Monte Carlo

Fabio Cannizzo

Risk Neutral Pricing

- In the context of trees, using a no arbitrage argument, we inferred that the price of a derivative can be obtained as the discounted **risk neutral** expectation of its payoff
- This results is a general result, not specific to the binomial model
- In fact, we implicitly used it already in the context of trinomial trees, where the replication argument is not applicable
- Before we start the discussion on Monte Carlo methods, it is useful to review this results and extend it in the context of continuous finance

Risk Neutral Pricing

- Remember our main result in the context of risk neutral binomial pricing:

$$V = \left[qV^{+} + (1-q)V^{-} \right] e^{-r\,\Delta t} \quad \text{where} \quad q = \frac{S\,e^{r\,\Delta t} - S^{-}}{S^{+} - S^{-}}$$

  - Since discounting just mean “take the present value”, this can be also be written as:

$$V_t = E\left[PV\left(V_T \right) \right]$$

  - But the PV() operator is not always that easy to handle, for example if the time of payment is uncertain.
  - Can we get a Martingale representation?

Risk Neutral Pricing

- Assuming *r* is constant, let's manipulate the expression obtained.
- Bt is the price of a money market account starting at B_t=1

$$V_t = E\left[e^{-r(T-t)} V_T\right] = E\left[\frac{V_T}{B_T}\right] = \underbrace{B_t}_{1} E\left[\frac{V_T}{B_T}\right]$$

$$\frac{V_t}{B_t} = E\left[\frac{V_T}{B_T}\right]$$

a Martingale!

- Normalizing the derivative price with respect to the price of a money market account, we obtain a martingale
- Conclusion: in the binomial economy, by no arbitrage, properly normalized prices are martingale if the expectation is taken under the risk neutral probability

Risk Neutral Pricing

- Since a non dividend paying stock is also a security, the result should still hold. Let's check that:

$$E\left[\frac{S_T}{B_T}\right] = \frac{p\,u\,S_t + (1-p)d\,S_t}{B_t e^{r(T-t)}}$$

p is risk neutral probability

$$= p\frac{(u-d)S_t}{B_t e^{r(T-t)}} + \frac{dS_t}{B_t e^{r(T-t)}}$$

$$= \frac{e^{r(T-t)} - d}{u-d}\frac{(u-d)S_t}{B_t e^{r(T-t)}} + \frac{dS_t}{B_t e^{r(T-t)}}$$

$$= \frac{S_t}{B_t}$$

and it does hold!

Risk Neutral Pricing

- Can we extend the result to continuous time?
- Let's consider a simple B&S economy:

$$dB_t = rB_t\,dt$$

$$\frac{dS_t}{S_t} = \mu\,dt + \sigma\,dW_t$$

- Let's consider the normalized stock process S/B
- There is nothing stochastic in B_t, hence by simple calculus

$$d\frac{1}{B_t} = -\frac{1}{B_t^2}\,dB_t = -\frac{r}{B_t}\,dt$$

Risk Neutral Pricing

- by product rule:

$$d\left(\frac{S_t}{B_t}\right) = dS_t\left(\frac{1}{B_t}\right) + d\left(\frac{1}{B_t}\right)S_t + \underbrace{dS_t d\left(\frac{1}{B_t}\right)}_{=0}$$

$$= \frac{S_t}{B_t}\left[\mu\, dt + \sigma\, dW_t\right] - r\frac{S_t}{B_t}dt = \frac{S_t}{B_t}\left[\left(\mu - r\right)dt + \sigma\, dW_t\right]$$

$$= \sigma\frac{S_t}{B_t}d\left[\frac{\mu - r}{\sigma}t + W_t\right] = \sigma\frac{S_t}{B_t}dW_t^Q$$

- the stock price normalized by a money market account is a martingale with respect to

$$W_t^Q = W_t + \frac{\mu - r}{\sigma}t$$

Is this still a Brownian motion?

Risk Neutral Pricing

- Girsanov Theorem: let W_t be a Brownian motion on probability measure P, then

$$W_t^Q = W_t + \int_0^t \lambda\left(u, W_u\right) du$$

or

$$dW_t^Q = dW_t + \lambda\left(t, W_t\right) dt$$

- is a new Brownian motion under a new probability measure Q

Risk Neutral Pricing

- Let's see what happens to the dynamic of the stock price under the new probability measure

$$\frac{dS_t}{S_t} = \mu\, dt + \sigma\, dW_t$$
$$= \mu\, dt + \sigma \left[dW_t^Q - \frac{\mu - r}{\sigma} dt \right]$$
$$= r\, dt + \sigma\, dW_t^Q$$

- As we found in the binomial case, we just need to replace the real world drift with the risk free rate

Risk Neutral Pricing

- Let's consider now a generic derivative V_t
- By no arbitrage its price is given by the dynamic replication strategy

$$V_t = \alpha_t B_t + \theta_t S_t$$

- Which under the risk neutral measure follows the dynamic

$$\begin{aligned} dV_t &= \alpha_t dB_t + \theta_t dS_t \\ &= \alpha_t r B_t dt + \theta_t S_t \left[r\,dt + \sigma\, dW_t^Q \right] \\ &= \left[\frac{V_t - \theta_t S_t}{B_t} \right] r B_t dt + \theta_t S_t \left[r\,dt + \sigma\, dW_t^Q \right] \\ &= r V_t dt + \sigma\, \theta_t S_t dW_t^Q \end{aligned}$$

Risk Neutral Pricing

- By product rule, the normalized derivative price

$$d\left(\frac{V_t}{B_t}\right) = V_t\, d\left(\frac{1}{B_t}\right) + \frac{1}{B_t} dV_t$$

$$= -\frac{r}{B_t} V_t\, dt + \frac{1}{B_t}\left[r V_t\, dt + \sigma \theta_t S_t\, dW_t^Q \right]$$

$$= \sigma \theta_t S_t\, dW_t^Q$$

- In the B&S economy, the normalized derivative price is a martingale under the risk neutral probability, implying:

$$\frac{V_t}{B_t} = E^Q\left[\frac{V_T}{B_T}\right]$$

Fundamental Theorem of Asset Pricing

- Let “real world” uncertainty be defined on probability measure *P,* it is possible to price by no arbitrage if and only if there exist a new probability measure *Q* such that

$$\frac{V_t}{B_t} = E^Q\left[\frac{V_T}{B_T}\right]$$

  where V_t is the price of any traded security, E^Q is the expectation under probability measure *Q*
- We do not demonstrate it here, but this result holds in more general economic environment than in the B&S economy

Fundamental Theorem of Asset Pricing

- So the pricing problem reduces to the simple computation of an expectation under the risk neutral probability

$$V_t = B_t E^Q\left[\frac{V_T}{B_T}\right]$$

- and, if interest rates are deterministic

$$V_t = B_t E^Q\left[\frac{V_T}{B_T}\right] = e^{-r(T-t)} E^Q\left[V_T\right]$$

Black & Scholes Formula Example

- The Black & Scholes formula, which is derived as solution of the famous Black & Scholes PDE, can be also derived via Martingale pricing

$$V_t = e^{-r(T-t)} E\left[\max\left(S_T - K, 0\right)\right] = e^{-r(T-t)} \int_0^\infty \left[\max\left(S_T - K, 0\right)\right] p\left(S_T\right) dS_T$$

$$= e^{-r(T-t)} \int_K^\infty \left(S_T - K\right) p\left(S_T\right) dS_T = e^{-r(T-t)} \left[\int_K^\infty S_T p\left(S_T\right) dS_T - K \int_K^\infty p\left(S_T\right) dS_T \right]$$

Black & Scholes Formula Example

$$I_1 = \int_K^{\infty} S_T \, p(S_T) \, dS_T$$

let $$\varphi = \frac{\ln \frac{S_T}{S_0} - \left(r - \frac{\sigma^2}{2} \right) T}{\sigma \sqrt{T}} - \sigma \sqrt{T} \sim N\left(-\sigma \sqrt{T}, 1 \right)$$ … change of variable

$$p(S_T) \, dS_T = \frac{e^{-\frac{1}{2}\left(\varphi + \sigma \sqrt{T}\right)^2}}{\sqrt{2\pi}} \, d\varphi$$ this is easy to verify

$$S_T = K \quad \Rightarrow \quad \varphi = -\left(\frac{\ln \frac{S_0 e^{rT}}{K}}{\sigma \sqrt{T}} + \frac{1}{2} \sigma \sqrt{T} \right) = -d_1$$ new integration bound

Black & Scholes Formula Example

$$I_1 = \int_K^{\infty} S_T\, p(S_T)\, dS_T = \int_{-d_1}^{\infty} S_0 e^{\varphi\sigma\sqrt{T} + \left(r + \frac{\sigma^2}{2}\right)T} \frac{e^{-\frac{1}{2}\left(\varphi + \sigma\sqrt{T}\right)^2}}{\sqrt{2\pi}}\, d\varphi$$

since N(a)=1-N(-a)

$$= S_0 e^{rT} \int_{-d_1}^{\infty} \frac{e^{-\frac{1}{2}\varphi^2}}{\sqrt{2\pi}}\, d\varphi = S_0 e^{rT}\left[1 - N(-d_1)\right] = S_0 e^{rT} N(d_1)$$

Black & Scholes Formula

$$I_2 = \int_K^{\infty} p(S_T)\,dS_T = P(S_T > K) = 1 - P(S_T < K)$$

$$= 1 - P(\ln S_T < \ln K)$$

$\ln S_t$ is normal

$$= 1 - P\left(\varphi = \frac{\ln\frac{S_T}{S_0} - \left(r - \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}} < \frac{\ln\frac{K}{S_0} - \left(r - \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}\right)$$

φ is N(0,1)

since N(a)=1-N(-a)

$$= 1 - N\left(\frac{\ln\frac{K}{S_0} - \left(r - \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}\right) = N\left(-\frac{\ln\frac{K}{S_0} - \left(r - \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}\right) = N(d_2)$$

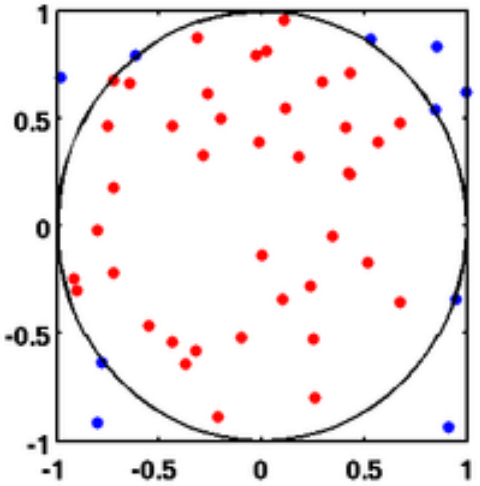
What is Monte Carlo?

