CHAPTER 9

Monte Carlo Option Pricings

9.1. The Monte Carlo Method

The Monte Carlo method provides numerical solution to a variety of mathematical problems by performing statistical samplings on a computer. In risk-neutral pricing of options, we are most interesting in evaluating the expected value of a function $g(x)$ under a random variable x as

$$E(\, g(x)\,) = \int_{-\infty}^{+\infty} dx\; g(x)\; \varphi(x) \tag{9.1}$$

where $\varphi(x)$ is the probability density function of x. In general, it would be difficult to derive an analytical formula for (9.1) and a numerical estimation seems to be the only way out. Monte Carlo simulation provides a simple and flexible numerical method to solve these types of problems[1]. Consider the random sample $\{ x_1, x_2, \ldots , x_n \}$ generated based on the probability density function $\varphi(x)$. The estimates of the mean and variance of $g(x)$ are given by

$$m = \frac{1}{n} \sum_{i=1}^{n} g(x_i) \qquad \text{and} \qquad s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (\, g(x_i) - m\,)^2 \tag{9.2}$$

According to the central limit theorem, the random variable defined as

$$\frac{m - E(\, g(x)\,)}{\frac{s}{\sqrt{n}}} \tag{9.3}$$

tends to follow a standard normal distribution with increasing sample size n and be irrespective of the distribution of $g(x)$. Thus, the sample average m approaches a normal distribution with mean $E(\, g(x)\,)$ and standard deviation $(s/\sqrt{n})$. On this basis, the confidence interval in the estimation of $E(\, g(x)\,)$ can be obtained as

$$E(\, g(x)\,) = m \pm z\, \frac{s}{\sqrt{n}} \tag{9.4}$$

with, for example, confidence level of 68.27% for $z = 1$. We refer to the term $(s/\sqrt{n})$ in (9.4) as the standard error in the estimation of $E(\, g(x)\,)$. To reduce the standard error by a factor of ten, the sample size has to be increased one hundredfold. The method is thus computationally inefficient in its fundamental form called the crude Monte Carlo simulation.

There are variance reduction techniques that focus on reducing instead the size of s in the standard error. The common approach is known as the control variate method. It takes the analytic solution of a similar but simpler problem to improve the accuracy in the estimated solution of a complex problem. Suppose that the expected value $E(\, h(x)\,)$ can be evaluated analytically as H. In related to the original function $g(x)$, we can introduce a new function given by

$$\tilde{g}(x) = g(x) - h(x) \tag{9.5}$$

through the control variate $h(x)$ and rewrite (9.1) as

$$E(\, g(x)\,) = H + \int_{-\infty}^{+\infty} dx\; \tilde{g}(x)\; \varphi(x) \tag{9.6}$$

[1] See P. P. Boyle, "Options : A Monte Carlo Approach", *Journal of Financial Economics*, Vol. **4**, Issue 3 (1977) p 323-338.

Thus, we can determine the confidence interval in the estimation of $E(\, g(x)\,)$ based on instead the estimates of the mean and variance of $\tilde{g}(x)$ given by

$$E(\, g(x)\,) = (\, H + \tilde{m}\,) \pm z\frac{\tilde{s}}{\sqrt{n}} \tag{9.7}$$

It can be shown that the variances of $g(x)$ and $\tilde{g}(x)$ can be related as

$$var(\, \tilde{g}(x)\,) = var(\, g(x)\,) + var(\, h(x)\,) - 2\, cov(\, g(x)\,,\, h(x)\,) \tag{9.8}$$

If $g(x)$ and $h(x)$ are similar problems, the covariance between them is positive. In (9.8), the variance of $\tilde{g}(x)$ will be less than the variance of $g(x)$ as long as $cov(\, g(x)\,,\, h(x)\,) > \frac{1}{2}\, var(\, h(x)\,)$. It is therefore possible to reduce the size of the standard error by identifying a highly correlated problem with known analytic solution.

An alternative approach is known as the antithetic variate method. In case of standard normal variable, it makes use of the symmetric property around zero in the density function. Again, we can introduce a new function given by

$$\hat{g}(x) = \tfrac{1}{2}[\, g(x) + g(-x)\,] \tag{9.9}$$

through antithetic variate of the form $-x$. We can rewrite (9.1) using the symmetric property of the standard normal variable x as

$$E(\, g(x)\,) = \int_{-\infty}^{+\infty} dx\, \hat{g}(x)\, \varphi(x) \tag{9.10}$$

Similarly, we can determine the confidence interval in the estimation of $E(\, g(x)\,)$ based on the estimates of the mean and variance of $\hat{g}(x)$ given by

$$E(\, g(x)\,) = \hat{m} \pm z\frac{\hat{s}}{\sqrt{n}} \tag{9.11}$$

The variance of $\hat{g}(x)$ is expected to be smaller than the variance of $g(x)$ as it is already an average quantity of two samples. It can be shown that the two variances can be related as

$$var(\, \hat{g}(x)\,) = \tfrac{1}{2}\, var(\, g(x)\,) + \tfrac{1}{2}\, cov(\, g(x)\,,\, g(-x)\,) \tag{9.12}$$

If the covariance between $g(x)$ and $g(-x)$ is negative, it is always efficient to consider the estimates for $\hat{g}(x)$ rather than doubling the size of independent samples.

	A	B	C	D	E
1		*Random Numbers*	*Crude Simulation*	*Control Variate Method*	*Antithetic Variate Method*
2		x	$g(x) = e^x$	$\tilde{g}(x) = e^x - x$	$\hat{g}(x) = \frac{1}{2}(e^x + e^{-x})$
3	1	1.049050384	2.854938735	1.805888351	1.602604474
4	2	-1.204751932	0.299766353	1.504518285	1.817848895
5	3	-1.811351962	0.163433032	1.974784994	3.141073571
10001	9999	1.365806708	3.918883171	2.553076463	
10002	10000	-0.96166111	0.382257387	1.343918497	
10003		*Mean*	1.660	1.651	1.658
10004		*Standard Error*	0.023	0.017	0.019
10005			(10000 samples)	(10000 samples)	(5000 samples)

Figure 9.1: Monte Carlo simulation for $E(\, e^x\,)$.

Figure 9.1 depicts, for example, the Monte Carlo estimation of the expected value $E(\, e^x\,)$ with x taken to be a standard normal variable. In EXCEL, random samples of $x = \varepsilon(\, 0\,,\, 1\,)$ can be generated

very easily by calling the function **NORMSINV**(**RAND**()). They will be used to evaluate the sample values of $g(x) = e^x$ for which mean and variance can be estimated. In the figure, we compare the confidence interval evaluated through the crude simulation with those based on the variance reduction techniques. For the control variate method, we have adopted a highly correlated problem $E(\, x\,)$ with known analytic solution of zero. Thus, we introduce $\tilde{g}(x) = e^x - x$ for which $E(\, e^x\,) = E(\, e^x - x\,)$. For the antithetic variate method, it is clear that the covariance between e^x and e^{-x} is negative. We thus define $\hat{g}(x) = ½\,(\, e^x + e^{-x}\,)$ with $E(\, e^x\,) = E(\, ½\,(\, e^x + e^{-x}\,)\,)$. In both cases, we have shown that there are significant reductions in the size of the standard errors.

9.2. Risk-Neutral Valuation

Under risk-neutral valuation, current price of an option can be defined based on the present value of its average maturity payoff at time T as

$$f_0 = \hat{E}(\, e^{-rT} f_T \,|\, S_0\,) \tag{9.13}$$

where r is the constant interest rate. Here, we are averaging over realized maturity payoffs of the option f_T in respect to sample underlying asset prices generated through its risk-neutral process that initiated at current price S_0. In stochastic model, asset price return during the time increment from t to $t + \Delta t$ is assumed to follow a random normal process as

$$\Delta S_t / S_t = \mu \Delta t + \sigma \sqrt{\Delta t}\, \varepsilon(\, 0\,,\, 1\,) \tag{9.14}$$

where μ and σ are respectively the mean rate and volatility of return. For traded asset such as stock, the risk-neutral process is simply given by (9.14) with μ replaced by r in the drift term. Practically, it is convenient to consider the asset price movement based on the risk-neutral process. For constant volatility of return, it is shown to follow an iterative equation with arbitrary time duration given by

$$S_{t+\tau} = S_t \exp\big(\,(\, r - ½\sigma^2\,)\tau + \sigma\sqrt{\tau}\,\varepsilon(\, 0\,,\, 1\,)\,\big) \tag{9.15}$$

In particular, we have

$$S_T = S_0 \exp\big(\,(\, r - ½\sigma^2\,)T + \sigma\sqrt{T}\,\varepsilon(\, 0\,,\, 1\,)\,\big) \tag{9.16}$$

that generates the maturity price S_T directly from S_0 in a single step. It will be useful when there is no intermediate boundary condition for the option.

