - K, 0)
}
```

```r
# Continuation function coefficients (beta's)
continuation.func <- function(S.itm, payoff.disc) {

  regressor = cbind(1, S.itm, S.itm^2)

  reg.coeffs = lm(payoff.disc ~ regressor - 1)$coefficients
}
```

```r
# Discounting factor to current time (t = 0)
disc.factor.func <- function(t) {

  1 / (1 + r)^(t * delta.t)
}
```

```r
# Longstaff-Schwartz Algorithm - Inverse Recursion

# Simulation of underlying paths
param       = c(mu_rn=(r - (1/2)*sigma^2), sigma_1=sigma)
sim.spot    = spot_price_simulate(x_0=log(S.0), parameters = param, t = maturity,
                                  dt = delta.t, N_simulations = N,
                                  antithetic = TRUE, verbose = FALSE)
sim.spot    = sim.spot[2:nrow(sim.spot), ] # price a t = 0 is not used here
nb.samples  = nrow(sim.spot)

disc.factor.mat <- outer(1:nb.samples, 1:N, FUN = function(i, j) disc.factor.func(i))
# Matrix of discounted payoff to current time (t = 0) for each path and timestep
payoff.disc.mat = payoff.func(sim.spot) * disc.factor.mat

# Option price at t = 0 (V_{i,t} for all i),
# knowing the option can't be exercised before t (here t = T)
option.price.vect = payoff.disc.mat[nb.samples, ]
# coefficients of each continuation function
# (the coeffs are not the same with the timestep)
continuation.coeffs.mat = matrix(0, nrow=nb.samples, ncol=3)
```

```
t = nb.samples - 1
while (t >= 1) {

  # To select only paths in the money for our regression
  mask.itm = payoff.disc.mat[t, ] > 0

  # Coefficients for \hat{C_{i, t}} estimated by regression of discounted ex-post
  # realized payoffs from continuation onto the basis functions
  continuation.coeffs.mat[t, ] = continuation.func(sim.spot[t, mask.itm],
                                                   option.price.vect[mask.itm])

  # Computation of continuation value \hat{C_{i, t}}
  continuation.val.vect = continuation.coeffs.mat[t, ] %*% rbind(1,
                                                                 sim.spot[t, ],
                                                                 sim.spot[t, ]^2)

  # To select only paths where it is better to exercise now
  mask.exercise.now = payoff.disc.mat[t, ] >= continuation.val.vect
  # Implementing recursive relation
  option.price.vect[mask.exercise.now] = payoff.disc.mat[t, mask.exercise.now]

  t = t - 1
}
```

```
# Longstaff-Schwartz algorithm - Forward iteration

# Re-simulation of paths
sim.spot = spot_price_simulate(x_0=log(S.0), parameters = param, t = maturity,
                               dt = delta.t, N_simulations = N,
                               antithetic = TRUE, verbose = FALSE)
sim.spot = sim.spot[2:nrow(sim.spot), ] # price a t = 0 is not used here
# Matrix of discounted payoff to current time (t = 0) for each path and timestep
payoff.disc.mat = payoff.func(sim.spot) * disc.factor.mat

# Option price at t = 0
option.price.vect = payoff.disc.mat[nb.samples, ]

# To select only paths that have not been exercised yet in the iterations
mask.paths.remain = rep(TRUE, N)

t = 1
while (sum(mask.paths.remain) >= 1 & t <= nb.samples) {

  # Continuation value, computed using coefficients
  # calculated during the inverse recursion
  basis.func.vals = rbind(1,
                          sim.spot[t, mask.paths.remain],
                          sim.spot[t, mask.paths.remain]^2)
  continuation.val.vect = continuation.coeffs.mat[t, ] %*% basis.func.vals

