SOR and PSOR Schemes for European and American Options

Introduction

In this lecture we look at the numerical techniques needed to price American and related contracts within the framework of an implicit finite-difference scheme. The inequalities implicit within an early exercise scheme require us to step outside the use of the simple tridiagonal solution method and work instead with iterative solution methods. This example happens to use a 3-time-level scheme but the ideas apply to any FD model.

Mathematical Reminders for the numerical implementation

We first do this with an explicit scheme, using the simple application of a maximum against the payoff. Here are some quick reminders of the changes of variable.

Transforming to the Diffusion Equation

The Black-Scholes differential equation

$$\frac{\sigma^2}{2} S^2 \frac{\partial^2 V}{\partial S^2} - r V + (r - q) S \frac{\partial V}{\partial S} + \frac{\partial V}{\partial t} = 0 \tag{1}$$

can be transformed into the diffusion equation by a standard change of variables, as given before. When all this is done, one obtains

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial \tau} \tag{2}$$

We now wish to solve this with appropriate modifications for early exercise.

The diffusion equation (49) is solved by introducing a discrete grid with steps Δx and $\Delta\tau$, and setting

$$u_n^m = u(\mathrm{n}\Delta x, \mathrm{m}\Delta\tau) \tag{3}$$

Recall that a key role is played by the parameter α, given by

$$\alpha = \frac{\Delta\tau}{(\Delta x)^2} \qquad \Delta \log S = \sigma\sqrt{\frac{\Delta T}{2\alpha}} \tag{4}$$

■ Black Scholes for Euro options

```
Off[General::"spell1"]
```

```
Ncdf[(z_)?NumberQ] := N[0.5*Erf[z/Sqrt[2]] + 0.5];
Ncdf[x_] := (1 + Erf[x/Sqrt[2]])/2;

done[s_, σ_, k_, t_, r_, q_] :=
((r - q)*t + Log[s/k])/(σ*Sqrt[t]) + (σ*Sqrt[t])/2;
dtwo[s_, σ_, k_, t_, r_, q_] :=
((r - q)*t + Log[s/k])/(σ*Sqrt[t]) - (σ*Sqrt[t])/2;

BlackScholesPut[s_, k_, σ_, r_, q_, t_] :=
k*Exp[-r*t]*Ncdf[-dtwo[s, σ, k, t, r, q]] - s*Exp[-q*t]*Ncdf[-do
BlackScholesPutDelta[s_,k_, v_, r_, q_, t_]=
Evaluate[Simplify[D[BlackScholesPut[s,k, v, r, q, t], s]]];
```

Equation Solvers

The solution of our finite-difference equations requires a solver. Furthermore, when treating American options or CBs, we wish to solve differential inequalities related to equation (1,2), based on an associated constraint surface (the payoff). We therefore need a solution mechanism that allows this. The tridiagonal solver used in the previous lecture does not allow for early exercise problems, so we replace it by the successive over-relaxation (SOR) method, and for early exercise problems, or those with other constraints, the projected form (PSOR). For details of the SOR and PSOR methods in general, you should see Wilmott *et al* (1993). Although we are using rather novel difference schemes here, the solution methods of SOR and PSOR are otherwise identical to those of Wilmott *et al*.

Basics of Gauss-Seidel methods, SOR and PSOR

Suppose we write the matrix equation to be solved as

$$A.x = r \tag{5}$$

and that further more we decompose A as

$$A = D + L + U \tag{6}$$

where D is the diagonal part, L the lower triangular part, U the upper triangular part. Note that the L and U here are totally different from the ones we had before. You should also watch out because some books and papers write $A = D - L - U$! With our conventions we can reorganize the equation as

$$(D + L).x = r - U.x \tag{7}$$

The Gauss-Seidel scheme is based on making this iterative

$$x^{k+1} = (D + L)^{-1} \left(r - U.x^k\right) \tag{8}$$

The SOR variation of Gauss-Seidel is obtained by injecting a "relaxation parameter" ω into the GS scheme and setting

$${x_0}^{k+1} = (D + L)^{-1} \left(r - U.x^k\right) \tag{9}$$

$$x^{k+1} = \omega\, x_0^{k+1} + x^k\,(1 - \omega) \tag{10}$$

The PSOR variation is obtained by taking the relevant maximum of equation (10) with the option payoff function. These methods work provided the matrix is diagonally dominant and $0 < \omega < 2$. Optimal values of ω best chosen (IMHO) by experiment.

