Master 203 in Financial Markets – Computational Finance

Quasi Monte Carlo pricing of path dependent basket options

To Kaiza Amouh

Julien Baudin, Corentin Leblond
17th May 2020

Table of Contents

Introduction

According to the latest data from the BIS (Bank for International Settlement), the total amount (in $ notional equivalent) of derivatives contracts traded was around 640 Trillion for the first half of 2019. Derivatives serve mostly hedging, replication and arbitrage purposes for different actors that are the market makers, Institutionals but also retail investors. They are used for themselves or through structured products and funds. Pricing these products represent both an opportunity for sellers to be competitive but also a great risk. In that spirit, practitioners must be able to produce fast pricing as well as metrics to be able to gauge the accuracy of the price obtained. For vanilla products, the existence of close formulae gives an unbiased benchmark, and there is no question of accuracy. On the other hand, when considering less traditional contracts, such as path dependent option the lack of close formulae creates a need for both pricing techniques and the ability to assess the pricing accuracy of these models while being able to minimize the time of computation.

The following report is an attempt to answer part of that question. In the context of the master 203 in Financial Markets and the course of Computational Finance, this report will explore the Monte Carlo pricing of path dependent Basket options (in the case of Bermudan exercises and American). Pricing path dependent options can be done today with three main approaches. The binomial tree that is highly time consuming, the machine learning algorithm (for example through neural networks – a topic that has been explored in the Machine Learning project of the Master 203) and finally through the Monte Carlo approach.

The Monte Carlo technique relies on the Central Limit Theorem in the sense that for n independent and identically distributed (iid) experiences and when n is sufficiently high, the empirical mean of these experiences converges in law toward the true mean. Therefore, if one can simulate n different paths for the underlying index of a basket option, then one can estimate the final payoff at maturity (for the European case), averaging and discounting to get the price of such option. However, the variance among all simulated prices can be quite high which could make the price unreliable. That is why (in the risk perspective mentioned before), one has to estimate that empirical variance and if it is statistically possible build a confidence interval in order to find what would be the optimal number of simulations that will allow, for a given level of uncertainty, consider the price as a reliable one.

While finding the optimal number of simulations will be quite easy, this quantity can be very large meaning that the computation time will be quite long making its usage impossible in practice (no client would wait for two hours before having a price). Therefore, we will explore some variance reduction techniques in order to reduce the variance and the number of simulations needed to enter our confidence interval. One will implement:

- Low discrepancy random number generation (Van Der Corput and Sobol sequences)
- Static Control Variate approach
- Antithetic variable
- Mixing the three previous techniques

Finally, one will implement the Longstaff Schwartz approach to price path dependent options in the American and Bermudan framework.

To keep things standard, the framework will be a N dimensional Black Scholes diffusion where the d components of the Basket are (possibly) correlated. One will take constant covariance and correlation matrix, a constant and unique risk-free rate.

All computations will be done in C++.

User Guide

Throughout this user guide, we display pieces of code to generate Monte Carlo simulations to value European and American basket call option. All the graphs and tables are obtained thanks to the code shown below.

Inputs in Main.cpp

All numerical parameters for volatility, rate, time, correlation should be input in yearly values.

Variable Name	Description
startTime	Starting point of the simulation in time
endTime	Ending point of the simulation in time
Nb_assets	Number of assets in the basket
Nbsteps	Number of time steps within each simulation
rate	Risk-free rate
K	Strike Price
Spot_vector	Initial spot vector
Sigma_vector	Volatility vector
Correl_mat	Correlation matrix
Weights_mat	How each asset weights in the basket

We show an example of the above inputs:

```
double startTime = 0.;
double endTime = 1.;
size_t nbsteps = 252;
size_t Nb_Assets = 3;
double rate = 0.05;
double K = 100;

std::vector<std::vector<double>> Spot_vector = {{100},{120},{80}};

std::vector<std::vector<double>> Sigma_vector = {{0.25},{0.20},{0.15}};

std::vector<std::vector<double>> Correl_mat = {{1,-0.2,0.4},
                                               {-0.2,1,0.4},
                                               {0.4,0.4,1}};

std::vector<std::vector<double>> Weights_mat ={{0.3,0.5,0.2}};
```

General Construction of a Monte Carlo Simulation for European Payoff

In order to build the Monte Carlo simulation, one always needs at least to initialize the following objects:

- A uniform generator
- A normal generator

- A gaussian vector
- A random process
- A payoff
- A Monte Carlo object finally

We show an example of how to build all the above objects to run a standard pricing:

```
//RANDOM NUMBER GENERATION
UniformGenerator* ugen = new EcuyerCombined();
Normal* ngen = new NormalBoxMuller(ugen, 0., 1.);
GaussianVector* corrG = new GaussianVectorCholesky(ngen, Sigma, Correl, CovarMatrix);

//RANDOM PROCESS
RandomProcess* path = new BSEulerND(corrG,spot_m,rate);

//PAYOFF
PayOffBasket* bsktcall = new PayOffBasketCall(Weights, spot_m,K);

//MONTE CARLO
double tolerated_error = 0.01;
size_t CompMC_Nb_Simulation = 2000;

EuropeanBasket MC(CompMC_Nb_Simulation, bsktcall, path);

//First Run: Companion MC to determinate the optimal number of simulation

clock::time_point startComp = clock::now();
MC.Simulate(startTime,endTime,nbsteps);
clock::time_point endComp = clock::now();
clock::duration execution_timeComp = endComp - startComp;


MC.OptimalNbSimul(tolerated_error);

size_t Optimal_Nb_Simulation = MC.GetNbSimul();
```

In the above example, we show the steps to run a first Monte Carlo simulation that is used to estimate the minimum number of simulations required to make the price enter into a 99% confidence interval with an error defined by the user (here 0.01). We can afterwards print the followings outputs as follows:

```
std::cout << "NB SIMULATION COMP: " << CompMC_Nb_Simulation << std::endl;
std::cout << "ERROR TOLERATED: " << tolerated_error << std::endl;
std::cout << "PRIX COMP: " << MC.GetPrice() << std::endl;
std::cout << "Variance COMP: " << MC.GetVariance() << std::endl;
std::cout << "NB SIMULATION OPTI: " << Optimal_Nb_Simulation << std::endl;
std::cout << "EXECUTION TIME COMP: " << std::chrono::duration
<double,std::ratio<1>> (execution_timeComp).count() << std::endl;
```

This will print the following kind of result:

```
NB SIMULATION COMP: 2000
ERROR TOLERATED: 0.01
PRIX COMP: 12.5194
Variance COMP: 171.685
NB SIMULATION OPTI: 11339627
EXECUTION TIME COMP: 65.7965
```

Note that the execution time is in seconds.

After this first run, the minimum number of simulations required is automatically stored so that one can directly run again the "simulate" method of the MC object. It will run with the optimal number of simulations. This corresponds to the following line:

```
MC.Simulate(startTime,endTime,nbsteps);
```

Construction of a Simulation for European Payoff with Variance Reduction techniques

One can either apply one of the following techniques or cumulate them together:

- Quasi random sequence
- Antithetic random variable
- Static Control Variate

We are going to show how to implement them individually.

Quasi Random Sequence

1. Sobol Sequence

Two quasi random sequences are available: Van Der Corput and Sobol ones.

In order to use the Sobol Sequence, one has to change "Uniform Generator" line as follows:

```
UniformGenerator* ugen = new Sobol();
```

Instead of:

```
UniformGenerator* ugen = new EcuyerCombined();
```

And repeat the code shown previously.

Regarding the algorithm implementation, we started our work from (Press, Teukolsky, Vetterling, & Flannery, 2002).

2. Van Der Corput Sequence

To use this sequence and implement it with the Box Muller Algorithm, one must create two Van Der Corput sequences with two different bases that are prime numbers:

```
UniformGenerator* ugen = new VanDerCorput(2,1);
UniformGenerator* ugen2 = new VanDerCorput(3,1);
Normal* ngen = new NormalBoxMullerVDC(ugen,ugen2, 0., 1.);
```

Antithetic Random Variable

The antithetic technique is implemented for the generation of correlated gaussian vectors. To use it, two rows must be replaced compared to the reference code we gave.

First, for the random process, one must write:

```
RandomProcess* path = new BSEulerNDAntithetic(corrG,spot_m,rate);
```

Instead of:

```
RandomProcess* path = new BSEulerND(corrG,spot_m,rate);
```

Second, one must also write:

```
EuropeanBasket_Antithetic MC(CompMC_Nb_Simulation, bsktcall, path);
```

Instead of:

```
EuropeanBasket MC(CompMC_Nb_Simulation, bsktcall, path);
```

Otherwise, the reference stays the same.