As our first example, we consider a European call option written on a non-dividend paying stock with strike price K. It should be noted that there exits analytic solution for this option known as the Black-Scholes formula given by

$$f_0^{BS} = S_0\, N(\, d\,) - K\, e^{-rT}\, N(\, d - \sigma\sqrt{T}\,) \quad , \quad d = \frac{ln(S_0/K) + (\, r + ½\,\sigma^2\,)T}{\sigma\sqrt{T}} \tag{9.17}$$

Here, we take this simple example to demonstrate the use of Monte Carlo procedure for option pricing. In this case, the option payoff will depend solely on the underlying asset price at maturity regardless of its intermediate values. We can simply use (9.16) to generate the maturity price of the asset by a single random number ε from $\varepsilon(\, 0\,,\, 1\,)$. The sample maturity price can then be used to evaluate the sample maturity payoff of the option according to the function

$$f_T(\varepsilon) = max\{\, S_T(\varepsilon) - K\,,\, 0\,\} \tag{9.18}$$

For variance reduction, we can adopt the maturity price S_T itself as the control variate and consider the new function

$$\tilde{f}_T(\varepsilon) = max\{\, S_T(\varepsilon) - K\,,\, 0\,\} - S_T(\varepsilon) \tag{9.19}$$

The control variate has an analytic solution given by $\hat{E}(\, e^{-rT} S_T \mid S_0\,) = S_0$ in which we can rewrite (9.13) as

$$f_0 = S_0 + \hat{E}(\, e^{-rT} \tilde{f}_T \mid S_0\,) \tag{9.20}$$

Alternatively, we can simply take $-\varepsilon$ as the antithetic variate and introduce the new function

$$\hat{f}_T(\varepsilon) = \tfrac{1}{2}\,[\, max\{\, S_T(\varepsilon) - K\,,\, 0\,\} + max\{\, S_T(-\varepsilon) - K\,,\, 0\,\}\,] \tag{9.21}$$

Figure 9.2 depicts the Monte Carlo results for the current price of the European call option with parameters defined in cells B2:B6. Random samples of ε are first generated in EXCEL spreadsheet, and they will be used to generate the sample maturity prices $S_T(\varepsilon)$ as well as their antithetic values $S_T(-\varepsilon)$. Sample values of the functions $f_T(\varepsilon)$, $\tilde{f}_T(\varepsilon)$, and $\hat{f}_T(\varepsilon)$ can then be evaluated for which mean and variance can be estimated. As shown in the figure, there are significant reductions in the size of the standard errors with the use of the variance reduction techniques. As reference, the Black-Scholes pricing in (9.17) is 12.34 for the European call option parameterized by B2:B6.

	A	B	C	D	E	F	G
1							
2	S_0 =	100					
3	K =	100					
4	T =	1.00	(years)				
5	r =	0.05	(per year)				
6	σ =	0.25	(per year)				
7					*Crude Simulation*	*Control Variate Method*	*Antithetic Variate Method*
8		ε	$S_T(\varepsilon)$	$S_T(-\varepsilon)$	$e^{-rT} f_T$	$e^{-rT} \tilde{f}_T$	$e^{-rT} \hat{f}_T$
9	1	1.049050384	162.1470287	75.38899024	28.85456642	-95.12942145	15.43228421
10	2	-1.204751932	75.39433978	137.7042362	0	-71.71731444	17.93268944
11	3	-1.811351962	64.78552479	160.2536987	0	-61.62589746	28.65754558
10007	9999	1.365806708	143.3618513	72.41898647	41.24706888	-95.12294245	
10008	10000	-0.96166111	80.11834936	129.5847962	0	-76.21093135	
10009					$E(\, e^{-rT} f_T\,)$	$S_0 + E(\, e^{-rT} \tilde{f}_T)$	$E(\, e^{-rT} \hat{f}_T)$
10010				*Mean*	12.38	12.20	12.29
10011				*Standard Error*	0.19	0.11	0.14
10012					(10000 samples)	(10000 samples)	(5000 samples)

Figure 9.2 : Monte Carlo simulation for the current price of a European call option.

An important application of Monte Carlo method is in the pricing of exotic options with intermediate boundary conditions. For example, a barrier knock-out option will have the maturity payoff depending on whether the underlying asset price has surpassed a predetermined barrier level prior to its maturity. Thus, it is sometimes necessary to generate the entire path of asset prices in discrete time increments $\{\, t_1, \ldots, t_N = T\,\}$ during the life of the option. To this end, we can use (9.15) to generate iteratively the risk-neutral asset price at t_{i+1} based on that at t_i as

$$S_{t_{i+1}} = S_{t_i}\, exp(\,(\, r - \tfrac{1}{2}\sigma^2\,)(\, t_{i+1} - t_i\,) + \sigma\sqrt{t_{i+1} - t_i}\;\varepsilon_{i+1}\,) \tag{9.22}$$

and trace out the path by running i from 0 to $N-1$ with random numbers $\{\, \varepsilon_1, \ldots, \varepsilon_N\,\}$ from $\varepsilon(\,0\,,\,1\,)$. The sample path can then be used to evaluate the sample option payoff according to some path-dependent conditions. It should be noted that we have defined in the iteration $t_0 = 0$ and the choice of time increments will depend on the boundary conditions of the option.

9.3. VBA Implementation

A crucial part of the Monte Carlo method is the generation of the standard normal random numbers. The procedure starts with a pseudo-random number generator (PRNG) that "randomly" produces a real number between 0 and 1 with uniform probability so that every number will have the same chance of being generated. The numbers generated by PRNG are not truly random in that they are completely determined by an initial *seed* integer. In fact, they are considered to be random only subject to standard statistical tests for randomness[2]. Uniform random numbers can be transformed into standard normal random numbers through Box-Muller algorithm as

$$\varepsilon_1 = \sqrt{-2ln(u_1)}\, cos(\, 2\pi u_2\,) \qquad (9.23)$$

$$\varepsilon_2 = \sqrt{-2ln(u_1)}\, sin(\, 2\pi u_2\,)$$

In (9.23), u_1 and u_2 are two uniform random numbers generated independently by PRNG. They can be transformed into a pair of standard normal random numbers ε_1 and ε_2 that are also independent. A more efficient transformation is to rewrite (9.23) so as to avoid the costly evaluation of trigonometric functions. This is known as the polar rejection algorithm given by

$$\varepsilon_1 = \sqrt{\frac{-2ln(w)}{w}}\,(\,2u_1 - 1\,) \qquad (9.24)$$

$$\varepsilon_2 = \sqrt{\frac{-2ln(w)}{w}}\,(\,2u_2 - 1\,) \quad , \text{ if } \; w = (\,2u_1 - 1\,)^2 + (\,2u_2 - 1\,)^2 \; < 1$$

The drawback in (9.24) is that the input uniform random numbers should stay inside the unit circle around the origin.

In EXCEL spreadsheet, PRNG is given by the **RAND**() function that automatically reseeds based on the system clock. The transformation of uniform random numbers into standard normal random numbers are conducted by the **NORMSINV**(**RAND**()) function that takes **RAND**() as input. In VBA implementation, it is highly inefficient to generate random numbers interactively through EXCEL as the number of calls will be enormous in Monte Carlo simulation. It is therefore essential to develop a VBA routine that implements for instance the polar rejection algorithm in (9.24). It can be defined through an external function called StdNormNum( ) that returns a standard normal random number on each request. The VBA code of this function is given by Code 9.1. In VBA, PRNG is given by the **Rnd**() function in which the generated sequence is completely determined by the default seed value. To prevent using repetitive sequences in Monte Carlo simulation, we can alter the seed based on the system clock by calling **Randomize** once in the main routine. It is important to note that we should not **Randomize** every time we call **Rnd**() as the random behavior can be badly skewed.

In both the Box-Muller and polar rejection algorithms, two usable standard normal random numbers are generated at the same time. In order to obtain maximum efficiency, we should utilize both of them by saving the unused random number for the next request. In Code 9.1, this can be done through two static variables *flagSave* and *snnSave* that retain their latest values after termination of the procedure. The static variable *flagSave* is initialized to 0 at the very first call to the function when *flagSave* is still an empty cell checked by the **IsEmpty** function. When *flagSave* = 0, the function generates two random numbers and returns only the one under *snnUse*. The second random number is saved under *snnSave* with *flagSave* set to 1. The two static variables will retain these values upon the

[2] See Chapter 7 of W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in C* : *The Art of Scientific Computing*, 2nd Edition, Cambridge University Press, 1997.

next request where the function simply returns the random number under *snnSave* and resets *flagSave* = 0.

```
Function StdNormNum() As Double
  Dim v1 As Double, v2 As Double, w As Double, fac As Double
  Dim snnUse As Double
  Static flagSave As Integer: If IsEmpty(flagSave) Then flagSave = 0
  Static snnSave As Double
  If (flagSave = 0) Then
newtrial:
    v1 = 2# * Rnd() - 1#
    v2 = 2# * Rnd() - 1#
    w = v1 ^ 2 + v2 ^ 2
    If (w >= 1#) Then GoTo newtrial
    fac = Sqr(-2# * Log(w) / w)
    snnSave = fac * v1
    snnUse = fac * v2
    flagSave = 1
  Else
    snnUse = snnSave
    flagSave = 0
  End If
  StdNormNum = snnUse
End Function
```

Code 9.1 : VBA code of the StdNormNum( ) function.

To generate a very long random sequence, it is advisable to use instead the standard algorithm for PRNG known as ran0( ) proposed by Park and Miller[8]. This generator has passed all theoretical tests so far and is definitely safer to use despite it is relatively slower than **Rnd**(). In Code 9.2 , we have included the VBA code of ran0( ) and refer the reader to the original literature for detail discussion of the algorithm. Like that in **Rnd**(), the seed value in ran0( ) should only be initiated once in the main routine. It can be any nonzero integer such as *seed* = 50789 and it must not be altered between successive calls in a random sequence.


```
Public seed As Long

Function ran0() As Double
  Dim IA As Long: IA = 16807
  Dim IM As Long: IM = 2147483647
  Dim IQ As Long: IQ = 127773
  Dim IR As Long: IR = 2836
  Dim MASK As Long: MASK = 123459876
  Dim AM As Double: AM = 1# / IM
  Dim k As Long
  seed = seed Xor MASK
  k = seed / IQ
  seed = IA * (seed - k * IQ) - IR * k
  If (seed < 0) Then seed = seed + IM
  ran0 = AM * seed
  seed = seed Xor MASK
End Function
```

Code 9.2 : VBA code of the ran0( ) function.