  # If discounted payoff above continuation value, we do exercise
  mask.exercise.now = payoff.disc.mat[t, mask.paths.remain] > continuation.val.vect
  # To select only paths to price (i.e. only if exercise at t)
  mask.path.update = mask.paths.remain
  inds.paths.remain = which(mask.paths.remain)
  mask.path.update[inds.paths.remain[!mask.exercise.now]] = FALSE

  # Update option price by discounted payoff if we have such case
  option.price.vect[mask.path.update] = payoff.disc.mat[t, mask.path.update]

  # removing paths for which we exercise now from remaining paths
  mask.paths.remain[inds.paths.remain[mask.exercise.now]] = FALSE

  t = t + 1
}

option.price.lower = mean(option.price.vect)
print(paste("Option price lower bound from LS algorithm : ",
            round(option.price.lower, 3)))
```

```
## [1] "Option price lower bound from LS algorithm :  1.955"
```

We we'll verify our result by checking this price is indeed below the price obtained with a binomial tree, using "fOptions" package (however, it should not be "too far below").

```
price_fOptions <- CRRBinomialTreeOption(TypeFlag = "ca", S = S.0, X = K,
    Time = maturity, r = r, b = r, sigma = sigma, n = 100)@price

print(paste("Option price using fOptions package : ",
            round(price_fOptions, 3)))
```

```
## [1] "Option price using fOptions package :  2.564"
```

The price found with Longstaff-Schwartz algorithm is indeed a lower bound. A next step, although not done here, would be to implement the procedure to get an upper bound as mentioned previously, see Glasserman; Haugh and Kogan; Rogers about duality approach and the use of a martingale for more details. Briefly, the resulting procedure would be for each path i to generate P "mini-paths", price them by the maximum between estimated continuation value (and not ex-post realized payoffs from continuation) versus discounted immediate payoff, use them to compute martingale values, and finally compute upper bound.

Swing option with M exercises

Now, let's consider a swing option that gives M rights of buying one unit of underlying at price K during the option life. The rights not used at maturity are considered to be lost. We can only exercise one right per period.

Upper and lower bounds

Let us first calculate upper and lower bounds of our option price.

An upper bound can be computed assuming we perfectly know the future (i.e. we know on which path we are and the full trajectory of this path). Thus, we just need to compute for each path the M exercises the most profitable, sum their discounted value, and take the mean on all paths.

```r
M = 10 # Number of exercise rights

# Simulation of underlying paths
sim.spot = spot_price_simulate(x_0=log(S.0), parameters = param, t = maturity,
                               dt = delta.t, N_simulations = N,
                               antithetic = TRUE, verbose = FALSE)
sim.spot = sim.spot[2:nrow(sim.spot), ] # price a t = 0 is not used here
nb.samples = nrow(sim.spot)
# Matrix of discounted payoff to current time (t = 0) for each path and timestep
payoff.disc.mat = payoff.func(sim.spot) * disc.factor.mat

# Function to get the M highest discounted payoffs on a path
sum.top.M <- function(path) {
  payoff.disc.highest.M = head(sort(path, decreasing = TRUE), M)
  sum(payoff.disc.highest.M)
}

# Applying the function defined before to each path
bound.upper = mean(apply(payoff.disc.mat, 2, sum.top.M))
print(paste("Option price upper bound = ", round(bound.upper, 3)))
```

