Finite-Difference Schemes for the Black-Scholes Equation with Non-smooth Payoff Initial Conditions

Issues with the Greeks in simple schemes

Introduction

[This lecture will be supplemented with C++ code in a lab session provided separately.]

In previous lectures I gave some examples which show how various commonly used finite-difference schemes behave with simple smooth "payoffs". With smooth payoffs the Douglas finite-difference scheme gives much better results than Crank-Nicolson. In this session we will look at how these methods behave when applied to real-world option-pricing problems. This will highlight the potentially nasty behaviour of both Crank-Nicolson and Douglas finite-difference schemes when applied to simple option-pricing problems. The discontinuous nature of the payoff or its derivative in the neighbourhood of the strike induces slowly decaying oscillations into the solution of the finite-difference equations, when we use a larger time-step (which is the main point of introducing FD schemes in the first place.) These introduce small errors into the valuation itself, and undermine attempts to compute δ, Γ and other "derivative" quantities in the neighbourhood of the strike. Note that, even with the oscillatory components small in the valuation error, the effect on the slope of the function is larger (generating larger errors in δ), and there are even larger errors in Γ.

In this session I also look at the three-time-level version of the Douglas scheme. This chapter shows how the problem of larger errors in the Greeks can be cured, in that the oscillations can be removed, at least for our test payoffs, and the method ("three-time-level Douglas") will be used initially as the basis of our programme to define benchmark numerical algorithms. Later we shall look at some recent work by Evans and Chawla, and Giles and Carter.

The problems with the Crank-Nicolson scheme, and its resolution, are exemplified by a test problem with $\alpha = 8$. Although the error in the valuation is small, we get *an error in Γ near the strike of* 12% *of its exact*

value, and an error in θ of 200%. In the corresponding three-time-level Douglas solution, the error in Γ near the strike is reduced to 0.2%, and that in θ to 2.6%.

Finite-Difference Schemes with Three Time-Levels

Various schemes have been proposed for treating problems introduced by considering discontinuous or non-smooth boundary/initial conditions. These schemes aim to damp fast oscillations more effectively, by adjusting the spectrum of eigenvalues of the difference matrix. Richtmyer and Morton (1957) give a list of 14 difference schemes for the diffusion equation, and recommend schemes (their numbering) 9, 11 and 13 for non-smooth initial data. Schemes 9 and 13 are also recommended for these purposes by Smith (1985). These two schemes may be regarded as the three-time-level versions of the Crank-Nicolson scheme and the Douglas scheme, since they have truncation errors of a similar character to their two-time-level counterparts. We already know that the truncation error characteristics of the Douglas scheme make it preferable, so we shall use this. As in previous lectures, we let m denote the time-step, and n denote the x-step. The three time-level Douglas scheme is then given by

$$\begin{aligned}&\left(\frac{1}{8}-\alpha\right)\left(u_{n-1}^{m+1}+u_{n+1}^{m+1}\right)+\left(\frac{5}{4}+2\,\alpha\right)u_n^{m+1}\\ &=\ \frac{1}{6}\left(u_{n-1}^{m}+u_{n+1}^{m}+10\,u_n^{m}\right)-\frac{1}{24}\left(u_{n-1}^{m-1}+u_{n+1}^{m-1}+10\,u_n^{m-1}\right)\end{aligned} \tag{1}$$

This type of process requires a kick-off procedure, since initially we only know u^1. We use the ordinary Douglas two-time-level scheme:

$$(1\ -\ 6\,\alpha)\left(u_{n-1}^{m+1}+u_{n+1}^{m+1}\right)\ +\ (10\ +\ 12\,\alpha)\,u_n^{m+1}=(1\ +\ 6\,\alpha)\left(u_{n-1}^{m}+u_{n+1}^{m}\right)\ +(10-12\,\alpha)\,u_n^{m} \tag{2}$$

once with $\alpha \to \alpha/4$, then the three-time-level scheme once with $\alpha/4$ and then again with $\alpha/2$.

This gives us our vector pair of vectors to allow the three-time-level iteration to proceed normally thereafter.

Some Required Functions from previous lectures

I am going to define many functions with similar sounding names so I will ask *Mathematica* to not give spelling warnings (only do this if you have sure there are no mistakes in function names!)

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

```
ExplicitSolver =
    Compile[{{initial, _Real, 1}, {lower, _Real, 1},
    {upper, _Real, 1}, alpha},
        Module[{wold = initial, wnew = initial, wvold = initial,
     m, k, tsize = Length[lower], xsize = Length[initial]},
           For[m = 2, m <= tsize, m++,
                (wvold = wold; wold = wnew;
                    For[k = 2, k < xsize, k++,

       (wnew[[k]] = alpha (wold[[k - 1]] + wold[[k + 1]]) +
          (1 - 2 alpha) wold[[k]])];
                        wnew[[1]] = lower[[m]];
                        wnew[[xsize]] = upper[[m]])];
            {wvold, wold, wnew}]
        ];
```

```
CompTridiagSolve =
  Compile[{{a, _Real, 1}, {b, _Real, 1}, {c, _Real, 1},
    {r, _Real, 1}},
  Module[{len = Length[r], solution = r, aux = 1 / (b[[1]]),
     aux1 = r, a1 = Prepend[a, 0.0], iter},
    solution[[1]] = aux * r[[1]];
    Do[aux1[[iter]] = c[[iter - 1]] aux;
            aux = 1 / (b[[iter]] - a1[[iter]] * aux1[[iter]]);
               solution[[iter]] =
      (r[[iter]] - a1[[iter]] solution[[iter - 1]]) aux,
      {iter, 2, len}];
    Do[solution[[iter]] -= aux1[[iter + 1]] solution[[iter + 1]],
      {iter, len - 1, 1, -1}];
        solution]];
```

Adjustment for non-zero boundary conditions

Our implicit schemes involve the solution of a matrix problem $A.x = r$ where A has the tridiagonal form :

```
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]
```

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

In the particular case, for example, of a fully implicit scheme, we have the matrix whose diagonal and off-diagonal terms are

```
A = {{1 + 2 α, -α, 0, …, …, 0, 0}, {-α, 1 + 2 α, -α, 0, …, …, 0},
   {0, -α, 1 + 2 α, -α, …, …, 0}, {0, …, ⋱, ⋱, ⋱, ⋱, 0},
   {0, …, …, 0, -α, 1 + 2 α, -α}, {0, …, …, …, 0, -α, 1 + 2 α}};
MatrixForm[A]
```

$$\begin{pmatrix} 1+2\alpha & -\alpha & 0 & \dots & \dots & 0 & 0 \\ -\alpha & 1+2\alpha & -\alpha & 0 & \dots & \dots & 0 \\ 0 & -\alpha & 1+2\alpha & -\alpha & \dots & \dots & 0 \\ 0 & \dots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & -\alpha & 1+2\alpha & -\alpha \\ 0 & \dots & \dots & \dots & 0 & -\alpha & 1+2\alpha \end{pmatrix}$$

This came from the difference equation

$$(1+2\alpha)\,u_n^{m+1} - \alpha\left(u_{n-1}^{m+1} + u_{n+1}^{m+1}\right) = u_n^m \tag{3}$$

When we are at the edge of the grid we need to allow for the possibly non-zero value of the boundary condition at the top at bottom end. In the fully implicit case we correct the first and last entries on the right hand side by $+\alpha\, u_p^{m+1}$ where $p = 1$ or $N_{\max}$. Similarly with other schemes. In the general θ-method case the discrete problem is

$$(1+2\alpha\theta)\,u_n^{m+1} - \alpha\theta\left(u_{n-1}^{m+1} + u_{n+1}^{m+1}\right) = (1-2\alpha(1-\theta))\,u_n^m + \alpha(1-\theta)\left(u_{n-1}^m + u_{n+1}^m\right) \tag{4}$$

and we will add a term $+\alpha\,\theta\,u_p^{m+1} + \alpha(1-\theta)\,u_p^m$ at the edges of the grid.NEEDS CARE - possible source of error. You need to think about boundary conditions. Usually such thought pays off. Sometimes a misplaced love of trees can arise from wanting to avoid thinking about boundary conditions. (But then you get bitten by exotics - see Boyle's work on barriers.)

