FMO6 — Web: https://tinyurl.com/ycaloqk6 Polls: https://pollev.com/johnarmstron561

Lecture 6

Dr John Armstrong

King's College London

August 22, 2020

Improving Monte Carlo Pricing

Revision: Antithetic Sampling

- Suppose we have a Monte Carlo pricer based on drawing n normally distributed random numbers ϵ_i
- It is often better to compute the price using a sample based on ϵ_i and $-\epsilon_i$ rather than to use a sequence of $2n$ independent random variables.
- Theory: If X_1 and X_2 are random variables with $E(X_1) = E(X_2)$ then

$$E(X_1) = E\left(\frac{X_1 + X_2}{2}\right)$$

But

$$\mathrm{Var}\left(\frac{X_1 + X_2}{2}\right) = \frac{1}{4}(\mathrm{Var}(X_1) + \mathrm{Var}(X_2) + 2\,\mathrm{Cov}(X_1, X_2))$$

Let X_1 be estimate based on the n variables ϵ_i. Let X_2 be estimate based on $-\epsilon_i$. We will often have $\mathrm{Cov}(X_1, X_2)$ is negative.

MATLAB implementation of Antithetic sampling

```
% Price a call option by antithetic sampling
function [price,errorEstimate] = callAntithetic( K,T, ...
                                 S0,r,sigma, ...
                                 nPaths )

logS0 = log(S0);
epsilon1 = randn( nPaths/2,1 );
epsilon2 = -epsilon1;
logST1 = logS0 + (r-0.5*sigma^2)*T + sigma*sqrt(T)*epsilon1;
logST2 = logS0 + (r-0.5*sigma^2)*T + sigma*sqrt(T)*epsilon2;
ST1 = exp( logST1 );
ST2 = exp( logST2 );
discountedPayoffs1 = exp(-r*T)*max(ST1-K,0);
discountedPayoffs2 = exp(-r*T)*max(ST2-K,0);
price = mean(0.5*(discountedPayoffs1+discountedPayoffs2));
errorEstimate = std(0.5*(discountedPayoffs1+discountedPayoffs2))/sqrt(nPaths/2)
end
```

Antithetic Sampling Results

- Parameters: $S0 = 100$, $K = 100$, $\sigma = 0.2$, $r = 0.14$, $T = 1$, $N = 10000$
- Results:

Method	Price	Standard error estimate
Black–Scholes Formula	3.0679	
Naive Monte Carlo	3.0794	0.197
Antithetic Sampling	3.0771	0.054

- Conclusion: Antithetic sampling is easy to implement and often rather effective.

Importance Sampling

- Monte Carlo pricing is an integration method.
- You can use substitution to change one integral to another integral by re-parameterizing
- Equivalently you can change the distribution from which you draw your samples so long as apply appropriate weights to correct for this.
- Monte Carlo integration is exact when the price function is constant
- If we can re-parameterize so the price function is nearer to being constant, we will have reduced the variance of the Monte Carlo algorithm.

Importance Sampling Example

- Suppose we want to price a far out of the money knock out call option
- Suppose that for 99% of price paths the option will end out of the money
- This means that 99% of price paths in the Monte Carlo calculation will give us no information.
- Instead: find a way to generate the 1% of price paths where the option ends up in the money; compute the expectation for these paths; re-weight by multiplying by 100.
- For simplicity, let's do this for a vanilla call option to see how it improves upon ordinary Monte Carlo.

Calculation

- Generate stocks prices at time T using the formula:

$$\log(S_T) = \log(S_0) + \left(r - \frac{1}{2}\sigma^2\right) T + \sigma\sqrt{T} N^{-1}(u)$$

  where u is uniformly distributed on $[0,1]$.
- Option is in the money only if $\log(S_T) \geq \log(K)$. Equivalently only if:

$$u \geq u_{\min} := N\left(\frac{\log(K) - \log(S_0) - (r - (1/2)\sigma^2) * T}{\sigma\sqrt{T}}\right)$$

- So only generate values u on the interval $[u_{\min}, 1]$, then multiply resulting expectation by $1 - u_{\min}$ to account for the fact that we have only generated $1 - u_{\min}$ of the possible samples.
- We know the other samples would have given 0 for the option payoff.