Static Control Variate

For this variance reduction technique, one should add a second payoff object for the control variate and compute its closed formula price before starting the Monte Carlo simulation.

First, one must add the 3 next lines to create the control variate and compute the closed formula price:

```
PayOffBasket* bsktcallCV = new PayOffControlVarBasketCall(Weights, spot_m,K);
ClosedFormulaBasketCall* CFbaskt = new ClosedFormulaBasketCall
(Weights, spot_m, CovarMatrix, K,rate, endTime);
double CV_closedprice = CFbaskt->operator()(spot_m,df);
```

Second, the Monte Carlo object must be changed to:

```
EuropeanBasket_controlvariable MC(CompMC_Nb_Simulation,
bsktcall,bsktcallCV, path,CV_closedprice);
```

Otherwise, the reference codes remain the same.

Use Multiple Techniques

One can easily use two or three of the previous techniques together, keeping the same structure of code. However, to use at the same time Antithetic and Control Variate, one must write:

```
EuropeanBasket_Antithetic_CV MC(CompMC_Nb_Simulation,
bsktcall,bsktcallCV, path,CV_closedprice);
```

Note that the Random process is still the same as shown in the Antithetic part.

Case where the Covariance Matrix is not invertible

To ensure that the generation of gaussian vectors is done properly, we coded two algorithms, one does the Cholesky decomposition into a lower triangular matrix, while the other uses the diagonalization process with eigenvalues and normalized eigenvectors. The latter can be used even when the covariance matrix is not invertible. The reference code must be modified as follows to compute its determined and then create the appropriate object:

```
UniformGenerator* ugen = new Sobol();
Normal* ngen = new NormalBoxMuller(ugen, 0., 1.);
GaussianVector* gvec;
double determinant = determinantOfMatrix(CovarMatrix,CovarMatrix.nb_rows());
if(determinant == 0.)
{
    gvec = new GaussianVectorDiag(ngen,  Sigma,  Correl, CovarMatrix);
}
else
{
    gvec = new GaussianVectorCholesky(ngen,  Sigma,  Correl, CovarMatrix);
}
```

Results for European Payoff

For all the following results, we show them with two different sets of inputs: Input 1 and Input 2.

Input 1 is the following: we value a basket call option containing 3 assets:

```
double startTime = 0.;
double endTime = 1.;
size_t nbsteps = 252;
size_t Nb_Assets = 3;
double rate = 0.05;
double K = 100;

std::vector<std::vector<double>> Spot_vector = {{95},{120},{80}};

std::vector<std::vector<double>> Sigma_vector = {{0.25},{0.20},{0.15}};

std::vector<std::vector<double>> Correl_mat = {{1,-0.2,0.4},
                                               {-0.2,1,0.4},
                                               {0.4,0.4,1}};

std::vector<std::vector<double>> Weights_mat ={{0.3,0.5,0.2}};
```

Input 2 is the following: we value a basket call option containing 4 assets with a different covariance structure:

```
double startTime = 0.;
double endTime = 1.;
size_t nbsteps = 252;
size_t Nb_Assets = 4;
double rate = 0.05;
double K = 100.;

std::vector<std::vector<double>> Spot_vector = {{105.},{100.},{100.},{98.}};

std::vector<std::vector<double>> Sigma_vector = {{0.4},{0.32},{0.38},{0.24}};

std::vector<std::vector<double>> Correl_mat = {{1.,-0.25,0.4,0.15},
                                               {-0.25,1.,-0.1,0.2},
                                               {0.4,-0.1,1.,-0.2},
                                               {0.15,0.2,-0.2,1.}};

std::vector<std::vector<double>> Weights_mat ={{0.2,0.35,0.25,0.2}};
```

No Variance Reduction

This is our benchmark simulation for the European basket call with the following inputs:

We obtain for Input 1 and 2:

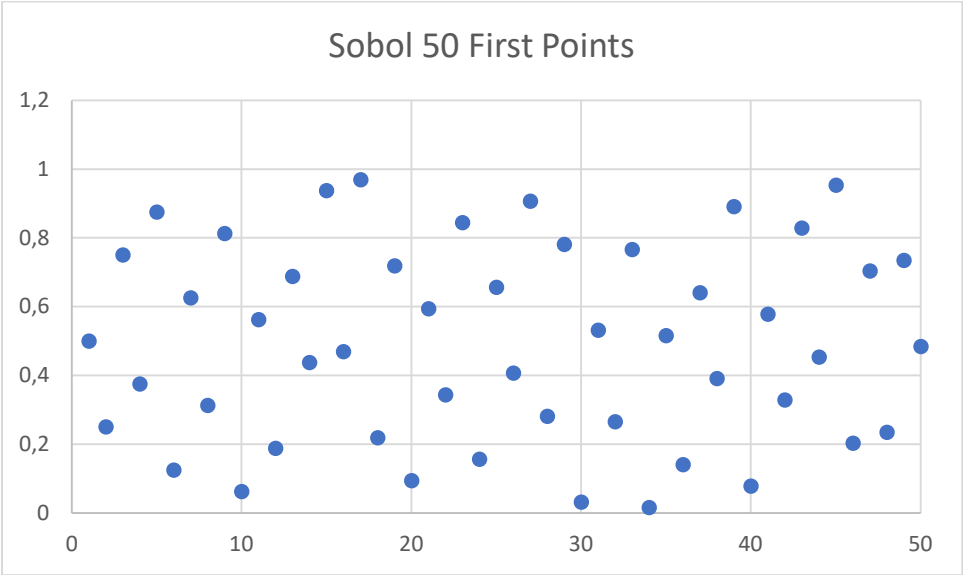
Input 1	**No Variance Reduction**
Number of Simulations	2 000
Error Tolerated	0.01
Price	12.52
Variance	171.00
Required Number of Simulations	11 339 627
Execution Time (Seconds)	66

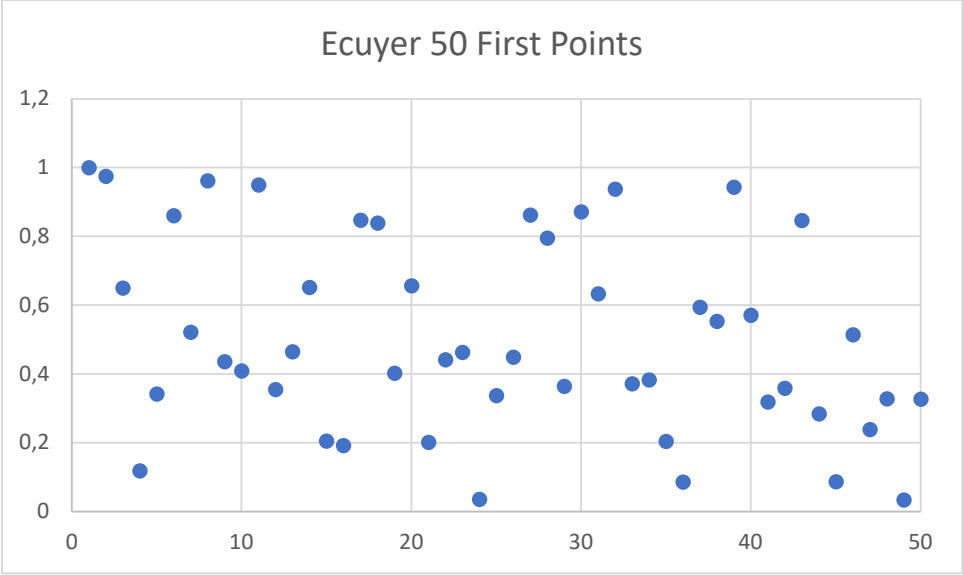
Input 2	**No Variance Reduction**	**No Variance Reduction**
Number of Simulations	10 000	3000
Error Tolerated	0.01	0.01
Price	10.5	10.18
Variance	223.4	211.6
Required Number of Simulations	14 755 310	13 977 378
Execution Time (Seconds)	432	134