- So now we know that all it takes to price a derivative is to be able to compute its expectation
- The expectation is just a Riemann integral, therefore all we have to do is to solve this integral (analytically or numerically)
- Monte Carlo offers us a possible numerical methodology to approximate the value of integral, which could be very useful in some situations, for instance if the context of high dimensionality

Monte Carlo Integration

- A simple and intuitive example of Monte Carlo integration is the estimation of the number π
- We consider a circle enscribed in the square [-1,1]x[-1,1]
- We use independent pairs of random numbers Unif(-1,1)^2 and count what proportion of them happens to be inside the circle β=NumPointsInside/NumPointsTotal
- Because the points are uniformly distributed in the square, The ratio between the area of the circle and the area of the square must match this proportion

$$\frac{AreaCircle}{AreaSquare} = \frac{\pi}{4} \approx \beta$$



Monte Carlo Integration

- Let's consider a function f(x) to be integrated in [0,1]

$$I = \int_0^1 f(x)\,dx$$

- We can multiply inside the integral the probability density function of the uniform distribution in [0,1], which is just 1 within the interval, 0 outside

$$I = \int_0^1 f(x)\cdot\left(I_{0\le x\le 1}\right)dx = E\left[f(x)\right], \qquad \text{where } x \quad \text{Unif}(0,1)$$

- and we recognize that the integral can be expressed as the expectation of the function of stochastic variable f(X), where X is a stochastic variable with uniform distribution in [0,1]

Monte Carlo Integration

- The fact that we chose a function integrated in the interval [0,1] does not cause any loss of generality, because we can always introduce a change of variable so that the integration boundaries become [0,1]

$$\int_a^b g(y)\,dy$$

$$\text{let} \quad x=\frac{y-a}{b-a} \quad \Rightarrow \quad y=a+x(b-a), \quad dy=(b-a)\,dx$$

$$\int_a^b g(y)\,dy=\int_0^1 g\big(a+x(b-a)\big)(b-a)\,dx=\int_0^1 f(x)\,dx$$

Monte Carlo Integration

- If X is a stochastic variable Unif(0,1), then also f(X) is a stochastic variable with some unknown distribution and some unknown mean *m* and variance s^2
- If we can generate some independent draws for the stochastic variable X: $\{X_1, X_2, \ldots, X_n\}$, then we can compute the corresponding values of the stochastic variable f(X)
- We could estimate the expectation of f(X) taking the sample mean estimator

Monte Carlo Integration

- Because the draws of f(X) are i.i.d. the sample estimator of the stochastic variable f(X), by central limit theorem, tend to the true mean *m* with probability 1, and its distribution tend to the normal distribution N(*m*,s^2/n)

$$\hat{m} = \frac{1}{n}\sum_{i=1}^{n} f\left(X_i\right) \quad \xrightarrow{n\to\infty} \quad m \qquad\qquad dist\left[\hat{m}\right] \quad \xrightarrow{N\to\infty} \quad N\left(m, \frac{s^2}{n}\right)$$

- The variance of the stochastic variable f(X) is unknown, but it can be estimated using the sample estimator from the same draws of f(X)

$$\hat{s}^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(f\left(X_i\right) - \hat{m}\right)^2$$

Monte Carlo Convergence

- It is easy to show that the convergence is of order $O(n^{-1/2})$ in statistical sense

$$O\left(1/\sqrt{n}\right) \quad \Rightarrow \quad \exists c: \quad |E_n| < c/\sqrt{n}, \quad \text{for } \forall n$$

for large c, this is verified in probabilistic sense. Given a confidence level ϑ

$$\hat{m}_n \sim N\left(m, \frac{s^2}{n}\right) \quad \Rightarrow \quad E_n = \hat{m}_n - m \sim N\left(0, \frac{s^2}{n}\right) \quad \Rightarrow \quad \varepsilon_n = E_n \frac{\sqrt{n}}{s} \sim N(0,1)$$

$$\vartheta < P\left(|E_n| < \frac{c}{\sqrt{n}}\right) = P\left(|\varepsilon_n| < \frac{c}{s}\right) = P\left(-\frac{c}{s} < \varepsilon_n < \frac{c}{s}\right)$$

$$\vartheta < 1 - 2N\left(-\frac{c}{s}\right) \quad \Rightarrow \quad c > -sN^{-1}\left(\frac{1-\vartheta}{2}\right)$$

P converges to 1 very fast!

c=5s ➔ ϑ = 99.9999%

Monte Carlo Convergence

- An order of convergence of $O(n^{-1/2})$ is not particularly good. It means that to reduce the error by 10 time n must increase by a factor of 100
- The method becomes quite competitive for high dimensional integrals, where the convergence remain $O(n^{-1/2})$, while for deterministic methods it deteriorates quickly
- For example, the trapezoid integration method, has an order of convergence of $O(n^{-2/d})$, where d is the dimension of the integral

Random Numbers Generation

- We still have not discussed how we can generate independent random draws of the variable X, which is Unif(0,1)
- We could use some physical experiment, e.g. flip a coin. But flipping a coin returns only a binary value (0 or 1), while we need something uniformly distributed in [0,1].
- We could flip a coin 32 times, and each time set a different bit of a 32 bit number, then divide the result by the maximum possible number achievable 2^{32}-1. We obtain something which is almost Unif(0,1)!

Random Number Generation

- This would work, but:
  - inside the computer there is no little guy very fast in flipping coin (if useful, I am sure some clever engineer could fit one)
  - truly random is not what we want, we want to be able to reproduce exactly our experiments at any time, which rules out true randomness

Pseudo Random Numbers

- Computers emulate randomness via completely deterministic algorithms. They generate random numbers which are not random at all. They just satisfy some of the properties of true random numbers, thus allowing the use of tools and laws from statistics and probability theory
- These numbers are called **pseudo random numbers** (**PRN**), and the algorithms which produce them are called **pseudo random number generators** (**PRNG**)

Pseudo Random Numbers

- PRNs should behave similarly to realizations of independent, identically distributed random variables with a certain distribution.
- PRNs are used in many fields, like finance, physics, weather forecasting, cryptography
- Every RNG has its deficiencies - no RNG is appropriate for all tasks.
- A good RNG is one that has been thoroughly analysed theoretically and is backed by convincing practical evidence, such as extensive statistical testing.
- In stochastic simulation, we need an RNG whose structure does not interfere with our simulation problem and yield inaccurate results.
- There are two main families of RNGs, **linear** generators and **nonlinear** ones (not used in finance).

PRNG

- A simple example of PRNG is a linear congruential generator
- They are based on the *remainder* operator
- Consider the trivial PRNG:

$$x_{i+1} = \left(a \cdot x_i\right) \bmod b$$

$$u_{i+1} = x_{i+1} / b$$

to rescale in (0,1)

- If a=6 and b=11, when initialized with a **seed** x0=1 we get the sequence: 1,6,3,7,9,10,5,8,4,2,1,6,…
- Note that at some point the sequence repeat itself, not good!

PRNG

- With a=3 and b=11, depending on the seed the generator can produce two different sequences
  - X0=1 ➔ 1, 3, 9, 5, 4, 1, …
  - X0=2 ➔ 2, 6, 7, 10, 8, 2, ….
- This is called a split sequence
- The maximum length which a generator can produce is called **period**.
- Based on *mod 11* the maximum theoretical period is 10, but this is achieved with a=6, but not with a=3.

PRNG

- Simply taking *b* very large does not ensure we have a long period (remember the split sequence example)
- Monte Carlo converges when $n \rightarrow \infty$, so we need a very long sequence of PRNs, and they must be i.i.d.
- After they start repeating themselves they are no longer independent!
- It is not easy to design a good PRNG. This is a field of very active research and the theory involved is complex

PRNG

- Most desirable properties of a good RNG suitable for stochastic simulation
  - Fast: invoked million of times
  - Long period: need i.i.d
  - Randomness: difficult to define, luckily there are rigorous tests written by clever people, and time is another important test, only the best survive
  - Reproducible: experiments must be repeatable
  - Portable: must provide same results on any platform

PRNG

- In Monte Carlo simulations, linear PRNGs are the most widely used ones. They can be fast and reliable. Good ones are:
  - **Mersenne Twister 19937**, by Makoto Matsumoto
    - no native skip ahead
    - period 2^{19937}
  - **MRG32K3a**, by Pierre L'Ecuyer
    - native skip ahead
    - period 2^{91}
  - Both available on the web and in many free source and vendor libraries
  - Both generate Uniform PRNs in [0,1], like most generators
- You may run into trouble with linear RNGs, because
  - they produce linear point structures in every dimension and this fact may interfere with your simulation problem (we'll see later).
  - They are predictable (cryptographers stay away from them!). Beware of RNGs provided by some commercial packages (Excel, Compilers, …). Only trust well established generators.