MATLAB implementation of Importance Sampling

```
function [price,error] = callImportance( K,T, ...
                          S0,r,sigma, ...
                          nPaths )

logS0 = log(S0);
% Generate random numbers u on the interval [lowestU,1]
lowestU = normcdf( (log(K)-logS0 - (r-0.5*sigma^2)*T)/(sigma*sqrt(T)) );
u = rand(nPaths,1)*(1-lowestU)+lowestU;

% Now generate stock paths using norminv( u ). lowestU was chosen
% so that the lowest possible stock price obtained is K. Note that
% we are only considering a certain proportion of possible stock prices
logST = logS0 + (r-0.5*sigma^2)*T + sigma*sqrt(T)*norminv(u);
ST = exp( logST );
discountedPayoff = exp(-r*T)*(ST-K);

% Since we only simulate a certain proportion of prices, the true
% epectation of the final option value must be weighted by proportion
proportion = 1-lowestU;
price = mean(discountedPayoff)*proportion;
error = std(discountedPayoff)*proportion/sqrt(nPaths);
```

Importance Sampling Results

- Parameters: $S_0 = 100$, $K = 200$, $\sigma = 0.2$, $r = 0.14$, $T = 1$, $n = 1000$.
- Note that this is far out of the money, so naive Monte Carlo will perform badly.
- Results:

Method	Price	Standard Error
Black–Scholes Formula	0.02241	
Naive Monte Carlo	0.05960	0.03469
Importance Sampling	0.02122	0.00066

- Conclusions: Importance Sampling is more difficult to implement than antithetic sampling, but can produce excellent improvement for far out of the money options

The Control Variate Method - Idea

- Suppose that we wish to price a Knock Out option
- We have an analytic formula for the price of a Call Option with the same strike.
- Maybe, rather than pricing a Knock Out option directly, it would be a better idea to estimate the difference between the price of a Knock Out option and the price of the Call Option using Monte Carlo instead.

$$\begin{aligned}\text{Price of Knockout Option} \approx \; & \text{Price of Call Option} \\ & + \text{Estimate of difference} \qquad (1)\end{aligned}$$

- Because the difference is probably smaller than the price we're trying to estimate, the variability in a Monte Carlo estimate of the difference is probably lower than the variablility in a Monte Carlo estimate of the price.

Control Variate - Example that proves it can work

- Consider the extreme case of pricing a knock out option where the barrier is so high it will very rarely be hit.
- In the control variate method, we will estimate that the difference between the call price and the knock-out option price is zero even if we use a tiny sample (e.g. a sample of one).
- The control variate method will converge to the exact answer immediately.
- The naive method will be no more accurate than pricing a call by Monte Carlo, so only converges slowly.

The Control Variate method

- Suppose we have a random variable M with $E(M) = \mu$ and wish to find μ.
- Suppose we have another random variable T with $E(T) = \tau$ with τ known.
- Define $M^* = M + c(T - \tau)$. $E(M^*) = \mu$ too for any $c \in \mathbb{R}$. Our previous example was the special case when $c = -1$.
- $\operatorname{Var}(M^*) = \operatorname{Var}(M) + c^2 \operatorname{Var}(T) + 2c \operatorname{Cov}(M, T)$
- Choose c to minimize this
$$c = \frac{-\operatorname{Cov}(M, T)}{\operatorname{Var}(T, T)}$$
- $$\operatorname{Var}(M^*) = (1 - \rho^2) \operatorname{Var}(M)$$
  where ρ is the correlation between M and T.

Control Variate method, worked example

- Let us price a Call Option by Monte Carlo
- We expect the price of a Call Option to be correlated with the price of the stock, so let's use the stock price as our control variate.

```
function [price,errorEstimate, c] = callControlVariate( K,T, ...
                         S0,r,sigma, ...
                         nPaths, ...
                         c)

% Usual pricing code
logS0 = log(S0);
epsilon = randn( nPaths,1 );
logST = logS0 + (r-0.5*sigma^2)*T + sigma*sqrt(T)*epsilon;
ST = exp( logST );
discountedPayoffs = exp(-r*T)*max(ST-K,0);

% Standard formula for control variate method
m = discountedPayoffs;
t = exp(-r*T)*ST;
tau = S0;
covMatrix = cov(m,t);
if nargin<7
    c = -covMatrix(1,2)/covMatrix(2,2);
end
mStar = m + c*(t-tau);

% Result
price = mean(mStar);
errorEstimate = std(mStar)/sqrt(nPaths);
```