In our calculations we shall just use some simple lists

```
FullyImpCMatrix[alpha_, nminus_, nplus_] :=
Sequence[Table[-alpha, {nplus+nminus-2}],
Table[1+2*alpha, {nplus+nminus-1}],
Table[-alpha, {nplus+nminus-2}]]
```

```
FullyImpCMatrix[α, 2, 2]
```

```
Sequence[{-α, -α}, {1 + 2 α, 1 + 2 α, 1 + 2 α}, {-α, -α}]
```

Case Study: the Vanilla European Put Option

We now begin a detailed study of the first of two examples that we have picked for detailed investigation. You might wonder why we are considering such a trivial case for which there is a known analytic solution. Our goal here is to try out various difference schemes and find out what works well, by testing them on a case for which the solution is known and where the errors can be precisely described. In this way we can see what is happening without all the complications of other real-world effects.

Factors Common to All Our FD Schemes

The Black-Scholes differential equation

$$\frac{1}{2} S^2 \frac{\partial^2 V}{\partial S^2} \sigma^2 - r\,V + (r-q)\,S \frac{\partial V}{\partial S} + \frac{\partial V}{\partial t} = 0 \tag{5}$$

with constant coefficients (in particular *r* is a constant) can be transformed into the diffusion equation (1) by a standard change of variables, as given by Wilmott *et al* (Chapter 17, equations 17.1, 17.2). We can therefore implement our four standard schemes on the problem, and use solutions of (3) with known payoffs. In the following example we use the vanilla European Put.

Standardization of Variables

With a strike K and constant parameters $r,\ q,\ \sigma$, we make the changes of variables

$$\tau = \frac{\sigma^2 (T-t)}{2} \qquad k_1 = \frac{2r}{\sigma^2} \qquad k_2 = \frac{2(r-q)}{\sigma^2}$$
$$V(S,\ t) = K\, e^{-\frac{1}{2}(k_2-1)x - \left(\frac{1}{4}(k_2-1)^2 + k_1\right)\tau}\, u(x,\ \tau) \tag{6}$$

The *Mathematica* implementation of this requires the following functions.

```
NonDimExpiry[T_, σ_] := σ^2 T / 2;

kone[r_, σ_] := 2 r / σ^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

$$\begin{aligned} V(S,\ T) &= K\, e^{-\frac{1}{2}(k_2-1)x}\, u(x,\ 0) \\ C(S,\ T) &= \text{Max}(S-K,\ 0) \\ P(S,\ T) &= \text{Max}(K-S,\ 0) \end{aligned} \tag{7}$$

So, e.g. for a Put

$$u_P(x,\ 0) = e^{+\frac{1}{2}(k_2-1)x}\, 1/K\, \text{Max}(K-S,0) = e^{+\frac{1}{2}(k_2-1)x}\, \text{Max}(1-S/K,0) = e^{+\frac{1}{2}(k_2-1)x}\, \text{Max}(1-e^x,0) \tag{8}$$

and for a Call

$$u_C(x,\ 0) = e^{+\frac{1}{2}(k_2-1)x}\, 1/K\, \text{Max}(S-K,0) = e^{+\frac{1}{2}(k_2-1)x}\, \text{Max}(S/K-1,0) = e^{+\frac{1}{2}(k_2-1)x}\, \text{Max}(e^x-1,0) \tag{9}$$

The *Mathematica* functions to do this are:

```
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];
```

Black-Scholes Model for Verification

From last time - nothing new here:

```
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;
```

```
BlackScholesCall[s_, k_, σ_, r_, q_, t_] :=
s*Exp[-q*t]*Ncdf[done[s, σ, k, t, r, q]] - k*Exp[-r*t]*Ncdf[dtwo[s, σ,
BlackScholesPut[s_, k_, σ_, r_, q_, t_] :=
k*Exp[-r*t]*Ncdf[-dtwo[s, σ, k, t, r, q]] - s*Exp[-q*t]*Ncdf[-done[s, σ
```

Just check the function calls are working:

```
{BlackScholesCall[8, 10, 0.2, 0.05, 0, 3],
 BlackScholesPut[8, 10, 0.2, 0.05, 0, 3]}
```

```
{0.86337, 1.47045}
```

Build Greeks automatically for verification

In *Mathematica* we can just calculcate the derivatives automatically

```
BlackScholesCallDelta[s_,k_, v_, r_, q_, t_]=
Evaluate[Simplify[D[BlackScholesCall[s,k, v, r, q, t], s]]];

BlackScholesPutDelta[s_,k_, v_, r_, q_, t_]=
Evaluate[Simplify[D[BlackScholesPut[s,k, v, r, q, t], s]]];

BlackScholesCallGamma[s_,k_, v_, r_, q_, t_]=
Evaluate[D[BlackScholesCall[s,k, v, r, q, t], {s, 2}]];

BlackScholesPutGamma[s_,k_, v_, r_, q_, t_]=
Evaluate[D[BlackScholesPut[s,k, v, r, q, t], {s, 2}]];

BlackScholesCallTheta[s_,k_, v_, r_, q_, t_]=
-Evaluate[D[BlackScholesCall[s,k, v, r, q, t], t]];

BlackScholesPutTheta[s_,k_, v_, r_, q_, t_]=
-Evaluate[D[BlackScholesPut[s,k, v, r, q, t], t]];

BlackScholesCallRho[s_,k_, v_, r_, q_, t_]=
Evaluate[D[BlackScholesCall[s,k, v, r, q, t], r]];

BlackScholesPutRho[s_,k_, v_, r_, q_, t_]=
Evaluate[D[BlackScholesPut[s,k, v, r, q, t], r]];

BlackScholesCallVega[s_,k_, v_, r_, q_, t_]=
Evaluate[D[BlackScholesCall[s,k, v, r, q, t], v]];

BlackScholesPutVega[s_,k_, v_, r_, q_, t_]=
Evaluate[D[BlackScholesPut[s,k, v, r, q, t], v]];
```

Put Boundary Conditions

For our boundary conditions the upper boundary condition is to set the function to zero, while the lower takes what we expect to be the Put value as the stock price approaches zero. In general the determination of suitable boundary conditions can take quite a bit of thought. It is also a common source of error, though sometimes a bad choice might be washed out by a payoff constraint in the interior of the solution. (Not a good idea to ever repy on that!) So what is the Put value as S becomes small, or equivalently as x becomes large and negative? If S is very small compared to K the Put will be exercised at maturity with a probability that tends to 1 as S tends to zero, so that we can take the present value of $K - S_T$. This present value (or solve the BS equation) is given by

$$e^{-r(T-t)} K - e^{-q(T-t)} S \tag{10}$$

and if we make the transformation to u (I messed this up in my book - the bits below do satisfy the diffusion equation!) you get

$$e^{\frac{1}{4}\tau(k_2-1)^2+\frac{1}{2}x(k_2-1)} - e^{\frac{1}{4}\tau(k_2+1)^2+\frac{1}{2}x(k_2+1)}; \tag{11}$$

Note that as x gets large and negative the second term is exponentially small compared to the first, but best not to use just the K piece as you have not yet thought about where to truncate the grid.

```
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];
```

Explicit Scheme for Put

```
dx = 0.025; dtau = 0.00025; alpha = dtau/dx^2;
```

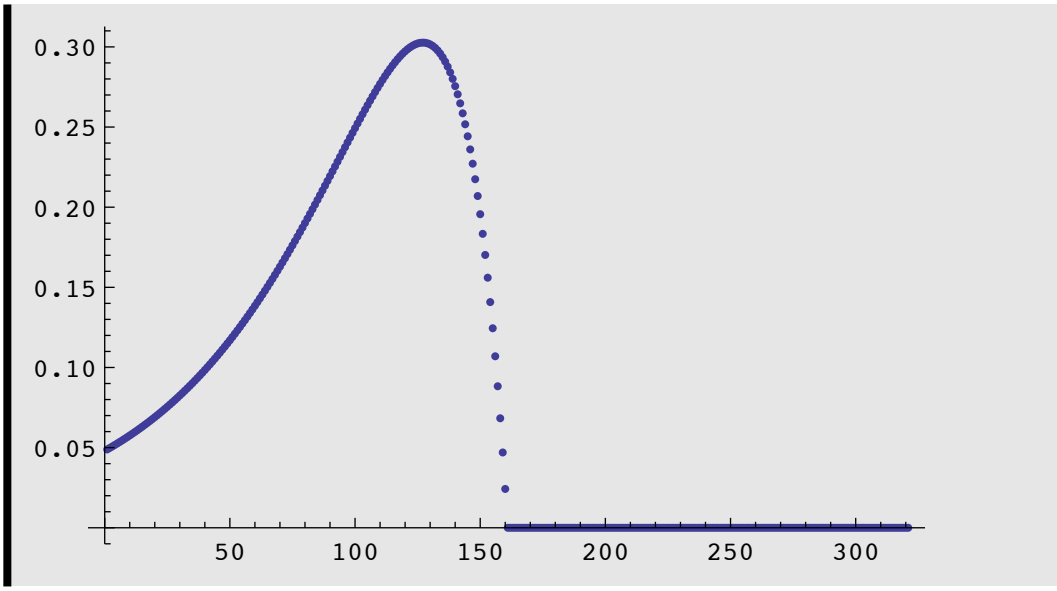
```
M = 400; nminus = 160; nplus = 160;
```

■ Setting the Initial (Expiry) Condition and Boundary Conditions

```
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}];
```

Note that the payoff looks slightly odd in these coordinates - the main point to notice is the discontinuity in slope at the strike.

