Pricing Derivatives with Stochastic Volatility and Stochastic Interest Rates

Jennifer Voitle
updated February, 2007
Jennifer@TreasuryFinance.com

Reference:
"Good-deal option price bounds with stochastic volatility and stochastic interest rate", John H. Cochrane and Jesus Saa Requejo, 1999
http://faculty.chicagogsb.edu/john.cochrane/research/Papers/stochvol.pdf

Description of Model
The following application prices a derivative instrument based on a standard stochastic volatility model with an instantaneous stochastic interest rate and correlated shocks after Professor John Cochrane's paper "Beyond Arbitrage: 'Good Deal' Asset Price Bounds in Incomplete Markets", The University of Chicago.) The underlying idea is that there are illiquid instruments for which we may not be able to find a replicating portfolio for pricing, but we can in some cases at least hope to find bounds on the price. The model may also be used to price standard European vanilla puts and calls by appropriate assignment of the constants below.

In this model, we solve the simultaneous set of SDEs

$$\frac{dS}{S} = \mu_s + \sqrt{V}\, dz_s$$
$$dV = \alpha_v\, (\bar{r} - r)\, dt + \beta_s \sqrt{V}\, dz_s + \beta_v \sqrt{V}\, dz_v$$
$$dr = \alpha_r\, (\bar{r} - r)\, dt + \sigma_s \sqrt{r}\, dz_s + \sigma_v \sqrt{r}\, dz_v + \sigma_r \sqrt{r}\, dz_r$$

where dzs, dzv and dzr are independent Brownian motions. We assume that the option price C = C(S,V,r,t) where S is the value of the underlying, which may be assumed to be a stock.

This leads to the system of PDEs

$$\frac{\partial C}{\partial t}+\frac{\partial C}{\partial s}Sr+\frac{\partial C}{\partial V}\left[\alpha_v(\overline{V}-V)-\beta(\mu_s-r)_s\right]+\frac{\partial C}{\partial r}\left[\alpha_r(\overline{r}-r)-(\mu_s-r)\sigma_s\sqrt{\frac{r}{V}}\right]$$
$$+\frac{1}{2}\left[\frac{\partial^2 C}{\partial S^2}S^2V+\frac{\partial^2 C}{\partial V^2}(\beta_s^2+\beta_v^2)V+\frac{\partial^2 C}{\partial r^2}(\sigma_s^2+\sigma_v^2+\sigma_r^2)r\right]+\frac{\partial^2 C}{\partial S\partial V}\beta_s SV+\frac{\partial^2 C}{\partial S\partial r}S\sqrt{rV}$$
$$+\frac{\partial^2 C}{\partial V\partial r}(\beta_s\sigma_s+\beta_v\sigma_v)\sqrt{rV}-rC=\pm\sqrt{A^2-\frac{(\mu_s-r)^2}{V}}\sqrt{\left(\frac{\partial C}{\partial V}\beta_v\sqrt{V}+\frac{\partial C}{\partial r}\sigma_v\sqrt{r}\right)^2+\left(\frac{\partial C}{\partial r}\right)^2\sigma_r^2 r}$$

An implicit finite difference model was written using *Mathematica* to solve the above equation. The method used was a three-step, implicit, alternating direction finite difference method using improved stability analysis methods based on the work of Brian. A specific example of a European call maturing in five months, having Smax = $100, X = $50, m = 8 stock steps, n = 5 time steps, interest rate varying between 1% and 11% and volatility ranging between 10% and 30% was priced and compared to the Black-Scholes price. Good agreement was obtained (see graph following.) The model is general in that it will also work with constant interest rate and/or volatility, if needed, for exact test case comparison to BS methodology.

The model outputs a matrix of prices on the stock, time, interest rate and volatility grid as well as correlation matrix plots, 3D surface plots, contour and line plots of specified variables. The model may also be used to construct bounds on prices where replicating portfolios do not exist and we thus could not otherwise obtain a price.

Methodology

The security (which need not be restricted to equity) is priced with the boundary condition C(ST,V,r,T) = max(ST-K,0); C(0,V,r,t) = 0 which is the condition for a call.