Control Variate Results

- Parameters: $S0 = 100$, $K = 100$, $\sigma = 0.2$, $r = 0.14$, $T = 1$, $n = 1000$.
- Results:

Method	Result	Standard Error
Black–Scholes Formula	15.721	
Naive Monte Carlo	16.263	0.564
Control variate	15.723	0.137

- Note, to compute the error I fixed c and then re-ran to compute the same error as I was concerned using the same data to find c and estimate error may lead to bias.
- Conclusions: The control variate technique is easy to implement. It can produce significant improvements in the Monte Carlo price.

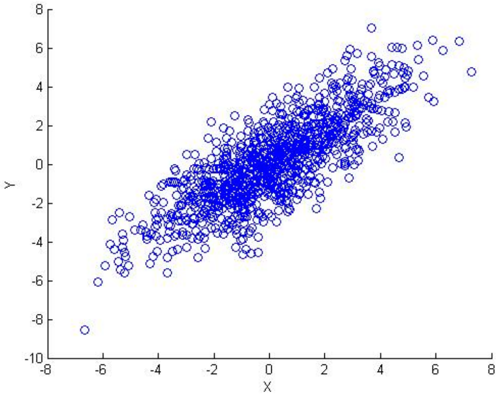
Simulating more interesting stochastic processes

Summary so far

- Simulating the Black–Scholes model has already given some interesting results
- Simulating in the ______-measure allows us to price derivatives
- Simulating in the ______-measure allows us to test trading strategies

Generating correlated random variables

Correlated normally distributed random variables



Pseudo square root

Definition

If Σ is a positive definite symmetric matrix, then a matrix satisfying

$$AA^T = \Sigma$$

is called a pseudo square root of Σ.

Lemma

Given a pseudo square root A of Σ then if X is a vector of independent normally distributed random variables with mean 0 and standard deviation 1 then AX is a multivariate normal variable with mean 0 and covariance Σ.

Proof, part 1

Let $Y_i = \sum_{a=1}^{n} A_{ia} X_a$. Then since $E(X_a X_b) = 1$ if $a = b$ and 0 otherwise we compute:

$$\begin{aligned}
E(Y_i Y_j) &= E\left((\sum_{a=1}^{n} A_{ia} X_a)(\sum_{b=1}^{n} A_{jb} X_b) \right) \\
&= \sum_{a=1}^{n} \sum_{b=1}^{n} A_{ia} A_{jb} E(X_a X_b) \\
&= \sum_{a=1}^{n} A_{ia} A_{ja} \\
&= (AA^T)_{ij}.
\end{aligned}$$

Which shows that the covariance matrix of the Y_i is $AA^T = \Sigma$. The mean of Y_i is zero since the mean of X_a is zero for each a.

Proof, part 2

The density of $\tilde{X}_i$ is

$$\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}.$$

Since the X_i are independent, the joint density of the random vector X is given by

$$(2\pi)^{-\frac{n}{2}} e^{-\frac{1}{2}x^T x}.$$

for $x \in \mathbb{R}^n$. $Y = AX$, so $A^{-1}Y = X$. Hence by the transformation rule for random vectors, Y has distribution

$$(2\pi)^{-\frac{n}{2}} \det(A^{-1}) e^{-\frac{1}{2}(A^{-1}y)^T(A^{-1}y)} = (2\pi)^{-\frac{n}{2}} \det(\Sigma)^{-\frac{1}{2}} e^{-\frac{1}{2}(y^T\Sigma^{-1}y)}$$

which by definition is a multivariate normal distribution with mean 0 and covariance matrix Σ.

Cholesky Decomposition

Theorem

If Σ is a symmetric positive definite matrix then there exists a unique lower triangular matrix L with $\Sigma = LL^T$ and positive diagonal.

Definition

L is called the *Cholesky decomposition* of Σ.