```
ListPlot[initial]
```



Solving the PDE (Explicit Method)

We use the same functions as we used for our smooth test problem in Chapter 14 - load the function **ExplicitSolver** now if you are using the electronic form:

```
soln = ExplicitSolver[initial, lower, upper, alpha ];
```

Interpolating to Supply a Continuous Function

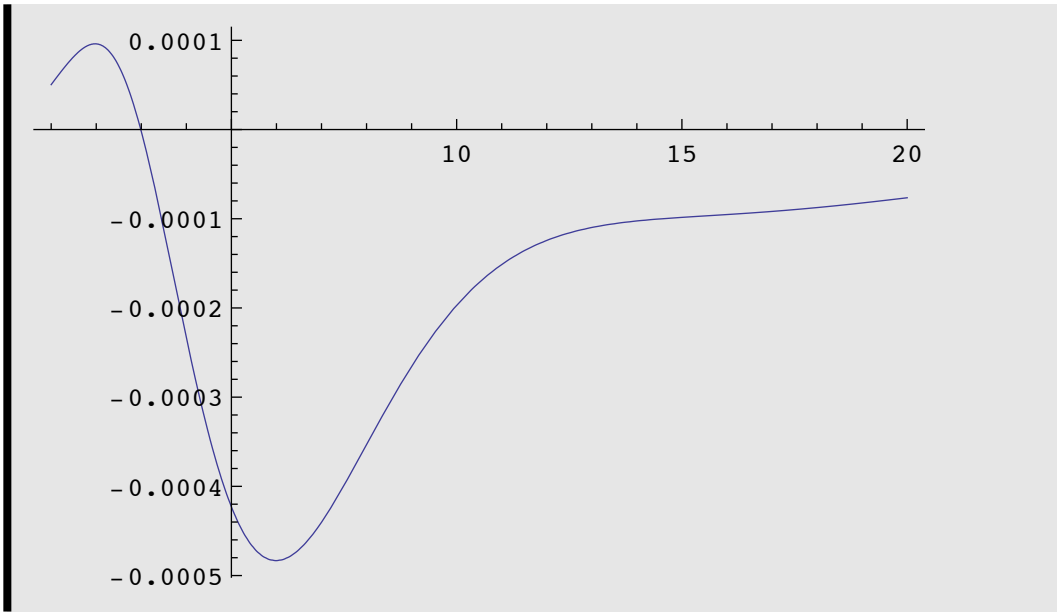
```
interpoldata = Table[{(k - nminus - 1) dx, soln[[3, k]]},
    {k, 1, nminus + nplus + 1}];
```

```
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]]
```

Error Plot

```
Plot[Valuation[10, 0.05`, 0, S, 5, 0.2`] -
  BlackScholesPut[S, 10, 0.2`, 0.05`, 0, 5], {S, 1, 20},
 PlotPoints → 50, PlotRange → All]
```


0.0001
10
15
20
-0.0001
-0.0002
-0.0003
-0.0004
-0.0005

```
samples = TableForm[Join[{{"S", "Explicit FD", "Exact", "Error"}},
    Table[(PaddedForm[N[#1], {5, 5}] &) /@
        {S, Valuation[10, 0.05, 0, S, 5, 0.2],
      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, 0, 5]},
            {S, 2, 16, 1}
   ]]]
```

S	Explicit FD	Exact	Error
2.00000	5.78870	5.78860	0.00010
3.00000	4.80050	4.80050	-2.16410×10^{-6}
4.00000	3.86130	3.86150	-0.00023
5.00000	3.02040	3.02090	-0.00042
6.00000	2.31040	2.31080	-0.00048
7.00000	1.73820	1.73870	-0.00044
8.00000	1.29290	1.29320	-0.00035
9.00000	0.95451	0.95478	-0.00027
10.00000	0.70167	0.70187	-0.00020
11.00000	0.51477	0.51492	-0.00015
12.00000	0.37754	0.37766	-0.00012
13.00000	0.27715	0.27726	-0.00011
14.00000	0.20384	0.20394	-0.00010
15.00000	0.15030	0.15040	-0.00010
16.00000	0.11116	0.11125	-0.00010

Fully Implicit Scheme for Put

These algorithms are by now self-explanatory - first the initialization:

```
dx = 0.025; dtau = 0.00025; alpha = dtau/dx^2;
M = 400; nminus = 160; nplus = 160;

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}];
wold = initial;
wvold = wold; wnew = wold;

CMat = FullyImpCMatrix[alpha, nminus, nplus];
```

Evolving the solution (this may take some time):

```
For[m=2, m<=M+1, m++,
(wvold = wold;
wold = wnew;
(* Adjust the rhs for non-zero BCs *)
rhs = Take[wold, {2, -2}]+
Table[
Which[
k==1, alpha*lower[[m]],
k== nplus + nminus-1,  alpha*upper[[m]],
True, 0],
{k, 1, nplus + nminus-1}];
temp = CompTridiagSolve[CMat, rhs];
wnew = Join[{lower[[m]]}, temp, {upper[[m]]}])
]
```

Interpolation and construction of valuation function:

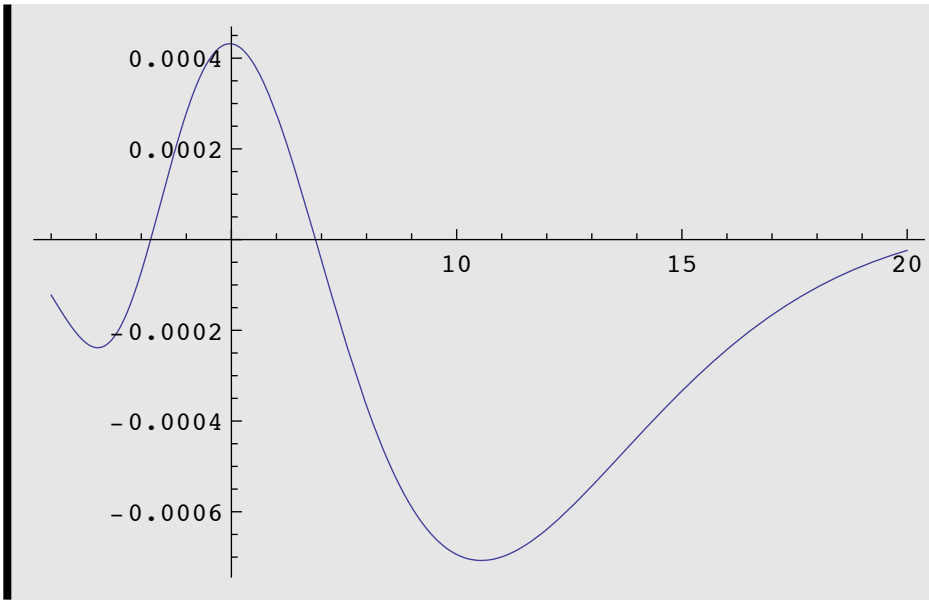
```
interpoldata = Table[{(k - nminus - 1)*dx, wnew[[k]]},
    {k, 1, nminus + nplus + 1}];
```

```
ufuncb = Interpolation[interpoldata, InterpolationOrder -> 3];
```

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

Error Plot

```
Plot[Valuation[10, 0.05`, 0, S, 5, 0.2`] -
  BlackScholesPut[S, 10, 0.2`, 0.05`, 0, 5], {S, 1, 20},
 PlotPoints → 50, PlotRange → All]
```



