Finite Difference Approach to Option Pricing

20 February 1998

CS522 Lab Note

1.0 Ordinary differential equation

An ordinary differential equation, or ODE, is an equation of the form

$$\frac{du}{dt} = f(u(t), t) \qquad \textbf{(1.1)}$$

where t is the time variable, u is a real or complex scalar or vector function of t, and f is a function.

Initial value problem is to find a differentiable function $u(t)$ such that

$$u(0) = u_0 \qquad \textbf{(1.2)}$$

$$\frac{du}{dt}(t) = f(u(t), t) \quad \text{for all } t \in [0, T]$$

For the solution of ordinary differential equations, one of the most powerful discretization strategies is linear multistep methods.

Let $k > 0$ be a real number, the time step, and let $t_0, t_1, t_2, \ldots.$ be defined $t_n = nk$. Our goal is to construct a sequence of values $v^0, v^1, \ldots.$ such that

$$v^n \approx u(t_n) \qquad n \geq 0 \qquad \textbf{(1.3)}$$

Let f^n be the abbreviation

$$f^n = f(v^n, t_n). \qquad \textbf{(1.4)}$$

A linear multistep method is a formula for calculating each new value v^{n+1} from some of the previous values $v^0, \ldots., v^n$ and $f^0, \ldots., f^n$. The simplest linear multistep method is a one step method : the Euler formula defined by

$$v^{n+1} = v^n + kf^n \qquad \textbf{(1.5)}$$

Euler method is an example of an explicit one-step formula.

A related linear multistep formula is the backward Euler, also a one-step formula, defined by

$$v^{n+1} = v^n + kf^{n+1} \qquad \textbf{(1.6)}$$

To implement an implicit formula, one must employ a scheme to solve for the unknown v^{n+1}, and this involves extra work.

The advantage of an implicit method is that in some situations it may be stable when an explicit one is catastrophically unstable.

Throughout the numerical solution of differential equations, there is a tradeoff between explicit methods, which tend to be easier to implement, and implicit ones, which tend ot be more stable.

Example - $\frac{du}{dt} = u, u(0) = 1$

$$v^{n+1} = v^n + kv^n, v^0 = 1 \quad \text{(Euler method)} \qquad \textbf{(1.7)}$$

$$v^{n+1} = v^n + kv^{n+1}, v^0 = 1 \quad \text{(Backward Euler method)} \qquad \textbf{(1.8)}$$

Example - More formulas

Trapezoid rule(implicit one-step formula)

$$v^{n+1} = v^n + \frac{k}{2}(f^n + f^{n+1}) \qquad \textbf{(1.9)}$$

Midpoint rule(explicit two-step formula)

$$v^{n+1} = v^{n-1} + 2kf^n \qquad \textbf{(1.10)}$$

2.0 Partial Differential Equation

Partial differential equations fall roughly into three great classes which can be loosely described as follows

elliptic -- time-dependent

parabolic -- time-dependent and diffusive

hyperbolic -- time-dependent and wave like

The simplest example of a hyperbolic equation is

$$\frac{\partial u}{\partial t} = \frac{\partial u}{\partial x} \qquad \textbf{(2.1)}$$

the one-dimensional first-order wave equation, which describes advection of a quantity $u(x, t)$ at the constant velocity -1.

The simplest example of a parabolic equation is

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \qquad \textbf{(2.2)}$$

the one-dimensional heat equation, which describes diffusion of a quantity such as heat or salinity.

Let $h > 0$ and $k > 0$ be a fixed space step and time step, respectively and set $x_j = jh$ and $t_n = nk$ for any integers j and n. The points (x_j, t_n) define a regular grid or mesh in two dimensions.

The aim of finite difference is to approximate continuous functions $u(x, t)$ by grid functions v_j^n,

$$v_j^n \approx u(x_j, t_n) \qquad \textbf{(2.3)}$$

v^n represents the spatial grid function $\{v_j^n, j \in Z\}$ for a fixed value n.

The simplest kind of finite procedure is an s-step finite difference formula, which is a fixed formula that prescribes v_j^{n+1} as a function of a finite number of other grid values at time steps $n + 1 - s$ through n (explicit case) or $n + 1$ (implicit case). To compute an approximation $\{v_j^n\}$ to $u(x, t)$, we shall begin with initial data v^0, ..., v^{s-1}, and compute values v^s, v^{s+1}, ... in succession by applying finite difference formula. This process is sometimes known as marching with respect to t.

