Goddard Consulting

Modeling. Simulation. Data Analysis. Visualization.

Home
Financial Engineering
Option Pricing
Econometrics
Portfolio Optimization
Technical Analysis
Control System Design
PID Controllers
Kalman Filters
Software Tutorials
MATLAB
Simulink
VBA
C++
Services
Financial Modeling
Control System Design
Generic Modeling
Auditing Code
Software Training
About Us
Contact Us

Jarrow-Rudd in MATLAB

This tutorial presents MATLAB code that implements the Jarrow-Rudd version of the binomial model (also known as the equal-probability model) as discussed in the Alternative Binomial Models tutorial.

The code may be used to price vanilla European or American, Put or Call, options. Given appropriate input parameters a full lattice of prices for the underlying asset is calculated, and backwards induction is used to calculate an option price at each node in the lattice. Creating a full lattice is wasteful (of memory and computation time) when only the option price is required. However the code could easily be modified to show how the price evolves over time in which case the full lattices would be required.

Note that the primary purpose of the code is to show how to implement the Jarrow-Rudd binomial model. The code contains no error checking and is not optimized for speed or memory use. As such it is not suitable for inclusion into a larger application without modifications.

A Pricing Example

Consider pricing a European Call option with the following parameters, $X = \$60$, $S_0 = \$50$, $r = 5\%$, $\sigma = 0.2$, $\Delta t = 0.01$, $N = 100$.

The Black-Scholes price for this option is *$1.624*.

A MATLAB function called **binPriceJR** is given below. The following shows an example of executing **binPriceJR** (and pricing the above option) in MATLAB,

```
>> oPrice = binPriceJR(60,50,0.05,0.2,0.01,100,'CALL',false)
oPrice =
      1.639
```

If the number of time steps is doubled then

```
>> oPrice = binPriceJR(60,50,0.05,0.2,0.005,200,'CALL',false)
oPrice =
      1.664
```

Note that for this particular option a larger number of time steps (i.e. a finer grid of points) does not lead to a more accurate solution. This is not necessarily unexpected due to the non-smooth convergence (to the Black-Scholes price) that is exhibited by many binomial model formulations.

The Lieson-Reimer method discussed in the Alternative Binomial Models tutorial was formulated to overcome this issue. The binomial tree generated by the Lieson-Reimer method guarantees that a larger number of points will generate a more accurate solution.

MATLAB Function: binPriceCRR

```
function oPrice = binPriceJR(X,S0,r,sig,dt,steps,oType,earlyExercise)
% Function to calculate the price of a vanilla European or American
% Put or Call option using an equal-probability (Jarrow-Rudd) binomial tree.
%
% Inputs: X - strike
%       : S0 - stock price
%       : r - risk free interest rate
%       : sig - volatility
%       : dt - size of time steps
%       : steps - number of time steps to calculate
%       : oType - must be 'PUT' or 'CALL'.
%       : earlyExercise - true for American, false for European.
%
% Output: oPrice - the option price
%
% Notes: This code focuses on details of the implementation of the equal
%        probability algorithm.
%        It does not contain any programatic essentials such as error
%        checking.
%        It does not allow for optional/default input arguments.
%        It is not optimized for memory efficiency or speed.

% Author: Phil Goddard (phil@goddardconsulting.ca)
% Date  : Q4, 2007

% Calculate the equal probability model parameters
p = 0.5;
u = exp((r-sig*sig/2)*dt + sig*sqrt(dt));
d = exp((r-sig*sig/2)*dt - sig*sqrt(dt));

% Loop over each node and calculate the JR underlying price tree
priceTree = nan(steps+1,steps+1);
priceTree(1,1) = S0;
for idx = 2:steps+1
    priceTree(1:idx-1,idx) = priceTree(1:idx-1,idx-1)*u;
    priceTree(idx,idx) = priceTree(idx-1,idx-1)*d;
end
```

```
% Calculate the value at expiry
valueTree = nan(size(priceTree));
switch oType
    case 'PUT'
        valueTree(:,end) = max(X-priceTree(:,end),0);
    case 'CALL'
        valueTree(:,end) = max(priceTree(:,end)-X,0);
end

% Loop backwards to get values at the earlier times
steps = size(priceTree,2)-1;
for idx = steps:-1:1
    valueTree(1:idx,idx) = ...
        exp(-r*dt)*(p*valueTree(1:idx,idx+1) ...
        + (1-p)*valueTree(2:idx+1,idx+1));
    if earlyExercise
        switch oType
            case 'PUT'
                valueTree(1:idx,idx) = ...
                    max(X-priceTree(1:idx,idx),valueTree(1:idx,idx));
            case 'CALL'
                valueTree(1:idx,idx) = ...
                    max(priceTree(1:idx,idx)-X,valueTree(1:idx,idx));
        end
    end
end

% Output the option price
oPrice = valueTree(1);
```

Back To Top | Option Pricing