```
samples = TableForm[Join[{{"S", "Implicit FD", "Exact", "Error"}},
    Table[(PaddedForm[N[#1], {5, 5}] &) /@
        {S, Valuation[10, 0.05, 0, S, 5, 0.2],

      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, 0, 5]},     {S, 2, 16, 1}]]]
```

```
S         Implicit FD  Exact    Error
 2.00000   5.78830     5.78860 -0.00024
 3.00000   4.80040     4.80050 -0.00007
 4.00000   3.86180     3.86150  0.00028
 5.00000   3.02130     3.02090  0.00043
 6.00000   2.31110     2.31080  0.00028
 7.00000   1.73860     1.73870 -0.00005
 8.00000   1.29290     1.29320 -0.00037
 9.00000   0.95419     0.95478 -0.00059
 10.00000  0.70118     0.70187 -0.00069
 11.00000  0.51422     0.51492 -0.00070
 12.00000  0.37702     0.37766 -0.00064
 13.00000  0.27672     0.27726 -0.00054
 14.00000  0.20351     0.20394 -0.00044
 15.00000  0.15007     0.15040 -0.00033
 16.00000  0.11101     0.11125 -0.00024
```

Note that we obtain no improvement in accuracy over the explicit scheme. The only advantage of the fully implicit scheme over the explicit scheme is the fact that we can, if we wish, increase α, that is, the time-step for a given price-step, without the system going unstable.

Crank-Nicolson

We now increase the time-step by a factor of 20, so that $\alpha = 8$. Otherwise all this proceeds as before:

Initialization:

```
M=20; nminus = 160; nplus = 160;
dx = 0.025; dtau = 0.005; alpha = dtau/dx^2
```

```
8.
```

```
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}];
wold = initial;
wvold = wold;
wnew = wold;
```

```
CNCMatrix[alpha_, nminus_, nplus_] :=
Sequence[Table[-alpha / 2, {nplus + nminus - 2}],
Table[1 + alpha, {nplus + nminus - 1}],
Table[-alpha / 2, {nplus + nminus - 2}]];

CNDMatrix[alpha_, vec_List] := Module[{temp},
temp = (1 - alpha) * vec +
     (alpha / 2) * (RotateRight[vec] + RotateLeft[vec]);
temp[[1]] = Simplify[First[temp] -
alpha * Last[vec] / 2];
temp[[-1]] = Simplify[Last[temp] - alpha * First[vec] / 2];
temp];

CMat = CNCMatrix[alpha, nminus, nplus];
```

Evolution:

```
For[m=2, m<=M+1, m++,
(wvold = wold;
wold = wnew;
rhs = CNDMatrix[alpha, Take[wold, {2, -2}]]+
Table[
Which[
k==1, alpha*(lower[[m-1]] + lower[[m]])/2,
k== nplus + nminus-1,  alpha*(upper[[m-1]] + upper[[m]])/2,
True, 0],
{k, 1, nplus + nminus-1}];
temp = CompTridiagSolve[CMat, rhs];
wnew = Join[{lower[[m]]}, temp, {upper[[m]]}])
]
```

Interpolation

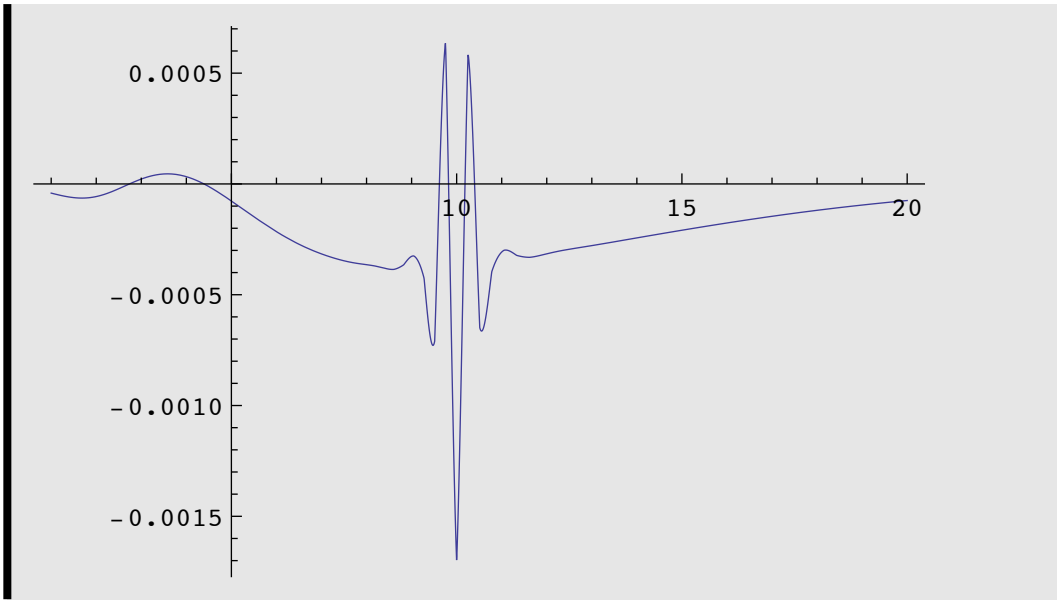
```
interpoldatab =
Table[{(k - nminus - 1)*dx, wnew[[k]]}, {k, 1, nminus+nplus+1}];
```

```
ufuncb = Interpolation[interpoldatab, InterpolationOrder -> 3];
```

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

Error Plot

```
Plot[Valuation[10, 0.05`, 0, S, 5, 0.2`] -
   BlackScholesPut[S, 10, 0.2`, 0.05`, 0, 5], {S, 1, 20},
 PlotPoints → 50, PlotRange → All]
```



```
samples = TableForm[Join[{{"S", "Crank-Nic", "Exact", "Error"}},
    Table[(PaddedForm[N[#1], {5, 5}] &) /@
        {S, Valuation[10, 0.05, 0, S, 5, 0.2],

      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, 0, 5]}, {S, 2, 16, 1}]]]
```

```
S         Crank-Nic  Exact    Error
 2.00000   5.78850   5.78860 -0.00006
 3.00000   4.80050   4.80050  0.00002
 4.00000   3.86160   3.86150  0.00003
 5.00000   3.02080   3.02090 -0.00008
 6.00000   2.31060   2.31080 -0.00022
 7.00000   1.73830   1.73870 -0.00032
 8.00000   1.29290   1.29320 -0.00036
 9.00000   0.95445   0.95478 -0.00033
 10.00000  0.70016   0.70187 -0.00171
 11.00000  0.51461   0.51492 -0.00031
 12.00000  0.37735   0.37766 -0.00032
 13.00000  0.27698   0.27726 -0.00028
 14.00000  0.20370   0.20394 -0.00024
 15.00000  0.15019   0.15040 -0.00021
 16.00000  0.11108   0.11125 -0.00018
```

Note that we obtain comparable accuracy to the explicit method with 1/20 the number of time-steps, but near the strike things are becoming "interesting". We obtain a sharply oscillatory error in the neighbourhood of the strike. Note that the size of the error is quite small, but it is steeply sloped. Bearing in mind the warnings given in our early lecture this is where we need to start thinking about getting the Greeks numerically..

Construction and Verification of Interpolated Valuation and Greeks

We begin by writing down a function that does numerical differentiation of a list. This function uses a simple central difference algorithm for points in the interior of the list. The end points are treated using a special difference algorithm.