Now in practice, although this matrix representation is rather neat, we write it out slightly differently for the tridiagonal problems of interest to us.

```
A = {{b0, c0, 0, …, …, 0, 0}, {a1, b1, c1, 0, …, …, 0},
    {0, a2, b2, c2, …, …, 0}, {0, …, ⋱, ⋱, ⋱, ⋱, 0},
    {0, …, …, 0, a"N-2", b"N-2", c"N-2"},
    {0, …, …, …, 0, a"N-1", b"N-1"}};
MatrixForm[A]
```

$$\begin{pmatrix} b_0 & c_0 & 0 & \dots & \dots & 0 & 0 \\ a_1 & b_1 & c_1 & 0 & \dots & \dots & 0 \\ 0 & a_2 & b_2 & c_2 & \dots & \dots & 0 \\ 0 & \dots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & a_{N-2} & b_{N-2} & c_{N-2} \\ 0 & \dots & \dots & \dots & 0 & a_{N-1} & b_{N-1} \end{pmatrix}$$

The equations to be solved are then

$$b_j x_j + c_j x_{j+1} + a_j x_{j-1} = r_j \tag{11}$$

$$x_j = \frac{1}{b_j}\left(r_j - a_j x_{j-1} - c_j x_{j+1}\right) \tag{12}$$

A stepping stone to the Gauss-Seidel scheme is the Jacobi iteration scheme, where we just use equation (12) to make an iteration:

$$x_j^{i+1} = \frac{1}{b_j}\left(r_j - a_j x_{j-1}^{i} - c_j x_{j+1}^{i}\right) \tag{13}$$

The Gauss-Seidel scheme is a variation on this idea where, as we work through equation (13), as soon as an updated approximation is available, we use it, so we have instead

$$x_j^{i+1} = \frac{1}{b_j}\left(r_j - a_j x_{j-1}^{i+1} - c_j x_{j+1}^{i}\right) \tag{14}$$

To go to SOR, we write

$$\tilde{x}_j^{i+1} = \frac{1}{b_j}\left(r_j - a_j x_{j-1}^{i+1} - c_j x_{j+1}^{i}\right) \tag{15}$$

then the iterate is

$$x_j^{i+1} = \omega \tilde{x}_j^{i+1} + (1-\omega) x_j^{i} \tag{16}$$

This reduces to Gauss-Seidel when $\omega = 1$. It is called over-relaxation when $1 < \omega < 2$, under relaxation when $0 < \omega < 1$. There is a theoretical optimal over-relaxation value, but empirical determination is usually easiest. The projection to the payoff surface is applied just after equation (16) is applied.

Compiled SOR and PSOR Solvers

The matrix equations resulting from our difference schemes are symmetric and also have constant values down the diagonal and off-diagonal lines, so we can simplify matters in constructing the SOR/PSOR solvers:

```
CompSORSolve = Compile[
            {{diag, _Real, 0}, {offdiag, _Real, 0},
    {r, _Real, 1}, {eps, _Real, 0}, {relax, _Real, 0},
    {kickoff, _Real, 1}},

   Module[{len = Length[r], yu = r, yold = kickoff,
     ynew = r, error = 100.0},
                  While[error > eps^2,

     (Do[
       (yu[[i]] = r[[i]] / diag -
          (offdiag / diag) *
           (If[i == 1, 0, ynew[[i - 1]]] +
             If[i < len, yold[[i + 1]], 0]);
        ynew[[i]] = yold[[i]] + relax * (yu[[i]] - yold[[i]])),
       {i, 1, len}];
    error = (ynew - yold).(ynew - yold);
    yold = ynew)]; yold]];
```

We make one change to this routine to define the corresponding projected form suitable for valuing American options:

```
CompPSORSolve = Compile[
            {{diag, _Real, 0}, {offdiag, _Real, 0},
    {r, _Real, 1}, {eps, _Real, 0}, {relax, _Real, 0},
    {kickoff, _Real, 1}},

   Module[{len = Length[r], yu = r, yold = kickoff,
     ynew = r, error = 100.0},
                  While[error > eps^2,

     (Do[
       (yu[[i]] = r[[i]] / diag -
          (offdiag / diag) *
           (If[i == 1, 0, ynew[[i - 1]]] +
             If[i < len, yold[[i + 1]], 0]);
        ynew[[i]] = Max[kickoff[[i]],
          yold[[i]] + relax * (yu[[i]] - yold[[i]])] ),
       {i, 1, len}];
    error = (ynew - yold).(ynew - yold);
    yold = ynew)];     yold]];
```