Example

Finite difference approximations for the heat equation $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$, with $\sigma = \frac{k}{h^2}$.

Euler : $\quad v_j^{n+1} = v_j^n + \sigma(v_{j+1}^n - 2v_j^n + v_{j-1}^n)$

Backward Euler : $\quad v_j^{n+1} = v_j^n + \sigma(v_{j+1}^{n+1} - 2v_j^{n+1} + v_{j-1}^{n+1})$

Crank-Nicholson : $\quad v_j^{n+1} = v_j^n + \frac{1}{2}\sigma(v_{j+1}^n - 2v_j^n + v_{j-1}^n) + \frac{1}{2}\sigma(v_{j+1}^{n+1} - 2v_j^{n+1} + v_{j-1}^{n+1})$

(The above two sections are from unpublished manuscript of Lloyd N. Trefethen)

3.0 Option Pricing via Finite Difference Method

3.1 Implicit method(See lecture note for derivation and notation)

Let $f_{i,j}$ denote the value of option price at the (i, j) point, i.e., when $t = i\Delta t$ and $S = j\Delta S$

Implicit method for American put option is

$$a_j f_{i,j-1} + b_j f_{i,j} + c_j f_{i,j+1} = f_{i+1,j} \quad \text{for} \quad i = 0, 1, ..., N-1 \text{ and } j = 1, 2, ..., M-1 \qquad \textbf{(3.1)}$$

where

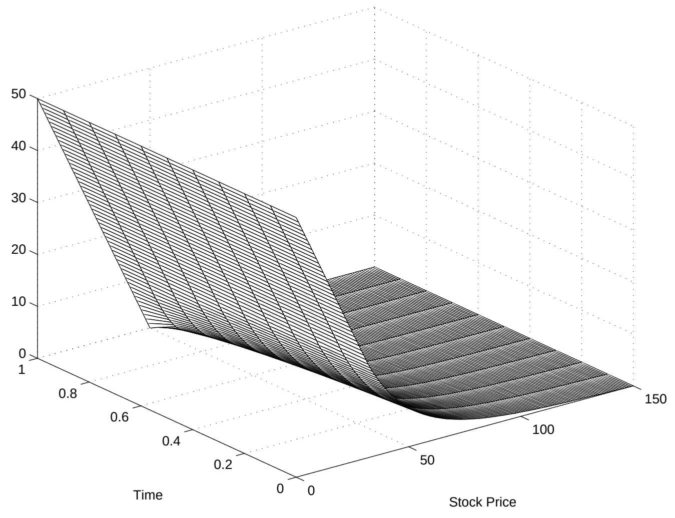
$$\begin{aligned} a_j &= \frac{1}{2}rj\Delta t - \frac{1}{2}\sigma^2 j^2 \Delta t \\ b_j &= 1 + \sigma^2 j^2 \Delta t + r\Delta t \\ c_j &= -\frac{1}{2}rj\Delta t - \frac{1}{2}\sigma^2 j^2 \Delta t \end{aligned} \qquad \textbf{(3.2)}$$

MATLAB Implementation

```
function put = imfdamput(Smax, dS, T, dT, X, R, SIG);
% put = imfdamput(Smax, dS, T, dT, X, R, SIG);
% Smax : maximum stock price
% dS : increment of stock price
% T  : maturity date
% dT : time step
% X  : exercise price
% R  : risk free interest rate
%
% reference : John C. Hull, Options, Futures, and Other Derivatives
%              3rd Ed., Chap 15

M = ceil(Smax/dS); ds = Smax / M;
N = ceil(T/dT);    dt = T / N;

J = 1:M-1;
a = .5*R*dt*J - .5*SIG^2*dt*J.^2;
b = 1 + SIG^2*dt*J.^2 + R*dt;
c = -.5*R*dt*J - .5*SIG^2*dt*J.^2;

A = diag(b) + diag(a(2:M-1), -1) + diag(c(1:M-2), 1);

put = zeros(N+1, M+1);
put(N+1, :) = max(X - [0:ds:Smax], 0);
put(:, 1) = X; put(:, M+1) = 0;

for i = N:-1:1
```

```
   y = put(i+1, 2:M)'; y(1) = y(1) - a(1)*X;
   put(i, 2:M) = [A \ y]';
   put(i, :) = max(X - [0:ds:Smax], put(i,:));

end
```

The following figure shows the American put option price when `Smax = 150; dS = 1; T = 1; dT = .1; X = 50; R = .1;` and `SIG = .5;`.



