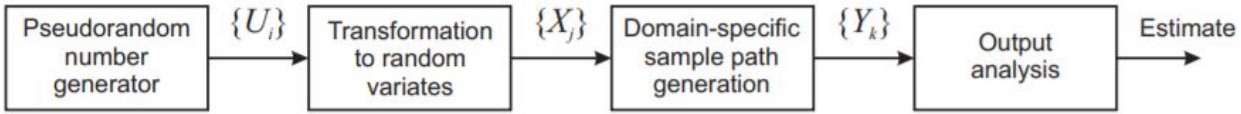
Monte Carlo Methods



Justification: SLLN

(Strong Law of Large Numbers) *Let* $(\xi^{(\ell)}, \ell \geq 1)$ *be a sequence of independent and identically distributed random variables with values in* $\mathbb{R}^d$. *Assume that*

$$\mathbb{E}\left|\xi^{(1)}\right| < \infty.$$

For $N \geq 1$, *denote the empirical mean of* $(\xi^{(1)}, \ldots, \xi^{(N)})$ *by*

$$\hat{S}_N := \frac{1}{N} \sum_{\ell=1}^{N} \xi^{(\ell)}.$$

Then, the Strong Law of Large Numbers holds true:

$$\lim_{N \to \infty} \hat{S}_N = \mathbb{E}\left(\xi^{(1)}\right), \quad \mathbb{P}\text{-a.s.}$$

The sequence of empirical means $(\hat{S}_N(\omega), N \geq 1)$ converges almost surely to $\mathbb{E}(\xi^{(1)})$ in ω.

Note: A property is said to hold almost surely (a.s.) if it holds except on an event of probability zero.

Justification: SLLN

Assume that there exists a function f and a family $(X^{(1)}, \ldots, X^{(N)})$ of independent and identically distributed random variables, which are easy to simulate and satisfy

$$\mathbb{E} f\left(X^{(1)}\right) = \gamma.$$

Then, except on an event of probability zero, γ can be approximated as follows.

Algorithm (Monte Carlo method) Draw a sample $(X^{(1)}(\omega), \ldots, X^{(N)}(\omega))$, and approximate γ by the empirical mean:

$$\gamma \simeq \hat{S}_N(\omega) := \frac{1}{N} \sum_{\ell=1}^{N} f\left(X^{(\ell)}(\omega)\right).$$

This is a "good" approximation as soon as N is chosen "large enough".

Benefits of Monte Carlo over Deterministic Methods

- Monte Carlo methods allow to compute solutions whose gradient is locally very large. Deterministic methods require thin grids in the areas where the gradient of the solution is large
- Monte Carlo methods are simpler and faster to code than deterministic methods, and the computer programs for stochastic numerical methods are easier to modify and adapt.

Application to Options

The price of a European style option is the expected value, under the risk-neutral measure Q, of the discounted payoff of the option: $f = e^{-rT}\mathsf{E}^{\mathbb{Q}}[f_T]$
where f_T is the payoff at the maturity date T.

$$f_T = \max\{0, S(0)e^{(r-\sigma^2/2)T+\sigma\sqrt{T}\epsilon} - K\}$$

```
function Price = BlsMC1(S0,K,r,T,sigma,NRepl)
nuT = (r - 0.5*sigma^2)*T;
siT = sigma * sqrt(T);
DiscPayoff = exp(-r*T)*max(0, S0*exp(nuT+siT*randn(NRepl,1))-K);
Price = mean(DiscPayoff);
```

```
>> S0=50;
>> K=60;
>> r=0.05;
>> T=1;
>> sigma=0.2;
>> BlsMC1(S0,K,r,T,sigma,1000)
```

And with the expected value of squared error below, increasing the number of replications improves the estimate.

$$\mathsf{E}[(\bar{X}(n) - \mu)^2] = \mathsf{Var}[\bar{X}(n)] = \frac{\sigma^2}{n}$$

Note: When you have a hammer, not everything is a nail

$$C = S e^{-qT} N(d_1) - K e^{-rT} N(d_2)$$

q = dividend rate

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r - q + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}} \qquad d_2 = \frac{\ln\left(\frac{S}{K}\right) + \left(r - q - \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}$$

Set $x = \sigma$

$$f(x) = S e^{-qT} N(d_1(x)) - K e^{-rT} N(d_2(x)) - C$$

$$f'(x) = \frac{1}{\sqrt{2\pi}} S e^{-qT} \sqrt{T} \exp\left(-\frac{d_1(x)^2}{2}\right)$$

w/ $x_0 = 25\%$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Ex: $C = 7$, $K = 20$, $S = 25$, $T = 1$, $r = .05$

Find σ:

Step	sigma guess	BS	BSDeriv	(BS-C)/BSDeriv	x(k)-previous
1	0.25	6.35313	3.36060235	-0.192486921	0.442486921
2	0.442486921	7.53695	4.961921197	0.108214549	0.334272372
3	0.334272372	6.81856	4.344779508	-0.041761029	0.376033401
4	0.376033401	7.08441	4.642727619	0.018181799	0.357851601
5	0.357851601	6.96652	4.524382121	-0.007399455	0.365251056
6	0.365251056	7.01413	4.574490943	0.003089223	0.362161833
7	0.362161833	6.99419	4.553907575	-0.001275546	0.363437379
8	0.363437379	7.00241	4.56246413	0.000529052	0.362908327
9	0.362908327	6.999	4.558925072	-0.00021902	0.363127348
10	0.363127348	7.00041	4.560391892	9.07419E-05	0.363036606
11	0.363036606	6.99983	4.559784469	-3.7583E-05	0.363074189
12	0.363074189	7.00007	4.560036098	1.5568E-05	0.363058621
13	0.363058621	6.99997	4.559931874	-6.44837E-06	0.363065069
14	0.363065069	7.00001	4.559975046	2.67102E-06	0.363062398
15	0.363062398	6.99999	4.559957164	-1.10637E-06	0.363063505
16	0.363063505	7	4.559964571	4.58274E-07	0.363063046
17	0.363063046	7	4.559961503	-1.89823E-07	0.363063236

```
Function BS(C, S, K, T, r, x)
Set wf = Application.WorksheetFunction

BS = S * wf.Norm_S_Dist(BS1(S, K, r, x, T), True) - K * Exp(-r * T) * _
    wf.Norm_S_Dist(BS2(S, K, r, x, T), True)

End Function

Function BS1(S, K, r, x, T)
BS1 = (Log(S / K) + (r + (x ^ 2) / 2) * T) / x * Sqr(T)
End Function
Function BS2(S, K, r, x, T)
BS2 = (Log(S / K) + (r - (x ^ 2) / 2) * T) / x * Sqr(T)
End Function

Function BSDeriv(S, K, T, r, x)
Set wf = Application.WorksheetFunction
BSDeriv = (1 / (2 * Sqr(wf.Pi()))) * S * Sqr(T) * _
          Exp(-(BS1(S, K, r, x, T) ^ 2) / 2)
End Function
```

You could randomly sample with Monte Carlo, or you could just use Newton's algorithm to get a solution in quadratic time.

Step	sigma guess	BS	BSDeriv	(BS-C)/BSDeriv	x(k)-previous
1	0.25	=@BS(7,25,20,1,0.05,B22)	=@BSDeriv(25,20,1,0.05,B22)	=(C22-$B$19)/D22	=B22-E22
2	=F22	=@BS(7,25,20,1,0.05,B23)	=@BSDeriv(25,20,1,0.05,B23)	=(C23-$B$19)/D23	=B23-E23
3	=F23	=@BS(7,25,20,1,0.05,B24)	=@BSDeriv(25,20,1,0.05,B24)	=(C24-$B$19)/D24	=B24-E24
4	=F24	=@BS(7,25,20,1,0.05,B25)	=@BSDeriv(25,20,1,0.05,B25)	=(C25-$B$19)/D25	=B25-E25

Producing the Numbers

A “Good” PRNG

• **Properties of PRNG :** A PRNG is called *good*, if it satisfies the following properties –

1. **Uniformity :** This property implies that, if we divide the set of numbers generated by the PRNG into K equal subintervals, then expected number of samples (e_i) in each subinterval $i, (1 \leq i \leq K)$ is equal; that is, $\forall i, e_i = \frac{N}{K}$, where N is the range of the numbers. This ensures that, the generated numbers are equally probable in every part of the number space.

2. **Independence :** The generated numbers are to be independent of each other; that is, there should not be any serial correlation between numbers generated in succession. So, any subsequence of numbers have no correlation with any other subsequences. This means, given any length of previous numbers, one can not predict the next number in the sequence by observing the given numbers.

3. **Large Period :** Every PRNG has a period after which the sequence is repeated. A PRNG is considered good if it has a very large period. Otherwise, if one can exhaust the period of a PRNG, the sequence of numbers become completely predictable.

4. **Reproducibility :** One of the prominent reason of developing PRNG is its property of reproducibility. This ensures that given the same seed s_0, the same sequence of numbers is to be generated. This is very useful in simulation, debugging and testing purposes.

5. **Consistency :** The above properties of the PRNG are to be independent of the seed. That is, all these properties are to be maintained for every seed value.

6. **Disjoint subsequences :** There is to be little or no correlation between subsequences generated by different seeds.

7. **Permutations :** Every permutation of a number generated by a PRNG is expected to be equally likely. Otherwise, the numbers can be biased and may help to predict successive numbers.

8. **Portability :** A PRNG is to be portable; that is, the same algorithm can work on every system. Given the same seed, different machines with varied configuration are to give the same output sequence.

9. **Efficiency :** The PRNG is to be very fast; which means, generation of a random number takes insignificant time. Moreover, a PRNG should not use much storage or computational overhead. This is to make certain that, the use of PRNG in an application is not a hindrance to its efficiency.

10. **Coverage :** This implies whether the PRNG covers the output space for any seed. Many PRNG has less coverage. In case, the PRNG has more than one cycle, then it may happen that, although it covers the whole output space, but only a part of it is covered by a particular seed.

11. **Spectral Characteristics :** A good PRNG does not generate numbers of one frequency higher than any other. If we plot the consecutive numbers, there are not to be any pattern visible for any length of the sequence.

12. **Cryptographically Secure :** To be used in cryptographic applications, the generated numbers should be cryptographically secure. This is desirable property often missing in most of the algorithmic PRNGs.

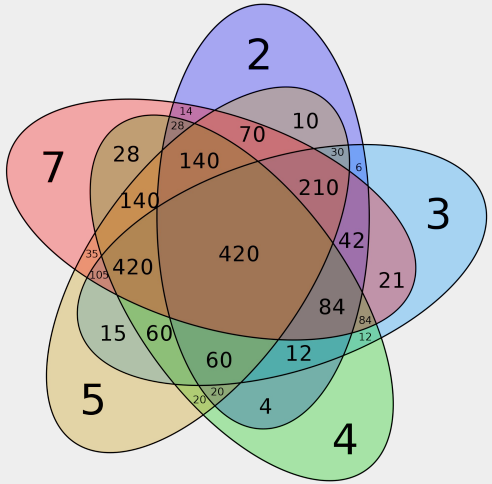
Linear Congruential Generators

$$x_{n+1} = (ax_n + c) \pmod{m}, \quad n \geq 0$$

1. c is relatively prime to m;
2. if m is multiple of 4, $a - 1$ is also multiple of 4;
3. for every prime divisor p of m, $a - 1$ is multiple of p.

