Monte Carlo Simulation and Binomial Pricing with MATLAB

Luca Regis

IMT Institute for Advanced Studies, Lucca

Additional Statistical Training

A.Y. 2015/2016

Contents

1. Introduction
2. Law of Large Numbers (LLN) and Central Limit Theorem (CLT)
3. Generating Random Variables
4. Generating Sample Paths
5. Pricing Derivatives via Monte Carlo simulation
6. Variance reduction techniques
7. Lattices and binomial pricing

Section

1 Introduction

2 Law of Large Numbers (LLN) and Central Limit Theorem (CLT)

3 Generating Random Variables

4 Generating Sample Paths

5 Pricing Derivatives via Monte Carlo simulation

6 Variance reduction techniques

7 Lattices and binomial pricing

Monte Carlo Simulation

- Monte Carlo methods are algorithms that make use of repeated random sampling to solve probabilistic problems.
- They make use of the analogy between probability and volumes (measures): each event is associated to a set of outcomes whose probability is a measure (volume) relative to the universe of possible outcomes.
- Monte Carlo computes the volume of the set as being a probability by generating many possible scenarios and taking the fraction of draws that fall in a given set as an estimate of the volume.

Monte Carlo Simulations/2

- Integrals, as well as expectations, can be effectively computed using Monte Carlo simulation.
- The law of large numbers ensures that, when the number of draws is large enough, our estimates of volumes or integrals get close to their real value. The central limit theorem controls the speed of convergence.
- They allow to tackle those problems for which no analytical solution is present.

Financial Applications

- Study the statistical properties of portfolios: expected returns, risk, effectiveness of hedging strategies,...
- Computing the values of derivatives, which can be represented as expected values in a risk neutral world.

How to use Monte Carlo in Finance

- Represent the relevant basic financial quantities using models;
- Define valuation models for more complex contracts (options/derivatives/contingent claims) that depend on the basic quantities above;
- Generate many trajectories (paths) of the possible evolutions of the basic quantities (stocks, interest rates, exchange rates,...), also considering their relationships (dependence), defining a sufficiently large number of scenarios.
- Use the results to compute statistics of a portfolio, such as risk measures, or to evaluate products.

What's up next?

- In the rest of our course we will:
  1. (Briefly) recall the principles on which MC is based on;
  2. (Briefly) recall how to generate random variables;
  3. Learn how to generate trajectories of different processes;
  4. Learn how to use MC to price derivatives and estimate risk measures using variance reduction techniques.

Section

1 Introduction

2 Law of Large Numbers (LLN) and Central Limit Theorem (CLT)

3 Generating Random Variables

4 Generating Sample Paths

5 Pricing Derivatives via Monte Carlo simulation

6 Variance reduction techniques

7 Lattices and binomial pricing

Monte Carlo Estimation

- Let us consider the usual $(\Omega, \mathcal{F}, \mathbb{P})$ probability space.
- X is a random variable with a certain distribution defined by

$$P_X(X \leq x) := P(\omega \in \Omega : X(\omega) \leq x)$$

- Suppose we want to estimate $\theta = \mathbb{E}[X]$.

- The sample mean of this variable is the average

$$\bar{\theta}_n = \frac{\sum_{i=1}^{n} X^{(i)}}{n}$$

  where $X = (X^{(1)}, X^{(2)}, ...X^{(n)})$ is a vector of independently and identically distributed (i.i.d.) components whose distribution is P_X.
- If $x = (x_1, ..., x_n)$ is a sample (a random draw) of this vector, so that $x_i = X(\omega), \omega \in \Omega$, then $\bar{\theta}_n$ can be taken as an approximation of θ.

- For sufficiently large n, $\bar{\theta}_n$ converges to θ, because:
- $\bar{\theta}_n$ has mean θ and variance $Var(X)/n$.

Law of Large Numbers

If $\{X_i\}_{i>0}$ is a set of i.i.d. random variables with mean μ and

$$Y(n) = \frac{1}{n}\sum_{i=1}^{n} X_i,$$

then

$$\lim_{n\to+\infty} P(|Y(n) - \mu| \geq \epsilon) = 0 \ \forall \epsilon > 0.$$

How good is the approximation?

- Recall the Central Limit Theorem:

Central Limit Theorem

Let $\{X_i\}_{i>0}$ be a set of i.i.d. random variables with mean μ and variance σ^2. Then

$$\frac{\sum_{i=1}^{n} X_i - n\mu}{\sigma\sqrt{n}} \rightarrow^d N(0,1) \text{ as } n \rightarrow +\infty$$

- As a consequence, we have that

$$\frac{\bar{\theta}_n - \theta}{\frac{\sigma}{\sqrt{n}}} \rightarrow^d N(0,1) \text{ as } n \rightarrow +\infty$$

and thus that the error in approximating $\mathbb{E}[X]$ via the estimator $\bar{\theta}$ is approximately normally distributed with mean 0 and standard deviation $\frac{\sigma}{\sqrt{n}}$:

$$\bar{\theta}_n - \theta \rightarrow^d N(0, \frac{\sigma^2}{n}) \text{ as } n \rightarrow +\infty.$$

- σ is usually unknown. Replacing it with its unbiased estimator

$$\bar{\sigma} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(X_i - \bar{\theta}_n\right)^2},$$

  we can state that the standard error is approximately $\bar{\sigma}/\sqrt{n}$, that is equivalent to say that to reduce it by a half, n has to be increased by a factor of 4.
- Monte Carlo converges at a rate $O(n^{-1/2})$.