```
## [1] "Option price upper bound =  34.531"
```

A lower bound is obtained by an heuristic, by testing decision rules defined a priori. We have chosen here to define a decision rule such that we exercise when $S_{i,t}$ is above a certain $\alpha_t \cdot \sigma\sqrt{t}$, as it means the payoff from immediate exercise is especially high at this time t and thus has more chances to be higher than continuation value. This level α_t will decrease as t increases to be sure all rights are exercised when $t = T$. Moreover, when all M rights are exercised for a path, we do not consider this path in the next timesteps.

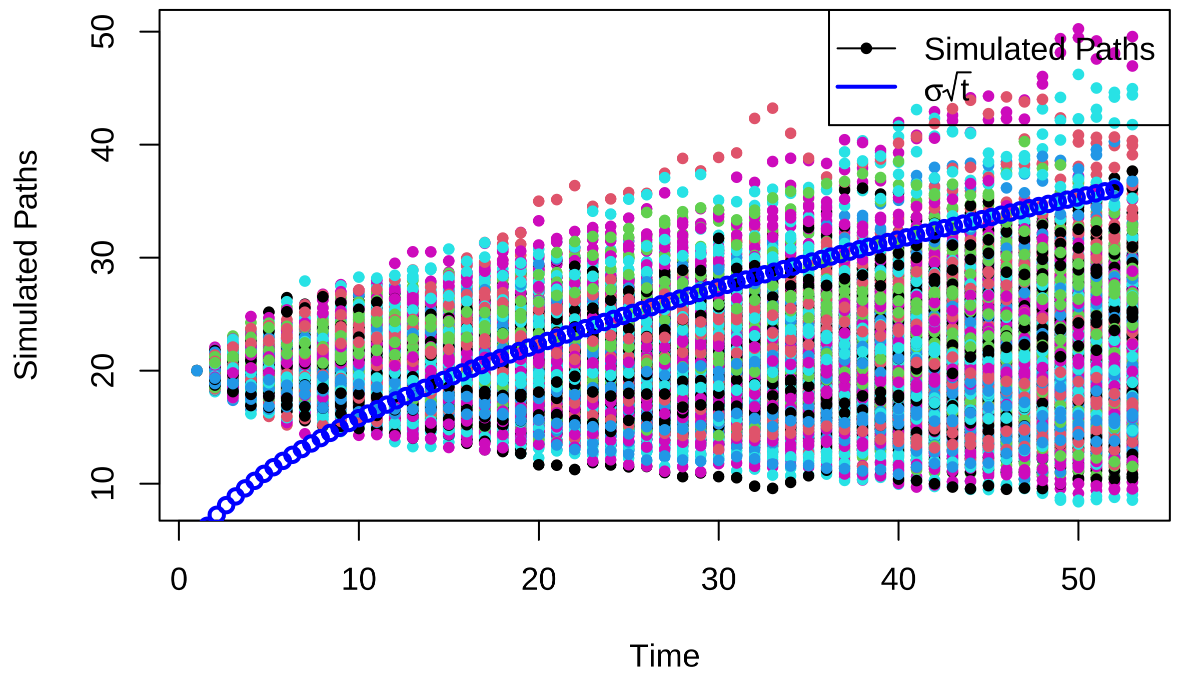
In order to find an adequate value for α_t, we have plotted the different simulations obtained using spot_price_simulate as well as a curve in $\sigma\sqrt{(t)}$. Our goal was to empirically find a good expression for α_t by finding a curve which variations was mostly above those of the simulations.

```r
N_ = 1000 # we diminish the number of paths


sim.spot = spot_price_simulate(x_0=log(S.0), parameters = param, t = maturity,
        dt = delta.t, N_simulations = N_, antithetic = TRUE, verbose = FALSE)