Note: to get a number in [0,1] let $u_k = x_n/m$

```
function [USeq, ZSeq] = LCG(a,c,m,seed,N)
ZSeq = zeros(N,1);
USeq = zeros(N,1);
for i=1:N
    seed = mod(a*seed+c, m);
    ZSeq(i) = seed;
    USeq(i) = seed/m;
end
```



Examples

1

$x_i = 13x_{i-1} + 5 \pmod{31}$

x	u	y=x^2 [0,1]	average value	actual value
13	0.41935484	0.1758585	0.300763094	0.328
19	0.61290323	0.3756504		
4	0.12903226	0.0166493		
26	0.83870968	0.7034339		
2	0.06451613	0.0041623		

2

$4(2x-1)^4 + 8(2y-1)^8 < 1 + 2(2y-1)^3(3x-2)^2$

Find the area of the set of points (x,y) that satisfy the above.

Let $(x,y) = (u_i, u_{i+1})$ and check 10,000 random points. See VBA code.

```
Sub areaCheck()
Count = 0
Total = 0
k = 1
Dim modNum As Double
modNum = 214748#
a = 6553
'matlab randu uses a = 2^16+3 = 65539, modNum =2^31

For num = 1 To 10000
'Set up the random numbers
    k = a * k Mod modNum
    i = k / modNum
    k = a * k Mod modNum
    j = k / modNum

'Set up the equations
    x = 4 * ((2 * i - 1) ^ 4) + 8 * (2 * j - 1) ^ 8
    y = 1 + 2 * ((2 * j - 1) ^ 3) * (3 * i - 2) ^ 2

'Check conditions
    If x < y Then
        Count = Count + 1
    End If
    Total = Total + 1
Next num

Debug.Print Count / Total
End Sub
```

Issues with LCGs

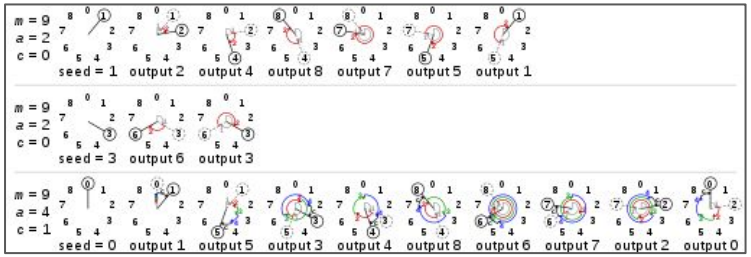
1

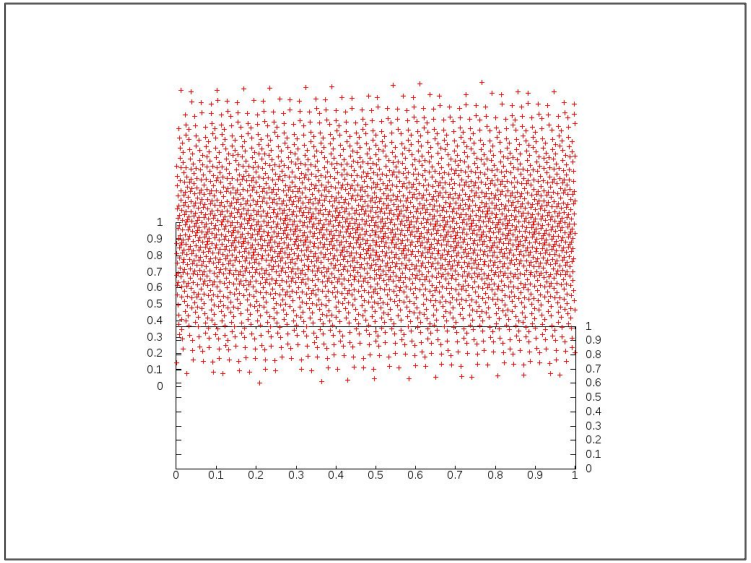
To avoid overflow, a straightforward implementation of a linear congruential generator in integer variables must be restricted to an unacceptably small modulus - e.g., $2^{15} - 1$.
One of many possible approaches is to take advantage of number theoretic properties and let m = aq +r and then:

$$ax_i \bmod m = a(x_i \bmod q) - \left\lfloor \frac{x_i}{q} \right\rfloor r + \left(\left\lfloor \frac{x_i}{q} \right\rfloor - \left\lfloor \frac{ax_i}{m} \right\rfloor \right) m$$

2

overlapping d-tuples formed from consecutive outputs of a linear congruential generator with modulus m lie on at most $(d!m)^{1/d}$ hyperplanes in the d-dimensional unit cube. For $m = 2^{31} - 1$, this is approximately 108 with $d = 3$ and drops below 39 at $d = 10$. Thus, particularly in high dimensions, the lattice structure of even the best possible linear congruential generators distinguishes them from genuinely random numbers.





Transforms

Uniform Transform to Poisson Distribution

- For $U \sim U(0,1)$
- Let $X = F^{-1}(U)$
- $P\{ X \leq x \} = P \{ F^{-1}(U) \leq x \} = P \{ U \leq F(x) \} = F(x)$
- A typical choice would be: $X \sim \exp(\mu)$ s.t. $F(x) = 1 - e^{-\mu x}$
- This yields: $x = -(1/\mu)\ln(1-U) = - \ln(U)/\mu$

Uniform Transform to Normal Distribution

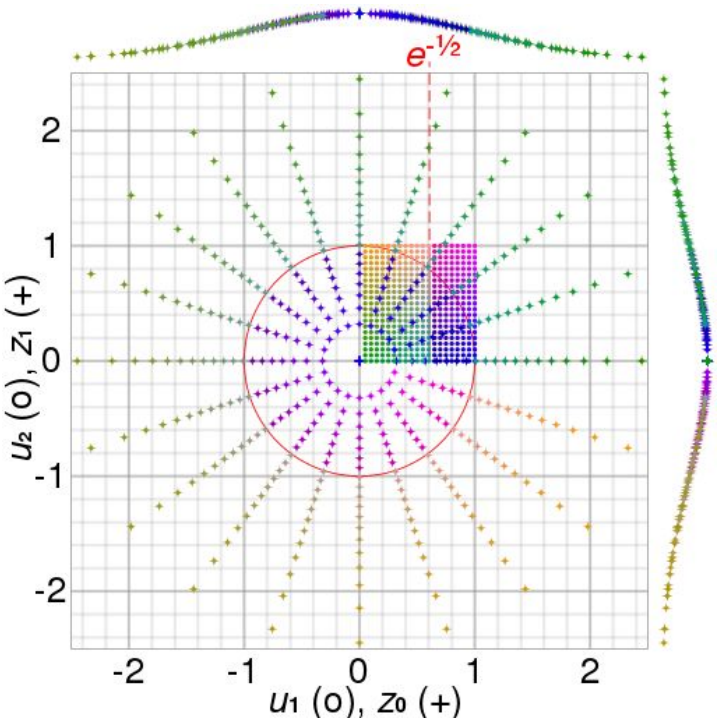
- If X ~ N(0,1), then μ + σX ~ N(μ,σ^2)
- We can use the *Box-Muller method* to convert from U to N. We use the polar form of Cartesian coordinates (X,Y) s.t.: $d = R^2 = X^2 + Y^2 \qquad \theta = \tan^{-1} Y/X$
- The joint density of X and Y is: $f(x,y) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \frac{1}{\sqrt{2\pi}} e^{-y^2/2} = \frac{1}{2\pi} e^{-(x^2+y^2)/2} = \frac{1}{2\pi} e^{-d/2}$
- And to express in (d,θ) terms, take the Jacobian to get: $f(d,\theta) = \frac{1}{2} \frac{1}{2\pi} e^{-d/2}$

Box-Muller Method

1. Generate two independent uniform variates $U_1, U_2 \sim U(0,1)$.
2. Set $V_1 = 2U_1 - 1$, $V_2 = 2U_2 - 1$, $S = V_1^2 + V_2^2$.
3. If $S > 1$, return to step 1; otherwise, return the independent standard normal variates:

$$X = \sqrt{\frac{-2 \ln S}{S}} V_1, \qquad Y = \sqrt{\frac{-2 \ln S}{S}} V_2.$$

```
Function BoxMuller()
Randomize
Do
        x = 2 * Rnd() - 1
        y = 2 * Rnd() - 1
        dist = x * x + y * y
Loop Until dist < 1
BoxMuller = x * Sqr(-2 * Log(dist) / dist)
End Function
```



Applying the Inverse Transform Method: Φ^{-1}

Because of the symmetry of the normal distribution,

$$\Phi^{-1}(1-u) = -\Phi^{-1}(u), \quad 0 < u < 1;$$

it therefore suffices to approximate Φ^{-1} on the interval $[0.5, 1)$ (or the interval $(0, 0.5]$) and then to use the symmetry property to extend the approximation to the rest of the unit interval. Beasley and Springer [43] provide a rational approximation

$$\Phi^{-1}(u) \approx \frac{\sum_{n=0}^{3} a_n (u - \frac{1}{2})^{2n+1}}{1 + \sum_{n=0}^{3} b_n (u - \frac{1}{2})^{2n}}, \tag{2.27}$$

for $0.5 \le u \le 0.92$, with constants a_n, b_n given in Figure 2.12; for $u > 0.92$ they use a rational function of $\sqrt{\log(1-u)}$. Moro [271] reports greater accuracy in the tails by replacing the second part of the Beasley-Springer approximation with a Chebyshev approximation

$$\Phi^{-1}(u) \approx g(u) = \sum_{n=0}^{8} c_n [\log(-\log(1-u))]^n, \quad 0.92 \le u < 1, \tag{2.28}$$

with constants c_n again given in Figure 2.12. Using the symmetry rule, this gives

$$\Phi^{-1}(u) \approx -g(1-u) \quad 0 < u \le .08.$$

With this modification, Moro [271] finds a maximum absolute error of 3×10^{-9} out to seven standard deviations (i.e., over the range $\Phi(-7) \le u \le \Phi(7)$). The combined algorithm from Moro [271] is given in Figure 2.13.

$a_0 =$	2.50662823884	$b_0 =$	-8.47351093090
$a_1 =$	-18.61500062529	$b_1 =$	23.08336743743
$a_2 =$	41.39119773534	$b_2 =$	-21.06224101826
$a_3 =$	-25.44106049637	$b_3 =$	3.13082909833

$c_0 = 0.3374754822726147$	$c_5 = 0.0003951896511919$
$c_1 = 0.9761690190917186$	$c_6 = 0.0000321767881768$
$c_2 = 0.1607979714918209$	$c_7 = 0.0000002888167364$
$c_3 = 0.0276438810333863$	$c_8 = 0.0000003960315187$
$c_4 = 0.0038405729373609$	

Fig. 2.12. Constants for approximations to inverse normal.