Section

1 Introduction

2 Law of Large Numbers (LLN) and Central Limit Theorem (CLT)

3 Generating Random Variables

4 Generating Sample Paths

5 Pricing Derivatives via Monte Carlo simulation

6 Variance reduction techniques

7 Lattices and binomial pricing

Generating RVs in Matlab

- MATLAB has many built-in functions that allow to draw random numbers from different distributions.
- We briefly recall now how well-known methodologies can be used to generate random variables from the uniform distribution.
- The *rand* command in MATLAB generates pseudo-random samples from the U[0,1] distribution. The sequence is determined by the state of the generator, which is reset every time at start up.
- In order to reset the state to a different value each time we can use the clock command:
rand('state',sum(10*clock)).

The inverse transform method

- Suppose we know the cumulative distribution function of a variable X, called F, and that we want to sample from that distribution using *rand*.
- We can make use of a transformation of the samples from the uniform G(U) such that the inverse of G matches F.
- If G is bijective and monotonically increasing, its inverse function $F^{-1}(x)$ is well defined and we can set $X = F^{-1}(U)$. Indeed:

$$P(G(U) \leq x) = P(U \leq G^{-1}(x)) = P(U \leq F(x)) = F(x).$$

How to apply the method

Algorithm

1. Generate u from the uniform distribution between 0 and 1;
2. using the inverse cdf, compute the value of x s.t. F(x)=u;
3. Consider x as a random number drawn from the distribution described by F.

- Remark: one can make use of the MATLAB function *icdf*!

Example

- The exponential distribution with mean θ has distribution

$$F(x) = 1 - e^{-\frac{x}{\theta}}, x \geq 0.$$

- Making use of the inverse cdf, we can state that:

$$X = -\theta \ln(1 - U) = -\theta \ln U,$$

where U is uniformly distributed between 0 and 1 and where we used the fact that U and 1-U are identically distributed.

MATLAB code

U=rand(1000,1);
X=$-\theta log(U)$;
hist(X)

Acceptance-Rejection Method

- This method samples from a distribution by first sampling from a more convenient one, and then rejecting a subset of candidates according to a random test.
- The rejection mechanism is designed so that the accepted values are distributed according to the target distribution.
- Let g be a density from which we know how to sample from, with

$$f(x) \leq cg(x) \text{ for all } x \in \mathcal{X},$$

where f is our target distribution and $\mathcal{X}$ is a subset of the real line, of $\mathbb{R}^d$ or even more general sets.

How to apply the method

- Then:

Algorithm

1. Generate X from distribution g;
2. Generate U from the Uniform[0,1] distribution;
3. if $U \leq f(X)/cg(X)$
   accept X
   else
   go to step 1.

Section

1 Introduction

2 Law of Large Numbers (LLN) and Central Limit Theorem (CLT)

3 Generating Random Variables

4 Generating Sample Paths

5 Pricing Derivatives via Monte Carlo simulation

6 Variance reduction techniques

7 Lattices and binomial pricing

Continuous diffusions

- Now that we know how to generate random variables, let us focus on how to generate paths from continuous purely diffusive processes.
- We will consider sampling from solutions to Stochastic Differential Equations:

$$dX(t) = \mu(t, X)dt + \sigma(t, X)dW(t), X(t_0) = X_0$$

- First of all, generating sample paths amounts to *discretizing* the process, i.e. approximating it by considering its realizations at a finite set of time points $[t_0, t_1, ..., t_N]$ and then interpolate to produce a continuous-time trajectory.
- We will usually split the sampling interval into equally spaced sub intervals.

Methods for generating diffusions

- Samples from continuous diffusions can be generated using:
  1. the exact transition density between two consecutive time points $p(t,x;s,y)$.
  2. the closed-form solution of the SDE, i.e. the exact dynamics followed by the process;
  3. a dynamics which approximates the original SDE.

1-Exact Transition density

- If the transition density

$$p(t_i, x_i; t_{i+1}, x_{i+1}) = P(X(t_{i+1}) = x_{i+1} | X(t_i) = x_i)$$

is known for any pair of consecutive times, then:

Algorithm

1. Initialise $X_0 = x_0$ and $\Delta t = T/N$, where [0,T] is our sampling interval and N is the number of time steps.
2. For i=1,...N, sample X_i, from the density $p(t_i, \cdot; t_{i-1}, X_{i-1})$;
3. $\{X_i\}_{i=0,\ldots,N}$ is a sample of process X on [0,T].

Example 1: Standard Brownian Motion

- Consider the simple process $dX(t) = dW(t)$. The standard brownian motion W(t) is a continuous stochastic process s.t.:
  1. W(0)=0;
  2. The increments W(t)-W(s) are independent;
  3. $W(t) - W(s)$ is distributed as a $N(0, t-s)$ for any $0 \leq s < t \leq T$.
- Subsequent values of the process W(t) can be generated easily, as we know the transition density between two consecutive time points:

$$W(t_i) = W(t_{i-1}) + \sqrt{t_i - t_{i-1}} Z_i, i = 1, ...N,$$

where $Z_1, ..., Z_n$ are independent standard normal random variables.