We consider again the pricing of a European call option written on a non-dividend paying stock as discussed in previous section. Here, it is efficient and convenient to develop VBA routines that separately perform the three different simulation tasks as demonstrated in Figure 9.2. The VBA codes of the McEuropeanCall( ) routine for crude simulation, the CMcEuropeanCall( ) routine for control variate method, and the AMcEuropeanCall( ) routine for antithetic variate method are given by Code 9.3. They all require the input parameters { S_0 , K , r , σ , T , n } and return the estimation of the current option price together with the standard error. In both routines, random sample of ε is generated by calling the StdNormNum( ) function, and it will be used to generate sample maturity price $S_T(\varepsilon)$

using (9.16). For antithetic variate method, the same random number with a negative sign will also be used to generate the antithetic price $S_T(-\varepsilon)$. Sample value of the functions $f_T(\varepsilon)$, $\tilde{f}_T(\varepsilon)$, and $\hat{f}_T(\varepsilon)$ are then evaluated respectively in different routines using (9.18), (9.19), and (9.21). The estimates of the mean and variance of their present values can be obtained by summing up all sample values and their squares as in (9.2) through a loop that runs from 1 to n. It is also convenient to implement an alternative expression for the variance estimate in (9.2) as

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} g^2(x_i) - \frac{n}{n-1} m^2 \tag{9.25}$$

In both routines, the mean estimate will be returned as the estimation of the current option price. For control variate method, it should be reminded to include in the mean estimate the analytic solution of S_0 for the control variate as discussed in (9.20). The standard error in the estimation can be determined by taking an additional factor of $1/\sqrt{n}$ in the variance estimate. Table 9.1 illustrates the relative performance of the three routines in terms of the sizes of standard errors and computation times. We adopt the same parameterization as in Figure 9.2 and consider one million samples in the Monte Carlo simulation. In this example, the control variate method is shown to be highly effective, while there is only mild improvement for the antithetic variate method.

	Crude Simulation	*Control Variate Method*	*Antithetic Variate Method*
Mean	12.338	12.334	12.340
Standard Error	0.018	0.011	0.010
Relative CPU Time	1.00	1.02	1.33

Table 9.1 : Relative performance of different Monte Carlo routines for European call option.

As our second example, we consider a European call option written on a stock with expected dividend payments $\{ D_1, \ldots, D_N \}$ at $\{ t_1, \ldots, t_N = T \}$ prior to its maturity. Assume S_{t_i} represents the asset price right after the dividend D_i has been paid at t_i. We can use (9.22) to first generate the asset price at t_{i+1} using S_{t_i} and then incorporate the price drop due to the payment D_{i+1} as

$$S_{t_{i+1}} = S_{t_i} \exp\left(\left(r - \tfrac{1}{2}\sigma^2 \right)\left(t_{i+1} - t_i \right) + \sigma\sqrt{t_{i+1} - t_i}\, \varepsilon_{i+1} \right) - D_{i+1} \tag{9.26}$$

In this way, the maturity price S_T of the asset can be generated iteratively by running i from 0 to $N-1$. If at any time the R.H.S. of (9.26) is less than or equal zero, the iteration should stop and restart all over again from $i = 0$. The maturity price can be used to evaluate the option payoff according to the function

$$f_T(\varepsilon_1, \ldots, \varepsilon_N) = max\{ S_T(\varepsilon_1, \ldots, \varepsilon_N) - K, 0 \} \tag{9.27}$$

For variance reduction, it is effective to adopt the case of a non-dividend paying stock as the control variate and consider the new function

$$\tilde{f}_T(\varepsilon_1, \ldots, \varepsilon_N) = max\{ S_T(\varepsilon_1, \ldots, \varepsilon_N) - K, 0 \} - max\{ S_T^{(0)}(\varepsilon_1, \ldots, \varepsilon_N) - K, 0 \} \tag{9.28}$$

The control price $S_T^{(0)}$ at maturity is generated simultaneously with S_T using the same random sequence but with zero dividend payments. The control variate has an analytic solution given by the Black-Scholes formula in (9.17) for which (9.13) can be written as

$$f_0 = f_0^{BS} + \hat{E}\left(e^{-rT} \tilde{f}_T \mid S_0 \right) \tag{9.29}$$

It is always straightforward to take $-\varepsilon$ as the antithetic variate and generate the antithetic value of S_T using the negative sequence $\{ -\varepsilon_1, \ldots, -\varepsilon_N \}$. It is then convenient to introduce the new function

$$\hat{f}_T(\varepsilon_1, \ldots, \varepsilon_N) = \tfrac{1}{2}\,[\,max\{\,S_T(\varepsilon_1, \ldots, \varepsilon_N) - K, 0\,\} + max\{\,S_T(-\varepsilon_1, \ldots, -\varepsilon_N) - K, 0\,\}\,] \qquad (9.30)$$

as the two payoffs are negatively correlated. However, it has been shown below that the effectiveness of the antithetic variate method in (9.30) is limited.

The pseudo code of the McEuropeanCallDiv( ) routine for crude simulation is given by Code 9.4. The routine reads in the parameters { S_0 , K , r , σ , n } as well as the details of dividend payments as N, { $t_1, \ldots, t_N = T$ }, and { $D_1, \ldots, D_N$ }. To initiate (9.26), the current time $t_0 = 0$ by definition should also be included in the input time array. Similarly, the routine returns the estimation of the current option price together with the standard error. In the inner loop, sample maturity price S_T of the asset is generated iteratively by running i in (9.26) from 0 to $N - 1$. The iteration starts off from S_0 and moves forward by calling the StdNormNum( ) function successively. It should restart from S_0 at the beginning of the loop if the asset price is less than or equal zero at any time. Sample payoff f_T in (9.27) and its present value can be evaluated using the asset price immediately after the iteration. The estimates of the mean and variance can be obtained based on the sample sum and squared sum accumulated through the outer loop that runs from sample number 1 to n. The VBA code of the McEuropeanCallDiv( ) routine is given by Code 9.5.

It is easy to modify the algorithm for crude simulation and include the appropriate procedures for variance reduction. The VBA code of the CMcEuropeanCallDiv( ) routine for the control variate method is given by Code 9.6. In the inner loop, the iteration of the control price is performed simultaneously with S_T using the same random sequence and the same starting price but with no dividend payment. Again, we should restart both iterations if either one of them is less than or equal zero. In this way, sample value of the function $\tilde{f}_T$ in (9.28) can be evaluated based on the iterated prices. In the estimation of the option price, we have also included the Black-Scholes formula given by the function BsEuropeanCall( ). The pseudo code of the AMcEuropeanCallDiv( ) routine for the antithetic variate method is also given by Code 9.6. In this case, the antithetic price is generated simultaneously with S_T using however the negative sequence. Similarly, sample value of the function $\hat{f}_T$ in (9.30) can be evaluated based on the prices immediately after the iteration. Table 9.2 illustrates the relative performance of the three routines based on one million simulation samples. We adopt the same parameterization as in Figure 9.2 and consider the dividend payments of $D_{1,2,3,4} = 0.25$ at quarterly time intervals { $t_1 = 0.25$, $t_2 = 0.50$, $t_3 = 0.75$, $t_4 = 1.00$ }. Again, the control variate method is shown to be much more effective than the antithetic variate method.

	Crude Simulation	*Control Variate Method*	*Antithetic Variate Method*
Mean	11.782	11.794	11.802
Standard Error	0.018	0.0005	0.010
Relative CPU Time	1.00	1.31	1.36

Table 9.2 : Relative performance of different Monte Carlo routines for European call option written on dividend paying asset.

9.4. Exotic Options

In this section, we apply the Monte Carlo method in the pricing of exotic options with path-dependent payoffs at maturity. For example, an Asian call option will have its payoff depending on the average asset price over a set of predetermined forward time { $t_1, t_2, \ldots, t_N = T$ } given by

$$A_T = \frac{1}{N}\,(\,S_{t_1} + \ldots + S_{t_N}\,) \qquad (9.31)$$

Again, we can generate iteratively the asset prices at every averaging time by running the index i from 0 to $N-1$ in (9.22) with random numbers $\{ \varepsilon_1, \ldots, \varepsilon_N \}$. The average asset price can be used to evaluate the option payoff according to the function

$$f_T(\varepsilon_1, \ldots, \varepsilon_N) = max\{ A_T(\varepsilon_1, \ldots, \varepsilon_N) - K, 0 \} \tag{9.32}$$

For variance reduction, it is effective to adopt the case for the geometric average asset price as the control variate and consider the new function

$$\tilde{f}_T(\varepsilon_1, \ldots, \varepsilon_N) = max\{ A_T(\varepsilon_1, \ldots, \varepsilon_N) - K, 0 \} - max\{ G_T(\varepsilon_1, \ldots, \varepsilon_N) - K, 0 \} \tag{9.33}$$

The geometric average asset price is defined as

$$G_T = \sqrt[N]{S_{t_1} \ldots S_{t_N}} \tag{9.34}$$

It is easy to show that G_T is lognormally distributed with risk-neutral average given by

$$\hat{E}(G_T \mid S_0) = S_0 e^{r\nu_1 + \frac{1}{2}\sigma^2(\nu_2 - \nu_1)} \quad , \quad \nu_\alpha = \frac{1}{N^\alpha} \sum_{i=1}^{N} (N - i + 1)^\alpha (t_i - t_{i-1}) \tag{9.35}$$

