OpenGamma Quantitative Research

# Numerical Solutions to PDEs with Financial Applications

Richard White
Richard@opengamma.com

OpenGamma Quantitative Research n. 10 | First version: September 24, 2012; this version February 26, 2013

**Abstract**

We present details of the 1D PDE solver used in the OpenGamma Platform, showing how it can price European and American options, with and without barrier features. All the results in this paper we generated using MATLAB, and this code is included here www.opengamma.com/downloads/financial-pde-solving-matlab-examples.zip.

# Contents

# 1 Introduction

Research into solving PDEs on a computer goes back almost to the invention of the programmable electronic computer, with finite-difference methods in the 1950s and finite-element methods in the 1960s. The sophistication used (and required) in finance tends to be lower than in other applied maths/engineering disciplines. Two monographs [TR00, Duf06] cover financial derivatives pricing with finite difference methods exclusively. Many finance textbooks have a few chapters on the finite-difference method to various levels of sophistication [Wil06, AP10, FR08, Cla11, Hir13] and some [Sey09] extend coverage to finite-element methods. There is of course a plethora of books from other fields dealing with numerical solutions of PDEs. The monograph [MM05] deals mainly with finite-difference, but also touches on finite-element and finite-volume methods. Books on numerical analysis such as [Fai11, PTVF07] have chapters on finite-difference methods. This is not an extensive literature search on the subject, but covers the material that the author has actually read.

The author does not have a background in numerical solutions to PDEs, nor a background in applied maths *per se*.[1] The idea of this note is to present all the details of how we have implemented the finite-difference method for pricing finance derivatives that depend on a single stochastic driver (i.e. the PDE is 1D in its spatial co-ordinate). To this end, we hope this will be accessible to anyone with a numerical background, without relying on the usual special case simplifications seen in many presentations.

# 2 The forms of Partial Differential Equations found in finance

For a contingent claim on a single asset, the generic PDE can be written as

$$\frac{\partial V}{\partial t} + a(x,t)\frac{\partial^2 V}{\partial x^2} + b(x,t)\frac{\partial V}{\partial x} + c(x,t)V = 0 \tag{1}$$

where $t$ either represents calendar time or time-to-expiry, $x$ represents either the value of the underlying asset or some monotonic function of it (e.g. $\log(S_t)$; log-spot) and $V$ is the value of the claim (as a function of $x$ and $t$). The terms $a(\cdot)$, $b(\cdot)$ and $c(\cdot)$ are the *diffusion*, *convection* and *reaction* coefficients respectively, and this type of PDE is known as a *convection-diffusion PDE*.[2] This type of PDE can also be written in the form

$$\frac{\partial V}{\partial t} + a(x,t)\frac{\partial}{\partial x}\left(\alpha(x,t)\frac{\partial V}{\partial x}\right) + b(x,t)\frac{\partial}{\partial x}\left(\beta(x,t)V\right) + c(x,t)V = 0 \tag{2}$$

This form occurs in the Fokker-Planck (Kolmogorov forward) equation that describes the evolution of the transition density of a stochastic quantity (e.g. a stock value). This can be put in the form of equation 1 if the functions $\alpha$ and $\beta$ are both once differentiable in $x$ - although it is usually better to directly discretise the form given. We do not consider this form further in the paper.

[1] actually experimental particle physics

[2] A source term $d(x,t)$ may also be present, but this does not change the way the PDE is solved.

## 2.1 Initial and Boundary Conditions

We will solve the PDE from an initial condition $V(x,0) = f(x)$ to some terminal time $T$,[3] so the time domain is naturally bound. In the space domain boundary conditions must be provided. These can either be determined by the 'physics' of the problem (a knock-out barrier option has a very clear choice of a boundary) or set sufficiently far out as to not affect the *interesting* part of the solution. The common choices are:

- *Dirichlet* boundary conditions, which take the form

$$V(\mathbf{x},t) = g(\mathbf{x},t) \quad \forall \mathbf{x} \in \Gamma(t) \tag{3}$$

  where $\Gamma(t)$ is the (possibly time-dependent) boundary of the region. In the 1D case this simplifies to

$$V(x_b(t),t) = g(t) \tag{4}$$

  meaning that on some time-dependent boundary $x_b(t)$ the solution has a deterministic value $g(t)$.

- *Neumann* boundary conditions, take the form

$$\frac{\partial V}{\partial n}(\mathbf{x},t) = h(\mathbf{x},t) \quad \forall \mathbf{x} \in \Gamma(t) \tag{5}$$

  where $\mathbf{n}$ is the vector normal to boundary $\Gamma(t)$.[4] In our 1D case this simplifies to

$$\frac{\partial V}{\partial x}(x_b(t),t) = h(t) \tag{6}$$

- *Robin* boundary condition, is a linear combination of Dirichlet and Neumann boundary conditions, and can be written as

$$\alpha(\mathbf{x},t)V(\mathbf{x},t) + \beta(\mathbf{x},t)\frac{\partial V}{\partial n}(\mathbf{x},t) = \gamma(\mathbf{x},t) \quad \forall \mathbf{x} \in \Gamma(t) \tag{7}$$

  which again simplifies to

$$\alpha(t)V(x_b(t),t) + \beta(t)\frac{\partial V}{\partial x}(x_b(t),t) = \gamma(t) \tag{8}$$

  in the 1D case.

- *Linearity* boundary condition is

$$\frac{\partial^2 V}{\partial n^2}(\mathbf{x},t) = 0 \quad \forall \mathbf{x} \in \Gamma(t) \tag{9}$$

  which is

$$\frac{\partial^2 V}{\partial x^2}(x_b(t),t) = 0 \tag{10}$$

  in the 1D case. This conveys less information that either Dirichlet or Neumann boundary conditions. It is useful when little is known about the solution other than it becomes linear sufficiently far from the interesting part of the solution.

[3] Depending on the application either the whole solution for $t \leq T$ (e.g. if we are solving the forward equation for call price as a function of expiry and strike) or just the $t = T$ case (e.g. if we are solving the backwards PDE for a single option price) will be of interest.

[4] $\frac{\partial V}{\partial n} \equiv \mathbf{n} \cdot \nabla V$

In the 1D case we must provide an initial condition and two boundary conditions. These conditions must be *consistent*, meaning that either

- $f(x_b) = g(0)$ for a Dirichlet boundary,
- $\frac{\partial f}{\partial x}(x_b) = h(0)$ for a Neumann boundary,
- $\alpha(0)f(x_b) + \beta(0)\frac{\partial f}{\partial x}(x_b) = \gamma(0)$ for a Robin boundary, or
- $\frac{\partial^2 f}{\partial x^2}(x_b) = 0$ for a linearity boundary,

at each of the two boundaries. Failure to provide consistent boundary conditions will greatly reduce accuracy of the solution.

## 2.2 Example of Financial PDEs

### 2.2.1 Generalised Black-Scholes Equation (Local Volatility)

For a stock $S_t$ following the SDE in the risk neutral measure

$$\frac{dS_t}{S_t} = (r_t - q_t)dt + \sigma(t, S_t)dW_t \tag{11}$$

where $r_t$ is the (term-structure of) risk free-rate, $q_t$ is the dividend yield[5] and $\sigma(S_t, t)$ is the instantaneous (or local) volatility, the price at $t = 0$ of a contingent claim that pays $H(S_T)$ at time $T$, is given by $V(S_0, 0)$ where $V$ satisfies the PDE

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma(S,t)^2 S^2 \frac{\partial^2 V}{\partial S^2} + (r_t - q_t)S\frac{\partial V}{\partial S} - r_t V = 0 \tag{12}$$

and the equation is solved backwards in time from the 'initial' condition $V(S_T, T) = H(S_T)$.

To put this in the generic form, we employ the change of variable $\tau = T - t$ to write

$$\frac{\partial \bar{V}}{\partial \tau} - \frac{1}{2}\bar{\sigma}(S,\tau)^2 S^2 \frac{\partial^2 \bar{V}}{\partial S^2} - (\bar{r}_\tau - \bar{q}_\tau)S\frac{\partial \bar{V}}{\partial S} + \bar{r}_\tau \bar{V} = 0 \tag{13}$$

where $\bar{V}(\cdot, \tau) \equiv V(\cdot, T - \tau)$, $\bar{\sigma}(\cdot, \tau) \equiv \sigma(\cdot, T - \tau)$, $\bar{r}_\tau \equiv r_{T-\tau}$ and $\bar{q}_\tau \equiv q_{T-\tau}$. We then solve the equation 'forward' in time from the initial condition $\bar{V}(S_T, 0) = H(S_T)$ at $\tau = 0$ to $\tau = T$.

### 2.2.2 Forward PDE

For European call (or put) options with strike, $K$, and expiry, $T$, the price $C(K, T)$ as a function of strike and expiry is given by the PDE (see for example [Whi12b, Cla11] for derivations)

$$\frac{\partial C}{\partial T} - \frac{1}{2}\sigma^2(K,T)K^2\frac{\partial^2 C}{\partial K^2} + (r_T - q_T)K\frac{\partial C}{\partial K} + q_T C = 0 \tag{14}$$

with initial condition $C(K, 0) = (\omega(S_0 - K))^+$, where $\omega = +1$ for calls and $-1$ for puts . In this case we solve up to some horizon $T^*$, and the whole of the time domain is in general of interest.

[5] if this were a forex example, $S$ would be the exchange rate (the cost of a unit of foreign currency in domestic currency ) $r_t \to r_t^d$ would be the domestic risk-free rate and $y_t \to r_t^f$ would be the foreign risk-free rate.

It is very common to make the change of variable $x = \log(K/S_0)$, in which case the PDE becomes

$$\frac{\partial C}{\partial T} - \frac{1}{2}\sigma^2(x,T)\frac{\partial^2 C}{\partial x^2} + (r - q + \frac{1}{2}\sigma^2(x,T))\frac{\partial C}{\partial x} + qC = 0 \tag{15}$$

with $C(x,0) = S_0(\omega(1-e^x))^+$, which is of course still in the generic form.[6]

# 3 Solving the PDE

We solve the PDE on a rectangular domain in 1D space and time with $t \in [0,T]$ and $x \in [x_{min}, x_{max}]$. This domain is discretised on a grid with $N+2$ nodes $x_0 = x_{min}, x_1, \ldots, x_N, x_{N+1} = x_{max}$ in the space direction and $M+2$ nodes $t_0 = 0, t_1, \ldots, t_M, t_{M+1} = T$ in the time direction. A general node is indicated by the two indices $i, j$ and is at $(x_i, t_j)$. Although uniform grids (i.e. $x_i - x_{i-1} = dx \,\forall\, i$ and/or $t_j - t_{j-1} = dt \,\forall\, j$) are common in text books, they are not necessary (or always desirable), and they are not assumed. Grid generation is discussed in appendix B.

Whenever possible, we will write systems of equations in compact matrix notation, which is detailed in appendix A.

## 3.1 Differentiation Matrix

If a function $f(x)$ is sampled at a set of points $\mathbf{x}$, then the function values at these points $y_i = f(x_i)$ form a vector $\mathbf{y}$ of length $N+2$. An estimate of the first derivative of the function $f'(x)$ at the points can be made by $\mathbf{y}' = \mathbf{D}^1\mathbf{y}$, where $\mathbf{D}^1$ is a first-order differentiation matrix which depends on $\mathbf{x}$. The second derivative is similarly given by $\mathbf{y}'' = \mathbf{D}^2\mathbf{y}$, where $\mathbf{D}^2$ is a second-order differentiation matrix.[7]

Below we give two examples of differentiation matrices, one of which is sparse (and tridiagonal), while the other is dense.

### 3.1.1 Three-point Estimate

If we fit a quadratic $y = a_i(x-x_i)^2 + b_i(x-x_i) + c_i$ through the three points at $i-1$, $i$ and $i+1$, it is easy to show that

$$\begin{aligned} a_i &= \frac{\Delta_{i-1}y_{i+1} - (\Delta_{i-1}+\Delta_i)y_i + \Delta_i y_{i-1}}{\Delta_{i-1}\Delta_i(\Delta_{i-1}+\Delta_i)} \\ b_i &= \frac{\Delta_{i-1}^2 y_{i+1} - (\Delta_{i-1}^2 - \Delta_i^2)y_i - \Delta_i^2 y_{i-1}}{\Delta_{i-1}\Delta_i(\Delta_{i-1}+\Delta_i)} \\ c_i &= y_i \end{aligned} \tag{16}$$

where $\Delta_i \equiv x_{i+1} - x_i$. The quadratic can be used to estimate the derivative at any point between $x_{i-1}$ and $x_{i+1}$. In particular for the central point $x_i$, we have

$$\begin{aligned} y_i' &= b_i \\ y_i'' &= 2a_i \end{aligned} \tag{17}$$

[6] We could of course have done the same thing with equation 13 by setting either $x = \log(S_\tau)$ or $x = \log(S_\tau/S_0)$.

[7] Clearly setting $\mathbf{D}^2 = (\mathbf{D}^1)^2$ is correct, although this may not always be the best thing to do.

at the left point,

$$\begin{aligned} y'_{i-1} &= -2a_i\Delta_{i-1} + b_i \\ y''_{i-1} &= 2a_i \end{aligned} \tag{18}$$

and at the right point

$$\begin{aligned} y'_{i+1} &= 2a_i\Delta_i + b_i \\ y''_{i+1} &= 2a_i \end{aligned} \tag{19}$$

In matrix form $\mathbf{D}^1$ and $\mathbf{D}^2$ are both tridiagonal except for the first and last rows, which have three rather than two non-zero entries (corresponding to the forward and backwards differencing done). If instead we had chosen to set $\mathbf{D}^2 = (\mathbf{D}^1)^2$ then $\mathbf{D}^2$ would be a pentadiagonal matrix.[8] While higher order polynomials could potentially give better derivative estimates,[9] tri-diagonal systems are desirable because they can be solved in order $N$ time.