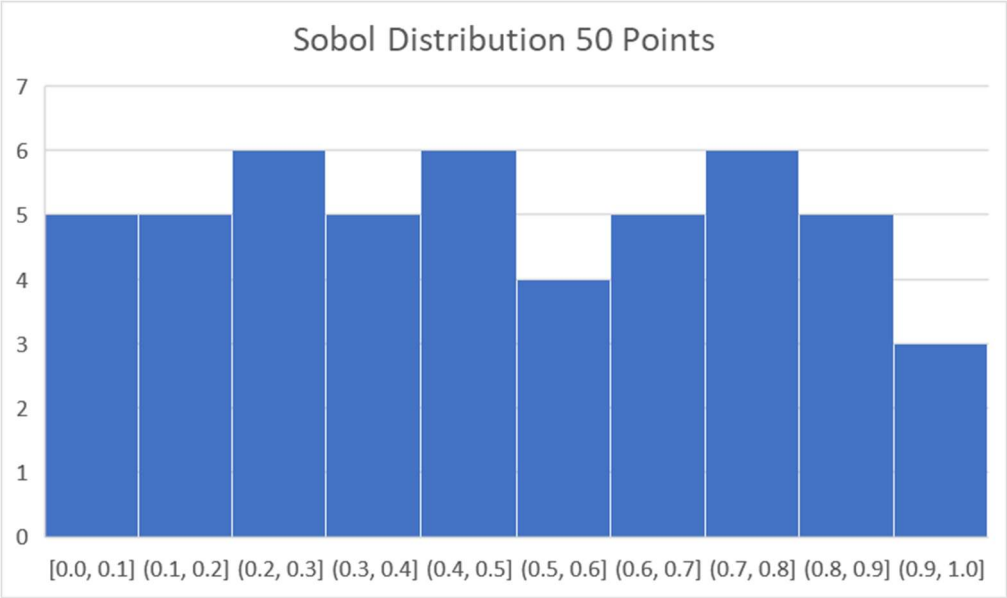
Quasi-Random Sequence

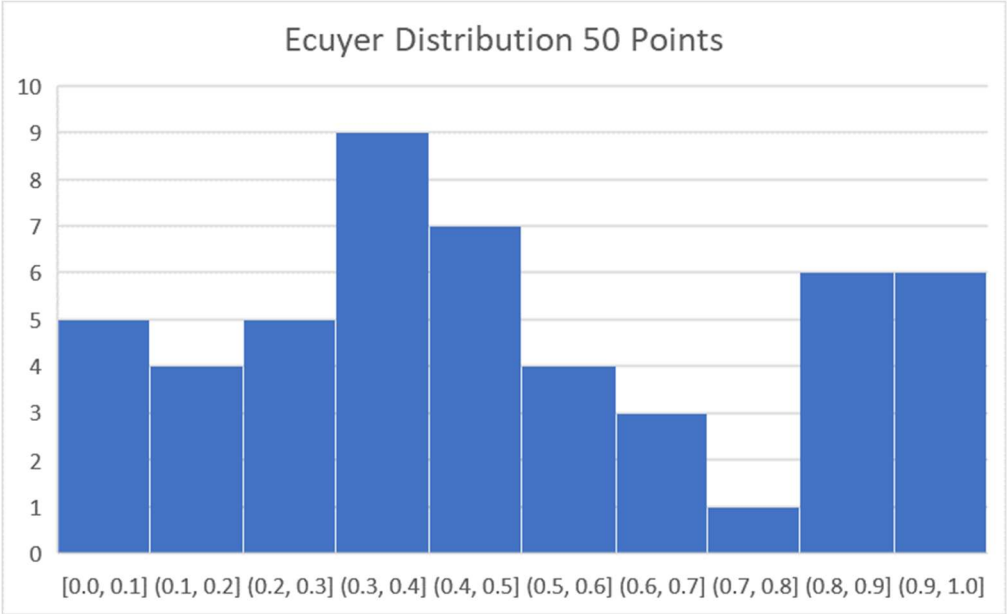
1. Sobol Sequence

The good properties of the Sobol sequence are shown below, we compare it to the Ecuyer generator. On the next graphs, we can compare the first 50 points of each sequence:

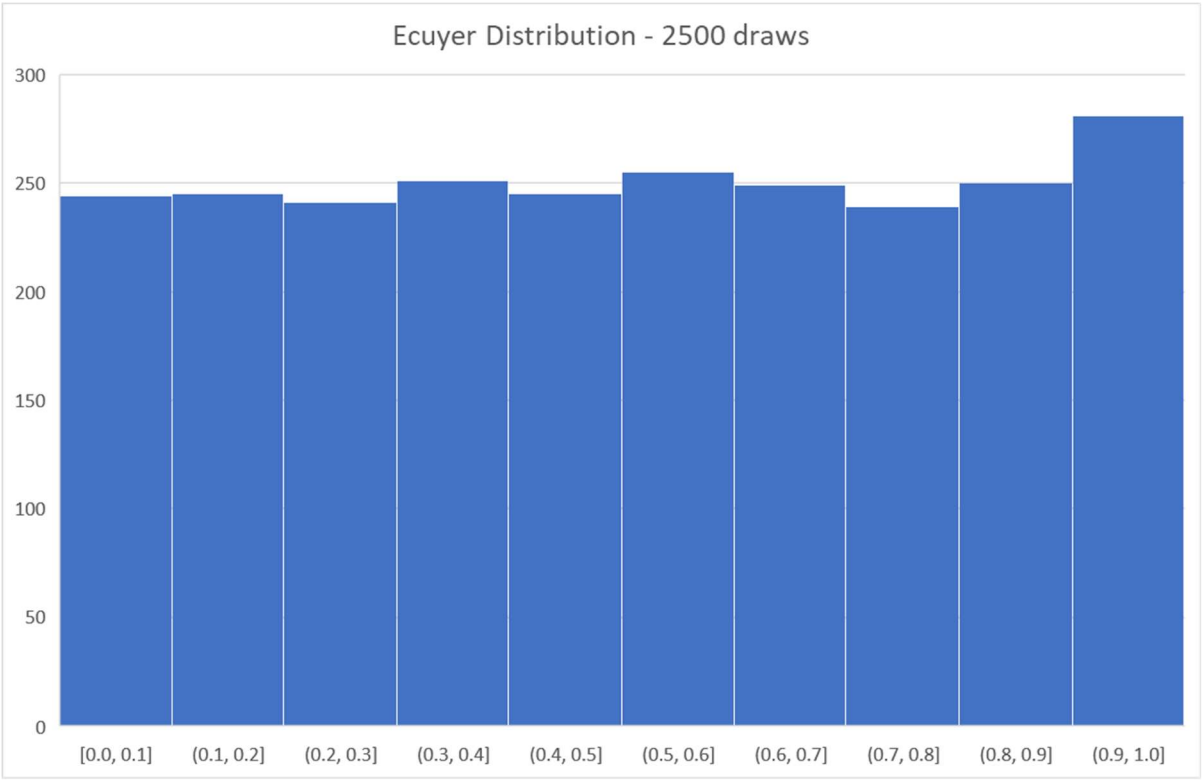


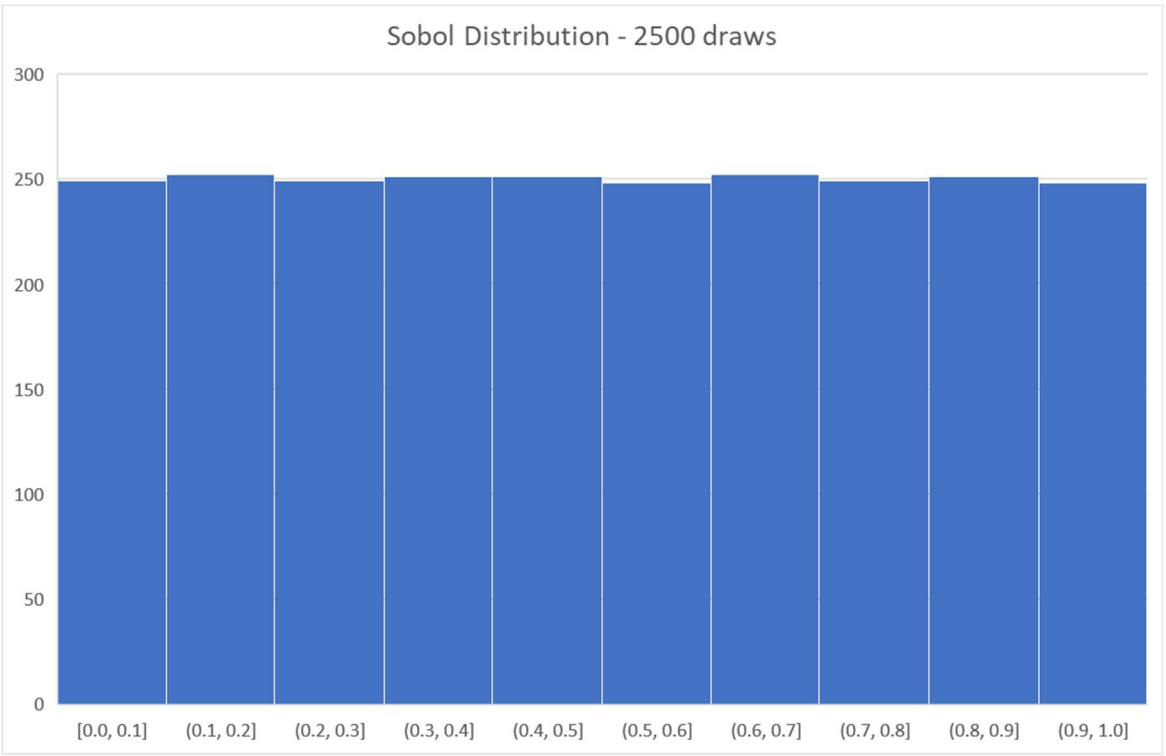






We observe that, even on small samples, the Sobol sequence replicates better the uniform distribution. Now, on a larger sample (2500), we obtain:





We can see that this property is also true for larger samples and that the Sobol sequence fits more a “true” uniform distribution.