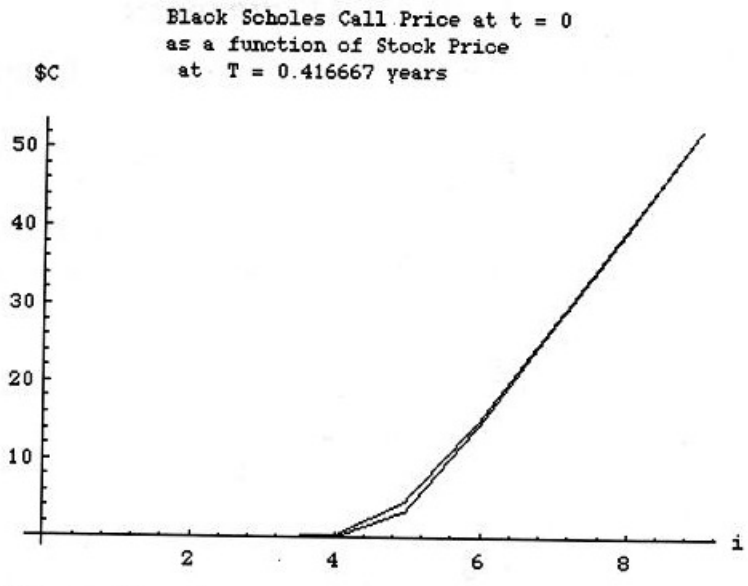


Figure 1: Comparison of Black Scholes Price to Cochrane Pricing Model

Finite Difference Valuation of Contingent Claims

```
Off[General::spell]
Off[General::spell1]
Clear[u,BlackScholes,f, nodalpoints]
Clear[u,i,j,k,a,b,c,equation]
<<Statistics`ContinuousDistributions`

Clear[MatrixSolve]

MatrixSolve[InteriorPoints_List,newres_List] :=
   Module[{i,j},
      eqnlist = {}; (* initialize list to null *)
      zerolist = Table[0,{Length[InteriorPoints] } ] ;
      (* make list of zeros for comparison *)
      Do[
         Do[
            If[SameQ[  (* no match ... write a zero *)
               Complement[Coefficient[equationlist[[j]],  InteriorPoints[[i]] ],
zerolist],{}], (* then *)
                  AppendTo[eqnlist,0],  (* else there is a matching coefficient *)
                  AppendTo[eqnlist,Complement[
                     Coefficient[equationlist[[j]],InteriorPoints[[i]]],zerolist] ]],
                        {i,Length[InteriorPoints]} ],
                           {j,Length[InteriorPoints]}];

 (* now treat the RHS of the matrix by picking off constants from the matrix and
subtracting from RHS *)

 newlist = {};
 Do[  rhssum=0;
    Do[
       If[  SameQ[  (* no match ... write a zero *)
               Complement[Coefficient[ equationlist[[j]], InteriorPoints[[i]] ],
zerolist], {} (* end SameQ test *)],

              (* then *)
              rhssum = rhssum + equationlist[[j]] [[i]] ],
               {i,Length[InteriorPoints]} ];
                  AppendTo[newlist,rhssum],{j,Length[InteriorPoints]}];

rhs = Table[Last[equationlist[[i]] ], (* the last element on the list is either a
constant or some coefficient of an unknown *)
   {i,Length[InteriorPoints]}] /. Times[x_,y_] :> 0;

   rhs = rhs - newlist;

   inversematrix = Inverse[Partition[Flatten[eqnlist],Length[InteriorPoints]]];
   (* return *)
   solution = inversematrix.rhs
   ]
```

Parameter and Boundary Definitions

```
Clear[BlackScholes]
BlackScholes[St_,X_,rf_,T_,t_,sig_]:=
Module[{},
(* Usage:: BlackScholes[100,100,.05,5/12,1/12,.01] *)
   d1 = (Log[St/X] + (rf + sig^2/2) (T-t))/(sig Sqrt[T-t]);
   d2 = d1 - sig Sqrt[T-t];
   (* return *)
   St CDF[NormalDistribution[0,1],d1]-
     X Exp[-rf (T-t)] CDF[NormalDistribution[0,1],d2] ]
```