### 3.1.2 Spectral methods

If $N+2$ Chebyshev points[10] are used, then the $N+1$ order polynomial is known as the Chebyshev polynomial, which can produce dramatically better estimates of the derivatives than equispaced points (see [Tre00] for details). The resultant Chebyshev differentiation matrices are dense, and while they are better than the three-point finite difference discussed above for many applications, we do not find a great advantage (or disadvantage) for simple applications such as pricing call options.

There is a deep connection between Chebyshev differentiation matrices and the Discrete Fourier Transform (again see [Tre00] for details), which means that the Fast Fourier Transform (FFT) can be used in place of solving a dense matrix system.

## 3.2 Discretising the PDE

### 3.2.1 Spatial Derivatives

To begin with, we just consider discretisation in the spatial dimension. Given the time-dependent vector $\mathbf{v}(t) = (V(x_0,t), V(x_1,t), \ldots, V(x_N,t), V(x_{N+1},t))^T$, the derivatives are given by

$$\begin{aligned} \frac{\partial V}{\partial x} &\approx \mathbf{D}^1\mathbf{v}(t) \\ \frac{\partial^2 V}{\partial x^2} &\approx \mathbf{D}^2\mathbf{v}(t) \end{aligned} \tag{20}$$

If we similarly define $\mathbf{a}(t) = (a(x_0,t), a(x_1,t), \ldots, a(x_N,j), a(x_{N+1},t)^T$, and likewise for $\mathbf{b}(t)$ and $\mathbf{c}(t)$, then equation 1 may be written as a set of $N+2$ coupled ordinary differential equations (ODEs)

$$\begin{aligned} \frac{d\mathbf{v}(t)}{dt} &= -\mathbf{L}(t)\mathbf{v}(t) \\ \text{where} \quad \mathbf{L}(t) &\equiv \mathbf{A}(t)\mathbf{D}^2 + \mathbf{B}(t)\mathbf{D}^1 + \mathbf{C}(t) \end{aligned} \tag{21}$$

[8] except for the first and last few rows

[9] depending on the placement of points, high order polynomials can spectacularly fail to represent the true function, oscillating wildly while equaling the function at the points. This is know as Runge's phenomenon.

[10] Chebyshev points are given by $x_i = \left(\frac{x_{min}+x_{max}}{2}\right) - \left(\frac{x_{max}-x_{min}}{2}\right)\cos\left(\frac{i\pi}{N+1}\right)$

and $\mathbf{A}(t) = \text{diag}(\mathbf{a}(t))$ etc (i.e. the diagonal matrix with the elements of $\mathbf{a}$ along the lead diagonal). This has a formal solution

$$\begin{aligned} \mathbf{v}(t) &= \exp\left(-\mathbf{\Lambda}(t)\right)\mathbf{v}(0) \\ \text{where} \quad \mathbf{\Lambda}(t) &\equiv \int_0^t \mathbf{L}(t')dt' \end{aligned} \tag{22}$$

the integral of a matrix is performed element-by-element and the matrix exponential is defined as the power series $\exp(\mathbf{A}) \equiv \mathbf{I} + \mathbf{A} + \frac{\mathbf{A}^2}{2!} + \frac{\mathbf{A}^3}{3!} + \ldots$. We now proceed to solve this ODE numerically.

### 3.2.2 Time Stepping

If time is also discretised, and we define $\mathbf{v}^j \equiv \mathbf{v}(t_j)$, the simplest approximation to the time derivative $\frac{d\mathbf{v}(t)}{dt}$ is forward differencing:

$$\frac{d\mathbf{v}(t)}{dt} \approx \frac{\mathbf{v}^{j+1} - \mathbf{v}^j}{\delta_j} \tag{23}$$

where $\delta_j \equiv t_{j+1} - t_j$. Using the usual notation that $\mathbf{L}^j \equiv \mathbf{L}(t_j)$, $\mathbf{A}^j \equiv \mathbf{A}(t_j)$ etc, we have

$$\begin{aligned} \mathbf{v}^{j+1} &= (\mathbf{I} - \delta_j \mathbf{L}^j)\mathbf{v}^j \\ \text{where} \quad \mathbf{L}^j &= \mathbf{A}^j\mathbf{D}^2 + \mathbf{B}^j\mathbf{D}^1 + \mathbf{C}^j \end{aligned} \tag{24}$$

This is known as the explicit method. While it is attractive in that it only involves matrix multiplication, it suffers from catastrophic stability problems. Appendix D outlines von Neumann stability analysis, which shows that for a uniform grid the time steps must obey

$$\delta \leq \frac{\Delta^2}{2a} \tag{25}$$

where $a$ is taken as the largest value of $a_{i,j}$ on the grid. For typical parameters, unless the time stepping is incredibly small, any noise in the system (e.g. from numerical rounding errors) will be amplified rather than damped at each time step, which will eventually lead to violent instability, usually manifesting as spikes at the Nyquist frequency.

If we keep the same time stepping, but evaluate the right hand side of equation 21 at time $t = t_{j+1}$ rather than $t = t_j$, then the system becomes

$$(\mathbf{I} + \delta_j \mathbf{L}^{j+1})\mathbf{v}^{j+1} = \mathbf{v}^j \tag{26}$$

This is know as the implicit method, since we must solve a matrix system at each time step. Stability analysis (see appendix D) shows that this method is unconditionally stable[11], which is a huge advantage over the explicit method despite having to solve matrix systems. However, since both the explicit and implicit methods are using one-sided differencing, they are only $\mathcal{O}(\delta_j)$ accurate - i.e. their accuracy scales linearly with the time step [Wil06, Duf06, TR00].

[11] any grid size is stable, in that the solution will not blow up - this is not the same as saying that any grid size will give an accurate answer.

### 3.2.3 The Theta Method

If we take a weighted sum of equations 24 and 26, we have the system

$$(\mathbf{I} + \theta\delta_j \mathbf{L}^{j+1})\mathbf{v}^{j+1} = (\mathbf{I} - (1-\theta)\delta_j \mathbf{L}^{j})\mathbf{v}^{j} \tag{27}$$

where $\theta \in (0, 1)$. The explicit and implicit cases are recovered by setting $\theta = 0$ and $\theta = 1$ respectively. We have in effect discretised the ODE at the virtual grid point $t_{j+\theta} = (1-\theta)t_j + \theta t_{j+1}$. If we actually carry out the discretisation at $t_{j+\theta}$ by letting $\mathbf{v}(t_{j+\theta}) \approx (1-\theta)\mathbf{v}_j + \theta\mathbf{v}_{j+1}$ and $\frac{d\mathbf{v}(t_{j+\theta})}{dt} \approx \frac{\mathbf{v}^{j+1}-\mathbf{v}^{j}}{\delta_j}$ then we arrive at

$$(\mathbf{I} + \theta\delta_j \mathbf{L}^{j+\theta})\mathbf{v}^{j+1} = (\mathbf{I} - (1-\theta)\delta_j \mathbf{L}^{j+\theta})\mathbf{v}^{j} \tag{28}$$

where $\mathbf{L}^{j+\theta} = \mathbf{L}(t_{j+\theta})$. When $\mathbf{L}$ is time independent[12], or $\theta = 0, 1$, then both forms are equivalent. In the general case, we will need to investigate which is the better form to use.

For either form, when $\theta = 0.5$, the time step is actually central differencing (since the time at which the spatial discretisation happens is $(t_j + t_{j+1})/2$), so is $\mathcal{O}(\delta_j^2)$ accurate. Furthermore, for $\theta \geq 0.5$ the system is unconditionally stable. This choice of $\theta = 0.5$ (at least in the case of differentiation matrices arising from 3-point central differencing) is know as the Crank-Nicolson method [CN47].

Another way of looking at this is to use the formal solution to write

$$\begin{aligned}\mathbf{v}^{j+1} &= \exp\left(-\int_{t_j}^{t_{j+1}} \mathbf{L}(t')dt'\right)\mathbf{v}^{j} \approx \exp\left(-\delta_j \mathbf{L}^{j+\frac{1}{2}}\right)\mathbf{v}^{j} \\ &= \left[\mathbf{I} - \delta_j \mathbf{L}^{j+\frac{1}{2}} + \frac{1}{2!}(\delta_j \mathbf{L}^{j+\frac{1}{2}})^2 - \frac{1}{3!}(\delta_j \mathbf{L}^{j+\frac{1}{2}})^3 + \dots\right]\mathbf{v}^{j}\end{aligned} \tag{29}$$

Expanding the theta method to third order gives

$$\mathbf{v}^{j+1} = \left[\mathbf{I} - \delta_j \mathbf{L}^{j+\theta} + \theta(\delta_j \mathbf{L}^{j+\theta})^2 - \theta^2(\delta_j \mathbf{L}^{j+\theta})^3 + \dots\right]\mathbf{v}^{j} \tag{30}$$

It is clear that for $\theta = 0.5$, these agree to the second order term (i.e. they differ in the third order and higher terms). The reason we don't do the exact step and 'simply' take the exponent of a matrix, is because this is a relatively expensive operation,[13] and better accuracy (for a given computation budget) will be obtained by using an approximation to the exponential (e.g. Crank-Nicolson) will smaller time steps.

This idea is explored further in [Duf06], where it is shown that second order accuracy can be obtained by combining an implicit step of size $\delta_j$ and two implicit steps of size $\delta_j/2$ (i.e. covering the same time step). The result at $t_{j+1}$ is then set to $\mathbf{v}^{j+1} = 2\mathbf{v}^{j+1}_{\text{half}} - \mathbf{v}^{j+1}_{\text{full}}$. While this does avoid the problems with Crank-Nicolson, it also means taking three times as many time steps. This is a quite general technique known as *Richardson extrapolation*. This is discussed further in section 4.1.2.

[12] i.e. there is no term structures of rate, dividends or volatility.

[13] MATLAB computes the matrix exponential by a scaling and squaring algorithm with a Padé approximation. An alternative methods is via the eigendecomposition of the matrix.

### 3.2.4 Other Time Stepping

Numerical solutions to ODEs like equation 21 can be found in standard text books, e.g. [PTVF07]. These generally involve multiple time levels and therefore need multiple initial conditions.[14] A $\mathcal{O}(\delta_j^2)$ scheme is central differencing, where the time derivative is approximated as

$$\frac{d\mathbf{v}(t)}{dt} \approx \frac{\delta_{j-1}^2 \mathbf{v}^{j+1} - (\delta_{j-1}^2 - \delta_j^2)\mathbf{v}^j - \delta_j^2 \mathbf{v}^{j-1}}{\delta_{j-1}\delta_j(\delta_{j-1} + \delta_j)} \tag{31}$$

which gives the system

$$\mathbf{v}^{j+1} = \left(\left(1 - \left(\frac{\delta_j}{\delta_{j-1}}\right)^2\right)\mathbf{I} - \frac{\delta_j}{\delta_{j-1}}(\delta_{j-1} + \delta_j)\mathbf{L}^j\right)\mathbf{v}^j + \left(\frac{\delta_j}{\delta_{j-1}}\right)^2 \mathbf{v}^{j-1} \tag{32}$$

This scheme, sometimes known as *leap frogging*, seems attractive, as it avoids solving matrix systems. However turns out to be unstable for any grid size. Other multi-level scheme are (or can be made) stable. This shows the importance of carrying out stability analysis before implementing a scheme.

## 3.3 Boundary Conditions

While it is possible to solve the matrix system without applying explicit boundary conditions if the boundaries are sufficiently far from the interesting part of the solution,[15] it is usually better to impose explicit boundary conditions, and essential when the boundary conveys additional information, such as for barrier options.

### 3.3.1 Dirichlet Boundary Conditions

Here the solution has known values on the boundaries $x_0$ and $x_{N+1}$. Since equation 27 represents a set of $N+2$ equations, the $i^{th}$ of which gives the relation between the values and derivatives at $x = x_i, t = t_{j+1}$ and those at $x = x_i, t = t_j$, we can ignore the first and last of these equations. Furthermore, the first and last columns of $\mathbf{L}$ will be multiplied by known values, so this can be extracted out into a separate known vector, $\mathbf{r}$. We can then work with a slightly smaller system, where $\bar{\mathbf{v}}^j = (V_{1,j}, \ldots, V_{N,j})^T$ is a length $N$ vector and $(\bar{\mathbf{L}}^j)_{l,m} = (\mathbf{L}^j)_{l+1,m+1}$ with $l, m \in (0, N-1)$ is a $N$ by $N$ matrix (i.e. the $\mathbf{L}$ with the first and last rows and columns removed). This gives

$$\begin{aligned} (\mathbf{I} + \theta\delta_j \bar{\mathbf{L}}^{j+1})\bar{\mathbf{v}}^{j+1} &= (\mathbf{I} - (1-\theta)\delta_j \bar{\mathbf{L}}^j)\bar{\mathbf{v}}^j - \mathbf{r}^j \\ \text{where} \quad r_i^j &= (1-\theta)\delta_j (L_{i+1,0}^j V_{0,j} + L_{i+1,N+1}^j V_{N+1,j}) \\ &+ \theta\delta_j (L_{i+1,0}^{j+1} V_{0,j+1} + L_{i+1,N+1}^{j+1} V_{N+1,j+1}) \end{aligned} \tag{33}$$

The system is solved for each time step, and the boundary values can be appended to $\bar{\mathbf{v}}^j$ to form the full vector $\mathbf{v}^j$. In the case that $\mathbf{D}^1$ and $\mathbf{D}^2$ come from three-point differencing, the matrices $\bar{\mathbf{L}}^j$ are tridiagonal, and thus the system will be fast to solve.