Because L is lower triangular, we can write the equation $LL^T = \Sigma$ out in detail as:

$$\begin{pmatrix} a_{11} & 0 & 0 & \dots & 0 \\ a_{21} & a_{22} & 0 & \dots & 0 \\ a_{31} & a_{32} & a_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} a_{11} & a_{21} & a_{31} & \dots & a_{n1} \\ 0 & a_{22} & a_{32} & \dots & a_{n2} \\ 0 & 0 & a_{33} & \dots & a_{n3} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & a_{nn} \end{pmatrix} = \Sigma$$

Proof continued

Take the lower triangular part of both sides. This gives $\frac{n(n-1)}{2}$ equations in the same number of unknowns. The first row gives:

$$a_{11}^2 = \Sigma_{11}$$

We can now solve for a unique positive a_{11}. The next row gives:

$$a_{21}a_{11} = \Sigma_{21}$$

$$a_{21}^2 + a_{22}^2 = \Sigma_{22}$$

We solve the first for a_{21}. Now we can read off the unique positive a_{22}. The third row gives:

$$a_{31}a_{11} = \Sigma_{31}$$

$$a_{31}a_{21} + a_{32}a_{22} = \Sigma_{32}$$

$$a_{31}^2 + a_{32}^2 + a_{33}^2 = \Sigma_{33}$$

Proof continued

We solve for a_{31} then a_{32} then a_{33}.

- Proceeding in this way gives an algorithm for computing the Cholesky decomposition.
- To compute all the a_{ij} will take $O(i)$ computations for each i (this is the number of coefficients in the equations we write down). There are n^2 coefficients to calculate. So the algorithm will take $O(n^3)$ steps.
- Note that a complete proof requires additionally showing that that Σ being positive definite means the quadratics we solve have real solutions, we'll skip this detail.

Summary

- If we can find a pseudo square root A of a covariance matrix Σ we can generate normally distributed random numbers with covariance matrix Σ by simulating a vector of independent standard random normal variables X and then computing AX.
- We can find a pseudo square root using Cholesky decomposition.
- A positive definite symmetric matrix has many pseudo square roots. Another way to find one is by diagonalizing the matrix.

Exercises

★ Use matlab's `chol` function to find the Cholseky decomposition of

$$\begin{pmatrix} 3 & 1 \\ 1 & 2 \end{pmatrix}$$

★ Use matlab to plot a scatter plot of 10000 points (X, Y) where X and Y are normally distributed with covariance matrix

$$\begin{pmatrix} 3 & 1 \\ 1 & 2 \end{pmatrix}$$

Exercises continued

★ X is normally distributed with mean 5 and standard deviation 3 and Y is normally distributed with mean 7 and standard deviation 1 and if X and Y have correlation $\rho = 0.5$. Generate a sample of points (X, Y) matching these properties. How have you tested your answer?

★ What is the transformation matrix B that reverses the order of the coordinates x_1, x_2, x_3? What is BB^T? Use this to find a pseudo square root of the matrix:

$$\begin{pmatrix} 5 & 1 & 1 \\ 1 & 6 & 1 \\ 1 & 1 & 4 \end{pmatrix}$$

which is not upper triangular

Exercises continued

★ Write a function `randnMultivariate(omega,n)` which generates `n` samples from a multivariate normal distribution with covariance matrix omega.

★ Compute the Cholesky decomposition of

$$\begin{pmatrix} 3 & 1 \\ 1 & 2 \end{pmatrix}$$

by hand.

Solving SDEs numerically

Reference

Kloeden and Platen “Numerical Solution of Stochastic Differential Equations".
We won’t give proofs.

Simulating Correlated Brownian Motion

- d-dimensional Brownian motion with correlation matrix P is defined to be a Markov process whose increments over time δt are independent random vectors which are normally distributed with covariance matrix $P\delta t$ and mean 0.
- Take A to be a pseudo-square root of P, so $P = AA^T$.
- Generate X_t by the difference equation:

$$X_{t+\delta t} = X_t + A\sqrt{\delta t}\epsilon$$

- Then X_t simulates Brownian motion with correlation matrix P.