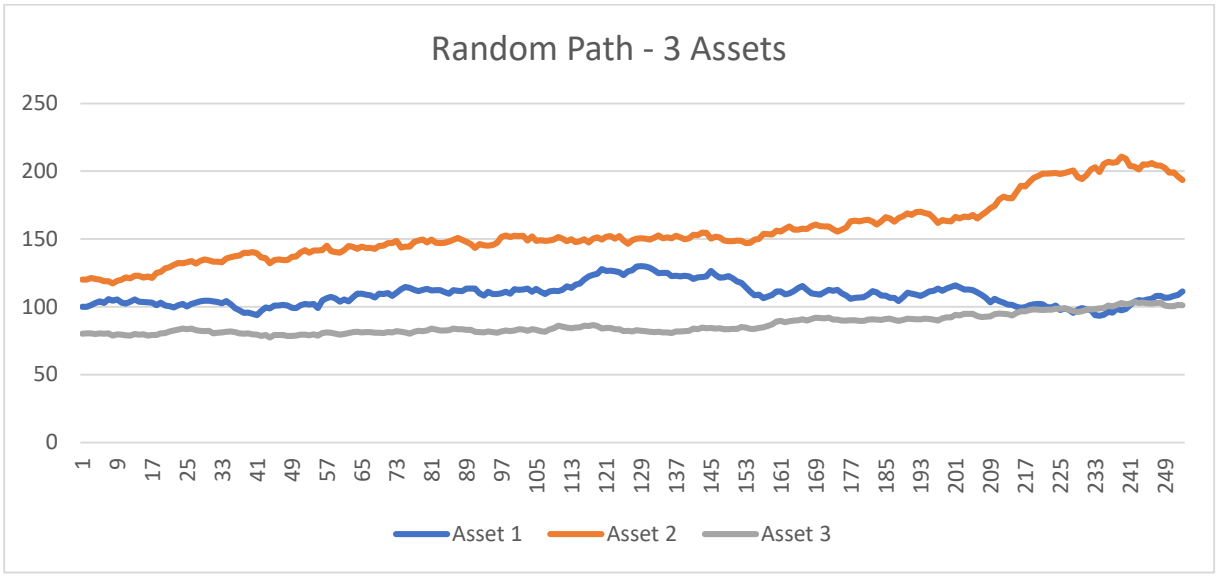
Input 1	Sobol Sequence
Number of Simulations	2 000
Error Tolerated	0.01
Price	10.91
Variance	88.03
Required Number of Simulations	5 759 728
Execution Time (Seconds)	66

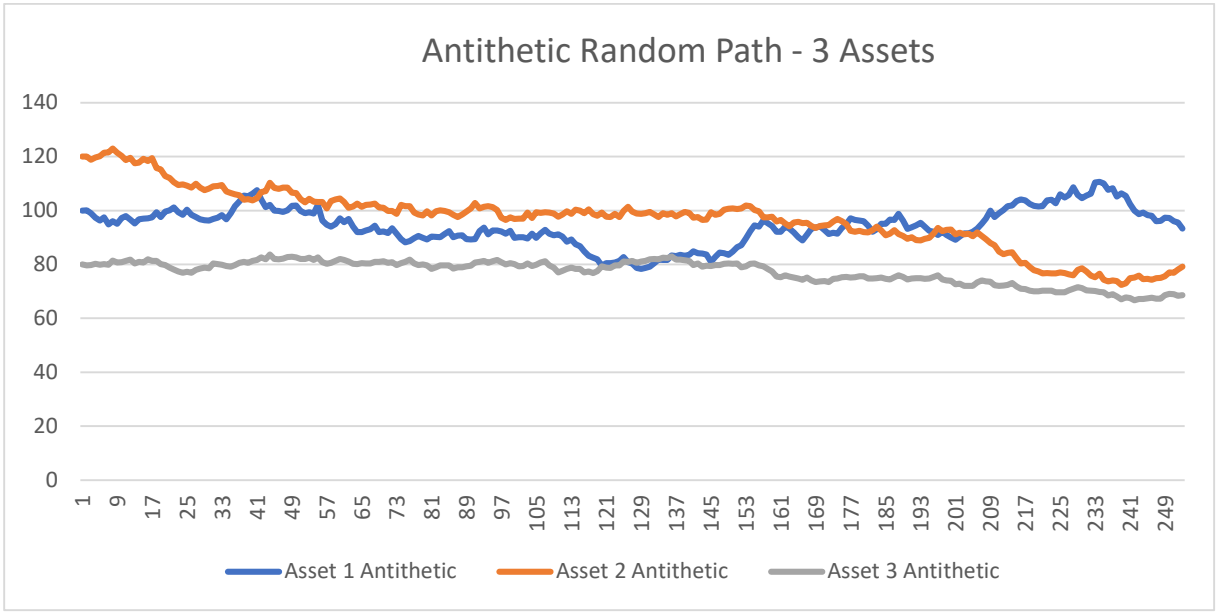
Input 2	Sobol Sequence	Sobol Sequence
Number of Simulations	10 000	3000
Error Tolerated	0.01	0.01
Price	1.24	1.24
Variance	0.72	0.72
Required Number of Simulations	47 692	47 727
Execution Time (Seconds)	432	144

We can see that only applying the Sobol Sequence to generate Uniform distribution has a significant impact not only on the estimated variance, but also on the option price. For Input 1, we almost divide by two the number of simulations required to enter the confidence interval. Regarding Input 2, the results are even more important: we go from 14 million to 50 thousand. Interestingly, we also see that the price is very different from the one obtained we Ecuyer generator. We tried to increase to 10 000 the number of simulations to see if this result was just coming from the low number of simulations. It seems not. We conclude that the Sobol Sequence ability to replicate the Uniform distribution can improve the pricing. This improvement depends on the inputs and the product priced. We can divide the number of simulations by two, sometimes more, and sometimes maybe less.