MATLAB code

```
function W=b_m_sim(T,N)
dt=T/N;
W=zeros(N+1,1);
W(1)=0;
Z=icdf('norm',rand(N,1));
for i=1:N
      W(i+1)= W(i)+sqrt{dt}*Z(i);
end
t=[0:dt,T];
plot(t,W,'*-')
end
```

A more interesting example: the Vasicek model

- The Vasicek (1977) model is a well-known interest rate model for the short rate:

$$dr(t) = a\left(b - r(t)\right)dt + \sigma dW(t), r(0) = r_0.$$

- The Vasicek model falls into the domain of Gaussian models for the short rate and its transition density $p(t, \cdot; s, y)$ is Normally distributed with mean $b + e^{-a(t-s)}(y - b)$ and variance $\frac{\sigma^2}{2a}\left(1 - e^{-2a(t-s)}\right)$.

MATLAB Simulation of Vasicek's model

```
function X=Vasicek_sim(a,b,s,r0,T,N)
X=zeros(N+1,1);
X(1)=r0;
dt=T/N;
for i=1:N
    mu=b+exp(-a*dt)*(X(i)-b);
    var=s^2*(1-exp(-2*a*dt))/(2*a);
    X(i+1)=icdf('norm',rand,mu,sqrt(var));
end
t=[0:dt:T];
plot(t,X,'*')
end
```

2-Exact Solution

- This method can be applied if the SDE can be solved explicitly, i.e. if there exists a functional of time t and the driving noise W up to t such that

$$X(t) = G(t, \{W_i\}_{i=0,...t})$$

- It consists in discretizing the underlying noise over a finite set of sampling times and apply the functional to obtain the value of the process X at those set of time points.

Algorithm

1. Set $X_0 = x_0$ and $\Delta t = T/N$;
2. For i=1,...,N, sample the brownian motion $W(t_i)$ and set

$$X_i = G(t_i, \{W(t_1), ..., W(t_i)\})$$

3. $(X_i)_{i=1,...,N}$ is a sample of the process X on [0,T].

Example: Geometric Brownian Motion

- The Geometric Brownian Motion is the most widely used model for stocks. It implies log-normal prices.

$$dS(t) = S(t)\left(rdt + \sigma dW(t)\right)$$

- The solution of this SDE is

$$S(t) = S_0 \exp\left(\left(r - \frac{\sigma^2}{2}\right)t + \sigma W(t)\right).$$

- If we discretize it, we get:

$$S_i = S_{i-1} \exp\left(\left(r - \frac{\sigma^2}{2}\right)\Delta t + \sigma\sqrt{\Delta t} Z_i\right), i = 1, ..., N.$$

where Z_is are independent samples from a standard normal random variable.

MATLAB Simulation of GBM

```
function S=GBM(r,s,S0,T,N)
S=zeros(N+1,1);
S(1)=S0;
for i=1:N
    S(i+1)=S(i)*...
    exp((r-s^2/2)*dt+s*sqrt(dt)*randn);
end
t=[0:dt:T];
plot(t,S,'*-')
```

3-Approximating the dynamics

- When the two previoulsy described methods are not applicable, it is possible to simulate the solution approximating the dynamics of the system, i.e. by solving the stochastic difference equation associated to the SDE.
- There are several ways of approximating, we will see the most common one, the Euler scheme.

The Euler scheme

- The SDE

$$dX(t) = \mu(t, X)dt + \sigma(t, X)dW(t), X(t_0) = X_0$$

can be discretized in the following way:

$$X_{i+1} = X_i + \mu(t_i, X_i)\Delta t + \sigma(t_i, X_i)\sqrt{\Delta t}Z_i,$$

where $\{Z_i\}$'s are i.i.d. standard normals.

Algorithm

1. Set $X_0 = x_0$ and $\Delta t = T/N$
2. For $i = 0, ..., N-1$, sample Z_i and compute X_{i+1}
3. X_i obtained this way is a sample of process X over [0,T]

Approximation Error

- Being based on an approximation, the use of the Euler scheme entails an error.
- Let X_{EU} be the approximate solution computed based on the Euler Scheme and X the exact one. Then

$$\mathbb{E}\left[sup_{t\in[0,T]}|X_{EU}(t) - X(t)|^2\right] \leq C\Delta t,$$

  with C constant.
- The approximation error is smaller the smaller Δt.

Example: Cox, Ingersoll, Ross (1985) process

- The CIR process is used in the domain of interest rates. It is a square root process, which may never become negative:

$$dr(t) = a(b - X(t))dt + \sigma\sqrt{r(t)}dW(t).$$

- We can discretize it using the Euler Scheme as:

$$r_{t+1} = r_t + a(b - r_t)\Delta t + \sigma\sqrt{r_t}\sqrt{\Delta t}Z_i,$$

where $\{Z_i\}$s are independent random samples from a standard normal distribution.