Standardization of Variables

Here we implement the standard transformations used to obtain the diffusion equation:

```
NonDimExpiry[T_, sd_] := sd^2 T / 2;

kone[r_, sd_] := 2 r / sd^2;

ktwo[r_, q_, sd_] := 2 (r - q) / sd^2;

ValuationMultiplier[strike_, r_, q_, x_, tau_, sd_] :=
 strike
  Exp[-1/2 (ktwo[r, q, sd] - 1) x -
    (1/4 (ktwo[r, q, sd] - 1)^2 + kone[r, sd]) tau]
```

Initial (Expiry) Conditions

```
CallExercise[x_, r_, q_, sd_] :=
  Max[Exp[1/2 (ktwo[r, q, sd] - 1) x] (Exp[x] - 1), 0];
PutExercise[x_, r_, q_, sd_] :=
  Max[Exp[1/2 (ktwo[r, q, sd] - 1) x] (1 - Exp[x]), 0];
```

Boundary Conditions and Payoff Constraints

Since the most interesting basic early-exercise problem arises for the Put, with zero dividends, we focus on that problem. We define both the boundary conditions and the payoff constraint function. Note that these are expressed in terms of the variables of the diffusion equation.

■ Payoff Constraint Function

```
PutPayoff[x_, tau_, r_, q_, sd_] :=
  Exp[((ktwo[r, q, sd] - 1)^2 / 4 + kone[r, sd]) * tau] *
Max[Exp[(ktwo[r, q, sd] - 1) * x / 2] (1 - Exp[x]), 0];
```

■ Put (q=0) Boundary Conditions

The American case is a situation where it makes more sense to write the lower boundary condition using the structure of the early exercise payoff. This is because if the price is low enough the option will be exercised. So this code will eventually be used:

```
(* g[x_,tau_,r_,q_,sd_]:=0;
f[x_,tau_,r_,q_,sd_]:=PutPayoff[x, tau, r, q, sd];*)
```

But while we are trying out SOR for the European case we need European boundary conditions as before:

```
g[x_, tau_, r_, q_, sd_] := 0;
f[x_, tau_, r_, q_, sd_] :=
  Exp[1/2 (ktwo[r, q, sd] - 1) x + 1/4 (ktwo[r, q, sd] - 1)^2 tau] -
   Exp[1/2 (ktwo[r, q, sd] + 1) x + 1/4 (ktwo[r, q, sd] + 1)^2 tau];
```

```
DougDiag[alpha_] := 10 + 12 alpha;
DougOffDiag[alpha_] := 1 - 6 alpha;
```

The matrix used in solving the three-time-level Douglas system is characterized by its own diagonal and off-diagonal entries:

```
ThreeDougDiag[alpha_] := 5 / 4 + 2 alpha;
ThreeDougOffDiag[alpha_] := 1 / 8 - alpha;
```

The right-hand side of our implicit difference scheme is constructed using the same matrices as before:

```
DougDMatrix[alpha_, vec_List] := Module[{temp},
temp = (10 - 12*alpha)*vec + (1+6*alpha)*(RotateRight[vec] + Rot
temp[[1]] = Simplify[First[temp] -  (1+6*alpha)*Last[vec]];
temp[[-1]] = Simplify[Last[temp] - (1 + 6*alpha)*First[vec]];
temp]
```

```
DougDDMatrix[vec_List] := Module[{temp},
temp = (10*vec + RotateRight[vec] + RotateLeft[vec]);
temp[[1]] = Simplify[First[temp] - Last[vec]];
temp[[-1]] = Simplify[Last[temp] - First[vec]];
temp/6]
```

Verification of 3 Time Level Douglas and SOR Algorithm on European Options

We quickly check that the replacement of the tridiagonal solver by the SOR solver makes no significant difference provided the tolerance and relaxation parameters are suitably set.

```
dx = 0.0125; dtau = 0.005; alpha = dtau/dx^2;
M=20; nminus = 320; nplus = 160; alpha
```

```
32.
```