Problem setting

- 1 dimensional stochastic differential equation

$$\mathrm{d}X_t = a(X,t)\mathrm{d}t + b(X,t)\mathrm{d}W_t$$

- n dimensional stochastic differential equation

$$\mathrm{d}\mathbf{X}_t = \mathbf{a}(\mathbf{X},t)\mathrm{d}t + \mathbf{b}(\mathbf{X},t)\mathrm{d}\mathbf{W}_t$$

  where

$$\mathbf{X}_t \in \mathbb{R}^n$$

$$\mathbf{a}(\mathbf{X},t) \in \mathbb{R}^n$$

$$\mathbf{b}(\mathbf{X},t) \text{ is an } n \times d \text{ matrix}$$

  $\mathbf{W}_r$ is a d-dimensional vector of correlated Brownian motions, with correlation matrix P.

In either case we also have an initial condition X_0.

Notation

- We will drop the bold face for vectors and matrices, the formulae are essentially the same in one or more dimensions.
- We will choose a time step δt and will find approximate solutions by discretization.
- We will write difference equations for our approximations. Time point i corresponds to the time $i\delta t$.
- If W_t is a Brownian motion that we have been given then we define

$$\delta W_i = W_{i\delta t} - W_{(i-1)\delta t}$$

so δW_i is a random variable.

Euler scheme

The Euler scheme for solving the SDE is to define

$$\tilde{X}_i = \tilde{X}_{i-1} + a(X_{i-1}, t)\delta t + b(X_{i-1}, t)\delta W_i$$

so each $\tilde{X}_i$ is a random variable determined by the δW_j with $j \leq i$. We claim that (in a sense to be explained later) the $\tilde{X}_i$ are a good approximation to $X_{i\delta t}$.

Application

To simulate the values X_i, at each time i independently generate a d-dimensional vector of normally distributed ϵ_i with mean 0 and standard deviation 1 and correlation matrix P.
We can do this using the Cholesky decomposition.
Define:

$$\tilde{X}_i = \tilde{X}_{i-1} + a(X_{i-1}, t)\delta t + b(X_{i-1}, t)(\sqrt{\delta t})\epsilon_i$$

NOTE THE $\sqrt{\delta t}$!
Note the slight distinction between:

- Solving the SDE when we are given values of W_t over time.
- Simulating the stochastic process, where we run many simulations and generate our own values of ϵ_i.

Theorem

Suppose that:

$$|a(x,t) - a(y,t)| + |b(x,t) - b(y,t)| < K_1|x - y|$$

and

$$|a(x,t)| + |b(x,t)| < K_2(1 + |x|)$$

and

$$|a(x,s) - a(x,t)| + |b(x,s) - b(x,t)| < K_3(1 + |x|)|s - t|^{-\frac{1}{2}}$$

for some constants K_1, K_2, K_3 and all s, t, x, y. Then

$$E(|X_T - \tilde{X}_{(T/\delta t)} \leq K_4 \delta t^{\frac{1}{2}}$$

for some constant K_4.

i.e. we have convergence in expectation.

Application

Corollary

Under the same conditions, our simulation converges in distribution.

- The rate of convergence $\delta t^{\frac{1}{2}}$ is very slow
- The conditions are very stringent (e.g. linear growth)

Example

Example

For the process

$$\mathrm{d}X_t = a\mathrm{d}t + b\mathrm{d}W_t$$

with a and b constants then the solution is Euler scheme is exact. Note this is elementary and does not use general result on convergence.

In general if a and b are slowly varying we can expect that the Euler scheme will be reasonably accurate.

Numerical test

We won't prove the theorem. But we can check it is true. We want to see if:

$$E(|X_T - \tilde{X}_T|) \leq K_4 \delta t^{\frac{1}{2}}$$

for an example process. Let's find an interesting process we can solve. Take

$$X_t = \sin(W_t)$$

so

$$\mathrm{d}X_t = -\frac{1}{2} X_t \mathrm{d}t + \sqrt{1 - X_t^2}\, \mathrm{d}W_t$$

Solving the SDE by the Euler scheme

```
function [ X ] = simulateSinEuler( X0, dW, dt, nSteps )
% Simulate the following process
% dX = -1/2 X + sqrt(1-X^2) dW
% Note this is obtained by taking the sin of brownian motion

currX = X0;
nPaths = size( dW, 1);
X = zeros(nPaths, nSteps );
for i=1:nSteps
    currDW = dW(1:end,i);
    X(1:end,i) = currX - 0.5*currX*dt + sqrt(1-currX.^2).* currDW;
    currX = X(1:end,i);
end

end
```