Input: u between 0 and 1
Output: x, approximation to $\Phi^{-1}(u)$.
$y \leftarrow u - 0.5$
if $|y| < 0.42$
  $r \leftarrow y * y$
  $x \leftarrow y * (((a_3 * r + a_2) * r + a_1) * r + a_0)/$
    $((((b_3 * r + b_2) * r + b_1) * r + b_0) * r + 1)$
else
  $r \leftarrow u$;
  if $(y > 0)$ $r \leftarrow 1 - u$
  $r \leftarrow \log(-\log(r))$
  $x \leftarrow c_0 + r * (c_1 + r * (c_2 + r * (c_3 + r * (c_4 +$
    $r * (c_5 + r * (c_6 + r * (c_7 + r * c_8)))))))$
  if $(y < 0)$ $x \leftarrow -x$
return x

From: Monte Carlo Methods in Financial Engineering - Glasserman

Or, one can use the Newton Method

The problem of computing $\Phi^{-1}(u)$ can be posed as one of finding the root x of the equation $\Phi(x) = u$ and in principle addressed through any general root-finding algorithm. Newton's method, for example, produces the iterates

$$x_{n+1} = x_n - \frac{\Phi(x_n) - u}{\phi(x_n)},$$

or, more explicitly,

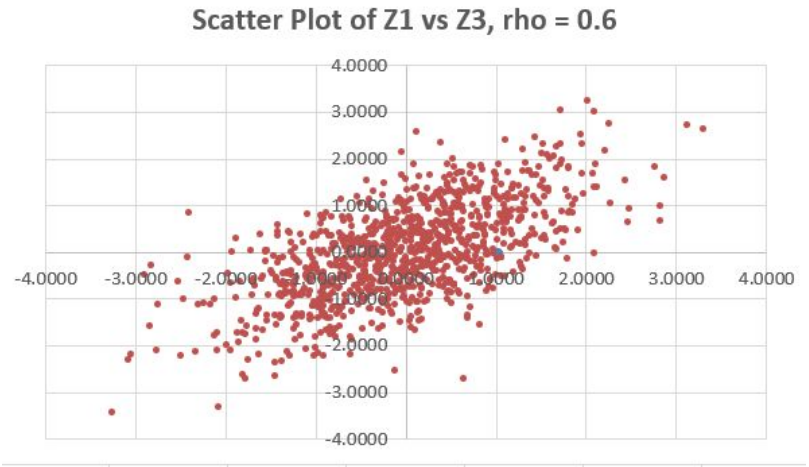
$$x_{n+1} = x_n + (u - \Phi(x_n)) \exp(-0.5x_n \cdot x_n + c), \quad c \equiv \log(\sqrt{2\pi}).$$

Marsaglia, Zaman, and Marsaglia [251] recommend the starting point

$$x_0 = \pm\sqrt{|-1.6\log(1.0004 - (1-2u)^2)|},$$

Multivariate Normals: Uniformity and Correlation

1,000 CORRELATED STANDARD NORMAL DEVIATES				
ρ	0.6			
Mean	-0.0173	0.0134	0.0003	<-- =AVERAGE(D12:D1011)
Sigma	0.9955	0.9633	0.9999	<-- =STDEV.S(D12:D1011)
Skewness	-0.0216	-0.0496	-0.0017	<-- =SKEW(D12:D1011)
Kurtosis	0.1695	-0.0744	0.1126	<-- =KURT(D12:D1011)
Count	1000	1000	1000	<-- =COUNT(D12:D1011)
Corr(z_1,z_3)	0.6386	<-- =CORREL(B12:B1011,D12:D1011)		
	Z_1	Z_2	Z_3	
=NORM.S.INV(RAND()) -->	1.0285	-0.9692	-0.1582	<-- =$B$2*B12+SQRT(1-$B$2^2)*C12
=NORM.S.INV(RAND()) -->	-1.1663	1.3459	0.3769	<-- =$B$2*B13+SQRT(1-$B$2^2)*C13
=NORM.S.INV(RAND()) -->	1.5847	0.4848	1.3387	<-- =$B$2*B14+SQRT(1-$B$2^2)*C14
	-1.0764	0.0596	-0.5982	
	-1.3718	-0.1565	-0.9482	
	-0.8330	-0.5360	-0.9286	



- Create two normal z_1 and z_2
- Let $z_3 = \rho z_1 + z_2\sqrt{1-\rho^2}$
- z_3 and z_1 are correlated with correlation ρ

A representation of Σ as AA^T with A lower triangular is a Cholesky factorization of Σ. If Σ is positive definite, it has a Cholesky factorization and the matrix A is unique up to changes in sign.

Var-Cov to Cholesky

2X2 Case

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \sigma_1\sigma_2\rho \\ \sigma_1\sigma_2\rho & \sigma_2^2 \end{pmatrix} \Rightarrow A = \begin{pmatrix} \sigma_1 & 0 \\ \rho\sigma_2 & \sqrt{1-\rho^2}\sigma_2 \end{pmatrix}$$

NXN Case

$$A_{ij} = \left(\Sigma_{ij} - \sum_{k=1}^{j-1} A_{ik}A_{jk} \right) / A_{jj}, \quad j < i,$$

$$A_{ii} = \sqrt{\Sigma_{ii} - \sum_{k=1}^{i-1} A_{ik}^2}.$$

General Formula

Input: Symmetric positive definite matrix $d \times d$ matrix Σ
Output: Lower triangular A with $AA^\top = \Sigma$

$A \leftarrow 0$ ($d \times d$ zero matrix)
for $j = 1, \ldots, d$
  for $i = j, \ldots, d$
    $v_i \leftarrow \Sigma_{ij}$
    for $k = 1, \ldots, j-1$
      $v_i \leftarrow v_i - A_{jk}A_{ik}$
    $A_{ij} \leftarrow v_i / \sqrt{v_j}$
return A

Cholesky Decomposition

The Cholesky decomposition of a Hermitian positive-definite matrix is: $\Sigma = AA^T$

	A	B	C	D	E
1	**Variance-covariance matrix, S**				
2	0.400	0.030	0.020	0.000	
3	0.030	0.200	0.000	-0.060	
4	0.020	0.000	0.300	0.030	
5	0.000	-0.060	0.030	0.100	
6					
7	**Cholesky decomposition, L**				
8	0.632	0.000	0.000	0.000	<-- {=cholesky(A2:D5)}
9	0.047	0.445	0.000	0.000	
10	0.032	-0.003	0.547	0.000	
11	0.000	-0.135	0.054	0.281	
12					
13	**Check: Multiply above matrix by its transpose**				
14	0.400	0.030	0.020	0.000	<-- {=MMULT(A8:D11,TRANSPOSE(A8:D11))}
15	0.030	0.200	0.000	-0.060	
16	0.020	0.000	0.300	0.030	
17	0.000	-0.060	0.030	0.100	

The Cholesky Decomposition allows one to create random numbers with the same distribution as the var-cov matrix.

BASIC MULTIVARIATE NORMAL SIMULATION				
Variance-covariance matrix				
0.40	0.03	0.02	0.01	
0.03	0.30	0.00	-0.06	
0.02	0.00	0.20	0.03	
0.01	-0.06	0.03	0.10	
Cholesky decomposition				
0.6325	0.0000	0.0000	0.0000	<-- {=cholesky(varcov)}
0.0474	0.5457	0.0000	0.0000	
0.0316	-0.0027	0.4461	0.0000	
0.0158	-0.1113	0.0654	0.2882	
Generating four random normals				
-0.9050	<-- =NORMSINV(RAND())			
0.4872	<-- =NORMSINV(RAND())			
0.2696	<-- =NORMSINV(RAND())			
-0.1155	<-- =NORMSINV(RAND())			
Generating multinomial normal output				
-0.5724	<-- {=MMULT($A$9:$D$12,A15:A18)}			
0.2229				
0.0903				
-0.0842				

Errors In Monte Carlo

- Sampling Error
  - Sampling error is due to the random nature of Monte Carlo methods. It can be mitigated using variance reduction strategies.
- Discretization Error
  - See next slide

To understand what discretization error is, let us consider how we can discretize an Ito stochastic differential equation:

$$dS_t = a(S_t, t)\, dt + b(S_t, t)\, dW_t.$$

The simplest discretization approach, known as Euler scheme, is:

$$\delta S_t = S_{t+\delta t} - S_t = a(S_t, t)\delta t + b(S_t, t)\sqrt{\delta t}\, \epsilon,$$

where δt is the discretization step and $\epsilon \sim \mathsf{N}(0, 1)$.

Convergence is a critical concept in stochastic differential equations, but we may guess that, by sampling realizations of the random variable ϵ from the standard normal distribution, we should be able to simulate a discrete-time stochastic process which is well related to the solution of the continuous-time equation.

We should realize that the discretization error may even change the probability distributions characterizing the solution.

For instance, consider the geometric Brownian motion model:

$$dS_t = \mu S_t dt + \sigma S_t dW_t.$$

The Euler scheme yields

$$S_{t+\delta t} = (1 + \mu\, \delta t) S_t + \sigma S_t \sqrt{\delta t}\, \epsilon.$$

This is very easy to grasp and to implement, but the marginal distribution of each value $S_i = S(i\, \delta t)$ is normal, rather than lognormal.

From slides that Paolo Brandimarte sent me

Technical Parameters

For any measurable non-negative function f from $\mathbb{R}^d$ to $\mathbb{R}$,

$$\mathbb{E}f(X) := \int_\Omega f\big(X(\omega)\big)\mathbb{P}(d\omega) = \int_{\mathbb{R}^d} f(x)\mathbb{P}^X(dx).$$

In practice, it will often be seen that the exact Monte Carlo method cannot be implemented, due to the fact that a probabilistic representation $\gamma = \mathbb{E}f(X)$, can only be found for a random variable X which is difficult (or impossible) to simulate. It is then necessary to find some other easily simulable random variable $\bar{X}$ which satisfies $\mathbb{E}f(\bar{X}) \simeq \mathbb{E}f(X)$.

The Central Limit Theorem and its non-asymptotic versions show that, the number N of simulations being fixed, a Monte Carlo approximation is all the more effective when the variance σ^2 of $\xi^{(1)}$ is small. Equivalently, the desired accuracy being fixed, the number of simulations which is necessary to achieve this accuracy is all the weaker when the σ^2 is small.

Definition 3.2 Let $(\bar{X}^n, n \in \mathbb{N})$ be a sequence of $\mathbb{R}^d$-valued random variables. It is said to converge in law to a random variable X, or to its law $\mathbb{P}^X$, if the laws $(\mathbb{P}^{\bar{X}^n}, n \in \mathbb{N})$ converge weakly to $\mathbb{P}^X$, i.e., if for every bounded continuous function f from $\mathbb{R}^d$ to $\mathbb{R}$,

$$\lim_{n\to\infty} \int f(x)\mathbb{P}^{\bar{X}^n}(dx) = \int f(x)\mathbb{P}^X(dx).$$

In terms of the random variables themselves, this can be written as

$$\lim_{n\to\infty} \mathbb{E}f\big(\bar{X}^n\big) = \mathbb{E}f(X).$$

The approximation of $\gamma = \mathbb{E}f(X)$ by a Monte Carlo method often depends on two parameters:

- the index n of the term $\bar{X}^n$ actually chosen as an approximation, in the sequence $(\bar{X}^n)$ which converges in law to X,
- the number N of the simulated independent and identically distributed copies $\bar{X}^{n(1)}, \ldots, \bar{X}^{n(N)}$ of $\bar{X}^n$.