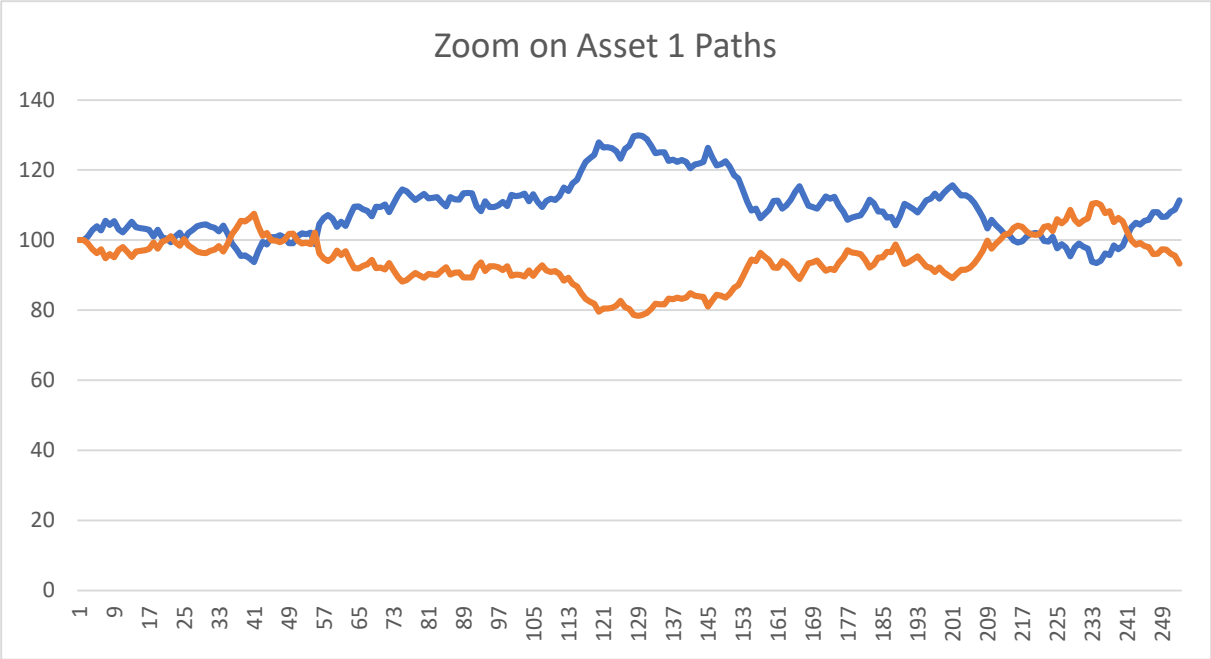
Antithetic Random Variable

On the next graphs, for a basket of 3 assets, we can see an example of a path generated for it and its corresponding antithetic path.





Zooming on asset 1, we can see that that the Antithetic path follows exactly the opposite path, and same thing for asset 2 and 3, which means the correlation between assets remains the same:



We obtain the following results:

Input 1	Antithetic
Number of Simulations	2 000
Error Tolerated	0.01
Price	12.42
Variance	20.45
Required Number of Simulations	1 350 579
Execution Time (Seconds)	42

Input 2	Antithetic
Number of Simulations	3000
Error Tolerated	0.01
Price	10.03
Variance	52.35
Required Number of Simulations	3 457 915
Execution Time (Seconds)	78

The Antithetic technique shows very good results regarding the variance of our simulations. For the variance and both inputs, it reduces by around 80% the number of simulations required. Another good aspect of using this technique is that it also reduces the algorithm speed. Indeed, we decrease the pricing speed by almost 40%.

We studied this technique to confirm we were going in the right direction while implementing it thanks to (Sigman, 2007).

Static Control Variate

Using this technique led us to the following results:

Input 1	**CV**
Number of Simulations	2 000
Error Tolerated	0.01
Price	13.01
Variance	7.30
Required Number of Simulations	491 893
Execution Time (Seconds)	67

Input 2	**CV**
Number of Simulations	3 000
Error Tolerated	0.01
Price	12.07
Variance	19.85
Required Number of Simulations	1 311 301
Execution Time (Seconds)	145

From the above two tables, we can conclude that the use of this Control Variate for the basket call option enables a drastic variance reduction, which is even larger than the one allowed by the Antithetic technique. However, there is no improvement of the algorithm speed as it is the case with the Antithetic Random Variable. Now the estimated variance is reduced by more than 90% for both inputs.

Quasi Monte Carlo and Antithetic Random Variable

Combining the two methods, we obtain:

Input 1	**Sobol and Antithetic**
Number of Simulations	2 000
Error Tolerated	0.01
Price	11.01
Variance	7.92
Required Number of Simulations	522 813
Execution Time (Seconds)	45

Input 2	**Sobol and Antithetic**
Number of Simulations	3 000
Error Tolerated	0.01
Price	1.23
Variance	0.015
Required Number of Simulations	988
Execution Time (Seconds)	53

For both inputs, combing Sobol Sequence and Antithetic Random Variable shows better result than using each technique individually. We now decrease the variance by more than 95%.

Quasi Monte Carlo and Control Variate

Instead of Antithetic variable, we now combine Quasi Monte Carlo with the Control Variate and obtain:

Input 1	Sobol and CV
Number of Simulations	2 000
Error Tolerated	0.01
Price	13.71
Variance	10.57
Required Number of Simulations	698 081
Execution Time (Seconds)	67

Input 2	Sobol and CV
Number of Simulations	3 000
Error Tolerated	0.01
Price	9.48
Variance	0.004
Required Number of Simulations	276
Execution Time (Seconds)	142

The level of variance reduction for Input 2 is surprising. We know that it comes from using the Sobol Sequence. It remains unclear why the Sobol Sequence has such an impact on this second set of inputs.

Control Variate and Antithetic Random Variable

Input 1	CV and Antithetic
Number of Simulations	2 000
Error Tolerated	0.01
Price	13.05
Variance	3.94
Required Number of Simulations	260 373
Execution Time (Seconds)	44

Input 2	CV and Antithetic
Number of Simulations	3 000
Error Tolerated	0.01
Price	12.03
Variance	8.74
Required Number of Simulations	577 166
Execution Time (Seconds)	79

Combined together, Control Variate and Antithetic Variable shows incredible results in terms of variance reduction, additionally to a faster algorithm. Again, the estimated variance is reduced by more than 95%.

Using All Three Techniques

Input 1	3 Techniques
Number of Simulations	2 000
Error Tolerated	0.01
Price	13.66
Variance	4.4
Required Number of Simulations	290 634
Execution Time (Seconds)	43

Input 2	3 Techniques
Number of Simulations	2 000
Error Tolerated	0.01
Price	9.48
Variance	0.0017
Required Number of Simulations	112
Execution Time (Seconds)	53

Cumulating all three techniques allows for sure to significantly decrease the number of simulations required. For Input 1, extrapolating the time needed to run the same pricing simulation without any variance reduction technique would require around 104 hours to enter the confidence interval. Using all techniques decrease the computation time to less than 2 hours.

Summary

Input 1:

	No Variance Reduction	Sobol	Antithetic	Control Variate
Number of Simulations	2 000	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01	0.01
Price	12.52	10.91	12.42	13
Variance	171.00	88.03	20.45	7.296
Required Number of Simulations	11 339 627	5 759 728	1 350 579	481 893

	Sobol and Antithetic	Sobol and CV	CV and Anti	3 Techniques
Number of Simulations	2000	2000	2000	2000
Error Tolerated	0.01	0.01	0.01	0.01
Price	11.01	13.71	13.05	13.66
Variance	7.92	10.5691	3.94	4.4
Required Number of Simulations	522 813	698 081	260 373	290 634
Execution Time (Seconds)	45	67	44	43

Input 2:

	No Variance Reduction	Sobol	Antithetic	Control Variate
Number of Simulations	3 000	3 000	3 000	3 000
Error Tolerated	0.01	0.01	0.01	0.01
Price	10.18	1.24	10.03	12.07
Variance	211.62	0.72	52.35	19.85
Required Number of Simulations	13 977 378	47 727	3 457 915	1 311 301
Execution Time (Seconds)	134	144	78	83

	Sobol and Antithetic	Sobol and CV	CV and Anti	3 Techniques
Number of Simulations	3 000	3 000	3 000	3 000
Error Tolerated	0.01	0.01	0.01	0.01
Price	1.23	9.48	12.0283	9.48
Variance	0.015	0.004	8.74	0.0017
Required Number of Simulations	988	276	577 166	112
Execution Time (Seconds)	53	85	79	53

American and Bermudan Options

Construction of a Simulation for path dependant Payoff with Variance Reduction techniques

Creating a MonteCarlo simulation for path dependant options (American Basket options and Bermudan basket options) may require additional parameters with respect to the European one. First, presenting quickly methodology will give insights on these additional parameters and then the possible features of the product will be taken into account.

The Longstaff-Schwartz approach

The following approach is based on the paper of Longstaff and Schwartz (Longstaff and Schwartz, 2001) who proposed a framework based on a general Least Square regression to estimate the continuation value of these products allowing to build Monte Carlo algorithms in order to price these products. In a nutshell, one look for the continuation value at each time step that is simply the conditional expectation of the payoff at one step ahead knowing the value taken by the underlying at the previous time step.

$$Continuation\,Value = E(Y|X)$$

The proposition of *LS* is to approach the above quantity using GLS (General Least Square) method that boils down to estimate the following regression:

$$e^{-rdt}Y = \sum_{m=1}^{M} \beta_t^j * F(X)$$

Where F is a set of functions (j functions) applied to the vector of the underlying spot value X.

After finding the Betas through the regression, of will compare this quantity with the immediate payoff and take the max between the two at each time step and for each in-the-money paths in order to update the value at each step. Finally, the monteCarlo price will be given by:

$$\frac{1}{M}\sum_{m=1}^{M} e^{-r*t_m^*}\, Y_m$$

Where the final discounting will take into account the optimal stopping time (optimal time to be exercised) for each path.