3.2 Explicit Method

The difference equation is

$$f_{i,j} = a^*_j f_{i+1,j-1} + b^*_j f_{i+1,j} + c^*_j f_{i+1,j+1} \quad \textbf{(3.3)}$$

where

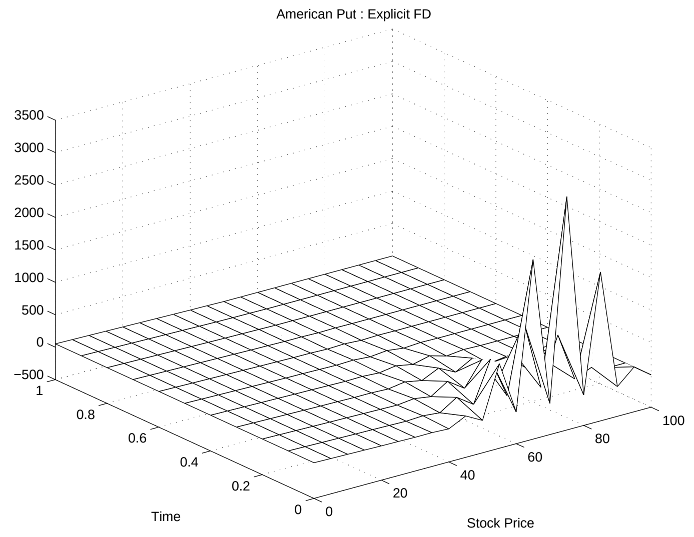
$$\begin{aligned} a^*_j &= \frac{1}{1+r\Delta t}\left(-\frac{1}{2}rj\Delta t + \frac{1}{2}\sigma^2 j^2 \Delta t\right) \\ b^*_j &= \frac{1}{1+r\Delta t}(1-\sigma^2 j^2 \Delta t) \\ c^*_j &= \frac{1}{1+r\Delta t}\left(\frac{1}{2}rj\Delta t + \frac{1}{2}\sigma^2 j^2 \Delta t\right) \end{aligned} \quad \textbf{(3.4)}$$

For the explicit method to be stable, $-\frac{1}{2}rj\Delta t + \frac{1}{2}\sigma^2 j^2 \Delta t$, $1-\sigma^2 j^2 \Delta t$, $\frac{1}{2}rj\Delta t + \frac{1}{2}\sigma^2 j^2 \Delta t$ should all be positive.

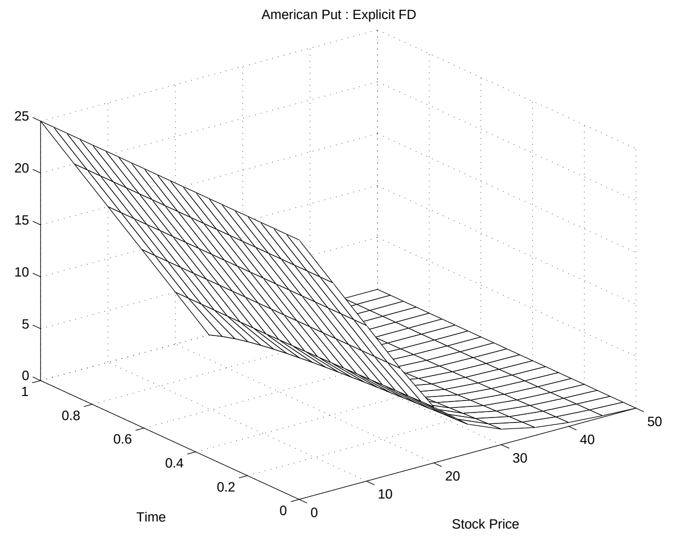
MATLAB Implementation

```
function put = exfdamput(Smax, dS, T, dT, X, R, SIG);
M = ceil(Smax/dS); ds = Smax / M;
N = ceil(T/dT);    dt = T / N;

J = 1:M-1;
a = (-.5*R*dt*J + .5*SIG^2*dt*J.^2) / (1+R*dt);
b = (1 - SIG^2*dt*J.^2) / (1+R*dt);
c = (.5*R*dt*J + .5*SIG^2*dt*J.^2) / (1 + R*dt);
A = diag(b) + diag(a(2:M-1), -1) + diag(c(1:M-2), 1);

put = zeros(N+1, M+1);
put(N+1, :) = max(X - [0:ds:Smax], 0);
put(:, 1) = X; put(:, M+1) = 0;

for i = N:-1:1

  y = zeros(1, M-1);
  y(1) = a(1)*put(i+1, 1); y(M-1) = c(M-1)*put(i+1,M+1);
  put(i, 2:M) = put(i+1, 2:M) * A' + y;
  put(i, :) = max(X - [0:ds:Smax], put(i,:));

end
```