Computing the error

```
dW = randn( nPaths, nSteps(j) )*sqrt(dt);

exactPaths = sin(X0+cumsum(dW,2));

eulerPaths = simulateSinEuler( X0, dW, dt, nSteps(j) );
eulerErrors = abs(eulerPaths(1:end,end)-exactPaths(1:end,end));
eulerError(j) = ninetyPercentConfidence(eulerErrors);
```

- `ninetyPercentConfidence` is a helper function which finds the upper bound on a ninety percent confidence interval for the mean.
- If we generate a log-log plot of the upper level of the confidence interval against the number of steps, what should we expect to see?
- (Shown in slide at end of lecture).

Application

By simulating stock price processes in the risk neutral measure we can compute option prices.

- Black Scholes model - no need, we have an exact simulation method
- Local volatility model - the parameters μ and σ vary with S and t
- Heston model - the volatility also follows a stochastic process.

Heston model with $r = 0$

Suppose that in the $\mathbb{Q}$ measure the stock price and volatility obey:

$$\mathrm{d}S_t = \sqrt{v_t} S_t \mathrm{d}W_t^1$$

$$\mathrm{d}v_t = \kappa(\theta - v_t)\mathrm{d}t + \xi\sqrt{v_t}\mathrm{d}W_t^2$$

where $\mathrm{d}W_t^1$ and $\mathrm{d}W_t^2$ are Brownian motions with correlation ρ

- θ is the long run variance
- κ is the mean reversion rate
- ξ is the volatility of volatility

Require $2\kappa\theta > \xi^2$ to keep volatility positive.

- $r = 0$ so we require that S is a martingale for this to be a valid $\mathbb{Q}$ measure model. This is why there is no drift term.
- In Black–Scholes model there is a unique compatible $\mathbb{Q}$ model for a given $\mathbb{P}$. This isn't true in general, so one usually takes $\mathbb{Q}$ as given.

The volatility smile

If we simulate $\mathbb{Q}$ measure stock prices in the Heston model and use this to compute risk neutral prices for options, will this "explain" the volatility smile?

- (Question: what is implied volatility? What is the volatility smile?)

Implied volatility

Definition

Given the market price of a European put or call option, the implied volatility is the value that you must put into the Black–Scholes formula to get that market price.

Monte carlo pricing code

Our pricing code looks like this:

```
function ret = priceByMonteCarlo(...)
    paths = generatePricePaths(...);
    payoffs = computeOptionPayoffs(...);
    ret = mean(payoffs)*exp(-r*T);
end
```

All we need to do is to change this to use the Heston model to generate price paths.

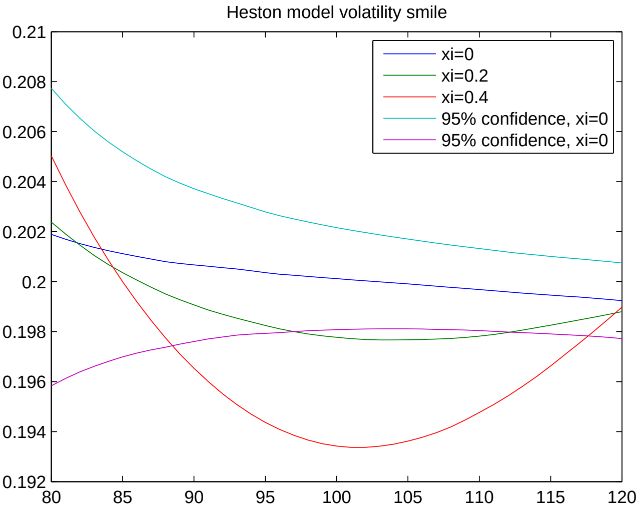
```
function [ prices, variances ] = generatePricePathsHeston( ...
    S0, v0, ...
    kappa, theta, xi, rho, ...
    T, nPaths, nSteps)
%GENERATEPRICEPATHSHESTON Generate price paths according to the
%    Heston model
prices = zeros( nPaths, nSteps );
variances = zeros( nPaths, nSteps );

currS = S0;
currv = v0;
dt = T/nSteps;
for i=1:nSteps
    epsilon = randnMultivariate( [1 rho; rho 1], nPaths  );
    dW1 = epsilon(1,:)*sqrt(dt);
    dW2 = epsilon(2,:)*sqrt(dt);
    currS = currS + sqrt( currv).* currS .* dW1';
    currv = currv + kappa*(theta - currv)*dt + xi*sqrt( currv).* dW2';
    currv = abs( currv ); % Forcibly prevent negative variances
    prices( :, i) = currS;
    variances( :, i) = currv;
end
```

