Option pricing and partial differential equations

Mohamed Ben Alaya

March 17th, 2014

1 Numerical illustrations

we will carry out some simulations on the Black & Scholes model by using different methods introduced in our lesson. Under risk-neutral probability, the price of the risky asset in the model is given by the SDE

$$dS_t = rS_t dt + \sigma S_t dW_t, \quad S_0 = s_0 > 0, , t \in [0, T]$$

with $(W_t)_{0 \leq t \leq T}$ is a standard Brownian motion. We denote by $(\mathcal{F}_t)_{0 \leq t \leq T}$ its canonical filtration.

- The SDE above has an explicit solution given by

$$S_t = s_0 exp\left((r - \frac{\sigma^2}{2})t + \sigma W_t\right).$$

We denote $(S_t^{0,s_0})_{0 \leq t \leq T}$ the solution starting at s_0, $S_0 = s_0$. An European option may be exercised only at the expiration date of the option T and it is well defined by the payoff $g(S_T)$ where g is a real valued function. The price of the option called the prime at time $t \in [0, T]$ is given by

$$\mathbb{E}\left(e^{-rT} g(S_T^{0,s_0})\right) = \mathbb{E}\left(f(W_T)|\mathcal{F}_t\right).$$

We are interested for computing

$$u(t, x) = \mathbb{E}\left(f(x + W_t)\right) \quad t \geq 0 \text{ et } x \in \mathbb{R}.$$

This function is solution to the heat equation

$$\begin{cases} \dfrac{\partial u}{\partial t} & = & \dfrac{1}{2}\dfrac{\partial^2 u}{\partial x^2} \quad \forall (t, x) \in [0, T] \times \mathbb{R} \\ u(0, x) & = & f(x). \end{cases}$$

Through this basic but important example, we will provide the different finite difference methods used to solve a parabolic partial differential equation. We will concentrate our efforts above all on the numerical issues.

- We recall that for european call option, the payoff $g(x) = (x-K)_+$ and the pricing of the option at time $t \in [0,T]$, is given by $V(t, S_t^{0,s_0})$ with

$$V(t,x) = xN(d_1) - Ke^{-r(T-t)}N(d_2),$$

where

$$d_1 = \frac{\log(x/K) + (r + \sigma^2/2)(T-t)}{\sigma\sqrt{T-t}} \quad et \quad d_2 = d_1 - \sigma\sqrt{T-t}$$

Similarly, for the european put option we have $V(t,x) = Ke^{-r(T-t)}N(-d_2) - xN(-d_1)$.

2 Introduction to finite difference methods

We consider a real function $u : \mathbb{R}_+ \times \mathbb{R} \longrightarrow \mathbb{R}$ solution to the heat equation

$$\begin{cases} \dfrac{\partial u}{\partial t} & = & \dfrac{1}{2}\dfrac{\partial^2 u}{\partial x^2} \quad \forall (t,x) \in \mathbb{R}_+ \times \mathbb{R} \\ u(0,x) & = & f(x). \end{cases}$$

To solve numerically the partial differential equation (PDE), we discretize the time variable with the time-step discretization $k = \; Deltat$ and we discretize the space variable with the step $h = \; Deltax$. We obtain a grid on $\mathbb{R}_+ \times \mathbb{R}$ given by points

$$(n\delta t, j\delta x) \text{ for } n \in \mathbb{Z} \text{ and } j \in \mathbb{N}.$$

We seek to approximate u on the grid by finding a sequence $U(n,j)$ such that

$$U(n,j) \approx u(n\delta t, j\delta x) \text{ for } n \in \mathbb{Z} \text{ and } j \in \mathbb{N}.$$

To do so, we approximate the partial derivative operators $\frac{\partial}{\partial t}$ by ∂_t and $\frac{\partial}{\partial x}$ by ∂_x or $\bar{\partial}_x$ with

$$\begin{cases} \partial_t U(n,j) & = & \dfrac{U(n+1,j) - U(n,j)}{\delta t} \\ \partial_x U(n,j) & = & \dfrac{U(n,j+1) - U(n,j)}{\delta x} \\ \bar{\partial}_x U(n,j) & = & \dfrac{U(n,j) - U(n,j-1)}{\delta x} \end{cases}$$

