Qfβ = Master en Banca y Finanzas Cuantitativas

Finite-difference Numerical Methods of Partial Differential Equations in Finance with Matlab.

Professor: Aitor Bergara
UPV/EHU

http://www.ehu.es/aitor
Email: a.bergara@ehu.es

Program

Objectives.
Bibliography.

1. Ordinary Differential Equations (ODE).
   - 1.1 Analytic resolution: second order ODE with constant coefficients (example: the harmonic oscillator).
   - 1.2 Some basics:
     - 1.2.a Geometric interpretation of a derivative.
     - 1.2.b Taylor series.
   - 1.3 Numerical resolution of ODEs:
     - 1.3.a Forward Euler (explicit) method (Matlab Program 1).
     - 1.3.b Backward Euler (implicit) method.
     - 1.3.c Midpoint method (Matlab Program 2).
     - 1.3.d Second-order Runge-Kutta method (or trapezoidal).
     - 1.3.e Fourth-order Runge-Kutta method.
   - 1.4 Order Reduction and a system of ODEs.
   - 1.5 Errors and Stability.

2. Introduction to Partial Differential Equations (PDEs): Finite-difference Methods I.
   - 2.1 Definition of a partial derivative. The gradient operator.
   - 2.1 Classification of second order linear PDEs depending on two variables.
   - 2.2 Analysis of differences in the solution of PDEs.
   - 2.3 Boundary and initial conditions.
   - 2.4 Finite-difference representations of advection (hyperbolic) PDE:
     - 2.4.a Explicit forward time centred space method (FTCS) (Matlab Program 3).
       - 2.4.a.1 Stability analysis: Von Neumann method.
     - 2.4.b Lax method (Matlab Program 4).
       - 2.4.b.1 Stability Analysis: Courant-Friedrichs-Lewy condition.
     - 2.4.c Staggered leapfrog method.

3. Finite-difference Methods II: The Heat (or Diffusion) Parabolic PDE.
   - 3.1 Explicit forward time centred space method (FTCS) (Matlab Program 5).
     - 3.1.a Stability analysis.
   - 3.2 Implicit methods:
     - 3.2.a Fully implicit method (Matlab Program 6).
       - 3.2.a.1 Stability analysis.
     - 3.2.b Crank-Nicholson method. (Matlab Program 7).
       - 3.2.b.1 Stability analysis.
       - 3.2.b.2 LU decomposition (Matlab Program 8).
       - 3.2.b.3 SOR (Successive Over-relaxation) Method:
         - ∞ Jacobi method.
         - ∞ Gauss-Seidel method.
         - ∞ Optimal SOR method.

4. The Black-Scholes Equation.
   - 4.1 Derivation of Black-Scholes equation.
   - 4.2 Analysis of advection and diffusion in the Black-Scholes equation.
   - 4.3 Basic assumptions of Black-Scholes equation.
   - 4.4 Boundary and initial/final conditions of Black-Scholes PDE.
   - 4.5 Different payoffs at expiry. European and American Options.
   - 4.6 Transformation to constant coefficient diffusion equation.
   - 4.7 Derivation of Black-Scholes Formulae.
   - 4.8 Analysis of the Greeks.
   - 4.9 Extensions of Black-Scholes Equation.

5. Finite-difference Representations for the Black-Scholes Equation.
   - 5.1 Explicit methods:
     - 5.1.a Derivation of explicit FTCS finite-difference representation.
     - 5.1.b Implementation of different boundary conditions.
     - 5.1.c Local and global errors.
     - 5.1.d Analysis of von Neumann stability (Matlab Program 9).
   - 5.2 Implicit Methods:
     - 5.2.a Fully implicit finite-difference representation.
     - 5.2.b Crank-Nicholson method.
       - 5.2.b.1 Implementation of boundary conditions.
       - 5.2.b.2 Matrix Inversion.
       - 5.2.b.3 LU decomposition.
       - 5.2.b.4 SOR Method.

6. Other Finite-difference Methods for the Black-Scholes Equation.
   - 6.1 Philosophy behind any new method.
   - 6.2 Douglas scheme.
   - 6.3 Three time level methods: Du-Fort Frankel.
   - 6.4 Richardson extrapolation.
   - 6.5 Free boundary problems: American options.
     - 6.5.a Early exercise and the explicit method (Matlab Program 10).
     - 6.5.b Early exercise and the implicit methods.

Objectives

The basics of this course stand on the **Black-Scholes** equation, which values the price of an option by using PDEs. The study of PDEs in complete generality is a vast undertaking. As almost all of them are not possible to solve analytically (however, one very useful exception are European Call/Put options) we must **rely on numerical methods,** and the most popular ones are the Finite-difference Methods.

With this course we do not intend to become experts in 15 hours in order to solve PDEs numerically, but **develop both intuition and technical strength** required to *survive* when such a problem needs to be solved.

Bibliography

P. Wilmott, S. Howison, and J. Dewinne, *The Mathematics of Financial Derivatives*, Cambridge University Press, 1996.

D. Tavella and C. Randall, *Pricing Financial Instruments*, John Willey Sons Inc., 2000.

G.D. Smith, *Numerical Solution of Partial Differential Equations: Finite Difference Methods*, Clarendon Press, Oxford, 1985.

J.C. Strikwerda, *Finite Difference Schemes and Partial Differential Equations*, Chapman & Hall, New York, 1990.

I. Peral, *Ecuaciones en Derivadas Parciales*, Addison Wesley, 1995.

J. C. Hull, Options, Futures & Other Derivatives, Prentice Hall, 2000.

1. Ordinary Differential Equations (ODEs)

The first step before PDEs

Analytical Solution of First Order ODEs

General First-order ODE:

$$\frac{dy}{dt} = f(t,y) \ , \ y(t_0) = y_0$$

Variable-separation method

This method just can be applied to solve the simplest cases. When the two variables contribute separately, so that, $f(t,y) = G(t)H(y)$, the ODE above can be rewritten as:

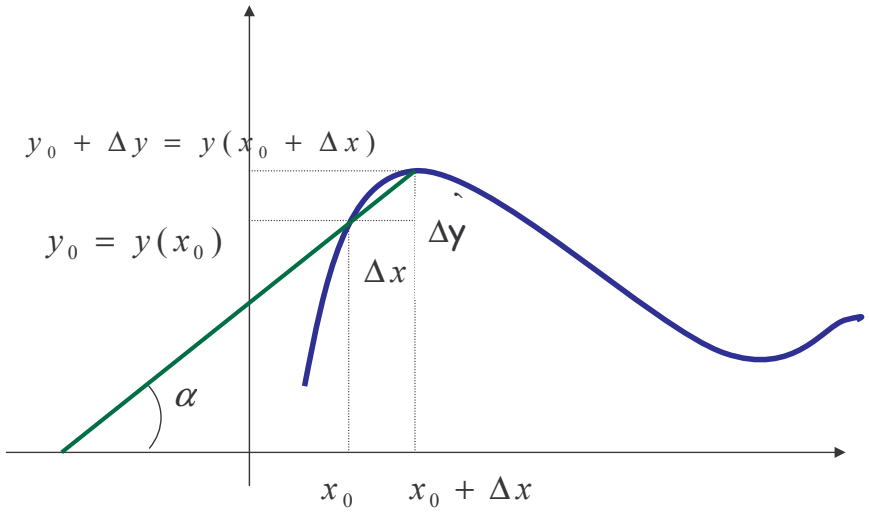
$$\frac{dy}{H(y)} = \frac{dt}{G(t)}, \ y(t_0) = y_0$$

After integrating both sides, the solution is:

$$\int_{y_0}^{y} \frac{dy}{H(y)} = \int_{t_0}^{t} \frac{dt}{G(t)}$$

Basic Technical Points

1. Geometric interpretation of a derivate

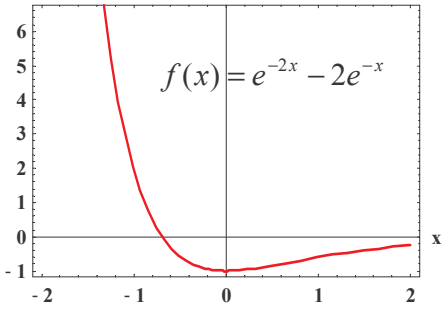


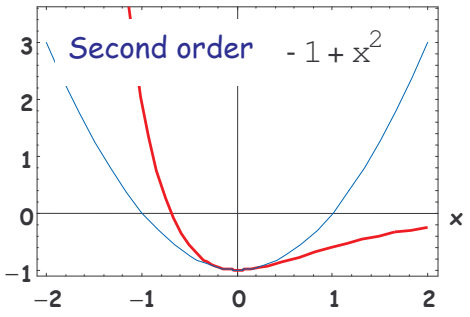
$$\left.\frac{dy}{dx}\right|_{x=x_0} = \lim_{\Delta x \to 0} \frac{y(x_0 + \Delta x) - y(x_0)}{\Delta x}$$

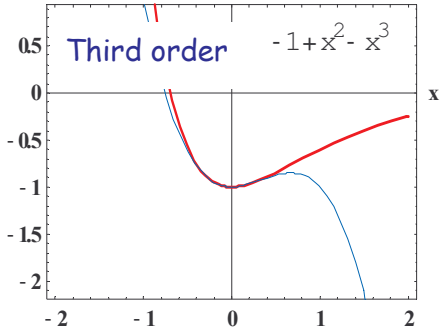
2. Taylor expansion:

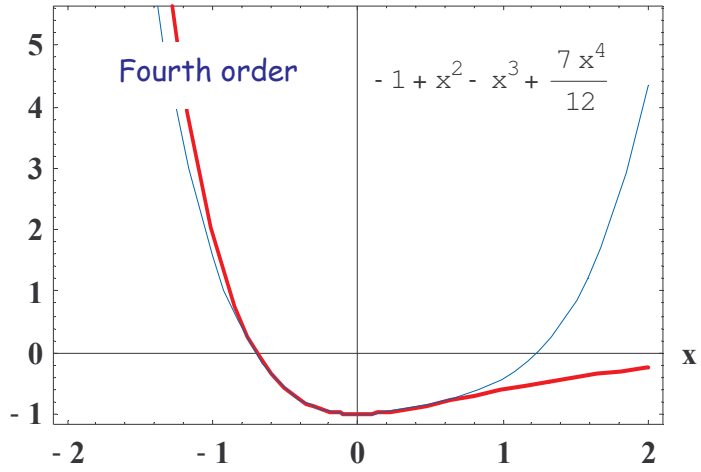
$$f(x)=f(x_0)+\sum_{n=1}^{N}\frac{1}{n!}\left.\frac{d^n f}{dx^n}\right|_{x=0}(x-x_0)^n+O(x-x_0)^{N+1}$$

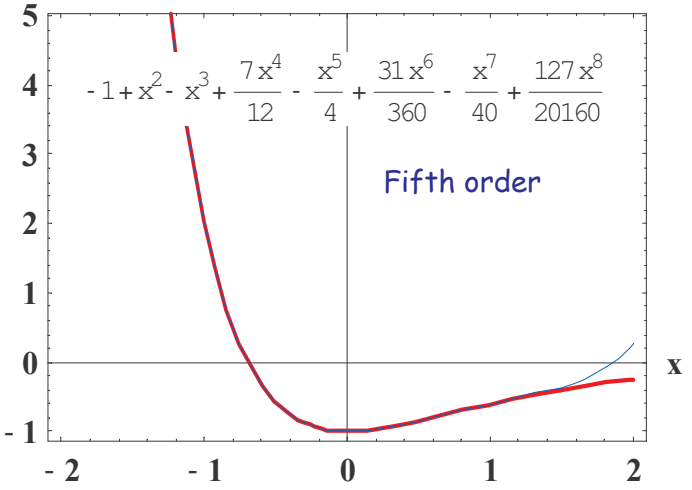
Application:











Numerical Solution of First Order ODEs

1. Forward Euler Method

Forward Euler (FE) method is the simplest and most obvious numerical ODE integrator. It uses the slope at each point, computed using the ODE, to extrapolate and find the next point:

$$y(t+\Delta t) \approx y(t) + \frac{dy(t)}{dt}\Delta t = y(t) + f(t,y)\Delta t$$

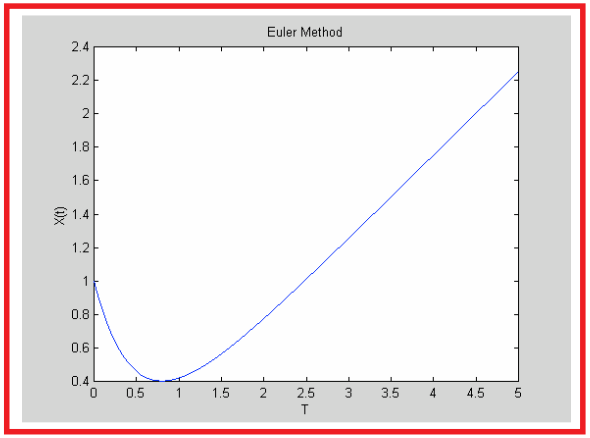
So:

$$y_{i+1} = y_i + f(t_i, y_i)\Delta t + \mathrm{O}(\Delta t^2) \qquad t_i = t_0 + i\Delta t$$

(Explicit method)

Matlab program 1: Integrate the first order ODE system $x'(t) = -2x(t) + t$ with $x(0) = 1$ using forward Euler.

```
clear;

% Coefficients of the equation: a x'=b x + c t
a=1.;
b=-2.;
c=1.;

% Initial and final times
tinit= 0.;
tmax=5.;

% Number of time steps
maxt = 3000;
dt = (tmax-tinit)/maxt;

% Initial condition
x(1)=1.;
t(1)=tinit;

% Time loop
for j=1:maxt;
x(j+1)=x(j)+dt*((b*x(j)+c*(j)*dt)/a);
t(j+1)=tinit+j*dt;
end;

% Figure
plot(t,x)
title('Euler Method')
xlabel('T')
ylabel('X(t)')
```



2. Backward Euler Method

In the Backward Euler (BE) the right-hand side is evaluated at a new location:

$$y_{i+1} = y_i + f(t_{i+1}, y_{i+1})\Delta t + O(\Delta t^2)$$

(Implicit method)

3. Midpoint Method

Symmetric method to evaluate the derivative, using a Taylor expansion that involves only odd powers of Δt,

More accurate !

$$y_{i+1} = y_{i-1} + 2 f(t_i, y_i)\Delta t + O(\Delta t^3)$$

but requires special initialisation to generate additional values that are needed from the past:

$$y_{-1} = y_0 - \Delta t \; f(t_0, y_0)$$


```
% Matlab Program 2: Solving the a x'=b x + c t ODE by using the
midpoint method
clear;

%  Coefficients of equation a x'=b x + c t
a=1.;
b=-2.;
c=1.;

%  Initial and Final Times
tinit= 0.;
tmax=5.;

%  Number of Time Steps
maxt = 3000;
dt = (tmax-tinit)/maxt;

%  Initial Condition
x(2)=1.;
x(1)=1.0-dt*((b*x(2)+c*(2)*dt)/a);
t(2)=tinit;

%  Time Loop
for j=2:(maxt+1);
x(j+1)=x(j-1)+2.*dt*((b*x(j)+c*(j)*dt)/a);
t(j+1)=tinit+(j-1)*dt;
end;

plot(t,x)
title('Midpoint Method')
xlabel('T')
ylabel('X(t)')
```

or Second-order Runge-Kutta (or Trapezoidal)

We can write the midpoint method as:

$$y_{i+1} = y_i + f\left(t_i + \frac{\Delta t}{2}, y\left(t_i + \frac{\Delta t}{2}\right)\right) \Delta t$$

and the initialisation problem is eliminated in the second-order Runge-Kutta method, by using extrapolation for the intermediate step:

$$\begin{aligned} k_1 &= \Delta t \; f(t_i, y_i) \\ k_2 &= \Delta t \; f\left(t_i + \frac{\Delta t}{2}, y_i + \frac{k_1}{2}\right) \\ y_{i+1} &= y_i + k_2 + O(\Delta t^3) \end{aligned}$$

4. Fourth-order Runge-Kutta

$$\begin{aligned} k_1 &= \Delta t \; f(t_i, y_i) \\ k_2 &= \Delta t \; f\left(t_i + \frac{\Delta t}{2}, y_i + \frac{k_1}{2}\right) \\ k_3 &= \Delta t \; f\left(t_i + \frac{\Delta t}{2}, y_i + \frac{k_2}{2}\right) \\ k_4 &= \Delta t \; f(t_i + \Delta t, y_i + k_3) \\ y_{i+1} &= y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(\Delta t^5) \end{aligned}$$

There are many other methods, i.e. step variable methods.

Source of Errors (will be extended in next chapter)

- Round-off error comes from finite precision arithmetic.
- Truncation error comes from the method.
- Stability.

Numerical dynamics are a pernicious problem: the effects of the integration method, the time step, the computer arithmetic, etc, …, can mess with the system's behaviour *in ways that look exactly like real effects*. You should always distrust your results and do some basic belief checks on them: change time step, use a different method, use double-precision arithmetic instead of single, etc…, and see if your results change. If they don't, it is safe(r) to trust on them.

Analytical Solution of Nth-Order LODEs

For example, a general 2nd-order LODE is:

$$p(x)y'' + q(x)y' + s(x)y = h(x) \ , \ y(t_0) = y_0 \text{ and } y'(t_0) = y_0'$$

Nth-Order LODE with constant coefficients

This method just can be applied to solve LODEs with constant coefficients, which for the 2nd-order LODE becomes: $py'' + qy' + sy = h$. Consider that r_1 and r_2 are the roots of the characteristic equation of the 2nd-order LODE $pr^2 + qr + s = 0$.

The following cases can be considered:

- $r_1 \neq r_2$: $y(t) = c_1 e^{r_1 t} + c_2 e^{r_2 t} + h/s.$
- $r_1 = r_2 = r \ (\in \Re)$: $y(t) = (c_1 t + c_2) e^{rt} + h/s.$

c_1 and c_2 are constants that can be obtained in terms of the two initial conditions.