MATLAB Simulation of CIR process using Euler Scheme

```
function X=CIR(a,b,s,r0,T,N)

X=zeros(N+1,1);
X(1)=r0;
dt=T/N;
for i=1:N
    X(i+1)=X(i)+a*(b-X(i))*dt+s*sqrt(X(i))*sqrt(dt)*randn;
end
t=[0:dt:T];
plot(t,X,'*-')
end
```

Transition density for the CIR process

- The CIR process has a known transition law:

$$r(t) = \frac{\sigma^2(1 - e^{-a(t-s)})}{4a} * \chi^2_d \left(\frac{4ae^{-a(t-s)}}{\sigma^2(1 - e^{-a(t-s)})} r(s) \right), t > s,$$

  where χ^2_d denotes the pdf of the non-central chi-squared distribution with $d = \frac{4ab}{\sigma^2}$ degrees of freedom.

Homework: Try to simulate the CIR process using its exact transition and compare the results with those obtained by using the Euler Scheme for different lengths of the time step dt.

Generating paths from jump processes

- A jump process varies according to discontinuities only.
- We consider a stochastic process

$$J(t) = \sum_{j=1}^{N(t)} Y_j,$$

where

- N represents the occurrence of jumps: for an event ω, for instance a jump trajectory, $N(t, \omega)$ counts the number of jumps between the initial time and current time t;
- Y_j represents the magnitude of the j-th jump.

Generating N and Y

- In order to generate a path from a jump process it is necessary to simulate both N and Y.
- N(t) is a counting process, i.e. a non-decreasing process that takes values in $\mathbb{N}$ and that defines the number of jumps occurred up to time t.
- Y is generated by sampling from its assigned distribution.
- N and Y are independent.

Simulating a counting process

1. Generate the inter-arrival times, i.e. the time spans between two consecutive jumps. They are random variables $T_1, ...T_N$
2. Jump times, $\tau_1, ...$ i.e. the instants at which jumps occur, are obtained as the cumulative sums of interarrival times:

$$\tau_j = \sum_{i=1}^{j} T_i$$

3. For each time instant t, $N(t)$ counts the number of jumps that have occurred since the beginning and prior to t:

$$N(t) = \sum_{n=1}^{+\infty} 1_{\tau_n \leq t}.$$

Example: Poisson process

- A homogeneous Poisson process is a jump process whose inter-arrival times are i.i.d. exponential.
- Recall the density of the exponential distribution

$$f(x) \sim Exp(\lambda) = \lambda e^{-\lambda x}$$

- N(t) is distributed according to a Poisson law $Po(\lambda t)$:

$$f_{N(t)}(n) = e^{-\lambda t} (\lambda t)^n / n!$$

- N(t) can be simulated via conditional simulation or via countdown simulation.

Simulating the counting process

Algorithm - Conditional simulation

The algorithm samples the number of jump occurrences first, and then their location. Conditional on $N(t) = n$ jump times of a homogeneous Poisson process are uniformly distributed on $[0, T]$.

1. Simulate $N(T) \sim Po(\lambda T)$.
2. Simulate the N(T) jump times $\tau_1, ..., \tau_{N(T)}$ as uniform samples on [0,T].
3. N(T) and jump times are a sample form the Poisson process.

MATLAB code

```
function [NT,tau]=sim_Pp_cond(lambda,T)
NT=icdf('Poisson',rand,lambda*T);
% alternatively, use NT=poissrnd(lambda*T);
tau=rand(NT,1).*T;
tau=sort(tau);
end
```

Countdown simulation

Algorithm

λ being constant, we know that N is a Poisson process and inter arrival times are exponentially distributed with parameter λ.

1. Let $\tau_0 = 0, i = 0$
2. Set $i = i + 1$
3. Generate inter-arrival times from the exponential, for instance using the inverse transform method: generating $U \sim U[0,1]$ and then $T_i = -\frac{1}{\lambda} lnU$.
4. Set $\tau_i = \tau_{i-1} + T_i$
5. If $\tau_i \leq T$, go to Step 2, otherwise return $\tau_1, ..., \tau_{i-1}$ and the Poisson sample $N(T) = i - 1$.

MATLAB code

```
function [NT,tau]=sim_Pp_count(lambda,T)
tau(1)=0;
i=0;
while (tau(i+1)<=T)
i=i+1;
tau(i+1)=tau(i)-log(rand)/lambda;
% alternatively, tau(i+1)=tau(i)+exprnd(lambda);
end
tau=tau(2:end);
NT=i-1;
end
```

Inhomogenous Poisson Process

- In many applications, for instance in life insurance, it is necessary to make use of a Poisson process with non-constant intensity.
- The previous algorithms can be easily adapted.