The brief presentation of the method shows the need for the first additional parameter that is a vector of pointers to basis function (F) that will allow to estimate the continuation value. In generality, one should check that the continuation value for a specified payoff belongs to a L^n space and because of that space will be a Hilbert one, the continuation value could be approximated as a linear combination of any elements of that space (these functions can be found in the Basis Functions class) and the parameter **polynomial** in the MonteCarlo constructor. As per the reference paper, simple polynomes are implemented (*ie* $X^n\ \forall n\ an\ integer$) as well as Laguerre polynomes and Hermite polynomes.

The polynomes functions will be constructed with the following manner:

```
std::vector<basis_functions*> basefunc_Hermite;

basefunc_Hermite.push_back(new Poly_Hermite(0));
basefunc_Hermite.push_back(new Poly_Hermite(1));
basefunc_Hermite.push_back(new Poly_Hermite(2));

std::vector<basis_functions*> basefunc_Laguerre;

basefunc_Laguerre.push_back(new Poly_Laguerre(0));
basefunc_Laguerre.push_back(new Poly_Laguerre(1));
basefunc_Laguerre.push_back(new Poly_Laguerre(2));

std::vector<basis_functions*> basefunc_Simple;

basefunc_Simple.push_back(new polynome_simple(0));
basefunc_Simple.push_back(new polynome_simple(1));
basefunc_Simple.push_back(new polynome_simple(2));
```

Path dependant exercice

As the Bermudan option can be exercised at a set of contractual predetermined dates, one need to take this feature into account. This will be done through two parameters.

Starting the pointer to a generic class of **CalendarManagement** functions that will allow to assemble the right underlying asset value sequence extracted from the stock path generated. This can be either an interpolation function, or a function that manages dates and calendar object to fit the term sheet. In all our simulations we worked with the **rounded_workingdays** class. This object takes only an int parameter that will set the number of following working days one need to take when searching for the reference spot. Taking the example of a Bermudan swaption that could be exercised each 28th of the month. If the 28th is a non-worked day than the parameter of the class constructor allow to take the following worked day (or two following worked day etc…) looking like this:

```
//creation of the calendar management to get th right spot vector at each simulation

CalendarManagement* wkday = new rounded_workingdays(0);
```

The second parameter inside the Bermudan MonteCarlo constructor will be a vector representing the different dates at which one can exercised the product. In the following simulations, the vector will contain doubles that are each time a fraction of the total life of the trade (in days) each fraction representing a possible date of exercise. Creation can be done in the following manner:

```
//creattion of the exercice schedule for the Bermudan option

size_t n = 12;
matrix Schedule_exec(n, 1);

double test = 0.08;

for (size_t i = 0; i < n; i++)
{

    Schedule_exec(i, 0) = test;
    test = 0.08 * (i + 1.);
}
```

For the implementation of the algorithm, we took as a reference the paper of (Gustafsson, 2015).

Results for American and Bermudan Payoff

One kept the same input parameters as the European case. The additional parameters will be as follow:

```
//// need to choose the polynome basis with which one will approxime the continuation val
std::vector<basis_functions*> basefunc_Hermite;

basefunc_Hermite.push_back(new Poly_Hermite(0));
basefunc_Hermite.push_back(new Poly_Hermite(1));
basefunc_Hermite.push_back(new Poly_Hermite(2));

std::vector<basis_functions*> basefunc_Laguerre;

basefunc_Laguerre.push_back(new Poly_Laguerre(0));
basefunc_Laguerre.push_back(new Poly_Laguerre(1));
basefunc_Laguerre.push_back(new Poly_Laguerre(2));

std::vector<basis_functions*> basefunc_Simple;

basefunc_Simple.push_back(new polynome_simple(0));
basefunc_Simple.push_back(new polynome_simple(1));
basefunc_Simple.push_back(new polynome_simple(2));


//Create the payoff of the options

PayOffBasket* bsktcall = new PayOffBasketCall(W, spot_m, 100.);

//creattion of the exercice schedule for the Bermudan option

size_t n = 12;
matrix Schedule_exec(n, 1);

double test = 0.08;

for (size_t i = 0; i < n; i++)
{

    Schedule_exec(i, 0) = test;
    test = 0.08 * (i + 1.);
}

//creation of the calendar management to get th right spot vector at each simulation

CalendarManagement* wkday = new rounded_workingdays(0);
```

In the following subsections, each time one will present results comparing the three different sets of polynomes (with orders 0, 1 and 2) to gauge a possible effect when choosing the basis. All simulations presented are based on a 2000 sample. Even if our primary goal is to price Bermudan options, pricing the American equivalent is also a good practice to check for consistency in the pricing. Indeed, as the optionality the price of the American should always be higher will the European one should be cheaper than the Bermudan. Having these boundaries in mind, one can proceed to the analysis of the following results.

No variance reduction

Input 1 – Bermudan case	**Laguerre**	**Hermite**	**Simples**
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	10.7975	10.7975	10.7975
Variance	128.81	128.81	128.81
Required Number of Simulations	8 507 767	8 507 767	8 507 767
Execution Time (Seconds)	29.89	24.82	26.41

Input 2 – Bermudan case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	9.23838	9.23838	9.23838
Variance	159.22	159.22	159.22
Required Number of Simulations	10 536 242	10 536 242	10 536 242
Execution Time (Seconds)	31.58	30.67	32.13

The pricing seems good as the American is more expensive than the European option price previously (11.34 vs 8.87) while the European is cheaper than the European one. However both show high variance and the number of simulations is quite low compared to the one needed to enter the confidence interval. Which seems to stay consistent in the second sample.

Input 1 – American case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	11.3403	11.3403	11.3403
Variance	114.714	114.714	114.714
Required Number of Simulations	9 558 184	9 558 184	9 558 184
Execution Time (Seconds)	45.91	45.29	44.8

Input 2 – American case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	9.94461	9.94461	9.94461
Variance	186.066	186.066	186.066
Required Number of Simulations	12 289 462	12 289 462	12 289 462
Execution Time (Seconds)	41.69	40.63	41.2969

It is on the other hand reassuring to see that the American case (Bermudan when the number of possible exercices tend to infinite) is more expensive than the Bermudan. However, the accuracy of the algorithm is still quite low and it requires a high number of simulations to enter the confidence interval. Finally, the choice of each the function family seems to have no impact on the price itself (which is expected as these are all elements from L^2 and therefore all good candidates to approach the continuation value) but it could be worth to choose the one that minimizes the time of computation for a given level of accuracy.

Quasi Random Sequence

In this subsection, one will use Van der Corput and Sobol sequences to generate uniform numbers in order to build the Brownian motion underlying the Black and Scholes diffusion. Implemented this approach aims at reducing the discrepancy of the simulated numbers resulting in reducing the variance among the simulations.

Van der Corput

Input 1 – Bermudan case	**Laguerre**	**Hermite**	**Simples**
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	6.97936	6.97936	6.97936
Variance	10.409	10.409	10.409
Required Number of Simulations	687 501	687 501	687 501
Execution Time (Seconds)	25.68	26.37	26.

Input 2 – Bermudan case	**Laguerre**	**Hermite**	**Simples**
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	0.978428	0.978428	0.978428
Variance	1.32632	1.32632	1.32632
Required Number of Simulations	87 602	87 602	87 602
Execution Time (Seconds)	38.86	32.55	32.11

Input 1 – American case	**Laguerre**	**Hermite**	**Simples**
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	7.13465	7.13465	7.13465
Variance	6.07449	6.07449	6.07449
Required Number of Simulations	401 214	401 214	401 214
Execution Time (Seconds)	49.44	51.29	49.52

Input 2 – American case	**Laguerre**	**Hermite**	**Simples**
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	1.04159	1.04159	1.04159
Variance	1.37274	1.37274	1.37274
Required Number of Simulations	90 667	90 667	90 667
Execution Time (Seconds)	43.6165	41.85	47.1037

If we can see a large reduction in variance and in the number of optimal simulations, it is quite strange to see that prices boundaries are violated. In that case this can be linked to the choice of the basis in the Van Der Corput generation that seems to affect largely the performance of the algorithm.