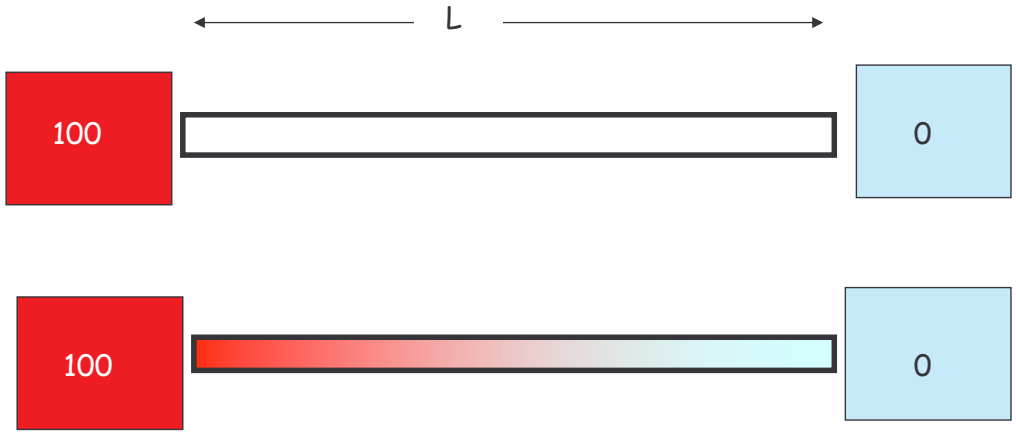
Numerical Solution of Nth-Order LODEs

N-order ordinary differential equations can be substituted by a system of coupled first-order differential equations. For example, any 2nd-order ODE: $p(x)y'' + q(x)y' + r(x)y = h(x)$ with the following initial conditions: $y(x_0) = y_0 \ , y'(x_0) = y_0$ can be replaced by the system: $\begin{matrix} y_1' = y_2 \\ y_2' = \dfrac{h(x)}{p(x)} - \dfrac{q(x)}{p(x)} y_2 - \dfrac{r(x)}{p(x)} y_1 \end{matrix}$ when the new functions are defined: $y_1(x) = y(x)$ and $y_2(x) = y'(x)$

2. Introduction to Partial Differential Equations (PDEs): Finite-difference Methods I

Transformation from a differential equation to a difference equation

2. Introduction to Partial Differential Equations (PDEs):
Finite-difference Methods I



Heat (or Diffusion) Equation

$$\frac{\partial T(x,t)}{\partial t} - D\frac{\partial^2 T(x,t)}{\partial x^2} = 0$$

Thermal conductivity
(or diffusion coefficient)

and

Boundary conditions: $T(0,t) = 100\ ^\circ$ and $T(L,t) = 0^\circ$

Initial condition: $T(x,0) = 100\cos\frac{\pi x}{2L}$

Finite-difference Numerical Methods
of Partial Differential Equations in Finance with Matlab

Fortunately, almost all PDEs encountered in Finance are linear and 2nd-order. A linear 2nd-order PDE satisfied by a function $u(x,t)$ depending on just two variables (space and time):

$$A\frac{\partial^2 u}{\partial t^2} + 2B\frac{\partial^2 u}{\partial x \partial t} + C\frac{\partial^2 u}{\partial x^2} = D\left(x,t,u,\frac{\partial u}{\partial t},\frac{\partial u}{\partial x}\right)$$

Linear in $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial t}$

Classified into three categories:

∞ $B^2 - AC > 0 \rightarrow$ Hiperbolic:

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 u}{\partial t^2}$$ Wave Equation

∞ $B^2 - AC = 0 \rightarrow$ Parabolic

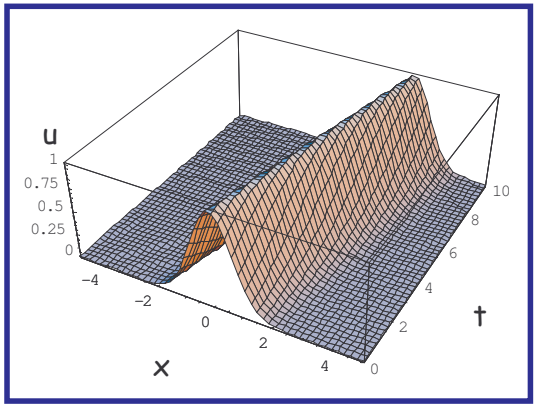
$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2}$$ Diffusion Equation (Black-Scholes)

∞ $B^2 - AC < 0 \rightarrow$ Elliptic (as not related to finance, will not be analyzed)

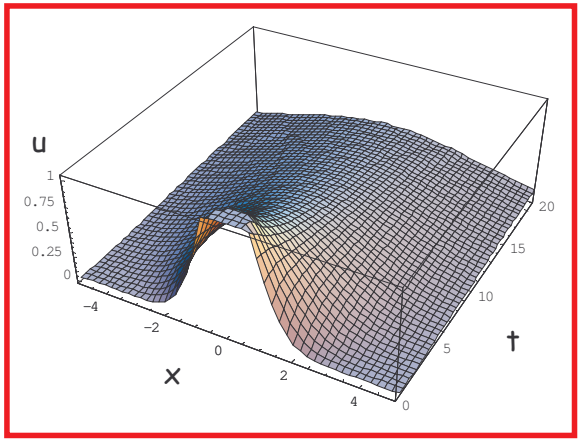
$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$ Laplace Equation

Being $u(x,0) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ (initial pulse) with free boundaries, the solution for each type of equation differs drastically:

∞ Wave (or Advection) Equation: $u(x,t) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x \pm ct)^2}{2}}$



∞ Diffusion Equation: $u(x,t) = \int_{-\infty}^{\infty} dz \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}} \frac{1}{\sqrt{4\pi Dt}} e^{-\frac{(x-z)^2}{4Dt}}$



Boundary/initial conditions

Depending on the problem, initial conditions (IC)

$$u(x,t=0)=u_0(x)$$

and/or boundary conditions (BC):

$$au + b\frac{\partial u}{\partial x} = c, \forall x \in \partial\Omega, \forall t$$

and are called Dirichlet (b=0), Newmann (a=0), or Robin (c=0).

In general terms, the following questions should be asked when considering a PDE:

- Does the equation make sense mathematically? What must we say about the solution on the boundary (boundary conditions) in order to obtain a well-posed problem?
- Can we develop analytical tools to solve the problem?
- How should we solve the equation numerically? What implications do the mathematical properties of the solution have for the numerical method we choose?

Similarity Method for the diffusion equation

Often happens that the solution of a PDE, together with its initial and boundary conditions, depends only on one special combination of the two independent variables. In such cases, the problem can be reduced to an ODE in which this combination is the independent variable. The solution to this ODE is called a similarity solution to the PDE.

The key to the Similarity Solutions in the diffusion equation is that both the equations and the initial and boundary conditions are invariant under the scalings $x \mapsto \lambda x, \tau \mapsto \lambda^2\tau$ for any real number. Such a scaling is called one-parameter group of transformations. Therefore, $x/\sqrt{\tau}$ is the only combination independent of λ in the transformation. In general, a good practical test to find similarity solutions in any equation is to try solutions of the form $u = \tau^\alpha f(x/\tau^\beta)$.

Example 1: Suppose that $u(x,\tau)$ satisfies the following diffusive problem on the semi-infinite interval $x > 0$:

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2}, x, \tau > 0,$$

With the initial condition: $u(x,0) = 0$, and a boundary condition at x=0, $u(x,\tau) = 1$; we also require that $u(x \to \infty, \tau) \to 0$.

As it is explained above, we can look for a solution which depends only on x and τ through the combination $\xi = x/\sqrt{\tau}$, so that, $u(x,\tau) = U(\xi)$ satisfies the following second-order ODE:

$$\frac{d^2U}{d\xi^2} + \frac{1}{2}\xi\frac{dU}{d\xi} = 0.$$

From the initial and boundary conditions, $U(0) = 1,\ U(\infty) = 0.$ Separating variables we find that:

$$U(\xi) = C\int_0^{\xi} e^{-s^2/4}ds + D,$$

where C and D are constants. After applying the boundary conditions, we find that:

$$U(\xi) = \frac{1}{\sqrt{\pi}}\int_{\xi}^{\infty} e^{-s^2/4}ds;$$

that is:

$$u(x,\tau) = \frac{1}{\sqrt{\pi}}\int_{x/\sqrt{\tau}}^{\infty} e^{-s^2/4}ds.$$

Example 2: We derive here the fundamental solution (as it can be considered the basis of a general solution with any initial condition) of the diffusion equation:

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2},\ x, \tau > 0,$$

with the initial condition: $u(x,0) = \delta(x)$ (the Dirac delta function) and $u(x \to \pm\infty, \tau) \to 0$.

We again look for solutions which depends only on x and τ through the combination $\xi = x/\sqrt{\tau}$, but we know try the form, $u_\delta(x,\tau) = \tau^{-1/2}U_\delta(\xi)$, which satisfies the following second-order ODE:

$$\frac{d^2U_\delta}{d\xi^2} + \frac{1}{2}\frac{d(\xi U_\delta)}{d\xi} = 0.$$

The $\tau^{-1/2}$ term is there to ensure $\int_{-\infty}^{\infty} u(x,\tau)dx$ is constant for all τ. The general solution of this is:

$$U_\delta(\xi) = Ce^{-\xi^2/4} + D,$$

where C and D are constants. Choosing D=0 and normalising the solution by setting $C = 1/(2\sqrt{\pi})$, so that $\int_{-\infty}^{\infty} u_\delta(x,\tau)dx = 1$, yields the fundamental solution:

$$u_\delta(x,\tau) = \frac{1}{2\sqrt{\pi\tau}} e^{-x^2/4\tau}$$

This fundamental solution of the diffusion equation can be used to derive an explicit solution to the diffusion equation for $-\infty < x < \infty$ and $\tau > 0$ with arbitrary initial conditions $u(x,0) = u_o(x)$.

The key to the solution is the fact that we can write the initial data as

$$u_o(x) = \int_{-\infty}^{\infty} u_o(\xi)\delta(x-\xi)d\xi.$$

As the function

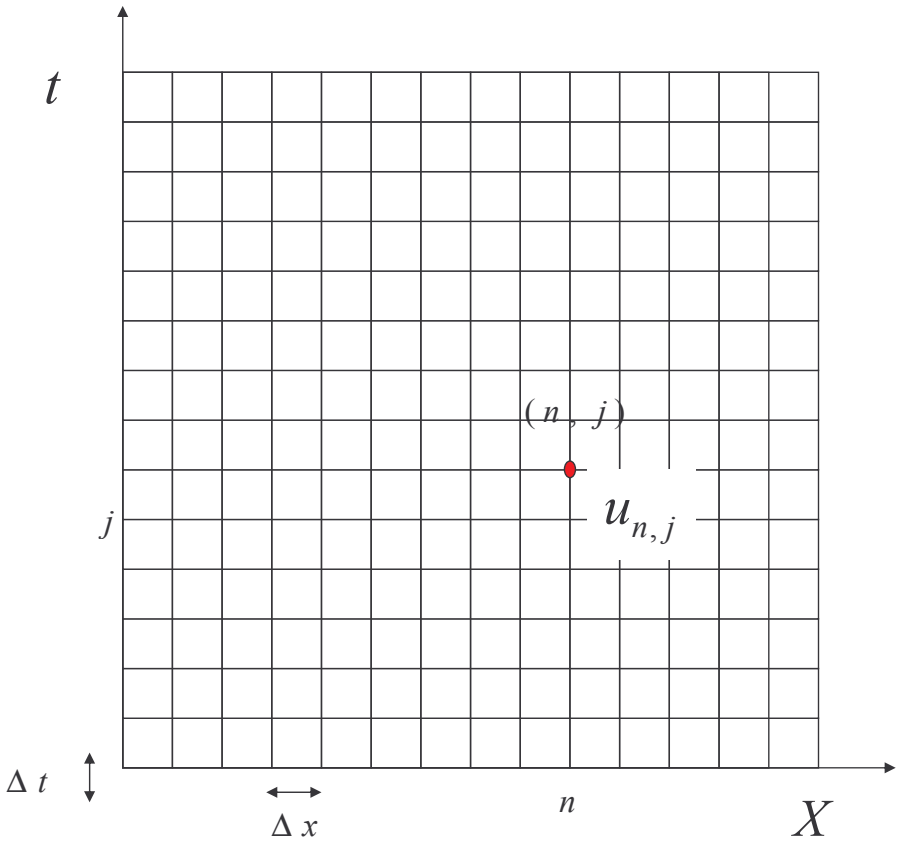
$$u_o(s)u_\delta(s-x,\tau) = \frac{u_o(s)}{2\sqrt{\pi\tau}} e^{-(s-x)^2/4\tau}$$

satisfies the diffusion equation with initial data $u_o(s)\delta(s-x)$ and the diffusion equation is linear, the general solution is found by superposing solutions of this form:

$$u\ (x,\tau) = \frac{1}{2\sqrt{\pi\tau}} \int_{-\infty}^{\infty} u_o(s) e^{-(x-s)^2/4\tau} ds.$$

A grid (mesh) is defined in (x,t) space so that each point is characterized by:

$$x_n = x_0 + n\Delta x,\ n = 0,1,\ldots,N_x$$
$$t_j = t_0 + j\Delta t,\ t = 0,1,\ldots,N_t$$
$$u_{n,j} \equiv u(x_n, t_j)$$



Example: Flux-conservative problem

(Advection Equation)

$$c\frac{\partial u}{\partial x} = \frac{\partial u}{\partial t}$$

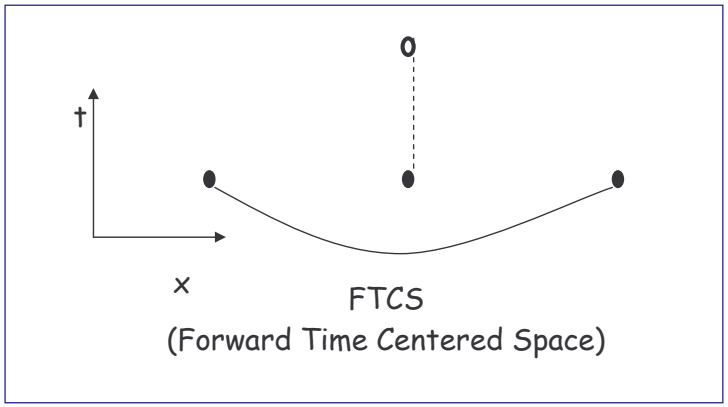
The conservation of "particles" inside a volume is applied: the change on the number of flowing "particles" inside a finite volume is equal to the flux of the current associated to these moving particles.

By using:

$$\frac{\partial u}{\partial t} = \frac{u_{n,j+1} - u_{n,j}}{\Delta t} + O(\Delta t) \quad \text{and} \quad \frac{\partial u}{\partial x} = \frac{u_{n+1,j} - u_{n-1,j}}{2\Delta x} + O(\Delta x^2)$$

Forward Euler

Second-order representation but still using only quantities known at time step j.



FTCS
(Forward Time Centered Space)

$$\frac{u_{n,j+1} - u_{n,j}}{\Delta t} = -c\frac{u_{n+1,j} - u_{n-1,j}}{2\Delta x}$$

or

$$u_{n,j+1} = u_{n,j} - \frac{c\Delta t}{2\Delta x}\left(u_{n+1,j} - u_{n-1,j}\right)$$

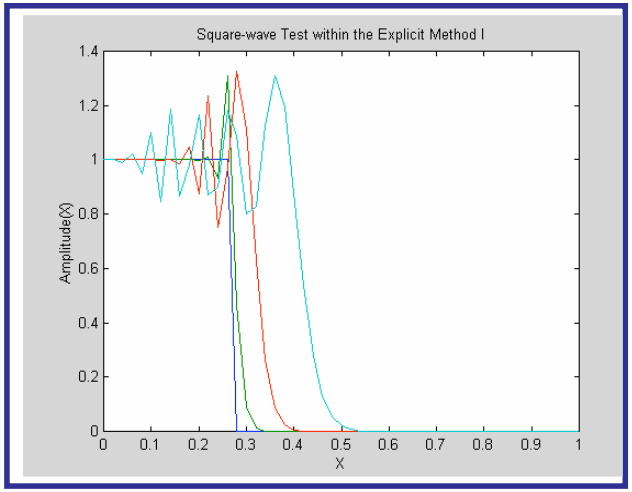
The FTCS representation is an *explicit* scheme: $u_{n,j+1}$ for each n can be calculated explicitly from the quantities that are already known. Latter we shall meet *implicit* schemes, which require us to solve implicit equations coupling the $u_{n,j+1}$ for various n. The FTCS algorithm is also an example of *single-level* scheme, since only values at time level j have to be stored to find values at time level $j+1$.

It is a fine example of an algorithm, that is easy to derive, takes little storage, and executes quickly.

```
%  Matlab Program 3: Square-wave Test for the Explicit Method to solve
%  the Advection Equation
clear;

% Parameters to define the advection equation and the range in space and
% time
Lmax = 1.0;      % Maximum length
Tmax = 1.;       % Maximum time
c = 1.0;         % Advection velocity

%  Parameters needed to solve the equation within the explicit method
maxt = 3000;     % Number of time steps
dt = Tmax/maxt;
n = 30;          %  Number of space steps
nint=15;         %  The wave-front: intermediate point from which u=0
dx = Lmax/n;
b = c*dt/(2.*dx);

%  Initial value of the function u (amplitude of the wave)
for i = 1:(n+1)
   if i < nint
      u(i,1)=1.;
   else
      u(i,1)=0.;
   end
   x(i) =(i-1)*dx;
end

%  Value of the amplitude at the boundary
for k=1:maxt+1
        u(1,k) = 1.;
        u(n+1,k) = 0.;
        time(k) = (k-1)*dt;
end

%  Implementation of the explicit method
for k=1:maxt     %  Time loop
   for i=2:n     %  Space loop
      u(i,k+1) =u(i,k)-b*(u(i+1,k)-u(i-1,k));
   end
end

% Graphical representation of the wave at different selected times
plot(x,u(:,1),'-',x,u(:,10),'-',x,u(:,50),'-',x,u(:,100),'-')
title('Square-wave test within the Explicit Method I')
xlabel('X')
ylabel('Amplitude(X)')
```


2. Introduction to Partial Differential Equations (PDEs):
Finite-difference Methods I

Too bad, it does not work !!??



Just play with different values of b (see in the program script above) to realize how unstable it may become !

Finite-difference Numerical Methods
of Partial Differential Equations in Finance with Matlab

Sources of Errors and Stability

In this example, we see that there are two fundamental sources of error: truncation error in the space and discretizations. The implication of truncation error is that the numerical scheme solves a problem that is not exactly the same as the problem we are trying to solve. The *approximate* solution of our PDE obtained with the numerical scheme can be viewed as the *exact* solution of a different problem. To characterize what numerical scheme does, we need to address three fundamental issues:

1. Consistency: A numerical scheme is said to be consistent if the finite difference representation converges to the PDE we are trying to solve as the space and time steps tend to zero. When spatial and time discretizations are kept separated (as it is the general case) consistency does not appear to be relevant, but has to be checked when both discretizations are mixed.
2. Stability: A numerical scheme is said to be stable if the difference between the numerical solution and the exact solution remains bounded as the number of steps tends to infinity.
3. Convergence: a scheme s said to converge if the difference between the numerical solution at a fixed point in the domain of interest tends to zero uniformly as the space and time discretizations tend to zero (not necessarily independently from each other).