Algorithm: conditional simulation

1. Simulate $N(T) \sim Po\left(\int_0^T \lambda(s)ds\right)$
2. Generate $N(T)$ independent inter-arrival times samples with common distribution density

$$f_\tau(t) = \frac{\lambda(t)}{\int_0^T \lambda(s)ds}$$

Value-at-risk and Loss probability

- The Value at Risk (VaR) of a portfolio is a percentile of its loss distribution over an horizon Δt. It is evaluated at a confidence level ϵ, which defines the percentile of interest.
- We define the loss L over an interval Δt as $-\Delta V$, where ΔV is the change in the value of the portfolio with assets valued S:

$$L = V(S, t) - V(S + \Delta S, t + \Delta t).$$

- We define as $F_L(x) = P(L < x)$ the distribution of L.
- The VaR_ϵ is the point x_ϵ that satisfies

$$1 - F_L(x_\epsilon) = P(L > x_\epsilon) = \epsilon$$

Simulating the distribution of losses

- Estimating the distribution of losses can be achieved by generating a large number of independent replications (say n) of the following algorithm:
  1. Generate the trajectories of assets and in particular their values at $t + \Delta t$, by simulating either $S(t + \Delta t)$ or $\Delta S = S(t + \Delta t) - S(t)$;
  2. Evaluate the portfolio at $\Delta t, V(S(t + \Delta t), t + \Delta t)$ and the loss: $V(S(t), t) - V(S(t + \Delta t), t + \Delta t)$
  3. Estimate $P(L > x)$ by

$$\bar{F}_{L,n}(x) = \frac{1}{n} \sum_{i=1}^{n} 1_{L_i > x},$$

  where L_i is the loss computed in the i-th replication.

Estimating VaR

- Given the empirical distribution of portfolio losses $F_{L,n}(x)$ it is easy to obtain a simple estimate of the VaR as the empirical quantile:

$$x_\epsilon = F_{L,n}^{-1}(1-\epsilon)$$

- In order to find x_ϵ numerically:
  1. Compute the empirical distribution of L $F_{L,n}(x_j), j = 1, ..., M$.
  2. Derive a continuous-time curve $\bar{F}_{L,n}$ by interpolating $F_{L,n}$
  3. Solve the equation $\bar{F}_{L,n}(x_\epsilon) = 1 - \epsilon$.

A very simple example

```
% Consider a portfolio made by
three independent assets whose dynamics
follow a GBM.

r=0.02;
S0=[12.5;18;9];
sigma=[0.2;0.3;0.08];
n=[4 8 -3];
T=1;
N_Sim=100000;
```

Derive the empirical cdf of the Loss of the portfolio and its 99% VaR.

A very simple example/2

```
% Monte Carlo Simulation of asset prices

mu=(r-sigma.^2/2).*T;
s=sigma.*sqrt(T);
L=zeros(N_Sim,1);
for i=1:N_Sim
U=randn(3,1);
ST=S0.*exp(mu+s.*U);
L(i)=n*(S0-ST);
end
```

A very simple example/3

```
% Deriving cdf of probability of loss
and computing 99% VaR

eps=0.01;
x=[-100:0.01:150];
for i=1:length(x)
F(i)=sum(L<=x(i))/N_Sim;
end

F_hat=@(xeps) interp1(x,F,xeps)
VaR=fsolve(@(xeps) F_hat(xeps)-(1-eps),20);
```

Section

1 Introduction

2 Law of Large Numbers (LLN) and Central Limit Theorem (CLT)

3 Generating Random Variables

4 Generating Sample Paths

5 Pricing Derivatives via Monte Carlo simulation

6 Variance reduction techniques

7 Lattices and binomial pricing

A first example

- Let us consider the pricing of a European call option in a Black and Scholes setting
- The option payoff is $(S(T) - K)^+$, where S(T) is the time T (maturity of the option) price of the stock and K is the strike price, while S follows the SDE

$$\frac{dS(t)}{S(t)} = rdt + \sigma dW(t), S(0) = S_0$$

under the so-called risk-neutral measure.

- We know that the price of the option is the expected discounted value of its terminal payoff:

$$C = \mathbb{E}\left[e^{-rT}\left(S(T) - K\right)^+\right]$$

Black and Scholes formula

- We know that the price of the European option is known in closed form in this setting:

$$\begin{aligned} C(S,\sigma,T,r,K) &= SN(d_1) - e^{-rT}KN(d_2), \\ d_1 &= \frac{log(S/K) + \left(r + 1/2 * \sigma^2\right)T}{\sigma\sqrt{T}}; \\ d_2 &= \frac{log(S/K) + \left(r - 1/2 * \sigma^2\right)T}{\sigma\sqrt{T}}, \end{aligned}$$

  where $N(\cdot)$ is the cdf of the standard normal distribution.

- Anyway, the fact that derivatives prices (which are solution to PDEs) can be written as expectations (via the Feynman-Kac theorem) allows us to state a link between derivatives pricing and Monte Carlo simulation.

Algorithm

- Indeed, $\mathbb{E}\left[e^{-rT}\left(S(T)-K\right)^{+}\right]$ can be estimated by the following algorithm:
  1. For i=1,...n generate asset prices at T, i.e. generate $Z_i \sim N(0,1)$
  2. Compute $S_i(T) = S_0 exp\left(\left(r-\frac{1}{2}\sigma^2\right)T+\sigma\sqrt{T}Z_i\right)$
  3. Set $C_i = e^{-rT}\left(S_i(T)-K\right)^{+}$
  4. Compute $\bar{C}_n = \frac{C_1+\ldots+C_n}{n}$.
- We know that by LLN and CLT $\bar{C}_n$ is an unbiased estimator of the price C of the Call and that as $n \to \infty$ $\bar{C}_n \to C$ with probability 1.