[14] One can start with a two level scheme, then switch to a higher level scheme once enough previous steps are known.

[15] that is, the optionality value is virtually zero at the boundary.

### 3.3.2 Robin Boundary Conditions

The trouble with imposing boundary conditions that involve derivatives of the solution at the boundary is that we are forced to used one sided differences, and thus loss accuracy. Following [Duf06], one can introduce an additional *ghost* point beyond the barrier, which allowed central differences to be used for the derivative estimates, however one never explicitly computes the solution value at this ghost point.

If one assumed that the PDE holds on the boundary, then one can explicitly write out the first line of equation 28, for three-point differencing with $x_0 - x_{-1} = \Delta_0$. This gives

$$\begin{aligned}\theta\delta_j\left(\frac{a_0}{\Delta_0^2}-\frac{b_0}{2\Delta_0}\right)v_{-1}^{j+1}+\left(1+\theta\delta_j\left(-2\frac{a_0}{\Delta_0^2}+c_0)\right)\right)v_0^{j+1}+\theta\delta_j\left(\frac{a_0}{\Delta_0^2}+\frac{b_0}{2\Delta_0}\right)v_1^{j+1}=\\ -(1-\theta)\delta_j\left(\frac{a_0}{\Delta_0^2}-\frac{b_0}{2\Delta_0}\right)v_{-1}^{j}+\left(1-(1-\theta)\delta_j\left(-2\frac{a_0}{\Delta_0^2}+c_0)\right)\right)v_0^{j}-(1-\theta)\delta_j\left(\frac{a_0}{\Delta_0^2}+\frac{b_0}{2\Delta_0}\right)v_1^{j}\end{aligned} \tag{34}$$

where $a_0 \equiv a_0^{j+\theta}$ etc (i.e. we have suppressed the time index on the PDE coefficients). The Robin condition is

$$-\frac{\beta^j}{2\Delta_0}v_{-1}^j+\alpha^j v_0^j+\frac{\beta^j}{2\Delta_0}v_1^j=\gamma^j \rightarrow v_{-1}^j=\frac{2\Delta_0\alpha^j}{\beta^j}v_0^j+v_1^j-\frac{2\Delta_0\gamma^j}{\beta^j} \tag{35}$$

Eliminating $v_{-1}$ terms gives

$$\begin{aligned}&\left(1+\theta\delta_j\left[-2\frac{a_0}{\Delta_0^2}-\frac{\alpha^{j+1}}{\beta^{j+1}}\left(\frac{2a_0}{\Delta_0}-b_0\right)+c_0\right]\right)v_0^{j+1}+\theta\delta_j\left(2\frac{a_0}{\Delta_0^2}\right)v_1^{j+1}=\\ &\left(1-(1-\theta)\delta_j\left[-2\frac{a_0}{\Delta_0^2}-\frac{\alpha^{j}}{\beta^{j}}\left(\frac{2a_0}{\Delta_0}-b_0\right)+c_0\right]\right)v_0^{j}-(1-\theta)\delta_j\left(2\frac{a_0}{\Delta_0^2}\right)v_1^{j}\\ &+\delta_j\left((1-\theta)\frac{\gamma^j}{\beta^j}+\theta\frac{\gamma^{j+1}}{\beta^{j+1}}\right)\left(\frac{2a_0}{\Delta_0}-b_0\right)\end{aligned} \tag{36}$$

In the special case that $\beta = 0$ (i.e. Dirichlet), this can be written simply (and obviously) as

$$v_0^{j+1}=\frac{\gamma^{j+1}}{\alpha^{j+1}} \tag{37}$$

and for $\alpha = 0$ (i.e. Neumann)

$$\begin{aligned}&\left(1+\theta\delta_j\left[-2\frac{a_0}{\Delta_0^2}+c_0\right]\right)v_0^{j+1}+\theta\delta_j\left(2\frac{a_0}{\Delta_0^2}\right)v_1^{j+1}=\\ &\left(1-(1-\theta)\delta_j\left[-2\frac{a_0}{\Delta_0^2}+c_0\right]\right)v_0^{j}-(1-\theta)\delta_j\left(2\frac{a_0}{\Delta_0^2}\right)v_1^{j}+\delta_j\left((1-\theta)\frac{\gamma^j}{\beta^j}+\theta\frac{\gamma^{j+1}}{\beta^{j+1}}\right)\left(\frac{2a_0}{\Delta_0}-b_0\right)\end{aligned} \tag{38}$$

If we rewrite equation 28 as

$$\begin{aligned}&\mathbf{P}^j\mathbf{v}^{j+1}=\mathbf{q}^j\\ &\text{where}\quad \mathbf{P}^j=\mathbf{I}+\theta\delta_j\mathbf{L}^{j+\theta}\quad\text{and}\quad \mathbf{q}^j=\left(\mathbf{I}-(1-\theta)\delta_j\mathbf{L}^{j+\theta}\right)\mathbf{v}^j\end{aligned} \tag{39}$$

and then set the first and last rows of $\mathbf{P}$ to the values given by the LHS of equation 36 (and its equivalent for the upper boundary), and the first and last entries of $\mathbf{q}$ to the values given by the RHS of equation 36, then we have a system that imposes the boundary conditions while preserving the $\mathcal{O}(\Delta^2)$ accuracy of three-point differencing.

Whether to use 33 (i.e. the reduced system) or equation 37 for Dirichlet boundary conditions is a matter of taste. We prefer the latter, since you do conceptually the same thing to apply Dirichlet or Neumann (or indeed Robin) boundary conditions.

The same mechanism can be used to apply the linearity boundary condition, by substituting $v_{-1} = 2v_0 - v_1$ into equation 34. This gives the same result as setting $a_0 = 0$ (i.e. turning off diffusion) and using one-sided differencing for the convection term. This is because when the second derivative is zero, a one-sided difference estimate of the second derivative is $\mathcal{O}(\Delta^2)$ accurate. Hence, when the linearity condition is met (e.g. the upper boundary of the call is deep out-the-money), using one sided differencing to apply Neumann/Robin boundary conditions does not give a loss of accuracy.

# 4 Applications

In this section we show the pricing of three types of options: European; European with a single (constant) barrier feature, and American. In all three cases we assume a constant volatility and constant risk-free rate and cost-of-carry (i.e. Black-Scholes-Merton assumptions). In the first two cases exact analytic formulae exist, while for American options only analytic approximations exist.

We use three-point differencing, so we are always working with a tridiagonal system. Since the Thomas algorithm runs in $\mathcal{O}(n)$[PTVF07], the total run time should scale as the total number of nodes (grid size = #space nodes × #time nodes). The calculations are performed on a near uniform grid - near uniform in the sense that the spatial node positions have been adjusted so that spot lies on the grid and strike between two grid points . Non-uniform grids can give better accuracy for the same compute budget and are discussed in section 4.1.5. Grid generation is discussed in appendix B. Finally, we choose $\theta = 0.5$, i.e. a Crank-Nicolson scheme.

All the results were produced in MATLAB[16] version R2012b, running on Mac Pro with 2 × 2.66 GHz Quad-Core Intel Xeon CPUs, and 24 GB of RAM.

## 4.1 European Options

Our example European option is a call with a strike of 13.0 and an expiry of 2 years. The spot is 10.0, risk free rate is 20%[17], cost-of-carry[18] is 10% and the volatility is 30%, giving it a price of 1.171339 according to the Black-Scholes formula.

We set the boundaries six-sigma from the spot, i.e.

$$\text{Boundary level} = s_0 \exp(\pm 6\sigma\sqrt{t}) \tag{40}$$

where $t$ is the expiry[19] - this is extremely conservative, and as we are using a uniform grid, it wastes a lot of nodes in unimportant regions.

[16]The code is available here www.opengamma.com/downloads/financial-pde-solving-matlab-examples.zip

[17]we have chosen large rates to highlight any numerical problems

[18]The cost-of-carry (often denoted by the letter 'b') is the rate of growth of an asset. For a yield $q$ the cost-of-carry is $b = r - q$.

[19]Strictly, this should be Boundary level $= s_0 \exp(bt \pm 6\sigma\sqrt{t})$, but we ignore the drift when setting the boundaries.

The table 4.1 below shows appropriate boundary conditions, where the probability of reaching the boundary is small. In our example we use Dirichlet on the lower boundary and Neumann on the upper; although we find no difference for other choices.

| | Dirichlet | | Neumann | |
|---|---|---|---|---|
| | lower | upper | lower | upper |
| call | 0.0 | $x_{N+1}e^{-q\tau} - ke^{-r\tau}$ | 0.0 | $e^{-q\tau}$ |
| put | $ke^{-r\tau} - x_0e^{-q\tau}$ | 0.0 | $-e^{-q\tau}$ | 0.0 |

Table 1: Lower ($x_0$) and upper ($x_{N+1}$) boundary conditions for European options. $x_0$ and $x_{N+1}$ are the lowest and highest spacial (spot) values represented on the grid.

### 4.1.1 Strike Position

We always setup the spatial grid such that the spot lies on the grid - this prevents the need to interpolate between grid points to read off the option price. The best position of grid points around the strike needs investigating. We set up a grid with a node at 13.0 (corresponding to our nominal strike), then vary the strike between 12.875 and 13.125 - a range that covers three spatial nodes, while keeping the same grid. The relative error against strike is show in figure 1. What is clear from this graph is that the error is greatest when the strike lies on a spatial

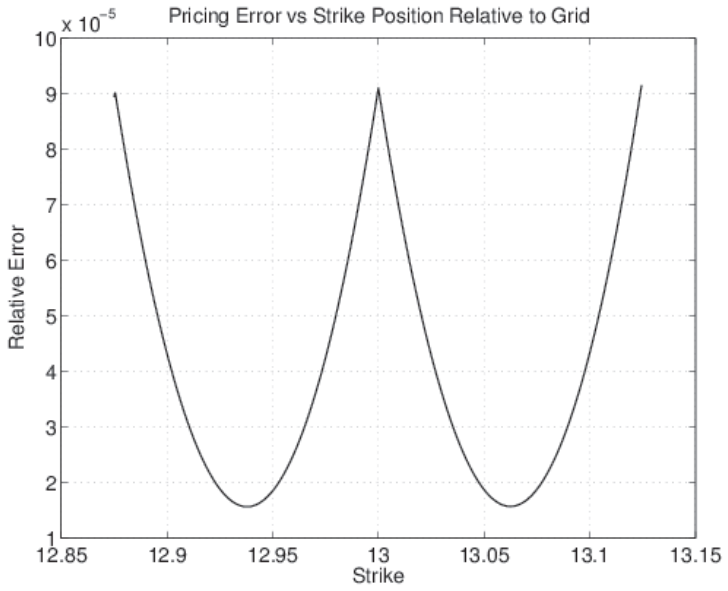


Figure 1: The relative error for an example European calls ($S_0 = 10$, $r = 0.2$, $b = q = 0.1$, $T = 2.0$ and $\sigma = 0.3$) where the strike varies between 12.85 and 13.125, on a fixed (uniform) grid of 1000 spatial and 100 temporal nodes. The peaks in the error correspond to the strike coinciding with a node.

node, and is minimised when it lies half-way between two nodes. The difference in accuracy is around an order of magnitude for no additional computational work, so it always makes sense to place spatial node points so that the strike falls between two of them. A way of achieving this is discussed in Appendix B.3, and we use this setup of all further examples.

### 4.1.2 Issues with the Crank-Nicolson Scheme

Problems with the Crank-Nicolson scheme are well known [Duf04]. Essentially, the eigenvalues associated with the discontinuity in the derivative of the pay-off are not damped down when $\theta = 0.5$ (they do not grow either), which pollutes the solution for strikes near the spot. This shows up most notably in plots of implied volatility, Delta and Gamma[20] against spot for a fixed strike. Figure 2 shows the delta against spot at six different times to expiry, using Crank-Nicolson. The initial oscillation is not fully damped out even after two years. Running with $\theta = 1.0$ (fully implicit) does damp out these eigenvalues, and there are no oscillations seen in the plots of either Delta or Gamma. The problem is, one only has $\mathcal{O}(\Delta t)$ rather than $\mathcal{O}(\Delta t^2)$ accuracy. As mentions in section 3.2.3, Richardson extrapolation can be used to make fully implicit stepping $\mathcal{O}(\Delta t^2)$ accurate. A small number of these steps are enough to damp down the unwanted eigenvalues, after this we can switch to Crank-Nicolson stepping. Figure 3 shows the implied volatility error[21] against spot for different numbers of initial Richardson steps (the final panel shows all Richardson steps). After just two of these steps the oscillations around the strike have disappeared. This short "burn-in" of a couple of Richardson steps is not too detrimental to performance,[22] and rectifies a serious problem with the Crank-Nicolson method.

The rest of the results in this paper use a "burn-in" of two Richardson steps.