Call Option Example

Suppose a share follows a GBM and interest rate is deterministic

$$dS_t = S_t\left(\mu\, dt + \sigma\, dW_t\right)$$

where μ is the risk neutral drift (it could be the same as *r* or different from r, e.g.: *r-y*)

A call option has payoff

$$Payoff = \max\left(S_T - K, 0\right)$$

By fundamental theorem of asset pricing its value is

$$V_t = e^{-rT} E\left[\max\left(S_T - K, 0\right)\right]$$

The expectation can be estimated via Monte Carlo

$$E\left[\max\left(S_T - K, 0\right)\right] \approx \frac{1}{N}\sum_{i=1}^{N} \max\left(S_{T,i} - K, 0\right)$$

Where $S_{T,i}$ are i.i.d.realizations with lognormal distribution

$$S_T \sim LogN\left(\log S_0 + \left(\mu - \frac{\sigma^2}{2}\right)T, \sigma^2 T\right)$$

Call Option Example

- One difficulty here is that most PRNGs generate PRNs uniformly distributed, while we need here PRNs lognormally distributed
- First do a variable transformation so that instead of lognormal PRNs we need N(0,1) PRNs

$$\sum_{i=1}^{N} \max\left(S_{T,i} - K, 0\right) = \sum_{i=1}^{N} \max\left(S_0 e^{\left(\mu - \frac{\sigma^2}{2}\right)T + \sigma\sqrt{T}\,\varepsilon_i} - K, 0\right)$$

$$\varepsilon_i \text{ i.i.d. and } \sim N(0,1)$$

Note that $\sqrt{T}\,\varepsilon$ has the same distribution as W_T

- But we do not know how to generate N(0,1) PRNs either!

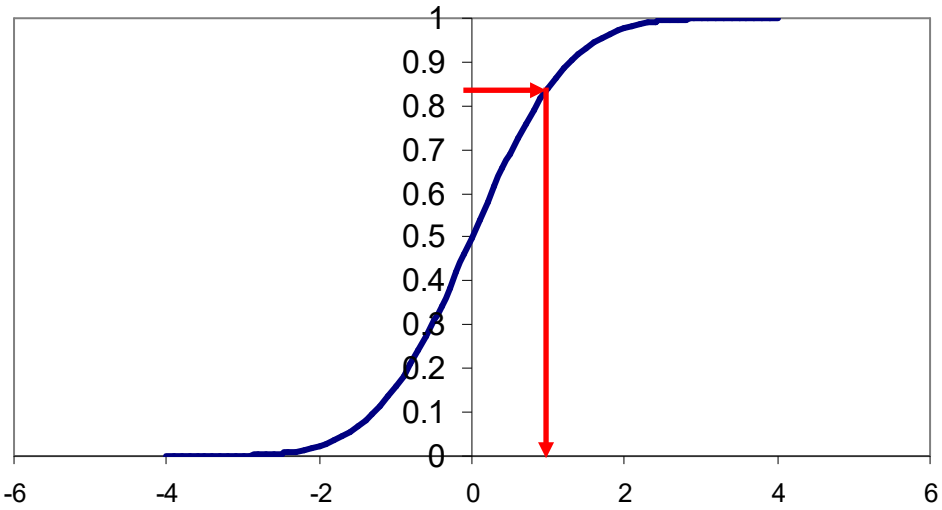
Distribution Sampling

- Sampling RNs with a certain distribution is achieved by sampling RNs with Uniform distribution and then transforming them to the desired distribution
- Most generators in fact produce **Discrete Uniform** random numbers in a quite large domain (e.g. 2^{32}), which, if needed can be easily rescaled to Discrete Uniform distribution with smaller range using the modulus operator or transformed to a **Quasi-Continuous Uniform** distribution in [0,1] by dividing for the domain size
  - X Discrete Unif [0, 2^{32}-1] ➔ Y=X/(2^{32}-1) almost Cont. Unif [0,1]
  - Sometimes we want to exclude the extremes 0 and/or 1
- Dependent on the target distribution, different methods are available

Distribution Sampling

- The **Inverse Transform Method** transforms a RN U Unif(0,1), into a RN with CDF F(X) via inversion of F(X)
  - Applicable to any distribution
  - Could be difficult or expensive to invert F(X)

$$X = F^{-1}(U), \qquad U \sim Unif(0,1)$$



Distribution Sampling

- **Acceptance Rejection** technique, due to **Von Neumann**, instead of generating RNs with the desired distribution f(x), generates RNs from a more convenient distribution g(x), but accepts them only if they fit certain criteria. The acceptance criteria is designed in such a way that the accepted numbers fit the distribution f(x)
- Used in many algorithms which convert Unif(0,1) into N(0,1)

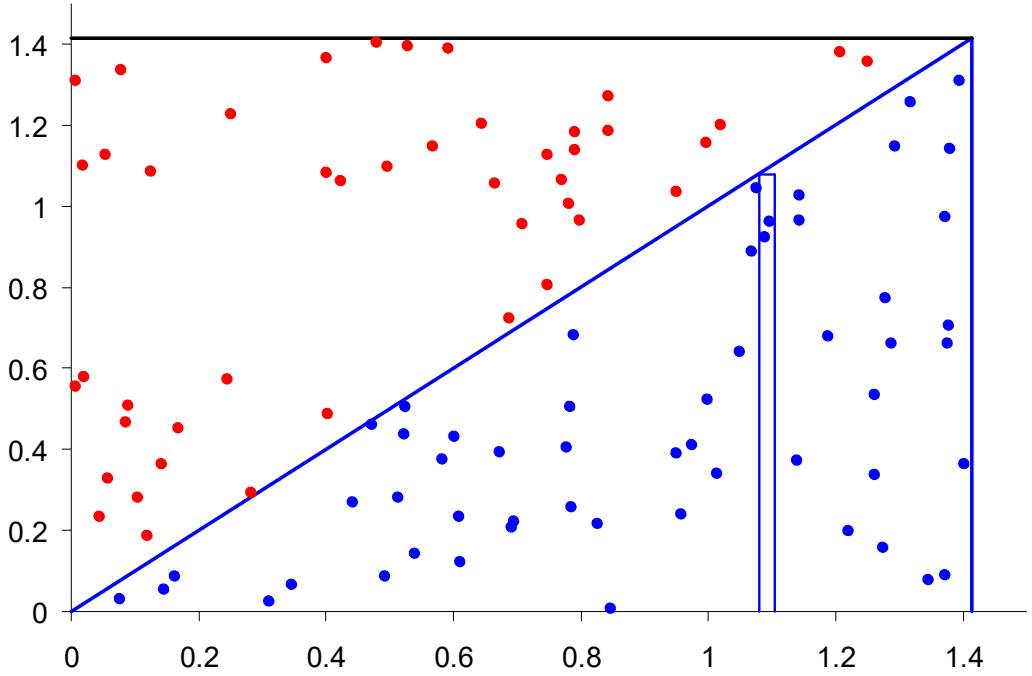
Distribution Sampling

- Let's consider the probability density distribution function

$$f(x) = \begin{cases} x, & x \in [0, \sqrt{2}] \\ 0, & \text{otherwise} \end{cases}$$

- It is trivial to generate PRN consistent with it using the ITM, but let' try and do it using **acceptance rejection**
  - enscribe the region described by the pdf in a more convenient rectangular region $[0,2^{0.5}]x[0,2^{0.5}]$
  - generate points uniformly distributed in it (just draws independent pairs of PRNs Unif$(0,2^{0.5})$)
  - reject points outside the pdf region (the remaining points are uniformly distributed inside the pdf region)

Distribution Sampling



Algorithm:

do

generate U1, U2 Unif[0,1]

while U2>U1

$Z1=U1*2^{0.5}$

$Z2=U2*2^{0.5}$

Generate number in pairs. Acceptance rate of 50%. On average need 4 Unif PRN for each 2 f(x) PRN. Not very efficient!

- The projection of the *accepted* points on the horizontal axis is distributed according with f(x)
- Therefore we just take the first coordinate of the points which are not rejected
- Because the same argument holds for projection of the *accepted* points on the vertical axis, we can also take the second coordinate of the points which are not rejected

Normal Sampling

- **Inverse Transform Method** is done in two steps
  1. a function which approximate $N^{-1}(x)$ (e.g. **Beasley**, **Springer**, **Moro** formula)
  2. one or two steps of a root search method (e.g. **Newton**, **Raphson**), which uses the previous result as initial guess
  3. This requires also a function which approximate N(X) (e.g **Hastings**, **Abramowitz**, **Stegun** or **Marsaglia**)
- Computationally this is usually expensive. Intel offers with MKL an implementation faster than any other method

Normal Sampling

- Rough approximation of Gaussian numbers can be **approximated** using the **Central Limit Theorem**
- Sum of random variables with any distribution **tend to** normal distribution.
- E.g. with 12 draws Unif(0,1):

$$X = -6 + \sum_{i=1}^{12} U_i, \qquad y \sim N(0,1)$$

- Note this is just an approximation of the normal distribution, and also an expensive one
- We know for certain how many Uniform RN are necessary for one Normal RN, but there is no 1 to 1 correspondence

Normal Sampling

- The **Box Muller** method is based on 2 properties of the bivariate normal N(0,I)
- Let Z=[Z1,Z2]~N(0,I)
  - $R=Z1^2+Z2^2$ is exponentially distributed with mean 2, i.e. $P(R<x)=1-\exp(-x/2)$
  - The point (Z1,Z2) is uniformly distributed on the circle of radius $R^{½}$
- This suggest that we can
  - generate a random angle alfa Unif in $[0,2\pi]$
  - generate a random R exponential with mean 2 via ITM
  - compute Z1 and Z2

Normal Sampling

- Pseudo code for Box Muller:
  1. Generate U1 and U2 Unif(0,1)
  2. $R=-2 \ln(U1)$
  3. $V=2\pi\ U2$
  4. $Z1=R^{½}\ \cos V$
  5. $Z2=R^{½}\ \sin V$
- So from a pair of RN Unif(0,1) we obtain a pair of independent RN N(0,1)

