Financial derivatives and PDE's
Lecture 15

Simone Calogero

February 11^{th}, 2021

Finite different solutions of PDE's

The finite difference methods are techniques to find (numerically) approximate solutions to ordinary differential equations (ODE's), stochastic differential equations (SDE's) and partial differential equations (PDE's).

They are based on the idea to replace the ordinary/partial derivatives with a finite difference quotient, e.g., $y'(x) \approx (y(x+h) - y(x))/h$. The various methods differ by the choice of the finite difference used in the approximation. We shall present a number of methods by examples.

ODE's

Consider the first order ODE

$$\frac{dy}{dt} = ay + bt, \quad y(0) = y_0, \quad t \in [0, T], \tag{1}$$

for some constants $a, b \in \mathbb{R}$ and $T > 0$. The solution is given by

$$y(t) = y_0 e^{at} + \frac{b}{a^2}(e^{at} - at - 1). \tag{2}$$

We shall apply three different finite difference methods to approximate the solution of (1).

In all cases we divide the time interval $[0, T]$ into a uniform partition,

$$0 = t_0 < t_1 < \cdots < t_n = T, \quad t_j = j\frac{T}{n},\ \Delta t = t_{j+1} - t_j = \frac{T}{n}$$

and define

$$y(t_j) = y_j, \quad j = 0, \ldots, n.$$

Forward Euler method

In this method we introduce the following approximation of dy/dt at time t:

$$\frac{dy}{dt}(t) = \frac{y(t + \Delta t) - y(t)}{\Delta t} + O(\Delta t),$$

i.e.,

$$y(t + \Delta t) = y(t) + \frac{dy}{dt}(t)\Delta t + O(\Delta t^2). \tag{3}$$

For Equation (1) this becomes

$$y(t + \Delta t) = y(t) + (ay(t) + bt)\Delta t + O(\Delta t^2).$$

Setting $t = t_j$, $\Delta t = T/n$, $t + \Delta t = t_j + T/n = t_{j+1}$ and neglecting second order terms we obtain

$$y_{j+1} = y_j + (ay_j + bt_j)\frac{T}{n}, \quad j = 0, \ldots, n - 1. \tag{4}$$

As y_0 is known, the previous iterative equation can be solved at any step j.

This method is called *explicit*, because the solution at the step $j + 1$ is given explicitly in terms of the solution at the step j.

It is a simple matter to implement this method numerically, for instance using the following Matlab function:

```
function [time,sol]=exampleODEexp(T,y0,n)
dt=T/n;
sol=zeros(1,n+1);
time=zeros(1,n+1);
a=1; b=1;
sol(1)=y0;
for j=2:n+1
sol(j)=sol(j-1)+(a*sol(j-1)+b*time(j-1))*dt;
time(j)=time(j-1)+dt;
end
```

Backward Euler method

This method consists in approximating dy/dt at time t as

$$\frac{dy}{dt}(t) = \frac{y(t) - y(t - \Delta t)}{\Delta t} + O(\Delta t),$$

hence

$$y(t + \Delta t) = y(t) + \frac{dy}{dt}(t + \Delta t)\Delta t + O(\Delta t^2). \tag{5}$$

The iterative equation for (1) now is

$$y_{j+1} = y_j + (ay_{j+1} + bt_{j+1})\frac{T}{n}, \quad j = 0, \ldots, n - 1. \tag{6}$$

This method is called implicit, because the solution at the step $j+1$ depends on the solution at both the step j *and* the step $j + 1$ itself.

Therefore implicit methods involve an extra computation, which is to find y_{j+1} in terms of y_j only.

For the present example this is a trivial step, as we have

$$y_{j+1} = \big(1 - \frac{aT}{n}\big)^{-1}\big(y_j + bt_{j+1}\frac{T}{n}\big), \tag{7}$$

provided $n \neq aT$. Here is a Matlab function implementing the backward Euler method for the ODE (1):

```
function [time,sol]=exampleODEimp(T,y0,n)
dt=T/n;
sol=zeros(1,n+1);
time=zeros(1,n+1);
a=1; b=1;
sol(1)=y0;
for j=2:n+1
time(j)=time(j-1)+dt;
sol(j)=1/(1-a*dt)*(sol(j-1)+b*time(j)*dt);
end
```