We approximate $\frac{\partial^2}{\partial x^2}$ by $\partial_x \bar{\partial}_x$ either

$$\partial_x \bar{\partial}_x U(n,j) = \frac{U(n,j+1) - 2U(n,j) + U(n,j-1)}{(\delta x)^2}$$

2.1 Explicit scheme

Considering these approximations, we obtain

$$\frac{U(n+1,j)-U(n,j)}{\delta t} = \frac{U(n,j+1)-2U(n,j)+U(n,j-1)}{2(\delta x)^2}.$$

The scheme is called explicit because $U(n+1,.)$ is computed directly from $U(n,.)$. Let $\lambda = \frac{k}{h^2} = \frac{\delta t}{(\delta x)^2}$, we have to compute at each step :

$$U(n+1,j) = \frac{\lambda}{2}U(n,j+1) + (1-\lambda)U(n,j) + \frac{\lambda}{2}U(n,j-1)$$

Remarks :

1. For $0 < \lambda \leq 1$, the scheme is stable according to the norm L^∞, in the sense that $||U(n+1,.)||_\infty \leq ||U(n,.)||_\infty$, for all $n \in \mathbb{N}$. We have also the convergence of the scheme, more precisely we have the scheme is conditionally convergent because the algorithm converges if h, k and k/h^2 tend to 0.
2. We can also restrict the problem to the bounded space $[0,T] \times [x_{min}, x_{max}]$. Les bornes des intervalles sont des paramètres à choisir soigneusement. We have to choose the interval carefully with somme boundary conditions $u(t, x_{min}) = g(t)$ and $u(t, x_{max}) = d(t)$.

We consider for example the following PDE

$$\begin{cases} \dfrac{\partial u}{\partial t} & = & \dfrac{1}{2}\dfrac{\partial^2 u}{\partial x^2} \quad \forall (t,x) \in [0,T] \times [x_{min}, x_{max}] \\ u(t,x_{min}) = 0 & \text{and} & u(t,x_{max}) = 0 \\ u(0,x) & = & f(x). \end{cases}$$

We proceed as follows : we put the time step $\delta t = T/N$ and the space step $\delta x = (x_{max} - x_{min})/(M+1)$, we consider the points

$$\begin{cases} x_0 & = & x_{min} \\ x_1 & = & x_{min} + \delta x \\ & \vdots & \\ x_j & = & x_{min} + j\delta x \\ & \vdots & \\ x_M & = & x_{min} + M\delta x \\ x_{M+1} & = & x_{max} \end{cases}$$

and we approximate

$$u(n\delta t, x_j) \approx U(n,j), \quad n \in \{0, \cdots, N\} \text{ et } j \in \{0, \cdots, M+1\},$$

with

$$U(n+1,j) = \frac{\lambda}{2}U(n,j+1) + (1-\lambda)U(n,j) + \frac{\lambda}{2}U(n,j-1), \ 1 \leq j \leq M.$$

We specify boundray conditions on x_{min} and x_{max} by $U(n+1,0) = 0$ et $U(n+1,M+1) = 0$.

Hence, if we put $U^n = \begin{pmatrix} U(n,1) \\ \vdots \\ U(n,M) \end{pmatrix}$ then

$$U^{n+1} = AU^n \quad \text{with} \quad A = \begin{pmatrix} a_2 & a_3 & 0 & 0 & 0 & 0 & 0 \\ a_1 & a_2 & a_3 & 0 & 0 & 0 & 0 \\ 0 & a_1 & a_2 & a_3 & 0 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 & 0 \\ 0 & 0 & 0 & a_1 & a_2 & a_3 & 0 \\ 0 & 0 & 0 & 0 & a_1 & a_2 & a_3 \\ 0 & 0 & 0 & 0 & 0 & a_1 & a_2 \end{pmatrix}$$

$a_1 = a_3 = \frac{\lambda}{2}$ and $a_2 = 1 - \lambda$.