Normal Sampling

- The **Polar Rejection** method is a variation of Box Muller developed by **Marsaglia, Bray** based on acceptance rejection
- The algorithm is:

  do

  generate U1, U2 Unif[-1,1]

  $X=U1^2+U2^2$

  while $X>1$

  $Y=(-2 \ln X / X)^{½}$

  $Z1=Y\ U1$

  $Z2=Y\ U2$

Normal Sampling

- The polar rejection method has a rate of success of $\pi/4$, despite that it is faster than the Box Muller method
- Not all Uniform RNs in a sequence are used to generate a pair of Normal RN
- It is not possible to know how many are needed
- This is a weak point, and this is not good for some applications

Normal Sampling

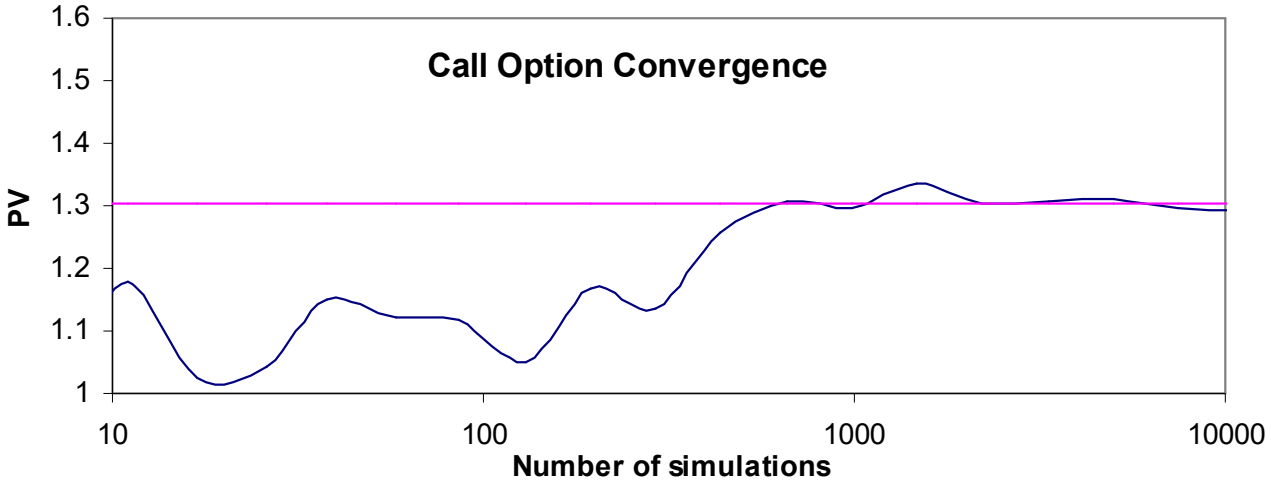
- The **Ziggurat** method is another acceptance rejection method
- Probably the fastest of all transformation methods
- Has a very high acceptance rate (98%)
- Suffers of the same drawbacks of other acceptance rejection methods

Call Option Example

$$V = e^{-rT} E\left[\max\left(S_0 e^{\left(\mu - \frac{\sigma^2}{2}\right)T + \sigma\sqrt{T}\varepsilon} - K, 0\right)\right]$$

$$= S_0 e^{\left(\mu - r - \frac{\sigma^2}{2}\right)T} E\left[\max\left(e^{\sigma\sqrt{T}\varepsilon} - \tilde{K}, 0\right)\right]$$

As less operations as possible inside E[•], which is part that gets repeated many times



Note how the approximation improves when *n* gets large

Call Option Example

```
function res = euroCallMC( rf, yield, vol, T, spot, strike, nSim )

    % generate gaussian random numbers
    % seed random number generator with an arbitrary seed
    rng(1347);
    % for efficiency generate all random number upfront
    rnds = randn( nSim, 1 );

    logStdDev = vol * sqrt( T );
    factor = spot * exp( ( (rf-yield) - 0.5 * vol * vol ) * T );
    k = strike / factor;

    price = 0; % loop accumulation variable
    for i = 1:nSim
        s = exp( logStdDev * rnds( i ) );
        payoff = max( s - k, 0.0 );
        price = price + payoff;
    end

    res = exp( -rf * T ) * factor * price / nSim;
```

Call Option Example (Vectorial)

```
function res = euroCallMCVec( rf, yield, vol, T, spot, strike,
nSim )

    % generate gaussian random numbers
    % seed random number generator with an arbitrary seed
    rng(1347);
    % for efficiency generate all random number upfront
    rnds = randn( nSim, 1 );

    logStdDev = vol * sqrt( T );
    factor = spot * exp( ( (rf-yield) - 0.5 * vol * vol ) * T );
    k = strike / factor;

    % vectorial operations
    s = exp( logStdDev * rnds);
    payoff = max( s - k, 0.0 );

    res = exp( -rf * T ) * factor * mean(payoff);
```

- We replaced the inner loop with Matlab vectorial operations
- This is faster then the non-vectorial implementation

Exchange Option Example

Suppose 2 shares follows GBM and interest rate is deterministic

$$dX_t = X_t\left(\mu_X\,dt + \sigma_X\,dW_t^X\right)$$
$$dY_t = Y_t\left(\mu_Y\,dt + \sigma_Y\,dW_t^Y\right)$$
$$dW_t^X\,dW_t^Y = \rho\,dt$$

A call spread option has payoff

$$Payoff = \max\left(X_T - Y_T, 0\right)$$

By fundamental theorem of asset pricing its value is:

$$V_t = e^{-rT}\,E\left[\max\left(X_T - Y_T, 0\right)\right]$$

The expectation can be estimated via Monte Carlo

$$\approx e^{-rT}\,\frac{1}{N}\sum_{i=1}^{N}\max\left(X_{T,i} - Y_{T,i}, 0\right)$$

Where the pairs (X_T, Y_T) are i.i.d.realizations with joint correlated lognormal distribution

$$\left(X_T, Y_T\right) \sim \text{jointly lognormal}$$

Exchange Option Example

- Transforming variables so that only Gaussian RNs are necessary:

$$E\left[\max\left(X_T - Y_T, 0\right)\right] = E\left[\max\left(X_0 e^{\left(\mu_X - \frac{\sigma_X^2}{2}\right)T + \sigma_X \sqrt{T}\,\varepsilon_X} - Y_0 e^{\left(\mu_Y - \frac{\sigma_Y^2}{2}\right)T + \sigma_Y \sqrt{T}\,\varepsilon_Y}, 0\right)\right]$$

$$\left(\varepsilon_X, \varepsilon_Y\right) \sim N\left(0, \Sigma\right), \qquad \Sigma = \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}$$

- To generate 1 realization of the payoff, we need a vector of 2 **correlated** Gaussian variables

Correlated Gaussian Numbers

- Let ϕ be a vector of RNs with distribution N(0,I)
- Let ε be a vector of RNs obtained as linear combinations of ϕ via a square matrix of coefficients A

$\bar{x}\varepsilon = A\,\phi$ ➔ $\varepsilon \sim N(0, AA^T)$

Proof:

- ε is Gaussian with null mean, because obtained as linear combination of Gaussian with null mean
- $Cov[\varepsilon]=E[\varepsilon\varepsilon^T]-E[\varepsilon]E[\varepsilon]^T=E[A\phi\phi^TA^T]=AE[\phi\phi^T]A^T$
  $=A(E[\phi\phi^T]-E[\phi]E[\phi]^T)A^T=AVar[\phi]A^T=AIA^T=AA^T$

Correlated Gaussian Numbers

- To generate a vector of RNs $N(0,\Sigma)$ we can multiply a vector of RN $N(0,I)$ by a matrix A such that $AA^T=\Sigma$
- There are different techniques to obtain a matrix A which factorize in this way Σ
  - Cholesky (with or without pivoting)
  - LDL
  - Diagonalization
- We can factorize either the covariance matrix Σ or the correlation matrix *C* and rescale by the variances afterwards. In fact: $\Sigma=V\,C\,V$, where V is a diagonal matrix with the standard deviations

Matrix Review

- A symmetric $\Leftrightarrow$ $A=A^T$
- A Positive Definite (PD) $\Leftrightarrow$ $xAx^T>0, \ \forall x\neq 0$
- A PD $\Leftrightarrow$ $\lambda>0, \forall\lambda$: $Ax=\lambda x$
- A Semi Positive Definite (SPD) $\Leftrightarrow$ $xAx^T\geq 0, \forall x \neq 0$
- A SPD $\Rightarrow$ $\lambda\geq 0, \forall\lambda$: $Ax=\lambda x$
- A sym and real $\Rightarrow$ λ real, $\forall\lambda$: $Ax=\lambda x$
- $\det(A)=\text{prod}(\lambda_i)$

Cholesky Decomposition

- Σ sym and PD $\leftarrow\rightarrow$ $\exists$ L triangular inferior: $LL^T=\Sigma$
- A covariance matrix is guaranteed by construction to be symmetric and SPD, but in absence of collinearities (which is the case most of the times), it is strictly PD
- The algorithm to compute the Cholesky matrix L
  - is stable even without pivoting
  - is simple and fast
  - available in many free source (e.g. LAPACK) and commercial libraries (e.g. MKL)
  - it can be obtained simply writing the linear system of equations associated with $LL^T=\Sigma$
- Since L is triangular inferior it can be multiplied by a vector in $n^2/2$ operations instead of n^2

Cholesky Decomposition

$$\mathbf{A} = \mathbf{L}\mathbf{L}^{\mathbf{T}} = \begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} L_{11} & L_{21} & L_{31} \\ 0 & L_{22} & L_{32} \\ 0 & 0 & L_{33} \end{pmatrix} = \begin{pmatrix} L_{11}^2 & & (\text{symmetric}) \\ L_{21}L_{11} & L_{21}^2 + L_{22}^2 & \\ L_{31}L_{11} & L_{31}L_{21} + L_{32}L_{22} & L_{31}^2 + L_{32}^2 + L_{33}^2 \end{pmatrix}$$

