BAROUX
Timothée

Monte Carlo project report
Bermudean Options

Our goal in this project is to compute the price of a Bermudean put option, where the payoff is $(K - X_T)^+$, where X_T is a Black-Scholes diffusion.

We will use a nested Monte Carlo approach, with an increasing number of additional techniques, and then a finite difference method.

I The nested approach.

Instead of simulating one sequence of $(X_1, \dots, X_T)$, computing the value of the put option V_0 and then averaging over a large number of simulation, as is the classical Monte Carlo method, here we choose a parameter b, and from X_0 we simulate b times X_1. For each simulated X_1, we simulate b X_2, and so on until X_T.

From this point there are several ways to compute an estimator of the option.
The first estimator will consist of calculating the simulated payoffs resulting from each X_T, then taking the maximum between the payoff at $X_{t_{m-1}}$ and

the average of the discounted payoff at all X_T that are issued from this $X_{t_{m-1}}$.
We then repeat this process until we reach X_0, at which point we get our first estimator of the value of the option.
This estimator is biased high, thus it is called the high estimator.

The second estimator follows the same principle, but instead of taking the maximum between the payoff at time t_{m-1} and the average of the discounted following payoffs at time T, we split the stack of payoffs at time T in two: call $\hat{v}_1$ the average discounted payoff of the first half and $\hat{v}_2$ the average discounted payoff of the second half. Then if the payoff at $X_{t_{m-1}}$ is greater than $\hat{v}_1$, we take this payoff as the estimator at this point, and if it is smaller than $\hat{v}_1$, we take $\hat{v}_2$ as the estimator.
We then repeat this process until we reach X_0, it gives us a second estimator.
This estimator is biased low, thus we call it the first low estimator.

The third estimator goes further than the second: instead of splitting the stack of payoffs issued from one $X_{t_{m-1}}$ in two halves, it first splits them in two stacks of one and all but the one. $\hat{v}_1$ will then be the average discounted payoff of the stack of all but one value of X_T (for this $X_{t_{m-1}}$) and $\hat{v}_2$ will be the discounted payoff of the remaining X_T. We compute an estimator as we did previously, but then we

average this method over all possible choices of the solitary X_T. This average is our estimator at the point $x_{t_{m-1}}$. By repeating this process until we reach X_0, we obtain our third estimator of the value of the Bermudean option. This estimator is biaised low, thus we call it the second low estimator; as we will see, it is better performing than the first low estimator and we will eventually discard the first and therefore call this one the low estimator.

I. 1) Naive method

The simplest way of computing these estimators is to compute all nodes, that is to say X_0, then b values for X_1, then b^2 values for X_2, and so on until X_T, and from there compute the estimators.

We then take the average value of a large number (n) of estimators, as in the classical Monte Carlo approach, which gives us confidence intervals at 95%.
By taking the lower bound of the confidence interval of the high estimator and the higher bound for the low estimator, we get a smaller confidence interval, although we can only say it contains the true value of the option with 90% probability.

This method requires that the biais/variance ratio remains low: if the biais is too great, then the confidence intervals can be empty, as we will see later.

We does not have a way of estimating the error resulting from the bias, so our confidence intervals are only valid when it is not too high.
As the bias tends to zero when b tends to infinity, this means we will try to have a b of at least 10 or for our computations, when this allows us to obtain results with a reasonable computational time.

Numerical results: with the naive method:
$m = 3$ (the number of time steps); $r = 0,1$; $\sigma = 0,2$;
$b = 50$, $n = 10$; $X_0 = k = 100$; $dt = 1$
(maturity of 3 years since $m = 3$ and $dt = 1$),
we have a computational time of 110 s;
the estimator is 5,34, with a 28% error margin
(i.e a confidence interval at 90% of $[4,57; 6,11]$.
The first low estimator had a 43% error margin
while the second low estimator had a 29% error margin.
By computing individual times, we see that the first low estimator took 2 s to compute while the second took 95 s to compute, however we will use techniques to reduce computing time so we will keep the better performing one.

The variance here is very high so we can afford to reduce b as it will reduce computational time even if we increase n.
With $b = 10$ and $n = 50$ (same parameters otherwise),
we have a computation time of 3 s;
the estimator is 4,85 with a 17% error margin
(i.e a confidence interval of $[4,43; 5,28]$.
Once again the first low estimator had a 43% error

margin while the second low estimator had a 25% error margin; they took 0,2 s and 2,7 s respectively to compute.

Here we see that while the bias must not be too high, it cannot be too low either : a b too great drastically increases computational times and does not improve the results! Worse, having some bias was actually an advantage for the second simulation, as it reduced the final error margin, while this did not happen meaningfully the first time.

A first improvement of this method is the use of a recursive function : while the computations are the same in the end, it reduces the storage requirements of the method by forgetting the values of the successors once it knows the value at a node X_{t_i}.