### 4.1.3 Ratio of Nodes, $\nu$

We define the ratio of spatial to temporal nodes as $\nu$. For a fixed compute budget, which corresponds to a fixed total number of nodes, the value of $\nu$ affects the accuracy of the solution. The correct choice of $\nu$ can improve accuracy by two orders of magnitude, with no additional computation time. Figure 4 shows the relative error against $\nu$ for a grid sizes of $10^4$ , $10^5$ and $10^6$. It is clear that (at least for this option and grid topology) $\nu \approx 30$ is optimal regardless of the grid size.

For a uniform grid, the error in the Crank-Nicholson method is $\mathcal{O}(\Delta^2 + \phi^2\delta^2)$, where $\phi^2$ is some constant of proportionality between the error contribution from the temporal and spatial discretisation. Given that $\Delta \approx \frac{x_{max}}{N+1}$, $\delta = \frac{T}{M+1}$ and $Q = (N+2)(M+2)$ where $Q$ is the total number of nodes (grid size), this can be rewritten as

$$\text{error} \sim \mathcal{O}\left(\frac{1}{Q}\left(\frac{x_{max}^2}{\nu} + \phi^2 T^2 \nu\right)\right) \tag{41}$$

This shows that the error scales as $\mathcal{O}(1/Q)$ - something that we will show experimentally in the next section. For fixed $Q$, $x_{max}$[23] and $T$, the optimal value of $\nu$ is

$$\nu = \frac{x_{max}}{\phi T}$$

[20] Delta and Gamma are of course calculated by finite-difference at every time step of the PDE solution anyway.

[21] Implied volatility from the option price produced by the PDE minus the 0.3 input volatility. This shows problems that are not visible on a plot of price against spot.

[22] It amounts to an extra 4 matrix system solves for 2 Richardson steps.

[23] The error term implies that $x_{max}$ should be made as small as possible. However if it is too small, the boundary conditions given in table 4.1 will be incorrect and the accuracy of the solution will be polluted.

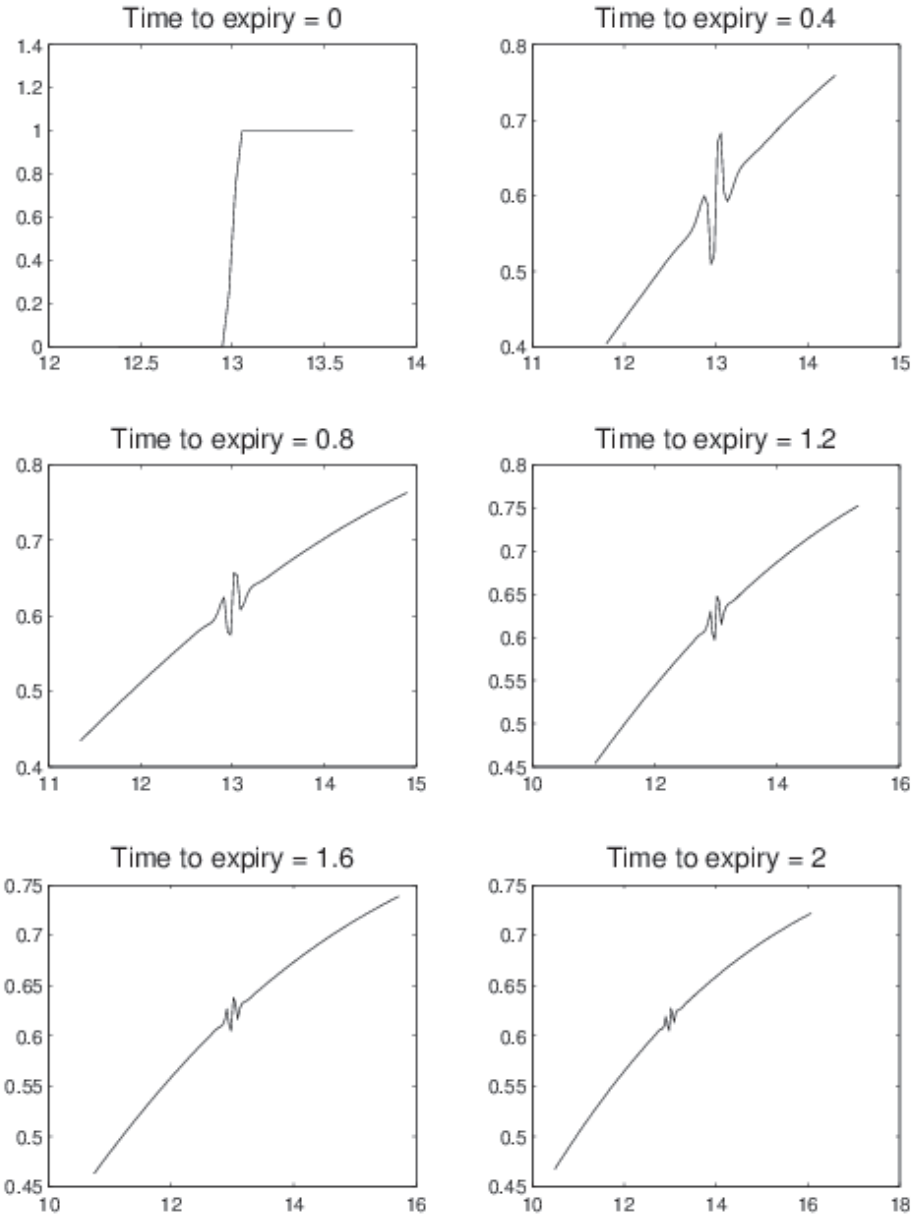


Figure 2: The delta for a European call ($k = 13.0$, $r = 0.2$, $b = q = 0.1$, $T = 2.0$ and $\sigma = 0.3$) against spot at six different times to expiry, using Crank-Nicolson. Note, the range of the x-axis (spot) widens with increased time-to-expiry.

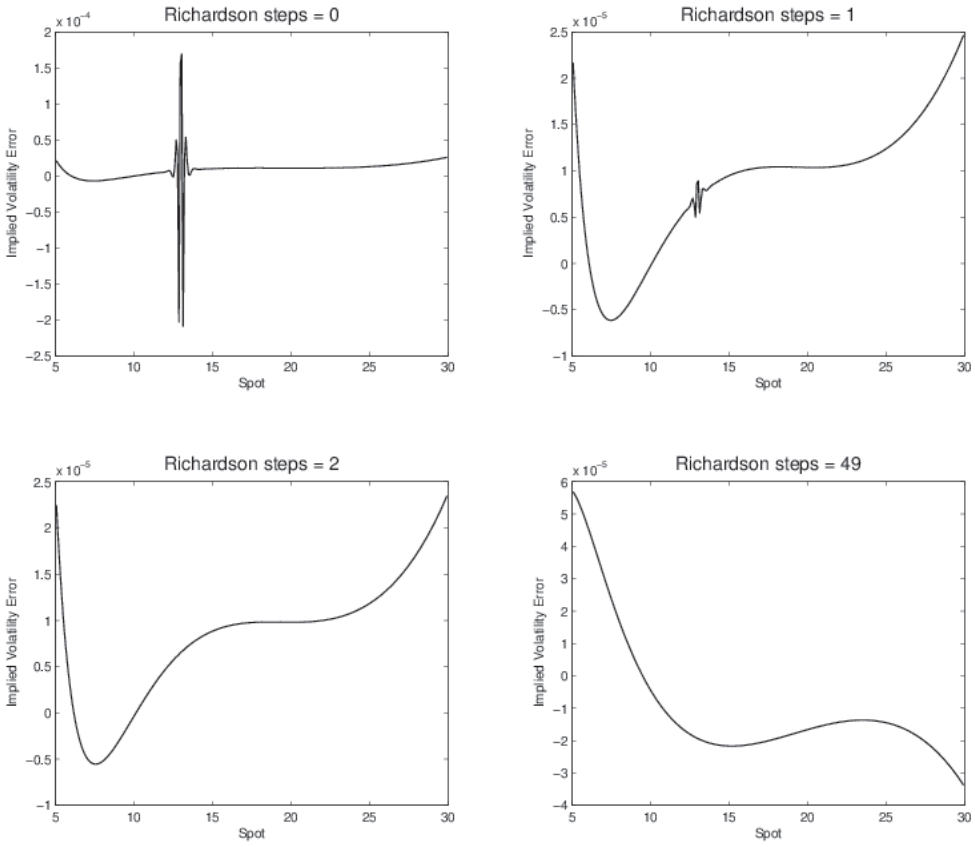


Figure 3: The implied volatility error for a European call ($k = 13.0$, $r = 0.2$, $b = q = 0.1$, $T = 2.0$ and $\sigma = 0.3$) against spot for different numbers of Richardson extrapolations of fully implicit stepping, before switching to Crank-Nicolson. The grid uses 50 time and 1500 space nodes.

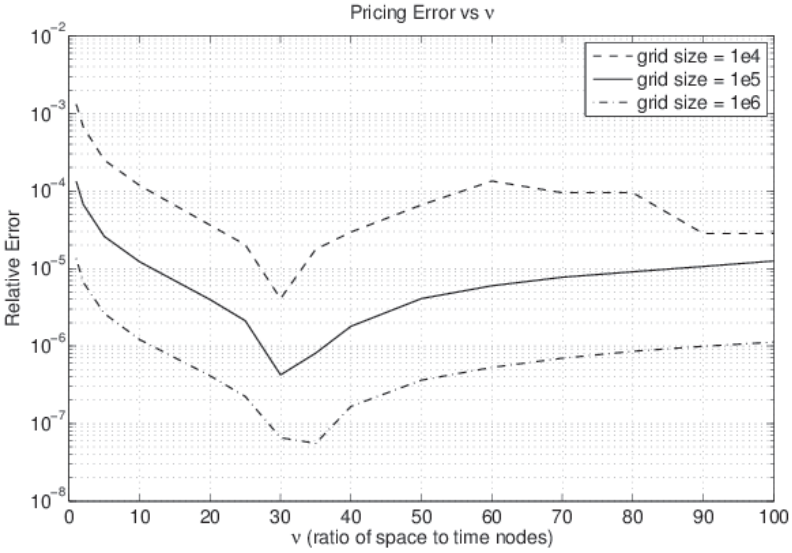


Figure 4: The relative error for an example European call ($S_0 = 10$, $k = 13$, $r = 0.2$, $b = q = 0.1$, $T = 2.0$ and $\sigma = 0.3$) against $\nu$ (the ratio of spatial to temporal nodes), for three values of the grid size (total number of nodes). In all three cases the optimal choice is $\nu \approx 30$.

However, without an explicit expression for $\phi$, this result is somewhat limited.

#### 4.1.4 Error Convergence

Using the optimal value of $\nu = 30$, we investigate how the relative error scales with the total number of nodes (grid size). Figure 5 below shows the the results. A minimum error of around $3 \times 10^{-11}$ is obtained for grid sizes of $2 - 3 \times 10^8$, after which the accumulation of round-off errors makes the error worse.[24] The relative error, $\epsilon$, scales as $\epsilon \sim \mathcal{O}(1/Q)$, where $Q$ is the grid size; to increase the accuracy by a factor of two, you simply need to run the solver for twice as long.

This type of error scaling is seen for all strikes, although, as you would expect, the accuracy worsens the further out the money we go. Figure 6 shows the absolute relative error for OTM options with strikes from 0.45 to 30.0 (note: a separate PDE is solved for each strike). Moving from low to high strikes, the relative error starts high (and positive, i.e. the PDE price is too high) and decreases as the strike approaches the forward. It then switches sign (three times) around the forward, then remains negative up until a strike of around 27, where it becomes positive again. This gives three points of spuriously high accuracy where the error switches sign; one either side of the forward and one at a high strike.

When an analytic solution is known for an option price, any numerical scheme can be tuned to provide very high accuracy for that price. However this does not mean that the same accuracy will be obtained for other options using the same scheme. What figure 6 shows is the (accidental) turning of the scheme for options with three particular strikes.

[24] This is a standard feature of finite difference methods.

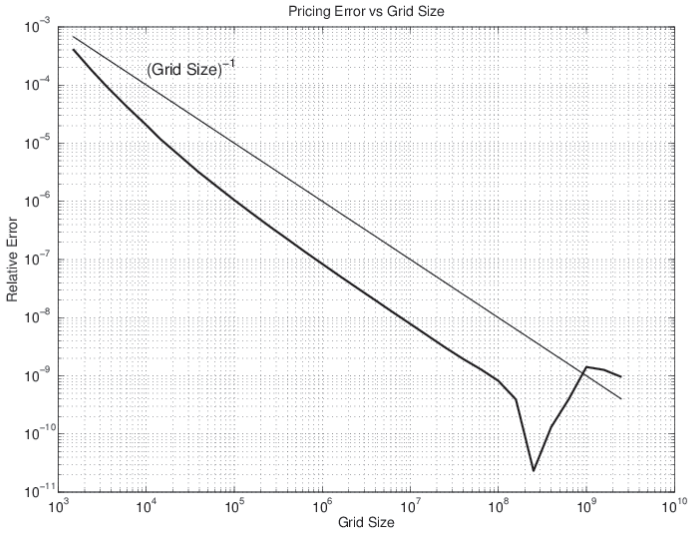


Figure 5: The relative error for an example European call ($S_0 = 10$, $k = 13$, $r = 0.2$, $b = q = 0.1$, $T = 2.0$ and $\sigma = 0.3$) against grid size. The ratio of spatial to temporal nodes ($\nu$) is 30. The thinner line shows relative error = (Grid Size)$^{-1}$. The total run time to generate this graph was around 45 minutes, with of course the bulk of the work for the very large grid sizes.

