Quantitative Finance

Least-Square Monte-Carlo

Patrick Hénaff

Version: 06 Feb 2025

Contents


```
library(lubridate)
library(fExoticOptions)
library(kableExtra)
library(ggplot2)
library(stats)
library(nleqslv)
library(reshape)
library(scales)
```

In this note we illustrate the Longstaff-Schwartz (Longstaff & Schwartz, 2001) Least-Square Monte Carlo (LSMC) algorithm for pricing American options.

Let $V(X_t, t)$ be the value at time t of an American option expiring at time T, where X_t is the value of underlying asset. The value of the option is determined by the optimal stopping time τ:

$$V(X_t) = \sup_{\tau: t \leq \tau \leq T} E(F(X_\tau) e^{-r(\tau - t)} | X_t)$$

where $F(X_t)$ is the payoff function. The stopping rule amounts to comparing, at each time step, the immediate exercise to the discounted expected value of the option, conditional upon the current value of the underlying asset:

$$E(V(X_{t+1})|X_t)$$

The main idea of the LSMC algorithm is to approximate this continuation value by a set of basis functions:

$$E(V(X_{t+1})|X_t)e^{-r\Delta t} = \sum_k \beta_k \phi(X_t)$$

The equation is estimated by linear regression, in a backward recursion starting at $T - \Delta t$. The data is provided by Monte-Carlo simulation. At time t, we know for each scenario k the value of the underlying asset X_t^k and the continuation value at step $t + \Delta t$. The regression minimizes the expected prediction error

$$\frac{1}{N}\sum_j^N \left(V(X_{t+\Delta t}^j)e^{-r\Delta t} - \sum_k \beta_k \phi(X_t^j) \right)^2$$

The algorithm involves two stages:

- In the first stage, a set of simulated paths is generated, and the continuation model () is evaluated by OLS for each time step, by backward recursion.
- In a second stage, a new set of paths is generated, and the continuation models are used to determine the optimal stopping time along each path, and the corresponding cash flow. The option value is the expected discounted value of these cash flows.

The calculation steps are next considered in details

1 Continuation value

There is a wide choice for the basis functions in equation (): polynomial functions, Laguerre polynomials, etc. In practice, the choice of basis functions does not seem to have a significant impact on the accuracy of the algorithm. Here, we implement polynomial, Laguerre and Hermite basis functions. Considering the wide range of values for the underlying asset, is seems useful to rescale to a range $[0, S_{max}]$ the input to the basis functions for the Hermite and Laguerre polynomials. We set $S_{max} = 3$. Rescaling does not seem to be needed for the polynomial basis.

```
Polynomial.basis <- function(degree, x) {
  value <- x^degree
  value
}

Hermite.basis <- function(degree, x) {
  if(degree == 0) {
    value = rep(1, length(x))
```

```
  } else if(degree == 1) {
    value= x
  } else if(degree==2) {
    value = x^2 - 1
  } else if(degree==3) {
    value = x^3 -3*x
  } else if(degree==4) {
    value = x^4 -6*x^2 + 3
  } else if(degree==5) {
    value = x^5 -10*x^3 + 15*x
  } else if(R==6) {
    value = x^6-15*x^4 + 45*x^2 - 15
  } else {
    stop(paste("Error, degree: ", degree, " is out of range"))
  }

  value
}

Laguerre.basis <- function(degree, x) {
  if(degree == 0) {
    value = rep(1, length(x))
  } else if(degree == 1) {
    value=-x+1
  } else if(degree==2) {
    value=0.5*(x^2-4*x+2)
  } else if(degree==3) {
    value=(1/6)*(-x^3+9*x^2-18*x+6)
  } else if(degree==4) {
    value=(1/24)*(x^4-16*x^3+72*x^2-96*x+24)
  } else if(degree==5) {
    value=(1/120)*(-x^5+25*x^4-200*x^3+600*x^2-600*x+120)
  } else if(R==6) {
    value=(1/720)*(x^6-36*x^5+450*x^4-2400*x^3+5400*x^2-4320*x+720)
  } else {
    stop(paste("Error, degree: ", degree, " is out of range"))
  }

  value
}

make.indep.vars <- function(ITM, state.var, type, scaled, degree) {
  nb.ITM <- sum(ITM)
  indep.vars <- matrix(0, nrow=nb.ITM, ncol=degree+1)
  if(scaled) {
```

```
    x <- rescale(state.var[ITM], to=c(0,3))
  } else {
    x <- state.var[ITM]
  }

  for(i in seq(0, degree)) {
    if(tolower(type) == "polynomial") {
      indep.vars[, i+1] <- Polynomial.basis(i, x)
    } else if(tolower(type) == "laguerre") {
      indep.vars[, i+1] <- Laguerre.basis(i, x)
    } else if(tolower(type) == "hermite") {
      indep.vars[,i+1] <- Hermite.basis(i, x)
    } else {
      stop(paste("Basis type: ", type, " is unknown."))
    }
  }
  indep.vars
}
```