# Plot the simulated paths
matplot(sim.spot, type = "p", pch = 20, xlab = "Time", ylab = "Simulated Paths")

# Plot the "heuristic" curve
t_values <- seq(0, 1/delta.t, length.out = 100)
sigma_sqrt_t <- 20*sigma * sqrt(t_values)
points(t_values, sigma_sqrt_t, col = "blue", lwd = 2)
legend("topright", legend = c("Simulated Paths", expression(sigma * sqrt(t))),
    col = c("black", "blue"), lwd = c(1, 2), pch = c(20, NA))
```



As expected, the paths obtained are quite widely distributed which led us to believe that the global "tendency" (high or low values) of the path should be taken into account. To do so we could, keeping in mind that we need to limit ourselves to the knowledge of past spot prices, we could take the mean of the spot prices from 0 to t and/or their standard deviation into the expression of α_t.

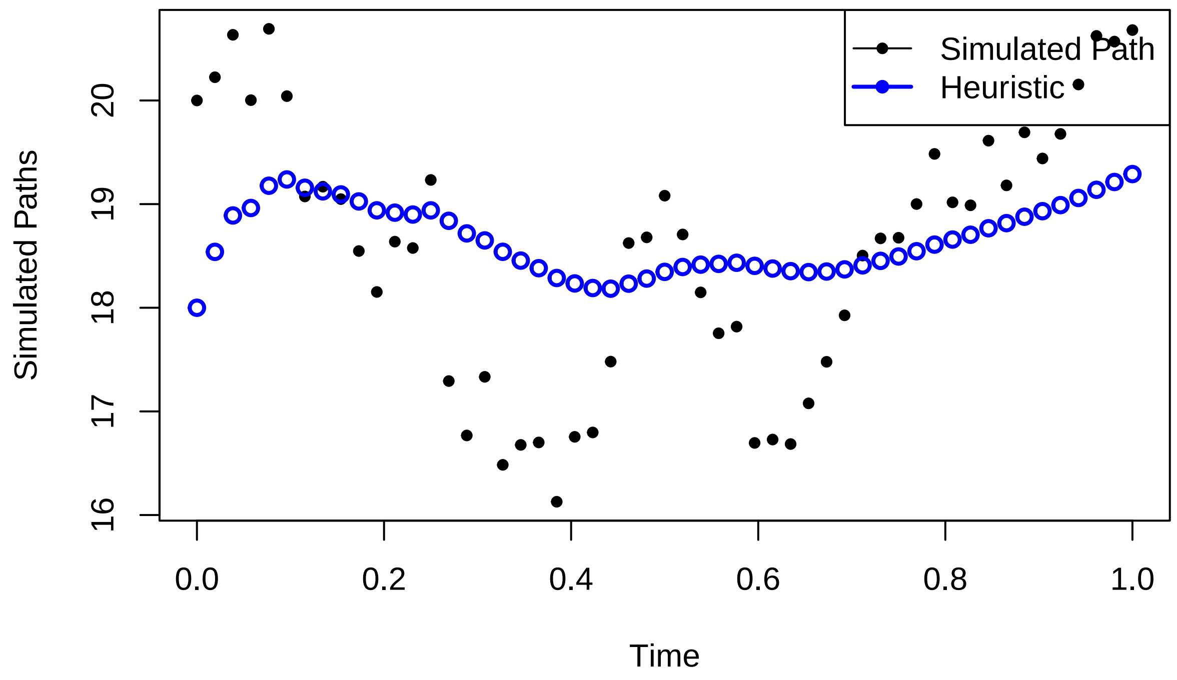
```r
# Only focusing on the first path for example
cum_avg <- cumsum(sim.spot[,1]) / seq_along(sim.spot[,1])

# Plot the simulated paths
matplot(seq(0, maturity, delta.t), sim.spot[, 1], type = "p", pch = 20, xlab =
          "Time", ylab = "Simulated Paths")

# Plot the "heuristic" curve using the mean of the values
t_values <- seq(0, maturity, length.out = length(sim.spot[, 1]))

# We used log to "contract" the curve and played around to find a function
# somewhat acceptable graphically
comparison_term <-  0.9*cum_avg*(1+sigma * sqrt(t_values) * log(cum_avg/10))
points(t_values, comparison_term, col = "blue", lwd = 2)

legend("topright", legend = c("Simulated Path", "Heuristic"), col = c("black",
        "blue"), lwd = c(1, 2), pch = c(20, 20))
```



Had we had more time we would have further explored around this idea and improved the 'comparison_term'. We can still try to use this general formula to which the payoff from immediate exercise can be compared at each time step. We hope to get a lower bound of the price of the swing option by being likely to exercise sooner than optimally.