2. Introduction to Partial Differential Equations (PDEs):
Finite-difference Methods I

The Lax Equivalence Theorem links these issues together: *Given a properly posed linear initial value problem and a consistent finite difference scheme, stability is the only requirement for convergence.* This theorem illustrates why we will dedicate much effort to analyzing and understanding stability.

The vague issue of *accuracy* of a scheme is not very relevant in its own right. A consistent scheme can be made increasingly accurate by decreasing the time and spatial steps. What matters is the cost (coding effort, memory requirements and computational requirements) of the accuracy.

Stability Analysis: Fourier Approach (von Neumann)

The stability problem arises because we are using finite precision computer arithmetic to solve the difference equations, which introduces rounding errors into the numerical solution. The system is said to be stable if these rounding errors are not magnified at each iteration. Ask the question, "if a small error is introduced into the solution, is it magnified by the numerical method or does it decay away?".

The Fourier method is based on decomposing the numerical solution into Fourier harmonics on the spatial grid. Although this method does not capture the influence of boundary conditions, it is quite easy to formulate and usually accurate enough to provide practical stability criteria.

Finite-difference Numerical Methods
of Partial Differential Equations in Finance with Matlab

We can decompose the solution into Fourier modes on the mesh:

$$u_{n,j} = \sum_m u_m(t_j) e^{ik_m(n\Delta x)},$$

where $u_m(t_j)$ is the amplitude and k_m is the wavenumber of the mode *m*. A further simplification is that for linear equations, he Fourier modes are uncoupled, so that we might consider them individually. Writing the time dependence of the amplitude[1] in terms of the amplification factor, $\xi(k)$, $u_{n,j} = \xi^j(k) e^{ikn\Delta x}$. If we find that $|\xi(k)| > 1$ then it is unstable.

After substituting this in the equation we have that $\xi(k) = 1 - i\frac{c\Delta t}{\Delta x}\sin k\Delta x$, so that $|\xi(k)| > 1$, therefore, the solution explodes (oscillatory) and becomes unstable!! for any Δt and Δx.

[1] PDEs to be considered have the form:

$$\frac{\partial u}{\partial t} = \mathbf{L}u,$$

where **L** is a partial differential operator containing no time derivatives. The application of spatial discretization to **L**u will result in the following system of equations:

$$\frac{d\mathbf{u}}{dt} = \mathbf{A}\mathbf{u},$$

where **A** matrix that after being diagonalized, $\mathbf{X}^{-1}\mathbf{A}\mathbf{X} = \boldsymbol{\lambda}$, where $\boldsymbol{\lambda}$ is the diagonal matrix with the eigenvalues $\boldsymbol{\lambda}_m$. Introducing the definition $\mathbf{v} = \mathbf{X}^{-1}\mathbf{u}$, the solution verifies: $v_m = c_m e^{\lambda_m t}$, and after time discretization $v_m = c_m e^{\lambda_m (j\Delta t)}$, which goes with the power of the time step, *j*.

To cure the instability: Lax Method

Simply replaces the term $u_{n,j}$ in the time derivative by its average:

$$u_{n,j} \to \frac{1}{2}\left(u_{n+1,j} + u_{n-1,j}\right)$$

The advection equation turns into:

$$u_{n,j+1} = \frac{1}{2}\left(u_{n+1,j} + u_{n-1,j}\right) - \frac{c\Delta t}{2\Delta x}\left(u_{n+1,j} - u_{n-1,j}\right)$$

Which is also explicit, and after applying the *von Neumann stability analysis* we get:

$$\xi(k) = \cos k\Delta x - i\frac{c\Delta t}{\Delta x}\sin k\Delta x$$

and the stability condition $|\xi(k)| < 1$ leads to the requirement:

$$\frac{|c|\Delta t}{\Delta x} \le 1$$

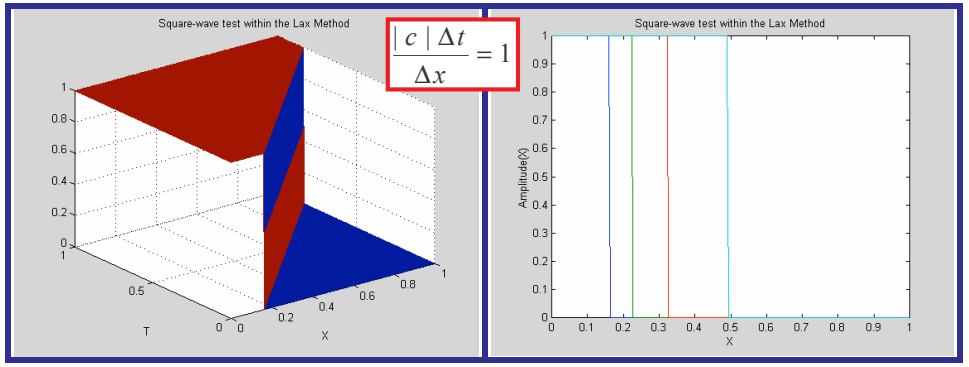
Courant-Friedrichs-Lewy condition

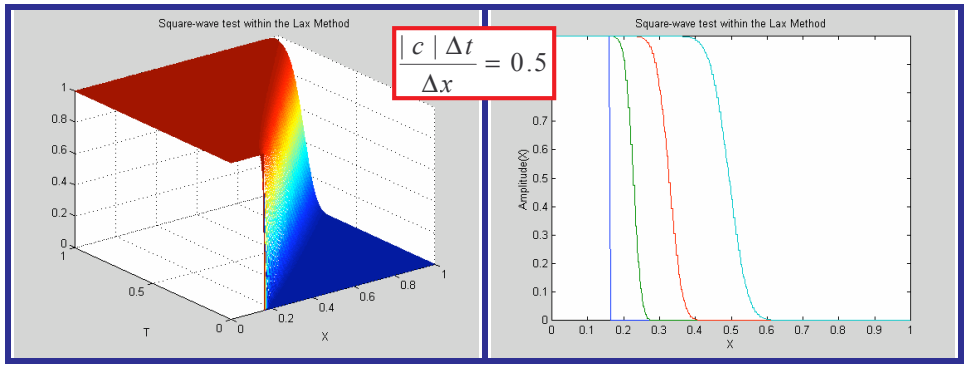
The method has to go faster than the wave !!

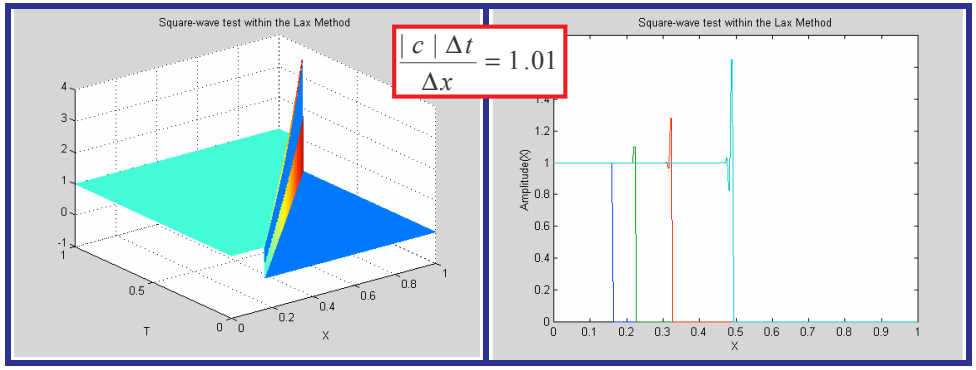
This is art !!

```
%  Matlab Program 4: Step-wave Test for the Lax method to solve the Advection
%  Equation
clear;
% Parameters to define the advection equation and the range in space and time
Lmax = 1.0;     % Maximum length
Tmax = 1.;      % Maximum time
c = 1.0;        % Advection velocity

%  Parameters needed to solve the equation within the Lax method
maxt = 350;        % Number of time steps
dt = Tmax/maxt;
n = 300;           %  Number of space steps
nint=50;           %  The wave-front: intermediate point from which u=0
(nint<n)!!
dx = Lmax/n;
b = c*dt/(2.*dx);
% The Lax method is stable for abs(b)=< 1/2 but it gets difussed unless abs(b)=
% 1/2

%  Initial value of the function u (amplitude of the wave)
for i = 1:(n+1)
   if i < nint
      u(i,1)=1.;
   else
      u(i,1)=0.;
   end
   x(i) =(i-1)*dx;
end

%  Value of the amplitude at the boundary at any time
for k=1:maxt+1
        u(1,k) = 1.;
        u(n+1,k) = 0.;
        time(k) = (k-1)*dt;
end

%  Implementation of the Lax method
for k=1:maxt       %  Time loop
   for i=2:n       %  Space loop
      u(i,k+1) =0.5*(u(i+1,k)+u(i-1,k))-b*(u(i+1,k)-u(i-1,k));
   end
end

% Graphical representations of the evolution of the wave
figure(1)
mesh(x,time,u')
title('Square-wave test within the Lax Method')
xlabel('X')
ylabel('T')

figure(2)
plot(x,u(:,1),'-',x,u(:,20),'-',x,u(:,50),'-',x,u(:,100),'-')
title('Square-wave test within the Lax Method')
xlabel('X')
ylabel('Amplitude(X)')
```


2. Introduction to Partial Differential Equations (PDEs):
Finite-difference Methods I







Finite-difference Numerical Methods
of Partial Differential Equations in Finance with Matlab

Cases:

- $\frac{|c|\Delta t}{\Delta x} > 1 \rightarrow$ The method gets unstable
- $\frac{|c|\Delta t}{\Delta x} < 1 \rightarrow$ The method gets diffusive (it gets worse to get smaller time steps)
- $\frac{|c|\Delta t}{\Delta x} = 1 \rightarrow$ The method converges to the exact result

Proof that the Lax scheme of the advection equation is exactly the FTCS representation of the equation:

$$\frac{\partial u}{\partial t} = c\frac{\partial u}{\partial x} + \frac{(\Delta x)^2}{2\Delta t}\frac{\partial^2 u}{\partial x^2}$$

Second-order Accuracy in Time: Staggered Leapfrog

The previous scheme is *expensive* (and dangerous) computationally. However, there are schemes that are second-order accurate in both space and time, and these can often be pushed right to their stability limit. With correspondingly smaller computation times:

$$\frac{u_{n,j+1} - u_{n,j-1}}{\Delta t} = c\frac{u_{n+1,j} - u_{n-1,j}}{\Delta x}$$

Staggered Leapfrog Model

The von Neumann stability analysis now gives a quadratic equation for $\xi(k)$ rather than a linear one:

$$\xi^2 - 1 = 2i\xi\frac{c\Delta t}{\Delta x}\sin k\Delta x$$

whose solution is:

$$\xi = i\frac{c\Delta t}{\Delta x}\sin k\Delta x \pm \sqrt{1 - \left(\frac{c\Delta t}{\Delta x}\sin k\Delta x\right)^2}$$

Thus the Courant condition is again required for stability, in fact, $|\xi(k)| = 1$ (no diffusion) for any $c\Delta t \le \Delta x$.

3. Finite-difference Methods II

The Heat (or Diffusion) Parabolic PDE

The Diffusive problem

(Heat or diffusion Equation)

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(D \frac{\partial u}{\partial x} \right)$$

The conservation of heat-energy inside a volume is applied: the change in the energy (so that, in the temperature) inside a volume equals the flux of heat, which within the Fourier Law is proportional to the gradient of the temperature (conductivity, D).

If D is constant:

$$\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2}$$

By using:

$$\frac{\partial u}{\partial t} = \frac{u_{n,j+1} - u_{n,j}}{\Delta t} + O(\Delta t) \quad \text{and} \quad \frac{\partial^2 u}{\partial x^2} = \frac{u_{n+1,j} - 2u_{n,j} + u_{n-1,j}}{(\Delta x)^2} + O(\Delta x)$$

Forward Euler.

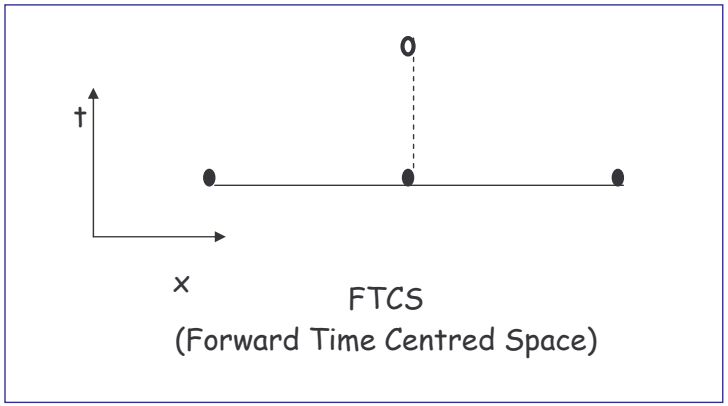
Central difference representation but still using only quantities known at timestep j.

$$\frac{u_{n,j+1}-u_{n,j}}{\Delta t} = D\frac{u_{n+1,j}-2u_{n,j}+u_{n-1,j}}{(\Delta x)^2}$$

Or

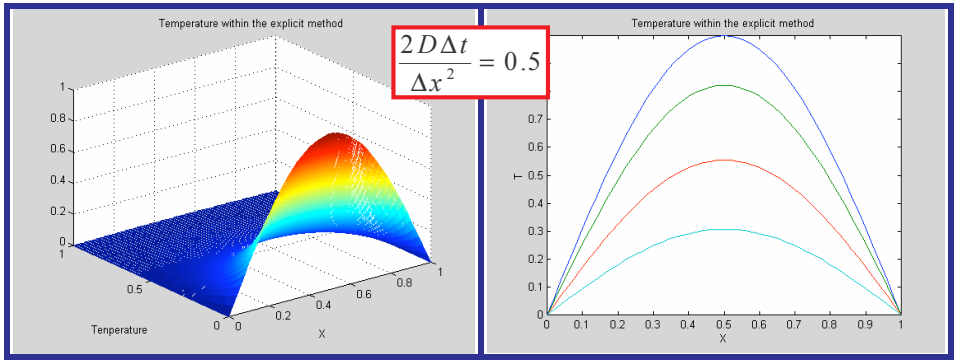
$$u_{n,j+1} = \alpha u_{n+1,j} + (1-2\alpha)u_{n,j} + \alpha u_{n-1,j}$$

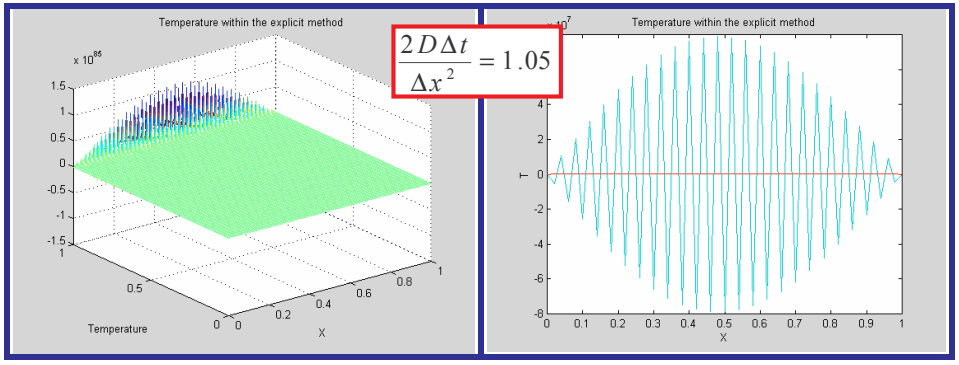
with $\alpha = \frac{D\Delta t}{(\Delta x)^2}$.



FTCS
(Forward Time Centred Space)


```
%  Matlab Program 5: Heat Diffusion in one dimensional wire within the
%  Explicit Method
clear;

% Parameters to define the heat equation and the range in space and time
L = 1.;       %  Length of the wire
T =1.;        %  Final time


%  Parameters needed to solve the equation within the explicit method
maxk = 2500;                %  Number of time steps
dt = T/maxk;
n = 50;                     %  Number of space steps
dx = L/n;
cond = 1/4;                 %  Conductivity
b = 2.*cond*dt/(dx*dx);     %  Stability parameter (b=<1)

%  Initial temperature of the wire: a sinus.
for i = 1:n+1
        x(i) =(i-1)*dx;
        u(i,1) =sin(pi*x(i));
end

%  Temperature at the boundary (T=0)
for k=1:maxk+1
        u(1,k) = 0.;
        u(n+1,k) = 0.;
        time(k) = (k-1)*dt;
end

%  Implementation of the explicit method
for k=1:maxk        %  Time Loop
   for i=2:n;       %  Space Loop
      u(i,k+1) =u(i,k) + 0.5*b*(u(i-1,k)+u(i+1,k)-2.*u(i,k));
   end
end

% Graphical representation of the temperature at different selected times
figure(1)
plot(x,u(:,1),'-',x,u(:,100),'-',x,u(:,300),'-',x,u(:,600),'-')
title('Temperature within the explicit method')
xlabel('X')
ylabel('T')

figure(2)
mesh(x,time,u')
title('Temperature within the explicit method')
xlabel('X')
ylabel('Temperature')
```





Stability Analysis: Von Neumann

This is a FTCS scheme again, but having a second derivative makes a world of difference ! The FTCS was unstable for the advection equation (hyperbolic), but trying independent solutions of the form $u_{n,j} = \xi^j(k)e^{ikn\Delta x}$, we have that $\xi(k) = 1 - \frac{4D\Delta t}{(\Delta x)^2}\sin^2\left(\frac{k\Delta x}{2}\right)$, so that the requirement $|\xi(k)| \le 1$, leads to the stability criterion:

$$\frac{2D\Delta t}{\Delta x^2} \le 1$$

The maximum allowed time step is the diffusion time across a cell of width Δx.

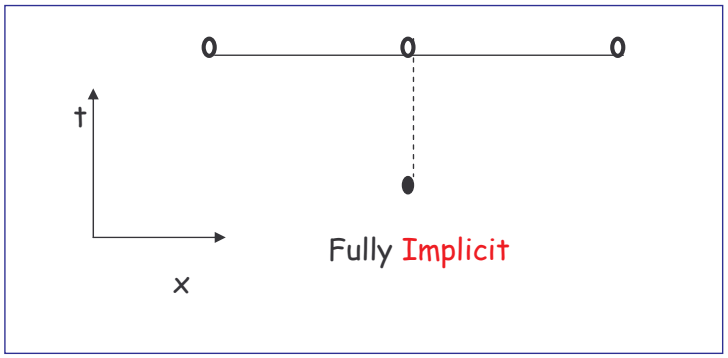
However, this condition implies huge limitations on the calculation procedure. For example, if we want to analyse with great detail in space ($\Delta x << 1$) implies that $\Delta t << 1$ so that a huge number of steps will be required until something interesting happens. The computational requirement may be enormous and, therefore, new methods are required.