The continuation value is the fitted value from the linear model.

```
continuation.value <- function(ITM, disc.future.value, state.var, type, scaled=T, degree=5) {

  indep.vars <- make.indep.vars(ITM, state.var, type, scaled, degree)

  y <- disc.future.value[ITM]
  model <- stats::lm(y ~ indep.vars - 1)

  cont.value <- vector("numeric", length=length(disc.future.value))
  cont.value[ITM] <- model$fitted.values

  # discounted continuation value cannot be lower than payoff
  cont.value[ITM] <- pmax(cont.value[ITM], payoff(state.var[ITM]))

  list(cont.value=cont.value, model=model)
}
```

Longstaff-Schwartz example

2 Scenario generation

We generate log-normal scenarios with antithetic variates.

```
GBMPathSimulator <-
  function(nb.steps, nb.paths, S0, mu, sigma, TTM, center = 'Mean') {
    delta.t <- TTM / nb.steps
    if (center == 'Mean') {
      Z <- rnorm(nb.steps * nb.paths,mean = 0,sd = 1)
      dim(Z) <- c(nb.steps, nb.paths)
      Z <- Z - mean(Z)
    } else {
      Z <- rnorm(nb.steps * nb.paths / 2,mean = 0,sd = 1)
      dim(Z) <- c(nb.steps, nb.paths / 2)
      Z <- cbind(Z,-Z)
    }

    path = (mu - sigma * sigma / 2) * delta.t + sigma * sqrt(delta.t) * Z
    if(length(S0) == 1) {
      S0 <- rep(S0, nb.paths)
    }
    S <- S0 * rbind(rep(1, nb.paths), exp(apply(path, 2, cumsum)))
    S
  }
```

3 LS backward recursion

The purpose of the backward recursion is to compute the continuation value model for each time step.

```
backward.recursion <- function(S) {
cash.flow <- vector("numeric", length=nb.paths)
exercise.time <- NA * vector("numeric", length=nb.paths)
PV <- vector("numeric", length=nb.paths)
cont.value <- vector("numeric", length=nb.paths)
continuation.model <- list()


# exercise at expiry

PV <- payoff(S[nb.steps+1,])

CRR.eu <- CRRBinomialTreeOption(TypeFlag = "ce", S=S.0, X=K, Time=TTM, r=r, b=r, sigma=sigma, n=
print(paste("CRR Eu price: ", round(CRR.eu, 2), "MC: ", round(exp(-r*TTM)*mean(PV), 2)))

ITM <- PV>0
cash.flow <- PV
exercise.time[ITM] <- TTM
```

```
for(t in nb.steps:1) {
  # browser()
  current.payoff <- payoff(S[t,])
  ITM <- current.payoff > 0
  if(t == 1) {
    cont.value = mean(PV * disc.factor)
    continuation.model[[t]] <- cont.value
  } else {
    if(sum(ITM) > 0) {
      res <- continuation.value(ITM, PV*disc.factor, S[t,], type=poly.type,
                                scaled=scaled, degree=degree)
      cont.value <- res$cont.value
      continuation.model[[t]] <- res$model
    } else {
      cont.value <- PV * disc.factor
    }
  }
  do.exercise <- current.payoff > cont.value
  cash.flow[do.exercise] <- current.payoff[do.exercise]
  PV <- PV * disc.factor
  PV[do.exercise] <- current.payoff[do.exercise]
  exercise.time[do.exercise] <- t-1
}

cashflow.idx <- !is.na(exercise.time)
disc <- exp(-r * delta.t * (exercise.time[cashflow.idx]))
option.value <- sum(disc * cash.flow[cashflow.idx])/nb.paths
option.sd <- sd(disc * cash.flow[cashflow.idx])/sqrt(nb.paths)

list(continuation.model=continuation.model, option.value=option.value, option.sd=option.sd)
}
```