```
M = 10 # Number of exercise rights
sim.spot = spot_price_simulate(x_0=log(S.0), parameters = param, t = maturity,
                               dt = delta.t, N_simulations = N_,
                               antithetic = TRUE, verbose = FALSE)


# To select only paths that have not already been M times
mask.paths.remain = rep(M, N_)

# Sum of exercise amounts for each path
sum_exercices = rep(0, N_)

t = 1
while (t <= nb.samples) {

  for (n in 1:N_){
    cum_avg <- cumsum(sim.spot[,n]) / seq_along(sim.spot[,n])

    if (mask.paths.remain[n] >= 0){
      comparison_term <- 0.8*cum_avg[t]*(1+sigma * sqrt(t) * log(cum_avg[t]/10))
      if (comparison_term < sim.spot[t, n]){
        # we exercise so have 1 less right
        mask.paths.remain[n] = mask.paths.remain[n] - 1

        # we discount the actual price and add it to the global sum
        disc.factor = 1 / (1 + r)^(t * delta.t)
        sum_exercices[n] = sim.spot[t, n] * disc.factor
      }
    }

  }

  t = t + 1
}
option.price.lower = mean(sum_exercices)
print(paste("Swing otion price lower bound from LS algorithm : ",
            round(option.price.lower, 3)))
```

```
## [1] "Swing otion price lower bound from LS algorithm :  20.259"
```

Pricing

In order to price a swing option using Longtsaff-Shcwartz algorithm, we cannot directly use this method. Indeed, we need to introduce a second state variable, which is the number of rights $m \in 0 : M$. At each timestep in the inverse recursion, the computation of continuation needs to be done for each value of the state variable m.

Thus, let us denote we denote $\forall i \in 1 : N$, $\forall t \in 1 : T$, $\forall m \in 0 : M$, $V_{i,t}^m$ the option price for path i at time $t = 0$ if the option can't be exercised before time t and we have m rights. Similarly, the discounted continuation value for path i at time $t = 0$ if we have m rights (i.e. the option price for path i at time $t = 0$ if the option can't be exercised before and at time t and we have m rights) is :

$$C_{i,t}^m = \mathbb{E}\left[V_{t+1}^m\left(S_{i,t+1}\right) \mid S_{i,t}\right]$$

Again, we'll estimate $C_{i,t}^m$ the same way as for the Longstaff Schwartz algorithm.

Thus, the new relation to update $V_{i,t}^m$ based on $V_{i,t+1}^m$ as we use dynamic programming. We have $\forall t \in 1 : T-1$, $\forall m \in 1 : M$:

$$V_{i,t}^m = \begin{cases} V_{i,t+1}^m & \hat{C_{i,t}^m} > \frac{h(S_{i,t})}{(1+r)^t} \\ \frac{h(S_{i,t})}{(1+r)^t} + V_{i,t+1}^{m-1} & \hat{C_{i,t}^m} \leq \frac{h(S_{i,t})}{(1+r)^t} \end{cases}$$

Notice that when $m = 0$, $\forall t \in 1 : T$, $V_{i,t}^0 = 0$. Indeed, if we exercise now, then we have discounted payoff from exercise and the other payoffs from exercises from $t + 1$ to T with $m - 1$ rights, as we have used one at time t. This ensures we indeed do not use more than our M exercise rights.

And when $t = T$, $\forall m \in 1 : M$:

$$V_{i,T}^m = \frac{h\left(S_{i,T}\right)}{\left(1+r\right)^T}$$

$$C_{i,T}^m = \hat{C_{i,T}}^m = 0$$

As before, notice that discounted to $t = 0$ ex-post realized payoffs from continuation at time t with m rights are defined as $V_{i,t+1}^m$.