Implicit Methods

A. Fully implicit scheme (or backward in time)

$$\frac{u_{n,j+1}-u_{n,j}}{\Delta t}=D\frac{u_{n+1,j+1}-2u_{n,j+1}+u_{n-1,j+1}}{\Delta x^2}$$

This is like the FTCS scheme except that the spatial derivatives on the right-hand side are evaluated at time step j+1.



One has to solve a set of simultaneous linear equations at each time step for the $u_{n,j+1}$. Fortunately, this is a *simple* problem because the system is tridiagonal: just group the terms in equation appropriately:

$$u_{n,j}=-\alpha u_{n-1,j+1}+(1+2\alpha)u_{n,j+1}-\alpha u_{n+1,j+1},$$
$$\text{with } n=1,2,\ldots,N-1 \text{ and } \alpha=\frac{D\Delta t}{\Delta x^2}$$

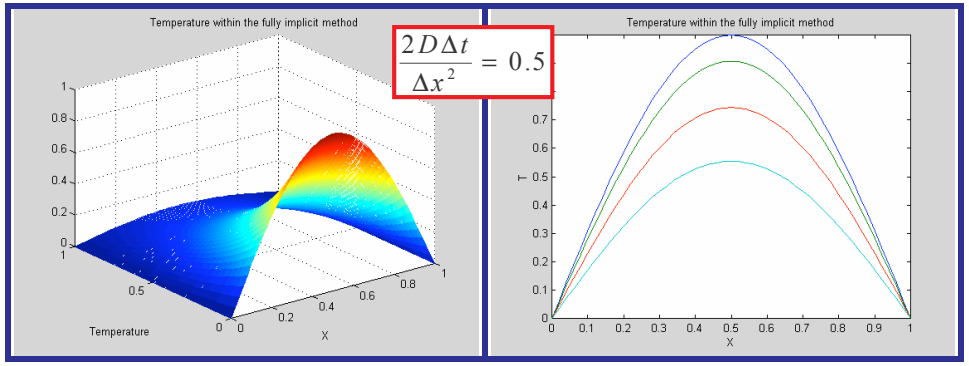
supplemented by Dirichlet or Neumann boundary conditions at n=0 and n=N. These equations will be discussed in depth shortly.

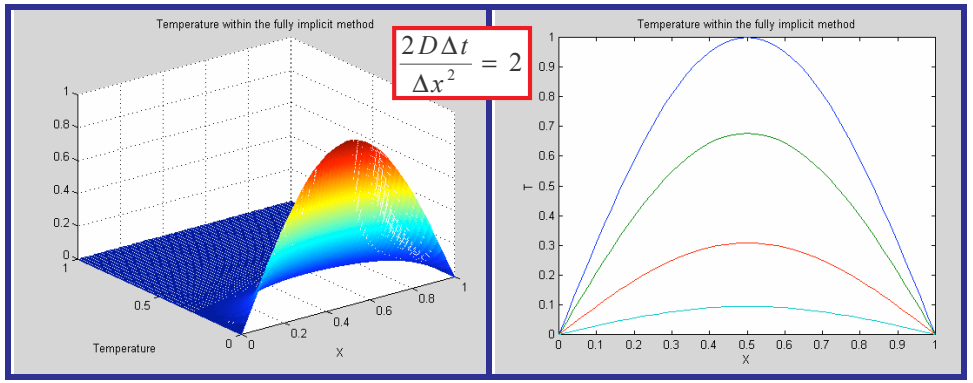
What about stability?

The amplification factor is: $\xi(k)=\dfrac{1}{1+4\alpha\sin^2\left(\dfrac{k\Delta x}{2}\right)}$, which clearly $|\xi(k)|\le 1$ for any Δt. The scheme is unconditionally stable. The details of the small-scale evolution form the initial conditions are obviously inaccurate for large Δt (it is only first-order in time), but the correct equilibrium solution is obtained for $\Delta t\to\infty$.

```
%  Matlab Program 6: Heat Diffusion in one dimensional wire within the Fully
%  Implicit Method
clear;

% Parameters to define the heat equation and the range in space and time
L = 1.;         %  Lenth of the wire
T =1.;          %  Final time

%  Parameters needed to solve the equation within the fully implicit method
maxk = 2500;                %  Number of time steps
dt = T/maxk;
n = 50.;                    %  Number of space steps
dx = L/n;
cond = 1./4.;                  %  Conductivity
b = cond*dt/(dx*dx);        %  Parameter of the method

%  Initial temperature of the wire: a sinus.
for i = 1:n+1
        x(i) =(i-1)*dx;
        u(i,1) =sin(pi*x(i));
end

%  Temperature at the boundary (T=0)
for k=1:maxk+1
        u(1,k) = 0.;
        u(n+1,k) = 0.;
        time(k) = (k-1)*dt;
end

aa(1:n-2)=-b;
bb(1:n-1)=1.+2.*b;
cc(1:n-2)=-b;
MM=inv(diag(bb,0)+diag(aa,-1)+diag(cc,1));


%  Implementation of the implicit method
for k=2:maxk      %  Time Loop
   uu=u(2:n,k-1);
   u(2:n,k)=MM*uu;
end

% Graphical representation of the temperature at different selected times
figure(1)
plot(x,u(:,1),'-',x,u(:,100),'-',x,u(:,300),'-',x,u(:,600),'-')
title('Temperature within the fully implicit method')
xlabel('X')
ylabel('T')

figure(2)
mesh(x,time,u')

title('Temperature within the fully implicit method')
xlabel('X')
ylabel('Temperature')
```


Keeping the mesh fixed and changing the conductivity:



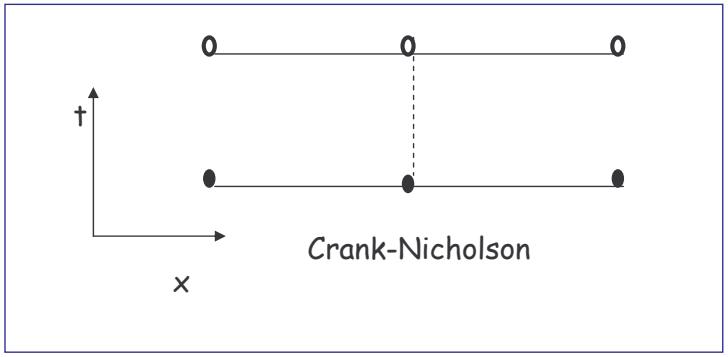


B. Crank-Nicholson Method

(highly recommended !)

Combines the stability of an implicit method with the accuracy of a method that is second-order in both space and time. Simply from the average of the explicit and implicit FTCS schemes (left- and right hand side are centred at time step j+1/2):

$$\frac{u_{n,j+1}-u_{n,j}}{\Delta t}=\frac{D}{2}\left(\frac{(u_{n+1,j+1}-2u_{n,j+1}+u_{n-1,j+1})+(u_{n+1,j}-2u_{n,j}+u_{n-1,j})}{\Delta x^2}\right)$$



3. Finite-difference Methods II: the Heat Equation

What about stability?

The amplification factor is: $\xi(k)=\dfrac{1-2\alpha\sin^2\left(\dfrac{k\Delta x}{2}\right)}{1+2\alpha\sin^2\left(\dfrac{k\Delta x}{2}\right)}$, which clearly $|\xi(k)|\le 1$ for any Δt. The scheme is unconditionally stable and second-order both in time and space. It is worthy to analyse it more deeply.

Finite-difference Numerical Methods
of Partial Differential Equations in Finance with Matlab

The Crank-Nicholson can be written as:

$$\alpha u_{n-1,j} + (2-2\alpha)u_{n,j} + \alpha u_{n+1,j} = -\alpha u_{n-1,j+1} + (2+2\alpha)u_{n,j+1} - \alpha u_{n+1,j+1}$$

These equations only holds for $1 \le n \le N-1$. The boundary conditions again supply the two missing equations. They are harder to handle than in the explicit method and I will discuss them.

The Crank-Nicholson method can be written in a matrix form:

$$\begin{pmatrix} -\alpha & 2+2\alpha & -\alpha & 0 & . & . & . \\ 0 & -\alpha & 2+2\alpha & -\alpha & . & . & . \\ . & 0 & . & . & . & . & . \\ . & . & . & . & 2+2\alpha & -\alpha & 0 \\ . & . & . & . & -\alpha & 2+2\alpha & -\alpha \end{pmatrix} \begin{pmatrix} u_{0,j+1} \\ u_{1,j+1} \\ . \\ . \\ . \\ u_{N-1,j+1} \\ u_{N,j+1} \end{pmatrix} =$$

$$\begin{pmatrix} \alpha & 2-2\alpha & \alpha & 0 & . & . & . \\ 0 & \alpha & 2-2\alpha & \alpha & . & . & . \\ . & 0 & . & . & . & . & . \\ . & . & . & . & 2-2\alpha & \alpha & 0 \\ . & . & . & . & \alpha & 2-2\alpha & \alpha \end{pmatrix} \begin{pmatrix} u_{0,j} \\ u_{1,j} \\ . \\ . \\ . \\ u_{N-1,j} \\ u_{N,j} \end{pmatrix}$$

The two matrices have *N-1* rows and *N+1* columns, which is a representations of the *N-1* equations and *N+1* unknowns. The two equations that we are missing come from the boundary conditions. Using these conditions, I am going to convert this system of equations into a system of equations involving a square matrix. The aim is to write a system of equations in the form:

$$\mathbf{M}_{j+1}^{L}\mathbf{u}_{j+1} + \mathbf{r}_{j+1}^{L} = \mathbf{M}_{j}^{R}\mathbf{u}_{j} + \mathbf{r}_{j}^{R}$$

For known square matrices $\mathbf{M}_{j+1}^{L}$ and $\mathbf{M}_{j}^{R}$, and a known vectors $\mathbf{r}$, where the details of the boundary conditions have been fully incorporated.

Example of boundary condition: given $u_{0,j+1}$ and $u_{N,j+1}$: Sometimes we know the value of the *u* function on the boundary (*n=0* and *n=N*). In this case, we can write:

$$\begin{pmatrix} -\alpha & 2+2\alpha & -\alpha & 0 & . & . & . \\ 0 & -\alpha & 2+2\alpha & -\alpha & . & . & . \\ . & 0 & . & . & . & . & . \\ . & . & . & . & 2+2\alpha & -\alpha & 0 \\ . & . & . & . & -\alpha & 2+2\alpha & -\alpha \end{pmatrix} \begin{pmatrix} u_{0,j+1} \\ u_{1,j+1} \\ . \\ . \\ . \\ u_{N-1,j+1} \\ u_{N,j+1} \end{pmatrix}$$

as

$$\begin{pmatrix} 2+2\alpha & -\alpha & 0 & . & . & . \\ -\alpha & 2+2\alpha & . & . & . & . \\ 0 & . & . & . & -\alpha & . \\ . & . & . & -\alpha & 2+2\alpha & -\alpha \\ . & . & . & 0 & -\alpha & 2+2\alpha \end{pmatrix} \begin{pmatrix} u_{1,j+1} \\ . \\ . \\ . \\ . \\ u_{N-1,j+1} \end{pmatrix} + \begin{pmatrix} -\alpha u_{0,j+1} \\ 0 \\ . \\ . \\ . \\ -\alpha u_{N,j+1} \end{pmatrix}$$

$$= \mathbf{M}^{L}_{j+1}\mathbf{u}_{j+1} + \mathbf{r}^{L}_{j+1}$$

and the same for the matrices on the right.

Whichever of the boundary conditions we have, the Crank-Nicholson scheme, with boundary conditions incorporated is:

$$\mathbf{M}^{L}_{j+1}\mathbf{u}_{j+1} = \mathbf{r}^{R}_{j} - \mathbf{r}^{L}_{j+1} + \mathbf{M}^{R}_{j}\mathbf{u}_{j}$$

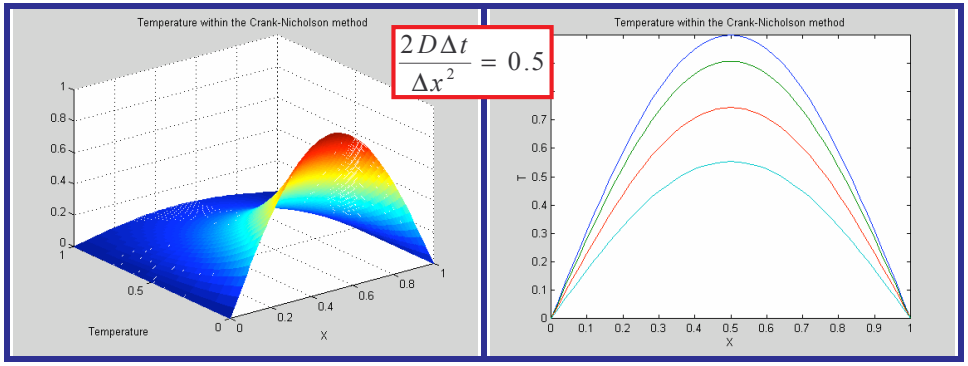
How do we then find $\mathbf{u}_{j+1}$? In principle, the matrix $\mathbf{M}^{L}_{j+1}$ could be inverted to give:

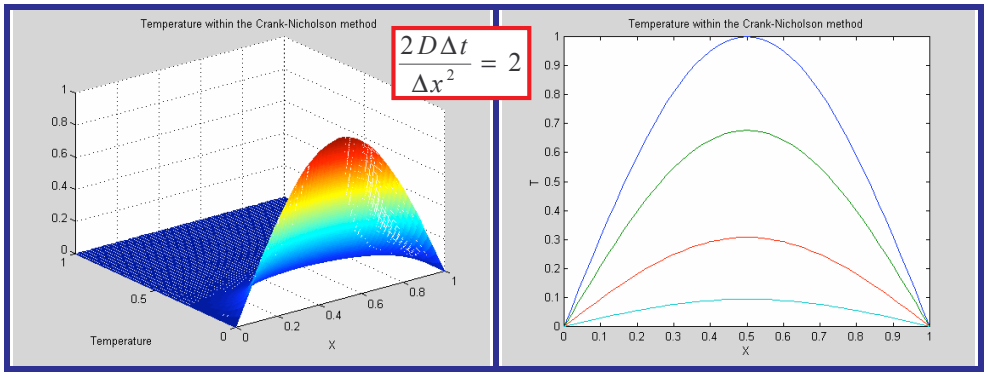
$$\mathbf{u}_{j+1} = (\mathbf{M}^{L}_{j+1})^{-1}(\mathbf{r}^{R}_{j} - \mathbf{r}^{L}_{j+1} + \mathbf{M}^{R}_{j}\mathbf{u}_{j})$$

However, matrix inversion is very time consuming and computationally inefficient. Two much better ways will be explained below:


```
%  Matlab Program 7: Heat Diffusion in one dimensional wire within the
%  Crank-Nicholson Method
clear;

% Parameters to define the heat equation and the range in space and time
L = 1.;         %  Lenth of the wire
T =1.;          %  Final time

%  Parameters needed to solve the equation within the  Crank-Nicholson method
maxk = 2500;                %  Number of time steps
dt = T/maxk;
n = 50.;                    %  Number of space steps
dx = L/n;
cond = 1/2;                 %  Conductivity
b = cond*dt/(dx*dx);        %  Parameter of the method

%  Initial temperature of the wire: a sinus.
for i = 1:n+1
        x(i) =(i-1)*dx;
        u(i,1) =sin(pi*x(i));
end

%  Temperature at the boundary (T=0)
for k=1:maxk+1
        u(1,k) = 0.;
        u(n+1,k) = 0.;
        time(k) = (k-1)*dt;
end

% Defining the Matrices M_right and M_left in the method
aal(1:n-2)=-b;
bbl(1:n-1)=2.+2.*b;
ccl(1:n-2)=-b;
MMl=diag(bbl,0)+diag(aal,-1)+diag(ccl,1);

aar(1:n-2)=b;
bbr(1:n-1)=2.-2.*b;
ccr(1:n-2)=b;
MMr=diag(bbr,0)+diag(aar,-1)+diag(ccr,1);


%  Implementation of the Crank-Nicholson method
for k=2:maxk      %  Time Loop
   uu=u(2:n,k-1);
   u(2:n,k)=inv(MMl)*MMr*uu;
end

% Graphical representation of the temperature at different selected times
figure(1)
plot(x,u(:,1),'-',x,u(:,100),'-',x,u(:,300),'-',x,u(:,600),'-')
title('Temperature within the Crank-Nicholson method')
xlabel('X')
ylabel('T')

figure(2)
mesh(x,time,u')

title('Temperature within the Crank-Nicholson method')
xlabel('X')
ylabel('Temperature')
```

Keeping the mesh fixed and changing the conductivity:





a. LU Decomposition (In Matlab: [L,U]=lu(M))

The matrix $\mathbf{M}_{j+1}^{L}$ is tridiagonal, and it is not hard to decompose into the product of two other matrices, one having nonzero elements along the diagonal and the subdiagonal ($\mathbf{L}$) and the other having non-zero elements along the diagonal and the superdiagonal ($\mathbf{U}$). So that: $\mathbf{M} = \mathbf{L}\,\mathbf{U}$ (that is why it is called LU decomposition).

$$\begin{pmatrix} 2+2\alpha & -\alpha & 0 & . & . & . \\ -\alpha & 2+2\alpha & . & . & . & . \\ 0 & . & . & . & -\alpha & . \\ . & . & . & -\alpha & 2+2\alpha & -\alpha \\ . & . & . & 0 & -\alpha & 2+2\alpha \end{pmatrix} =$$

$$\begin{pmatrix} 1 & 0 & 0 & . & . & . \\ l_2 & 1 & 0 & . & . & . \\ 0 & l_3 & . & . & 0 & . \\ . & . & . & l_{N-2} & 1 & 0 \\ . & . & . & 0 & l_{N-1} & 1 \end{pmatrix} \begin{pmatrix} d_1 & p_1 & 0 & . & . & 0. \\ 0 & d_2 & p_2 & . & . & . \\ 0 & 0 & . & . & p_{N-3} & 0 \\ . & . & . & 0 & d_{N-2} & p_{N-2} \\ . & . & . & 0 & 0 & d_{N-1} \end{pmatrix}$$

Where, without loss of generality, I have chosen the diagonal elements of $\mathbf{L}$ to be one.

It can be seen that the following relations are verified:

$$d_1 = 2 + 2\alpha\,,$$

$$l_n d_{n-1} = p_{n-1} = -\alpha \quad \text{and} \quad d_n = 2 + 2\alpha - l_n p_{n-1} \quad \text{for } 2 \le n \le N-1$$

Notice that we work form n=1 to n=N sequentially.