More precisely, the global approximation error can be decomposed as follows:

$$\gamma - \frac{1}{N}\sum_{\ell=1}^{N} f\big(\bar{X}^{n(\ell)}\big) = \underbrace{\mathbb{E}f(X) - \mathbb{E}f\big(\bar{X}^n\big)}_{\varepsilon_d(n)} + \underbrace{\mathbb{E}f\big(\bar{X}^n\big) - \frac{1}{N}\sum_{\ell=1}^{N} f\big(\bar{X}^{n(\ell)}\big)}_{\varepsilon_s(n,N)},$$

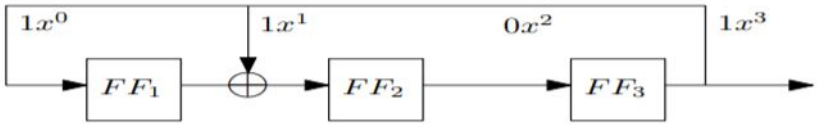
- where the term $\varepsilon_d(n)$ measures the approximation error on the exact probability distribution, due to convergence in law,
- and the term $\varepsilon_s(n, N)$ measures the statistical error resulting from the application of the Strong Law of Large Numbers.

From: Stochastic Simulation and Monte Carlo Methods, Graham, Springer, 2013

Another PRNG: Linear Feedback Shift Register

A LFSR is linear recurrence based in GF(2) using primitive characteristic polynomials that generate a maximal length sequence of period $2^k - 1$, where k is the degree of the polynomial.

LFSR are shift registers where the output of some bit positions are XOR-ed and feed as input to the register. This feedback connection ensures that the register cycles endlessly through repetitive sequences of values. The position of the XOR determines the characteristic polynomial of the LFSR, whereas the number of flip-flops (k) determines the degree of the polynomial.



A schematic diagram of 3-bit LFSR with characteristic polynomial $P(x) = 1+x+x^3$

LFSR

LFSRs can be implemented on hardware, are cost-efficient and are very fast. They are often used in cryptography for stream cipher encryption.

Bits (n)	Feedback polynomial	Period ($2^n - 1$)
2	$x^2 + x + 1$	3
3	$x^3 + x^2 + 1$	7
4	$x^4 + x^3 + 1$	15
5	$x^5 + x^3 + 1$	31
6	$x^6 + x^5 + 1$	63
7	$x^7 + x^6 + 1$	127
8	$x^8 + x^6 + x^5 + x^4 + 1$	255
9	$x^9 + x^5 + 1$	511
10	$x^{10} + x^7 + 1$	1,023
11	$x^{11} + x^9 + 1$	2,047
12	$x^{12} + x^{11} + x^{10} + x^4 + 1$	4,095
13	$x^{13} + x^{12} + x^{11} + x^8 + 1$	8,191
14	$x^{14} + x^{13} + x^{12} + x^2 + 1$	16,383
15	$x^{15} + x^{14} + 1$	32,767
16	$x^{16} + x^{15} + x^{13} + x^4 + 1$	65,535
17	$x^{17} + x^{14} + 1$	131,071
18	$x^{18} + x^{11} + 1$	262,143
19	$x^{19} + x^{18} + x^{17} + x^{14} + 1$	524,287
20	$x^{20} + x^{17} + 1$	1,048,575

The sequence

0100001001011001111100011011101010000100101100111 1

can be described by giving the initial values

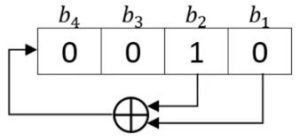
$$x_1 \equiv 0,\ x_2 \equiv 1,\ x_3 \equiv 0,\ x_4 \equiv 0,\ x_5 \equiv 0$$

and the linear recurrence relation

$$x_{n+5} \equiv x_n + x_{n+2} \pmod 2.$$

1	0	0	1	0
2	1	0	0	1
3	1	1	0	0
4	0	1	1	0
5	1	0	1	1
6	0	1	0	1
7	1	0	1	0
8	1	1	0	1
9	1	1	1	0
10	1	1	1	1

$$b_1 \leftarrow b_2'$$
$$b_2 \leftarrow b_3'$$
$$b_3 \leftarrow b_4'$$
$$b_4 \leftarrow b_1' + b_2'$$




Brownian Motion

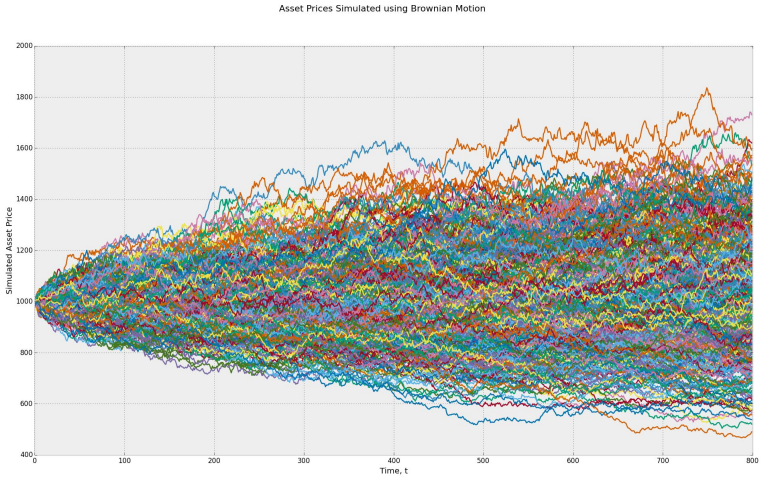
By a *standard* one-dimensional Brownian motion on $[0,T]$, we a mean a stochastic process $\{W(t), 0 \le t \le T\}$ with the following properties:

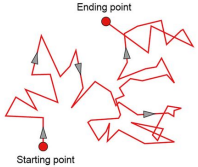
(i) $W(0) = 0$;

(ii) the mapping $t \mapsto W(t)$ is, with probability 1, a continuous function on $[0,T]$;

(iii) the increments $\{W(t_1) - W(t_0), W(t_2) - W(t_1), \ldots, W(t_k) - W(t_{k-1})\}$ are independent for any k and any $0 \le t_0 < t_1 < \cdots < t_k \le T$;

(iv) $W(t) - W(s) \sim N(0, t-s)$ for any $0 \le s < t \le T$.





For constants μ and $\sigma > 0$, we call a process $X(t)$ a Brownian motion with drift μ and diffusion coefficient σ^2 (abbreviated $X \sim \mathrm{BM}(\mu, \sigma^2)$) if

$$\frac{X(t) - \mu t}{\sigma}$$

is a standard Brownian motion. Thus, we may construct X from a standard Brownian motion W by setting

$$X(t) = \mu t + \sigma W(t).$$

It follows that $X(t) \sim N(\mu t, \sigma^2 t)$. Moreover, X solves the stochastic differential equation (SDE)

$$dX(t) = \mu\, dt + \sigma\, dW(t).$$

The assumption that $X(0) = 0$ is a natural normalization, but we may construct a Brownian motion with parameters μ and σ^2 and initial value x by simply adding x to each $X(t)$.

For deterministic but time-varying $\mu(t)$ and $\sigma(t) > 0$, we may define a Brownian motion with drift μ and diffusion coefficient σ^2 through the SDE

$$dX(t) = \mu(t)\, dt + \sigma(t)\, dW(t);$$

i.e., through

$$X(t) = X(0) + \int_0^t \mu(s)\, ds + \int_0^t \sigma(s)\, dW(s),$$

Suppose W is a standard Brownian motion and X satisfies

$$dX(t) = \mu\, dt + \sigma\, dW(t),$$

so that $X \sim \mathrm{BM}(\mu, \sigma^2)$. If we set $S(t) = S(0)\exp(X(t)) \equiv f(X(t))$, then an application of Itô's formula shows that

$$\begin{aligned} dS(t) &= f'(X(t))\, dX(t) + \tfrac{1}{2}\sigma^2 f''(X(t))\, dt \\ &= S(0)\exp(X(t))[\mu\, dt + \sigma\, dW(t)] + \tfrac{1}{2}\sigma^2 S(0)\exp(X(t))\, dt \\ &= S(t)(\mu + \tfrac{1}{2}\sigma^2)\, dt + S(t)\sigma\, dW(t). \end{aligned} \tag{3.17}$$

In contrast, a geometric Brownian motion process is often specified through an SDE of the form

$$\frac{dS(t)}{S(t)} = \mu\, dt + \sigma\, dW(t), \tag{3.18}$$

an expression suggesting a Brownian model of the "instantaneous returns" $dS(t)/S(t)$. Comparison of (3.17) and (3.18) indicates that the models are inconsistent and reveals an ambiguity in the role of "μ." In (3.17), μ is the drift of the Brownian motion we exponentiated to define $S(t)$ — the drift of $\log S(t)$. In (3.18), $S(t)$ has drift $\mu S(t)$ and (3.18) implies

$$d\log S(t) = (\mu - \tfrac{1}{2}\sigma^2)\, dt + \sigma\, dW(t), \tag{3.19}$$

as can be verified through Itô's formula or comparison with (3.17).

We will use the notation $S \sim \mathrm{GBM}(\mu, \sigma^2)$ to indicate that S is a process of the type in (3.18). We will refer to μ in (3.18) as the *drift parameter* though it is not the drift of either $S(t)$ or $\log S(t)$. We refer to σ in (3.18) as the *volatility parameter* of $S(t)$; the diffusion coefficient of $S(t)$ is $\sigma^2 S^2(t)$.

From (3.19) we see that if $S \sim \mathrm{GBM}(\mu, \sigma^2)$ and if S has initial value $S(0)$, then

$$S(t) = S(0)\exp\left([\mu - \tfrac{1}{2}\sigma^2]t + \sigma W(t)\right). \tag{3.20}$$

A bit more generally, if $u < t$ then

$$S(t) = S(u)\exp\left([\mu - \tfrac{1}{2}\sigma^2](t-u) + \sigma(W(t) - W(u))\right), \tag{3.21}$$

from which the claimed independence of the returns in (3.16) becomes evident. Moreover, since the increments of W are independent and normally distributed, this provides a simple recursive procedure for simulating values of S at $0 = t_0 < t_1 < \cdots < t_n$:

$$\begin{aligned} S(t_{i+1}) = S(t_i)\exp\left([\mu - \tfrac{1}{2}\sigma^2](t_{i+1} - t_i) + \sigma\sqrt{t_{i+1} - t_i}\, Z_{i+1}\right), \\ i = 0, 1, \ldots, n-1, \end{aligned} \tag{3.22}$$

if $S \sim \mathrm{GBM}(\mu, \sigma^2)$ then $(S(t)/S(0)) \sim LN([\mu - \tfrac{1}{2}\sigma^2]t, \sigma^2 t)$ and

$$\mathsf{E}[S(t)] = e^{\mu t} S(0), \quad \mathsf{Var}[S(t)] = e^{2\mu t} S^2(0)\left(e^{\sigma^2 t} - 1\right).$$

Issues with this approach

1

It has been widely observed across many markets that option prices are incompatible with a GBM model for the underlying asset (Glasserman, 103)

2

Consider a market in which several options with various strikes and maturities are traded simultaneously on the same underlying asset. Suppose the market is sufficiently liquid that we may effectively observe prices of the options without error. If the assumptions underlying the Black-Scholes formula held exactly, all of these option prices would result from using the same volatility parameter σ in the formula. In practice, one usually finds that this implied volatility actually varies with strike and maturity. It is therefore natural to seek a minimal modification of the Black-Scholes model capable of reproducing market prices.