In the following, f[i,j,k,n] is the value of the contingent claim C{S,V,r,t] at underlying price j dS, volatility j dv, interest rate k dr and time n dt.

```
X = 50; (* strike price *)
m = 8; (* number of stock subintervals *)
NumberofTimeSteps = 5;
nmax = 4 NumberofTimeSteps + 1; (* boundary condition on time *)
volsteps = 4;
intsteps = 4;
Smax = 100.; Smin = 0.;
tmax = 5/12; (* years *)
vmax = 0.3; vmin = 0.1; (* annual volatility, %/year *)
rmax = 0.11; rmin = 0.01; (* interest rate, %/year *)
rf = 1.0484; (* risk free rate *)
betav = 0.5;
alphar = 1.214;
alphav = 1.4;
sigma = 0.1409;
vbar = sigma^2;
deltar = (rmax-rmin)/intsteps;
deltav = (vmax - vmin)/volsteps;
deltaS = (Smax - Smin)/m;
deltat = tmax/nmax;
h = 0.5; (* assumed upper bound on Sharpe ratio *)
A = Sqrt[(1+h^2)]/rf;

rbar = rf;
lambda[1] = deltat/(4 deltav);
lambda[2] = deltat deltar/4;
lambda[3] = Sqrt[A^2 - h^2] betav lambda[1] sigma;
lambda[4] = deltat/(4 deltar);
lambda[6] = lambda[1] (deltav/deltaS)^2;
```

Geometry and Discretization of Domain

The payoff space is subdivided into m+1 underlying prices for S such that dS = Smax/m, and n+1 time values. One of the most difficult aspects of this problem is the proper description of the boundary surface. At least two possibilities suggest themselves:

1) use *Mathematica*'s Table function to build up a list of points or
2) define functions that describe the boundary.

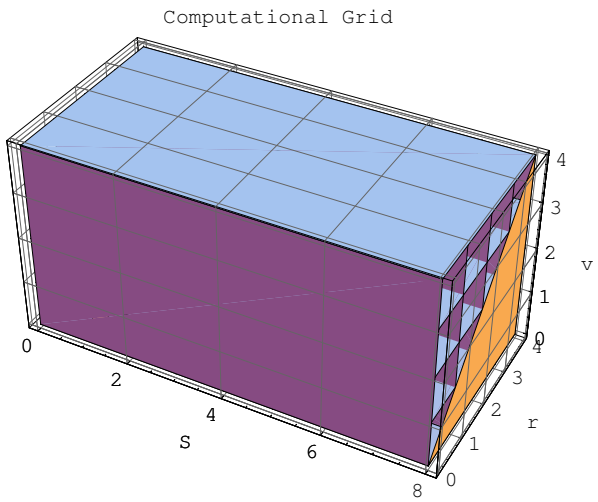
Here, we show the first method since it seems more natural in *Mathematica*. The solution will proceed in the following steps:

1) Discretize the domain
2) Write the finite difference equation (FDE) of the governing PDEs and boundary conditions
3) The FDE is written at each node at which the solution is unknown (that is, all interior points) and
4) The resulting system of equations is solved and plotted.

Discretization

Describing the domain takes some experimentation. We'll use a rectangular domain, m = 8 and n= 5 as stated above. All steps of creating the boundary surface are shown below for illustrative purposes.

```
Clear[list]
nodalpoints = Table[Table[Table[u[i,j,k],{i,0,m}],{j,0,volsteps}],{k,0,intsteps}];
list[1] = Table[
  Polygon[{
     First[nodalpoints[[i,1]] ],First[nodalpoints[[i,volsteps+1]] ],
     Last[nodalpoints[[i,volsteps+1]] ],Last[nodalpoints[[i,1]] ],
        First[nodalpoints[[i,1]] ]} /. u[x_,y_,z_] ->{x,y,z}], {i,1,intsteps+1}];

list[2] = Table[
  Polygon[{Part[nodalpoints[[1,1]],i ],Part[nodalpoints[[1,volsteps + 1]],i ],
     Part[nodalpoints[[intsteps+1,volsteps+1]],i ],Part[nodalpoints[[1,1]],i ],
     Part[nodalpoints[[1,1]],i ]} /. u[x_,y_,z_] ->{x,y,z}], {i,1,m+1}];

list[3] = Table[
  Polygon[{Part[nodalpoints[[1,i]],1 ],Part[nodalpoints[[1,i]],m+1 ],
     Part[nodalpoints[[volsteps+1,i]],m+1 ],Part[nodalpoints[[volsteps + 1,i]],1 ],
     Part[nodalpoints[[1,i]],1 ]} /. u[x_,y_,z_] ->{x,y,z}], {i,intsteps+1}];


Show[Graphics3D[{Flatten[Join[Table[list[i],{i,1,3}] ]],
  EdgeForm[Thickness[0.001] ]},
     Lighting->True,FaceGrids->All,Axes->True,AxesLabel->{"S","r","v"}],
       PlotLabel->"Computational Grid"];
```