So we have a very large value of $\alpha = 32$. Next we introduce the additional parameters for the SOR solver. A variety of experiments, not reported here, reveal that a good value for the relaxation parameter is about 1.1. The choice of tolerance parameter affects the accuracy and speed of solution. Here we are being fairly demanding on accuracy, and have set a value of 10^{-6} at the price of speed. You are encouraged to explore the impact of changing this parameter for yourselves.

```
relax = 1.1;
tolerance = 0.000001;
```

```
initial = Table[PutExercise[(k - 1 - nminus) dx, 0.05, 0, 0.2],
   {k, nminus + nplus + 1}];
lower = Table[f[-nminus dx, (m - 1) dtau, 0.05, 0, 0.2],
   {m, 1, M + 1}];
upper = Table[g[+nplus dx,  (m - 1) dtau, 0.05, 0, 0.2],
   {m, 1, M + 1}];
```

```
w = Table[0, {m, 1, 3},   {k, 1, nminus+nplus+1}];
```

```
vold = Take[initial, {2, -2}];
payoff = vold;
```

```
w[[1,1]] = f[-nminus*dx, dtau/4, 0.05, 0, 0.2];
w[[1, nminus+nplus+1]] = g[nplus*dx, dtau/4, 0.05, 0, 0.2];
w[[2,1]] = f[-nminus*dx, dtau/2, 0.05, 0, 0.2];
w[[2, nminus+nplus+1]] = g[nplus*dx, dtau/2, 0.05, 0, 0.2];
w[[3,1]] = f[-nminus*dx, dtau, 0.05, 0, 0.2];
w[[3, nminus+nplus+1]] = g[nplus*dx, dtau, 0.05, 0, 0.2];
```

■ First Kick-Off Component

Simple Douglas with two time-levels and $d\,t/4$ time-step.

```
rhs = DougDMatrix[alpha/4, vold]+
Table[
Which[
k==1, (6*alpha/4+1)*lower[[1]] + (6*alpha/4-1)*w[[1, 1]],
k== nplus + nminus-1,  (6*alpha/4+1)*upper[[1]] +
(6*alpha/4-1)*w[[1, nplus+nminus+1]],
True, 0],
{k, 1, nplus + nminus-1}];
vnew = CompSORSolve[DougDiag[alpha/4], DougOffDiag[alpha/4], rhs
w[[1]] = Join[{w[[1,1]]}, vnew, {w[[1,nplus+nminus+1]]}];
vvold = vold;
vold = vnew;
```

Now we have two iterations of the three-time-level Douglas system, doubling the time-step at each stage:

```
rhs = DougDDMatrix[vold] - DougDDMatrix[vvold]/4 +
Table[
Which[
k==1, (alpha/4-1/8)*w[[2, 1]] + w[[1,1]]/6 - lower[[1]]/24,
k==nplus + nminus-1,  (alpha/4-1/8)*w[[2, nplus+nminus+1]] +
w[[1,nplus+nminus+1]]/6 - upper[[1]]/24,
True, 0],
{k, 1, nplus + nminus-1}];
vnew = CompSORSolve[ThreeDougDiag[alpha/4], ThreeDougOffDiag[alp
w[[2]] = Join[{w[[2,1]]}, vnew, {w[[2,nplus+nminus+1]]}];
vold = vnew;
```

```
rhs = DougDDMatrix[vold] - DougDDMatrix[vvold]/4 +
Table[
Which[
k==1, (alpha/2-1/8)*w[[3, 1]] + w[[2,1]]/6 - lower[[1]]/24,
k==nplus + nminus-1,  (alpha/2-1/8)*w[[3, nplus+nminus+1]] +
w[[2,nplus+nminus+1]]/6 - upper[[1]]/24,
True, 0],
{k, 1, nplus + nminus-1}];
vnew = CompSORSolve[ThreeDougDiag[alpha/2], ThreeDougOffDiag[alp
w[[3]] = Join[{w[[3,1]]}, vnew, {w[[3,nplus+nminus+1]]}];
wold = initial;
wvold = initial;
wnew = w[[3]];
```