Solving this system of equations leads to the recursive algorithm:

$$L_{j,j} = \sqrt{A_{j,j} - \sum_{k=1}^{j-1} L_{j,k}^2}.$$

$$L_{i,j} = \frac{1}{L_{j,j}} \left(A_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k} \right), \qquad \text{for } i > j.$$

Cholesky Decomposition

- For a 2x2 correlation matrix the Cholesky decomposition is:

$$\begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2} \end{bmatrix} \begin{bmatrix} 1 & \rho \\ 0 & \sqrt{1-\rho^2} \end{bmatrix}$$

Diagonalization

- Since Σ is sym and SPD ➔ $\exists$ U,D: $UDU^T=\Sigma$, where D diagonal and U orthogonal
- This is the eigensystem decomposition of Σ, where D is the matrix of eigenvalues and U is the matrix of eigenvectors
- $(UD^{½})(D^{½}U^T)=\Sigma$
- The factor $UD^{½}$ is not triangular, hence multiplying it by a vector has a cost n^2
- Obtaining the eigensystem decomposition is much slower than Cholesky
- Algorithm for eigensystem decomposition of real and symmetric matrices are available in many free source (e.g. LAPACK) and commercial libraries (e.g. MKL)

Cholesky with Pivoting

- If the covariance matrix has not full rank (i.e. it is SPD but not PD), Cholesky will not work
- A possibility is to introduce pivoting, and stop the algorithm when all residual element of the matrix are null
- We still obtain a triangular matrix L, but also a permutation matrix P (remember permutation matrices are orthogonal)
- $(PL)(L^T P^T)=\Sigma$
- The whole factor PL is a full matrix, hence the cost of multiplying it by a vector is n^2

Exchange Option Example (Vectorial)

```
function [price, stderr] = exchangeOption(rf, yx, yy, sx, sy, rho, x0, y0, T, nsim)
   mux = rf - yx;
   muy = rf - yy;

   driftx = (mux - 0.5*sx^2)*T;
   drifty = (muy - 0.5*sy^2)*T;

   diffx = sx * sqrt(T);
   diffy = sy * sqrt(T);

   df = exp(-rf*T);  % discount factor

   L = chol([1.0 rho; rho 1.0]);  % compute cholesky matrix

    % seed random number generator with an arbitrary seed
   rng(1347);
   rnd_uncorrelated = randn(nsim, 2);
   rnd_correlated   = rnd_uncorrelated * L;
   xT = x0 * exp(driftx + diffx * rnd_correlated(:,1));
   yT = y0 * exp(drifty + diffy * rnd_correlated(:,2));
   payoff = max(0, xT - yT );

   price = df * mean(payoff);
   stderr = sqrt(var(payoff) / nsim);
```

Asian Price Call Option Example

Suppose a share follows a GBM and interest rate is deterministic

$$dS_t = S_t\left(\mu\, dt + \sigma\, dW_t\right)$$

An Asian Call option has payoff

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

By fundamental theorem of asset pricing its value is

$$V_t = e^{-rT} E\left[\max\left(\frac{1}{n}\sum_{i=1}^{n} S_{t_i} - K, 0\right)\right]$$

and the MC estimator is:

$$\approx e^{-rT}\frac{1}{m}\sum_{j=1}^{m}\max\left(\frac{1}{n}\sum_{i=1}^{n} S_{t_i} - K, 0\right)$$

where the vector of $\{S_{ti}\}$ are i.i.d.realizations with correlated jointly lognormal distribution

$$S_{t_i} \sim LogN\left(\log S_0 + \left(\mu - \frac{\sigma^2}{2}\right)t_i, \sigma^2 t_i\right)$$

Asian Price Call Option Example

- Note that to compute one realization of the payoff we need the price S_t at n different times t_i
- Obviously these prices are correlated, because the price at time t_i depends on all prices before time t_i

$$E\left[\max\left(\frac{1}{n}\sum_{i=1}^{n} S_{t_i} - K, 0\right)\right] = E\left[\max\left(\frac{S_0}{n}\sum_{i=1}^{n} e^{\left(\mu-\frac{\sigma^2}{2}\right)t_i+\sigma W_{t_i}} - K, 0\right)\right]$$

$$W \sim N(0, \Sigma), \quad \Sigma_{i,j} = Cov\left[W_{t_i}, W_{t_j}\right] \quad \text{i.e.} \quad \Sigma = \begin{bmatrix} t_1 & t_1 & \cdots & t_1 \\ t_1 & t_2 & \cdots & t_2 \\ \vdots & \vdots & \ddots & \vdots \\ t_1 & t_2 & \cdots & t_n \end{bmatrix}$$

we know how to generate a vector of RNs with this distribution

Asian Price Call Option Example

- There is another way. We can express the price S_t at time t_i as a function of the price at time t_{i-1}

$$E\left[\max\left(\frac{1}{n}\sum_{i=1}^{n} S_{t_i} - K, 0\right)\right] = E\left[\max\left(\frac{1}{n}\sum_{i=1}^{n} S_{t_{i-1}} e^{\left(\mu - \frac{\sigma^2}{2}\right)\left(t_i - t_{i-1}\right) + \sigma\sqrt{t_i - t_{i-1}}\varepsilon_t} - K, 0\right)\right]$$

$$\varepsilon \sim N(0, I)$$

- Now we only need RNs N(0,1) to compute the payoff
- All values of S_t can be resolved recursively using ε and starting from S_0, which is known
- We have simulated **a full path** for S_t

Asian Price Call Option Example

S_0 =	10	(current asset price)
σ =	30%	(annualised volatility)
μ =	5%	(risk neutral drift)
T =	3	(maturity in years)
K =	9.50	(ATM strike price)
rf =	8%	(interest rate c.c.)
dT =	0.5	(time step)

Asianing details: Arithmetic average of 7 half yearly price observations starting with today's price

	Normal random deviates N(0,1)				Price paths			
Time	Serie 1	Serie 2	Serie 3	Serie 4	Path 1	Path 2	Path 3	Path 4
0					10	10	10	10
0.5	0.16385	-1.00582	-1.60575	1.943454	10.37961	8.098814	7.131013	15.14021
1	1.129533	-0.12767	-0.04194	0.298869	13.22294	7.902156	7.085547	16.17155
1.5	-1.09882	0.232403	0.879755	-0.27024	10.49981	8.322276	8.560695	15.30878
2	-1.48763	-1.04271	-0.10113	0.068391	7.677417	6.687529	8.399968	15.57137
2.5	0.341673	-0.3757	0.574098	-0.44848	8.275199	6.190694	9.511604	14.19368
3	0.200921	-0.45899	1.704116	0.609557	8.657143	5.630401	13.68787	16.19339
				Payoff	0.32	0.00	0.00	5.15
				Avg	1.367539			
				Premium	1.075744	(discounted)		

Asian Price Call Option Example

```
function res = asianCall( rf, yield, vol, T, spot, strike, nSteps, nSim )

    dt = T / nSteps;
    nFixings = nSteps + 1;
    logDrift = ( (rf-yield) - 0.5 * vol * vol ) * dt;
    logStdDev = vol * sqrt( dt );

    % seed random number generator with an arbitrary seed
    rng(1347);
    rnds = randn( nSim, nSteps ); % generate Gaussian numbers

    priceSum= 0.0;
    for i = 1:nSim %loop on simulations
        pathSum = spot;
        s = spot;
        for step = 1:nSteps % loop on time steps
            s = s * exp( logDrift + rnds( i, step ) * logStdDev );
            pathSum = pathSum + s;
        end
        avg = pathSum / nFixings; % compute average
        payoff = max( avg - strike, 0.0 );
        priceSum = priceSum + payoff;
    end

    res = exp( -rf * T ) * priceSum / nSim;
```

Note there are 2 loops now, 1 on simulation and 1 on time steps

Asian Price Call Option (Vectorial)

```
function res = asianCallVec( rf, yield, vol, T, spot, strike, nSteps, nSim )

    dt = T / nSteps;
    nFixings = nSteps + 1;
    logDrift = ( (rf-yield) - 0.5 * vol * vol ) * dt;
    logStdDev = vol * sqrt( dt );

    % seed random number generator with an arbitrary seed
    rng(1347);
    rnds = randn( nSim, nSteps ); % generate Gaussian numbers

    s = spot * ones(nSim, 1); % init spot
    pathSum = s;
    for step = 1:nSteps % loop on time steps
        s = s .* exp( logDrift + rnds( :, step ) * logStdDev );
        pathSum = pathSum + s;
    end

    avg = pathSum / nFixings; % compute average
    payoff = max( avg - strike, 0.0 );

    res = exp( -rf * T ) * mean(payoff);
```

- We re-ordered the 2 loops, then replaced the inner loop (the one on simulations) with Matlab vectorial operations

Price Simulation

- If the explicit solution of the SDE is known, as in the case of a GBM:

$$\frac{dS_t}{S_t} = \mu\, dt + \sigma\, dW_t, \qquad dW_t \sim N\left(0, \sqrt{dt}\right)$$

  then we can simulate **exact** paths:

$$S_{t+\Delta t} = S_t\, e^{\left(\mu - \frac{\sigma^2}{2}\right)\cdot \Delta t + \sigma\sqrt{\Delta t}\cdot \varepsilon_t}, \qquad \varepsilon_t \sim N\left(0,1\right)$$

- we can simulate only time step strictly necessary to evaluate the payoff

Price Simulation

- Otherwise **approximations** can be used.
- Given:

$$dS_t = \mu\left(S_t, t\right)dt + \sigma\left(S_t, t\right)dW_t, \quad dW_t \sim N\left(0, \sqrt{dt}\right)$$

  typical approximation schemes used are:

  - **Euler** (1st order **weak** approximation in sqrt(Δt)):

$$S_{t+\Delta t} - S_t = \mu\left(S_t, t\right)\Delta t + \sigma\left(S_t, t\right)\sqrt{\Delta t}\,\varepsilon_t, \quad \varepsilon_t \sim N\left(0,1\right)$$