Confidence intervals

- Going back to our initial example and estimator $\bar{\theta}$ we know that by the CLT:

$$\mathbb{P}\left(\frac{\bar{\theta}_n - \theta}{\sigma/\sqrt{n}} < x\right) \to \Phi(x).$$

- Thus, as $n \to \infty$ the probability that the true value of θ falls into the interval

$$\left(\bar{\theta}_n - a\frac{\bar{\sigma}}{\sqrt{n}}, \bar{\theta}_n + b\frac{\bar{\sigma}}{\sqrt{n}}\right)$$

goes to $\Phi(b) - \Phi(-a)$ as n goes to infinity.

Confidence intervals/2

- We can then say that if $1-d$ denotes the quantile of the standard normal distribution, i.e. $\Phi(z_d) = 1-d$:

$$\bar{\theta}_n \pm z_{d/2}\frac{\bar{\sigma}}{\sqrt{n}}$$

  is a $1-d$ confidence interval for θ as $n \to \infty$.
- For instance, to construct a 95% confidence interval, one should take $d = 0.05$, i.e. $z_{d/2} = 1.96$.

CPU time, number of replications and approximation error

- Let us now write a simple piece of code that computes the exact price and Monte Carlo price of a European call, its variance and the CPU time with different number of simulations.

MATLAB code

```
function [C C_MC S_MC time]=Eur_C(r,sigma,S0,K,T,N_Sim)

% This function computes the exact and MC price
% of a European Call in a BS setting

d1=(log(S0/K)+(r+1/2*sigma^2)*T)/(sigma*sqrt(T));
d2=(log(S0/K)+(r-1/2*sigma^2)*T)/(sigma*sqrt(T));
C=S0*normcdf(d1)-exp(-r*T)*K*normcdf(d2);
c=cputime;
mu=(r-1/2*sigma^2)*T;
s=sigma*sqrt(T);
```

MATLAB code/2

```
% For loop to generate S(T) and compute price

for i=1:N_Sim
ST(i)=S0.*exp(mu+s*randn);
C(i)=exp(-r*T)*max(0,ST(i)-K);
end
    C_MC=mean(C);
S_MC=std(C);

time=cputime-c;
end
```

Some refinements of the previous code and computational time

Consider the previous code and compute the computational time needed to run it with 100 simulations. Then recompute it after refining the code:

1. Pre-allocate ST and C;
2. Avoid the for loop.

Section

1 Introduction

2 Law of Large Numbers (LLN) and Central Limit Theorem (CLT)

3 Generating Random Variables

4 Generating Sample Paths

5 Pricing Derivatives via Monte Carlo simulation

6 Variance reduction techniques

7 Lattices and binomial pricing

Improving the accuracy of MC estimates

- The confidence interval associated to our MC estimate is

$$\bar{\theta}_n \pm z_{d/2} \frac{\bar{\sigma}}{\sqrt{n}}.$$

- It can thus be reduced either by increasing the number of simulations n or by reducing the variance.
- This second way entails the use of so-called variance reduction techniques, which, given the number of iterations, increase the precision of the estimate. We will see two methods:
  1. Antithetic variables;
  2. Control variates.

Antithetic variables

- The goal is to estimate $\theta = \mathbb{E}[g(X)]$, where g is a monotone function of the r.v. X.
- The sample mean is $\bar{\theta}_n(X) := \frac{1}{n}\sum_{i=1}^{n} g(X_i)$, which is an unbiased estimator whose precision can be measured by the unbiased estimator of the variance:

$$\hat{\sigma^2}(g(X)) = \frac{1}{n-1}\sum_{i=1}^{n}\left(g(X_i) - \bar{\theta}_n(X)\right)^2.$$

Antithetic variables/2

- If X^1 and X^2 are samples from the common cdf F then

$$Var\left(\frac{g(X^1)+g(X^2)}{2}\right) = \frac{1}{2}\left[Var(g(X^1)) + Cov(g(X^1), g(X^2))\right]$$

  since $Var(g(X^1)) = Var(g(X^2))$.
- If the samples are independent, then the variance is $\mathrm{Var}(\bar{\theta})$, while if X^1 and X^2 are negatively correlated, then

$$Var\left(\frac{g(X^1)+g(X^2)}{2}\right) \leq \frac{1}{2}\left[Var(g(X^1))\right].$$

Antithetic variables/3

- Making use of the inverse transform method that generates samples from F, we can obtain negatively correlated random samples X^1 and X^2 by generating negatively correlated uniform r.v.s U_1 and U_2 so that $F^{-1}(U_1)$ and $F^{-1}(U_2)$.
- Given a uniform sample U^1, we can introduce negative correlation by defining $U^2 = 1 - U^1$
- Given a standard normal sample U^1, we can introduce negative correlation by taking $U^2 = -U^1$.

Antithetic variables - Algorithm

Algorithm

1. Generate $U_1, ..., U_n$ i.i.d. uniform random variables.
2. Estimate

$$\hat{\theta_{AV}} = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} \left[g(F^{-1}(U_i) + g(F^{-1}(1 - U_i)) \right]$$

In general, an antithetic variable estimator is

$$\hat{\theta}_{AV} = \frac{g(X) + g(Y)}{2},$$

where Y is a random path identically distributed with respect to X and which displays negative covariance with X.