The control variate has an analytic solution given by the Black-Scholes formula as[3]

$$f_0^{BS} = \hat{E}(G_T \mid S_0) e^{-rT} N(\rho) - K e^{-rT} N(\rho - \sigma\sqrt{\nu_2}), \quad \rho = \frac{ln(\hat{E}(G_T \mid S_0) / K) + \frac{1}{2}\sigma^2\nu_2}{\sigma\sqrt{\nu_2}} \tag{9.36}$$

The pseudo code of the CMcArAsianCall( ) routine for the pricing of an Asian call option is given by Code 9.7. The routine reads in the parameters $\{ S_0, K, r, \sigma, n \}$, a set of forward averaging time $\{ t_1, \ldots, t_N = T \}$, and N. Similarly, the input time array should also include the current time defined as $t_0 = 0$ to initiate (9.22). Again, the routine returns the estimation of the current option price together with the standard error. In the inner loop, asset prices at every averaging time are generated iteratively for which their arithmetic and geometric averages can be evaluated. It should be noted that the multiplication of a long sequence of prices could possibly overflow the numerical procedure. It is safer to determine the geometric average by summing up the logarithmic prices. Sample payoff $\tilde{f}_T$ in (9.33) and its present value can be evaluated using the iterated average prices. The estimates of the mean and variance can be obtained based on the sample sum and squared sum. We have also included the Black-Scholes formula for the geometric case given by the function BsGeAsianCall( ). The VBA code of the CMcArAsianCall( ) routine is given by Code 9.8. Table 9.3 demonstrates the effectiveness of the use of geometric average price as control variate based on one million simulation samples. We adopt the same parameterization as in Figure 9.2 and consider the set of averaging time $\{ t_1 = 0.50, t_2 = 0.75, t_3 = 1.00 \}$.

	Crude Simulation	*Control Variate Method*
Mean	9.691	9.6787
Standard Error	0.014	0.0004
Relative CPU Time	1.00	1.12

Table 9.3 : Relative performance of the CMcArAsianCall( ) routine as compared with crude simulation.

As our second example on exotic option, we consider a double barrier knock out (DKO) European call option with maturity payoff depending on the breaching condition of the asset price with respect to two predefined barriers ($H > S_0 > L$). In principle, the intermediate knock-out condition is

[3] See A. Kemna and A. Vorst, "A pricing method for options based on average asset values", *Journal of Banking and Finance*, Vol. **14**, Issue 1 (1990) p 113-129.

subjected to continuous asset price movement between time 0 and T. In (9.22), we can adopt equal time interval of $\Delta t = T/N$ for large N such that the generated prices $\{ S_{t_0} = S_0, S_{t_1}, \ldots, S_{t_N} = S_T \}$ at discrete time $t_i = i\Delta t$ will be a good approximation of a continuous path. Thus, we can generate iteratively the asset price movement for the entire life of the option by running i from 0 to $N - 1$ as

$$S_{t_{i+1}} = S_{t_i} \exp\left((r - \tfrac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t}\, \varepsilon_{i+1} \right) \tag{9.37}$$

with random numbers $\{ \varepsilon_1, \ldots, \varepsilon_N \}$. If the asset price has hit either one of the barriers at any time prior to the option maturity, the option is immediately knocked out and the payoff is set to zero or to a constant amount of rebate c payable upon hitting. Otherwise, it will be evaluated according to the maturity price based on a call-type payoff function as

$$f_T(\varepsilon_1, \ldots, \varepsilon_N) = \begin{cases} c\, e^{r(T - t_h)}, & \text{if } S_{t_h}(\varepsilon_1, \ldots, \varepsilon_h) \geq H \text{ or } S_{t_h}(\varepsilon_1, \ldots, \varepsilon_h) \leq L \text{ for } t_h \leq T \\ max\{ S_T(\varepsilon_1, \ldots, \varepsilon_N) - K, 0 \}, & \text{otherwise} \end{cases} \tag{9.38}$$

For variance reduction, it is difficult in this case to identify a control variate that works consistently and effectively for a general barrier condition. We have also checked that the antithetic price movement generated through the negative random sequence doesn't work well for barrier option.

The pseudo code of the McDKOCall( ) routine for the crude pricing of a DKO European call option is given by Code 9.9. The routine reads in the parameters $\{ S_0, K, H, L, \sigma, r, c, N, T, n \}$ and returns the estimation of the option price together with the standard error. Here, the intermediate knock-out condition is examined through an external procedure called DKOBoundary( ) that assigns *crossflag* = TRUE if the input price has surpassed either one of the barriers H and L. It is essential to first examine the knock-out condition at current time and return the trivial solution $f_0 = 0$ with no estimation error. In the inner loop, sample asset price movement with time interval Δt is generated iteratively using (9.37) starting from its current value S_0. The knock-out condition is checked at every time step by calling DKOBoundary( ) for the newly iterated price. If *crossflag* = TRUE at any time, the hitting time is tagged as t_h and the iteration should stop immediately. Otherwise, it should continue through the entire loop and exit as the maturity price S_T. In either case, sample payoff f_T in (9.38) can be evaluated according to the flag *crossflag* in conjunction with the hitting time or the maturity price. The VBA code of the McDKOCall( ) routine is given by Code 9.10.

There are two sources of error in the Monte Carlo pricing of the DKO option. Firstly, the method relies on statistical estimation and the statistical error is governed by the sample size. Secondly, the statistical estimates are actually referring to the case with discrete time steps. Thus, there is systematic error in the method as we are not estimating the true option price under continuous price movement. The error would depend on the size of Δt in the configuration of the problem. It can be improved by taking smaller time steps, but there is a tradeoff between computational time and accuracy. Table 9.4 demonstrates the significance of the size of Δt in the pricing based on one million simulation samples. We adopt the same parameterization as in Figure 9.2 and consider the barrier levels of $H = 120$ and $L = 80$ with zero rebate $c = 0$. It can be seen in the table that the effect is quite critical especially for the mean estimate. We have shown in Figure 9.3 that the systematic error[4] for the mean estimate will only

[4] Consider the systematic error given by $| m_{\Delta t} - m_{true} | \approx a(\Delta t)^b$ where a and b are constants. As m_{true} is not known a priori, we can choose m_{best} to be the best estimate in Table 9.4 and write

$$log_{10}| m_{\Delta t} - m_{best} | = b\, log_{10}\Delta t + log_{10} a + log_{10}| 1 - \delta |, \quad \delta = (m_{best} - m_{true})/(m_{\Delta t} - m_{true})$$

In the region where Δt is not too small, the term δ is negligible and b can be estimated by the slope of the plot $log_{10}| m_{\Delta t} - m_{best} |$ *versus* $log_{10}\Delta t$ in Figure 9.3. It can be shown that $b \approx 0.5$ in the linear fitting.

decrease like the square root of Δt. If the size of Δt is not small enough, the error will be quite substantial.

	$\Delta t = 0.1$ ($N = 10$)	$\Delta t = 0.01$ ($N = 100$)	$\Delta t = 0.001$ ($N = 1000$)	$\Delta t = 0.0001$ ($N = 10000$)
Mean	1.2328	0.7533	0.6089	0.5661
Standard Error	0.0035	0.0027	0.0024	0.0023
Relative CPU Time	1.00	7.51	69.61	697.22

Table 9.4 : Relative performance of the McDKOCall( ) routine under different configurations of time step.

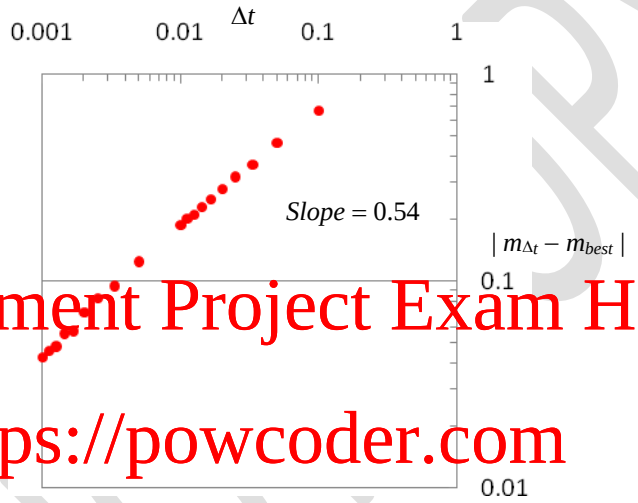

Figure 9.3 : The plot of systematic error versus Δt for the pricing of the DKO call option.