Example: Short Rate Model

Assume an instantaneous continuously compounded short rate r(t). An investment in a money market account earning interest at rate r(u) at time u grows from a value of 1 at time 0 to a value of $\beta(t) = \exp\left(\int_0^t r(u)\,du\right)$ at time t. And the the time-0 price of a bond paying 1 at T is given by $B(0,T) = \mathrm{E}\left[\exp\left(-\int_0^T r(u)\,du\right)\right]$

The Vasicek model describes the short rate through an Ornstein-Uhlenbeck process: $dr(t) = \alpha(b - r(t))\,dt + \sigma\,dW(t)$.

W is a standard Brownian motion and *a, b,* and σ are positive constants. Notice that the drift is positive if r(t) < *b* and negative if r(t) > *b*; thus, r(t) is pulled toward level *b*. This is called *mean reversion*. We interpret *b* as a long-run interest rate level and *a* as the speed at which r(t) is pulled toward *b.*

For time varying *b*, the solution is: $r(t) = e^{-\alpha t}r(0) + \alpha\int_0^t e^{-\alpha(t-s)}b(s)\,ds + \sigma\int_0^t e^{-\alpha(t-s)}\,dW(s)$

Cox, Ingersoll, and Ross derived an expression for the price of a bond

$$B(t,T) = \mathsf{E}\left[\exp\left(-\int_t^T r(u)\,du\right)|r(t)\right]$$

when the short rate evolves according to (3.62). The bond price has the exponential affine form

$$B(t,T) = e^{-A(t,T)r(t)+C(t,T)}$$

as in a Gaussian short rate model, but with

$$A(t,T) = \frac{2(e^{\gamma(T-t)}-1)}{(\gamma+\alpha)(e^{\gamma(T-t)}-1)+2\gamma}$$

and

$$C(t,T) = \frac{2\alpha b}{\sigma^2}\log\left(\frac{2\gamma e^{(\alpha+\gamma)(T-t)/2}}{(\gamma+\alpha)(e^{\gamma(T-t)}-1)+2\gamma}\right)$$

and $\gamma = \sqrt{\alpha^2+2\sigma^2}$.

Variance Reduction Techniques

Monte Carlo integration has an error variance of σ^2/n. We get a better answer by sampling with a larger value of n, but the computing time grows with n. In Monte Carlo we use methods with the same answer as our original one but with a lower σ. Methods that do this are known as variance reduction techniques.

Common Techniques

Note that there are many more techniques specifically tailored to a problem that are often significantly more effective.

1. Control Variates
2. Antithetic Reduction
3. Stratified Sampling
4. Importance Sampling
5. Common Random Numbers
6. Conditioning
7. Moment Matching
8. Reweighting
9. Latin Hypercube Sampling
10. Score Function Approach

In statistics, we estimate an unknown mean $\mu = E(X)$ of a distribution by collecting n iid samples from the distribution, $X_1, \ldots, X_n$ and using the sample mean

$$\overline{X}(n) = \frac{1}{n}\sum_{j=1}^{n} X_j. \tag{1}$$

Letting $\sigma^2 = Var(X)$ denote the variance of the distribution, we conclude that

$$Var(\overline{X}(n)) = \frac{\sigma^2}{n}. \tag{2}$$

The *central limit theorem* asserts that as $n \to \infty$, the distribution of $Z_n \stackrel{\text{def}}{=} \frac{\sqrt{n}}{\sigma}(\overline{X}(n) - \mu)$ tends to $N(0,1)$, the unit normal distribution. Letting Z denote a $N(0,1)$ rv, we conclude that for n sufficiently large, $Z_n \approx Z$ in distribution. From here we obtain for any $z \geq 0$,

$$P(|\overline{X}(n) - \mu| > z\frac{\sigma}{\sqrt{n}}) \approx P(|Z| > z) = 2P(Z > z).$$

(We can obtain any value of $P(Z > z)$ by referring to tables, etc.)

For any $\alpha > 0$ no matter how small (such as $\alpha = 0.05$), letting $z_{\alpha/2}$ be such that $P(Z > z_{\alpha/2}) = \alpha/2$, we thus have

$$P(|\overline{X}(n) - \mu| > z_{\alpha/2}\frac{\sigma}{\sqrt{n}}) \approx \alpha,$$

which implies that the unknown mean μ lies within the interval $\overline{X}(n) \pm z_{\alpha/2}\frac{\sigma}{\sqrt{n}}$ with (approximately) probability $1 - \alpha$.

This allows us to construct *confidence intervals* for our estimate:

> *we say that the interval* $\overline{X}(n) \pm z_{\alpha/2}\frac{\sigma}{\sqrt{n}}$ *is a* $100(1-\alpha)\%$ *confidence interval for the mean* μ.

Typically, we would use (say) $\alpha = 0.05$ in which case $z_{\alpha/2} = z_{0.025} = 1.96$, and we thus obtain a 95% confidence interval $\overline{X}(n) \pm (1.96)\frac{\sigma}{\sqrt{n}}$.

The length of the confidence interval is $2(1.96)\frac{\sigma}{\sqrt{n}}$ which of course tends to 0 as the sample size n gets larger.

In practice we would not actually know the value of σ^2; it would be unknown (just as μ is). But this is not really a problem: we instead use an estimate for it, the *sample variance* $s^2(n)$ defined by

$$s^2(n) = \frac{1}{n-1}\sum_{j=1}^{n}(X_j - \overline{X}_n)^2.$$

It can be shown that $s^2(n) \to \sigma^2$, with probability 1, as $n \to \infty$ and that $E(s^2(n)) = \sigma^2$, $n \geq 2$.

So, in practice we would use $s(n)$ is place of σ when constructing our confidence intervals. For example, a 95% confidence interval is given by $\overline{X}(n) \pm (1.96)\frac{s(n)}{\sqrt{n}}$.

The following recursions can be derived; they are useful when implementing a simulation requiring a confidence interval:

$$\overline{X}_{n+1} = \overline{X}_n + \frac{X_{n+1} - \overline{X}_n}{n+1},$$

$$S^2_{n+1} = \left(1 - \frac{1}{n}\right)S^2_n + (n+1)(\overline{X}_{n+1} - \overline{X}_n)^2.$$

Monte Carlo integration typically has an error variance of the form σ^2/n. We get a better answer by sampling with a larger value of n, but the computing time grows with n. In Monte Carlo we use methods with the same answer as our original one but with a lower σ. Methods to do this are known as variance reduction techniques

Confidence Intervals and Errors

http://www.columbia.edu/~ks20/4703-Sigman/4703-07-Notes-ATV.pdf

Confidence intervals in MATLAB may be computed using the `normfit` function.

By default, `normfit` returns a 95% confidence interval, and different values may be specified, as usual in MATLAB, by passing an optional parameter.

```
function [Price, CI, dif] = BlsMC2(S0,K,r,T,sigma,NRepl)

DiscPayoff = exp(-r*T)*max(0, S0*exp((r - 0.5*sigma^2)*T+sigma * sqrt(T)*randn(NRepl,1))-K);

%we have three outputs for normfit, even though we discard varPrice
[Price, varPrice, CI] = normfit(DiscPayoff);
dif = (CI(2)-CI(1))/Price;

end
```

```
S0=50;
K=55;
r=0.05;
T=5/12;
sigma=0.2;
Call = blsprice(S0,K,r,T,sigma)
[CallMC, CI, dif] = BlsMC2(S0,K,r,T,sigma,50000)
[CallMC, CI, dif] = BlsMC2(S0,K,r,T,sigma,1000000)
```

```
Call =

    1.1718


CallMC =

    1.1932


CI =

    1.1685
    1.2180


dif =

    0.0415
```

```
Call =

    1.1718


CallMC =

    1.1732


CI =

    1.1677
    1.1787


dif =

    0.0094
```

Control Variates

To describe the method, we let $Y_1, \ldots, Y_n$ be outputs from n replications of a simulation. For example, Y_i could be the discounted payoff of a derivative security on the ith simulated path. Suppose that the Y_i are independent and identically distributed and that our objective is to estimate $\mathsf{E}[Y_i]$. The usual estimator is the sample mean $\bar{Y} = (Y_1 + \cdots + Y_n)/n$. This estimator is unbiased and converges with probability 1 as $n \to \infty$.

Suppose, now, that on each replication we calculate another output X_i along with Y_i. Suppose that the pairs (X_i, Y_i), $i = 1, \ldots, n$, are i.i.d. and that the expectation $\mathsf{E}[X]$ of the X_i is known. (We use (X, Y) to denote a generic pair of random variables with the same distribution as each (X_i, Y_i).) Then for any fixed b we can calculate

$$Y_i(b) = Y_i - b(X_i - \mathsf{E}[X])$$

from the ith replication and then compute the sample mean

$$\bar{Y}(b) = \bar{Y} - b(\bar{X} - \mathsf{E}[X]) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - b(X_i - \mathsf{E}[X])). \tag{4.1}$$

This is a control variate estimator; the observed error $\bar{X} - \mathsf{E}[X]$ serves as a control in estimating $\mathsf{E}[Y]$.

Each $Y_i(b)$ has variance

$$\begin{aligned}\mathsf{Var}[Y_i(b)] &= \mathsf{Var}\left[Y_i - b(X_i - \mathsf{E}[X])\right] \\ &= \sigma_Y^2 - 2b\sigma_X\sigma_Y\rho_{XY} + b^2\sigma_X^2 \equiv \sigma^2(b),\end{aligned} \tag{4.2}$$

where $\sigma_X^2 = \mathsf{Var}[X]$, $\sigma_Y^2 = \mathsf{Var}[Y]$, and ρ_{XY} is the correlation between X and Y. The control variate estimator $\bar{Y}(b)$ has variance $\sigma^2(b)/n$ and the ordinary sample mean $\bar{Y}$ (which corresponds to $b = 0$) has variance σ_Y^2/n. Hence, the control variate estimator has smaller variance than the standard estimator if $b^2\sigma_X < 2b\sigma_Y\rho_{XY}$.

The optimal coefficient b^* minimizes the variance (4.2) and is given by

$$b^* = \frac{\sigma_Y}{\sigma_X}\rho_{XY} = \frac{\mathsf{Cov}[X, Y]}{\mathsf{Var}[X]}.$$

Substituting this value in (4.2) and simplifying, we find that the ratio of the variance of the optimally controlled estimator to that of the uncontrolled estimator is

$$\frac{\mathsf{Var}[\bar{Y} - b^*(\bar{X} - \mathsf{E}[X])]}{\mathsf{Var}[\bar{Y}]} = 1 - \rho_{XY}^2.$$

See note below

- With the optimal coefficient b^*, the effectiveness of a control variate, as measured by the variance reduction ratio (4.4), is determined by the strength of the correlation between the quantity of interest Y and the control X. The sign of the correlation is irrelevant because it is absorbed in b^*.
- If the computational effort per replication is roughly the same with and without a control variate, then (4.4) measures the computational speed-up resulting from the use of a control. More precisely, the number of replications of the Y_i required to achieve the same variance as n replications of the control variate estimator is $n/(1-\rho_{XY}^2)$.
- The variance reduction factor $1/(1-\rho_{XY}^2)$ increases very sharply as $|\rho_{XY}|$ approaches 1 and, accordingly, it drops off quickly as $|\rho_{XY}|$ decreases away from 1. For example, whereas a correlation of 0.95 produces a ten-fold speed-up, a correlation of 0.90 yields only a five-fold speed-up; at $|\rho_{XY}| = 0.70$ the speed-up drops to about a factor of two. This suggests that a rather high degree of correlation is needed for a control variate to yield substantial benefits.