■ Main Block

We evolve this using the SOR solver. You are warned that this method takes a lot longer than the tridiagonal solution method. It is not recommended as a working model for European options. Here we are carrying out a verification of the SOR solution method, before proceeding to the PSOR approach for American options. (In the following we have suppressed the output of **`Print`** that shows progress.)

```
For[m=3, m<=M+1, m++,
(wvold = wold;
wold = wnew;
Print[m];
rhs = DougDDMatrix[Take[wold, {2, -2}]] -
DougDDMatrix[Take[wvold, {2, -2}]]/4 +
Table[
Which[
k==1,
(alpha-1/8)*lower[[m]] + lower[[m-1]]/6 - lower[[m-2]]/24,
k==nplus + nminus-1,
(alpha-1/8)*upper[[m]] + upper[[m-1]]/6 - upper[[m-2]]/24,
True, 0],
{k, 1, nplus + nminus-1}];
temp = CompSORSolve[ThreeDougDiag[alpha], ThreeDougOffDiag[alpha
wnew = Join[{lower[[m]]}, temp, {upper[[m]]}])
]
```

```
3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19
```

20

21

```
points = Table[(k-nminus-1)*dx,  {k, 1, nplus+nminus+1}];
finalstep = wnew;
prevstep = wold;
pprevstep = wvold;
interpoldata = Transpose[{points,finalstep}];
ufunc = Interpolation[interpoldata, InterpolationOrder -> 3];
Valuation[strike_, r_, q_, S_, T_, sd_] :=
ValuationMultiplier[strike, r, q, Log[S/strike], sd^2*T/2, sd]*
ufunc[Log[S/strike]]
```

Sample Errors in Valuation

We now tabulate, for verification purposes, the computed and exact values and the absolute error. Note that this is a five-year option and we have only used 20 time-steps ($\alpha = 32$).

```
samples = TableForm[Join[{{"S", "SOR", "Exact", "Absolute Error"
Table[Map[PaddedForm[N[#], {5, 4}]&, {S, Valuation[10, 0.05, 0,
BlackScholesPut[S,10,0.2,0.05,0,5],
Valuation[10, 0.05, 0, S, 5, 0.2]-BlackScholesPut[S,10,0.2,0.05,
{S, 2, 16, 1}]]]
```

S	SOR	Exact	Absolute Error
2.0000	5.7889	5.7886	0.0003
3.0000	4.8012	4.8005	0.0007
4.0000	3.8621	3.8615	0.0006
5.0000	3.0209	3.0209	0.0000
6.0000	2.3105	2.3108	-0.0003
7.0000	1.7383	1.7387	-0.0004
8.0000	1.2930	1.2932	-0.0002
9.0000	0.9547	0.9548	-0.0001
10.0000	0.7019	0.7019	0.0000
11.0000	0.5150	0.5149	0.0000
12.0000	0.3777	0.3777	0.0000
13.0000	0.2772	0.2773	-0.0000
14.0000	0.2039	0.2039	-0.0001
15.0000	0.1503	0.1504	-0.0001
16.0000	0.1111	0.1113	-0.0002

The Greeks are all good as well - see Chapter 16 of MFDwM if you are interested.

3TLD-PSOR Analysis of American Options

Now we proceed to the first "real" application of these methods, and look at the valuation of American Puts using a PSOR scheme based on 3TLD. We shall keep most parameters to be identical with the previous European case, but consider expiry times of one, five and ten years, and also we shall turn on "early exercise" within the solution scheme by turning on the projection onto the payoff constraint surface. We shall fix the number of time-steps at 20, so in the case of one year to expiry, $\alpha = 6.4$. This case has been contrived to reveal an important issue regarding the calculation of Greeks near the critical price.

```
dx = 0.0125; dtau = 0.001; alpha = dtau / dx^2;
M = 20; nminus = 320; nplus = 160;
alpha
```

```
6.4
```

```
relax = 1.1;
tolerance = 0.000001;
```

```
initial = Table[PutExercise[(k - 1 - nminus) dx, 0.05, 0, 0.2],
   {k, nminus + nplus + 1}];
lower = Table[f[-nminus dx, (m - 1) dtau, 0.05, 0, 0.2], {m, 1, M + 1}];
upper = Table[g[+nplus dx,  (m - 1) dtau, 0.05, 0, 0.2], {m, 1, M + 1}];
```

```
w = Table[0, {m, 1, 3},   {k, 1, nminus+nplus+1}];
```

```
vold = Take[initial, {2, -2}];
```

```
w[[1,1]] = f[-nminus*dx, dtau/4, 0.05, 0, 0.2];
w[[1, nminus+nplus+1]] = g[nplus*dx, dtau/4, 0.05, 0, 0.2];
w[[2,1]] = f[-nminus*dx, dtau/2, 0.05, 0, 0.2];
w[[2, nminus+nplus+1]] = g[nplus*dx, dtau/2, 0.05, 0, 0.2];
w[[3,1]] = f[-nminus*dx, dtau, 0.05, 0, 0.2];
w[[3, nminus+nplus+1]] = g[nplus*dx, dtau, 0.05, 0, 0.2];
```