4 Forward iteration

In the forward iteration stage, the continuation models are used to determine the optimal stopping time along each path.

```
forward.iteration <- function(continuation.model, S) {
  cash.flow <- vector("numeric", length=nb.paths)
  exercise.time <- NA * vector("numeric", length=nb.paths)
  immediate.payoff <- payoff(S.0)
  cont.value <- continuation.model[[1]]
```

```
  if(immediate.payoff > cont.value) {
    option.value <- immediate.payoff
    option.sd <- 0
  } else {
    is.alive <- rep(TRUE, nb.paths)
    for(t in 2:(nb.steps+1)) {
      is.alive.index <- which(is.alive)

      current.payoff <- payoff(S[t,is.alive])
      if(t < (nb.steps+1)) {
        indep.vars <- make.indep.vars(is.alive, S[t,], type=poly.type, scaled, degree)
        cont.value <- indep.vars %*% continuation.model[[t]]$coefficients
        # among the is.alive=TRUE list
        do.exercise <- current.payoff >= cont.value
      } else {
        do.exercise <- current.payoff > 0
      }
      exercise.index <- is.alive.index[do.exercise]
      cash.flow[exercise.index] <- current.payoff[do.exercise]
      exercise.time[exercise.index] <- t-1
      is.alive[exercise.index] <- FALSE
    }

    cashflow.idx <- !is.na(exercise.time)
    disc <- exp(-r * delta.t * (exercise.time[cashflow.idx]))
    option.value <- sum(disc * cash.flow[cashflow.idx]) / nb.paths
    option.sd <- sd(disc * cash.flow[cashflow.idx])/sqrt(nb.paths)
  }
  list(option.value=option.value, option.sd=option.sd)
}
```

5 Example

As an example, we price an American call by LSMC and compare the result to the value obtained from the CRR model. This example illustrates the importance of running the forward iteration stage, which yields a more accurate estimate of the option value than the backward recursion.

```
nb.paths <- 200000
nb.steps <- 100
r <- .03
sigma <- .3
S.0 <- 100
TTM <- 1
delta.t <- TTM / nb.steps
```

```
disc.factor <- exp(-r*delta.t)
K <- 110
payoff <- function(S) {
  pmax(S-K, 0)
}
scaled <- TRUE
poly.type <- "polynomial"
degree <- 3
```

We execute the two stages of the LSMC algorithm, and price the same option with the Cox-Ross-Rubinstein binomial tree:

```
S <- GBMPathSimulator(nb.steps, nb.paths, S.0, r, sigma, TTM)
stage.1 <- backward.recursion(S)
```

```
## [1] "CRR Eu price:  9.27 MC:  9.26"
```

```
option.value.back <- stage.1$option.value
option.sd.back <- stage.1$option.sd

S <- GBMPathSimulator(nb.steps, nb.paths, S.0, r, sigma, TTM)
stage.2 <- forward.iteration(stage.1$continuation.model, S)
option.value.fwd <- stage.2$option.value
option.sd.fwd <- stage.2$option.sd
CRR.value <- CRRBinomialTreeOption(TypeFlag = "ca", S=S.0, X=K, Time=TTM,
                                   r=r, b=r, sigma=sigma, n=50)@price
```

The results are summarized below, they illustrate the importance of running the forward iteration stage, which yields a more accurate estimate of the option value than the backward recursion.

Table 1: Pricing an American option, maturity 1 year. $S_0 = 100$, $K = 110$. The backward price is the expected continuation value at $t = 0$. The forward price is expected discounted payoff at the optimal stopping time along each path.

	$E(S_T)e^{-rT}$	CRR Binomial	LSMC Backward	LSMC Forward
Mean	99.99	9.23	9.54	9.22
SD	0.07	NA	0.05	0.05

References

Longstaff, F., & Schwartz, E. (2001). Valuing American Options by Simulation: A Simple Least-Squares Approach. *Review of Financial Studies*, *14*(1), 113–147.