  - **Milstein** (1st order **strong** approximation), where an the extra term is added to the above

$$+\frac{1}{2}\frac{\partial \sigma\left(S_t, t\right)}{\partial S_t}\sigma\left(S_t, t\right)\left(\Delta t\,\varepsilon_t^2 - \Delta t\right)$$

Price Simulation

- Let's compare the three alternatives for a GBM:
  - Exact:

$$S_{t+\Delta t} = S_t e^{\left(\mu - \frac{\sigma^2}{2}\right)\Delta t + \sigma\sqrt{\Delta t}\,\varepsilon_t}$$

  - Euler:

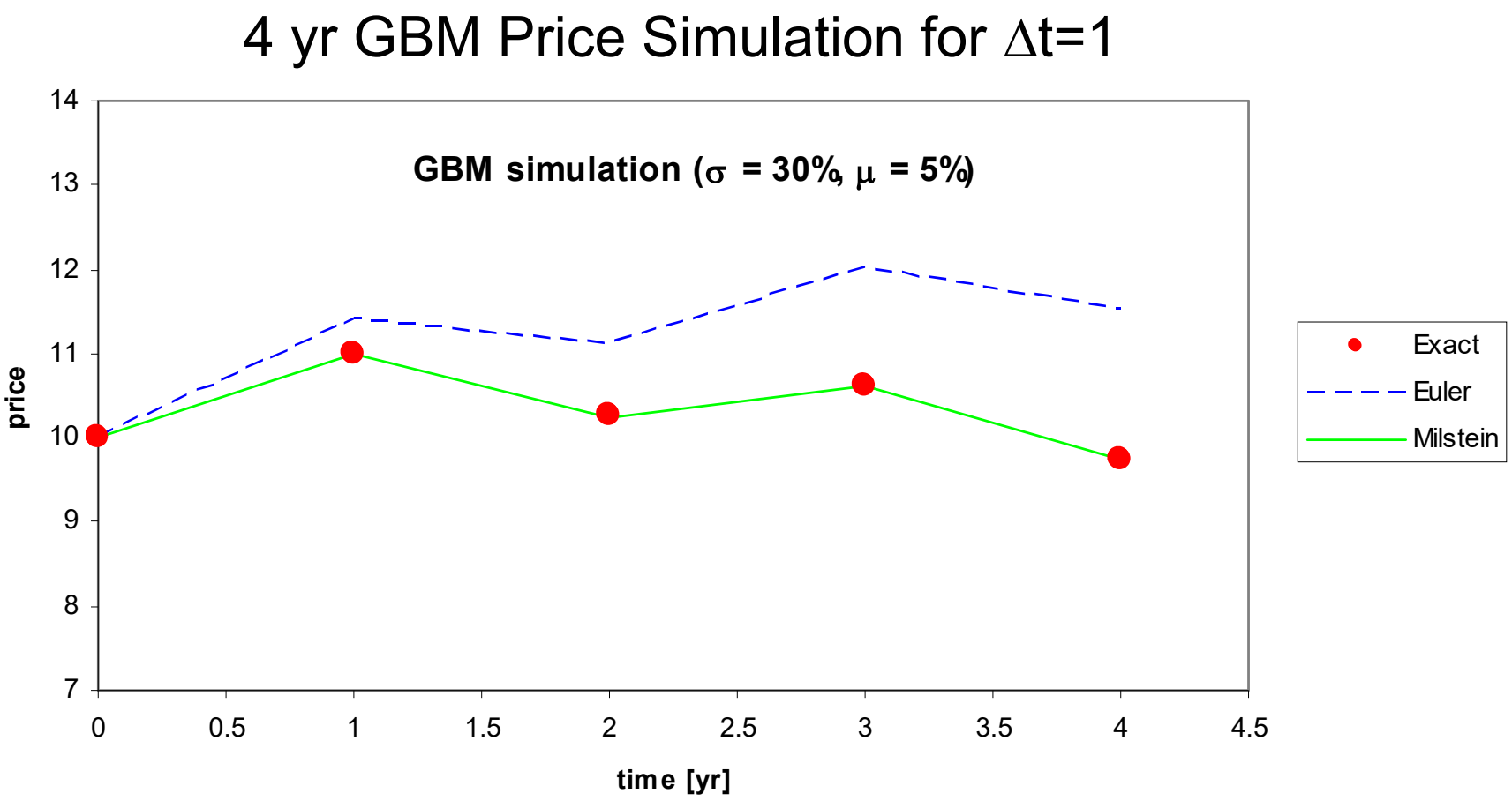
$$S_{t+\Delta t} = S_t\left[1 + \mu\,\Delta t + \sigma\sqrt{\Delta t}\,\varepsilon_t\right]$$

$$\varepsilon_t \sim N(0,1)$$

  - Milstein:

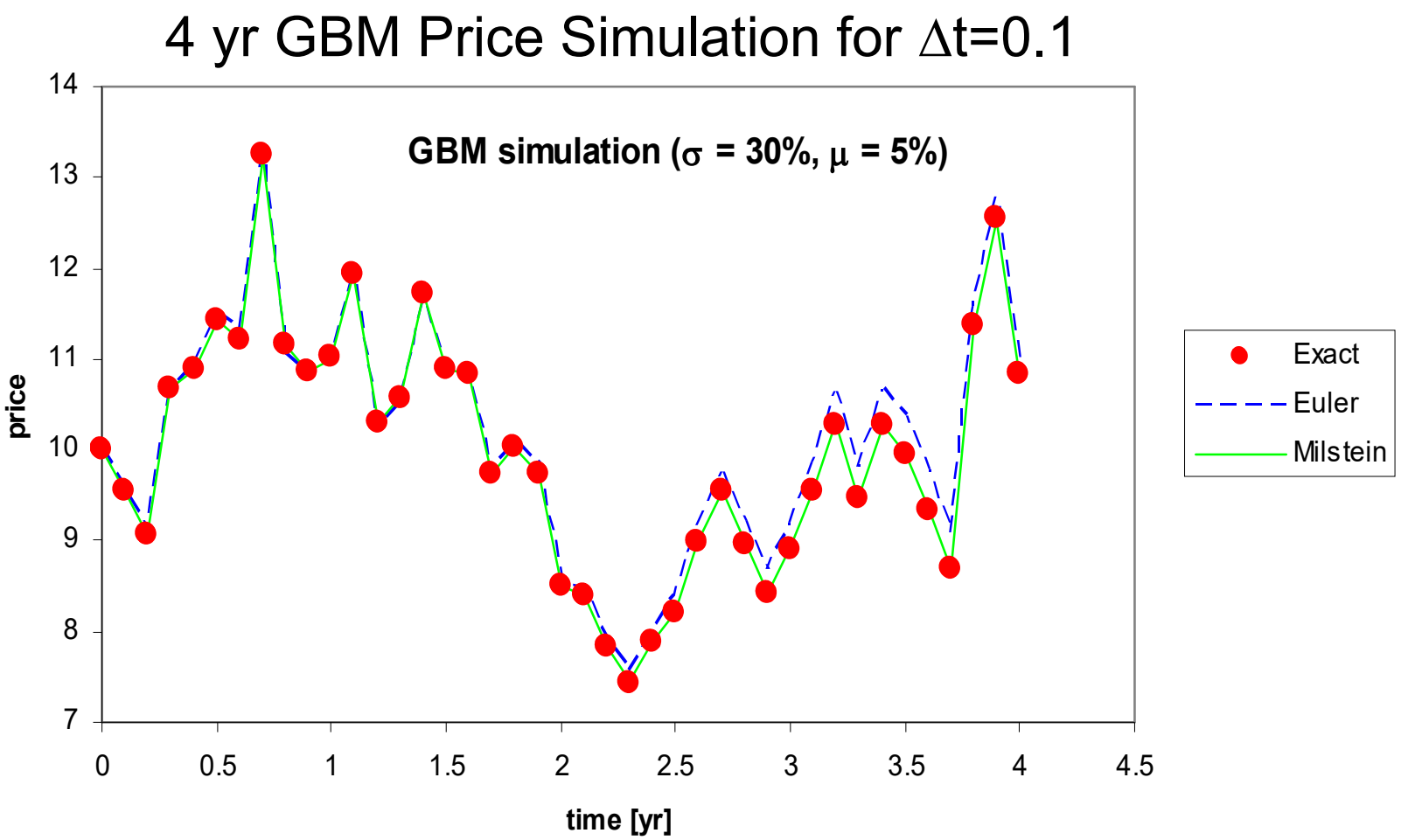
$$S_{t+\Delta t} = S_t\left[1 + \mu\,\Delta t + \sigma\sqrt{\Delta t}\,\varepsilon_t + \frac{\sigma^2}{2}\Delta t\left(\varepsilon_t^2 - 1\right)\right]$$