Central difference method

By a Taylor expansion,

$$y(t+\Delta t) = y(t) + \frac{dy}{dt}(t)\Delta t + \frac{1}{2}\frac{d^2y}{dt^2}(t)\Delta t^2 + O(\Delta t^3), \tag{8}$$

and replacing Δt with $-\Delta t$,

$$y(t-\Delta t) = y(t) - \frac{dy}{dt}(t)\Delta t + \frac{1}{2}\frac{d^2y}{dt^2}(t)\Delta t^2 + O(\Delta t^3). \tag{9}$$

Subtracting the two equations we obtain the following approximation for dy/dt at time t:

$$\frac{dy}{dt}(t) = \frac{y(t+\Delta t) - y(t-\Delta t)}{2\Delta t} + O(\Delta t^2),$$

which is called central difference approximation. Hence

$$y(t+\Delta t) = y(t-\Delta t) + 2\frac{dy}{dt}(t)\Delta t + O(\Delta t^3). \tag{10}$$

Note that, compared to (3) and (5), we have gained one order in accuracy.

The iterative equation for (1) becomes

$$y_{j+1} = y_{j-1} - 2(ay_j + bt_j)\frac{T}{n}, \quad j = 0, \ldots, n-1. \tag{11}$$

The first step $j = 0$ requires y_{-1}. This is fixed by the backward method

$$y_{-1} = y_0 - \frac{T}{n} a y_0, \tag{12}$$

which is (6) for $j = -1$.

A second order ODE

Consider the second order ODE for the harmonic oscillator:

$$\frac{d^2y}{dt^2} = -\omega^2 y, \quad y(0) = y_0, \quad \dot{y}(0) = \widetilde{y}_0. \tag{13}$$

The solution to this problem is given by

$$y(t) = y_0 \cos(\omega t) + \frac{\widetilde{y}_0}{\omega} \sin(\omega t). \tag{14}$$

One can define forward/backward/central difference approximations for second derivatives in a way similar as for first derivatives.

For instance, adding (8) and (9) we obtain the following central difference approximation for d^2y/dt^2 at time t:

$$\frac{d^2y}{dt^2}(t) = \frac{y(t + \Delta t) - 2y(t) + y(t - \Delta t)}{\Delta t^2} + O(\Delta t),$$

which leads to the following iterative equation for (13):

$$y_{j+1} = 2y_j - y_{j-1} - \left(\frac{T}{n}\right)^2 \omega^2 y_j, \quad j = 1, \ldots, n-1, \tag{15}$$

$$y_1 = y_0 + \widetilde{y}_0 \frac{T}{n}. \tag{16}$$

The approximate solution y_1 at the first node is computed using the forward method and the initial datum $\dot{y}(0) = \widetilde{y}_0$. The Matlab function solving this iteration is the following.

```
function [time,sol]=harmonic(w,T,y0,n)
dt=T/n;
sol=zeros(1,n+1);
time=zeros(1,n+1);
sol(1)=y0(1);
sol(2)=sol(1)+y0(2)*dt;
for j=3:n+1
sol(j)=2*sol(j-1)-sol(j-2)-dt^2*w^2*sol(j-1);
time(j)=time(j-1)+dt;
end
```

SDE's

The Euler method can be straightforwardly generalized to SDE's, see [**?**]. In this section we present briefly the so-called Euler-Maruyama method, which is the generalization to SDE's of the forward Euler method for ODE's.

Consider the SDE

$$dX(t) = \alpha(t, X(t))\, dt + \beta(t, X(t))\, dW(t), \tag{17}$$

where we assumed that the assumptions in Theorem **??** are satisfied. Given the uniform partition

$$0 = t_0 < t_1 < \cdots < t_n = T, \quad t_j = j\frac{T}{n},\ \Delta t = t_{j+1} - t_j = \frac{T}{n}$$

of the interval $[0, T]$, we define

$$X(t_j) = X_j, \quad j = 0, \ldots, n, \quad W_j = W(t_j).$$

Note that X_j, W_j are random variables and that

$$G_j = \frac{W_j - W_{j-1}}{\sqrt{\Delta t}}$$

are independent standard normal random variables for $j = 1, \ldots, n$.