9.5. American Options

One of the most important problems in option pricing is the valuation of American-style options with early exercising features. For example, an American put option can be exercised at any time prior to its maturity with intrinsic value evaluated according to the payoff function $\psi(S) = max\{ K - S , 0 \}$. At time t with underlying asset price S_t, the fair value of an American put option can be defined based on the risk-neutral expectation of its discounted payoff as

$$F(S_t , t) = \hat{E}(e^{-r(\tau_s - t)} \psi(S_{\tau_s}) \mid S_t) \tag{9.39}$$

where τ_s is the first exercising time along a random asset price movement starting off from S_t. It can occur at option's maturity or at any earlier time when the intrinsic value is greater than the fair value at that point. Similarly, it is optimal to early exercise the option at time t based on the criteria as

$$\psi(S_t) \geq F(S_t , t) \tag{9.40}$$

The price of the American put option should also include this feature and is given by

$$f(S_t , t) = max\{ F(S_t , t) , \psi(S_t) \} \tag{9.41}$$

The equality condition in (9.40) can serve to define a critical price $S_c(t)$ at time t for which the early exercising criteria will be satisfied[5] whenever $S_t \le S_c(t)$. In this way, the criteria can be implemented very easily by checking the asset price with respect to a critical price. It should be noted that the fair value $F(S_t, t)$ can only be evaluated when all critical prices, or a boundary, at forward time of t have been determined. This means that critical price $S_c(t)$ can only be determined through backward iteration starting off from the option's maturity with $S_c(T) = K$. To determine the current price of the American put option, we need to first generate the entire critical boundary prior to its maturity and then evaluate the risk-neutral expectation as

$$f_0 = max\{ \hat{E}(e^{-r\tau_s} \psi(S_{\tau_s}) \mid S_0) , \psi(S_0) \} \tag{9.42}$$

Again, we can adopt equal time interval of $\Delta t = T/N$ for large N and consider the discrete time steps $t_i = i\Delta t$ for the entire life of the option with i runs from 0 to N. At time t_i, consider $S_{t_i} = x$ and suppose the critical boundary at forward time is given by $\{ S_c(t_{i+1}) , \dots , S_c(t_N) = K \}$. The fair value of the American put option can be estimated by Monte Carlo simulation as

$$F(S_{t_i} = x , t_i) = \hat{E}(e^{-r(\tau_s - t_i)} \psi(S_{\tau_s}) \mid S_{t_i} = x) \tag{9.43}$$

In (9.43), we can generate iteratively the asset price movement for the remaining life of the option by running j in (9.44) from i to $N - 1$ starting from $S_{t_i} = x$.

$$S_{t_{j+1}} = S_{t_j} exp[(r - \tfrac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t}\, \varepsilon_j] \tag{9.44}$$

If the generated price $S_{t_{j+1}} \le S_c(t_{j+1})$ at any forward time prior to the option's maturity, the option should be exercised immediately and $\tau_s = t_{j+1}$ in (9.43). Otherwise, it will be exercised at maturity for positive payoff and $\tau_s = T$. The critical price $S_c(t_i)$ at time t_i can be determined by identifying the value of x for which the mean estimate in (9.43) matches the intrinsic value. This is the same as solving the root of the function given by

$$y(x) = F(x , t_i) - \psi(x) \tag{9.45}$$

where $y(x)$ is subjected to the standard error in the estimation of $F(x, t_i)$. It can be done by performing a linear regression for $y(x) \approx \beta x + \alpha$ in the neighborhood[6] of its trial value $x = S_c(t_{i+1})$ with $x_{root} \approx -\alpha/\beta$. It is essential for the size of $y(x_{root})$ to be less than the standard error in $F(x, t_i)$. Otherwise, a new x_{root} should be determined through another linear regression around the old one until $y(x_{root})$ becomes zero within the statistical error. In this way, the entire critical boundary can be determined by iterating the above procedure for all t_i in a backward scheme starting from t_{N-1} to t_0 with $S_c(t_N) = K$.

We first develop a routine called FvAmericanPut( ) capable of estimating the fair value of an American put option at forward time based on Monte Carlo simulation. The pseudo code of FvAmericanPut( ) is given by Code 9.11. It reads in the parameters $\{ x , K , r , \sigma , N , T , n \}$, a time pointer i for the reference forward time t_i, and the critical prices $\{ S_c(t_{i+1}) , \dots , S_c(t_N) \}$ thereafter. The routine returns the Monte Carlo estimation of the fair value together with the standard error. In the inner loop, asset price movement between t_i and T is generated iteratively using (9.44) with time interval of Δt. At each time step, the early exercising criteria is examined through an external procedure called APBoundary( ) that assigns *exerciseflag* = TRUE if the iterated asset price is less than the critical price. If *exerciseflag* = TRUE at any time, the iteration should terminate immediately with stopping time taken to be the updated τ_s in the loop. Otherwise, it should continue until the

[5] This is true as both $\psi(S_t)$ and $F(S_t, t)$ are decreasing function with S_t. Also, $\psi(S_t)$ vanishes for large S_t.
[6] Avoid taking the out-of-money region of the option by virtue of the linear approximation for (9.45).

option matures at which $\tau_s = T$. The discounted payoff in (9.43) can then be evaluated using the stopping time and the terminal asset price in the iteration. The VBA code of FvAmericanPut( ) is given by Code 9.12.

We then develop a routine called McAmericanPut( ) that performs the backward iteration based on the FvAmericanPut( ) routine. The pseudo code of McAmericanPut( ) is given by Code 9.13. It reads in the parameters { S_0 , K , r , σ , N , T , n } and returns the estimation of the option price together with the standard error. In the backward iteration, we can generate the critical boundary by running the time pointer i in FvAmericanPut( ) backward from $N - 1$ to 0 with $S_c(t_N) = K$ being the starting value. At time t_i in the loop, the critical prices from t_{i+1} to t_N are presumably calculated in previous steps. It is then straightforward to estimate $F(x , t_i)$ in (9.45) by calling FvAmericanPut( ) and the main task is to determine the root of $y(x)$ that corresponds to the critical price $S_c(t_i)$ at time t_i. We adopt $S_c(t_{i+1})$ as the trial value of the root and consider a linear regression for $y(x)$ in the neighborhood of this value. In the inner loop, we generate $h = 100$ data points for the regression with values of x that spread across a small region of $\delta = 1\%$ below the trial value. The linear regression can be performed through an external routine called Regn( ) and it should be noted that the errors in $y(x)$ are roughly constant[7]. The resulting α and β would provide a good approximation for x_{root} and the corresponding $y(x_{root})$ should be calculated by calling FvAmericanPut( ). In the IF statement, it is important to check explicitly that $y(x_{root})$ is zero within the standard error in the estimation. If so, we can assign x_{root} to be the critical price $S_c(t_i)$. Otherwise, it can be used to update the trial value and the regression should be repeated starting from the line labeled as "NewTrialValue". The VBA code of McAmericanPut( ) is given by Code 9.14.

Once we have determined the critical boundary in the backward iteration, the current price of the American put option can be estimated by calling FvAmericanPut( ) with S_0 and compares the fair value with the intrinsic value according to (9.42). The systematic error here would depend on the accuracy of the critical boundary determined under discrete time interval. Figure 9.4 illustrates the deviation in the critical boundaries determined based on, for example, $\Delta t = 0.1$ and $\Delta t = 0.001$. Table 9.5 demonstrates the overall significance of the size of Δt in the pricing of the American put option. We adopt the same parameterization as in Figure 9.2 and consider again one million samples in the simulation. It has been shown in Figure 9.5 that the systematic error for the mean estimate will only decrease like the linear order of Δt. Thus, the error will be less substantial here than in the pricing of DKO option.

	$\Delta t = 0.1$ ($N = 10$)	$\Delta t = 0.01$ ($N = 100$)	$\Delta t = 0.001$ ($N = 1000$)
Mean	7.909	7.982	7.985
Standard Error	0.009	0.009	0.009
Relative CPU Time	1.00	26.76	752.62

Table 9.5 : Relative performance of the McAmericanPut( ) routine under different configurations of time step.

[7] In the fitting of a set of h data points (x_k , y_k) to a straight line $y = \beta x + \alpha$ where y is subjected to a constant measurement error, the best fit parameters in the linear model are given by

$$\beta = (\langle xy \rangle - \langle x \rangle \langle y \rangle) / \Delta \quad , \quad \alpha = (\langle x^2 \rangle \langle y \rangle - \langle x \rangle \langle xy \rangle) / \Delta \quad , \quad \Delta = \langle x^2 \rangle - \langle x \rangle^2$$

The estimation errors of α and β are both in the order of $1/\sqrt{h}$. Thus, the size of h must be sufficiently large to ensure stability in the iteration of the critical boundary.

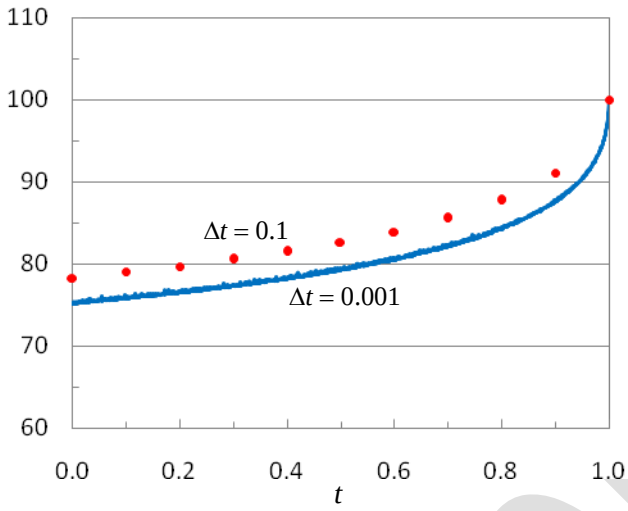

Figure 9.4 : Critical boundary of the American put option.