■ First Kick-Off Component

Simple Douglas with two time levels and $d\,t/4$ time-step. We treat the American case by replacing SOR with PSOR based on the payoff function. You should note that there are only two differences between this analysis and the previous European case:

(a) the payoff constraint function is re-computed for each time step;

(b) the solver is the **`CompPSORSolve`** function, which projects the solution onto the payoff constraint surface.

```
payoff = Table[PutPayoff[(k - 1 - nminus) * dx, dtau / 4, 0.05, 0, 0.2],
    {k, 2, nminus + nplus}];
```

```
rhs = DougDMatrix[alpha/4, vold]+
Table[
Which[
k==1, (6*alpha/4+1)*lower[[1]] + (6*alpha/4-1)*w[[1, 1]],
k== nplus + nminus-1,  (6*alpha/4+1)*upper[[1]] +
(6*alpha/4-1)*w[[1, nplus+nminus+1]],
True, 0],
{k, 1, nplus + nminus-1}];
vnew = CompPSORSolve[DougDiag[alpha/4], DougOffDiag[alpha/4], rhs, tole
w[[1]] = Join[{w[[1,1]]}, vnew, {w[[1,nplus+nminus+1]]}];
vvold = vold;
vold = vnew;
```

Now we have two iterations of the three-time-level Douglas system, doubling the time-step at each stage, and re-computing the payoff.

```
payoff = Table[PutPayoff[(k - 1 - nminus) * dx, dtau / 2, 0.05, 0, 0.2],
    {k, 2, nminus + nplus}];
```

```
rhs = DougDDMatrix[vold] - DougDDMatrix[vvold]/4 +
Table[
Which[
k==1, (alpha/4-1/8)*w[[2, 1]] + w[[1,1]]/6 - lower[[1]]/24,
k==nplus + nminus-1,  (alpha/4-1/8)*w[[2, nplus+nminus+1]] +
w[[1,nplus+nminus+1]]/6 - upper[[1]]/24,
True, 0],
{k, 1, nplus + nminus-1}];
vnew = CompPSORSolve[ThreeDougDiag[alpha/4], ThreeDougOffDiag[alpha/4],
w[[2]] = Join[{w[[2,1]]}, vnew, {w[[2,nplus+nminus+1]]}];
vold = vnew;
```

```
payoff = Table[PutPayoff[(k - 1 - nminus) * dx, dtau, 0.05, 0, 0.2],
    {k, 2, nminus + nplus}];
```

```
rhs = DougDDMatrix[vold] - DougDDMatrix[vvold]/4 +
Table[
Which[
k==1, (alpha/2-1/8)*w[[3, 1]] + w[[2,1]]/6 - lower[[1]]/24,
k==nplus + nminus-1,  (alpha/2-1/8)*w[[3, nplus+nminus+1]] +
w[[2,nplus+nminus+1]]/6 - upper[[1]]/24,
True, 0],
{k, 1, nplus + nminus-1}];
vnew = CompPSORSolve[ThreeDougDiag[alpha/2], ThreeDougOffDiag[alpha/2],
w[[3]] = Join[{w[[3,1]]}, vnew, {w[[3,nplus+nminus+1]]}];
wold = initial;
wvold = initial;
wnew = w[[3]];
```