Sobol

Input 1 – Bermudan case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	8.81811	8.81811	8.81811
Variance	62.1244	62.1244	62.1244
Required Number of Simulations	4 103 256	4 103 256	4 103 256
Execution Time (Seconds)	24.81	27.09	27.39

Input 2 – Bermudan case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	0.835996	0.835996	0.835996
Variance	0.633332	0.633332	0.633332
Required Number of Simulations	41 830	41 830	41 830
Execution Time (Seconds)	31.1	32.13	30.94

Input 1 – American case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	9.43546	9.43546	9.43546
Variance	72.6542	72.6542	72.6542
Required Number of Simulations	4 798 737	4 798 737	4 798 737
Execution Time (Seconds)	49.48	47.43	46.01

Input 2 – American case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	1.071	1.071	1.071
Variance	0.712879	0.712879	0.712879
Required Number of Simulations	47 084	47 084	47 084
Execution Time (Seconds)	43.17	43.8	45.24

In the Sobol case we encounter the same issue that in the above subjection with Van Der Corput, that prices are quite far from what should be the true price. In addition, it seems that the Van der Corput sequence seems to yield better results in terms of variance reductions and optimal simulations.

Quasi Monte Carlo and Antithetic Random Variable

Van der Corput

Input 1 – Bermudan case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	6.97631	6.97631	6.97631
Variance	0.0401633	0.0401633	0.0401633
Required Number of Simulations	2652	2652	2652
Execution Time (Seconds)	16.70	18.55	18.46

Input 2 – Bermudan case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	0.980551	0.980551	0.980551
Variance	0.196778	0.196778	0.196778
Required Number of Simulations	12 996	12 996	12 996
Execution Time (Seconds)	18.76	19.67	20.26

Input 1 – American case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	7.13456	7.13456	7.13456
Variance	0.0160825	0.0160825	0.0160825
Required Number of Simulations	1062	1062	1062
Execution Time (Seconds)	39.47	46.44	47

Input 2 – American case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	1.01147	1.01147	1.01147
Variance	0.201214	0.201214	0.201214
Required Number of Simulations	13 289	13 289	13 289
Execution Time (Seconds)	31.78	31.61	32.69

Sobol

Input 1 – Bermudan case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	9.14256	9.14256	9.14256
Variance	6.78828	6.78828	6.78828
Required Number of Simulations	448 359	448 359	448 359
Execution Time (Seconds)	16.83	18.5	26.12

Input 2 – Bermudan case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	1.21804	1.21804	1.21804
Variance	0.0283748	0.0283748	0.0283748
Required Number of Simulations	1874	1874	1874
Execution Time (Seconds)	19.28	20.45	20.05

Input 1 – American case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	9.86449	9.86449	9.86449
Variance	8.44818	8.44818	8.44818
Required Number of Simulations	557 993	557 993	557 993
Execution Time (Seconds)	37.37	40.31	43

Input 2 – American case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	1.27057	1.27057	1.27057
Variance	0.0308042	0.0308042	0.0308042
Required Number of Simulations	2034	2034	2034
Execution Time (Seconds)	36.36	35.11	35.67

Adding the Antithetic variable does enhance the variance reduction and the number of optimal simulations needed. However, one can see that the pricing on the second set of parameters is still inconsistent.

Quasi Monte Carlo and Static Control Variate

Van der Corput

Input 1 – Bermudan case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	11.7092	11.7092	11.7092
Variance	0.552085	0.552085	0.552085
Required Number of Simulations	36 464	36 464	36 464
Execution Time (Seconds)	28.26	29.47	26.08

Input 2 – Bermudan case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	9.94713	9.87037	10.0616
Variance	0.0104723	1.60847	1.14419
Required Number of Simulations	691	106 237	75 572
Execution Time (Seconds)	31.8864	33.29	33.31

Input 1 – American case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	11.6002	14.1668	14.1668
Variance	0.261268	5.56391	5.56391
Required Number of Simulations	17 256	367 490	367 490
Execution Time (Seconds)	73.54	75.1	73.73

Input 2 – American case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	10.1255	9.44823	9.55913
Variance	1.95975	1.15183	1.91568
Required Number of Simulations	129 439	76 076	126 528
Execution Time (Seconds)	68.11	58.06	57.13

Interestingly, the first test does show a lack of consistency as the Bermudan is more expensive than the American counterpart while the condition is respected in the second one. Nonetheless, one can see from table above that the pricing on the second set of parameters is well closer to the reality than in the previous case (antithetic and quasi random generation).

Sobol

Input 1 – Bermudan case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	12.8095	12.8095	12.8095
Variance	8.30296	8.30296	8.30296
Required Number of Simulations	548 402	548 402	548 402
Execution Time (Seconds)	27.32	27.44	27.21

Input 2 – Bermudan case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	9.96012	9.72048	9.5904
Variance	0.00875142	0.58483	0.5604
Required Number of Simulations	578	38 627	37 148
Execution Time (Seconds)	31.99	32.88	32.33

Input 1 – American case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	13.2108	16.5827	13.2108
Variance	12.6466	72.2583	12.6466
Required Number of Simulations	835 295	4 772 588	835 295
Execution Time (Seconds)	66.05	66.05	65.4

Input 2 – American case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	10.248	10.2175	10.1326
Variance	0.477764	0.505947	0.562468
Required Number of Simulations	31 555	33 417	37 150
Execution Time (Seconds)	60.85	60.63	61.2

From the Van der Corput shown above, one can see that the pricing boundaries are no more violated here giving therefore results that are more consistent with our expectations.

Antithetic Random Variable

The aim of the Antithetic Random Variable is to lower the variance of the simulated price by splitting the subset into two sub samples in order to simulated our target variable (Y) as the mean between X_1 and X_2, where both are negatively correlated. This should result in a lower variance of the simulated

price in addition to lower time of computation and a lower optimal number of simulations to enter our confidence interval.

Input 1 – Bermudan case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	10.6055	10.6055	10.6055
Variance	16.6115	16.6115	16.6115
Required Number of Simulations	1 097 171	1 097 171	1 097 171
Execution Time (Seconds)	16.64	19.15	18.44

Input 2 – Bermudan case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	9.37993	9.37993	9.37993
Variance	39.3047	39.3047	39.3047
Required Number of Simulations	2 596 037	2 596 037	2 596 037
Execution Time (Seconds)	19.08	19.43	19.82

Input 1 – American case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	11.2381	11.2381	11.2381
Variance	19.8566	19.8566	19.8566
Required Number of Simulations	1 311 507	1 311 507	1 311 507
Execution Time (Seconds)	35.2	62.69	40.64

Input 2 – American case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	10.0334	10.0334	10.0334
Variance	46.75	46.75	46.75
Required Number of Simulations	3 087 790	3 087 790	3 087 790
Execution Time (Seconds)	32.57	31.61	31.31

If the quasi random generation did not yield good results with the antithetic method, this is not the case when using traditional random generation through the Ecuyer Combined generator. Indeed, in the four table above, relationship is respected between the Bermudan and the American options. In addition, the gain in computation time is large which has to be seen in pair with the large reduction in variance (from the vanilla case displayed at the beginning of this section) in addition to the large decrease in optimal simulations needed.

Static Control Variate

This approach aims at using the possible existence of a payoff that has the same mean as our initial payoff but at a lower price and as nearly the same cost of simulation. Simulating both at the same take and adjusting by the close form price of the control payoff should compensate some way for the downward looking bias inherent to the MonteCarlo approach. In the same spirit as before, this should result This should result in a lower variance of the simulated price in addition to a lower optimal number of simulations to enter our confidence interval. However, the computation time could be higher as one simulates two payoffs at the same time (multiplying by almost two the computations made).

Input 1 – Bermudan case	**Laguerre**	**Hermite**	**Simples**
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	12.2127	12.2127	12.2127
Variance	6.58268	6.58268	6.58268
Required Number of Simulations	434 779	434 779	434 779
Execution Time (Seconds)	28.18	26.65	27.46s

Input 2 – Bermudan case	**Laguerre**	**Hermite**	**Simples**
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	12.2048	12.2048	12.2048
Variance	14.5196	14.5196	14.5196
Required Number of Simulations	959 007	959 007	959 007
Execution Time (Seconds)	32.18	31.9	32.29

Input 1 – American case	**Laguerre**	**Hermite**	**Simples**
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	12.3211	12.3211	12.4089
Variance	7.44122	7.44122	10.2849
Required Number of Simulations	491 485	491 485	679 307
Execution Time (Seconds)	64.6	62.78	63.77

Input 2 – American case	**Laguerre**	**Hermite**	**Simples**
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	12.5272	12.5272	12.5272
Variance	17.9082	17.9082	17.9082
Required Number of Simulations	1 182 817	1 182 817	1 182 817
Execution Time (Seconds)	55.84	56.2	54.78

Same conclusion can be inferred here than in the previous subsection (antithetic) with a large reduction in variance and optimal simulations needed. However, this method is more intensive in computation as there are no split in the simulation but there is twice as much as calculus needed.