Computations

```
Clear[Boundarys]
Boundarys := Join[
   Table[
      Table[
         Table[u[i,j,k],{i,0,m}],{k,0,intsteps}],{j,0,volsteps,volsteps}], (* front
and rear faces *)
         Table[
            Table[
               Table[u[i,j,k],{j,1,volsteps-1}],{i,0,m}],{k,0,intsteps},{intsteps}],
(* top and bottom faces *)
         Table[
            Table[
               Table[u[i,j,k],{j,0,volsteps}],{i,0,m,m}],{k,1,intsteps-1}]]

(* Boundary Conditions *)
   (* Initial Condition *)

u[i_,j_,k_,nmax] := Max[i deltaS - X,0] /; i >= 0 && i <= m;

(* Underlying Boundary Conditions *)

u[0,j_,k_,n_] := 0 (* /; j >= 0 && j <= m; *)

   (* u[m,j_,k_,n_] := Smax - X Exp[-(k deltar + rmin) n deltat] /; t >=0 && i <=n;
   Forward BC converged to by BS, will use actual BS here *)
u[m,j_,k_,n_] := BlackScholes[Smax,X, k deltar+rmin, tmax, n deltat, j deltav + vmin];

  (* Volatility BCs *)

u[i_, 0, k_, n_] := BlackScholes[i deltaS,X, k deltar+rmin, tmax, n deltat,  vmin];
u[i_, volsteps, k_, n_] := BlackScholes[i deltaS,X, k deltar+rmin, tmax, n deltat,
vmax];

(* Interest Rate BCs *)

u[i_, j_, 0, n_] := BlackScholes[i deltaS,X, rmin, tmax, n deltat,   j deltav + vmin];
u[i_, j_, intsteps, n_] := BlackScholes[i deltaS,X, rmax, tmax, n deltat,   j deltav
+ vmin];

(* Finite Difference Equations for Interior Points *)

Do[ (* outer time step loop which takes us over one full time step dt *)
   nn = 4 nnn;
   Clear[a,b,c,equation,i,j,k];(* just in case *)
   a[i_,j_,k_] = j lambda[6] - i k lambda[2];
   b[i_,j_,k_] = 1 - 2 j lambda[6];
   c[i_,j_,k_] = j lambda[6] + i k lambda[2];

 (* equation for first dt/2 time step, implicit in S *)

   equation[i_,j_,k_,n_] =
      (a[i,j,k] u[i-1,j,k,n] + b[i,j,k] u[i,j,k,n] + c[i,j,k] u[i+1,j,k,n] ==
```

```
      (lambda[1](-betav^2 (j) + alphav(vbar-(j deltav + vmin)))- lambda[3])
u[i,j-1,k,n+1] -
      alphar(rbar-((k) deltar+rmin)) lambda[4] u[i,j,k-1,n+1] +
      (2 (lambda[2] k + lambda[1] betav^2 j)+1) u[i,j,k,n+1] +
      (-lambda[1] (betav^2 (j) + alphav(vbar-((j) deltav+vmin))) +
      lambda[3]) u[i,j+1,k,n+1] + alphar(rbar-((k) deltar + rmin)) lambda[4]
u[i,j,k+1,n+1]);

      InteriorPoints =
         Complement[Flatten[nodalpoints],Flatten[Boundarys] ];

res = InteriorPoints /. u[i_,j_,k_] :> equation[i,j,k,nn];

newres = res /. u[i_,j_,k_,z_] :> u[i,j,k];

equationlist = Table[Flatten[newres[[i]] /.
{Equal->List,Plus->List}],{i,Length[InteriorPoints]}];

Unknowns = InteriorPoints; (*MatrixSolve[InteriorPoints,newres] *)

solution = Flatten[Solve[newres,Unknowns] ] /. u[x_,y_,z_] :> u[x,y,z,nn];

(* map the solution back onto the grid *)

MapThread[Set,{Unknowns /. u[x_,y_,z_] :>
u[x,y,z,nn],Table[solution[[i,2]],{i,Length[Unknowns]}]}];

(* now solve the implicit equation in V using these values as inputs, full first dt/2
time step which is implicit in v *)

Clear[a,b,c,equation,newres, i,j,k];(* just in case *)
a[i_,j_,k_] = lambda[1](betav^2 j - alphav(vbar - (j deltav + vmin)))-lambda[3];
b[i_,j_,k_] = 2 lambda[1] betav^2 j + 1;
c[i_,j_,k_] =  lambda[1](betav^2 j + alphav(vbar - (j deltav + vmin)))-lambda[3];

equation[i_,j_,k_,n_] =
      (a[i,j-1,k] u[i,j-1,k,n] + b[i,j,k] u[i,j,k,n] + c[i,j+1,k] u[i,j+1,k,n] ==
      (-j lambda[6] + (i-1) k lambda[2]) u[i-1,j,k,n+1]
      -alphar(rbar-(k-1) deltar) lambda[4] u[i,j,k-1,n+2] +
      2 lambda[6] j u[i,j,k,n+1] +
      (1 + 2 lambda[2] k) u[i,j,k,n+2]+
     (-j lambda[6] - (i+1) k lambda[2]) u[i+1,j,k,n+1]+
     alphar(rbar-((k+1) deltar+rmin)) lambda[4] u[i,j,k+1,n+2]);

res = InteriorPoints /. u[i_,j_,k_] :> equation[i,j,k,nn-1];

newres = res /. u[i_,j_,k_,z_] :> u[i,j,k];

equationlist = Table[Flatten[newres[[i]] /.
{Equal->List,Plus->List}],{i,Length[InteriorPoints]}];
Unknowns = InteriorPoints;
solution = Flatten[Solve[newres,Unknowns] ] /. u[x_,y_,z_] :> u[x,y,z,nn-1];
MapThread[Set,{Unknowns /. u[x_,y_,z_] :>
u[x,y,z,nn-1],Table[solution[[i,2]],{i,Length[Unknowns]}]}];

     (* solve the implicit equation in V using these values as inputs; implicit in r
*)
```