■ Main Block

Once the kick-off phase has been completed, we go into the main block. Again, the only difference between this case and European is re-computation of the payoff constraint and the use of the PSOR solver. The output of the **`Print`** statement is not shown.

```
For[m=3, m<=M+1, m++,
(wvold = wold;
wold = wnew;
Print[m];
rhs = DougDDMatrix[Take[wold, {2, -2}]] -
DougDDMatrix[Take[wvold, {2, -2}]]/4 +
Table[
Which[
k==1,
(alpha-1/8)*lower[[m]] + lower[[m-1]]/6 - lower[[m-2]]/24,
k==nplus + nminus-1,
(alpha-1/8)*upper[[m]] + upper[[m-1]]/6 - upper[[m-2]]/24,
True, 0],
{k, 1, nplus + nminus-1}];
payoff = Table[PutPayoff[(k-1-nminus)*dx, (m-1)*dtau, 0.05, 0, 0.2], {k
temp = CompPSORSolve[ThreeDougDiag[alpha], ThreeDougOffDiag[alpha], rhs
wnew = Join[{lower[[m]]}, temp, {upper[[m]]}])
]
```

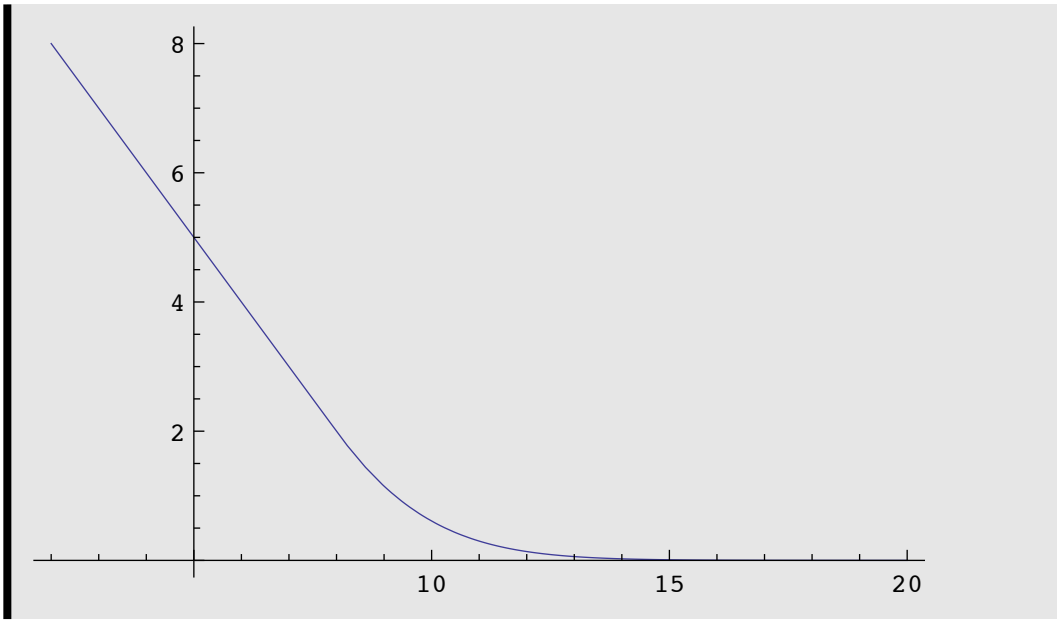
```
3

4

5
```

```
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```

```
points = Table[(k - nminus - 1) * dx,  {k, 1, nplus + nminus + 1}];
finalstep = wnew;
prevstep = wold;
pprevstep = wvold;
interpoldata = Transpose[{points, finalstep}];
ufunc = Interpolation[interpoldata, InterpolationOrder -> 3];
Valuation[strike_, r_, q_, S_, T_, sd_] :=
ValuationMultiplier[strike, r, q, Log[S / strike], sd^2 * T / 2,
   sd] *
ufunc[Log[S / strike]]
```

Plot Sample Valuation

```
Plot[Valuation[10, 0.05, 0, S, 1, 0.2], {S, 2, 20}, PlotPoints → 50,
 PlotRange → All]
```



Construction and Verification of Interpolated Valuation and Greeks - First Simple Method

In this subsection we define some functions to compute Delta, Gamma and Theta directly from the finite-difference grid. Note that these are based on the same simple differencing and interpolation methods as were used for the European case, but we should anticipate that these methods introduce some inappropriate smoothing in the neighbourhood of the critical price frontier, where Γ is actually discontinuous.