```
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
]
```

Next we fix the values of the *k*-parameters:

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

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

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

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

```
points = Table[(k - nminus - 1)*dx, {k, 1, nminus + nplus + 1}];
```

```
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];
```

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

```
CNGamma[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]])
```

```
CNTheta[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]]
```

Delta Analysis

To investigate the errors in Delta we make a comparison with the exact solution, which we have already loaded as a function:

```
? BlackScholesPutDelta
```

```
Global`BlackScholesPutDelta
```

```
BlackScholesPutDelta[s_, k_, v_, r_, q_, t_] =
 -1/2 e^(-q t) (1 + Erf[-(t (-2 q+2 r+v^2)+2 Log[s/k])/(2 Sqrt[2] Sqrt[t] v)])
```

We make a table of values of the percentage errors in delta, given in the last column of the following table:

```
deltasamples = TableForm[
Table[Map[PaddedForm[N[#], {5, 4}]&,
{S, CNDelta[10, 0.05, 0, S, 5, 0.2],
BlackScholesPutDelta[S,10,0.2,0.05,0, 5],
100*(CNDelta[10,0.05,0,S,5,0.2]/BlackScholesPutDelta[S,10,0.2,0.05,0, 5
{S, 2, 16, 1}]]
```

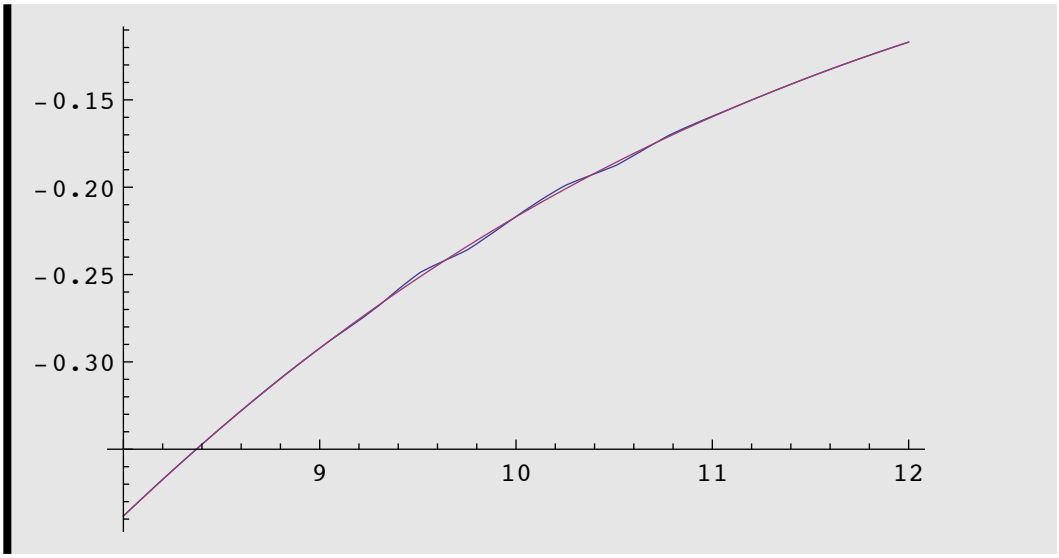
```
 2.0000  -0.9979 -0.9976  0.0324
 3.0000  -0.9721 -0.9719  0.0234
 4.0000  -0.8975 -0.8973  0.0205
 5.0000  -0.7786 -0.7785  0.0130
 6.0000  -0.6404 -0.6404 -0.0004
 7.0000  -0.5059 -0.5060 -0.0147
 8.0000  -0.3882 -0.3883 -0.0256
 9.0000  -0.2921 -0.2922 -0.0166
10.0000 -0.2168 -0.2169 -0.0631
11.0000 -0.1594 -0.1597 -0.2056
12.0000 -0.1169 -0.1170 -0.0505
13.0000 -0.0854 -0.0855 -0.0394
14.0000 -0.0624 -0.0624 -0.0436
15.0000 -0.0456 -0.0456 -0.0467
16.0000 -0.0333 -0.0334 -0.0518
```

The error in Delta is at most 0.2 % of the exact value, near the strike. This is still not too bad, and we can plot the error in Delta.

■ Plot of Computed and Exact Delta

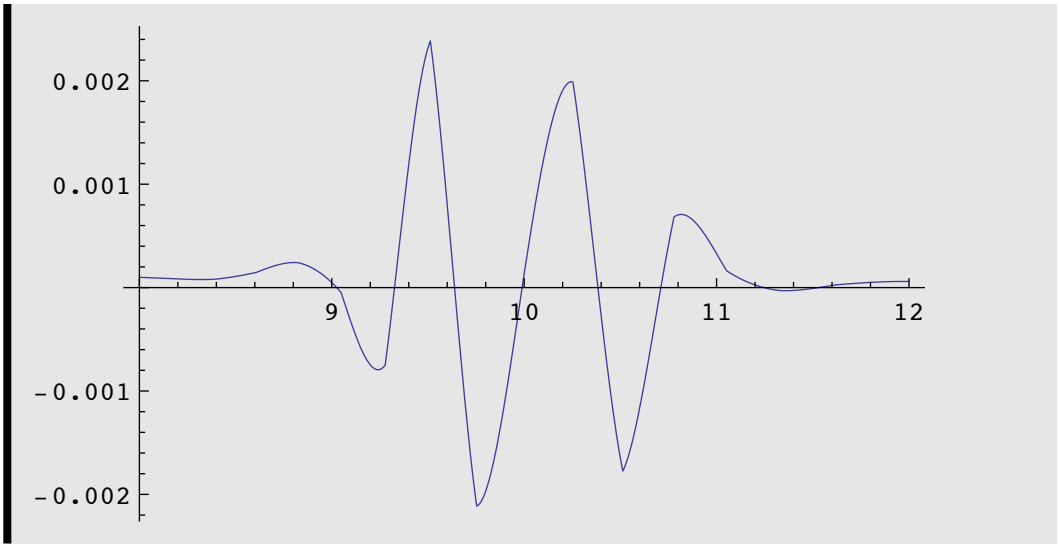
If we overlay the computed and exact Delta, the wobbles are starting to show:

```
Plot[{CNDelta[10, 0.05`, 0, S, 5, 0.2`],
  BlackScholesPutDelta[S, 10, 0.2`, 0.05`, 0, 5]}, {S, 8, 12},
 PlotPoints → 50, PlotRange → All]
```



Plotting the difference makes the problem clear:

```
Plot[CNDelta[10, 0.05`, 0, S, 5, 0.2`] -
  BlackScholesPutDelta[S, 10, 0.2`, 0.05`, 0, 5], {S, 8, 12},
 PlotPoints → 50, PlotRange → All]
```



Gamma Analysis

Here we tabulate the exact and numerical solution, with the percentage error in Gamma in the last column:

```
gammasamples = TableForm[Table[Map[PaddedForm[N[#], {5, 4}]&,
{S, CNGamma[10, 0.05, 0, S, 5, 0.2],
BlackScholesPutGamma[S,10,0.2,0.05,0, 5],
100*(CNGamma[10,0.05,0,S,5,0.2]/BlackScholesPutGamma[S,10,0.2,0.05,0,5]
```

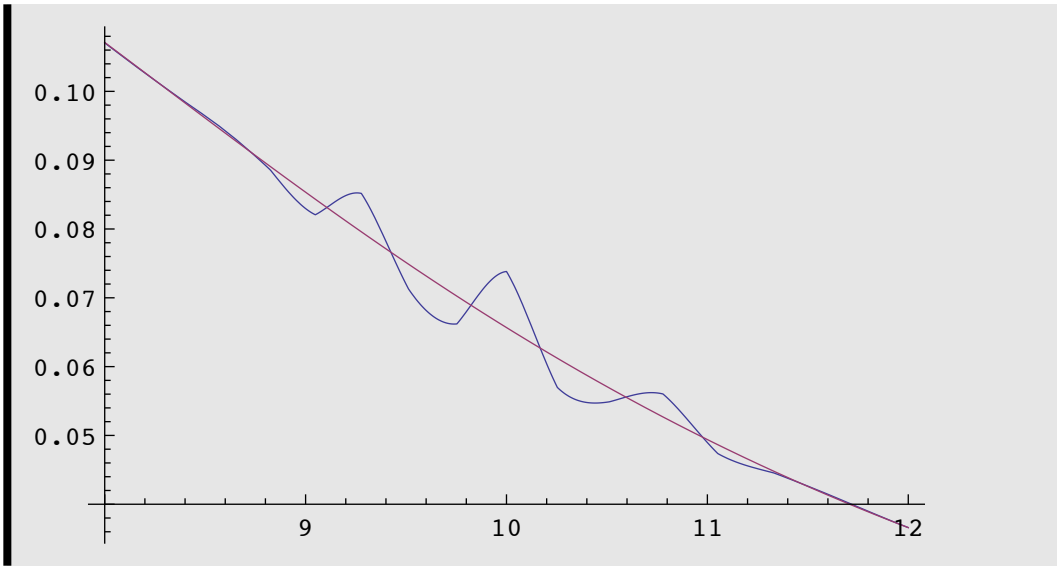
```
 2.0000   0.0084  0.0085 -1.2307
 3.0000   0.0481  0.0480  0.2056
 4.0000   0.1002  0.1000  0.2123
 5.0000   0.1332  0.1329  0.1812
 6.0000   0.1395  0.1394  0.1186
 7.0000   0.1275  0.1274  0.0496
 8.0000   0.1070  0.1071 -0.0589
 9.0000   0.0830  0.0853 -2.7845
10.0000   0.0738  0.0657  12.4110
11.0000   0.0490  0.0494 -0.8800
12.0000   0.0366  0.0366 -0.0030
13.0000   0.0269  0.0269 -0.0717
14.0000   0.0196  0.0196 -0.0693
15.0000   0.0143  0.0143 -0.0595
16.0000   0.0104  0.0104 -0.0500
```

The error in Γ is 12.4% of the exact value at the strike.

■ Gamma Plot

The wobbles in Γ are now manifest in a plot of the computed and exact version - the error is of a similar scale to the value of Γ:

```
Plot[{CNGamma[10, 0.05`, 0, S, 5, 0.2`],
  BlackScholesPutGamma[S, 10, 0.2`, 0.05`, 0, 5]}, {S, 8, 12},
 PlotPoints → 50, PlotRange → All]
```



Theta Analysis

Now we make a table of percentage errors with the Crank-Nicolson scheme:

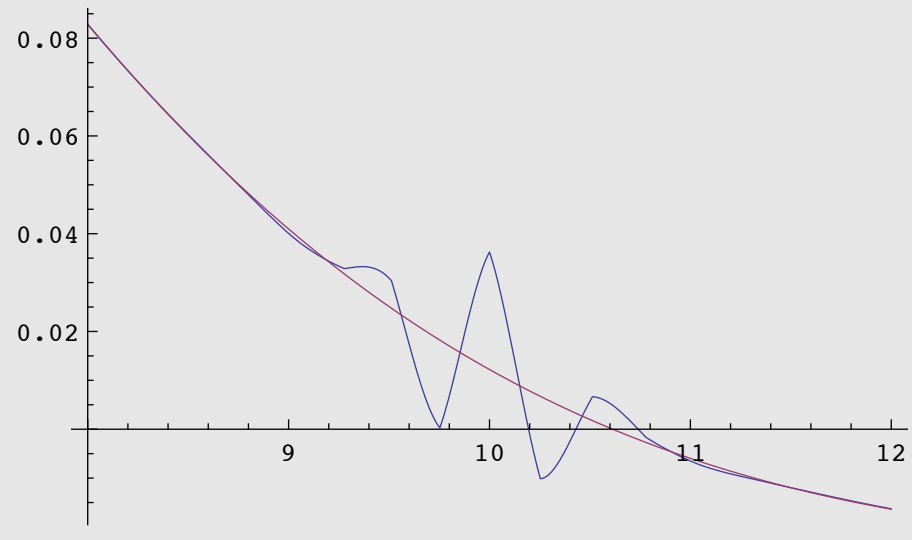
```
thetasamples = TableForm[
Table[Map[PaddedForm[N[#], {5, 4}]&,
{S, CNTheta[10, 0.05, 0, S, 5, 0.2],
BlackScholesPutTheta[S,10,0.2,0.05,0, 5],
100*(CNTheta[10,0.05,0,S,5,0.2]/BlackScholesPutTheta[S,10,0.2,0.05,0,5]
```

```
 2.0000    0.3885   0.3885   0.0008
 3.0000    0.3772   0.3772   0.0125
 4.0000    0.3406   0.3405   0.0134
 5.0000    0.2792   0.2792  -0.0157
 6.0000    0.2072   0.2073  -0.0541
 7.0000    0.1390   0.1391  -0.0780
 8.0000    0.0828   0.0829  -0.1037
 9.0000    0.0401   0.0410  -2.2123
 10.0000   0.0362   0.0122   196.8900
 11.0000  -0.0065  -0.0060   8.7857
 12.0000  -0.0163  -0.0164  -0.5050
 13.0000  -0.0214  -0.0214  -0.0244
 14.0000  -0.0230  -0.0230   0.0136
 15.0000  -0.0226  -0.0225   0.0668
 16.0000  -0.0209  -0.0209   0.1011
```

Near the strike the error in Theta peaks at 197% of the exact value.

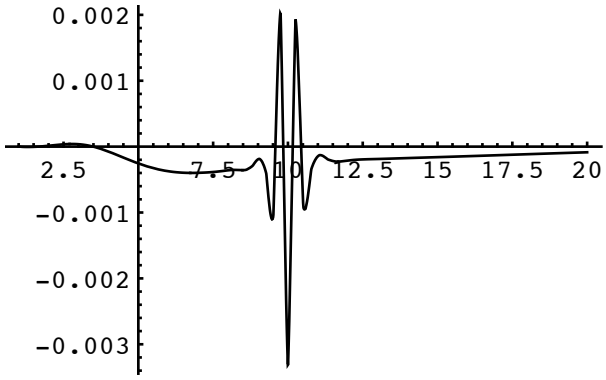
■ Theta Plots

```
Plot[{CNTheta[10, 0.05`, 0, S, 5, 0.2`],
  BlackScholesPutTheta[S, 10, 0.2`, 0.05`, 0, 5]}, {S, 8, 12},
 PlotPoints → 50, PlotRange → All]
```



Remark on Two-Time-Level Douglas

This time, there is no benefit in going to the simple Douglas scheme as described in Chapter 14. The oscillations in the neighbourhood of the strike are just as bad. For example, here is the error in valuation with $\alpha = 8$, for comparison. There are corresponding problems in the Greeks.



Douglas Three-Time-Level Solution

We have to work a little harder to develop a scheme that eliminates the oscillation problem. What we do is to implement the three-time-level extension of the Douglas scheme discussed previously. We shall not go into a detailed theoretical discussion of this algorithm - we shall content ourselves with an explicit demonstration that it works.

■ Necessary Functions

```
DougCMatrix[alpha_, nminus_, nplus_] :=
Sequence[Table[1-6*alpha, {nplus+nminus-2}], Table[10+12*alpha, {nplus+
Table[1-6*alpha, {nplus+nminus-2}]]
```

```
DougCCMatrix[alpha_, nminus_, nplus_] :=
Sequence[Table[1/8-alpha, {nplus+nminus-2}],
Table[5/4+2*alpha, {nplus+nminus-1}],
Table[1/8-alpha, {nplus+nminus-2}]]
```

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

Douglas Three Time-Level Solution Evolution

The initialization consists of defining the grid parameters, setting boundary and initial conditions and defining the various matrices.

```
dx = 0.025;
dtau = 0.005;
alpha = dtau/dx^2;
M=20;
nminus = 160;
nplus = 160;
alpha
```

```
8.
```

```
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];
```

```
CMat = DougCMatrix[alpha,nminus,nplus];
CCMat = DougCCMatrix[alpha,nminus,nplus];

CMatQ = DougCMatrix[alpha/4,nminus,nplus];
CMatH = DougCMatrix[alpha/2,nminus,nplus];

CCMatQ = DougCCMatrix[alpha/4,nminus,nplus];
CCMatH = DougCCMatrix[alpha/2,nminus,nplus];
```

■ Kick-Off Phase

This begins with the simple Douglas scheme with two time-levels and 1/4 the basic 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 = CompTridiagSolve[CMatQ, 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 = CompTridiagSolve[CCMatQ, rhs];
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 = CompTridiagSolve[CCMatH, rhs];
w[[3]] = Join[{w[[3,1]]}, vnew, {w[[3,nplus+nminus+1]]}];
wold = initial;
wvold = initial;
wnew = w[[3]];
```

■ Main Evolution Phase and Interpolation of Solution

```
For[m=3, m<=M+1, m++,
(wvold = wold;
wold = wnew;
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 = CompTridiagSolve[CCMat, rhs];
wnew = Join[{lower[[m]]}, temp, {upper[[m]]}])
]
```

```
points = Table[(k - nminus - 1) * dx,
{k, 1, nplus + nminus + 1}];
```

```
finalstep = wnew;
```

```
prevstep = wold;
```

```
pprevstep = wvold;
```