Exercice Test the explicit scheme developed above on the example of European call.

```
// TP 3
// The finite difference method for the heat equation
// We introduce our function
// The grid is uniform for time and space.

stacksize(1.e7);
clear
xbasc()

////////////////////////////
//Financial parameters
////////////////////////////

// payoff : call

T=1.;
r=0;
S0=100;
K=100;
sigma=0.05;

////////////////////////////
// Numerical parameters
////////////////////////////

//  Explicit scheme
M=1000; // number of space step discretization
```

```
N=1000; // number of time step discretization
//////////////////////
// Localisation
///////////////////////
Smax=5*K;
xmax=(log(Smax/S0)-(r-sigma**2/2)*T)/sigma;
xmin=-xmax;
dx=(xmax-xmin)/(M+1);
dt=T/N;
lambda=dt/(dx*dx);


/////////////////////////
// Matrices
/////////////////////////

disp('Definition de la matrice A');

// Diagonal matrices, lower diagonal, diagonal centered
// Upper diagonal.

A=diag(ones(1,M)*1-lambda)+diag(ones(1,M-1)*(lambda/2),1)+diag(ones(1,M-1)*(lambda/2),-1

//for i=1:M-1
//A(i,i)=1-lambda;
//A(i,i+1)=lambda/2;
//A(i+1,i)=lambda/2;
//end
//A(M,M)=1-lambda;

// initial condition
x=(xmin+dx:dx:xmax-dx)';
Unew=exp(-r*T)*max(0,S0*exp((r-sigma**2/2)*T+sigma*x)-K);


///////////////////////////////
// loop of time
///////////////////////////////

disp('loop of time...');

for i=1:M
Unew=A*Unew;
end
```

```
/////////////////////////////////
// Display prices
/////////////////////////////////


i=floor(xmax/dx);
P=Unew(i);
// We can  interpolate for the price S0
disp(P,'prix calculé')

// The exact price
d1=(1/(sigma*sqrt(T)))*(log(S0/K)+(r+sigma**2/2)*T);
d2=d1-sigma*sqrt(T);

// The European call option
  prix=S0*cdfnor("PQ",d1,0,1)-K*exp(-r*T)*cdfnor("PQ",d2,0,1);
//  The European put option
// prix=-(S0-K*exp(-r*T))+(S0*cdfnor("PQ",d1,0,1)-K*exp(-r*T)*cdfnor("PQ",d2,0,1));

disp(prix,'prix exact');
```

2.2 Implicit method

We take the previous discretization that we can write $\partial_t U_j^n = \frac{1}{2}\partial_x \bar{\partial}_x U_j^n$. For the implicit method we compute the right hand side of the relation at time $n+1$ and we can write $\partial_t U_j^n = \frac{1}{2}\partial_x \bar{\partial}_x U_j^{n+1}$. Hence we obtain

$$U_j^n = -\frac{\lambda}{2}U_{j+1}^{n+1} + (1+\lambda)U_j^{n+1} - \frac{\lambda}{2}U_{j-1}^{n+1}, \quad 1 \le j \le M, \quad \text{and } U_0^{n+1} = U_{M+1}^{n+1} = 0,$$

with $\lambda = \frac{\delta t}{(\delta x)^2}$. Equivalently we have

$$BU^{n+1} = U^n \quad \text{with} \quad B = \begin{pmatrix} b_2 & b_3 & 0 & 0 & 0 & 0 & 0 \\ b_1 & b_2 & b_3 & 0 & 0 & 0 & 0 \\ 0 & b_1 & b_2 & b_3 & 0 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 & 0 \\ 0 & 0 & 0 & b_1 & b_2 & b_3 & 0 \\ 0 & 0 & 0 & 0 & b_1 & b_2 & b_3 \\ 0 & 0 & 0 & 0 & 0 & b_1 & b_2 \end{pmatrix}$$

$b_1 = b_3 = -\frac{\lambda}{2}$ and $b_2 = 1+\lambda$.