Even though we don't know b^* we can use an optimal estimate:

$$\hat{b}_n = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^n (X_i - \bar{X})^2}$$

A particularly effective example of this idea was suggested by Kemna and Vorst [209] for the pricing of Asian options. Accurate pricing of an option on the arithmetic average

$$\bar{S}_A = \frac{1}{n}\sum_{i=1}^n S(t_i)$$

requires simulation, even if S is geometric Brownian motion. In contrast, calls and puts on the geometric average

$$\bar{S}_G = \left(\prod_{i=1}^n S(t_i)\right)^{1/n}$$

can be priced in closed form, as explained in Section 3.2.2. Thus, options on $\bar{S}_G$ can be used as control variates in pricing options on $\bar{S}_A$.

Pricing an arithmetic average Asian option

The option payoff is

$$\max\left\{\frac{1}{N}\sum_{i=1}^{N} S(t_i) - K, 0\right\},$$

where the option maturity is T years, $t_i = i\,\delta t$, and $\delta t = T/N$.

```
function [P,CI] = AsianMC(S0,K,r,T,sigma,NSamples,NRepl)
Payoff = zeros(NRepl,1);
for i=1:NRepl
   Path=AssetPaths(S0,r,sigma,T,NSamples,1);
   Payoff(i) = max(0, mean(Path(2:(NSamples+1))) - K);
end
[P,aux,CI] = normfit( exp(-r*T) * Payoff);
```

Note that `NSamples` is the number N of sampled points to compute the arithmetic average, which should not be confused with the number of replications `NRepl`.

In this case, we have to generate whole sample paths; we need observations only at the time instants specified by the contract, but we may still have to generate a large amount of data.

Using sum of prices as a control variate

The expected value of the sum of the stock prices Y, as defined in (9), is (under the risk-neutral measure):

$$\begin{aligned} \mathsf{E}[Y] &= \mathsf{E}\left[\sum_{i=0}^{N} S(t_i)\right] = \sum_{i=0}^{N} \mathsf{E}[S(i\,\delta t)] \\ &= \sum_{i=0}^{N} S(0)e^{r i\,\delta t} = S(0)\sum_{i=0}^{N}[e^{r\,\delta t}]^i = S(0)\frac{1-e^{r(N+1)\delta t}}{1-e^{r\,\delta t}}, \end{aligned}$$

where we have used the following formula:

$$\sum_{i=0}^{N} \alpha^i = \frac{1-\alpha^{N+1}}{1-\alpha}.$$

Asian options have the payoff is determined by the average underlying price over some pre-set period of time.
In 1987 Standish and Spaughton were in Tokyo on business when "they developed the first commercially used pricing formula for options linked to the average price of crude oil." They called this exotic option the Asian option because they were in Asia.

```
function [P,CI] = AsianMCCV(S0,K,r,T,sigma,NSamples,NRepl,NPilot)
% pilot replications to set control parameter
TryPath=AssetPaths(S0,r,sigma,T,NSamples,NPilot);
StockSum = sum(TryPath,2);
PP = mean(TryPath(:,2:(NSamples+1)) , 2);
TryPayoff = exp(-r*T) * max(0, PP - K);
MatCov = cov(StockSum, TryPayoff);
c = - MatCov(1,2) / var(StockSum);
dt = T / NSamples;
ExpSum = S0 * (1 - exp((NSamples + 1)*r*dt)) / (1 - exp(r*dt));
% MC run
ControlVars = zeros(NRepl,1);
for i=1:NRepl
   StockPath = AssetPaths(S0,r,sigma,T,NSamples,1);
   Payoff = exp(-r*T) * max(0, mean(StockPath(2:(NSamples+1))) - K);
   ControlVars(i) = Payoff + c * (sum(StockPath) - ExpSum);
end
[P,aux,CI] = normfit(ControlVars);
```

Using geometric average Asian option as a control variate

This is based on the exploitation of much deeper knowledge.

The payoff of the discrete-time, geometric average Asian option is

$$\max\left\{\left(\prod_{i=1}^{N} S(t_i)\right)^{1/N} - K,\ 0\right\}.$$

Since the product of lognormal random variables is still lognormal, it is possible to find an analytical formula for the price of the geometric average option, which looks like a modified Black–Scholes formula.

Here m is the last time at which we observed the price of the underlying asset, q is the continuous dividend yield, and G_t is the current geometric average:

$$P_{GA} = e^{-rT}\left[e^{a+\frac{1}{2}b}N(x) - KN\left(x-\sqrt{b}\right)\right],$$

where

$$a = \frac{m}{N}\log(G_t) + \frac{N-m}{N}\left[\log(S_0) + \nu(t_{m+1}-t) + \frac{1}{2}\nu(T-t_{m+1})\right]$$

$$b = \frac{(N-m)^2}{N^2}\sigma^2(t_{m+1}-t) + \frac{\sigma^2(T-t_{m+1})}{6N^2}(N-m)(2(N-m)-1)$$

$$\nu = r - q - \frac{1}{2}\sigma^2$$

$$x = \frac{a - \log(K) + b}{\sqrt{b}}.$$

```
function P = GeometricAsian(S0,K,r,T,sigma,delta,NSamples)
dT = T/NSamples;
nu = r - sigma^2/2-delta;
a = log(S0)+nu*dT+0.5*nu*(T-dT);
b = sigma^2*dT + sigma^2*(T-dT)*(2*NSamples-1)/6/NSamples;
x = (a-log(K)+b)/sqrt(b);
P = exp(-r*T)*(exp(a+b/2)*normcdf(x) - K*normcdf(x-sqrt(b)));
```

```
function [P,CI] = AsianMCGeoCV(S0,K,r,T,sigma,NSamples,NRepl,NPilot)
% precompute quantities
DF = exp(-r*T);
GeoExact = GeometricAsian(S0,K,r,T,sigma,0,NSamples);
% pilot replications to set control parameter
GeoPrices = zeros(NPilot,1);
AriPrices = zeros(NPilot,1);
for i=1:NPilot
    Path=AssetPaths(S0,r,sigma,T,NSamples,1);
    GeoPrices(i)=DF*max(0,(prod(Path(2:(NSamples+1))))^(1/NSamples) - K);
    AriPrices(i)=DF*max(0,mean(Path(2:(NSamples+1))) - K);
end
MatCov = cov(GeoPrices, AriPrices);
c = - MatCov(1,2) / var(GeoPrices);
% MC run
ControlVars = zeros(NRepl,1);
for i=1:NRepl
   Path = AssetPaths(S0,r,sigma,T,NSamples,1);
   GeoPrice = DF*max(0, (prod(Path(2:(NSamples+1))))^(1/NSamples) - K);
   AriPrice = DF*max(0, mean(Path(2:(NSamples+1))) - K);
   ControlVars(i) = AriPrice + c * (GeoPrice - GeoExact);
end
[P,aux,CI] = normfit(ControlVars);
```

Note: It can also be done pretty simply in VBA

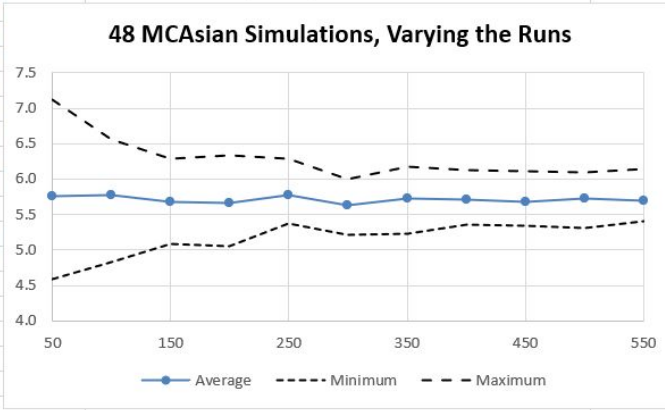
```
Function MCAsian(initial, Exercise, Up, Down, Interest, Periods, Runs)
    Dim PricePath() As Double
    ReDim PricePath(Periods + 1)

    'Risk-neutral probabilities
    piup = (Interest - Down) / (Up - Down)
    pidown = 1 - piup

    Temp = 0

    For Index = 1 To Runs
        'Generate path
        For i = 1 To Periods
            PricePath(0) = initial
            pathprob = 1
            If Rnd > pidown Then
              PricePath(i) = PricePath(i - 1) * Up
              Else:
              PricePath(i) = PricePath(i - 1) * Down
            End If
         Next i

        PriceAverage = Application.Sum(PricePath) / (Periods + 1)
            callpayoff = Application.Max(PriceAverage - Exercise, 0)
            Temp = Temp + callpayoff

   Next Index

   MCAsian = (Temp / Interest ^ Periods) / Runs

End Function
```

Excel Example of Asian Option with MC

	A	B	C	D	E	F
2	S_0, current stock price	50				
3	X, exercise price	45				
4	T, time to option exercise	0.4				
5	r, interest rate	8%				
6	μ, mean stock return	15%				
7	σ, standard deviation of stock return	22%				
8						
9	n, number of sub-intervals of T	80				
10	Delta t	0.0125	<-- =1/B9			
11						
12	Up over 1 sub-interval	1.0268	<-- =EXP(B6*B10+B7*SQRT(B10))			
13	Down over 1 sub-interval	0.9775	<-- =EXP(B6*B10-B7*SQRT(B10))			
14	Interest over 1 sub-interval	1.0010	<-- =EXP(B5*B10)			
15						
16	Runs	100				
17						
18	6.0663	5.5749	5.1989	6.5943	5.8307	6.2219
19	6.0150	5.1820	6.0853	5.3960		
20	5.9617	6.0517	6.0473	5.3968		
21	5.2084	5.5224	5.5413	6.1105		
22	5.5034	5.9003	5.7040	5.8275		
23	5.5131	5.7839	5.5618	5.3770		
24	5.7846	5.8097	5.3368	5.7791		
25	5.5469	5.2169	5.8193	6.1604		
26						
27	Average of above	5.7145	<-- =AVERAGE(A18:F25)			
28	Minimum	4.9899	<-- =MIN(A18:F25)			
29	Maxmimum	6.7172	<-- =MAX(A18:F25)			
30	Standard deviation	0.3870	<-- =STDEV(A18:F25)			