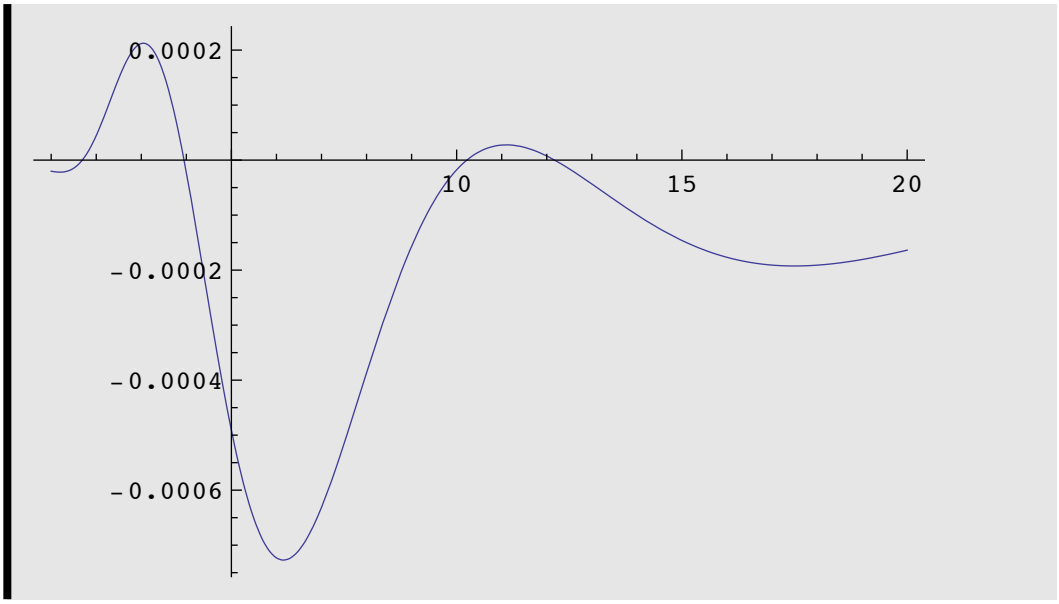
```
interpoldata = Transpose[{points, finalstep}];
```

```
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]]
```

Valuation Analysis

```
Plot[Valuation[10, 0.05`, 0, S, 5, 0.2`] -
   BlackScholesPut[S, 10, 0.2`, 0.05`, 0, 5], {S, 1, 20},
 PlotPoints → 50, PlotRange → All]
```



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

```
 2.0000   5.7886   5.7886   0.0000
 3.0000   4.8007   4.8005   0.0002
 4.0000   3.8615   3.8615  -0.0000
 5.0000   3.0204   3.0209  -0.0005
 6.0000   2.3101   2.3108  -0.0007
 7.0000   1.7380   1.7387  -0.0006
 8.0000   1.2928   1.2932  -0.0004
 9.0000   0.9546   0.9548  -0.0002
10.0000   0.7019   0.7019  -0.0000
11.0000   0.5149   0.5149   0.0000
12.0000   0.3777   0.3777   7.4967 × 10^-6
13.0000   0.2772   0.2773  -0.0000
14.0000   0.2038   0.2039  -0.0001
15.0000   0.1503   0.1504  -0.0001
16.0000   0.1111   0.1113  -0.0002
```

Construction and Verification of Interpolated Valuation and Greeks

In this section we define some functions to compute Delta, Gamma and Theta directly from the finite-difference grid.

```
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];
```

```
deltadata = listd[finalstep, dx] - 1/2 (kt - 1) finalstep;
```

```
gammadata = listd[deltadata, dx] - 1/2 (kt + 1) deltadata;
```

```
thetadata = (3 finalstep - 4 prevstep + pprevstep)/(2 dtau) -
    (1/4 (kt - 1)^2 + ko) 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];
```

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

```
DougGamma[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]])
```

```
DougTheta[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]]
```

Delta Analysis

The percentage error in Delta is given in the last column of the following table.

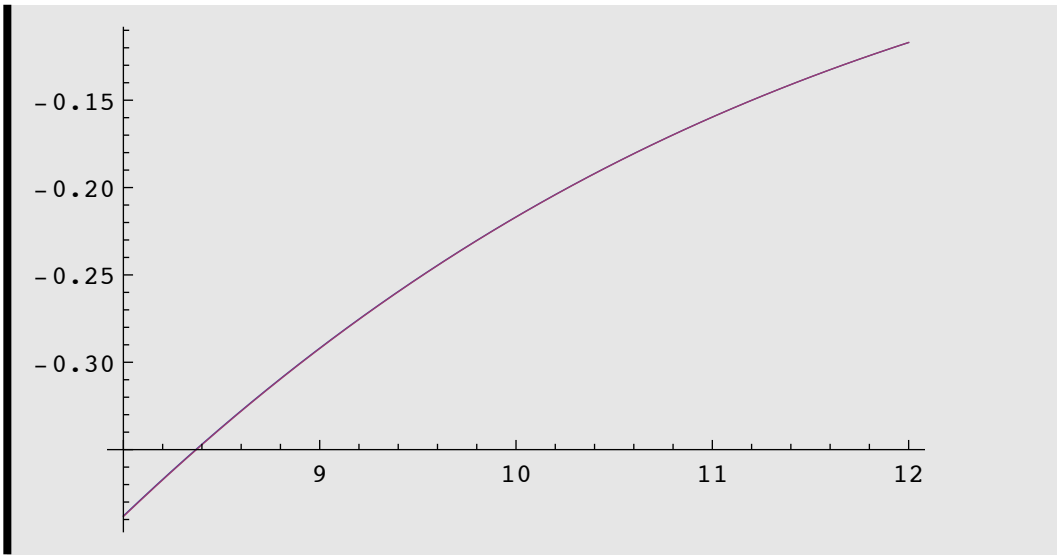
```
deltasamples = TableForm[
Table[Map[PaddedForm[N[#], {5, 4}]&,
{S, DougDelta[10, 0.05, 0, S, 5, 0.2],
BlackScholesPutDelta[S,10,0.2,0.05,0,5],
100*(DougDelta[10,0.05,0,S,5,0.2]/BlackScholesPutDelta[S,10,0.2,0.05,0,
{S, 2, 16, 1}]]
```

```
 2.0000  -0.9978 -0.9976  0.0186
 3.0000  -0.9722 -0.9719  0.0288
 4.0000  -0.8979 -0.8973  0.0634
 5.0000  -0.7789 -0.7785  0.0464
 6.0000  -0.6404 -0.6404 -0.0119
 7.0000  -0.5056 -0.5060 -0.0693
 8.0000  -0.3879 -0.3883 -0.0997
 9.0000  -0.2919 -0.2922 -0.0978
10.0000 -0.2168 -0.2169 -0.0695
11.0000 -0.1596 -0.1597 -0.0258
12.0000 -0.1170 -0.1170  0.0225
13.0000 -0.0855 -0.0855  0.0660
14.0000 -0.0625 -0.0624  0.0975
15.0000 -0.0456 -0.0456  0.1126
16.0000 -0.0334 -0.0334  0.1088
```

The error in Delta is at most 0.1% of the exact value.

■ Delta Plots

```
Plot[{DougDelta[10, 0.05`, 0, S, 5, 0.2`],
  BlackScholesPutDelta[S, 10, 0.2`, 0.05`, 0, 5]}, {S, 8, 12},
 PlotPoints → 50, PlotRange → All]
```



Gamma Analysis

The percentage error in Gamma is given by the last column of the following table:

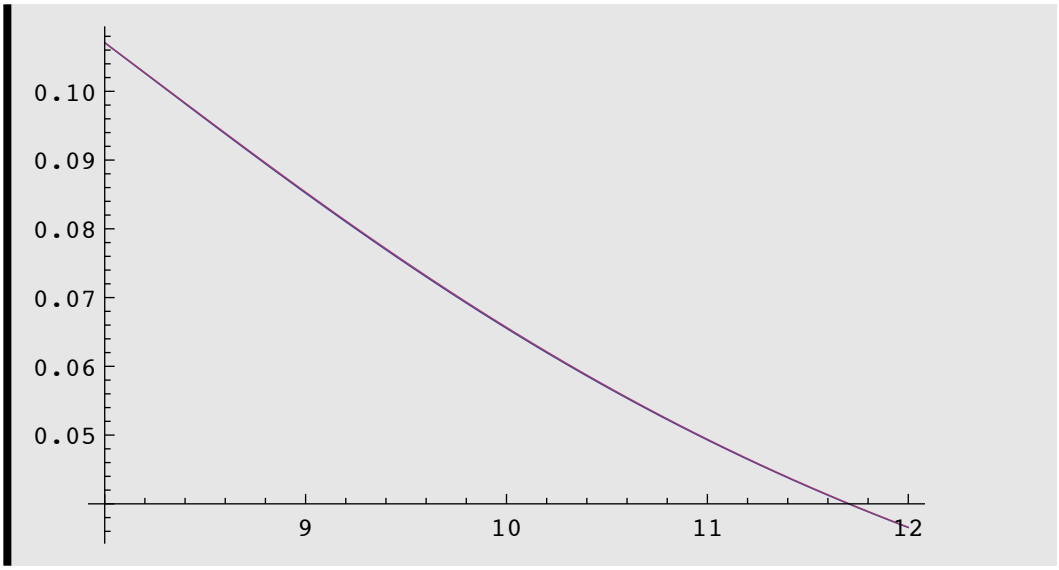
```
gammasamples = TableForm[Table[Map[PaddedForm[N[#], {5, 4}]&,
{S, DougGamma[10, 0.05, 0, S, 5, 0.2],
BlackScholesPutGamma[S,10,0.2,0.05,0,5],
100*(DougGamma[10,0.05,0,S,5,0.2]/BlackScholesPutGamma[S,10,0.2,0.05,0,
{S,2,16,1}]]
```

```
2.0000   0.0084  0.0085 -0.3310
3.0000   0.0477  0.0480 -0.6616
4.0000   0.1001  0.1000  0.0932
5.0000   0.1335  0.1329  0.4113
6.0000   0.1398  0.1394  0.3334
7.0000   0.1276  0.1274  0.1276
8.0000   0.1070  0.1071 -0.0616
9.0000   0.0852  0.0853 -0.1835
10.0000  0.0655  0.0657 -0.2327
11.0000  0.0493  0.0494 -0.2225
12.0000  0.0365  0.0366 -0.1710
13.0000  0.0268  0.0269 -0.0961
14.0000  0.0196  0.0196 -0.0127
15.0000  0.0143  0.0143  0.0671
16.0000  0.0104  0.0104  0.1348
```

The error is at most 0.7% of the exact value - near the strike it is now only 0.2%.

■ Plot of Computed and Exact Gamma

```
Plot[{DougGamma[10, 0.05`, 0, S, 5, 0.2`],
   BlackScholesPutGamma[S, 10, 0.2`, 0.05`, 0, 5]}, {S, 8, 12},
 PlotPoints → 50, PlotRange → All]
```



Theta Analysis

The percentage error in Theta is given by the last column of the following table:

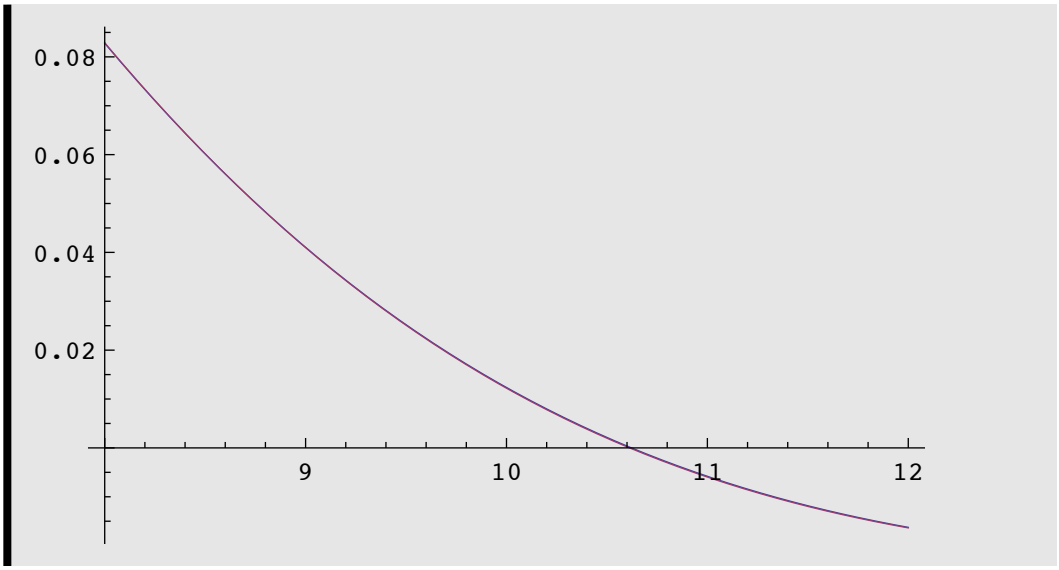
```
thetasamples = TableForm[
Table[Map[PaddedForm[N[#], {5, 4}]&,
{S, DougTheta[10, 0.05, 0, S, 5, 0.2],
BlackScholesPutTheta[S,10,0.2,0.05,0,5],
100*(DougTheta[10,0.05,0,S,5,0.2]/BlackScholesPutTheta[S,10,0.2,0.05,0,
{S, 2, 16, 1}]]
```

```
 2.0000   0.3885  0.3885 -0.0085
 3.0000   0.3773  0.3772  0.0260
 4.0000   0.3407  0.3405  0.0498
 5.0000   0.2792  0.2792 -0.0232
 6.0000   0.2071  0.2073 -0.1296
 7.0000   0.1389  0.1391 -0.1795
 8.0000   0.0828  0.0829 -0.1116
 9.0000   0.0410  0.0410  0.1566
10.0000   0.0124  0.0122  1.2132
11.0000 -0.0058 -0.0060 -2.5962
12.0000 -0.0163 -0.0164 -0.6827
13.0000 -0.0214 -0.0214 -0.2323
14.0000 -0.0230 -0.0230  0.0372
15.0000 -0.0226 -0.0225  0.2296
16.0000 -0.0210 -0.0209  0.3672
```

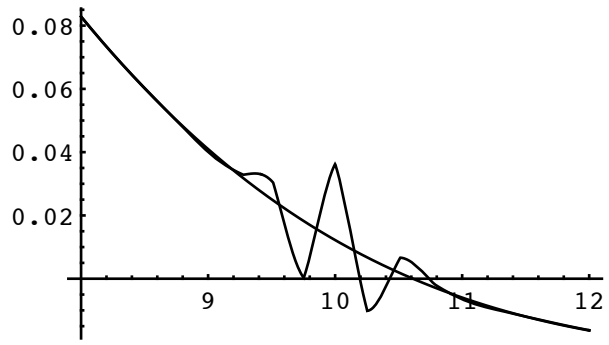
Near the strike the error in θ peaks at 2.6% of the exact value. Note that with the two-time-level Crank-Nicolson scheme the error was almost 200%.

Theta Plots

```
Plot[{DougTheta[10, 0.05`, 0, S, 5, 0.2`],
  BlackScholesPutTheta[S, 10, 0.2`, 0.05`, 0, 5]}, {S, 8, 12},
 PlotPoints → 50, PlotRange → All]
```



As a reminder, here is the corresponding plot for the Crank-Nicolson scheme.



Summary

The introduction of implicit schemes is motivated by the desire to get accurate and stable solutions for larger values of α. When we need to compute the values of both a function and its first and second derivatives, popular schemes such as the ordinary two-time-level Crank-Nicolson scheme are unsuitable for option-pricing problems, unless small to moderate values of α are used. This is because of the fact that non-smoothness in initial data (almost always present in option payoffs) propagates through the solution causing small oscillatory errors. When amplified by the process of differentiation, these may cause substantial errors in the Greeks. The errors in the valuation may be deceptively small - to quote Wilmott *et al* (1993) when discussing a verification example with the Crank-Nicolson scheme, they remark that "Even with $\alpha = 10$, the numerical and exact results differ only marginally." This is absolutely right, at least when we just look at the valuation. When we inspect the Greeks a rather more disturbing picture emerges. We have given an example where estimates of θ from a computation with $\alpha = 8$ are in error by about 200%.

The oscillation problem and the resulting corruption of the Greeks can be cured by the use of a more suitable difference scheme. When the initial data were smooth, we saw previously that the two-time-level Douglas scheme was sufficient, but in the presence of non-smooth data the corresponding three-time-level Douglas scheme cures the problems quite dramatically.

The transition to such a scheme for practitioners should not be a great leap. We have already seen that the two-time-level Douglas scheme is the natural implicit generalization of the trinomial model, when carried out on a grid rather than a tree. The next step to a three-time-level scheme allows larger time steps still to be taken without a loss of accuracy or corruption of the Greeks.