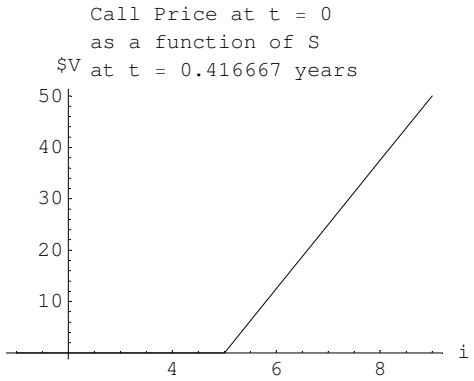
```
Clear[a,b,c,equation,newres, i,j,k];
a[i_,j_,k_] = alphar(rbar - k deltar) lambda[4];
b[i_,j_,k_] = 1;
c[i_,j_,k_] = -alphar(rbar - k deltar) lambda[4];
equation[i_,j_,k_,n_] =
  Simplify[(
  a[i,j,k-1] u[i,j,k-1,n] + b[i,j,k] u[i,j,k,n] + c[i,j,k+1] u[i,j,k+1,n]) == (
    (-j lambda[6] + (i-1) k lambda[2]) u[i-1,j,k,n+2] +
    (lambda[1] (alphav(vbar-(j-1) deltav)-betav^2 (j-1))-lambda[3]) u[i,j-1,k,n+1] +
     2 lambda[6] j u[i,j,k,n+2] +
     2 lambda[1] betav^2 j u[i,j,k,n+1] +
     (1 + 2 lambda[2] k) u[i,j,k,n+3] +
     (lambda[3] - lambda[1] (alphav(vbar-(j+1) deltav)+
     betav^2 (j+1))) u[i,j+1,k,n+1]+
     (-j lambda[6] - (i+1) k lambda[2]) u[i+1,j,k,n+2])];

res = InteriorPoints /. u[i_,j_,k_] :> equation[i,j,k,nn-2];
newres = res /. u[i_,j_,k_,z_] :> u[i,j,k];
equationlist = Table[Flatten[newres[[i]] /.
{Equal->List,Plus->List}],{i,Length[InteriorPoints]}];
Unknowns = InteriorPoints;
solution = Flatten[Solve[newres,Unknowns] ] /. u[x_,y_,z_] :> u[x,y,z,nn-2];
MapThread[Set,{Unknowns /. u[x_,y_,z_] :>
u[x,y,z,nn-2],Table[solution[[i,2]],{i,Length[Unknowns]}]}];

(* now solve last bit by direct evaluation of equation *)

Clear[a,b,c,equation,newres, i,j,k];
a[i_,j_,k_] = 2 (lambda[2] i k - lambda[6] j);
b[i_,j_,k_] = 2 lambda[1](alphav(vbar-j deltav) - betav^2 j)-2 lambda[3];
c[i_,j_,k_] = -2 lambda[4] alphar(rbar-k deltar);

c1[i_,j_,k_] = 4 lambda[1] betav^2 j;
c2[i_,j_,k_] = 4 lambda[6] j;
c3[i_,j_,k_] = 4 lambda[2] k+1;
c4[i_,j_,k_] = 2 lambda[4] alphar(rbar-k deltar);
c5[i_,j_,k_] = -2 lambda[1] (alphav(vbar-j deltav)+betav^2 j) + 2 lambda[3];
c6[i_,j_,k_] = -2(lambda[2] i k + lambda[6] j);

tempres = Flatten[Table[Rule@@{u[i,j,k],a[i-1,j,k] u[i-1,j,k,nn] + b[i,j-1,k]
u[i,j-1,k,nn-1]+ c[i,j,k-1] u[i,j,k-1,nn-2]+
     c1[i,j,k] u[i,j,k,nn-1]+ c2[i,j,k] u[i,j,k,nn]+ c3[i,j,k]
u[i,j,k,nn+1]+c4[i,j,k+1] u[i,j,k+1,nn-2]+c5[i,j+1,k] u[i,j+1,k,nn-1]+c6[i+1,j,k]
u[i+1,j,k,nn]},
     {i,1,m-1},{j,1,volsteps-1},{k,intsteps-1}]];

solution = tempres /. u[x_,y_,z_] :> u[x,y,z,nn-3];

MapThread[Set,{Unknowns /. u[x_,y_,z_] :>
u[x,y,z,nn-3],Table[solution[[i,2]],{i,Length[Unknowns]}]}],
{nnn,NumberofTimeSteps,1,-1}]
```