Static Control Variate and Antithetic Control Variable

One can see that the two previous techniques proved to reduce the variance as well as the number of optimal simulations. The following results present the combination of both approach *ie* pricing both the option and its static control counterpart on the initial set of simulation and its antithetic counterpart. There is not so much hope for reduction in computation time however the aim will be still reducing the variance and the optimal number of simulations needed.

Input 1 – Bermudan case	**Laguerre**	**Hermite**	**Simples**
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	9.62734	9.62734	9.62734
Variance	27.3046	27.3046	27.3046
Required Number of Simulations	1 803 442	1 803 442	1 803 442
Execution Time (Seconds)	17.9	16.47	15.79

Input 2 – Bermudan case	**Laguerre**	**Hermite**	**Simples**
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	9.33723	9.33723	9.33723
Variance	36.0617	36.0617	36.0617
Required Number of Simulations	2 381 836	2 381 836	2 381 836
Execution Time (Seconds)	20.95	20.44	20.20

Input 1 – American case	**Laguerre**	**Hermite**	**Simples**
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	12.39	16.1072	15.447
Variance	3.72902	28.3134	32.4408
Required Number of Simulations	246 298	1 870 073	2 142 681
Execution Time (Seconds)	61.9	61.39	60.75

Input 2 – American case	**Laguerre**	**Hermite**	**Simples**
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	12.4697	14.2727	12.8953
Variance	8.07072	29.5586	13.3781
Required Number of Simulations	533 062	1 952 314	883 611
Execution Time (Seconds)	46.83	45.80	47.45

Using all three techniques

Now that we have seen quite an improvement taking either quasi random numbers or variance reduction techniques, one can build the algorithm using quasi random numbers generation in addition to both Static control and antithetic variable methods at the same time.

Van Der Corput

Input 1 – Bermudan case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	10.6244	10.6244	10.6262
Variance	1.19748	1.19748	1.19612
Required Number of Simulations	79 092	79 092	79 002
Execution Time (Seconds)	19.93	16.27	16.16

Input 2 – Bermudan case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	10.0856	10.0179	9.7883
Variance	0.516945	0.483086	0.806035
Required Number of Simulations	34 143	31 907	53 237
Execution Time (Seconds)	20.27	21.24	22.5728

Input 1 – American case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	12.8841	14.0899	14.1445
Variance	1.02584	0.687073	0.322617
Required Number of Simulations	67 755	45 380	21 308
Execution Time (Seconds)	71.45	74.76	72.01

Input 2 – American case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	9.69798	9.66306	9.83905
Variance	0.414627	0.450068	0.511462
Required Number of Simulations	27 385	29 726	33 781
Execution Time (Seconds)	48.46	46.55	56.64

It is interesting to see that the choice of polynomes is no more transparent and should be therefore considered carefully in the pricing methodology. The reduction in variance and in needed optimal simulations is large which goes hand in hand with a low time of computation in the Bermudan case. If the prices seem quite in line in what we saw previously, the relationship between the Bermudan and

the American does not hold in the second set of inputs (already seen before) this is quite strange and should be explore more carefully in order to have consistent results all the time.

Sobol

Input 1 – Bermudan case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	10.5318	10.5318	10.5318
Variance	13.4129	13.4129	13.4129
Required Number of Simulations	885 910	885 910	885 910
Execution Time (Seconds)	16.96	15.92	16.2

Input 2 – Bermudan case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	9.86563	9.79156	9.84446
Variance	0.324443	0.450334	0.267298
Required Number of Simulations	21 429	29 744	17 654
Execution Time (Seconds)	21.29	21.87	21.53

Input 1 – American case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	15.1277	15.1277	14.9222
Variance	14.2255	14.2255	14.5566
Required Number of Simulations	939 581	939 581	961 448
Execution Time (Seconds)	65.94	60.31	62.15

Input 2 – American case	Laguerre	Hermite	Simples
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	10.2699	10.2406	10.1541
Variance	0.138033	0.204668	0.08824
Required Number of Simulations	9 116	13 518	5 828
Execution Time (Seconds)	55.88	58.83	55.7

We can draw the same conclusion than in the Van der Corput case above, large decrease in variance in addition to a large decrease in the number of optimal number of simulations (under 10 000 for the case with simple polynomes) in addition to see that the relationship holds all the time as the American is always more expensive than the Bermudan.

Increasing the number of polynomes

In order to gauge for the effect of increasing the number of polynome orders in the continuation value approximation, the following results are based on the Input 1 set of parameters all else being equal and changing only the number of Laguerre polynomes to 6 (.*versus* 2 previously) et taking only the American case as an illustration.

Vanilla	**Ecuyer**	**VdC**	**Sobol**
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	11.2831	7.13115	9.19896
Variance	143.275	6.06289	71.788
Required Number of Simulations	9 463 165	400 447	4 741 526
Execution Time (Seconds)	498.28	512.08	493.

Static Control Variable	**Ecuyer**	**VdC**	**Sobol**
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	12.3058	11.6111	13.0176
Variance	7.47551	0.28583	11.1713
Required Number of Simulations	493 749	18 664	737 856
Execution Time (Seconds)	1029.72	1160.03	1093.62

Antithetic Variable	**Ecuyer**	**VdC**	**Sobol**
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	11.1759	43.026	9.69566
Variance	19.9744	11.2002	8.26003
Required Number of Simulations	1 319 289	739 761	545 567
Execution Time (Seconds)	890.6	915.453	893.259

Static Control Variable & Antithetic	**Ecuyer**	**VdC**	**Sobol**
Number of Simulations	2 000	2000	2000
Error Tolerated	0.01	0.01	0.01
Price	12.3937	11.6066	13.0234
Variance	3.7392	0.0175979	4.17973
Required Number of Simulations	246 970	1162	276 066
Execution Time (Seconds)	1752.16	1793.32	2163.42

Increasing the number of polynomes to approach the continuation value does not yield interesting results in the sense that, it increases largely the time of computation without improving much either the variance nor the number of optimal simulations whatever approach selected in terms of techniques or random numbers generation.

Conclusion

In most of cases the relationship of order between the all three type of options holds which show a high consistency in the results obtained. It is reassuring to see that using quasi random sequences and/or variance reduction techniques always lead to lower variance and lower optimal simulations. In addition, Sobol sequences seems to perform better than Van Der Corput, which is subject to the choice of the basis of both generators. This is the heart of the pricing war both in terms of time and accuracy, choosing the right sequence, wisely calibrated and performing well in any case of variance reduction techniques. The choice of a 1% error is a good approximation for a theoretical exercise, in practice one should aim for 1 bps maximum (the lower, the better). Indeed, pricing should be the most accurate possible in order to be both competitive but also to be able to have a proper hedge and the lowest risk possible associated to the pricing. In addition, if one adds features such as barrier, the greeks and hence the hedging could suffer from a too large confidence interval that would mislead the trading create opportunity for high variations in P&L (both in negative or positive territories). Finally, this can be also used in order to estimate the value at risk of portfolios composed of path dependant assets.

Going further?

This report is a broad first attempt to assess the accuracy and the usability of the quasi Monte Carlo pricing method. In order to go further we could have implemented much more. First of all, try other quasi random sequences such as Kakutani sequences to see if it yields better results than Sobol or Van Der Corput and if it is consistent among all mehods. One should also consider to test other situation by implemented more advanced diffusion than Black Scholes hence taking into account for stochastic rates, volatility and correlation (especially in our context of Covid-19, where large drawbacks in the market yield to higher correlation and volatility most of the time). In order to fit more the pricing industry, one could have also implemented bootstrapping of yield curve to have a proper discounting in addition to a real calendar management that is key for the Bermudan pricing.

References

Gustafsson, W. (2015). Evaluating the Longstaff-Schwartz method for pricing of American options. *UPPSALA Universitet*, 9-13.

Longstaff and Schwartz. (2001). Valuing American Options by Simulation: A Simple Least-Squares Approach. *The Review of Financial Studies, Vol. 14, No. 1, Spring*, 113-147.

Press, W., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2002). *Numerical Recipes in C*. Retrieved from https://www.cec.uchile.cl/cinetica/pcordero/MC_libros/NumericalRecipesinC.pdf

Sigman, K. (2007). Retrieved from http://www.columbia.edu/~ks20/4703-Sigman/4703-07-Notes-ATV.pdf