Data table: Sensitivity of results on number of runs

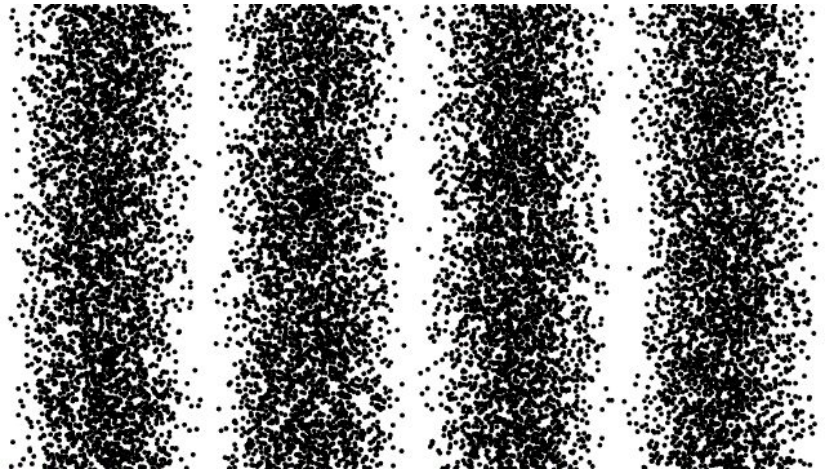
Runs	Average of 48 MCAsian	Minimum	Maximum	Standard deviation
				<-- =B30 , data table header (hidden)
50	5.7577	4.5936	7.1255	0.5749
100	5.7789	4.8304	6.5544	0.3679
150	5.6701	5.0797	6.2946	0.2655
200	5.6659	5.0480	6.3283	0.2698
250	5.7706	5.3664	6.2948	0.2355
300	5.6322	5.2075	5.9967	0.1776
350	5.7254	5.2372	6.1741	0.2085
400	5.7041	5.3640	6.1228	0.1683
450	5.6792	5.3484	6.1136	0.1661
500	5.7302	5.3146	6.1009	0.1750
550	5.6921	5.3985	6.1492	0.1378



Antithetic Reduction

The method of *antithetic variates* attempts to reduce variance by introducing negative dependence between pairs of replications. The method can take various forms; the most broadly applicable is based on the observation that if U is uniformly distributed over $[0,1]$, then $1-U$ is too. Hence, if we generate a path using as inputs $U_1,\ldots,U_n$, we can generate a second path using $1-U_1,\ldots,1-U_n$ without changing the law of the simulated process. The variables U_i and $1-U_i$ form an antithetic pair in the sense that a large value of one is accompanied by a small value of the other. This suggests that an unusually large or small output computed from the first path may be balanced by the value computed from the antithetic path, resulting in a reduction in variance.

These observations extend to other distributions through the inverse transform method: $F^{-1}(U)$ and $F^{-1}(1-U)$ both have distribution F but are antithetic to each other because F^{-1} is monotone. For a distribution symmetric about the origin, $F^{-1}(1-u)$ and $F^{-1}(u)$ have the same magnitudes but opposite signs. In particular, in a simulation driven by independent standard normal random variables, antithetic variates can be implemented by pairing a sequence $Z_1, Z_2, \ldots$ of i.i.d. $N(0,1)$ variables with the sequence $-Z_1, -Z_2, \ldots$ of i.i.d. $N(0,1)$ variables, whether or not they are sampled through the inverse transform method. If the Z_i are used to simulate the increments of a Brownian path, then the $-Z_i$ simulate the increments of the reflection of the path about the origin. This again suggests that running a pair of simulations using the original path and then its reflection may result in lower variance.



Let X_i denote our copies of X (each has the same distribution hence the same mean μ and variance σ^2) but let us not assume that they are independent. Let $n = 2m$, for some $m \geq 1$, that is, n is even. Note that

$$\overline{X}(n) = \frac{1}{2m}\sum_{j=1}^{2m} X_j = \frac{1}{m}\sum_{j=1}^{m} Y_j = \overline{Y}(m), \tag{3}$$

where

$$\begin{aligned} Y_1 &= \frac{X_1 + X_2}{2} \\ Y_2 &= \frac{X_3 + X_4}{2} \\ &\vdots \\ Y_m &= \frac{X_{n-1} + X_n}{2}, \end{aligned}$$

and we conclude that

The two estimators $\overline{Y}(m)$ and $\overline{X}(n)$ for $E(X)$ in (3) are identical.

Because they are identical, we can and will use $\overline{Y}(m)$ in what follows. Moreover, $E(Y_i) = E(X) = \mu$ (remember we are assuming that the X_i all have the same distribution hence the same mean). This means that for purposes of argument here we can view each Y_i as the end

"copy" that we wish to simulate from (instead of the X_i). We let $Y = \frac{X_1+X_2}{2}$ denote a generic Y_i. The problem of estimation can be re-cast as "we are trying to estimate $\mu = E(Y)$".

Computing variances,

$$\begin{aligned} Var(Y) &= (1/4)(\sigma^2 + \sigma^2 + 2Cov(X_1, X_2)) \\ &= (1/2)(\sigma^2 + Cov(X_1, X_2)). \end{aligned}$$

In the case when the X_i are iid, $Cov(X_1, X_2) = 0$ and thus $Var(Y) = \sigma^2/2$ yielding (as we already know, recall (2)) $Var(\overline{Y}(m)) = \frac{\sigma^2}{n}$.

But if $Cov(X_1, X_2) < 0$, then $Var(Y) < (1/2)\sigma^2$ yielding $Var(\overline{Y}(m)) < \frac{\sigma^2}{n}$; variance is reduced. So it is in our interest to somehow create some negative correlation within each pair (X_1, X_2), $(X_3, X_4), \ldots$, but keep the pairs iid so that the Y_i are iid (and thus the CLT still applies); for then $Var(\overline{Y}(m))$ will be lowered from what it would be if we simply used iid copies of the X_i.

```
function [Price, CI, dif] = BlsMCAV(S0,K,r,T,sigma,NPairs)

Payoff1 = max( 0 , S0*exp((r - 0.5*sigma^2)*T+sigma * sqrt(T)*randn(NPairs,1)) - K);
Payoff2 = max( 0 , S0*exp((r - 0.5*sigma^2)*T+sigma * sqrt(T)*(-randn(NPairs,1))) - K);
DiscPayoff = exp(-r*T) * 0.5 * (Payoff1+Payoff2);
[Price, VarPrice, CI] = normfit(DiscPayoff);
dif = (CI(2)-CI(1))/Price;

end
```

```
S0=50;
K=55;
r=0.05;
T=5/12;
sigma=0.2;
Call = blsprice(S0,K,r,T,sigma);
[CallMC1, CI1, dif1] = BlsMC2(S0,K,r,T,sigma,1000000);
[CallMC2, CI2, dif2] = BlsMCAV(S0,K,r,T,sigma,1000000);

Call
CallMC1
CallMC2
```

```
Call =

    1.1718


CallMC1 =

    1.1745


CallMC2 =

    1.1675
```

Stratified Sampling

Suppose we have u_i random numbers and expect to take $z_i=N^{-1}(u_i)$. Because of random variation, 100 uniform numbers will not be exactly uniformly distributed and therefore the z_i will not be exactly normal. Instead, break it up into strata. For instance, take $u_1/100$. This will be uniformly distributed over [0, .01]. And take $u_2/100$ +0.1, which will now be distributed over (.01,.02), etc.

An interesting note related to probability

If the mapping $\alpha \to Y(\alpha)$ is "nice" enough, we can interchange the order of taking expected value and derivative,

$$\frac{dE(Y)}{d\alpha} = E\Big[\frac{dY}{d\alpha}\Big]. \tag{1}$$

Under this scenario, $K'(\alpha)$ itself is an expected value so we can estimate it by standard Monte Carlo: Simulate n iid copies of $\frac{dY}{d\alpha}$ and take the empirical average.

To dispense with the notion that such an interchange as in (1) is always possible (no, it is not!) one merely need consider the Δ of a digital option with payoff $Y = Y(S_0) = e^{-rT} I\{S(T) > K\}$, where $S(T) = S_0 e^{X(T)} = S_0 e^{\sigma B(T) + (r - \sigma^2/2)t}$. (The risk-neutral probability is being used for pricing purposes.) In this case, $\frac{dY}{dS_0} = 0$ since the indicator is a piecewise constant function of S_0, thus $E\Big[\frac{dY}{dS_0}\Big] = 0$. But $E(Y) = e^{-rT} P(S(T) > K)$ is a nice smooth function of $S_0 > 0$, with a non-zero derivative. (Yes, the sample paths of Y are not differentiable (nor continuous even) at the value of S_0 for which $S(T) = K$, but $P(S(T) = K) = 0$, so this point can be ignored.)

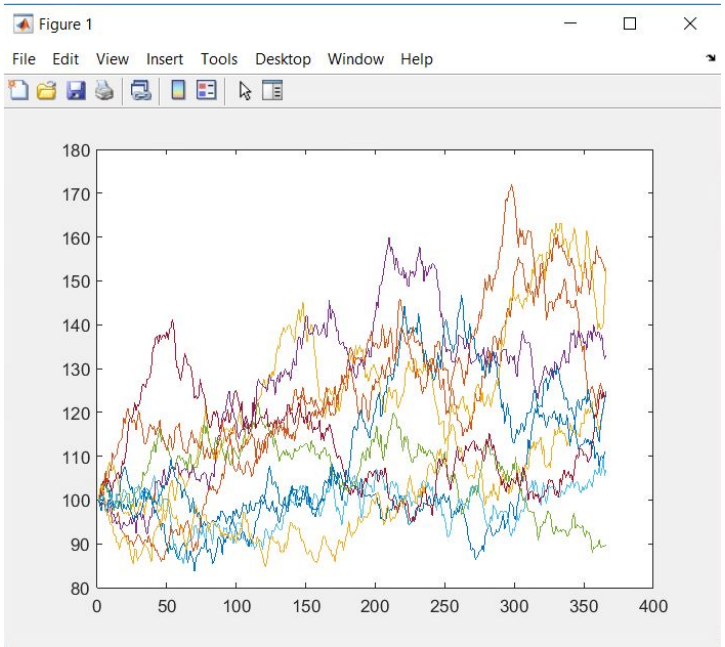
On the other hand, a European call option, with payoff $Y = e^{-rT}(S(T) - K)^+$ satisfies $\frac{dY}{dS_0} = e^{-rT} e^{X(T)} I\{S(T) > K\}$ which can be re-written as $\frac{dY}{dS_0} = e^{-rT} \frac{S(T)}{S_0} I\{S(T) > K\}$, and indeed it can be proved that (1) holds.

http://www.columbia.edu/~ks20/4703-Sigman/4703-07-Notes-Greeks.pdf

Comparison Testing

```
function sample = MCpaths1(start,mu,sigma,T,steps,replications)
sample = zeros(replications,1+steps); %nXk matrix for the paths
sample(:,1) = start; %start every path at S0

for i = 1:replications
    for j = 1:steps
        sample(i,j+1) = sample(i,j)*exp((mu-.5*sigma^2)*(T/steps)+sigma*sqrt(T/steps)*randn);
    end
end

end
```

```
numPaths = 10;
test = MCpaths1(100,.2,.3,1,365,numPaths);
for i = 1:numPaths
    t = 1:length(test);
    x = [test(i,:)];
   plot(t,x);
   hold on
end
hold off
```