The following figure shows the unstability of the explict method with the wrong choice of `Smax = 100; dS = 5; T = 1; dT = .1; R = .2; SIG = .4;` and `X = 50`.



It's stable when `Smax = 50; dS = 5; T = 1; dT = .1; X = 50; R = .1;` and `SIG = .5`.



3.3 Crank-Nicholson Method

The Crank-Nicholson scheme is an average of the explicit and implicit methods.

It's given by

$$g_{i,j} = f_{i,j} - a^*{}_j f_{i,j-1} - b^*{}_j f_{i,j} - c^*{}_j f_{i,j+1} \tag{3.5}$$

and

$$g_{i,j} = a_j f_{i-1,j-1} + b_j f_{i-1,j} + c_j f_{i-1,j+1} - f_{i-1,j} \tag{3.6}$$

The implementation of the Crank-Nicholson method is similar to that of the implicit method.

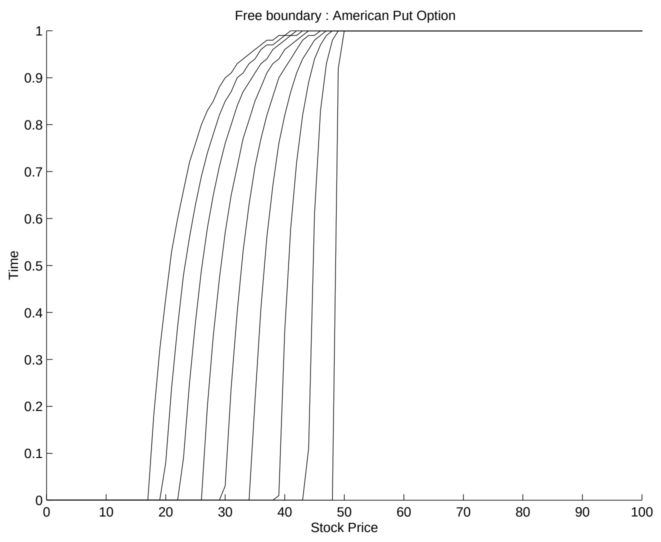
MATLAB Implementation

```
function put = cnfdamput(Smax, dS, T, dT, X, R, SIG);
M = ceil(Smax/dS); ds = Smax / M;
N = ceil(T/dT);    dt = T / N;

J = 1:M-1;
a = (-.5*R*dt*J + .5*SIG^2*dt*J.^2) / (1+R*dt);
b = (1 - SIG^2*dt*J.^2) / (1+R*dt);
c = (.5*R*dt*J + .5*SIG^2*dt*J.^2) / (1 + R*dt);
A = diag(1-b) + diag(-a(2:M-1), -1) + diag(-c(1:M-2), 1);

aa = .5*R*dt*J - .5*SIG^2*dt*J.^2;
bb = 1 + SIG^2*dt*J.^2 + R*dt;
cc = -.5*R*dt*J - .5*SIG^2*dt*J.^2;
B = diag(bb-1) + diag(aa(2:M-1), -1) + diag(cc(1:M-2), 1);
```

```
g = zeros(1, M-1);
put = zeros(N+1, M+1);
put(N+1, :) = max(X - [0:ds:Smax], 0);
put(:, 1) = X; put(:, M+1) = 0;

for i = N:-1:1

  g = put(i+1, 2:M) * A'; g(1) = g(1) - a(1)*X - aa(1)*X;
  put(i, 2:M) = [B \ g']';
  put(i, :) = max(X - [0:ds:Smax], put(i,:));

end
```

Using finite difference methods, we can get the boundary between the regions where it's optimal to exercise an American option or not.

The following figure shows different boundaries for `SIG = .1:.1:.9`. The first curve from the right is when `SIG = .1`.


Free boundary : American Put Option
Time
Stock Price