```
listd[data_, step_] :=
Module[{dleft,dright,len},
len = Length[data];
dleft = (4*data[[2]]-3*data[[1]]-data[[3]])/(2*step);
dright = (3*data[[len]]-4*data[[len-1]]+data[[len-2]])/(2*step);
Join[{dleft}, Take[RotateLeft[data]-RotateRight[data], {2, -2}]/(2*step
]
```

```
kt = ktwo[0.05, 0, 0.2];
ko = kone[0.05, 0.2];
```

$$\text{deltadata} = \text{listd}[\text{finalstep}, \text{dx}] - \frac{1}{2}\,(\text{kt} - 1)\,\text{finalstep};$$

$$\text{gammadata} = \text{listd}[\text{deltadata}, \text{dx}] - \frac{1}{2}\,(\text{kt} + 1)\,\text{deltadata};$$

$$\text{thetadata} = \frac{3\,\text{finalstep} - 4\,\text{prevstep} + \text{pprevstep}}{2\,\text{dtau}} - \left(\frac{1}{4}\,(\text{kt} - 1)^2 + \text{ko}\right)\text{finalstep};$$

```
deltainterpoldata = Transpose[{points, deltadata}];
gammainterpoldata = Transpose[{points, gammadata}];
thetainterpoldata = Transpose[{points, thetadata}];
```

```
dfunc = Interpolation[deltainterpoldata, InterpolationOrder → 3];
gfunc = Interpolation[gammainterpoldata, InterpolationOrder → 3];
tfunc = Interpolation[thetainterpoldata, InterpolationOrder → 3];
```

```
SORDelta[strike_, r_, q_, S_, T_, sd_] :=
 1/S (ValuationMultiplier[strike, r, q, Log[S/strike], sd^2 T/2, sd]
    dfunc[Log[S/strike]])
```

```
SORGamma[strike_, r_, q_, S_, T_, sd_] :=
 1/S^2 (ValuationMultiplier[strike, r, q, Log[S/strike], sd^2 T/2, sd]
    gfunc[Log[S/strike]])
```

```
SORTheta[strike_, r_, q_, S_, T_, sd_] :=
 -1/2 sd^2 ValuationMultiplier[strike, r, q, Log[S/strike], sd^2 T/2, sd]
  tfunc[Log[S/strike]]
```

Summary of Finite-Difference Results - One Year Expiry

In the following table we give a table of underlying values, option valuations, Deltas, Gammas and Thetas for a range of underlying prices with the strike at 10. (1 year to expiry, 20% volatility, 5% interest rate, 20 time-steps, 480 price-steps)

```
samples = TableForm[Join[{{"S", "SOR Value", "SOR Delta", "SOR Gamma",
Table[Map[PaddedForm[N[#], {5, 3}]&, {S, Valuation[10, 0.05, 0, S, 1, 0
SORDelta[10, 0.05, 0, S, 1, 0.2],SORGamma[10, 0.05, 0, S, 1, 0.2],
SORTheta[10, 0.05, 0, S, 1, 0.2]}],
{S, 2, 16, 1}]]]
```

S	SOR Value	SOR Delta	SOR Gamma	SOR Theta
2.000	8.000	-1.000	-0.000	1.528×10^{-6}
3.000	7.000	-1.000	-0.000	1.337×10^{-6}
4.000	6.000	-1.000	-0.000	1.146×10^{-6}
5.000	5.000	-1.000	-0.000	9.552×10^{-7}
6.000	4.000	-1.000	-0.000	7.642×10^{-7}
7.000	3.000	-1.000	-0.000	5.731×10^{-7}
8.000	2.000	-1.000	0.065	0.000
9.000	1.149	-0.683	0.312	-0.141
10.000	0.609	-0.411	0.229	-0.223
11.000	0.299	-0.224	0.147	-0.217
12.000	0.137	-0.111	0.082	-0.164
13.000	0.059	-0.051	0.042	-0.104
14.000	0.024	-0.022	0.019	-0.058
15.000	0.010	-0.009	0.008	-0.030
16.000	0.004	-0.004	0.003	-0.014

Note the value of Gamma at a stock price of 8 - the option value and Delta would suggest exercise has occurred, but $\Gamma \neq 0$. Differencing around the front! Watch out as Γ is discontinuous there.