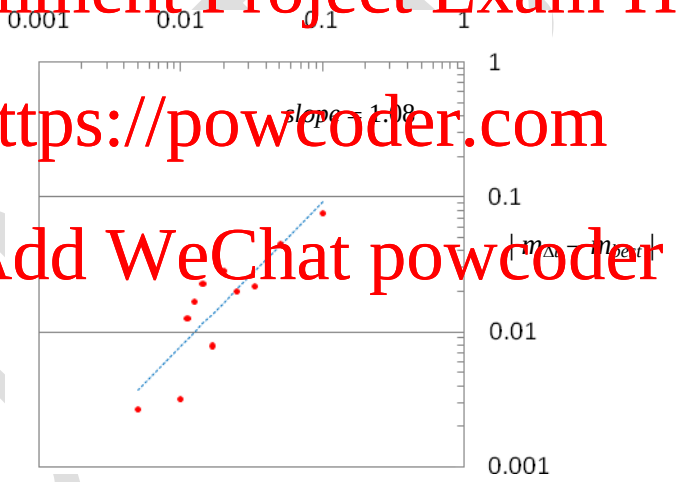

Figure 9.5 : The plot of systematic error versus Δt for the pricing of the American put option.

```
Sub McEuropeanCall(assetPrice As Double, strike As Double, riskFree As Double, sigma As Double, maturity As Double,
                   nsample As Long, ByRef optionPrice As Double, ByRef stdErr As Double)
  Dim sum As Double, sum2 As Double, gen As Double
  Dim mean As Double, sd As Double, Ls As Long
  Dim St As Double, fT As Double, pV As Double
  sum = 0
  sum2 = 0
  For Ls = 1 To nsample
    gen = StdNormNum()
    St = assetPrice * Exp((riskFree - 0.5 * sigma ^ 2) * maturity + sigma * Sqr(maturity) * gen)
    fT = CallPayoff(strike, St)
    pV = Exp(-riskFree * maturity) * fT
    sum = sum + pV
    sum2 = sum2 + pV * pV
  Next Ls
  mean = sum / nsample
  sd = Sqr(sum2 / (nsample - 1) - (nsample / (nsample - 1)) * mean ^ 2)
  optionPrice = mean
  stdErr = sd / Sqr(nsample)
End Sub
```

```
Sub CMcEuropeanCall(assetPrice As Double, strike As Double, riskFree As Double, sigma As Double, maturity As Double,
                    nsample As Long, ByRef optionPrice As Double, ByRef stdErr As Double)
  Dim sum As Double, sum2 As Double, gen As Double
  Dim mean As Double, sd As Double, Ls As Long
  Dim St As Double, fT As Double, pV As Double
  sum = 0
  sum2 = 0
  For Ls = 1 To nsample
    gen = StdNormNum()
    St = assetPrice * Exp((riskFree - 0.5 * sigma ^ 2) * maturity + sigma * Sqr(maturity) * gen)
    fT = CallPayoff(strike, St) - St
    pV = Exp(-riskFree * maturity) * fT
    sum = sum + pV
    sum2 = sum2 + pV * pV
  Next Ls
  mean = sum / nsample
  sd = Sqr(sum2 / (nsample - 1) - (nsample / (nsample - 1)) * mean ^ 2)
  optionPrice = assetPrice + mean
  stdErr = sd / Sqr(nsample)
End Sub
```

```
Sub AMcEuropeanCall(assetPrice As Double, strike As Double, riskFree As Double, sigma As Double, maturity As Double,
                    nsample As Long, ByRef optionPrice As Double, ByRef stdErr As Double)
  Dim sum As Double, sum2 As Double, gen As Double
  Dim mean As Double, sd As Double, Ls As Long
  Dim St As Double, Sta As Double, fT As Double, pV As Double
  sum = 0
  sum2 = 0
  For Ls = 1 To nsample
    gen = StdNormNum()
    St = assetPrice * Exp((riskFree - 0.5 * sigma ^ 2) * maturity + sigma * Sqr(maturity) * gen)
    Sta = assetPrice * Exp((riskFree - 0.5 * sigma ^ 2) * maturity + sigma * Sqr(maturity) * (-gen))
    fT = (CallPayoff(strike, St) + CallPayoff(strike, Sta)) / 2
    pV = Exp(-riskFree * maturity) * fT
    sum = sum + pV
    sum2 = sum2 + pV * pV
  Next Ls
  mean = sum / nsample
  sd = Sqr(sum2 / (nsample - 1) - (nsample / (nsample - 1)) * mean ^ 2)
  optionPrice = mean
  stdErr = sd / Sqr(nsample)
End Sub
```

```
Function CallPayoff(strike As Double, assetPrice As Double) As Double
  CallPayoff = Max(assetPrice - strike, 0)
End Function

Function Max(x As Double, y As Double) As Double
  If x > y Then Max = x Else Max = y
End Function
```

Code 9.3 : VBA codes of the McEuropeanCall( ), CMcEuropeanCall( ), and AMcEuropeanCall( ) routines.

McEuropeanCallDiv(S_0 , K , r , σ , N , $t(0:N)$, $D(1:N)$, n , f_0 , *error*)

zeroize the sample sum and squared sum
$sum = 0$, $sum2 = 0$

For($L_s = 1$ to n){

initialize the asset price
$S_t = S_0$

# generate the asset price at each dividend date
For($i = 0$ to $N - 1$){

ε = StdNormNum( )

$S_t = S_t \exp\big((r - \tfrac{1}{2}\sigma^2)[t(i + 1) - t(i)] + \sigma\sqrt{ t(i + 1) - t(i) }\, \varepsilon \big) - D(i + 1)$

If($S_t \le 0$) go back to the statement $S_t = S_0$ and restart all over again }

evaluate the option payoff function as in (9.27)
f_T = CallPayoff(K, S_t)
$PV = e^{-rT} f_T$

accumulate the sample sum and squared sum
$sum = sum + PV$
$sum2 = sum2 + PV^2$ }

evaluate the estimates of mean and variance
$m = sum / n$

$$s = \sqrt{\frac{1}{n-1} sum2 - \frac{n}{n-1} m^2}$$

return the estimation of option price and standard error
$f_0 = m$
$error = s / \sqrt{n}$

Code 9.4 : Pseudo code of the McEuropeanCallDiv( ) routine.

```
Sub McEuropeanCallDiv(assetPrice As Double, strike As Double, riskFree As Double, sigma As Double, nDiv As Integer,
                      timeDiv() As Double, paymentDiv() As Double, nsample As Long, ByRef optionPrice As Double,
                      ByRef stdErr As Double)
  Dim sum As Double, sum2 As Double, gen As Double
  Dim mean As Double, sd As Double, Ls As Long
  Dim St As Double, fT As Double, pV As Double, i As Integer
  sum = 0
  sum2 = 0
  For Ls = 1 To nsample
NewTrial:
    St = assetPrice
    For i = 0 To nDiv - 1
      gen = StdNormNum()
      St = St * Exp((riskFree - 0.5 * sigma ^ 2) * (timeDiv(i + 1) - timeDiv(i)) + sigma * Sqr(timeDiv(i + 1) - timeDiv(i)) * gen) - paymentDiv(i + 1)
      If (St <= 0) Then GoTo NewTrial
    Next i
    fT = CallPayoff(strike, St)
    pV = Exp(-riskFree * timeDiv(nDiv)) * fT
    sum = sum + pV
    sum2 = sum2 + pV * pV
  Next Ls
  mean = sum / nsample
  sd = Sqr(sum2 / (nsample - 1) - (nsample / (nsample - 1)) * mean ^ 2)
  optionPrice = mean
  stdErr = sd / Sqr(nsample)
End Sub
```

Code 9.5 : VBA code of the McEuropeanCallDiv( ) routine.

```
Sub CMcEuropeanCallDiv(assetPrice As Double, strike As Double, riskFree As Double, sigma As Double, nDiv As Integer,
                       timeDiv() As Double, paymentDiv() As Double, nsample As Long, ByRef optionPrice As Double,
                       ByRef stdErr As Double)
  Dim sum As Double, sum2 As Double, gen As Double
  Dim mean As Double, sd As Double, Ls As Long
  Dim St As Double, St0 As Double, fT As Double, pV As Double, i As Integer
  sum = 0
  sum2 = 0
  For Ls = 1 To nsample
NewTrial:
    St = assetPrice
    St0 = assetPrice
    For i = 0 To nDiv - 1
      gen = StdNormNum()
      St = St * Exp((riskFree - 0.5 * sigma ^ 2) * (timeDiv(i + 1) - timeDiv(i)) + sigma * Sqr(timeDiv(i + 1) - timeDiv(i)) * gen) - paymentDiv(i + 1)
      St0 = St0 * Exp((riskFree - 0.5 * sigma ^ 2) * (timeDiv(i + 1) - timeDiv(i)) + sigma * Sqr(timeDiv(i + 1) - timeDiv(i)) * gen)
      If (St <= 0 Or St0 <= 0) Then GoTo NewTrial
    Next i
    fT = CallPayoff(strike, St) - CallPayoff(strike, St0)
    pV = Exp(-riskFree * timeDiv(nDiv)) * fT
    sum = sum + pV
    sum2 = sum2 + pV * pV
  Next Ls
  mean = sum / nsample
  sd = Sqr(sum2 / (nsample - 1) - (nsample / (nsample - 1)) * mean ^ 2)
  optionPrice = BsEuropeanCall(assetPrice, strike, riskFree, sigma, timeDiv(nDiv)) + mean
  stdErr = sd / Sqr(nsample)
End Sub
```



```
Sub AMcEuropeanCallDiv(assetPrice As Double, strike As Double, riskFree As Double, sigma As Double, nDiv As Integer,
                       timeDiv() As Double, paymentDiv() As Double, nsample As Long, ByRef optionPrice As Double,
                       ByRef stdErr As Double)
  Dim sum As Double, sum2 As Double, gen As Double
  Dim mean As Double, sd As Double, Ls As Long
  Dim St As Double, Sta As Double, fT As Double, pV As Double, i As Integer
  sum = 0
  sum2 = 0
  For Ls = 1 To nsample
NewTrial:
    St = assetPrice
    Sta = assetPrice
    For i = 0 To nDiv - 1
      gen = StdNormNum()
      St = St * Exp((riskFree - 0.5 * sigma ^ 2) * (timeDiv(i + 1) - timeDiv(i)) + sigma * Sqr(timeDiv(i + 1) - timeDiv(i)) * gen) - paymentDiv(i + 1)
      Sta = Sta * Exp((riskFree - 0.5 * sigma ^ 2) * (timeDiv(i + 1) - timeDiv(i)) + sigma * Sqr(timeDiv(i + 1) - timeDiv(i)) * (-gen)) - paymentDiv(i + 1)
      If (St <= 0 Or Sta <= 0) Then GoTo NewTrial
    Next i
    fT = (CallPayoff(strike, St) + CallPayoff(strike, Sta)) / 2
    pV = Exp(-riskFree * timeDiv(nDiv)) * fT
    sum = sum + pV
    sum2 = sum2 + pV * pV
  Next Ls
  mean = sum / nsample
  sd = Sqr(sum2 / (nsample - 1) - (nsample / (nsample - 1)) * mean ^ 2)
  optionPrice = mean
  stdErr = sd / Sqr(nsample)
End Sub
```



```
Function BsEuropeanCall(assetPrice As Double, strike As Double, riskFree As Double, sigma As Double, maturity As Double) As Double
  Dim d1 As Double, d2 As Double
  d1 = (Log(assetPrice / strike) + (riskFree + 0.5 * sigma ^ 2) * maturity) / (sigma * Sqr(maturity))
  d2 = d1 - sigma * Sqr(maturity)
  With Application.WorksheetFunction
    BsEuropeanCall = assetPrice * .NormSDist(d1) - strike * Exp(-riskFree * maturity) * .NormSDist(d2)
  End With
End Function
```