```r
M = 10

# Longstaff-Schwartz Algorithm - Inverse Recursion

# Simulation of underlying paths
param       = c(mu_rn=(r - (1/2)*sigma^2), sigma_1=sigma)
sim.spot    = spot_price_simulate(x_0=log(S.0), parameters = param, t = maturity,
                                  dt = delta.t, N_simulations = N,
                                  antithetic = TRUE, verbose = FALSE)
sim.spot    = sim.spot[2:nrow(sim.spot), ] # price a t = 0 is not used here
nb.samples  = nrow(sim.spot)

disc.factor.mat <- outer(1:nb.samples, 1:N, FUN = function(i, j) disc.factor.func(i))
# Matrix of discounted payoff to current time (t = 0) for each path and timestep
payoff.disc.mat = payoff.func(sim.spot) * disc.factor.mat

# Option price at t = 0 (V_{i,t}^m for all i for all m),
# knowing the option can't be exercised before t (here t = T)
option.price.mat = matrix(
  rep(payoff.disc.mat[nb.samples, ], M+1),
  nrow = M+1,
  byrow = TRUE
  )
option.price.mat[1, ] = 0 # if no rights to exercise (m = 0), option price is null
# coefficients of each continuation function for level of m
# (the coeffs are not the same with the timestep)
continuation.coeffs.tens = array(0, dim = c(nb.samples, 3, M+1))

t = nb.samples - 1
while (t >= 1) {

  for (m in 2:(M+1)) {

    # To select only paths in the money for our regression
    mask.itm = payoff.disc.mat[t, ] > 0

    # Coefficients for \hat{C_{i, t}} estimated by regression of discounted ex-post
    # realized payoffs from continuation onto the basis functions
    continuation.coeffs.tens[t, , m] = continuation.func(
      sim.spot[t, mask.itm],
      option.price.mat[m, mask.itm]
      )

    # Computation of continuation value \hat{C_{i, t}^m}
    continuation.val.vect = continuation.coeffs.tens[t, , m] %*% rbind(
      1,
      sim.spot[t, ],
      sim.spot[t, ]^2
      )

    # To select only paths where it is better to exercise now
    mask.exercise.now = payoff.disc.mat[t, ] >= continuation.val.vect
    # Implementing recursive relation
    option.price.mat[m, mask.exercise.now] =
      payoff.disc.mat[t, mask.exercise.now] + option.price.mat[m-1, mask.exercise.now]

  }

  t = t - 1

}

# option price is the option price with M rights available
option.price.ls = mean(option.price.mat[M+1, ])
print(paste("Option price from LS algorithm : ",
            round(option.price.ls, 3)))
```

```
## [1] "Option price from LS algorithm :  12.029"
```

It seems there are some issues as the swing option price obtained is often far from the upper bound found previously (and the variance of the resulting price is high) and below the lower bound we found ! Unfortunately, we did not have time to investigate further the issue. However, when setting $M = 1$, i.e. the case of American call of previous part, we do find a price around 2.0 which is alike the lower bound price obtained with Longstaff-Schwartz algorithm in previous part (doing inverse recursion and then forward iteration), but below the american call price given by a binomial tree.

Conclusion :

This lab gave us the opportunity to first implement the Longstaff-Schwartz model on the classical example of an American call option. This algorithm is based on Monte-Carlo simulation and dynamic programming. At each timestep, we compare the payoff from immediate exercise with the expected payoff from continuation (estimated by least squares approach) and then exercise if the immediate payoff is higher. We have been able to derive a lower bound based on this idea, but the upper bound of the option price remains to implement.

The second part of this lab was about implementing this algorithm, but for a swing option. This is mch more challenging as a swing option has not a unique exercise right, but multiple ones. We thus have 2 state variables, and we needed to modify the original version of the Longstaff-Schwartz algorithm to adapt to this case. We also implemented an upper bound of this swing option price by computing basically the M highest payoffs for each path, and a lower bound by an heuristic.

Note : Concerning LLM, they were not as proficient in R as they can be in other languages such as Python. Moreover, the .Rmd version of exercices solutions and of the slides available to us were great sources of information. As a result, we ended up using LLMs on really rare occasions and mainly focused on the files in IMT-GP-2024 and on the rdocumentation site.