Now we exploit the decomposition to solve:

$$\mathbf{M}\,\mathbf{u} \leftarrow \mathbf{M}^{L}_{j+1}\mathbf{u}_{j+1} = \mathbf{r}^{R}_{j} - \mathbf{r}^{L}_{j+1} + \mathbf{M}^{R}_{j}\mathbf{u}_{j} \rightarrow \mathbf{q}$$

$$\mathbf{M}\,\mathbf{u} = \mathbf{q}\,,\ \mathbf{LU}\,\mathbf{u} = \mathbf{q}\,,\ \mathbf{L}\mathbf{w} = \mathbf{q}\,,\ \mathbf{U}\,\mathbf{u} = \mathbf{w}$$

And then we are almost done.

- The first step gives: $w_1 = q_1$ and $w_n = q_n - l_n w_{n-1}$ for $2 \le n \le N-1$, where we again must work sequentially.

- The second step involves working backwards from n=N-2 to n=1: $u_{N-1} = {w_{N-1}}/{d_{N-1}}$ and $u_n = \dfrac{w_n - p_n u_{n+1}}{d_n}$ for $N-2 \ge n \ge 1$

If our matrix equation is time independent the LU decomposition needs to be done only once.


```
%  Matlab Program 8: Heat Diffusion in one dimensional wire within the
%  Crank-Nicholson Method
%  by using the LU decomposition.
clear;
% Parameters to define the heat equation and the range in space and time
L = 1.;         %  Lenth of the wire
T = 1.;         %  Final time
%  Parameters needed to solve the equation within the  Crank-Nicholson method
%  by using the LU decomposition
maxk = 2500;                    %  Number of time steps
dt = T/maxk;
n = 50.;                        %  Number of space steps
dx = L/n;
cond = 1./4.;                   %  Conductivity
b = cond*dt/(dx*dx);            %  Parameter of the method

%  Initial temperature of the wire: a sinus.
for i = 1:n+1
        x(i) =(i-1)*dx;
        v(i,1) =sin(pi*x(i));
end

%  Temperature at the boundary (T=0)
for k=1:maxk+1
        v(1,k) = 0.;
        v(n+1,k) = 0.;
        time(k) = (k-1)*dt;
end

% Defining the Matrices M_right and M_left in the method
aal(1:n-2)=-b;
bbl(1:n-1)=2.+2.*b;
ccl(1:n-2)=-b;
MMl=diag(bbl,0)+diag(aal,-1)+diag(ccl,1);
[L,U]=lu(MMl);        % LU decomposition
aar(1:n-2)=b;
bbr(1:n-1)=2.-2.*b;
ccr(1:n-2)=b;
MMr=diag(bbr,0)+diag(aar,-1)+diag(ccr,1);
%  Implementation of the LU decomposition within the Crank-Nicholson method
for k=2:maxk      %  Time Loop
   vv=v(2:n,k-1);
   qq=MMr*vv;
   w(1)=qq(1);
   for j=2:n-1
      w(j)=qq(j)-L(j,j-1)*w(j-1);
   end
   v(n,k)=w(n-1)/U(n-1,n-1);
   for i=n-1:-1:2
      v(i,k)=(w(i-1)-U(i-1,i)*v(i+1,k))/U(i-1,i-1);
   end
end
% Graphical representation of the temperature at different selected times
figure(1)
plot(x,v(:,1),'-',x,v(:,100),'-',x,v(:,300),'-',x,v(:,600),'-')
title('Temperature within the Crank-Nicholson method (LU)')
xlabel('X')
ylabel('T')
figure(2)
mesh(x,time,v')
title('Temperature within the Crank-Nicholson method (LU)')
xlabel('X')
ylabel('Temperature')
```

b. SOR (Successive Over-relaxation)

This is an indirect method (has to be solved iteratively) to solve $\mathbf{Mu} = \mathbf{q}$ matrix equation and although the resulting solution will never be exact we can find it to whatever accuracy we want. Besides, indirect methods can be applied to wider range of problems, for example, $\mathbf{M}$ matrix need not be tridiagonal (I will describe the ideas more generally).

The system of equations can be written as:

$$\begin{aligned} M_{11}u_1 + M_{12}u_2 + \ldots + M_{1N}u_N &= q_1 \\ M_{21}u_1 + M_{22}u_2 + \ldots + M_{2N}u_N &= q_2 \\ \ldots & \\ M_{N1}u_1 + M_{N2}u_2 + \ldots + M_{NN}u_N &= q_N \end{aligned}$$

Where now N is the number of equations (the size of the matrix), and can be rewritten as:

$$\begin{aligned} M_{11}u_1 &= q_1 - (M_{12}u_2 + \ldots + M_{1N}u_N) \\ M_{22}u_2 &= q_2 - (M_{21}u_1 + \ldots + M_{2N}u_N) \\ & \ldots \\ M_{NN}u_N &= q_N - (M_{N1}u_1 + M_{N2}u_2 + \ldots) \end{aligned}$$

The system is easily solved iteratively using:

$$\begin{aligned} u_1^{i+1} &= \frac{1}{M_{11}}\left[q_1 - (M_{12}u_2^i + \ldots + M_{1N}u_N^i)\right] \\ u_2^{i+1} &= \frac{1}{M_{22}}\left[q_2 - (M_{21}u_1^i + \ldots + M_{2N}u_N^i)\right] \\ & \ldots \\ u_N^{i+1} &= \frac{1}{M_{NN}}\left[q_N - (M_{N1}u_1^i + M_{N2}u_2^i + \ldots)\right] \end{aligned}$$

Where the superscript denotes the level of the iteration, which is started with some initial guess $\mathbf{u}^0$. This iterative method is called the Jacobi Method.

I can write the matrix $\mathbf{M}$ as the sum of a diagonal matrix $\mathbf{D}$, an upper triangular matrix $\mathbf{T}$ (with zeros in the diagonal) and a lower triangular matrix $\mathbf{L}$ (with zeros in the diagonal): $\mathbf{M} = \mathbf{D} + \mathbf{T} + \mathbf{L}$. I can use this representation to write the Jacobi and other methods quite elegantly:

$$\mathbf{u}^{i+1} = \mathbf{D}^{-1}\left[\mathbf{q} - (\mathbf{T} + \mathbf{L})\mathbf{u}^i\right]$$

When the Jacobi method is implemented some of the values u_n^{i+1} are evaluated before others. In the Gauss-Seidel method we use the updated values as soon as they are calculated. This method can be written as:

$$u_n^{i+1} = \frac{1}{M_{nn}}\left[q_n - \sum_{j=1}^{n-1} M_{nj}u_j^{i+1} - \sum_{j=n}^{N} M_{nj}u_j^i\right]$$

Generally, iterate methods usually converge to the correct solution from one side (the correction $u_n^{i+1} - u_n^i$ stays on the same side of the sign as i increases). This is used by the SOR method to speed up the convergence. This method can be written as:

$$u_n^{i+1} = (1-\omega)u_n^i + \frac{\omega}{M_{nn}}\left[q_n - \sum_{j=1}^{n-1} M_{nj} u_j^{i+1} - \sum_{j=n}^{N} M_{nj} u_j^i\right]$$

Acceleration or
Over-relaxation parameter,
which must lie between 1 and 2

$$\mathbf{u}^{i+1} = (\mathbf{I} + \omega\mathbf{D}^{-1}\mathbf{L})^{-1}\left[\left((1-\omega)\mathbf{I} - \omega\mathbf{D}^{-1}\mathbf{T}\right)\mathbf{u}^i + \omega\mathbf{D}^{-1}\mathbf{q}\right]$$

Optimal choice of ω:

The error $\mathbf{e}^i = \mathbf{u}^i - \mathbf{u}$, where $\mathbf{u}$ is the exact solution, satisfies:

$$\mathbf{e}^{i+1} = (\mathbf{I} + \omega\mathbf{D}^{-1}\mathbf{L})^{-1}\left[\left((1-\omega)\mathbf{I} - \omega\mathbf{D}^{-1}\mathbf{T}\right)\right]\mathbf{e}^i$$

The SOR method will converge provided that the largest of the moduli of the eigenvalues (the spectral radius), of the SOR matrix is less than 1. There is a theoretical optimum value for ω, that is when the spectral radius is minimum. In practice it is very simple to iterate on ω to find the optimal value.

4. The Black-Scholes Equation

A very special portfolio

We can write the option value as $V(S,t;\sigma, \ ;E,T;r) \equiv V(S,t)$.

One simple observation is that a call option will rise in value if the underlying asset rises (positive correlation) and the opposite for a put option.

Use Π to denote a portfolio of one long option position and a short position in some quantity, Δ , of the underlying, S :

$$\Pi = V(S,t) - \Delta S$$

The underlying follows a lognormal random walk: $dS = Sdt + \sigma SdX$

The change on the value of the portfolio from t to dt is:

$$d\Pi = dV - \Delta dS$$

Not always accurate

and from Ito we have:

$$dV = \frac{\partial V}{\partial t}dt + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2}dt + \frac{\partial V}{\partial S}dS$$

Thus the portfolio changes by:

$$d\Pi = \frac{\partial V}{\partial t}dt + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2}dt + \frac{\partial V}{\partial S}dS - \Delta dS$$

Elimination of risk: Delta Hedging

$$d\Pi = \frac{\partial V}{\partial t}dt + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2}dt + \frac{\partial V}{\partial S}dS - \Delta dS$$

Randomness = Risk

If we chose $\Delta = \frac{\partial V}{\partial S}$ then the randomness is reduced to zero!

Delta hedging is an example of dynamic hedging.

No arbitrage

After choosing the quantity Δ as suggested above, we hold a portfolio whose value changes by the amount:

$$d\Pi = \left(\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2}\right)dt$$

This change is completely riskless, then it must be the same as the growth we would get if we put the equivalent amount of cash in a risk-free interest-bearing account:

$$d\Pi = r\Pi dt$$

Add together two solutions of the equation and you will get a third.

Even if we start with a discontinuity in the final data, due to a discontinuity in the payoff, this immediately gets smoothed out, due to the diffusive nature of the equation.

Fisher Black-Myron Scholes equation (1973):

(Two-dimensional Linear Parabolic PDE)

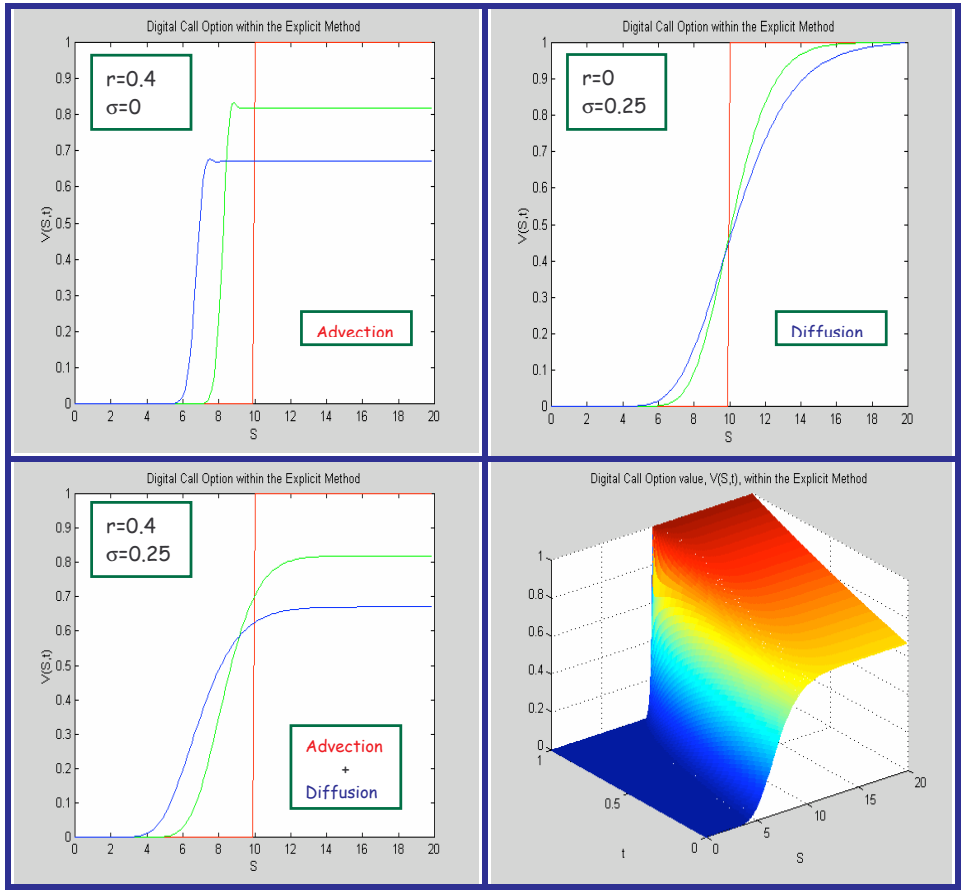
$$\frac{\partial V}{\partial t}+\frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2}+rS\frac{\partial V}{\partial S}-rV=0$$

Basic diffusion contribution, so that any discontinuity in the payoff would be instantly diffused away. The diffusion coefficient is a function of S.

Represents the advection term moving V in a preferred direction (like breeze blows the smoke).

This is a reaction term. Balancing this term and the time derivative would give a model for decay of a radioactive body.

Just an example (see the differences): Digital Call European Option with E=10€, T=1 year.



In the curves red represents the value of the option at expiry, green half a year before that, and the blue one year before, that is, when the contract is signed (the price).

The Black-Scholes assumptions:

(Even though all of the assumptions can be shown to be wrong to a greater or lesser extent, the *Black-Scholes model is profoundly important both in theory and in practice*)

- The underlying follows a lognormal random world.
- The risk-free interest rate is a known function of time.
- There are no dividends on the underlying.
- Delta hedging is done continuously.
- There are no transaction costs on the underlying.
- There are no arbitrage opportunities.

Boundary and Initial/Final Conditions

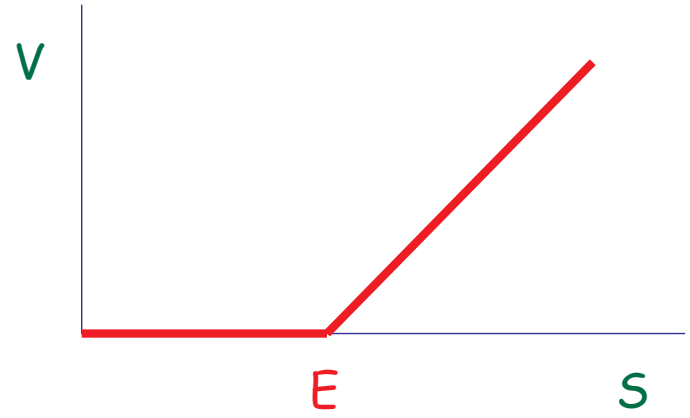
The Black-Scholes equation knows nothing about what kind of option we are valuing, whether it is a call or a put, nor what is the strike and the expiry, which are dealt with the final condition.

To uniquely specify the problem we must prescribe boundary conditions (how the solution must behave for all time at certain values of the asset, usually at $S = 0$ and $S \to \infty$) and initial, $t = 0$, or final conditions, $t = T$.

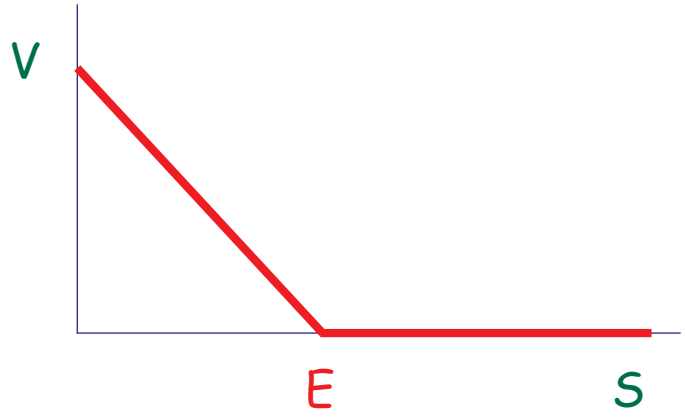
The Black-Scholes equation is a backward equation (the signs of the t derivative and the second S derivative in the equation are the same when written on the same side), therefore, a final condition (usually the payoff function $V(S,T)$ at expiry) has to be imposed.