#### 4.1.5 Non-linear Grids

In our examples we use a (near) uniform spatial grid with points from 0 to 127 to price an option with a strike of 13.0 when the spot (forward) is 10.0 (12.2). Since we have the same density of points between 10.0 and 13.0, as between 100.0 and 103.0, this is an incredibly inefficient allocation. A hyperbolic mesh (see appendix B), concentrates points in a certain region, with the strength of the concentration controlled by a parameter $\beta$ ($\beta > 10.0$ is (near) uniform, while small values ($> 0$) give a large concentration).

Figure 7 shows the relative error for our example European call as a function of $\beta$ for three different grid sizes. The points are concentrated around the spot (10.0) and $\nu = 20$. For larger grids, we can achieve over two orders of magnitude improvement in accuracy (compared to a uniform grid, $\beta \gg 1.0$), if we choose the correct value of $\beta$. In this case the optimal value is $\beta = 0.007$, but we have no way, *a priori*, of knowing this. Furthermore, the improvement occurs rather suddenly, and the optimal value of $\beta$ depends on the option. Although it makes no difference in this example, we find it is better to concentrate nodes around the strike rather than the spot. Doing this we find that values of $\beta$ between $5 \times 10^{-4}$ and $10^{-1}$ will usually improve the accuracy by a factor of at least 5.

The conclusion is that for a fixed budget of node points (equivalent to a fixed computation time), the exact configuration of the grid has a massive effect on the accuracy, but beyond broad ranges for the parameters that control the configuration ($\nu$ and $\beta$), we know of no 'rules of thumb' to set the optimal parameter values for a given option to be priced. That said, a grid configuration that prices a European option accurately (and for which we have an analytic

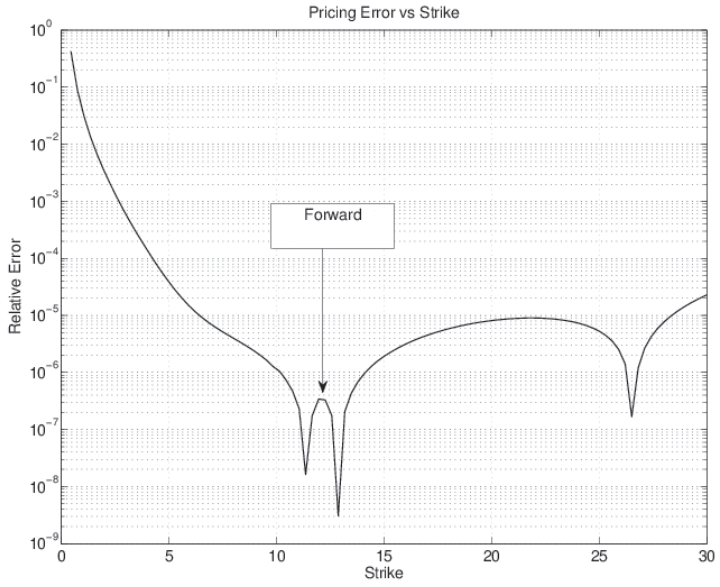


Figure 6: The relative error for Out-The-Money (OTM) European options ($S_0 = 10$, $r = 0.2$, $b = q = 0.1$, $T = 2.0$ and $\sigma = 0.3$) against strike in the rage 0.45 to 30.0.

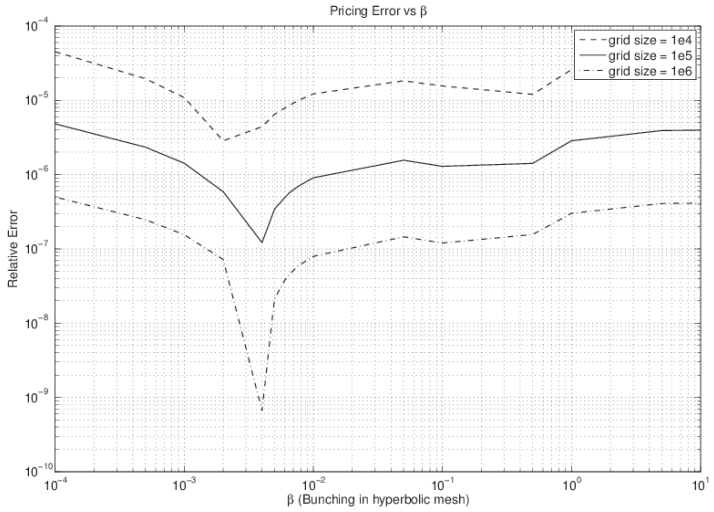


Figure 7: The relative error for a European call ($S_0 = 10$, $k = 13.0$, $r = 0.2$, $b = q = 0.1$, $T = 2.0$ and $\sigma = 0.3$) against $\beta$ - the concentration parameter for a hyperbolic mesh concentrated around the spot.

formula), should also price the equivalent American option accurately.

#### 4.1.6 Log-Spot Transformation

In this final section on pricing European options under Black-Scholes-Merton assumptions, we consider the transformation $x = \log\left(\frac{S_T}{S_0}\right)$ applied to equation 13. This eliminates the dependence of the diffusion and convection coefficients on the level of the underlying asset $S$ (assuming the volatility does not depend on $S$, i.e. local volatility). A uniform grid is used (with the usual adjustments for $S_0$ and strike) to price the option, with obvious changes to the initial and boundary conditions. Figure 8 shows the relative error as a function of grid size ($\nu = 20$ is optimal). The accuracy is actually a little bit worse that working directly with spot as the spatial variable on a uniform grid.

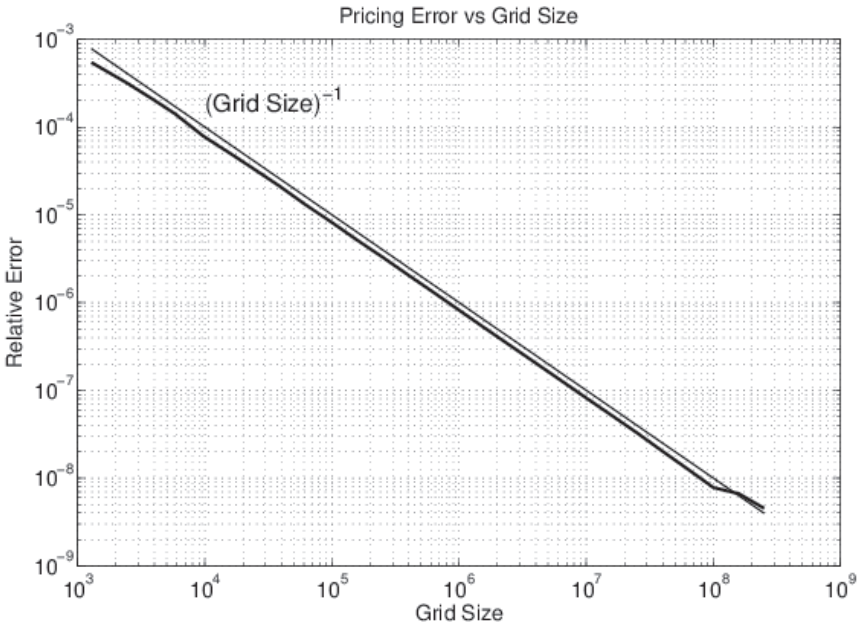


Figure 8: The relative error for an example European call ($S_0 = 10$, $k = 13$, $r = 0.2$, $b = q = 0.1$, $T = 2.0$ and $\sigma = 0.3$) against grid size, by solving the log-spot transformed PDE. The ratio of spatial to temporal nodes ($\nu$) is 20. The thinner line shows relative error = (Grid Size)$^{-1}$.

## 4.2 Barrier Options

Barrier options essentially come in two flavours, those for which the barrier is (effectively) continuously monitored, and those for which the barrier is only observed at a fixed set of times. The first is easy to handle in a PDE setting, since a boundary can be placed at the barrier. In the second case the solution can diffuse beyond the barrier between observation times, with large discontinuities at the barrier observations.[25] We discuss the first case below; the second case is discussed here [TR00].

In the Black-Scholes world, options with call or put payoffs at $T$, conditional on whether a single, continuously monitored barrier was hit or not at any time $t \leq T$, have exact analytical formulae. The two types of barrier are: "in" - the option becomes alive only if the barrier is hit; and "out" - the option expires worthless (or a rebate is paid) if the barrier is hit. If the barrier is above the current spot level it is an "up", otherwise it is a "down". Finally, the payoff can be a call or a put. This gives 8 ($= 2^3$) cases, with names such as "up-and-in put", the formulae for which are listed in [Hau07]. For "out" style barriers, if the barrier is not hit before expiry, the payoff is that of a European put or call; if the barrier is hit before expiry a rebate (which can be zero) is paid immediately and the option is cancelled. For "in" style barriers, the option only becomes alive if the barrier is hit. Once the barrier is hit you have a simply a European call or put; If the barrier is not hit by expiry, a rebate can be paid then.

"Out" type barriers are easily handled - one simply imposes a Dirichlet type boundary with value equal to the rebate at the barrier level.[26] "In" type barriers are a little more complex: firstly consider a modification of the "out" barrier that pays the rebate at expiry rather than when the barrier is hit - call this "out-special". Holding an "in" plus a "out-special" with the same parameters (and of course on the same underlying), is equivalent to holding a standard European option plus a zero coupon bond with expiry equal to that of the option, and notional amount equal to the rebate. The price of the "in" is then given by

$$V_{\text{in}} = V_{\text{European}} + \text{rebate} \times P(0,T) - V_{\text{out-special}}$$

The "out-special" is priced by modifying the Dirichlet boundary condition to be equal to the discounted rebate. Figure 9 shows the relative error for an "up-and-out call" option with a barrier level of 17.0; all the other parameters are as discussed in the European option example. We have used an exponential mesh for the spatial gird which give a high density of points near the barrier (i.e. $\lambda = -2.0$).

As with European call/puts, the density of (spatial) grid points around important levels (spot, strike and the barrier) will greatly affect the accuracy. In the European option example, since the (uniform) grid extended to very large spot level, the optimal $\nu$ was very high to ensure sufficient density.[27] In this case the grid is naturally constrained by the barrier, so a high density will be obtained with a much lower $\nu$ (recall we are using a nonuniform grid). The critical point is that, regardless of the grid configuration, the error seems to scale as $\mathcal{O}(1/Q)$,[28] where $Q$ is the grid size, i.e. the same scaling as for European options.

[25] This is a good example of when you must have the observation dates lie on the grid as described in appendix B.

[26] This will be either the lower or upper boundary depending on whether the barrier is below or above the current spot value.

[27] The density is of course more or less constant across spot values.

[28] Consistency between the initial and boundary conditions is vital here. When we originally set this problem up, we did not incorporate the barrier/rebate into the initial condition, which resulted in the error scaling as $\mathcal{O}(1/Q^{\frac{1}{2}})$ rather than $\mathcal{O}(1/Q)$.

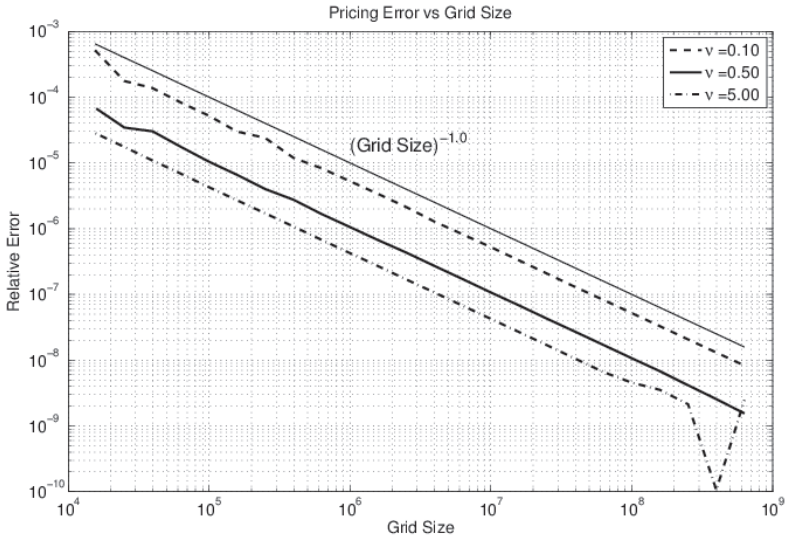


Figure 9: The relative error for an example European barrier up-and-out call ($S_0 = 10$, $k = 13$, $h = 17.0$, $r = 0.2$, $b = q = 0.1$, $T = 2.0$ and $\sigma = 0.3$) against grid size. The ratio of spatial to temporal nodes is given by $\nu$. The thinner line shows relative error = (Grid Size)$^{-1}$.

### 4.3 American Options

American options can be exercised at any time before expiry. Formally, the value of an American call option is

$$V(0,k) = \sup\left(0 \le u \le T : \mathbb{E}(e^{-ru}(S_u - k)^+\right)$$

The optimal exercise time $u$ is the value that maximises the expected payoff - any scheme to price an American must calculate this.