Plots of Stochastic Volatility, Stochastic Interest Rate Results

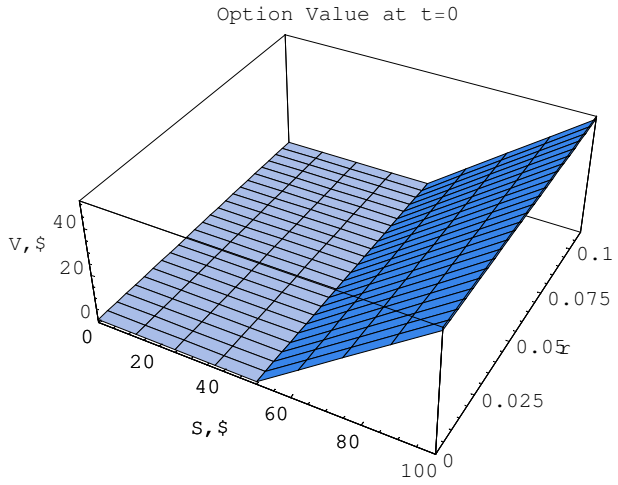
```
nmax
```

21

```
nodalpointszvals = nodalpoints /. u[x_,y_,z_] :> u[x,y,z,nmax];
data = Partition[Flatten[nodalpointszvals],m+1];
ListPlot[nodalpointszvals[[5]][[3]],PlotJoined->True,
PlotLabel->"Call Price at t = 0\nas a function of S\nat t = "<>ToString[N[tmax]]<>"
years",AxesLabel->{"i","$V"}];
```