I ran the simulation with the following parameters

- `nPaths=100000`
- `nSteps=50`
- $T = 1$
- $S_0 = 1$
- $\kappa = 2$
- $\theta = 0.04$
- $v_0 = 0.04$
- $\rho = 0$

I then used three different values of ξ.

I plotted the Black–Scholes implied volatility for a number of strikes


Heston model volatility smile
0.21
0.208
0.206
0.204
0.202
0.2
0.198
0.196
0.194
0.192
80
85
90
95
100
105
110
115
120
xi=0
xi=0.2
xi=0.4
95% confidence, xi=0
95% confidence, xi=0

- When $\xi = 0$ this becomes equivalent to the Black Scholes model. So in theory the implied volatility should be a constant equal to 0.2.
- As well as computing monte carlo prices, I've plotted error bounds for the computation when $\xi = 0$. The Black Scholes prediction fits within the error bounds as one would hope.
- Other values of ξ do give rise to a smile.

Other applications

- Repeat the delta hedging simulation we did last week, but generate stock prices using different _____-measure models. We can use this to see how well delta hedging performs with e.g. fat tailed stock prices.
- Repeat the VaR Monte Carlo simulations we'll carry out next week with more interesting models
- etc. etc.

The Milstein Scheme

- The Euler scheme isn't the end of the story.
- The Milstein scheme for:

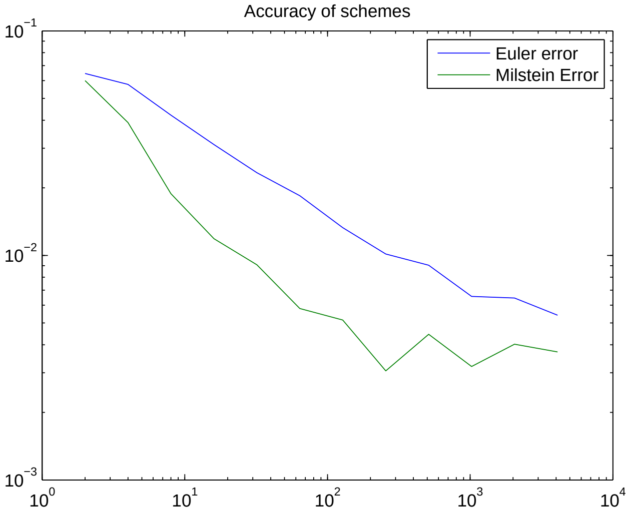
$$\mathrm{d}X_t = a(X_t, t)\mathrm{d}t + b(X_t, t)\mathrm{d}W_t$$

is to take

$$\tilde{X}_i = \tilde{X}_{i-1} + a\delta t + b\delta W_t + \frac{1}{2} b \frac{\partial b}{\partial x} \left((\delta W_t)^2 - \delta t \right)$$

- This is Euler scheme plus one more term
- Under certain bounds on the coefficients, converges in expectation at rate $O(\delta t)$
- n-d versions exist but are more complex.

Plot of errors of Euler and Milstein for process $\sin(W_t)$



Exercises

★ Simulate the process

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

using the Euler scheme and find the exact solution too. Use this to generate a log-log plot of errors for the Euler scheme.

★ Simulate the Vasicek interest rate model

$$dr_t = a(b - r_t)dt + \sigma dW_t$$

using the Euler scheme. Generate plots of interest rate paths with varying parameters so you get a feel for this kind of model. How could you simulate the Vasicek model without using the Euler scheme? (HINT: The increments of the Vasicek model over any time period are known to be normally distributed and there are formulae for their mean and variance)

★ Modify the delta hedging code from last week so that one still delta hedges as though one believed the Black–Scholes model was true, but in fact the interest rates are stochastic and follow the Vasicek model. How does the delta hedging strategy perform?