For American options with payoff $X(S)$, the equivalent of equation 13 is

$$\begin{aligned} \frac{\partial \bar{V}}{\partial \tau} - \frac{1}{2}\bar{\sigma}(S,\tau)^2 S^2 \frac{\partial^2 \bar{V}}{\partial S^2} - (\bar{r}_\tau - \bar{q}_\tau) S \frac{\partial \bar{V}}{\partial S} + \bar{r}_\tau \bar{V} \ge 0 \\ \bar{V}(S,\tau) \ge X(S) \\ \left[\frac{\partial \bar{V}}{\partial \tau} - \frac{1}{2}\bar{\sigma}(S,\tau)^2 S^2 \frac{\partial^2 \bar{V}}{\partial S^2} - (\bar{r}_\tau - \bar{q}_\tau) S \frac{\partial \bar{V}}{\partial S} + \bar{r}_\tau \bar{V}\right]\left[\bar{V} - X(S)\right] = 0 \end{aligned} \tag{42}$$

For our discretised scheme this means that the system $\mathbf{P}^j \mathbf{v}^{j+1} = \mathbf{r}^j$ becomes the Linear Complementarity Problem (LCP)

$$\begin{aligned} \mathbf{P}^j \mathbf{v}^{j+1} &\ge \mathbf{r}^j \\ \mathbf{v}^{j+1} &\ge \mathbf{x} \\ (\mathbf{v}^{j+1} - \mathbf{x})^T (\mathbf{P}^j \mathbf{v}^{j+1} - \mathbf{r}^j) &= 0 \end{aligned} \tag{43}$$

where $\mathbf{x}$ is the payoff. This again must be solved at every time step. A crude approximation is to solve the system $\mathbf{P}^j\mathbf{z} = \mathbf{r}^j$, then set $\mathbf{v}^{j+1} = \max(\mathbf{z}, \mathbf{x})$. This of course satisfies the second LCP condition, but not necessarily the other two.

This problem can be expressed as a quadratic programming (QP) problem and solved using algorithms for QP. The problem becomes, minimise $f(\mathbf{v}^{j+1}) = (\mathbf{v}^{j+1} - \mathbf{x})^T(\mathbf{P}^j\mathbf{v}^{j+1} - \mathbf{r}^j)$ subject to the first two LCP constraints. The preferred method in the finance literature seems to be modified Gauss-Seidel and Successive Over-Relaxation (SOR) methods [TR00, Wil06, PTVF07, Fai11].

### 4.3.1 A Brief Review of Fixed Point methods

The system[29] $\mathbf{Pv} = \mathbf{r}$ can be solved using a fixed point iteration

$$\mathbf{v}^{(k)} = \mathbf{G}\mathbf{v}^{(k-1)} + \mathbf{c}$$

where $\mathbf{v}^{(k)}$ is the approximation to $\mathbf{v}$ on the $k^{th}$ iteration, and $\mathbf{G}$ & $\mathbf{c}$ are a fixed matrix and vector respectively. A suitable stopping criterion is

$$\frac{\|\mathbf{v}^{(k)} - \mathbf{v}^{(k-1)}\|}{\|\mathbf{v}^{(k)}\|} < \epsilon$$

where $\epsilon$ is some tolerance and $\|\cdot\|$ denotes a vector norm - we use the Euclidean norm, but any vector norm will work in this situation. We write the matrix $\mathbf{P}$ as $\mathbf{P} = \mathbf{L} + \mathbf{D} + \mathbf{U}$ where $\mathbf{D}$ is the diagonal, and $\mathbf{L}$ and $\mathbf{U}$ and the lower and upper triangles respectively. The Jacobi fixed point iteration is

$$\mathbf{v}^{(k)} = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{v}^{(k-1)} + \mathbf{D}^{-1}\mathbf{r} \tag{44}$$

where $\mathbf{G} = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})$ is the Jacobi matrix.

Gauss-Siedel is similarly written as

$$\mathbf{v}^{(k)} = (\mathbf{D} + \mathbf{L})^{-1}\mathbf{U}\mathbf{v}^{(k-1)} + (\mathbf{D} + \mathbf{L})^{-1}\mathbf{r} \tag{45}$$

If $v_i^{(k)}$ denotes the $i^{th}$ element of $\mathbf{v}^{(k)}$, this can be rewritten element-by-element as

$$v_i^{(k)} = \frac{1}{P_{ii}}\left[r_i - \sum_{l=1}^{i-1} P_{il}v_l^{(k)} - \sum_{l=i+1}^{N} P_{il}v_l^{(k-1)}\right] \tag{46}$$

The presence of terms $v_l^{(k)}$ on the right-hand side is not a problem as these elements have been updated by the time they are used. This immediate use of the updated elements is what differentiates this from the Jacobi method.

The SOR algorithm can be written element-by-element as

$$v_i^{(k)} = (1-\omega)v_i^{(k-1)} + \frac{\omega}{P_{ii}}\left[r_i - \sum_{l=1}^{i-1} P_{il}v_l^{(k)} - \sum_{l=i+1}^{N} P_{il}v_l^{(k-1)}\right] \tag{47}$$

where $\omega$ is the relaxation parameter. If $\mathbf{P}$ is Symmetric Positive Definite (SPD), and *consistently ordered* (tridiagonal matrices always are)[Urb07], the system will always converge for $0 < \omega < 2$ [Fai11].[30]

[29] we drop the time-index superscript to avoid clutter.

[30] $\omega < 1$ is under-relaxation, $\omega > 1$ is over-relaxation and $\omega = 1$ is just the Gauss-Seidel method.

If the above criteria are met, then the optimal[31] value for $\omega$ is given by[Fai11]

$$\omega = \frac{2}{1 + \sqrt{1 - \rho(\mathbf{J})^2}} \tag{48}$$

where $\rho(\mathbf{J})$ is the spectral radius of the Jacobi matrix. However, our system is neither symmetric or positive definite, which means this value of $\omega$ is not necessarily optimal, and SOR will converge slowly, if at all.

### 4.3.2 Solving the LCP

For the LCP the simple modification $v_i^{(k)} = \max(v_i^{(k)}, x_i)$ is applied at each step[32] of equation 47. This is known as *Projected* SOR (PSOR). The crude approximation mentioned above should be used as the starting value. For a tridiagonal matrix, there are seven operations[33] per vector element. As the number of iterations (for a given tolerance) is largely independent of the size of the system, $N$, we expect PSOR to also be $\mathcal{O}(N)$: we find that it is, but it is 600 to 1000 times more expensive than solving the tridiagonal system $\mathbf{Pv} = \mathbf{r}$, even when $\omega$ has been (experimentally) optimised. This makes PSOR a severe bottleneck in pricing an American option, since the computation time for using the same sized grid as a European option is 600 to 1000 times more.

LCP is a vast research area in applied mathematics in its own right[CPS09], and there are many alternative algorithms available to solve it. A Newton-based method[Fis95] is available on MATLAB Central,[34] which we use with a slightly modified API.

This Newton-based method is about 20 times faster than PSOR (which of course still makes the American pricing 30 to 50 times slower than the equivalent European), which makes accurate pricing in a reasonable time frame achievable. We use this algorithm for the results shown below. No doubt a domain-specific LCP algorithm would be faster still.

### 4.3.3 Boundary Conditions for American Options

The boundary conditions are also different from those for European options. Table 4.3.3 below shows appropriate boundary conditions for American options, where the probability of reaching the boundary is small.

| | Dirichlet | | Neumann | |
|---|---|---|---|---|
| | lower | upper | lower | upper |
| call | 0.0 | $x_{N+1} - k$ | 0.0 | 1.0 |
| put | $k - x_0$ | 0.0 | $-1.0$ | 0.0 |

Table 2: Boundary conditions for American options. $x_0$ and $x_{N+1}$ are the lowest and highest spatial (spot) values represented on the grid.

[31] meaning fastest convergence.
[32] It is critical that this is applied as each element is updated.
[33] these operations are $+$, $-$ and $*$. The time actually taken for each is architecture specific.
[34] www.mathworks.com/matlabcentral/fileexchange/20952

### 4.3.4 Results

For American call options with $b \geq r$ (equivalently yield $q \leq 0$) it is not optimal to exercise early, thus the price is just that of a European call option. We use this as our first benchmark (just a sanity check really), using the same parameters as for the European option except that $b = r\,(q = 0)$. Figure 10 shows the relative error against grid size, which shows the same behaviour as European option pricing (as expected).

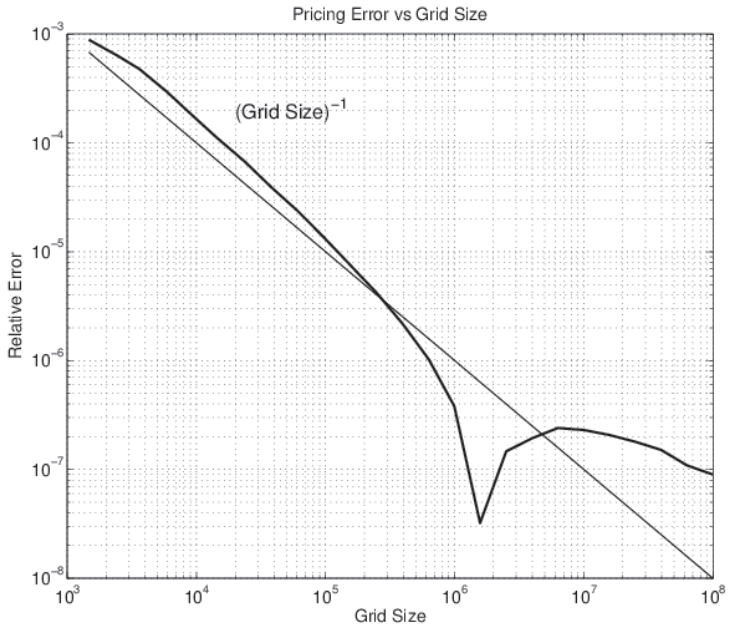


Figure 10: The relative error for an example American call ($S_0 = 10$, $k = 13$, $r = 0.2$, $b = 0.2\,(q = 0.0)$, $T = 2.0$ and $\sigma = 0.3$) against grid size. The ratio of spatial to temporal nodes is $\nu = 30$. The thinner line shows relative error = (Grid Size)$^{-1}$. It is never optimal to exercise this option early, so its price should be equal to a European call.

Since the is no exact analytic American option price, we must benchmark against numerical values for the non-trivial cases.[35] We price an American put option with a strike of 7.0 (all other parameters as for the European option example), with $3 \times 10^3$ time nodes and $9 \times 10^4$ space nodes ($2.7 \times 10^8$ grid size). This price[36] is our benchmark number. We then run as usual with various grid sizes and $\nu = 30$. The result is shown in figure 11, along with the result for just using the crude approximation to the LCP (i.e. just applying the max function, rather than actually solving the LCP). The notable point here is that the error scales as $\mathcal{O}(Q^{-\frac{1}{2}})$, whether or not we solve the LCP (the error is much better when the LCP is solved). We find this a somewhat surprising result, which merits further scrutiny. What this implies is that, depending

[35] Approximations such as Bjerksund & Stensland (2002) [BS02] are not accurate enough to test the accuracy of the finite difference scheme.

[36] The PDE result is 0.14459568 while Bjerksund-Stensland (2002) gives 0.14275

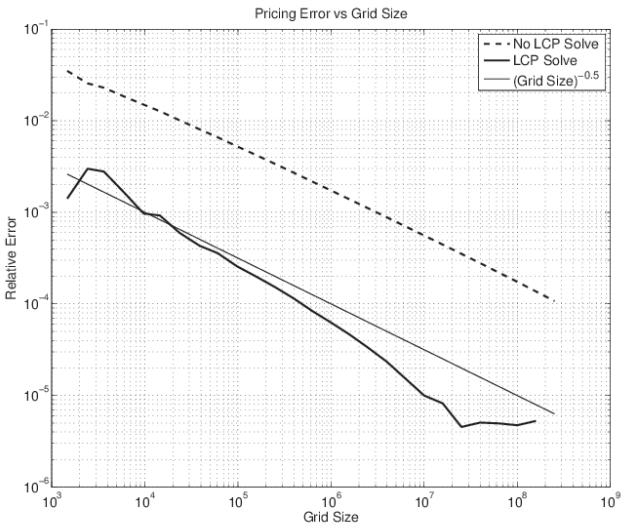


Figure 11: The relative error for an example American put ($S_0 = 10$, $k = 7$, $r = 0.2$, $b = q = 0.1$, $T = 2.0$ and $\sigma = 0.3$) against grid size. The ratio of spatial to temporal nodes is $\nu = 30$. The thinner line shows relative error = (Grid Size)$^{-0.5}$. The error is with respect to the price calculated on a very large grid (see main text).

on the speed of the LCP solver on a particular architecture, it may be more efficient not to solve the LCP - this is not the case for the LCP solver we used, on our architecture.

### 4.4 Other Applications

All three examples have used a flat volatility surface (and constant rates), with a standard option payoff. Other applications that can be handled by the set up described here include:

- Local volatility; all three instruments could equally be priced with a local volatility with no modifications to the code.
- Double barriers; The upper and lower boundaries level at set equal to the barriers.
- Other payoff can priced by simply changing the initial (and boundary) conditions.
- Options on dividend paying stock can be solved by modelling the price between dividend payments on a separate grid, then pasting the solution from one grid as the initial condition for the next (see [Whi12a] for a discussion).

## 5 About the Code

The OpenGamma Platform is written in Java. However MATLAB is used for rapid development of many of the analytics models. Since the graphical output of MATLAB is better than our

alternative (Excel), we generate the figures in MATLAB, and present the code for consistency.[37] In some examples in this paper, we have run very large grids, such that the rounding error begins to dominate. These examples can take over an hour to run.[38] In the provided code the maximum grid size is smaller, so that the examples all run in under a few minutes. The interested reader can increase this.