Monte Carlo sample option path creation

```
function P = stopLoss(K,r,T,Paths)
[NRepl,NSteps] = size(Paths);
NSteps = NSteps - 1; %true number of steps
Cost = zeros(NRepl,1);
dt = T/NSteps;
DiscountFactors = exp(-r*(0:1:NSteps)*dt);
for k = 1:NRepl
    CashFlows = zeros(NSteps+1,1);
    if(Paths(k,1) >= K)
        Covered = 1;
        CashFlows(1) = -Paths(k,1);
    else
        Covered = 0;
    end
    for t = 2:NSteps+1
        if(Covered == 1) & (Paths(k,t) < K )
            %sell
            Covered = 0;
            CashFlows(t) = Paths(k,t);
        elseif (Covered == 0) & (Paths(k,t) > K)
            %buy
            Covered = 1;
            CashFlows(t) = -Paths(k,t);
        end
    end

    if Paths(k,NSteps+1) >= K
        %exercise
        CashFlows(NSteps+1) = CashFlows(NSteps+1)+K;
    end
    Cost(k) = -dot(DiscountFactors,CashFlows);
end
P = mean(Cost);
```

```
test = MCpaths2(50,.1,.4,(5/12),1000,100000);
stopLoss(50,.05,(5/12),test)
blsprice(50,50,.05,5/12,.4)
```

```
>> checkStopLoss

ans =

    5.6591


ans =

    5.6150
```

The option price is essentially the cost of delta-hedging due to a no arbitrage argument. Also the continuous time hedging strategy for a call requires holding some amount of the underlying asset.
A simpler stop-loss strategy wherein we cover our position by holding one share when the option is in the money and going naked (holding no shares) when the option is out of the money should work just as well.
We buy a share above K and sell it below K.
Since we assume deterministic and constant interest rates, we will account for the borrowed money by recording the cash flows and discounting them back to time 0 with *DiscountFactors*.
Covered indicates we have bought the stock.
Cash flow is negative when we buy a stock and positive when we sell.
The option is evaluated as the average total discounted cash flow, with a change in sign.
If the option is exercised we will also make the strike price.

Simple Stop Loss Strategy modeled with Monte Carlo

```
function P = deltaHedge(K,sigma,r,T,Paths)
[NRepl,NSteps] = size(Paths);
NSteps = NSteps -1;
Cost = zeros(NRepl,1);
CashFlows = zeros(1,NSteps+1);
dt = T/NSteps;
DiscountFactors = exp(-r*(0:1:NSteps)*dt);
for i = 1:NRepl
    Path = Paths(i,:);
    Position = 0;
    Deltas = blsdelta(Path(1:NSteps),K,r,T-(0:NSteps-1)*dt,sigma);
    for j = 1:NSteps
        CashFlows(j) = (Position-Deltas(j))*Path(j);
        Position = Deltas(j);
    end
    if Path(NSteps+1) > K
        CashFlows(NSteps+1) = K - (1-Position)*Path(NSteps+1);
    else
        CashFlows(NSteps+1) = Position*Path(NSteps+1);
    end
    Cost(i) = -CashFlows*DiscountFactors';
end
P= mean(Cost);
```

Comparison for n= 100 -> 100,000 steps in MC for Stop/Loss strategy vs Delta-Hedge.

```
S0 = 50;
K = 52;
mu = .1;
sigma = .4;
r = .05;
T = 5/12;
NRepl = 100;
NSteps = 10;

fprintf(1,'%s %f\n', 'true price = ', blsprice(S0,K,r,T,sigma));

for i = 1:5
    NSteps = NSteps*10;
    Paths = paths(S0,mu,sigma, T, NSteps,NRepl);
    A = stopLoss(K,r,T,Paths);
    fprintf(1,'cost of stop/loss (S) = %f\n', A);
    B = deltaHedge(K,sigma,r,T,Paths);
    fprintf(1,'cost of delta-hedging = %f\n', B);
    fprintf(1,'\n');
end
```

```
>> test
true price =  4.732837
cost of stop/loss (S) = 4.665015
cost of delta-hedging = 4.730173

cost of stop/loss (S) = 4.397751
cost of delta-hedging = 4.715636

cost of stop/loss (S) = 4.411557
cost of delta-hedging = 4.734747

cost of stop/loss (S) = 4.783660
cost of delta-hedging = 4.733745

cost of stop/loss (S) = 4.583152
cost of delta-hedging = 4.732826
```

```
S0 = 50; K = 52; mu = .1; sigma = .4; r = .05; T = 5/12;
NRepl = 100;NSteps = 10;
n = 5;

error1 = zeros(1,n);
error2 = zeros(1,n);
bls =  blsprice(S0,K,r,T,sigma);

for i = 1:n
    NSteps = NSteps*10;
    Paths = paths(S0,mu,sigma, T, NSteps,NRepl);
    A = stopLoss(K,r,T,Paths);
    B = deltaHedge(K,sigma,r,T,Paths);
    error1(i) = A;
    error2(i) = B;
end

rmse1 = sqrt(sum((error1-bls).^2)/n);
rmse2 = sqrt(sum((error2-bls).^2)/n);
fprintf(1,'Stop/Loss RMSE: %f\n',rmse1);
fprintf(1,'Delta-Hedge RMSE: %f\n',rmse2);
```

```
>> test
Stop/Loss RMSE: 0.401440
Delta-Hedge RMSE: 0.018182
>> test
Stop/Loss RMSE: 0.425890
Delta-Hedge RMSE: 0.006103
>> test
Stop/Loss RMSE: 0.434500
Delta-Hedge RMSE: 0.018633
```

Stop/Loss Vs Delta-Hedge Script and Error Comparison

```
>> AsianOpt

call =

    5.2227
    5.3225
    5.3468
    5.3520
    5.3552


error =

    0.0158
    0.0049
    0.0015
    0.0005
    0.0002
```

```
CI =

    5.0609    5.3845
    5.2713    5.3736
    5.3306    5.3631
    5.3469    5.3572
    5.3536    5.3568
```

Monte Carlo Asian Option Pricing with Error and Confidence Intervals. Using runs from 10^2 to 10^7.

```
r = .07; sigma = .2; K =35; S0 = 40; T = 4/12; n = 88; dt = T/n;
upTo = 5;
call = ones(upTo,1); error = ones(upTo,1); width = ones(upTo,1);
CI = ones(upTo,2);

for j = 1:upTo
    N = 100;
    N = N*10^j;
    X = ones(N,1);
    for i = 1:N
        path = cumprod([S0,exp((r-sigma^2/2)*dt + sigma *sqrt(dt)*randn(1,n))]);
        X(i) = exp(-r*T)*max(mean(path)-K,0);
    end
    call(j) = mean(X);
    error(j) = std(X)/call(j)/sqrt(N);
    width(j) = std(X)*norminv(.975)/sqrt(N);
    CI(j,1) = call(j)-width(j); CI(j,2) = call(j)+width(j);

end
    call, error, CI
```

```
r = .07; sigma = .2; K =35; S0 = 40; T = 4/12; n = 88; dt = T/n;  N = 1000; X = ones(N,1); tX = X;

for i = 1:N
     path = cumprod([S0,exp((r-sigma^2/2)*dt + sigma *sqrt(dt)*randn(1,n))]);
     X(i) = exp(-r*T)*max(mean(path)-K,0);
     tX(i) = exp(-r*T)*max(prod(path)^(1/(n+1))-K,0);
end

call = mean(X);
width = std(X)*norminv(.975)/sqrt(N);
CI = [call-width,call+width];

%expectation of control variable - geometric Asian option price
a1 = (log(S0/K)+(r-sigma^2/6+sigma^2/3)*T/2)/sigma/sqrt(T/3);
a2 = (log(S0/K)+(r-sigma^2/6-sigma^2/3)*T/2)/sigma/sqrt(T/3);
geoCall = exp(-(6*r+sigma^2)*T/12)*S0*normcdf(a1)-K*exp(-r*T)*normcdf(a2);

Cov = cov([X,tX]);
alpha = Cov(1,2)/Cov(1,1);

ellCall = call-alpha*mean(tX-geoCall) %variance reduction
width = 1.96*sqrt((1-Cov(1,2)^2/Cov(1,1)/Cov(2,2))/N*Cov(1,1));
CI = [ellCall-width,ellCall+width]
```

```
>> AsianOptCV

ellCall =

    5.3556


CI =

    5.3533    5.3580
```

We use a Geometric Asian option as the control variable for the Arithmetic Asian option. It is highly correlated (and has a closed form). We see that N = 1000 is equivalent 10^8 trial runs without the control variable!

Comparison of Crude Monte Carlo against Conditional Monte Carlo for Down and Out Options

```
function [Pdon CI, NCrossed] = DOPutMCCond(S0,K,r,T,sigma,Sb,NSteps,NRepl)
dt = T/NSteps;
[Call,Put] = blsprice(S0,K,r,T,sigma);
NCrossed = 0; Payoff = ones(NRepl,1); Times = ones(NRepl,1); StockVals = ones(NRepl, 1);

for i = 1:NRpel
    Path = AssetPaths(S0,r,sigma,T,NSteps,1);
    tcrossed = min(find(Path <= Sb));
    if not(isempty(tcrossed))
        NCrossed = NCrossed +1;
        Times(NCrossed) = (tcrossed-1)*dt;
        StockVals(NCrossed) = Path(tcrossed);
    end
end
if (NCrossed >0)
    [CallAux, PutAux] = blsprice(StockVals(1:NCrossed),K,r,T-Times(1:NCrossed),sigma);
    Payoff(1:NCrossed) = exp(-r*Times(1:NCrossed)).*PutAuc;
end
[Pdo, aux, CI] = normfit(Put-Payoff);
```

```
function sample = AssetPaths(start,mu,sigma,T,steps,replications)
sample = zeros(replications,1+steps); %nXk matrix for the paths
sample(:,1) = start; %start every path at S0

for i = 1:replications
    for j = 1:steps
        sample(i,j+1) = sample(i,j)*exp((mu-.5*sigma^2)*(T/steps)+sigma*sqrt(T/steps)*randn);
    end
end

end
```

```
function [P CI, NCrossed] = DOPutMCCrude(S0,K,r,T,sigma,Sb,NSteps,NRepl)

[Call,Put] = blsprice(S0,K,r,T,sigma);
NCrossed = 0; Payoff = ones(NRepl,1);

for i = 1:NRepl
    Path = AssetPaths(S0,r,sigma,T,NSteps,1);
    crossed = any(Path <= Sb);
    if crossed == 0
        Payoff(i) = max(0,K-Path(NSteps+1));
    else
        Payoff(i) = 0;
        NCrossed = NCrossed +1;
    end
end

[P, aux, CI] = normfit(exp(-r*T)*Payoff);
```

References

1. Monte Carlo Methods in Financial Engineering, Glasserman, Springer, 2003
2. Stochastic Simulation and Monte Carlo Methods, Graham, Springer, 2013
3. A Search for Good Pseudo-random Number Generators : Survey and Empirical Studies, Bhattacharjee, arXiv, 3 Nov 2018
4. Numerical Methods in Finance and Economics, Wiley, Brandimarte, 2006
5. Derivatives Markets, MacDonald, Pearson, 2006
6. Handbook of Monte Carlo Methods, Kroese, Wiley, 2011