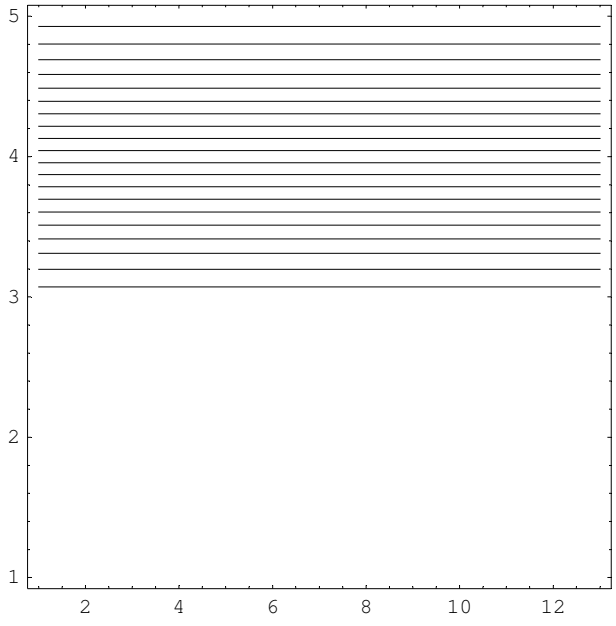


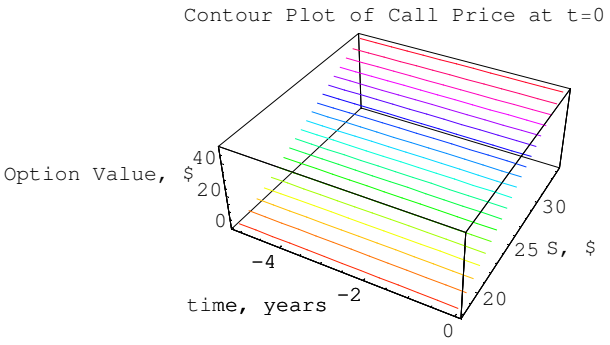

```
ListPlot3D[data,Axes->True,AxesLabel->{"S,$","r","V,$"},PlotLabel->"Option Value at
t=0",MeshRange->{{Smax,0},{0,rmax}}];
```



Code to Generate 3D Contour Plot

```
Clear[scale,data]
plottitle="Contour Plot of Call Price at t=0";
labels = {"time, years","S, $","Option Value, $"};
TempData=Partition[Flatten[nodalpointszvals],m+1];
data =
Table[TempData[[i,j]],{i,1,First[Dimensions[TempData]],2},{j,1,Last[Dimensions[TempDat
a]],2}];
{zmin,zmax}={0,X};
xmin=ymin=0;xmax=tmax;ymax=Smax;
xgrid=deltat;ygrid=deltaS;

scale[z_]:= (z-zmin)/(zmax-zmin);
ydist[i_]:= (ymax-ymin)/(ygrid-1) (i-1)+ymin;
xdist[i_]:= (xmax-xmin)/(xgrid-1) (i-1)+xmin;

contourplot=ListContourPlot[Transpose[data],ContourShading->False,Contours->20];
lines1=Graphics[contourplot];
lines2=Graphics[Cases[lines1,_Line,{3}]];
h = Interpolation[Partition[Flatten[Table[{x,y,data[[x,y]]},
{x,1,First[Dimensions[data]]},{y,1,Last[Dimensions[data]]}]],3]];
lines3=Graphics3D@@ (lines2 /. {x_?NumberQ,y_}:>{xdist[x],ydist[y],h[x,y]});
Show[lines4=(lines3 /.
ln:Line[{{___,z_},___}]:>{Hue[scale[z]],ln}),BoxRatios->{1,1,.4},PlotLabel->plottitle,
Axes->True,AxesLabel->labels];
```






```
<<ExtendGraphics`CorrelationPlot`   (* may be deprecated *)
```

```
CorrelationPlot[Partition[Flatten[nodalpointszvals],m+1],Labels->{"C","S","V","r"},Axes->True,TickFont->3];
```