Payoffs at Expiry

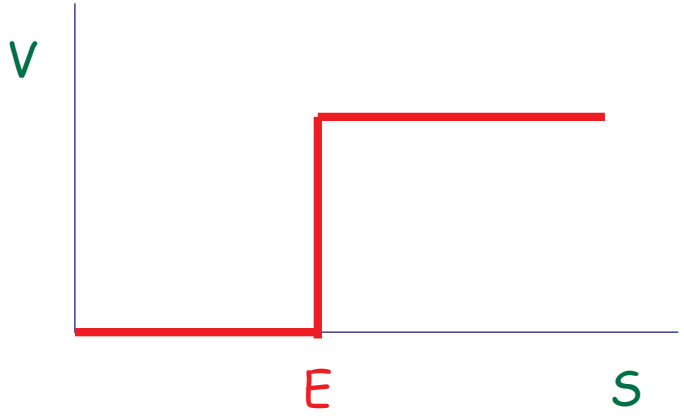
∞ European Call : $V(S,T) = \max(S-E,0)$



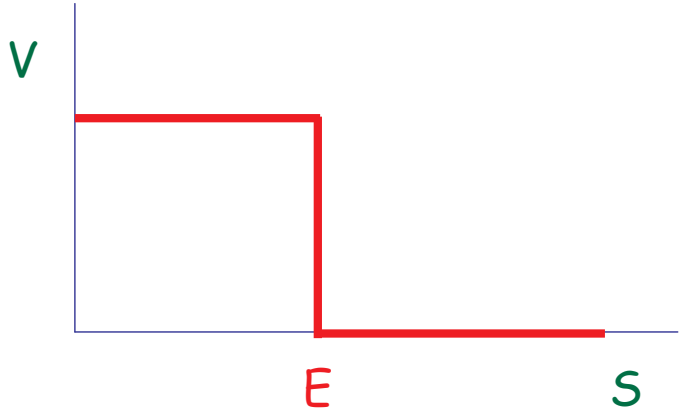
∞ European Put: $V(S,T) = \max(E-S,0)$



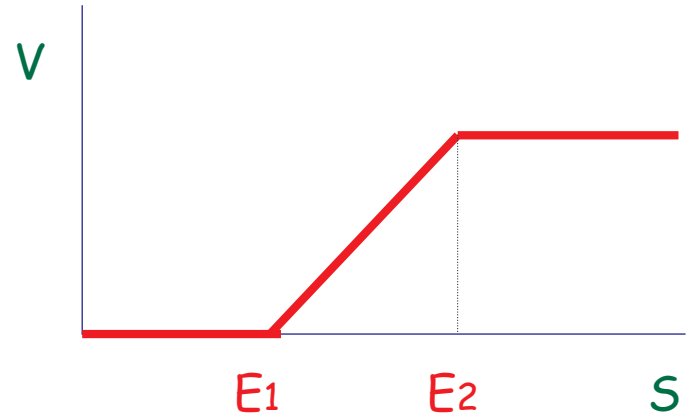
∞ Binary (or digital) Call : $V(S,T) = \Theta(S-E)$



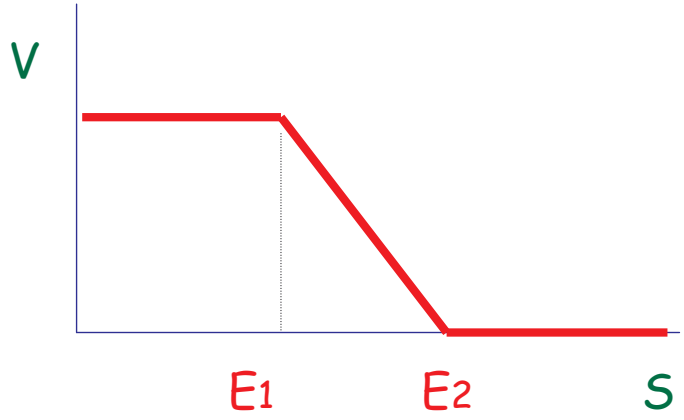
∞ Binary (or digital) Put : $V(S,T) = \Theta(E-S)$



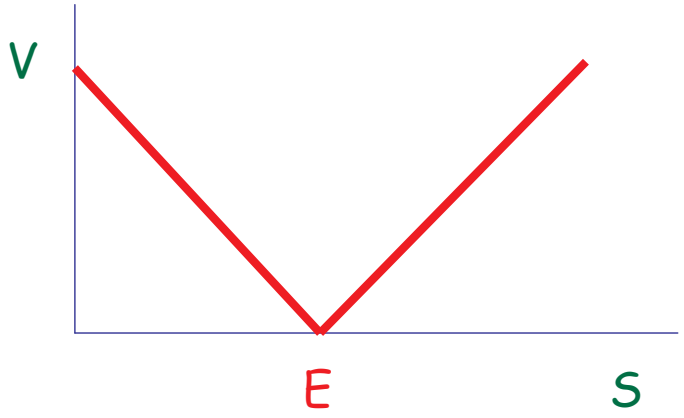
∞ Bull Spread : $V(S,T) = \max(S - E_1,0) - \max(S - E_2,0)$



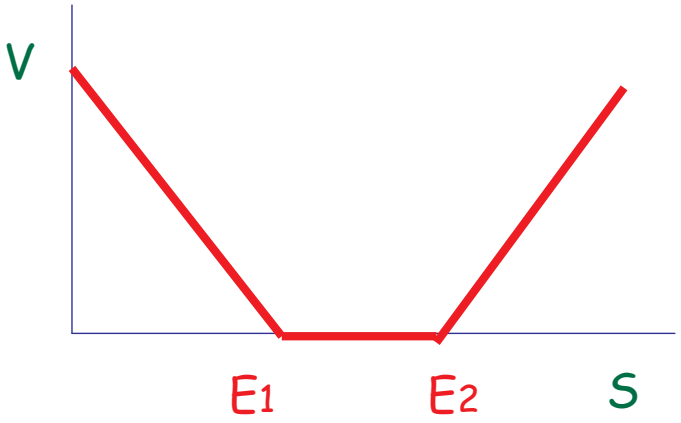
∞ Bear Spread : $V(S,T) = \max(E_1 - S,0) - \max(E_2 - S,0)$



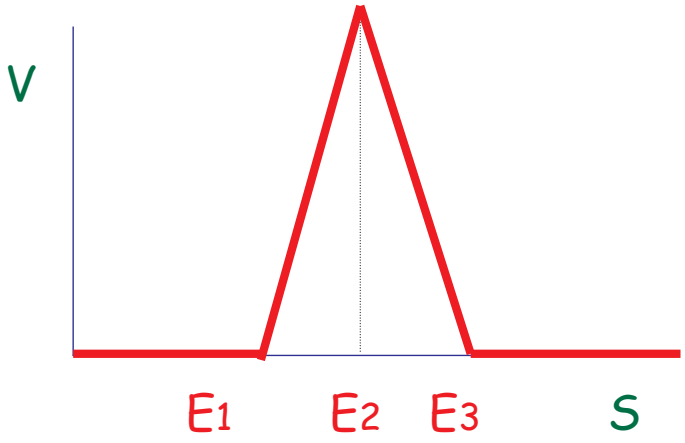
∞ Straddle : A call and a put with the same strike price.



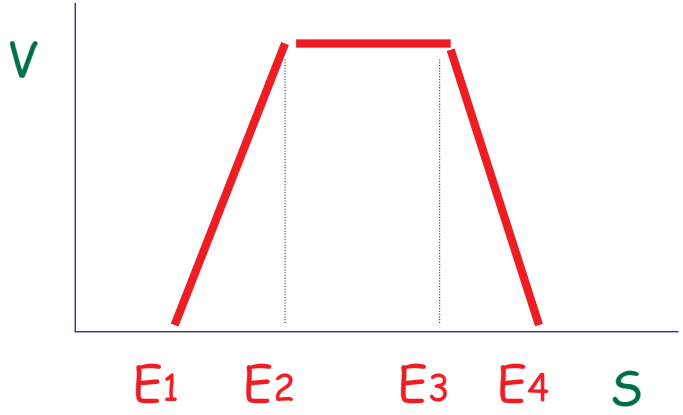
∞ Strangle: A call and a put with different strike prices.



∞ **Butterfly :** Involves the purchase and sale of options with three different expiries.



∞ **Condor:** like a butterfly except that for four strikes.



Transformation to Constant Coefficient Diffusion Equation

It can sometimes be useful to transform the basic Black-Scholes equation into something a little bit simpler by a change of variables. If we write:

$$V(S,t) = e^{\alpha x + \beta \tau} U(x,\tau)$$

where

$$\alpha = -\frac{1}{2}\left(\frac{2r}{\sigma^2} - 1\right), \quad \beta = -\frac{1}{4}\left(\frac{2r}{\sigma^2} + 1\right)^2, \quad S = e^x \text{ and } t = T - \frac{2\tau}{\sigma^2}$$

Then $U(x,\tau)$ satisfies the basic diffusion equation:

$$\frac{\partial U}{\partial t} = \frac{\partial^2 U}{\partial x^2}$$

Derivation of Black-Scholes Formulae

(However majority of contracts do not have explicit solutions)

$$\frac{\partial V}{\partial t}+\frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2}+rS\frac{\partial V}{\partial S}-rV=0$$

We will worry about the final conditions (which make the solution unique) later. For the moment concentrate on manipulating into something we can *easily* solve.

1. Change from present value to future value term $V(S,t)=e^{-r(T-t)}U(S,t)$:

$$\frac{\partial U}{\partial t}+\frac{1}{2}\sigma^2 S^2\frac{\partial^2 U}{\partial S^2}+rS\frac{\partial U}{\partial S}=0$$

2. As we are solving a backward equation: $\tau=T-t$

$$\frac{\partial U}{\partial \tau}=\frac{1}{2}\sigma^2 S^2\frac{\partial^2 U}{\partial S^2}+rS\frac{\partial U}{\partial S}$$

3. From the asset price to its return, as building up the stochastic differential equation (lognormality): $\xi=\log S$

$$\frac{\partial U}{\partial \tau}=\frac{1}{2}\sigma^2\frac{\partial^2 U}{\partial \xi^2}+(r-\frac{1}{2}\sigma^2)\frac{\partial U}{\partial \xi}$$

4. Translation of coordinate system: $x=\xi+(r-\frac{1}{2}\sigma^2)\tau$ and $U=W(x,\tau)$, like using the forward instead of the spot price.

$$\frac{\partial W}{\partial \tau}=\frac{1}{2}\sigma^2\frac{\partial^2 W}{\partial x^2}$$

The special solution of this equation is:

$$W(x,\tau)=\frac{1}{\sqrt{2\pi\tau}\sigma}e^{-\frac{(x-x')^2}{2\sigma^2\tau}}$$

For any x'
(Infinite solutions)

(Green Function)

In the limit $\tau\to 0$ this solution becomes a delta function $\delta(x-x')$ which has de following special property:

$$\lim_{\tau\to 0}\frac{1}{\sqrt{2\pi\tau}\sigma}\int_{-\infty}^{\infty}e^{-\frac{(x-x')^2}{2\sigma^2\tau}}g(x')dx'=g(x)$$

Now it is time to consider the payoff: $V(S,T)=\text{Payoff}(S)$, and in our new variables: $W(x,0)=\text{Payoff}(e^x)$.

Then I claim that the solution of this equation for $\tau > 0$ is:

$$W(x,\tau) = \frac{1}{\sqrt{2\pi\tau}\sigma}\int_{-\infty}^{\infty} e^{-\frac{(x-x')^2}{2\sigma^2\tau}}\,\text{Payoff}(e^{x'})dx'$$

Or retracing our steps:

$$V(S,t) = \frac{e^{-r(T-t)}}{\sqrt{2\pi(T-t)}\sigma}\int_{0}^{\infty} e^{-\frac{(\log(S/S')+(r-1/2\sigma^2)(T-t))^2}{2\sigma^2(T-t)}}\,\text{Payoff}(S')\frac{dS'}{S'}$$

For example,

∞ a call (european) option value is: $V_C(S,t) = SN(d_1) - Ee^{-r(T-t)}N(d_2)$, where

$$d_1 = \frac{\log(S/E)+(r+\frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}},\quad d_2 = \frac{\log(S/E)+(r-\frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}$$

and $N(d) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{d} e^{-\frac{1}{2}x^2}dx$ is the cumulative distribution function for a Normal distribution.

∞ Put (european) option value is: $V_P(S,t) = -SN(-d_1) + Ee^{-r(T-t)}N(-d_2)$.

The Greeks

∞ Delta: $\Delta = \frac{\partial V}{\partial S}$: *Delta hedging* means holding one of the option and short a quantity Δ of the underlying., which varies as S and t varies (*dynamic hedging*). If Δ is very big the approximation collapses.

∞ Gamma: $\Gamma = \frac{\partial^2 V}{\partial S^2}$: Since Γ is the sensitivity of the Δ to the underlying it is a measure of by how often a position must be rehedged in order to maintain a delta neutral position. The hedging requirement is decreased by a Γ-neutral strategy.

∞ Theta: $\Theta = \frac{\partial V}{\partial t}$: In the Black-Scholes equation contributes to ensure that a Δ hedged position earns the risk-free rate.

∞ Vega: $Vega = \frac{\partial V}{\partial \sigma}$: Is the sensitivity to volatility. It is different (it is not even e greek letter) as it is the derivate with respect a parameter (which is not known accurately) and not a variable. We can also Vega hedge to reduce sensitivity to volatility, which might be the major step reduce model risk.

∞ Rho: $\rho = \frac{\partial V}{\partial r}$: One often uses a whole term structure of interest rates.

Extensions of Black-Scholes Equation

I. Options on dividend-paying equities:

Let's assume that the asset receives a constant dividend yield, D. That is in a time dt each asset receives an amount of DSdt.

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

II. Currency Options:

In holding the foreign currency we receive interest at the foreign rate of interest, r_f.

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

III. Commodity Options:

The relevant feature of commodities is that they have a cost of carry. Let's introduce q as the fraction of the value of the commodity that goes to pay the cost of carry. This is just a negative dividend,

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

IV. Options on Futures:

The future price, F, of a non-dividend paying equity is related to the spot price by: $F=e^{r(T-t)}S$. We can easily change variables $V(S,t)=U(F,t)$ to get:

$$\frac{\partial U}{\partial t}+\frac{1}{2}\sigma^2 F^2\frac{\partial^2 U}{\partial F^2}-rU=0$$

V. Options on time-dependent parameters:

The parameters are known functions of time.

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

VI. Power Options:

An option with a payoff that depends on the asset price at expiry raised to some power, α. That is, if $P=S^\alpha$ we can write:

$$\frac{\partial V}{\partial t}+\frac{1}{2}\alpha^2\sigma^2 P^2\frac{\partial^2 V}{\partial P^2}+\alpha\left[\frac{1}{2}\sigma^2(\alpha-1)+r\right]P\frac{\partial V}{\partial P}-rV=0$$

VII. Two-factor Options:

When interest rates are also stochastic, $dr=u(r,t)dt+w(r,t)dX$, but not correlated to the asset, the value of the option is determined by:

$$\frac{\partial V}{\partial t}+\frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2}+\frac{1}{2}w^2\frac{\partial^2 V}{\partial r^2}+rS\frac{\partial V}{\partial S}+(u-\lambda w)\frac{\partial V}{\partial r}-r(t)V=0$$

VII. Early Exercise Options:

These options can be exercised early, prior to expiry (American, Bermudan, …). The same Black-Scholes equation is verified.

VIII. And a huge many other derivatives …

5. Finite-difference Representations for the Black-Scholes Equation

The extra complexity of implicit methods is out weighted by their superior stability properties.

Rarely we can find closed-form solutions for the values of options. Unless the problem is very simple indeed we are going to have to solve a partial differential equation numerically.

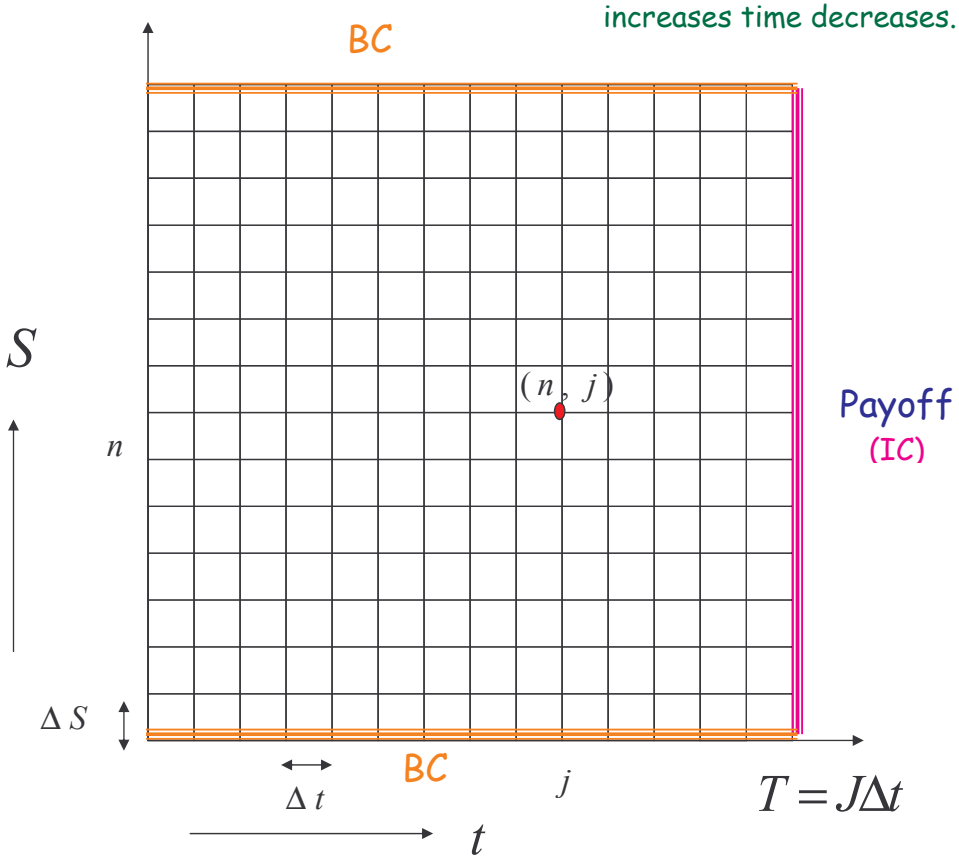
Willmott: "I would say that I use finite-difference methods about 75% of the time, Monte Carlo simulations 20%, and the rest would be explicit formulae. Those explicit formulae are almost always just the regular Black-Scholes formulae for calls and puts. "

Let us introduce some notation:

As Black-Scholes is to be solved in $0 \le S < \infty$, $N\Delta S$ will be our approximation to ∞.

$$S_n = n\Delta S,\ n = 0,1,\ldots,N$$
$$t_j = T - j\Delta t,\ j = 0,1,\ldots,J$$
$$V_{n,j} \equiv V(S_n, t_j)$$

I have changed the direction of time, as j increases time decreases.



The Black-Scholes equation is:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0$$

I am going to write this as to emphasize the wide applicability of the numerical method:

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

After taking explicit approximations to the derivatives:

$$\frac{V_{n,j+1} - V_{n,j}}{\Delta t} + a_{n,j}\left(\frac{V_{n+1,j} - 2V_{n,j} + V_{n-1,j}}{\Delta S^2}\right) + b_{n,j}\left(\frac{V_{n+1,j} - V_{n-1,j}}{2\Delta S}\right) + c_{n,j}V_{n,j} = \mathrm{O}(\Delta t, \Delta S^2)$$

I am going to rearrange this difference equation to put all of the j+1 terms on the left-hand side:

$$\begin{aligned} V_{n,j+1} &= (\nu_1 a_{n,j} - \frac{1}{2}\nu_2 b_{n,j})\, V_{n-1,j} \\ &+ (1 - 2\nu_1 a_{n,j} + \Delta t c_{n,j})\, V_{n,j} \\ &+ (\nu_1 a_{n,j} + \frac{1}{2}\nu_2 b_{n,j})\, V_{n+1,j} \\ &+ \mathrm{O}(\Delta t^2, \Delta t \Delta S^2) \end{aligned}$$

Local truncation error

where $\nu_1 = \frac{\Delta t}{\Delta S^2}$ and $\nu_2 = \frac{\Delta t}{\Delta S}$.

Which can also be written as:

$$\begin{aligned} V_{n,j+1} &= \frac{1}{2}(\sigma^2 n^2 - rn)\Delta t\, V_{n-1,j} \\ &+ \left[1 - (\sigma^2 n^2 + r)\Delta t\right] V_{n,j} \\ &+ \frac{1}{2}(\sigma^2 n^2 + rn)\Delta t\, V_{n+1,j} \\ &+ \mathrm{O}(\Delta t^2, \Delta t \Delta S^2) \end{aligned}$$

This equation holds for $n = 1,\ldots,N-1$ since $V_{-1,j}$ and $V_{N+1,j}$ are not defined. Thus there are $N-1$ equations for $N+1$ unknowns. The remaining two equations come from the two boundary conditions on n=0 and n=N (they are treated separately). I will give some examples:

Example 1

Suppose we want to price a call option. At S=0 we know that the value is 0, therefore: $V_{0,j} = 0, \forall j.$

Example 2

For large S the call value goes to $S_{\max} - Ee^{-r(T-t)}$. Thus our upper boundary condition could be: $V_{N,j} = N\Delta S - Ee^{-rj\Delta t}$.