European Call pricing with antithetic variables

```
%%
mu=(r-sigma^2/2)*T;
s=sigma*sqrt(T);
X=randn(N_Sim,1);
Y=-X;

ST1=S0.*exp(mu+s.*X);
ST2=S0.*exp(mu+s.*Y);

C=exp(-r*T).*(max(0,ST1-K)+max(0,ST2-K))/2;
  C_MC=mean(C);
S_MC=std(C);
```

Control Variates

- Most effective and broadly applicable technique for improving MC efficiency.
- Let $Y_1, ..., Y_n$ be the outputs from n replications of a simulation, e.g. prices of a derivative for each simulation.
- Our objective is to estimate $\mathbb{E}[Y_i]$.
- We make use of another output X_i along with Y_i, computed at each replication and whose expectation $\mathbb{E}[X]$ is known in order to reduce the variance of our estimate.

Control Variates/2

- Consider the i.i.d. pairs (X_i, Y_i). For any fixed b we can compute

$$Y_i(b) = Y_i - b\left(X_i - \mathbb{E}[X]\right),$$

whose sample mean is

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

- The last expression defines the control variate estimator, as the observed error $\bar{X} - \mathbb{E}[X]$ serves as a control in estimating $\mathbb{E}[Y]$.
- The control variate estimator is unbiased ($\mathbb{E}[\bar{Y}(b)] = \mathbb{E}[Y]$) and consistent ($\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} Y_i(b) = \mathbb{E}[Y]$ with probability 1).

Control Variates/Variance reduction

- The variance of each $Y_i(b)$ is

$$Var(Y_i(b)) = \sigma_Y^2 - 2b\sigma_X\sigma_Y\rho_{XY} + b^2\sigma_X^2.$$

- The variance of the control variate estimator is thus $Var(Y_i(b))/n$ while the variance of the ordinary sample average $\bar{Y}$ is σ_Y^2/n.
- Since b is some constant which we can pick, we choose its optimal value b^* that minimizes the variance of $Var(Y_i(b))$:

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

- The ratio between the variances of the control variate estimator and of the average sample standard estimate is

$$\frac{Var[\bar{Y}_i(b)]}{Var[\bar{Y}]} = 1 - \rho_{XY}^2 \leq 1.$$

Control Variates/Remarks

- As in principle $Cov(X, Y)$ and $Var(X)$ may be unkown, we substitute b by its population estimate:

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

- Notice that covariance and variance of X might themselves be estimated using MC simulation!
- The variance reduction factor, $\frac{1}{1-\rho_{XY}^2}$, increases sharply as $|\rho_{XY}|$ approaches 1: a high correlation is needed to provide substantial computational benefits.

Control Variates - An example

- The most universal example of a control variate is the discounted value of the underlying asset of a derivative, which can be used being a martingale under the risk-neutral measure.
- Suppose Y_i is the discounted payoff of a derivative written on asset S for the independent replications $i = 1, ...n$. We know that $e^{-rt}S(t)$ is a martingale, and thus $\mathbb{E}[e^{-rt}S(t)] = S(0)$. We can then use the control variate estimator

$$\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - b[S_i(T) - e^{rT}S(0)]\right).$$

- Notice that the effectiveness of the control variate depends on the correlation between $S(T)$ and Y, which in the case of European options may depend on the strike.