Code 9.6 : VBA codes of the CMcEuropeanCallDiv( ), and AMcEuropeanCallDiv( ) routines.

CMcArAsianCall(S_0 , K , r , σ , N , $t(0:N)$, n , f_0 , *error*)

zeroize the sample sum and squared sum
sum = 0 , *sum*2 = 0

For($L_s = 1$ to n){

initialize the asset price, sum of price and logarithmic sum of price
$S_t = S_0$, $\Sigma_S = 0$, $\Sigma_{lnS} = 0$

# generate the asset price at each averaging time
For($i = 0$ to $N - 1$){ ε = StdNormNum( )

$$S_t = S_t \, exp\Big((r - \tfrac{1}{2}\sigma^2)[t(i+1) - t(i)] + \sigma \sqrt{ t(i+1) - t(i) }\, \varepsilon \Big)$$

$$\Sigma_S = \Sigma_S + S_t$$

$$\Sigma_{lnS} = \Sigma_{lnS} + ln(S_t) \quad \}$$

evaluate the arithmetic and geometric average asset prices
$A_T = \Sigma_S / N$
$G_T = exp(\Sigma_{lnS} / N)$

evaluate the option payoff function as in (9.33)
$f_T = CallPayoff(K , A_T) - CallPayoff(K , G_T)$
$PV = e^{-rT} f_T$

accumulate the sample sum and squared sum
$sum = sum + PV$
$sum2 = sum2 + PV^2$ }

evaluate the estimates of mean and variance
$m = sum / n$

$$s = \sqrt{ \frac{1}{n-1} sum2 - \frac{n}{n-1} m^2 }$$

return the estimation of option price and standard error
$f_0 = BsGeAsianCall(S_0 , K , r , \sigma , N , t(0:N)) + m$
$error = s / \sqrt{n}$

Code 9.7 : Pseudo code of the CMcArAsianCall( ) routine.

```
Sub CMcArAsianCall(assetPrice As Double, strike As Double, riskFree As Double, sigma As Double, nAvg As Integer,
                   timeAvg() As Double, nsample As Long, ByRef optionPrice As Double, ByRef stdErr As Double)
  Dim sum As Double, sum2 As Double, gen As Double
  Dim mean As Double, sd As Double, Ls As Long
  Dim St As Double, fT As Double, pV As Double, i As Integer
  Dim sumPrice As Double, sumLnPrice As Double
  Dim amAvg As Double, gmAvg As Double
  sum = 0
  sum2 = 0
  For Ls = 1 To nsample
    St = assetPrice
    sumPrice = 0
    sumLnPrice = 0
    For i = 0 To nAvg - 1
      gen = StdNormNum()
      St = St * Exp((riskFree - 0.5 * sigma ^ 2) * (timeAvg(i + 1) - timeAvg(i)) + sigma * Sqr(timeAvg(i + 1) - timeAvg(i)) * gen)
      sumPrice = sumPrice + St
      sumLnPrice = sumLnPrice + Log(St)
    Next i
    amAvg = sumPrice / nAvg
    gmAvg = Exp(sumLnPrice / nAvg)
    fT = CallPayoff(strike, amAvg) - CallPayoff(strike, gmAvg)
    pV = Exp(-riskFree * timeAvg(nAvg)) * fT
    sum = sum + pV
    sum2 = sum2 + pV * pV
  Next Ls
  mean = sum / nsample
  sd = Sqr(sum2 / (nsample - 1) - (nsample / (nsample - 1)) * mean ^ 2)
  optionPrice = BsGeAsianCall(assetPrice, strike, riskFree, sigma, nAvg, timeAvg) + mean
  stdErr = sd / Sqr(nsample)
End Sub
```



```
Function BsGeAsianCall(assetPrice As Double, strike As Double, riskFree As Double, sigma As Double, nAvg As Integer,
                       timeAvg() As Double) As Double
  Dim meanG As Double
  Dim nu1 As Double, nu2 As Double, i As Integer
  Dim d1 As Double, d2 As Double
  nu1 = 0
  nu2 = 0
  For i = 1 To nAvg
    nu1 = nu1 + (nAvg - i + 1) * (timeAvg(i) - timeAvg(i - 1))
    nu2 = nu2 + (nAvg - i + 1) ^ 2 * (timeAvg(i) - timeAvg(i - 1))
  Next i
  nu1 = nu1 / nAvg
  nu2 = nu2 / nAvg ^ 2
  meanG = assetPrice * Exp(riskFree * nu1 + 0.5 * sigma ^ 2 * (nu2 - nu1))
  d1 = (Log(meanG / strike) + 0.5 * sigma ^ 2 * nu2) / (sigma * Sqr(nu2))
  d2 = d1 - sigma * Sqr(nu2)
  With Application.WorksheetFunction
    BsGeAsianCall = Exp(-riskFree * timeAvg(nAvg)) * (meanG * .NormSDist(d1) - strike * .NormSDist(d2))
  End With
End Function
```


Code 9.8 : VBA code of the CMcAsianCall( ) routine.

McDKOCall(S_0 , K , H , L, c , r , σ , N , T , n , f_0 , *error*)

check the knock-out condition for the current asset price
Call *DKOBoundary*(H , L , S_0 , *crossflag*)
If(*crossflag*) then
  $f_0 = 0$, *error* = 0
  Exit subroutine
Endif

define the size of time interval
$\Delta t = T / N$

zeroize the sample sum and squared sum
sum = 0 , *sum*2 = 0

For($L_s = 1$ to n){

initialize the asset price
$S_t = S_0$

# generate the asset price at each intermediate time and check the knock-out condition
For($i = 0$ to $N - 1$){ ε = StdNormNum( )

$S_t = S_t \, exp((r - ½\sigma^2) \Delta t + \sigma\sqrt{\Delta t}\, \varepsilon)$

Call *DKOBoundary*(H , L , S_t , *crossflag*)
If(*crossflag*) then
$t_h = i\Delta t$
exit
Endif }

# evaluate the option payoff function as in (9.38)
If(*crossflag*) then
  $f_T = c \, e^{r(T - t_h)}$
Else
  f_T = *CallPayoff*(K , S_t)
Endif

$PV = e^{-rT} f_T$

accumulate the sample sum and squared sum
$sum = sum + PV$
$sum2 = sum2 + PV^2$ }

evaluate the estimates of mean and variance
$m = sum / n$

$$s = \sqrt{\frac{1}{n-1} sum2 - \frac{n}{n-1} m^2}$$

return the estimation of option price and standard error

$f_0 = m$
$error = s / \sqrt{n}$

Code 9.9 : Pseudo code of the McDKOCall( ) routine.

```
Sub McDKOCall(assetPrice As Double, strike As Double, upperBarrier As Double, lowerBarrier As Double, rebate As Double,
                riskFree As Double, sigma As Double, nStep As Integer, maturity As Double, nsample As Long,
                ByRef optionPrice As Double, ByRef stdErr As Double)
  Dim sum As Double, sum2 As Double, gen As Double
  Dim mean As Double, sd As Double, Ls As Long
  Dim St As Double, fT As Double, pV As Double, i As Integer
  Dim crossFlag As Boolean
  Dim timeHit As Double
  Dim dtime As Double: dtime = maturity / nStep

  Call DKOBoundary(upperBarrier, lowerBarrier, assetPrice, crossFlag)
  If (crossFlag) Then
    optionPrice = 0
    stdErr = 0
    Exit Sub
  End If

  sum = 0
  sum2 = 0
  For Ls = 1 To nsample
    St = assetPrice
    For i = 0 To nStep - 1
      gen = StdNormNum()
      St = St * Exp((riskFree - 0.5 * sigma ^ 2) * dtime + sigma * Sqr(dtime) * gen)
      Call DKOBoundary(upperBarrier, lowerBarrier, St, crossFlag)
      If (crossFlag) Then
        timeHit = i * dtime
        Exit For
      End If
    Next i
    If (crossFlag) Then
      fT = rebate * Exp(riskFree * (maturity - timeHit))
    Else
      fT = CallPayoff(strike, St)
    End If
    pV = Exp(-riskFree * maturity) * fT
    sum = sum + pV
    sum2 = sum2 + pV * pV
  Next Ls
  mean = sum / nsample
  sd = Sqr(sum2 / (nsample - 1) - (nsample / (nsample - 1)) * mean ^ 2)
  optionPrice = mean
  stdErr = sd / Sqr(nsample)
End Sub
```

```
Sub DKOBoundary(upperBarrier As Double, lowerBarrier As Double, assetPrice As Double, ByRef crossFlag As Boolean)
  If (assetPrice >= upperBarrier Or assetPrice <= lowerBarrier) Then
    crossFlag = True
  Else
    crossFlag = False
  End If
End Sub
```

Code 9.10 : VBA codes of the McDKOCall( ) and DKOBoundary( ) routines.