Remarks :

1. We show that B is invertible, the scheme is stable according to the norm L^∞ for all $\lambda > 0$, we have also the convergence of the scheme, more precisely we have the scheme is unconditionally convergent because the algorithm converges if h, k and tend to 0.
2. The error is of order $O(h^2 + k)$ which suggests that we should take $k \approx h^2$.

Exercice Test the implicit scheme developed above on the example of European call.

```
// TP 3
// The finite difference method for the heat equation
// We introduce our function
// The grid is uniform for time and space.


stacksize(1.e7);
clear
xbasc()

///////////////////////////
// Financial parameters
///////////////////////////

// payoff : call

T=1.;
r=0;
S0=100;
K=100;
sigma=0.05;

///////////////////////////
// Numerical parameters
///////////////////////////

// Implicit scheme
M=1000; // number of space step discretization
N=1000; // number of time step discretization
Smax=5*K; // Localisation
xmax=(log(Smax/S0)-(r-sigma**2/2)*T)/sigma;
xmin=-xmax;
dx=(xmax-xmin)/(M+1);
dt=T/N;
lambda=dt/(dx*dx);
```

```
/////////////////////////
// Matrices
/////////////////////////

disp('Definition de la matrice A');

// Diagonal matrices, lower diagonal, diagonal centered
// Upper diagonal.

A=diag(ones(1,M)*1-lambda)+diag(ones(1,M-1)*(lambda/2),1)+diag(ones(1,M-1)*(lambda/2),-1

//for i=1:M-1
//A(i,i)=1-lambda;
//A(i,i+1)=lambda/2;
//A(i+1,i)=lambda/2;
//end
//A(M,M)=1-lambda;

// Initial  condition
x=(xmin+dx:dx:xmax-dx)';
Unew=exp(-r*T)*max(0,S0*exp((r-sigma**2/2)*T+sigma*x)-K);


///////////////////////////////
// Loop of time
///////////////////////////////

disp('Loop of time...');

for i=1:M
Unew=A*Unew;
end


///////////////////////////////
// Display price
///////////////////////////////


i=floor(xmax/dx);
P=Unew(i);
// We interpolate to have the  price for S0
disp(P,'prix calculé')
```

```
// Exact price
d1=(1/(sigma*sqrt(T)))*(log(S0/K)+(r+sigma**2/2)*T);
d2=d1-sigma*sqrt(T);

// European call option
  prix=S0*cdfnor("PQ",d1,0,1)-K*exp(-r*T)*cdfnor("PQ",d2,0,1);
// European put option
// prix=-(S0-K*exp(-r*T))+(S0*cdfnor("PQ",d1,0,1)-K*exp(-r*T)*cdfnor("PQ",d2,0,1));

disp(prix,'prix exact');
```

2.3 Crank Nicholson scheme

It is a mixture of the explicit and implicit discretizations, we consider

$$\partial_t U_j^n = \frac{1}{2}\left(\frac{1}{2}\partial_x\bar{\partial}_x U_j^n + \frac{1}{2}\partial_x\bar{\partial}_x U_j^{n+1}\right), \quad \text{pour } 1 \leq j \leq M, \quad \text{and } U_0^{n+1} = U_{M+1}^{n+1} = 0.$$

For $\lambda = \dfrac{\delta t}{(\delta x)^2}$, we can write

$$-\frac{\lambda}{4}U_{j+1}^{n+1} + (1+\frac{\lambda}{2})U_j^{n+1} - \frac{\lambda}{4}U_{j-1}^{n+1} = \frac{\lambda}{4}U_{j+1}^n + (1-\frac{\lambda}{2})U_j^n + \frac{\lambda}{4}U_{j-1}^n, \;\; 1 \leq j \leq M,$$