The MATLAB code provided here should not be considered "industrial strength". It will produce the figures shown in this paper, however if the example files are changed, there is no guarantee that the code will run correctly, or produce clear error messages. The code should also run under Octave, although this has not been explicitly tested.[39]

The code can be downloaded from here www.opengamma.com/downloads/financial-pde-solving-matlab-examples.zip.

# A Matrix Notation

We try to stick to the convention of representing scalars as non-bold characters (of either case), vectors as bold lower-case characters and matrices as bold upper-case characters, so $a$ and $A$ are both scalars, $\mathbf{a}$ is a vector and $\mathbf{A}$ is a matrix. When indexing a vector or matrix we use subscripts on the same letter, so $a_i$ is the $i^{th}$ element of the vector $\mathbf{a}$, and $A_{i,j}$ is the entry in the $i^{th}$ row and $j^{th}$ column of the matrix $\mathbf{A}$. Other sundry conventions are:

- Indexing starts from 0 (as with C lineage languages) rather than 1.
- The transpose is written as $\mathbf{a}^T$ or $\mathbf{A}^T$, and vectors are considered row vectors unless stated otherwise, i.e. $\mathbf{a}^T$ is considered a column vector.
- Where vectors or matrices are functions of some variable (usually time), they are written as $\mathbf{a}(t)$ or $\mathbf{A}(t)$
- Where vectors or matrices appear in a series (usually the index $j$ of discretised time), we write the $j^{th}$ member of the series as $\mathbf{a}^j$ or $\mathbf{A}^j$. So $\mathbf{A}(t_j) = \mathbf{A}^j$ etc. To distinguish this from taking the power of a matrix, we write the latter as $(\mathbf{A})^j$. When we must show a single entry, this is written as $a_i^j$ for the $i^{th}$ element of the vector $\mathbf{a}^j$, and $A_{i,k}^j$ for the $i,k$ element of the matrix $\mathbf{A}^j$.
- The notation $A_{i,:}$ signifies the $i^{th}$ row of $\mathbf{A}$, and $A_{:,j}$ the $j^{th}$ column. So $\mathbf{a}^i = A_{i,:}$ and $\mathbf{a}^i = (A_{:,i})^T$ are valid statements.
- $\mathbf{A} = \text{diag}(\mathbf{a})$ means that $\mathbf{A}$ is a diagonal matrix, with the elements of $\mathbf{a}$ along the leading diagonal.
- $.*$ and $./$ signifies element-wise multiplication and division, so $\mathbf{a}./\mathbf{b} = \mathbf{c}$ means $c_i = a_i/b_i \,\forall\, i$.
- If the identity matrix, $\mathbf{I}$, appears, it is assumed to be the correct size for that equation.

[37] similar figures could be generated from the Java, but we do not have explicit examples to do so.

[38] Using MATLAB version R2012b, running on Mac Pro with 2 × 2.66 GHz Quad-Core Intel Xeon CPUs, and 24 GB of RAM.

[39] Any errors will most likely be in the configuration of figures.

# B Grids Generation

Initially we have the trivial mapping for our integer indices to real numbers between 0 and 1. This is given by

$$z_i = \frac{i}{(N+1)}$$

where $i$ runs from 0 to $N+1$. This is then mapped to the physical space,[40] $x$, by $x = f(z)$, where $f(\cdot)$ is a smooth[41] monotonic function. In the simple case of a uniform grid between $x_{min}$ and $x_{max}$ this is given by

$$x_i = x_{min} + (x_{max} - x_{min})z_i$$

## B.1 Non-uniform Grids

We use two schemes to generate non-uniform grids. The first we call exponential meshing, with points between $x_{min}$ and $x_{max}$ exclusive, given by

$$x_i = (x_{min} - \eta) + \eta \exp(\lambda z_i)$$
$$\text{where} \quad \eta = \frac{x_{max} - x_{min}}{e^{\lambda} - 1}$$

The parameter $\lambda$ controls the distribution of points: $\lambda > 0$ gives a higher density of points near $x_{min}$, with the effect greater for larger $\lambda$; $\lambda < 0$ gives a higher density of points near $x_{max}$ with the effect greater for smaller (more negative) $\lambda$. $\lambda = 0$ gives a uniform distribution of points.[42] If one sets $\lambda = \log(x_{max}/x_{min})$, it is equivalent to setting $x_i = e^{z_i}$, where the $z_i$ are uniformly distributed - in this way, you can have the same distribution of spot working with a non-uniform grid, as working with log-spot on a uniform grid.

The second scheme we call hyperbolic meshing, with points given by

$$x_i = x^c + \beta \sinh(\gamma z_i + \delta)$$
$$\text{where} \quad \beta = \alpha(x_{max} - x_{min}) \quad \delta = \sinh^{-1}\left(\frac{x_{min} - x^c}{\beta}\right) \text{and} \, \gamma = \sinh^{-1}\left(\frac{x_{max} - x^c}{\beta}\right) - \delta$$

This produces a higher density of points around $x^c$ (although $x^c$ itself is not necessarily a grid point - see below), with the effect decreasing as $\alpha$ increases.

Both these functions are easily invertible, which is an important property.

## B.2 Specifying Exact Positions of certain Nodes

It is often necessary to specify the exact position of one or more internal points. This can be achieved by moving the nearest point to a new position. However, this will produce a sudden change in the grid spacing, which could be detrimental to the accuracy of the solution. A better approach ([TR00]) is to introduce an intermediate step $y_i = g(z_i)$, $x_i = f(y_i)$, where the function $g(\cdot)$ is also smooth, monotonic, and as close to linear as possible, such that $y_i \in (0, 1)$. If the

[40] In most cases this will be the value of the underlying asset.

[41] We take smooth to mean the function is $C^n$ $n \geq 2$, i.e. at least continuous up to the second derivative.

[42] provided the limit has been taken correctly.

physical grid must hit a certain value $x^*$ (i.e. a strike), then the corresponding value of $y$ is given by $y^* = f^{-1}(x^*)$. The nearest $z$ point to this is given by

$$z^* = \frac{\text{round}(y^*(N+1))}{N+1}$$

where round$(\cdot)$, rounds a real number to its nearest integer value. There are three fixed values of $y$ $(0, y^*, 1)$ and three corresponding values of $z$ $(0, z^*, 1)$, so we can simply fit a quadratic through these points, which gives the function $g(\cdot)$. For multiple fixed internal points, we use a cubic spline[43] to define $g(\cdot)$. Figure 12 shows the mapping between index and physical space, $S$, when an approximately uniform grid is required.

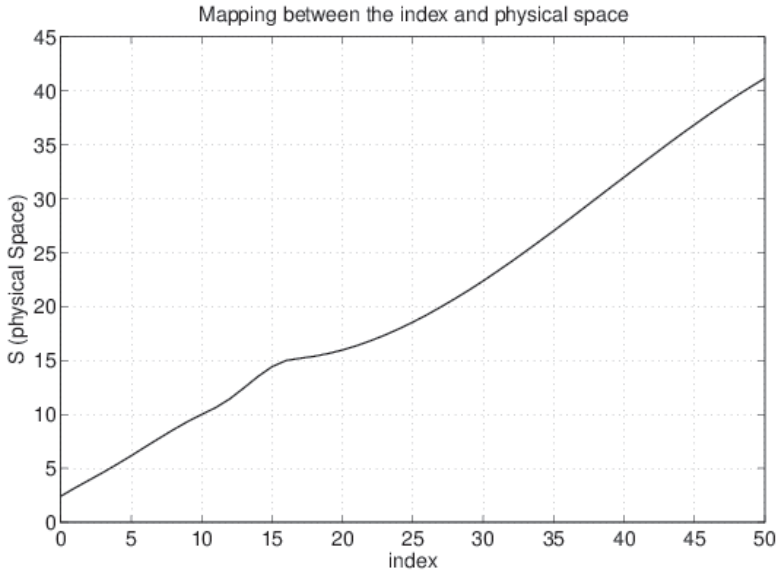


Figure 12: Mapping of index to physical space, where (approximately) uniform spacing is required but a number of grid points must be at specified positions. There are 49 internal points, $S_{min} = 2.43$, $S_{max} = 41.13$, the fixed points are at $(10, 15, 15.2)$ and the anti-points are at $(5, 11)$.

Figure 13 shows an example of a space-time grid where the time points are generated with an exponential mesh ($\lambda = 3.0$ and 25 points between 0 and 2.0) and the spatial points with a hyperbolic mesh ($\alpha = 0.05$ and 40 points between 0 and 500, centred at 100).

## B.3 Specifying Nodes Symmetrically Around Fixed Value (anti-point)

We achieve greatest accuracy by having the strike lie midway between two nodes. It is usually acceptable to only approximate this condition. For a fixed value $y^{**}$, which we call an anti-point, there are a number of approaches:

[43] If two (or more) points must be specified that both share the same closest point, an extra point must be added.

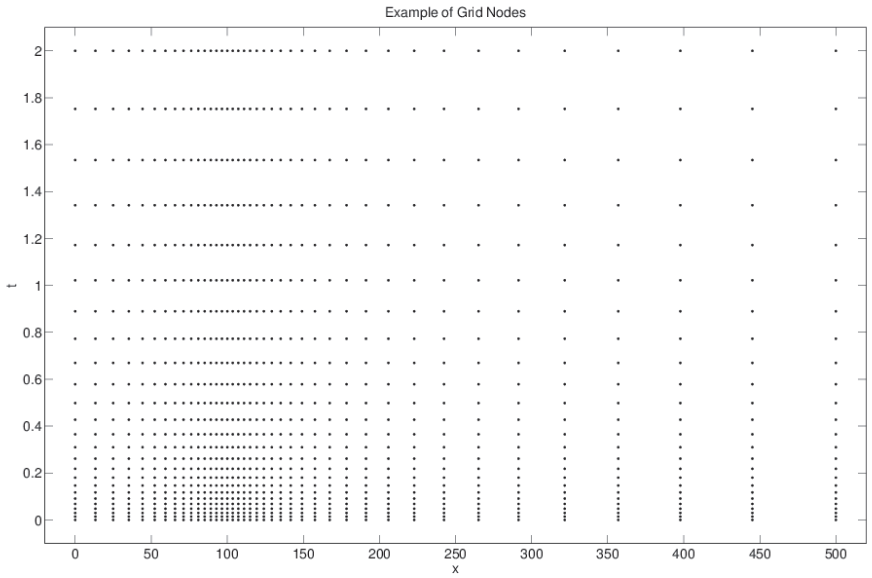


Figure 13: The PDE grid for the parameters detailed in the text. The grids we use in the examples have more points, but are otherwise the same.

- Compute the (approximate) grid spacing $dy = \frac{1}{N}$, then define two fixed points as $y_1^* = y^{**} - dy$ and $y_2^* = y^{**} + dy$. These can then be treated as fixed points as in section B.2. The point $y^{**}$ lies exactly midway between two nodes, although not midway between two physical nodes. For the physical nodes the point is a fraction from the upper node given by

$$\frac{1}{2}\left[1 + \frac{\Delta y}{2}\frac{f''(y^{**})}{f'(y^{**})}\right]$$

  which is acceptable if the curvature of $f(\cdot)$ is not too large.

- Find the nearest half-integer point (anti-node) given by

$$z^{**} = \frac{\min\left(N+1, \text{round}(y^{**}(N+1) + \frac{1}{2})\right) - \frac{1}{2}}{N+1}$$

  We can then use these 'anti-node' points , along with any exact points, to fit a cubic spline to define $g(\cdot)$. The point $y^{**}$ is not exactly halfway between nodes, its fraction position is

$$\frac{1}{2}\left[1 + \frac{1}{4(N+1)}\frac{g''(z^{**})}{f'(z^{**})}\right]$$

  This is usually very close to 0.5 since the curvature of $g(\cdot)$ should be small. As above, the transformation to the physical nodes, introduces an additional shift away from the centre point.

We prefer the second method as it introduces half the number of nodes into the cubic spline, and very close nodes (which the first method has) can produce spurious results. Both methods can generate poor grids if anti-points are required too close together, and/or points are specified too close the anti-points. A particular case is if the strike is at the spot - then we require a point and anti-point at the same position, which is of course impossible. Since the error is maximum if the strike is on a node, the best solution is to make the strike (and thus the spot) lie halfway between nodes. We then have to interpolate between nodes to find the price at the initial spot.