FvAmericanPut(x , K , r , σ , N , T , i , $S_c(\, i+1 : N\,)$, n , F , *error*)

define the size of time interval and the reference forward time
$\Delta t = T / N$, $t_i = i\Delta t$

zeroize the sample sum and squared sum
$sum = 0$, $sum2 = 0$

For($L_s = 1$ to n){

initialize the asset price
$S_t = x$

# generate the asset price at each time step, check early exercising criteria, and update the latest time
For($j = i$ to $N - 1$){ ε = StdNormNum( )

$S_t = S_t \, exp\big(\, (r - \tfrac{1}{2}\sigma^2)\, \Delta t + \sigma\sqrt{\Delta t}\, \varepsilon \,\big)$

Call *APBoundary*(S_t , $S_c(\, j+1\,)$, *exerciseflag*)

$\tau_s = (j+1)\Delta t$

If(*exerciseflag*) exit j }

evaluate the discounted payoff in (9.43)
$PV = e^{-r(\,\tau_s - t_i\,)}\, PutPayoff(\, K\,,\, S_t\,)$

accumulate the sample sum and squared sum
$sum = sum + PV$
$sum2 = sum2 + PV^2$ }

evaluate the estimates of mean and variance
$m = sum / n$

$$s = \sqrt{\frac{1}{n-1}\, sum2 - \frac{n}{n-1}\, m^2}$$

return the estimation of option price and standard error
$F = m$
$error = s / \sqrt{n}$

Code 9.11 : Pseudo code of the FvAmericanPut( ) routine.

```
Sub FvAmericanPut(assetPrice As Double, strike As Double, riskFree As Double, sigma As Double, nStep As Integer, maturity As Double,
                  iptr As Integer, Scritical() As Double, nsample As Long, ByRef fairValue As Double, ByRef stdErr As Double)
  Dim sum As Double, sum2 As Double, gen As Double
  Dim mean As Double, sd As Double, Ls As Long
  Dim St As Double, pV As Double, j As Integer
  Dim exerciseFlag As Boolean
  Dim timeStop As Double
  Dim dtime As Double: dtime = maturity / nStep
  Dim timeRef As Double: timeRef = iptr * dtime
  sum = 0
  sum2 = 0
  For Ls = 1 To nsample
    St = assetPrice
    For j = iptr To nStep - 1
      gen = StdNormNum()
      St = St * Exp((riskFree - 0.5 * sigma ^ 2) * dtime + sigma * Sqr(dtime) * gen)
      timeStop = (j + 1) * dtime
      Call APBoundary(St, Scritical(j + 1), exerciseFlag)
      If (exerciseFlag) Then Exit For
    Next j
    pV = Exp(-riskFree * (timeStop - timeRef)) * PutPayoff(strike, St)
    sum = sum + pV
    sum2 = sum2 + pV * pV
  Next Ls
  mean = sum / nsample
  sd = Sqr(sum2 / (nsample - 1) - (nsample / (nsample - 1)) * mean ^ 2)
  fairValue = mean
  stdErr = sd / Sqr(nsample)
End Sub
```



```
Sub APBoundary(assetPrice As Double, criticalPrice As Double, exerciseFlag As Boolean)
  If (assetPrice <= criticalPrice) Then
    exerciseFlag = True
  Else
    exerciseFlag = False
  End If
End Sub
```



```
Function PutPayoff(strike As Double, assetPrice As Double) As Double
  PutPayoff = Max(strike - assetPrice, 0)
End Function
```

Code 9.12 : VBA code of the FvAmericanPut( ) routine.

McAmericanPut(S_0 , K , r , σ , N , T , n , f_0 , *error*)

define the configuration of the linear regression
$h = 100$, $\delta = 0.01$

define the critical price at option's maturity
$S_c(N) = K$

# generate the critical boundary
For($i = N - 1$ to 0, -1){

define the trial value for the root of (9.45)
$x_{root} = S_c(i + 1)$

NewTrialValue : inc = $\delta\ x_{root} / h$

# generate plotting points for (9.45) in the neighborhood of the trial value
For($k = 1$ to h){ $X_{fit}(k) = x_{root} - k\ inc$

Call *FvAmericanPut*($X_{fit}(k)$, K , r , σ , N , T , i , $S_c(i + 1 : N)$, n , F , *error*)
$Y_{fit}(k) = F - PutPayoff(K , X_{fit}(k))$ }

perform a linear regression for (9.45) and obtain an approximation for its root
Call Regn($X_{fit}(1 : h)$, $Y_{fit}(1 : h)$, h , β , α)
$x_{root} = - \alpha / \beta$

check the validity of the approximated root and, if necessary, use it as a new trial value.
Call *FvAmericanPut*(x_{root} , K , r , σ , N , T , i , $S_c(i + 1 : N)$, n , F , *error*)
If($| F - PutPayoff(K , x_{root}) | > error$) Then GoTo NewTrialValue

assign the valid root of (9.45) as the critical price

$S_c(i) = x_{root}$ }

knowing the critical boundary, evaluate the option price and standard error according to (9.42)
Call *FvAmericanPut*(S_0 , K , r , σ , N , T , i , $S_c(0 : N)$, n , F , *error*)
$f_0 = max(F , PutPayoff(K , S_0))$

Code 9.13 : Pseudo code of the McAmericanPut( ) routine.

```
Sub McAmericanPut(assetPrice As Double, strike As Double, riskFree As Double, sigma As Double, nStep As Integer, maturity As Double,
                  nsample As Long, ByRef optionPrice As Double, ByRef stdErr As Double)
  Dim Scritical() As Double: ReDim Scritical(0 To nStep)
  Dim iptr As Integer, k As Integer
  Dim nFit As Integer: nFit = 100
  Dim delta As Double: delta = 0.01
  Dim inc As Double
  Dim xroot As Double, fairValue As Double
  Dim xFit() As Double: ReDim xFit(1 To nFit)
  Dim yFit() As Double: ReDim yFit(1 To nFit)
  Dim slope As Double, intercept As Double
  Scritical(nStep) = strike
  For iptr = nStep - 1 To 0 Step -1
    xroot = Scritical(iptr + 1)
NewTrialValue: inc = delta * xroot / nFit
    For k = 1 To nFit
      xFit(k) = xroot - k * inc
      Call FvAmericanPut(xFit(k), strike, riskFree, sigma, nStep, maturity, iptr, Scritical, nsample, fairValue, stdErr)
      yFit(k) = fairValue - PutPayoff(strike, xFit(k))
    Next k
    Call Regn(xFit, yFit, nFit, slope, intercept)
    xroot = -intercept / slope
    Call FvAmericanPut(xroot, strike, riskFree, sigma, nStep, maturity, iptr, Scritical, nsample, fairValue, stdErr)
    If (Abs(fairValue - PutPayoff(strike, xroot)) > stdErr) Then GoTo NewTrialValue
    Scritical(iptr) = xroot

    Range("A1").Offset(nStep - 1 - iptr, 0) = iptr * (maturity / nStep)
    Range("B1").Offset(nStep - 1 - iptr, 0) = xroot
    Application.StatusBar = "Done simulation i : " & iptr

  Next iptr
  Call FvAmericanPut(assetPrice, strike, riskFree, sigma, nStep, maturity, 0, Scritical, nsample, fairValue, stdErr)
  optionPrice = Max(fairValue, PutPayoff(strike, assetPrice))
End Sub
```

```
Sub Regn(xFit() As Double, yFit() As Double, nFit As Integer, ByRef slope As Double, ByRef intercept As Double)
  Dim avgX As Double, avgY As Double, avgX2 As Double, avgY2 As Double, avgXY As Double
  Dim i As Integer
  avgX = 0
  avgY = 0
  avgX2 = 0
  avgY2 = 0
  avgXY = 0
  For i = 1 To nFit
    avgX = avgX + xFit(i)
    avgY = avgY + yFit(i)
    avgX2 = avgX2 + xFit(i) ^ 2
    avgY2 = avgY2 + yFit(i) ^ 2
    avgXY = avgXY + xFit(i) * yFit(i)
  Next i
  avgX = avgX / nFit
  avgY = avgY / nFit
  avgX2 = avgX2 / nFit
  avgY2 = avgY2 / nFit
  avgXY = avgXY / nFit
  slope = (avgXY - avgX * avgY) / (avgX2 - avgX ^ 2)
  intercept = (avgX2 * avgY - avgX * avgXY) / (avgX2 - avgX ^ 2)
End Sub
```

Code 9.14 : VBA code of the McAmericanPut( ) routine.