Section

1 Introduction

2 Law of Large Numbers (LLN) and Central Limit Theorem (CLT)

3 Generating Random Variables

4 Generating Sample Paths

5 Pricing Derivatives via Monte Carlo simulation

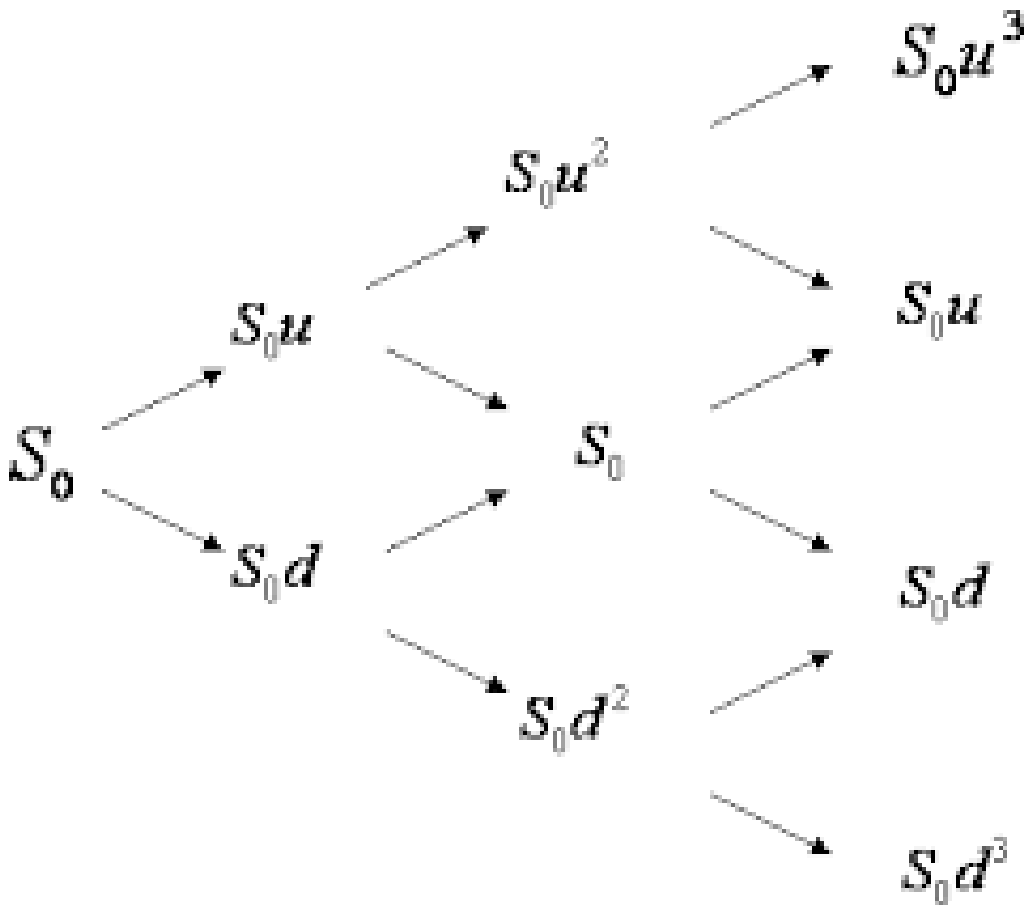
6 Variance reduction techniques

7 Lattices and binomial pricing

Lattices

- Lattices provide a discrete-time, discrete-space approximation to the evolution of a diffusion process.
- Each node in the tree is associated to a value of the asset, which is linked for each time step with successive nodes, i.e. with a set of possible future values. In the case of a binomial tree, these possible future values correspond to an up and a down move of the price.
- By varying the distance between the nodes in terms of space and time it is possible to vary the conditional mean and variance of the change in the underlying, approximating virtually any diffusion process.
- It is useful to price derivatives by backward induction, computing the payoffs at terminal nodes and then going back.

The binomial tree



Pricing derivatives in the binomial tree

- Option valuation using the binomial tree requires

Algorithm

1. generation of the prices in the tree;
2. calculation of the option value at each terminal node;
3. calculation of the option value at each previous node until the first node, which defines the price of the option.

Asset price generation

- Asset prices are generated at each node in the tree, assuming that the underlying moves up or down by a specific factor u or d with a certain probability p and $1-p$.
- Cox, Ross and Rubinstein (1979), the "fathers" of binomial pricing, chose u and d as

$$u = e^{\sigma\sqrt{\Delta t}}, d = 1/u$$

where Δt is the time step and σ is the volatility of the underlying.

- This choice ensures that the tree is recombinant, thus reducing the number of nodes, and that for each node

$$S_{nj} = S(0) * u^{Nu} d^{Nd} = S(0) * u^{Nu-Nd},$$

where Nu and Nd denote the number of up and down moves respectively.

Calculation of option value at final nodes

- At each last node, which corresponds to the maturity of the option, the payoff of the option, i.e. its exercise value, must be calculated.
- For instance, for a European call, the value of the option at each terminal node n, j will be

$$\max(S_{n,j} - K, 0),$$

where K is the strike price and $S_{n,j}$ denotes the asset price at the j-th final node.

Calculation of option value at each node

- For each node, going backwards from the next to last to the first, we compute the risk-neutral value of the option.
- Following the risk-neutral approach, the fair price of a derivative is the expected discounted value of its future payoff.
- At each node, the expected value is computed by averaging the values of the two following nodes weigthed by their probabilities and discounting them at rate r.

Cox, Ross, Rubinstein and risk-neutrality

- Risk-neutrality in the CRR framework requires

$$q = \frac{e^{r\Delta t} - d}{u - d},$$

and the value of a derivative at each node becomes

$$f_j = e^{-r\Delta t} \left(q f_{u,j+1} + (1 - q) * f_{d,j+1} \right),$$

where f_u and f_d define the price of the derivative in case of a successive upward or downward move respectively.

An example: European Call option

```
function C=Eur_call(S0,K,r,T,sigma,N)
dt=T/N;u=exp(sigma*sqrt(dt));d=1/u;
q=(exp(r*dt)-d)/(u-d);
M=zeros(N+1,N+1); % pre-allocating the tree grid
% computing exercise values at last node
for i=1:N+1
M(i,N+1)=max(S0*u^(N+1-i)*d^(i-1)-K,0);
end
% Compute the intrinsic value at each node
for j=N:-1:1
for i=1:j
    M(i,j)=exp(-r*dt)*(q*M(i,j+1)+(1-q)*M(i+1,j+1));
end
end
price=M(1,1);
```

Useful references

- P. Brandimarte, 2006, Numerical methods in finance and economics: a MATLAB-based introduction, John Wiley & Sons.
- P. Glasserman, 2003, Monte Carlo Methods in Financial Engineering, Spinger.
- D. Higham, Nine Ways to implement the binomial method for option valuation in MATLAB, SIAM Review.
- Cox, J., Ross, S., and Rubinstein, M., 1979, Option Pricing: a simplified approach, Journal of Financial Economics 7, 229-263.