# C Coordinate transforms

## C.1 Time Independent transforms

In the last section, we began with a near uniform[44] set of points $y_i$ and transformed to the physical space by $x_i = f(y_i)$. We then take first and second derivatives using these non-uniform points. An alternative is to apply a change of variable to the original PDE first. Letting $x = f(y)$, $f'(y) = \frac{dx}{dy}$, $f''(y) = \frac{d^2x}{dy^2}$, and $\bar{V}(y,t) = V(f(y),t)$, the spatial derivatives become

$$\frac{\partial V(x,t)}{\partial t} = \frac{\partial \bar{V}(y,t)}{\partial t}$$

$$\frac{\partial \bar{V}(y,t)}{\partial y} = \frac{\partial V(x,t)}{\partial x} f'(y) \quad \rightarrow \quad \frac{\partial V(x,t)}{\partial x} = \frac{1}{f'(y)} \frac{\partial \bar{V}(y,t)}{\partial y}$$

$$\frac{\partial^2 V(x,t)}{\partial x^2} = \frac{1}{f'(y)} \frac{\partial}{\partial y} \left( \frac{1}{f'(y)} \frac{\partial \bar{V}(y,t)}{\partial y} \right) = -\frac{f''(y)}{(f'(y))^3} \frac{\partial \bar{V}(y,t)}{\partial y} + \frac{1}{(f'(y))^2} \frac{\partial^2 \bar{V}(y,t)}{\partial y^2}$$

The first expression for the second derivative can be useful if there is not an analytic form for the second derivative of the transform $f$. Dropping the over-bars on $V$, our generic PDE becomes

$$\frac{\partial V}{\partial t} + \frac{a(y,t)}{(f'(y))^2} \frac{\partial^2 V}{\partial y^2} + \left( \frac{b(y,t)}{f'(y)} - \frac{a(y,t) f''(y)}{(f'(y))^3} \right) \frac{\partial V}{\partial y} + c(y,t) V = 0 \tag{49}$$

While this means we will now be working with a near uniform grid, we could of course work with an exactly uniform grid $z$ by setting $x = f(g(z))$ and replacing terms

$$f'(y) \rightarrow f'(g(z)) g'(z) \qquad f''(y) \rightarrow f'(g(z))(g'(z))^2 + f'(g(z)) g''(z)$$

where $g'(z) = \frac{dg(z)}{dz}$ and $g''(z) = \frac{d^2 g(z)}{dz^2}$ are known if $g(\cdot)$ is formed by a spline method.

### C.1.1 Non-uniform Grid versus Change of Variable

Without any loss of generality we assume that $y$ is a uniform grid with spacing $h$. The non-uniform (physical) grid spacings are then

$$\Delta_i = x_{i+1} - x_i = f(y_i + h) - f(y_i) = h f'(y_i) + \frac{h^2}{2} f''(y_i) + \mathcal{O}(h^3)$$

The first spatial derivative of $V$ on this non-uniform grid is

$$\frac{\partial V}{\partial x} \approx \frac{1}{f'(y_i)} \frac{\left(1 - \frac{h f''(y_i)}{f'(y_i)}\right) V_{i+1} + 2 \frac{h f''(y_i)}{f'(y_i)} V_i - \left(1 + \frac{h f''(y_i)}{f'(y_i)}\right) V_{i-1}}{2h}$$

[44] they are only non-uniform if we have fixed points in the interior.

where $V_i = V(y_i)$ and we have only retained terms up to $\mathcal{O}(h)$. Hence, only in the limit that $\frac{hf''(y_i)}{f'(y_i)} \ll 1$, are differentiating on a non-uniform grid and changing variables equivalent.

It is of course also possible to solve a PDE with a change of variable on a non-uniform grid. This may be desirable if the change of variable rendered some coefficients constant or zero, but we also wanted to concentrate grid points in a certain region (e.g. near a barrier).

### C.2 Time Dependent Change of Variables

Our physical grid is time independent. If it were time dependent, $x_{i,j} = f(y_i, t_j)$, at every time step we would need to paste the solution from the previous time step (formed on the spatial grid $x_{i,j-1}$) onto the new grid. For three-point central differencing, cubic splines will be sufficient to maintain accuracy, although this does involve solving another tridiagonal system (for the cubic spline) at each time step.

If instead we transform the original PDE via $x = f(y,t)$ with $f'(y,t) = \frac{\partial x}{\partial y}$ and $f''(y,t) = \frac{\partial^2 x}{\partial y^2}$, and $\dot{f}(y,t) = \frac{\partial x}{\partial t}$, we have[45]

$$\frac{\partial V}{\partial t} + \frac{a(y,t)}{(f'(y))^2}\frac{\partial^2 V}{\partial y^2} + \left(\frac{b(y,t)}{f'(y)} - \frac{\dot{f}(y,t)}{f'(y,t)} - \frac{a(y,t)f''(y)}{(f'(y))^3}\right)\frac{\partial V}{\partial y} + c(y,t)V = 0 \tag{50}$$

The transformed system should be checked for stability (see appendix D), in particular any transform that makes the new convection coefficient large at any point could be unstable.

An example of when this is useful, is for time dependent barriers. In the double barrier case, if $L(t)$ is the lower barrier and $U(t)$ is the upper, then the transformation[TR00]

$$x = f(y,t) = L(t) + y(U(t) - L(t))$$

is appropriate.

## D Stability Analysis

A simple (although not exact since it ignores boundary conditions) way to analyse the stability of the system is von Neumann's method which considers the independent eigenmodes of the system[PTVF07, TR00]. We consider a solution for a uniform grid of the form

$$v_{i,j} = \xi^j \exp(\sqrt{-1}ki\Delta)$$

where $k$ is the wavenumber and $\xi$ (which is a function of $k$) is the complex amplification number (our previous use of the letter $i$ means we must use $\sqrt{-1}$ to denote an imaginary number). A necessary condition for stability is that $|\xi| \leq 1$. Plugging this into equation 24 we find

$$\xi = \left(1 + \delta\left(\frac{4}{\Delta^2}a_{i,j}\sin^2\frac{k\Delta}{2} + c_{i,j}\right)\right) + \sqrt{-1}\left(\delta\frac{b_{i,j}}{\Delta}\sin k\Delta\right)$$

In the simple case that $b = c = 0$, the stability condition becomes

$$\delta\frac{2}{\Delta^2}|a_{i,j}|\sin^2\frac{k\Delta}{2} \leq 1 \quad \rightarrow \quad \delta \leq \frac{\Delta^2}{2a}$$

[45] A similar expression appears in [TR00], however that contains two typos.

where we have exploited the fact that $a_{i,j} < 0$ in any stable system, and $a$ is taken as the largest value of $|a_{i,j}|$ on the grid [Wil06]. This is known as the *Courant condition*. For the Black-Scholes equation, with $N+2$ space points and $M+2$ time points, this becomes

$$M \geq (N+1)^2 \sigma^2 T - 1$$

which for reasonable parameters of $T = 2.0$, $\sigma = 0.3$ and $N = 100$, means we require $M \geq 1836$, which is excessive. Furthermore, with a non-uniform grid the stability condition will be approximately

$$\delta_j \leq \min_i \left( \frac{\Delta_i^2}{2a_{i,j}} \right)$$

which means that if you concentrate grid points around some financially significant value (e.g. near at-the-money), as in figure 13, the required time steps will become absolutely tiny.

## D.1 Stability of the Theta Method

Plugging the eigenmodes into equation 27 we have

$$\xi = \frac{\left(1 + \theta\delta\left(\frac{4}{\Delta^2} a_{i,j+1} \sin^2 \frac{k\Delta}{2} + c_{i,j+1}\right)\right) + \sqrt{-1}\left(\theta\delta \frac{b_{i,j+1}}{\Delta} \sin k\Delta\right)}{\left(1 - (1-\theta)\delta\left(\frac{4}{\Delta^2} a_{i,j} \sin^2 \frac{k\Delta}{2} + c_{i,j}\right)\right) - \sqrt{-1}\left((1-\theta)\delta \frac{b_{i,j}}{\Delta} \sin k\Delta\right)}$$

Again setting $b = c = 0$ and $a = max(|a_{i,j}|)$, the amplification factor becomes

$$\xi = \frac{1 - \theta\delta\left(\frac{4}{\Delta^2} a \sin^2 \frac{k\Delta}{2}\right)}{1 + (1-\theta)\delta\left(\frac{4}{\Delta^2} a \sin^2 \frac{k\Delta}{2}\right)}$$

For $\theta < 0.5$, the Courant condition is

$$\delta \leq \frac{\Delta^2}{2a(1-2\theta)}$$

while for $\theta \geq 0.5$ we have $|\xi| \leq 1$ for any grid size. This of course does not mean we will get accurate answers for any grid size. More rigorous treatments of stability can be found here [TR00, MM05].

### D.1.1 Lax Equivalence Theorem

A finite difference system is *convergent* if the error (between its solution and that of the continuous PDE) tends to zero as the grid size tends to zero.

The Lax or Lax-Richtmyer theorem[LR56] states that *for a consistent finite difference method for a well-posed linear initial value problem, the method is convergent if and only if it is stable.* Hence to prove that the system is convergent, it is enough to show that the system is stable. Stability here refers to the more rigorous Lax-Richtmyer stability in which the matrix norm is at most unity.

# References


[AP10] Leif Andersen and Vladimir Piterbarg. *Interest Rate Modeling – Volume I: Foundations and Vanilla Models*. Atlantic Financial Press, 2010. 1

[BS02] Petter Bjerksund and Gunnar Stensland. Closed form valuation of american options. *Working paper NHH*, 2002. 23

[Cla11] Iain J. Clark. *Foreign exchange option pricing*. Wiley Finance, 2011. 1, 3

[CN47] J Crank and P Nicolson. A practical method for numerical evaluation of solutions of partial differential equations of the heat conduction type. *Proc. Camb. Phil. Soc.*, 43 (1):50–67, 1947. 7

[CPS09] Richard W. Cottle, Jong-Shi Pang, and Richard E. Stone. *The Linear Complementarity Problem*. Siam, 2009. 22

[Duf04] Daniel J. Duffy. A critique of the crank nicolson scheme strengths and weaknesses for financial instrument pricing. *Wilmott magazine*, 2004. 12

[Duf06] Daniel J. Duffy. *Finite Difference Methods in Financial Engineering: A Partial Differential Equation Approach*. Wiley, 2006. 1, 6, 7, 9

[Fai11] Richard L. Burden & J. Douglas Faires. *Numerical Analysis*. Brooks/Cole, 2011. 1, 21, 22

[Fis95] A. Fischer. A newton-type method for positive-semidefinite linear complementarity problems. *Journal of Optimization Theory and Applications*, 86:585–608, 1995. 22

[FR08] Gianluca Fusai and Andrea Roncoroni. *Implementing Models in Quantitative Finance: Methods and Cases*. Springer, 2008. 1

[Hau07] Espen Gaarder Haug. *Option Pricing Formulas*. McGraw-Hill, 2007. 19

[Hir13] Ali Hirsa. *Computational Methods in Finance*. Chapman & Hall, 2013. 1

[LR56] P. D. Lax and R. D. Richtmyer. Survey of the stability of linear finite difference equations. *Comm. Pure Appl. Math*, 9:267–293, 1956. 31

[MM05] K. W. Morton and D. F. Mayers. *Numerical Solutions of Partial Differential Equations*. Cambridge university Press, 2005. 1, 31

[PTVF07] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brain P. Flannery. *Numerical Recipes*. Cambridge University Press, 2007. 1, 8, 10, 21, 30

[Sey09] Rüdiger U. Seydel. *Tools for Computational Finance*. Springer, 2009. 1

[TR00] Domingo Tavella and Curt Randall. *Pricing Financial Instruments - the Finite Difference Method*. Wiley, 2000. 1, 6, 19, 21, 26, 30, 31

[Tre00] Lloyd N Trefethen. *Spectral Methods in Matlab*. Siam, 2000. 5

[Urb07] Prof. Dr. Karsten Urban. Numerical finance. Technical report, Universität Ulm, Institut für Numerische Mathematik, 2007. http://www.mathematik.uni-ulm.de/numerik/teaching/ss09/NumFin/NumFin_SS09.pdf. 21

[Whi12a] Richard White. Equity variance swap with dividends. Technical report, OpenGamma working paper, 2012. http://developers.opengamma.com/quantitative-research/Equity-Variance-Swaps-with-Dividends-OpenGamma.pdf. 24

[Whi12b] Richard White. Local volatility. OG notes, OpenGamma, 2012. http://developers.opengamma.com/quantitative-research/Local-Volatility-OpenGamma.pdf. 3

[Wil06] Paul Wilmott. *Quantitative Finance*. Wiley, 2006. 1, 6, 21, 31

# OpenGamma Quantitative Research


1. Marc Henrard. Adjoint Algorithmic Differentiation: Calibration and implicit function theorem. November 2011.
2. Richard White. Local Volatility. January 2012.
3. Marc Henrard. My future is not convex. May 2012.
4. Richard White. Equity Variance Swap with Dividends. May 2012.
5. Marc Henrard. Deliverable Interest Rate Swap Futures: Pricing in Gaussian HJM Model. September 2012.
6. Marc Henrard. Multi-Curves: Variations on a Theme. October 2012.
7. Richard White. Option pricing with Fourier Methods. April 2012.
8. Richard White. Equity Variance Swap Greeks. August 2012.
9. Richard White. Mixed Log-Normal Volatility Model. August 2012.
10. Richard White. Numerical Solutions to PDEs with Financial Applications. February 2013.

## About OpenGamma

OpenGamma helps financial services firms unify their calculation of analytics across the traditional trading and risk management boundaries.

The company's flagship product, the OpenGamma Platform, is a transparent system for front-office and risk calculations for financial services firms. It combines data management, a declarative calculation engine, and analytics in one comprehensive solution. OpenGamma also develops a modern, independently-written quantitative finance library that can be used either as part of the Platform, or separately in its own right.

Released under the open source Apache License 2.0, the OpenGamma Platform covers a range of asset classes and provides a comprehensive set of analytic measures and numerical techniques.

**Find out more about OpenGamma** **Download the OpenGamma Platform**


**Europe**
OpenGamma
185 Park Street
London SE1 9BL
United Kingdom

**North America**
OpenGamma
230 Park Avenue South
New York, NY 10003
United States of America