The numerical results with this improvement are similar to those of the naive implementation.

I. 2) Pruning.

There are several methods to reduce the computational time needed for these estimators.
The first two consist of determining when the value at a node X_{t_k} of the payoff $(K - X_{t_k})^+$ is "too low," and generating a single successor instead of b when it is the case.
As these methods are much simpler to implement when we use a recursive function, we will discard the naive, non-recursive approach from now on.

To determine if the value of the payoff is "too low", we have two possibilities.
One option is to see if this payoff is zero.
This gives us our first pruning method.

The second option is to compare it with the value of an european option starting and t_k with value X_{t_k} and expiring at $t_m = T$: If the payoff is lower than the value of the option then it is too low and it is optimal to not exercise the option.

Numerical results: (We always keep the same values for x, σ, m, dt, X_0, K)
with $b = 50$ and $n = 500$

The first pruning method takes 21 s to compute a value of 4,91 with an error margin of 25%
and a confidence interval at 90% of [4,30; 5,52]

The second pruning method takes 17 s to compute a value of 5,12 with an error margin of 26%
and a confidence interval at 90% of [4,46; 5,79].

A third method to reduce computation times is the removal of the last step: at time t_{m-1}, instead of generating b X_T from each $X_{t_{m-1}}$, we can take the maximum between the payoff $(k - X_{t_{m-1}})^+$ and the value of an european option starting at t_{m-1} on $X_{t_{m-1}}$ and expiring at T, for each value of $X_{t_{m-1}}$.

Numerical results: $b = 25$ $n = 50$ for the last step removal
time: 15 s estimator: 4,82 error: 12% interval: [4,51; 5,13]

with $b=50$, $n=10$, we get a time of 12s for a result of 4,36 and an error of 14%.

with $b=10$, $n=50$ we get a time of 2,3s for a result of 4,92 and an error of 17%.

When compared to the method without pruning, the first two are immensely faster, allowing for a much greater n while retaining a large b. However the last step removal only performed better than the initial one when b was great, which is logical as the size of the last step depends only on b.

As the second pruning method performs better than the first, we will only keep using the second and not the first. We will keep comparing the results obtained with and without the removal of the last step.

II Variance reduction techniques.

We are going to use two variance reduction techniques: the antithetic variate and the control variate. We will use both individually and together, with the pruning method and without them, and compare the results.

The control variate is an european option with initial value x_0 expiring at n.

Numerical results:

Antithetic variate: • $b = 50$, $n = 10$:

time = 143 s, estimator = 5,12, error = 8 %

• $b = 10$, $n = 50$

time = 36 s, estimator = 4,97, error = 7 %

Antithetic variate with pruning: $b = 50$, $n = 150$:

time = 12 s, estimator = 5,2, error = 27 %.

Antithetic variate with last step removal: • $b = 50$, $n = 10$:

time = 22 s estimator = 4,91 error = 11 %

• $b = 10$, $n = 50$:

time = 4 s estimator = 5,07 error = 7 %

Antithetic variate with pruning and last step removal:

$b = 100$ $n = 500$

time = 22 s estimator = 5,07 error = 12 %.

Control variate: • $b = 50$, $n = 10$:

time = 118 s estimator = 5,00 error = 2,5 %.

• $b = 10$ $n = 50$:

time = 4,1 s estimator = 4,75 error = 0,7 %.

We see here the problem when b is not high enough: the confidence intervals of the high and low estimator tend to be disjointed in this case, as the bias is too high in relation to the variance, thus making the results innacceptable.

We would have the same problem using the control variate with the antithetic variate and no pruning method: either the results would be with too high bias to be acceptable or the computational times would be too high.

Control variate with pruning : $b = 50$ $n = 100$
time = 3s estimator = 5,02 error = 16%.
Control variate with antithetic variate and pruning :
$b = 50$ $n = 100$
time = 10s estimator = 5,18 error = 8%.

The control variate with pruning takes 10 s to compute with $b = 50$ and $n = 250$ and gives us an error margin of 9%, from which we conclude that the best method among those we have currently is the control variate with antithetic variate and pruning.

III Finite difference method.

We can also compute the value of the option using a finite difference method.

We discretize the equation ; with $u(t, x_t)$ the value of the option

$$\partial_t u + \left(r - \frac{\sigma^2}{2}\right) x \partial_x u + \frac{1}{2} \sigma^2 x^2 \partial^2_{xx} u - ru = 0$$

We use the implicit method, therefore our scheme is stable ; and the error is majored by

$$C(dx^2 + dt)$$; we ignore the C

and still obtain an approximate value of the error.

at $t = T$, $u(T, x) = g(x)$ where g is the payoff.

This gives us an estimator of the value:
with $dx = 0,1$; $dt = 0,1$; we have an estimated value of 4,96 and an error of 22% for a time of 2 s.