Example 3

For a put option we have the condition at S=0 that $V(0,t) = Ee^{-r(T-t)}$, which becomes: $V_{0,j} = Ee^{-rj\Delta t}$.

Example 4

The put option becomes worthless for large S and so $V_{N,j} = 0.$

Example 5

A useful boundary condition to apply at S=0 for most contracts (including calls and puts) is that the diffusion and drift terms switch off. This means that on S=0 the payoff is guaranteed, resulting in the condition:

$$\frac{\partial V}{\partial t}(0,t) - rV(0,t) = 0$$

which numerically becomes: $V_{0,j} = (1 - r\Delta t)V_{0,j-1}$.

Example 6

When the option has a payoff that is almost linear in the underlying for large values of S then you an use the upper boundary condition

$$\frac{\partial^2 V}{\partial t^2}(S,t) \to 0 \text{ as } S \to \infty$$

Almost all common contracts have this property. The finite-difference representation is: $V_{N,j} = 2V_{N-1,j} - V_{N-2,j}$.

This is particularly useful since it is independent of the contract being valued.

Error and Stability

What about the error?

I can write the value of option at any *n* point at the final time step J as: $V_{n,J} = V_{n,0} + \sum_{j=0}^{J-1}(V_{n,j+1} - V_{n,j})$. Each of the terms in this summation has a local error of $O(\Delta t^2, \Delta t \Delta S^2)$, therefore, the global error in the final option value is $O(J\Delta t^2, J\Delta t \Delta S^2)$. If we value the option at a finite value of T then $J = O(\Delta t^{-1})$ so that the error in the final value option is $O(\Delta t, \Delta S^2)$.

What about the stability?

Trying solutions of the form $V_{n,j} = \xi^j(k)e^{ikn\Delta S}$ (von Neumann), we get:

$$\xi(k) = \left[1 + c_{n,j}\Delta t + 2a_{n,j}\nu_1(\cos k\Delta S - 1)\right] + ib_{n,j}\nu_2 \sin k\Delta S$$

It turns out that to have $|\xi(k)|<1$, for stability, we require (If we assume that all the coefficients are slowly varying over the ΔS scales):

$$c_{n,j} \le 0,$$
$$2\nu_1 a_{n,j} - \Delta t c_{n,j} \le 1 \text{ and}$$
$$(\nu_2 |b_{n,j}|)^2 \le 2\nu_1 a_{n,j}$$

∞ In finance the first constraint is almost always satisfied, very often is -r.

∞ Typically we chose ν_1 to be O(1) so that the second constraint is approximately: $\nu_1 < 1/2a_{n,j}$, which implies a serious limitation on the size of the time step:

Time step constraint

$$\Delta t \le \frac{\Delta S^2}{2a_{n,j}} = \frac{1}{\sigma^2 N^2}$$

Number of steps in S

∞ The third constraint can also be a serious restriction:

$$\Delta t \le \frac{2a_{n,j}}{(b_{n,j})^2} = \frac{\sigma^2}{r^2}$$

This restriction does not make much difference in practice unless the volatility is very small.

This last constraint can be avoided if we use a one-sided difference instead of a central difference for the first derivative of the option value with respect to the asset. Generally, the approximation could depend on the sign of b (upwind differencing):

$$\text{if } b(S,t) \ge 0 \quad \text{then} \quad b(S,t)\frac{\partial V}{\partial S} = b_{n+1/2,j}\frac{V_{n+1,j} - V_{n,j}}{\Delta S},$$

and

$$\text{if } b(S,t) \le 0 \quad \text{then} \quad b(S,t)\frac{\partial V}{\partial S} = b_{n-1/2,j}\frac{V_{n,j} - V_{n-1,j}}{\Delta S},$$

This method improves the stability but the numerical method is less accurate $O(\Delta S)$. To get back the $O(\Delta S^2)$ accuracy of the central difference with a one-sided difference you can use the following for:

∞ Forward difference:

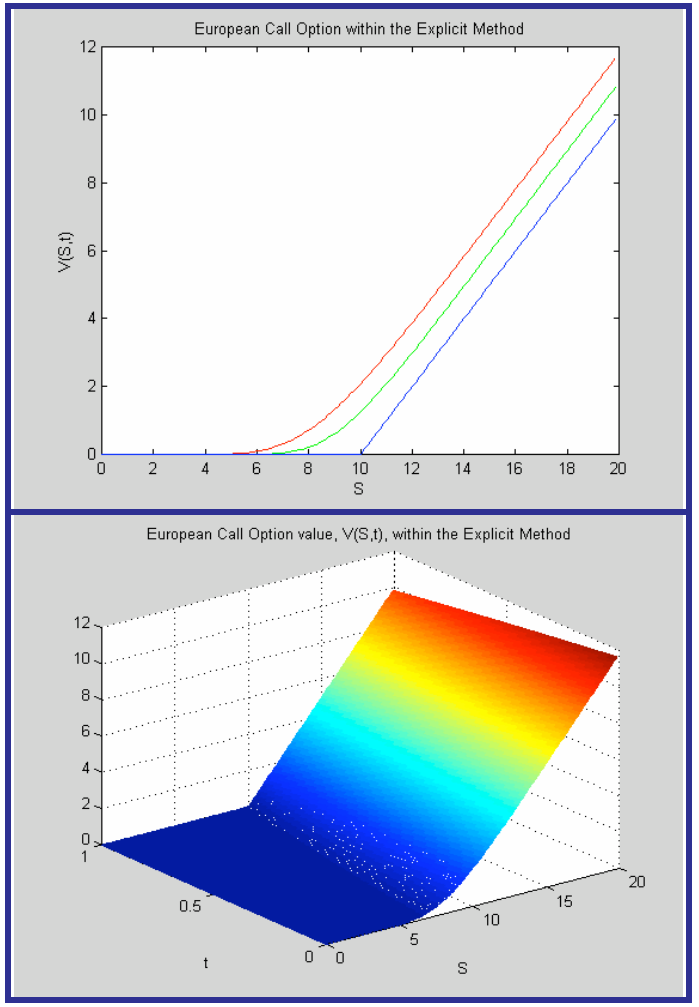
$$\frac{\partial V}{\partial S}(S,t) = \frac{-3V(S,t) + 4V(S+\Delta S,t) - V(S+2\Delta S,t)}{2\Delta S} + O(\Delta S^2)$$

∞ Backward difference:

$$\frac{\partial V}{\partial S}(S,t) = \frac{3V(S,t) - 4V(S-\Delta S,t) + V(S-2\Delta S,t)}{2\Delta S} + O(\Delta S^2)$$

```
% Matlab Program 9: Evaluates an European Call option by using an explicit
method
% Parameters of the problem:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r=0.2;              % Interest rate
sigma=0.25;         % Volatility of the underlying
M=1600;             % Number of time points
N=160;              % Number of share price points
Smax=20;            % Maximum share price considered
Smin=0;             % Minimum share price considered
T=1.;               % Maturation (expiry)of contract
E=10;               % Exercise price of the underlying
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dt=(T/M);           % Time step
ds=(Smax-Smin)/N; % Price step

% Initializing the matrix of the option value
v(1:N,1:M) = 0.0;

% Initial conditions prescribed by the European Call payoff at expiry:
V(S,T)=max(S-E,0)
v(1:N,1)=max((Smin+(0:N-1)*ds-E),zeros(size(1:N)))';

% Boundary conditions prescribed by the European Call:
v(1,2:M)=zeros(M-1,1)';                          % V(0,t)=0
v(N,2:M)=((N-1)*ds+Smin)-E*exp(-r*(1:M-1)*dt);  % V(S,t)=S-Eexp[-r(T-t)] as S ->
infinity

% Determining the matrix coeficients of the explicit algorithm
aa=0.5*dt*(sigma*sigma*(1:N-2).*(1:N-2)-r*(1:N-2))';
bb=1-dt*(sigma*sigma*(1:N-2).*(1:N-2)+r)';
cc=0.5*dt*(sigma*sigma*(1:N-2).*(1:N-2)+r*(1:N-2))';

% Implementing the explicit algorithm
for i=2:M,
v(2:N-1,i)=bb.*v(2:N-1,i-1)+cc.*v(3:N,i-1)+aa.*v(1:N-2,i-1);
end

% Reversal of the time components in the matrix as the solution of the Black-
Scholes
% equation was performed backwards
v=fliplr(v);

% Figure of the value of the option, V(S,t), as a function of S
% at three different times:t=0, T/2 and T (expiry).
figure(1)
plot(Smin+ds*(0:N-1),v(1:N,1)','r-',Smin+ds*(0:N-1),v(1:N,round(M/2))','g-
',Smin+ds*(0:N-1),v(1:N,M)','b-');
xlabel('S');
ylabel('V(S,t)');
title('European Call Option within the Explicit Method');

% Figure of the Value of the option, V(S,t)
figure(2)
mesh(Smin+ds*(0:N-1),dt*(0:M-1),v(1:N,1:M)')
title('European Call Option value, V(S,t), within the Explicit Method')
xlabel('S')
ylabel('t')
```


European Call Option with E=10€, T=1 year, r=0.2, and σ=0.25.



In the curves blue represents the value of the option at expiry, green half a year before that, and red one year before, that is, when the contract is signed (the price).

Implicit Methods

The extra complexity of implicit methods is out weighted by their superior stability properties.

A. Fully Implicit Method

$$\frac{V_{n,j+1}-V_{n,j}}{\Delta t} + a_{n,j+1}\left(\frac{V_{n+1,j+1}-2V_{n,j+1}+V_{n-1,j+1}}{\Delta S^2}\right) + b_{n,j+1}\left(\frac{V_{n+1,j+1}-V_{n-1,j+1}}{2\Delta S}\right) + c_{n,j+1}V_{n,j+1} = \mathrm{O}(\Delta t, \Delta S^2)$$

Actually, it does not matter much whether the coefficients *a, b* and *c* are evaluated at time step *j* or *j+1.*

Which can be written as:

$$V_{n,j} = (-\nu_1 a_{n,j+1} - \frac{1}{2}\nu_2 b_{n,j+1})V_{n-1,j+1} + (1 + 2\nu_1 a_{n,j+1} - \Delta t c_{n,j+1})V_{n,j+1} + (-\nu_1 a_{n,j+1} + \frac{1}{2}\nu_2 b_{n,j+1})V_{n+1,j+1} + \mathrm{O}(\Delta t^2, \Delta t \Delta S^2)$$

Local truncation error

where $\nu_1 = \frac{\Delta t}{\Delta S^2}$ and $\nu_2 = \frac{\Delta t}{\Delta S}$.

Again, this equation holds for $n = 1,\ldots,N-1$ since $V_{-1,j}$ and $V_{N+1,j}$ are not defined. Thus there are $N-1$ equations for $N+1$ unknowns. The remaining two equations come from the two boundary conditions at n=0 and n=N (they are treated separately). However, as we already know, there is a huge difference between this scheme and the explicit finite-difference scheme: stability (it is highly improved) and the solution procedure (it is no longer so straightforward).

This method can be significantly improved upon with little extra computation effort of the following scheme.

B. The Crank-Nicholson Method

An be considered as an average of the explicit and fully implicit methods (*it uses 6 points*!):

$$
\begin{aligned}
&\frac{V_{n,j+1}-V_{n,j}}{\Delta t} \\
&+\frac{a_{n,j+1}}{2}\left(\frac{V_{n+1,j+1}-2V_{n,j+1}+V_{n-1,j+1}}{\Delta S^2}\right) \\
&+\frac{a_{n,j}}{2}\left(\frac{V_{n+1,j}-2V_{n,j}+V_{n-1,j}}{\Delta S^2}\right) \\
&+\frac{b_{n,j+1}}{2}\left(\frac{V_{n+1,j+1}-V_{n-1,j+1}}{2\Delta S}\right) \\
&+\frac{b_{n,j}}{2}\left(\frac{V_{n+1,j}-V_{n-1,j}}{2\Delta S}\right) \\
&+\frac{1}{2}c_{n,j+1}V_{n,j+1}+\frac{1}{2}c_{n,j}V_{n,j}=\mathrm{O}(\Delta t^2,\Delta S^2)
\end{aligned}
$$

The Crank-Nicholson can be written as:

$$
\begin{aligned}
&A_{n,j+1}V_{n-1,j+1}+(1+B_{n,j+1})V_{n,j+1}+C_{n,j+1}V_{n+1,j+1}= \\
&-A_{n,j}V_{n-1,j}+(1-B_{n,j})V_{n,j}-C_{n,j}V_{n+1,j}
\end{aligned}
$$

with

$$
\begin{aligned}
A_{n,j} &= \frac{1}{2}\nu_1 a_{n,j}+\frac{1}{4}\nu_2 b_{n,j}\ , \\
B_{n,j} &= -\nu_1 a_{n,j}+\frac{1}{2}\Delta t c_{n,j}\ , \\
C_{n,j} &= \frac{1}{2}\nu_1 a_{n,j}-\frac{1}{4}\nu_2 b_{n,j}\ .
\end{aligned}
$$

These equations only holds for $1\le n\le N-1$ and the boundary conditions again supply the two missing equations.

The Crank-Nicholson method can be written in a matrix form:

$$\begin{pmatrix} A_{1,j+1} & 1+B_{1,j+1} & C_{1,j+1} & 0 & . & . & . \\ 0 & A_{2,j+1} & 1+B_{2,j+1} & C_{2,j+1} & . & . & . \\ . & 0 & . & . & . & . & . \\ . & . & . & . & 1+B_{N-2,j+1} & C_{N-2,j+1} & 0 \\ . & . & . & . & A_{N-1,j+1} & 1+B_{N-1,j+1} & C_{N-1,j+1} \end{pmatrix} \begin{pmatrix} V_{0,j+1} \\ V_{1,j+1} \\ . \\ . \\ . \\ V_{N-1,j+1} \\ V_{N,j+1} \end{pmatrix} =$$

$$\begin{pmatrix} -A_{1,j} & 1-B_{1,j} & -C_{1,j} & 0 & . & . & . \\ 0 & -A_{2,j} & 1-B_{2,j} & -C_{2,j} & . & . & . \\ . & 0 & . & . & . & . & . \\ . & . & . & . & 1-B_{N-2,j} & -C_{N-2,j} & 0 \\ . & . & . & . & -A_{N-1,j} & 1-B_{N-1,j} & -C_{N-1,j} \end{pmatrix} \begin{pmatrix} V_{0,j} \\ V_{1,j} \\ . \\ . \\ . \\ V_{N-1,j} \\ V_{N,j} \end{pmatrix}$$

The two matrices have *N-1* rows and *N+1* columns, which is a representations of the *N-1* equations and *N+1* unknowns. The two equations that we are missing come from the boundary conditions. Using these conditions, I am going to convert this system of equations into a system of equations involving a square matrix. The aim is to write a system of equations in the form:

$$\mathbf{M}_{j+1}^{L}\mathbf{V}_{j+1} + \mathbf{r}_{j+1}^{L} = \mathbf{M}_{j}^{R}\mathbf{V}_{j} + \mathbf{r}_{j}^{R}$$

For known square matrices $\mathbf{M}_{j+1}^{L}$ and $\mathbf{M}_{j}^{R}$, and known vectors $\mathbf{r}$, and where the details of the boundary conditions have been fully incorporated.

Example 1 of boundary condition: given $V_{0,j+1}$ and $V_{N,j+1}$:
Sometimes we know that the option has a particular value on the boundary, *n=0* and *n=N*, For example, if we have an European put we know that $V(0,t) = 0$ and $V(S_{\max},t) = S_{\max} - Ee^{-r(T-t)}$. This translates to knowing that $V_{0,j+1} = 0$ and $V_{N,j+1} = N\Delta S - Ee^{-r(j+1)\Delta t}$:

$$\begin{pmatrix} A_{1,j+1} & 1+B_{1,j+1} & C_{1,j+1} & 0 & . & . & . \\ 0 & A_{2,j+1} & 1+B_{2,j+1} & C_{2,j+1} & . & . & . \\ . & 0 & . & . & . & . & . \\ . & . & . & . & 1+B_{N-2,j+1} & C_{N-2,j+1} & 0 \\ . & . & . & . & A_{N-1,j+1} & 1+B_{N-1,j+1} & C_{N-1,j+1} \end{pmatrix} \begin{pmatrix} V_{0,j+1} \\ V_{1,j+1} \\ . \\ . \\ . \\ V_{N-1,j+1} \\ V_{N,j+1} \end{pmatrix}$$

as

$$\begin{pmatrix} 1+B_{1,j+1} & C_{1,j+1} & 0 & . & . & . \\ A_{2,j+1} & 1+B_{2,j+1} & . & . & . & . \\ 0 & . & . & . & 0 & . \\ . & . & . & . & 1+B_{N-2,j+1} & C_{N-2,j+1} \\ . & . & . & 0 & A_{N-1,j+1} & 1+B_{N-1,j+1} \end{pmatrix} \begin{pmatrix} V_{1,j+1} \\ . \\ . \\ . \\ . \\ V_{N-1,j+1} \end{pmatrix} + \begin{pmatrix} A_{1,j+1}V_{0,j+1} \\ 0 \\ . \\ . \\ . \\ C_{N-1,j+1}V_{N,j+1} \end{pmatrix}$$

$$= \mathbf{M}_{j+1}^{L}\mathbf{V}_{j+1} + \mathbf{r}_{j+1}^{L}$$

and similarly for matrices on the right.

Example 2 of boundary condition: given $\frac{\partial^2 V}{\partial S^2} = 0$: This condition is particularly useful since it is independent of the type of the contract, as long as the contract has a payoff that is almost linear in the underlying. This condition is, in central difference form:

$$V_{0,j+1} = 2V_{1,j+1} - V_{2,j+1} \quad \text{and} \quad V_{N,j+1} = 2V_{N-1,j+1} - V_{N-2,j+1}$$

Thus, we can write the left hand side as:

$$\begin{pmatrix} 1+B_{1,j+1}+2A_{1,j+1} & C_{1,j+1}-A_{1,j+1} & 0 & . & . & . \\ A_{2,j+1} & 1+B_{2,j+1} & . & . & . & . \\ 0 & . & . & . & 0 & . \\ . & . & . & . & 1+B_{N-2,j+1} & C_{N-2,j+1} \\ . & . & . & 0 & A_{N-1,j+1}-C_{N-1,j+1} & 1+B_{N-1,j+1}+2C_{N-1,j+1} \end{pmatrix} \begin{pmatrix} V_{1,j+1} \\ \\ . \\ . \\ . \\ V_{N-1,j+1} \end{pmatrix}$$

and similarly for the right hand side.