and $U_0^{n+1} = U_{M+1}^{n+1} = 0$. Let

$$B = \begin{pmatrix} b_2 & b_3 & 0 & 0 & 0 & 0 & 0 \\ b_1 & b_2 & b_3 & 0 & 0 & 0 & 0 \\ 0 & b_1 & b_2 & b_3 & 0 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 & 0 \\ 0 & 0 & 0 & b_1 & b_2 & b_3 & 0 \\ 0 & 0 & 0 & 0 & b_1 & b_2 & b_3 \\ 0 & 0 & 0 & 0 & 0 & b_1 & b_2 \end{pmatrix} \text{ and } A = \begin{pmatrix} a_2 & a_3 & 0 & 0 & 0 & 0 & 0 \\ a_1 & a_2 & a_3 & 0 & 0 & 0 & 0 \\ 0 & a_1 & a_2 & a_3 & 0 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 & 0 \\ 0 & 0 & 0 & a_1 & a_2 & a_3 & 0 \\ 0 & 0 & 0 & 0 & a_1 & a_2 & a_3 \\ 0 & 0 & 0 & 0 & 0 & a_1 & a_2 \end{pmatrix}$$

with $b_1 = b_3 = -\frac{\lambda}{4}$, $b_2 = 1 + \frac{\lambda}{2}$, $a_1 = a_3 = \frac{\lambda}{4}$ and $a_2 = 1 - \frac{\lambda}{2}$. Equivalently, we can write

$$BU^{n+1} = AU^n, \quad \text{or } U^{n+1} = B^{-1}AU^n.$$

Remarks :

1. We show that B is invertible, the scheme is stable according to the norm L^∞ for all $\lambda > 0$, we have also the convergence of the scheme, more precisely we have the scheme is unconditionally convergent because the algorithm converges if h, k and tend to 0.
2. The error is of order $O(h^2 + k^2)$ which suggests that we should take $k \approx h$. This scheme is usually chosen for this type of problem.

Exercice Test the Crank Nicholson scheme developed above on the example of European call.

```
/ TP 3
// The finite difference method for the heat equation
// We introduce our function
// The grid is uniform for time and space.


stacksize(1.e7);
clear
xbasc()

///////////////////////////
// Financial parameters
///////////////////////////



// payoff : call

T=1.;
r=0.1;
S0=100;
K=100;
sigma=0.1;

///////////////////////////
// Numerical parameters
///////////////////////////

// Crank Nicholson scheme
M=900; // number of space step discretization
N=900; // number of time step discretization
Smax=5*K; // localisation
xmax=(log(Smax/S0)-(r-sigma**2/2)*T)/sigma;
xmin=-xmax;
dx=(xmax-xmin)/(M+1);
dt=T/N;
lambda=dt/(dx*dx);


/////////////////////////
// Matrices
```

```
//////////////////////////

disp('Definition de la matrice A');

// Diagonal matrices, lower diagonal, diagonal centered
// Upper diagonal.


A=diag(ones(1,M)*1-lambda)+diag(ones(1,M-1)*(lambda/2),1)+diag(ones(1,M-1)*(lambda/2),-1

//for i=1:M-1
//A(i,i)=1-lambda;
//A(i,i+1)=lambda/2;
//A(i+1,i)=lambda/2;
//end
//A(M,M)=1-lambda;

//  Initial condition
x=(xmin+dx:dx:xmax-dx)';
Unew=exp(-r*T)*max(0,S0*exp((r-sigma**2/2)*T+sigma*x)-K);


///////////////////////////////
// loop of time
///////////////////////////////

disp('Boucle en temps...');

for i=1:M
Unew=A*Unew;
end


///////////////////////////////
// Display of price
///////////////////////////////


i=floor(xmax/dx);
P=Unew(i);
// We can intepolate to the price S0
disp(P,'prix calculé')

// exact pricz
```

```
d1=(1/(sigma*sqrt(T)))*(log(S0/K)+(r+sigma**2/2)*T);
d2=d1-sigma*sqrt(T);

// case 'call'
  prix=S0*cdfnor("PQ",d1,0,1)-K*exp(-r*T)*cdfnor("PQ",d2,0,1);
// case 'put'
// prix=-(S0-K*exp(-r*T))+(S0*cdfnor("PQ",d1,0,1)-K*exp(-r*T)*cdfnor("PQ",d2,0,1));

disp(prix,'prix exact');
```