Price Simulation Example



- Euler final error is much larger than Milstein

Price Simulation Example

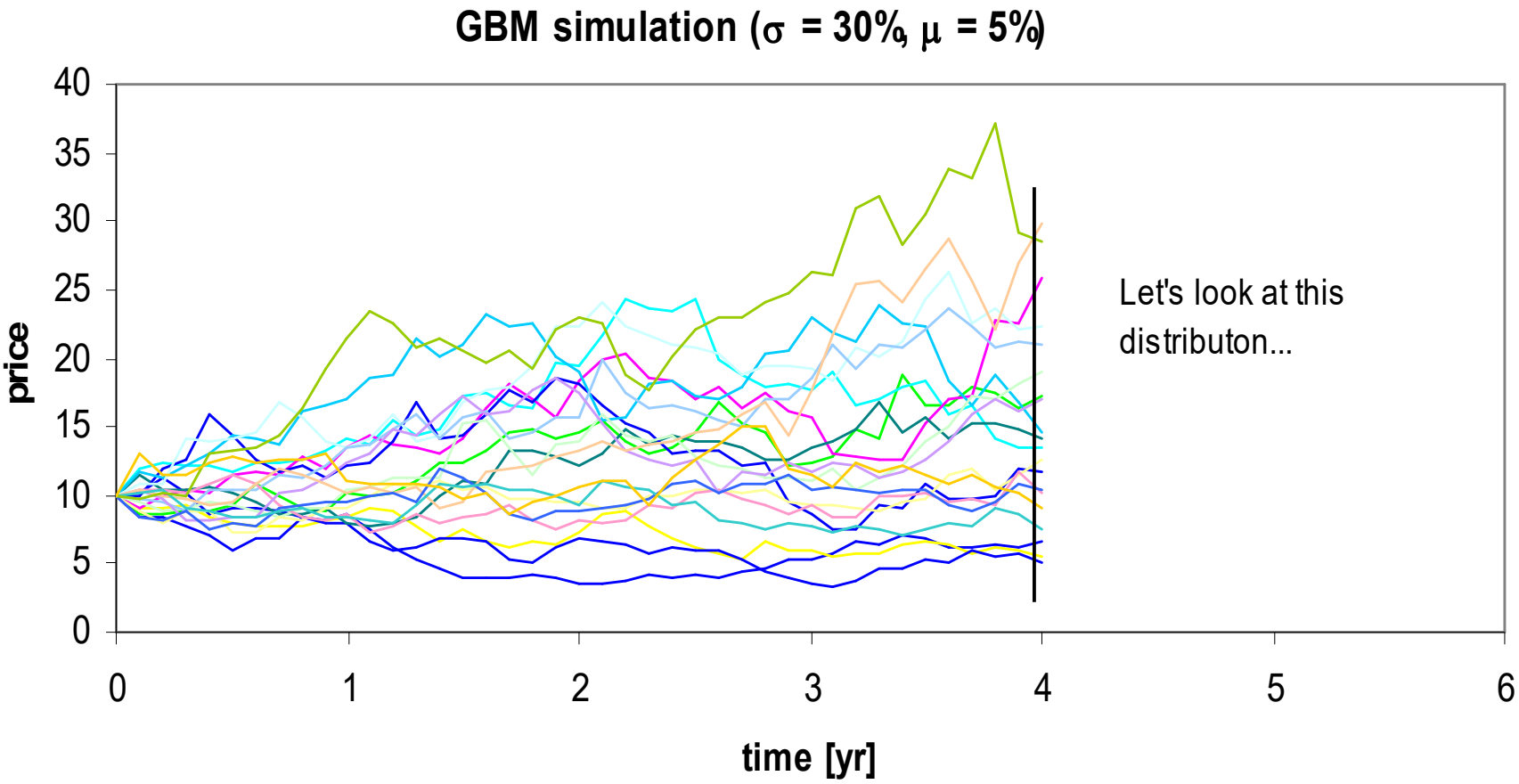


- With a smaller Δt (and consequently more steps) the Euler final error is smaller and comparable with Milstein

Price Simulation - Considerations

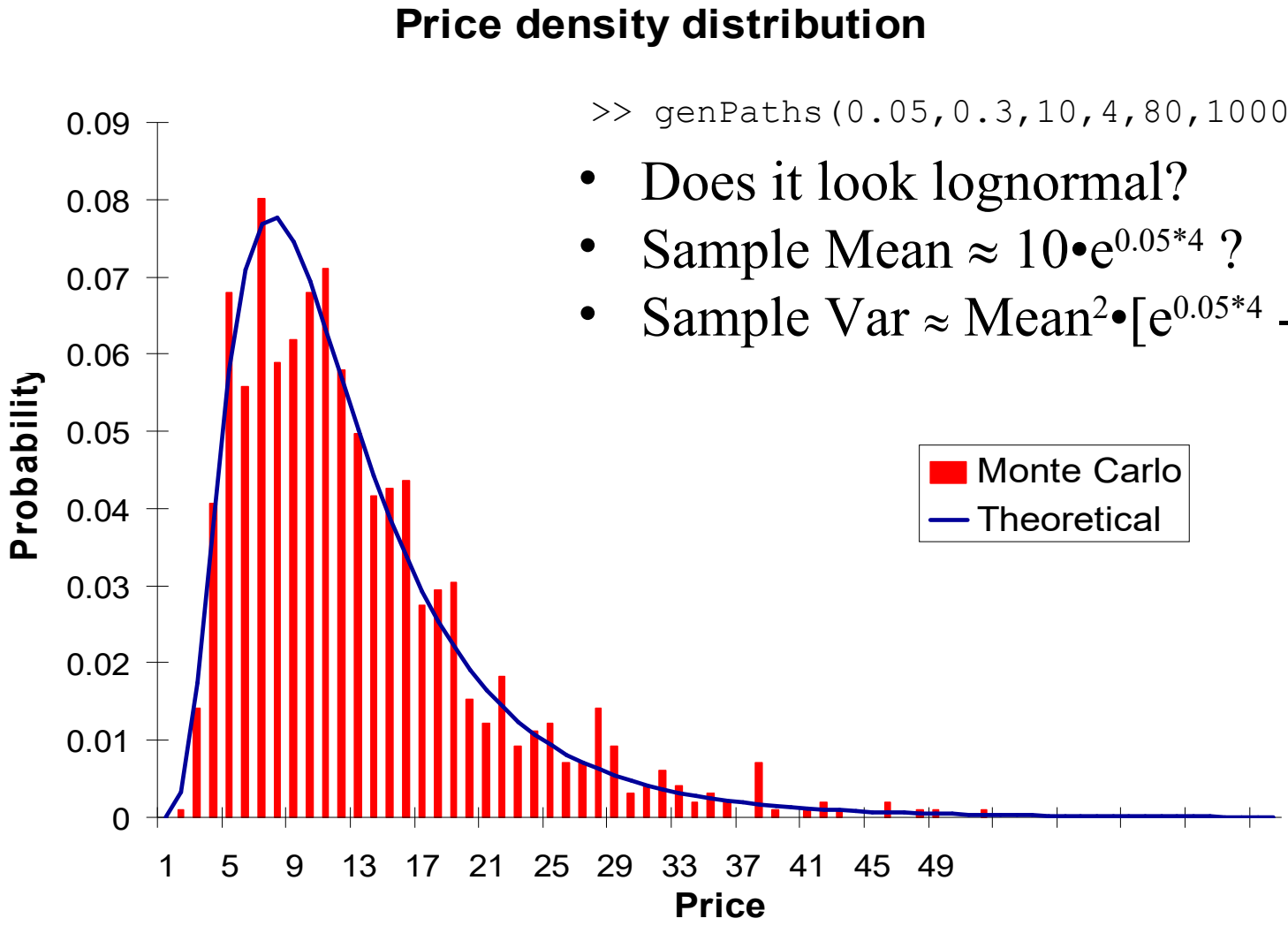
- We observed that when Δt is small, approximate solutions do not deviate too much from the exact solution, but we need more time steps and computation cost is higher
- If instead Δt is big, the error can be significant.
- Care should be taken to ensure that the cross-path properties of the price distribution are consistent with the original process
- Approximation can be computationally "lighter" than the exact solution (e.g. in GBM there is no *exponential*), but needs a small Δt and potentially more time steps
- If the exact solution is used Δt can be as large as we want, which might allow to reduce number of time steps (only those instants strictly needed for the pricing), thus saving computational time

Price Distribution Properties



- A good check: are price properties consistent with theoretical values?

Price Distribution Properties




```
>> genPaths(0.05,0.3,10,4,80,10000);
```

- Does it look lognormal?
- Sample Mean $\approx 10 \bullet e^{0.05*4}$?
- Sample Var $\approx \text{Mean}^2 \bullet [e^{0.05*4} - 1]$?

Problem Dimension

- The **problem dimension** is defined as the total number of RN necessary to generate one realization of the payoff
- It is: dim = nTimeSteps x nRiskFactors, assuming that 1 RNs is sufficient to generate 1 realization of each risk factor
- In practice, depending on the probability distribution of the risk factors, and the method used to generate RN, often it is impossible to specify exactly the problem dimension. For instance, if we are using polar rejection to generate Gaussian RNs, we do not know exactly how many Uniform RNs we need

Problem Dimension

- Consider an Asian Call options on a equally weighted basket of 2 assets, with 5 yearly observations
  - The price of the 1st asset is driven by a GBM (we have an exact formula)
  - The price of the 2nd asset is driven by a more complex process with 2 Gaussian risk factors. There is no close form solution for it and we use Euler, with the recommendation that Δt does not exceed half year
- For the payoff we would only need 5 time steps, and that would be ok for the GBM, but the second process pose a limit on Δt and requires 10 time steps. Because all processes must have the same time steps, we use 10 time steps for all processes
- The total number of Gaussian risk factors is 1+2=3
- We are using ICT to generate Gaussian RNs from Uniform RNs, therefore for 1 Gaussian RN we use exactly 1 Uniform RN
- The problem dimension is = 3x10 = 30 (Uniform RNs)

Choice of State Variables

- When we simulate paths, sometime we have flexibility in the choice of the state variables we diffuse
- Given a process S_t, if we can express it as a function of some other process X_t, i.e. $S_t=\text{f}(X_t,t)$, we have the choice to diffuse directly S_t or to diffuse X_t and then reconstruct S_t only for the time steps where needed using the equation $S_t=\text{f}(X_t,t)$
- We define
  - S_t, where t is needed for the payoff, is an **observable**
  - X_t is the **state variable**
  - $\text{d}X_t=\ldots$ is the **diffused process**
  - $S_t=\text{f}(X_t,t)$ is the **reconstruction equation**
- If $X_t=S_t$, it means that the observable is diffused directly (it is the state variable) and there is no need for reconstruction