The Crank-Nicholson scheme can be written again as:

$$\mathbf{M}^{L}_{j+1}\mathbf{V}_{j+1} = \mathbf{r}^{R}_{j} - \mathbf{r}^{L}_{j+1} + \mathbf{M}^{R}_{j}\mathbf{V}_{j}$$

where this time vectors $\mathbf{r} = 0$.

a. Matriz Inversión:

$$\mathbf{V}_{j+1} = (\mathrm{M}^{L}_{j+1})^{-1}(\mathbf{r}^{R}_{j} - \mathbf{r}^{L}_{j+1} + \mathbf{M}^{R}_{j}\mathbf{V}_{j})$$

However, matrix inversion is very time consuming and computationally very inefficient.

b. LU Decomposition

The matrix $\mathbf{M}^{L}_{j+1}$ is tridiagonal, and it is not hard to decompose into the product of two other matrices, one having nonzero elements along the diagonal and the subdiagonal ($\mathbf{L}$) and the other having non-zero elements along the diagonal and the superdiagonal ($\mathbf{U}$). So that, $\mathbf{M} = \mathbf{L}\,\mathbf{U}$:

$$\begin{pmatrix} 1+B_1 & C_1 & 0 & . & . & . \\ A_2 & 1+B_2 & C_2. & . & . & . \\ 0 & . & . & . & C_{N-3} & . \\ . & . & . & A_{N-2} & 1+B_{N-2} & C_{N-2} \\ . & . & . & 0 & A_{N-1} & 1+B_{N-1} \end{pmatrix} =$$

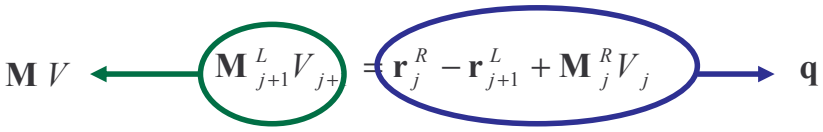
$$\begin{pmatrix} 1 & 0 & 0 & . & . & . \\ l_2 & 1 & 0 & . & . & . \\ 0 & l_3 & . & . & 0 & . \\ . & . & . & l_{N-2} & 1 & 0 \\ . & . & . & 0 & l_{N-1} & 1 \end{pmatrix} \begin{pmatrix} d_1 & u_1 & 0 & . & . & 0. \\ 0 & d_2 & u_2 & . & . & . \\ 0 & 0 & . & . & u_{N-3} & 0 \\ . & . & . & 0 & d_{N-2} & u_{N-2} \\ . & . & . & 0 & 0 & d_{N-1} \end{pmatrix}$$

The following relations are verified:

$$d_1 = 1 + B_1 ,$$
$$l_n d_{n-1} = A_n , \; u_{n-1} = C_{n-1} \text{ and } d_n = 1 + B_n - l_n u_{n-1} \text{ for } 2 \le n \le N-1$$

Notice that we work form n=1 to n=N sequentially.

Now we exploit the decomposition to solve:



$$\mathbf{M}\,\mathbf{V} = \mathbf{q}\,,\; \mathbf{L}\mathbf{U}\,\mathbf{V} = \mathbf{q}\,,\; \mathbf{L}\mathbf{w} = \mathbf{q}\,,\; \mathbf{U}\,\mathbf{V} = \mathbf{w}$$

Two steps more and we are done:

- The first step gives: $w_1 = q_1$ and $w_n = q_n - l_n w_{n-1}$ for $2 \le n \le N-1$, where we again must work sequentially.

- The second step involves working backwards from n=N-2 to n=1: $V_{N-1} = {w_{N-1}}/{d_{N-1}}$ and $V_n = \dfrac{w_n - p_n V_{n+1}}{d_n}$ for $N-2 \ge n \ge 1$

c. SOR (Successive Over-relaxation)

1. Jacobi method:

The system is easily solved iteratively using:

$$V_1^{i+1} = \frac{1}{M_{11}}\left[q_1 - (M_{12}N_2^i + \ldots + M_{1N}V_N^i)\right]$$
$$V_2^{i+1} = \frac{1}{M_{22}}\left[q_2 - (M_{21}V_1^i + \ldots + M_{2N}V_N^i)\right]$$
$$\ldots$$
$$V_N^{i+1} = \frac{1}{M_{NN}}\left[q_N - (M_{N1}V_1^i + M_{N2}V_2^i + \ldots)\right]$$

Where the superscript denotes the level of the iteration, which is started with some initial guess $\mathbf{V}^0$.

I can write the matrix $\mathbf{M}$ (tridiagonal) as the sum of a diagonal matrix $\mathbf{D}$, an upper triangular matrix $\mathbf{T}$ (with zeros in the diagonal) and a lower triangular matrix $\mathbf{L}$ (with zeros in the diagonal): $\mathbf{M} = \mathbf{D} + \mathbf{T} + \mathbf{L}$. Then:

$$\mathbf{V}^{i+1} = \mathbf{D}^{-1}\left[\mathbf{q} - (\mathbf{T} + \mathbf{L})\mathbf{V}^{i}\right]$$

2. Gauss-Seidel method: we use the updated values as soon as they are calculated:

$$V_n^{i+1} = \frac{1}{M_{nn}}\left[q_n - \sum_{j=1}^{n-1} M_{nj} V_j^{i+1} - \sum_{j=n}^{N} M_{nj} V_j^{i} \right]$$

3. Successive over-relaxed (SOR) method: iterate methods usually converge to the correct solution from one side (the correction $V_n^{i+1} - V_n^i$ stays on the same side of the sign as i increases). This is used by the SOR method to speed up the convergence. This method can be written as:

$$V_n^{i+1} = (1-\omega)V_n^i + \frac{\omega}{M_{nn}}\left[q_n - \sum_{j=1}^{n-1} M_{nj} V_j^{i+1} - \sum_{j=n}^{N} M_{nj} V_j^{i} \right]$$

Acceleration or
Over-relaxation parameter,
which must lie between 1 and 2
(We should find the optimum
value for w)

$$\mathbf{V}^{i+1} = (\mathbf{I} + \omega\mathbf{D}^{-1}\mathbf{L})^{-1}\left[\left((1-\omega)\mathbf{I} - \omega\mathbf{D}^{-1}\mathbf{T}\right)\mathbf{V}^{i} + \omega\mathbf{D}^{-1}\mathbf{q}\right]$$

6. Other Finite-difference Methods for the Black-Scholes Equation

Improving and extending explicit and implicit methods

The questions that arise with any new method are:

- What is the error in the method in terms of Δt and ΔS?
- What are the restrictions (stability) on the time step and/or asset step?
- Can I solve the resulting finite-difference equations quickly?
- Is the method flexible enough to cope with changes in coefficients, boundary conditions, etc…? That is, do you have to start from scratch if the contract changes slightly, or you can simply change a subroutine to cope with a new contract? Important example: American options.

Douglas Schemes

This is a method that manages to have a local truncation error of $O(\Delta S^4, \Delta t^2)$ for the same computational effort as the Crank-Nicholson scheme.

It may be expected that to get a higher order of accuracy than Crank-Nicholson would require the use of more points in the S direction: this is not always so !

For simplicity and clarity, I will describe the method using the basic-diffusion like equation (the extension to Black-Scholes becomes trivial):

$$\frac{\partial V}{\partial t} = \frac{\partial^2 V}{\partial S^2}$$

The explicit method applied to this equation is just:

$$\frac{V_{n,j+1} - V_{n,j}}{\Delta t} = \frac{V_{n+1,j} - 2V_{n,j} + V_{n-1,j}}{\Delta S^2}$$

and the fully implicit method is:

$$\frac{V_{n,j+1} - V_{n,j}}{\Delta t} = \frac{V_{n+1,j+1} - 2V_{n,j+1} + V_{n-1,j+1}}{\Delta S^2}$$

and, as we already know, the Crank-Nicholson method is just an average of the two methods. Is there any advantage of taking a weighted average (θ method)?

$$\frac{V_{n,j+1}-V_{n,j}}{\Delta t}=(1-\theta)\left(\frac{V_{n+1,j}-2V_{n,j}+V_{n-1,j}}{\Delta S^2}\right)+\theta\left(\frac{V_{n+1,j+1}-2V_{n,j+1}+V_{n-1,j+1}}{\Delta S^2}\right)$$

When $\theta=\frac{1}{2}$ we are back to the Crank-Nicholson method. For a general value of θ the local truncation error is:

$$O\left(\frac{1}{2}\Delta t-\frac{1}{12}\Delta S^2-\theta\Delta t,\Delta S^4,\Delta t^2\right)$$

For $\theta=0,\frac{1}{2}$ or 1 we get the results we have seen so far, but if $\theta=\frac{1}{2}-\frac{\Delta S^2}{12\Delta t}$ the local truncation is improved.

Three Time-level Methods

Numerical schemes are not restricted to the use of just two time levels if it gave us a better local truncation error or had better convergence properties. For simplicity, I shall still concentrate on the basic diffusion equation.
The obvious first method to try uses a central difference for the time derivative in an explicit scheme (Leap Frog):

$$\frac{V_{n,j+1}-V_{n,j-1}}{2\Delta t}=\frac{V_{n+1,j}-2V_{n,j}+V_{n-1,j}}{\Delta S^2}$$

However, this is unstable for any time step! However, an explicit scheme that is stable for all time steps is (Du-Fort Frankel):

$$\frac{V_{n,j+1}-V_{n,j-1}}{2\Delta t}=\frac{V_{n+1,j}-V_{n,j+1}-V_{n,j-1}+V_{n-1,j}}{\Delta S^2}$$

leading to:

$$(1+2\nu_1)V_{n,j+1}=2\nu_1(V_{n+1,j}+V_{n-1,j})+(1-2\nu_1)V_{n,j-1}$$

which, to get started, requires an initial condition and data at the first time level. The local error is $O\left[\Delta t^2,\Delta S^2,\left(\frac{\Delta t}{\Delta S}\right)^2\right]$, and, therefore, we should be aware (!!) of the relation between time and price steps.

Richardson Extrapolation

In the Explicit method the error is $O(\Delta S^2, \Delta t)$. If we assume that the approach to the correct solution as the time step and asset step tend to zero is in a sense "regular" (there is no guarantee that it is always the case, take care!) then we could postulate that (Taylor series):

$$\text{approx. sol.} = \text{exact. sol.} + \varepsilon_1 \Delta t + \varepsilon_2 \Delta S^2 + \varepsilon_3 \Delta t^2 + \cdots$$

Suppose that we have two approximate solutions (V_1, V_2) using different grid sizes with the same method:

$$V_1 = \text{exact. sol.} + \varepsilon_1 \Delta t_1 + \varepsilon_2 \Delta S_1^2 + \varepsilon_3 \Delta t_1^2 + \cdots =$$

$$= \text{exact. sol.} + \Delta S_1^2 \left(\varepsilon_1 \frac{\Delta t_1}{\Delta S_1^2} + \varepsilon_2 \right) + \cdots$$

and

$$V_2 = \text{exact sol.} + \varepsilon_1 \Delta t_2 + \varepsilon_2 \Delta S_2^2 + \varepsilon_3 \Delta t_2^2 + \cdots =$$

$$= \text{exact. sol.} + \Delta S_2^2 \left(\varepsilon_1 \frac{\Delta t_2}{\Delta S_2^2} + \varepsilon_2 \right) + \cdots$$

Then, if we choose:

$$\frac{\Delta t_1}{\Delta S_1^2} = \frac{\Delta t_2}{\Delta S_2^2}$$

we can find a better (more accurate) solution than both (V_1, V_2) by eliminating the leading-order error terms. This better approximation is given by:

$$V_{\text{NEW}} = \frac{\Delta S_2^2 V_1 - \Delta S_1^2 V_2}{\Delta S_2^2 - \Delta S_1^2}$$

Free Boundary Problems: American Options

The value of American options must always be greater than the payoff, otherwise there will be an arbitrage opportunity:

$$V(S,t) \geq \text{Payoff}(S)$$

The payoff function may also be time dependent. For example, if the option is Bermudan, i.e. exercise is only allowed on certain dates, then the payoff function is cero except on the special dates, when it is some prescribed function on the underlying. So I am going to write Payoff(S,t) and I need never mention Bermudan options again.

American options are examples of *free boundary problems*. We must solve a partial differential equation with an unknown boundary, the position of which is determined by having one more boundary condition than if the boundary were prescribed.

Early Exercise and the Explicit Method

Suppose that we have found $V_{n,j}$ for all *n* at the timestep *j*, proceed to find the option value at time j+1 by using the finite-difference scheme:

$$V_{n,j+1} = A_{n,j}\, V_{n-1,j} + (1 + B_{n,j})V_{n,j} + C_{\mathrm{n,j}}\, V_{n+1,j}$$

Do not worry about whether or not you have violated the American option constraint until you have found the option values $V_{n,j+1}$ for all *n*. Now let's check whether the new option values are greater or less than the payoff. If they are less than the payoff then we have arbitrage. We cannot allow that to happen so at every value of *n* for which the option value has allowed arbitrage, replace the value by the payoff at that asset value. That's all.

Early Exercise and Crank-Nicholson

Implementing the American constraint in the Crank-Nicholson is a bit harder but the rewards come in the accuracy. The only complication arises because the Crank-Nicholson is implicit, and every value of the option at *j+1* timestep is linked to every other value at that timestep. We can have two practical possibilities:

1. Like in the explicit method, replace the option value with the payoff (in case it is necessary) after the values at timestep j+1 have all been calculated. In this case, the accuracy of this method is then reduced to O(Δt).

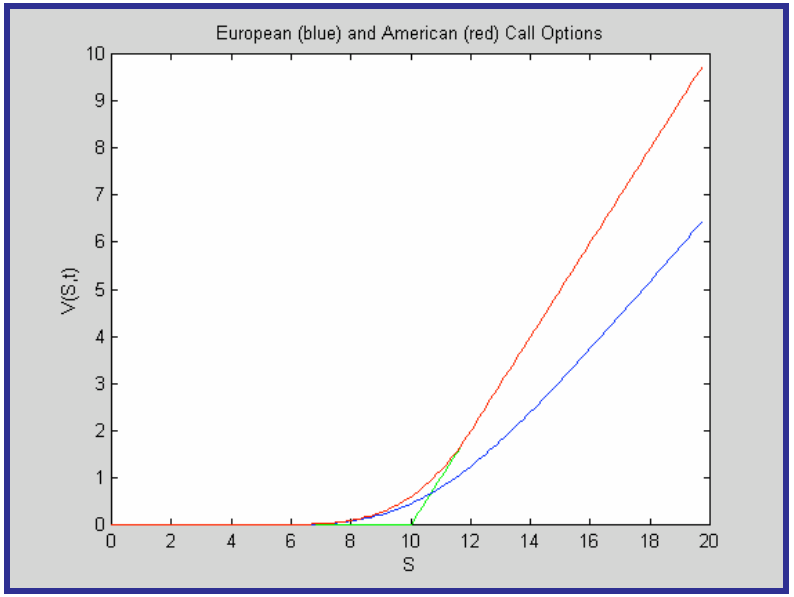
2. Replace the option values at the same time as they are found. For example in the SOR method:

$$V_n^{i+1} = \max\left((1-\omega)V_n^i + \frac{\omega}{M_{nn}}\left[q_n - \sum_{j=1}^{n-1} M_{nj} V_j^{i+1} - \sum_{j=n}^{N} M_{nj} V_j^i \right], Payoff \right)$$

The payoff is evaluated at n and j+1 in the obvious manner, this is called projected SOR.

```
% Matlab Program 10: Compares European and American Call options by using an
% explicit method Parameters of the problem:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r=0.2;              % Interest rate
sigma=0.25;         % Volatility of the underlying
d=0.3;              % Continuous dividend yield
M=1600;             % Number of time points
N=160;              % Number of share price points
Smax=20;            % Maximum share price considered
Smin=0;             % Minimum share price considered
T=1.;               % Maturation (expiry)of contract
E=10;               % Exercise price of the underlying
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dt=(T/M);           % Time step
ds=(Smax-Smin)/N; % Price step

% Initializing the matrix of the option values: v is the European and vam is the
American option
v(1:N,1:M) = 0.0;
vam(1:N,1:M) = 0.0;

% Initial conditions prescribed by the Call payoff at expiry: V(S,T)=max(E-S,0)
v(1:N,1)=max((Smin+(0:N-1)*ds-E),zeros(size(1:N)))';
vam(1:N,1)=max((Smin+(0:N-1)*ds-E),zeros(size(1:N)))';

% Boundary conditions prescribed by Call Options with dividends:
          % V(0,t)=0
v(1,2:M)=zeros(M-1,1)';
vam(1,2:M)=zeros(M-1,1)';
          % V(S,t)=Se^(-d*(T-t))-Ee^(-r(T-t)) as S -> infininty.
v(N,2:M)=((N-1)*ds+Smin)*exp(-d*(1:M-1)*dt)-E*exp(-r*(1:M-1)*dt);
vam(N,2:M)=((N-1)*ds+Smin)*exp(-d*(1:M-1)*dt)-E*exp(-r*(1:M-1)*dt);

% Determining the matrix coeficients of the explicit algorithm
aa=0.5*dt*(sigma*sigma*(1:N-2).*(1:N-2)-(r-d)*(1:N-2))';
bb=1-dt*(sigma*sigma*(1:N-2).*(1:N-2)+r)';
cc=0.5*dt*(sigma*sigma*(1:N-2).*(1:N-2)+(r-d)*(1:N-2))';

% Implementing the explicit algorithm
for i=2:M,
   v(2:N-1,i)=bb.*v(2:N-1,i-1)+cc.*v(3:N,i-1)+aa.*v(1:N-2,i-1);
   % Checks if early exercise is better for the American Option
   vam(2:N-1,i)=max(bb.*vam(2:N-1,i-1)+cc.*vam(3:N,i-1)+aa.*vam(1:N-2,i-
1),vam(2:N-1,1));
end

% Reversal of the time components in the matrix as the solution of the Black-
Scholes
% equation was performed backwards
v=fliplr(v);
vam=fliplr(vam);

% Compares the value today of the European (blue) and American (red) Calls,
V(S,t), as a function of S.
% The green curve represents the payoff at expiry.
plot(Smin+ds*(0:(N-2)),v(1:(N-1),M)','g-',Smin+ds*(0:(N-2)),v(1:(N-1),1)','b-
',Smin+ds*(0:(N-2)),vam(1:(N-1),1)','r-');
xlabel('S');
ylabel('V(S,t)');
title('European (blue) and American (red) Call Options');
```


European and American Call Options with E=10€, T=1 year, r=0.2, σ=0.25 and d=0.3 (dividend yield).



In the figure it is represented the **value today** (when the contract is signed, that is, one year before expiry) of an American (red) and European (blue) options. The green line represents the payoff at expiry.