Choice of State Variables

- The choice is usually dictated by considerations about efficiency and it is very useful when:
  - We simulate more time steps than needed for the payoff, and diffusing X_t is computationally less expensive than diffusing S_t
  - The payoff requires many observables all dependent on the same state variable (very common for commodity and interest rates models)

State Variables Example

$$dS_t = S_t\left(\mu\, dt + \sigma\, dW_t\right)$$

$$S_{t+\Delta t} = S_t \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)\Delta t + \sigma\, \Delta W_t\right)$$
$$= S_t \exp\left(a_t + \sigma\, \Delta W_t\right) \quad \text{(diffusion)}$$

$$W_{t+\Delta t} = W_t + \Delta W_t \quad \text{(diffusion)}$$
$$S_t = S_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma\, W_t\right)$$
$$= a_t \exp\left(\sigma\, W_t\right) \quad \text{(reconstruction)}$$

1. Diffuse directly the observable
   - diffusion is expensive (use exp())
   - no need for reconstruction

2. Split the problem into diffusion of W_t and reconstruction of S_t
   - diffusion is very cheap
   - reconstruction is expensive (use exp())
   - overall cheaper if the reconstruction is called less times than the diffusion

State Variables Example 2

- A more advanced example of choice of state variables is related to a simple mean reverting commodity model which is described by the SDE

$$\frac{dF_t(T)}{F_t(T)} = \sigma \exp\left[-\alpha(T-t)\right] dW_t \qquad \sigma, \alpha > 0$$

  where $F_t(,T)$ is the price of a future contract on some asset *X* (e.g. Oil) expiring at time *T* and observed at time *t*
- Here the SDE describes the **simultaneous** evolution of the prices of **many different assets** (future contracts with different maturities), all driven by **the same source of risk**
- Note that, because future prices are Martingale, and this SDE is drift-less, this SDE is already risk neutral

State Variables Example 2

- Consider the following option pricing problem:

  payoff=max[$F_{\tau}(T_2)$-$F_{\tau}(T_1)$, 0]

  where:

  $F_t(T)$ is driven by the SDE introduced in previous slide with σ=30% and α=0.5

  $t_0 = 0$

  τ = 25 days

  T_1 = 1 month, $F(0, T_1)$=10

  T_2 = 2 months, $F(0, T_2)$=12

- Here the payoffs depends on two observables, but the SDE which drives both observables is driven by one single source of risk.

State Variables Example 2

- The Euler approximation of the SDE is:

$$F_{t+\Delta t}(T) \approx F_t(T)\left[1+\sigma \exp\left[-\alpha(T-t)\right]\sqrt{\Delta t}\,\varepsilon\right], \qquad \varepsilon \sim N(0,1)$$

- One single ε affects both prices but in different extent
- If we jump from t_0 to τ with one single Euler step, and the random variable ε =0.3, then the prices of the two contracts become approximately:
  - $F_\tau(T_1) = 10*(1+0.3*\exp(-0.5*1/12)*(25/365)^{½}*0.3) = 10.226$
  - $F_\tau(T_2) = 12*(1+0.3*\exp(-0.5*2/12)*(25/365)^{½}*0.3) = 12.260$
- Note that we are **diffusing two state variables** simultaneously, $F_t(T_1)$ and $F_t(T_2)$!

State Variables Example 2

- How can we diffuse just one state variable instead of 2?
- Let's formalize the problem: we look for a state variable X_t **not dependent on** ***T***, such that:

$$\forall t \;\; \exists \;\; f\left(T, X_t\right) \;:\; F_t(T) = f\left(T, X_t\right)$$

- The state variable X_t is what get diffused
- The observable variables $F_t(T)$ can be reconstructed purely from the value of X_t, **without the need to know its history**

State Variables Example 2

- To find such state variable we perform a variable transformations: we set $X_t(T)=\ln F_t(T)$ and apply Ito's lemma, then integrate

$$dX_t(T)=\frac{\partial X_t}{\partial t}dt+\frac{\partial X_t}{\partial F_t(T)}dF_t(T)+\frac{1}{2}\frac{\partial^2 X_t}{\partial F_t(T)^2}\left(dF_t(T)\right)^2$$

$$\frac{\partial X_t}{\partial t}=0,\qquad \frac{\partial X_t}{\partial F_t(T)}=\frac{1}{F_t(T)},\qquad \frac{\partial^2 X_t}{\partial F_t(T)^2}=-\frac{1}{F_t(T)^2}$$

$$dX_t(T)=-\frac{1}{2}\sigma^2\exp\left[-2\alpha(T-t)\right]dt+\sigma\exp\left[-\alpha(T-t)\right]dW_t$$

$$X_t(T)=X_0(T)-\frac{\sigma^2}{2}\int_0^t e^{-2\alpha(T-u)}du+\sigma\int_0^t e^{-\alpha(T-u)}dW_u$$

State Variables Example 2

- Going back to the original variables

$$F_t(T) = \exp\left(X_t(T)\right)$$
$$= F_0(T)\exp\left[-\frac{\sigma^2}{2}\int_0^t e^{-2\alpha(T-u)}du + \sigma\int_0^t e^{\alpha(T-u)}dW_u\right]$$

- The stochastic integral, which is the candidate to be the state variable, depends on *T*, therefore we cannot find a single state variable which works for any *T*. We still need to have two state variables, one for T_1 and one for T_2

State Variables Example 2

- Luckily in this case the part of the stochastic integral dependent on *T* can be separated by the rest of the integral, and moved outside of the integral

$$\int_0^t e^{-\alpha(T-u)} dW_u = e^{-\alpha T} \int_0^t e^{\alpha u} dW_u$$

- We could pick as state variable the Ito integral which appears in the solution, and the one above would be the reconstruction equation

$$Y_t = \int_0^t e^{\alpha u} dW(u)$$ diffusion

$$F_t(T) = F_0(T) \exp\left[-\frac{\sigma^2}{2} e^{-2\alpha T} \int_0^t e^{2\alpha u} du + \sigma e^{-\alpha T} Y(t) \right]$$ reconstruction

State Variables Example 2

- The exact diffusion process in an interval $[t, t+\Delta t]$ is

$$Y_{t+\Delta t} - Y_t = \int_0^{t+\Delta t} e^{\alpha u} dW_u - \int_0^{t} e^{\alpha u} dW_u = \underbrace{\int_t^{t+\Delta t} e^{\alpha u} dW_u}_{N\left(0, v_t\right)}$$

path increments for Y_t are just $N(0, v_t)$

$$v_t = Var\left[\int_t^{t+\Delta t} e^{\alpha u} dW_u\right] = \int_t^{t+\Delta t} e^{2\alpha u} du = \frac{e^{2\alpha \Delta t} - 1}{2\alpha} e^{2\alpha t} \xrightarrow[t \to \infty]{} \infty$$

- the variance v_t changes at every time step, but do not be confused by the notation: the sub-index t does not mean it is a stochastic process
- $Y(t)$ is not a very nice state variable, because in every successive time interval its increment has variance exponentially growing in time
- We could live with it in Monte Carlo, if t does not get too large, but it would be hard to deal with in a tree

State Variables Example 2

- In this case we can transform the process further:

$$dY_t = e^{\alpha t}\, dW_t$$

$$\text{let} \quad Z_t = Y_t e^{-\alpha t}$$

$$\frac{\partial Z_t}{\partial t} = -\alpha Z(t), \qquad \frac{\partial Z_t}{\partial Y_t} = e^{-\alpha t}, \qquad \frac{\partial^2 Z_t}{\partial Y_{t^2}} = 0$$

$$dZ_t = \frac{\partial Z_t}{\partial t} dt + \frac{\partial Z_t}{\partial Y_t} dY(t) + \frac{1}{2} \frac{\partial^2 Z_t}{\partial Y_{t^2}} \left(dY_t\right)^2$$

$$= -\alpha Z_t\, dt + dW_t$$

- obtaining an Ornstein Uhlenbeck process mean reverting around zero

State Variables Example 2

- and from its solution we write diffusion and reconstruction equations:

$$Z_{t+\Delta t} = Z_t e^{-\alpha \Delta t} + \underbrace{e^{-\alpha \Delta t} \int_t^{t+\Delta t} e^{-\alpha(t-u)} dW_u}_{N(0, v_t)} = Z_t e^{-\alpha \Delta t} + \sqrt{v_t} \underbrace{\varepsilon_t}_{N(0,1)}$$

diffusion

$$v_t = Var\left[e^{-\alpha \Delta t} \int_0^{\Delta t} e^{\alpha u} dW_u \right]$$

$$= e^{-2\alpha \Delta t} \int_0^{\Delta t} e^{2\alpha u} du = \frac{1}{2\alpha}\left(1 - e^{-2\alpha \Delta t}\right)$$

These coefficients are constant across paths and can be pre-computed

$$F_t(T) = F_0(T) \exp\left(-\frac{\sigma^2}{2} \int_0^t e^{-2\alpha(T-u)} du + \sigma e^{-\alpha(T-t)} Z_t \right)$$

reconstruction

Monte Carlo Summary

- Monte Carlo strengths:
  - simple and flexible (with a clear trade-off between simplicity and efficiency)
  - easy parallel speedup
  - easily able to handle high-dimensional problems (avoids “curse of dimensionality” of finite difference methods)
- Monte Carlo weaknesses:
  - not as efficient as finite differences for very low dimensions (1-3?)
  - not yet efficient for applications with optional exercise (American options, Bermudan options)

Further Readings

- Glasserman, *Monte Carlo Methods in Financial Engineering*
- Jackel, *Monte Carlo methods in Finance*
- Marsaglia, Tsang, *The Ziggurat method for Generating Random Variables,* *http://www.jstatsoft.org/v05/i08/paper*
- Higham*, Cholesky Factorization,* *http://eprints.ma.man.ac.uk/1199/01/covered/MIMS_ep2008_116.pdf*
- Prof Giles' lecture notes*, http://people.maths.ox.ac.uk/~gilesm/mc/*