The (explicit) finite difference approximation of (17) is

$$X_j = X_{j-1} + \alpha(t_{j-1}, X_{j-1})\frac{T}{n} + \beta(t_{j-1}, X_{j-1})\sqrt{\frac{T}{n}}\, G_j. \tag{18}$$

The following Matlab function applies the iterative equation (18) to the linear SDE (**??**), for which (18) becomes

$$X_j = X_{j-1} + (a + bX_{j-1})(T/n) + \gamma\sqrt{T/n}\, G_j, \quad X_0 = x_0,$$

where x_0 is the (constant) initial datum. The output `sol` contains one path of the stochastic process $X(t)$ along the time partition $\{t_0 = 0, t_1, \ldots, t_n = T\}$, that is, a path $(X_0 = x_0, X(t_1), \ldots, X(t_n) = X(T))$.

```
function [time,sol] = linearSDEexample(T,x0,n,a,b,gamma)
dt=T/n;
sol=zeros(1,n+1);
time=zeros(1,n+1);
G=randn(1,n);
sol(1)=x0;
for j=2:n+1
sol(j)=sol(j-1)+(a+b*sol(j-1))*dt+gamma*sqrt(dt)*G(j-1);
time(j)=time(j-1)+dt;
end
```

PDE's

In this section we present three finite difference methods to find approximate solutions to the one-dimensional heat equation

$$\partial_t u = \partial_x^2 u, \quad u(0,x) = u_0(x), \tag{19}$$

where u_0 is continuous.

We refer to t as the time variable and to x as the spatial variable, since this is what they typically represent in the applications of the heat equation.

As before, we let $t \in [0,T]$. As to the domain of the spatial variable x, we distinguish two cases

(i) x runs over the whole real line, i.e., $x \in (-\infty, \infty)$, and we are interested in finding an approximation to the solution $u \in C^{1,2}(\overline{\mathcal{D}_T})$.

(ii) x runs over a finite interval, say $x \in (x_{\min}, x_{\max})$, and we want to find an approximation of the solution $u \in C^{1,2}(\overline{\mathcal{D}})$, where $\mathcal{D} = (0,T) \times (x_{\min}, x_{\max})$, which satisfies the boundary conditions[1]

$$u(t, x_{\min}) = u_L(t), \quad u(t, x_{\max}) = u_r(t), \quad t \in [0,T],$$

for some given continuous functions u_L, u_R. We also require $u_L(0) = u_0(x_{\min})$, $u_R(0) = u_0(x_{\max})$, so that the solution is continuous on the boundary.

In fact, for numerical purposes, problem (i) is a special case of problem (ii), for the domain $(-\infty, \infty)$ must be approximated by $(-A, A)$ for $A >> 1$ when we solve problem (i) in a computer.

Note however that in the finite domain approximation of problem (i), *the boundary conditions at $x = \pm A$ cannot be prescribed freely!* Rather they have to be given by suitable approximations of the limit values at $x = \pm\infty$ of the solution to the heat equation on the real line.

By what we have just said we can focus on problem (ii). To simplify the discussion we assume that the domain of the x variable is given by $x \in (0, X)$ and we assign zero boundary conditions, i.e., $u_L = u_R = 0$. Hence we want to study the problem

[1] These are called Dirichlet type boundary conditions. Other types of boundary conditions can be imposed, but the Dirichlet type is sufficient for our forthcoming applications to financial problems.

$$\partial_t u = \partial_x^2 u, \quad (t,x) \in (0,T) \times (0,X), \tag{20a}$$
$$u(0,x) = u_0(x), \quad u(t,0) = u(t,X) = 0, \quad x \in [0,X],\ t \in [0,T]; \quad u_0(0) = u_0(X) = 0. \tag{20b}$$

We introduce the partition of the interval $(0,X)$ given by

$$0 = x_0 < x_1 < \cdots < x_m = X, \quad \Delta_j x = x_{j+1} - x_j, \quad j = 0, \ldots, m-1,$$

and the partition of the time interval $[0,T]$ given by

$$0 = t_0 < t_1 < \cdots < t_n = T, \quad t_i = i\frac{T}{n}, \quad \Delta t = t_{i+1} - t_i = \frac{T}{n}.$$

Note that we use a uniform partition for the time interval while the partition for the spatial domain is in general not uniform. Of course

$$\Delta_0 x + \Delta_1 x + \cdots + \Delta_{m-1} x = X$$

and the spatial partition is uniform if and only if

$$\Delta_0 x = \Delta_1 x = \cdots = \Delta_{m-1} x = \Delta x = \frac{X}{m}, \quad x_j = j\Delta x. \tag{21}$$

Using non-uniform partitions is important when one needs more accuracy in some region. For instance, when computing the price of options with the finite difference methods, a more refine partition is recommended in the "nearly at the money" region. In the rest of this section we assume that the spatial partition is uniform and leave the generalization to non-uniform partitions as an exercise.

We denote

$$u_{i,j} = u(t_i, x_j), \quad i = 0, \ldots, n, \quad j = 0, \ldots, m.$$

Hence $u_{i,j}$ is a $n+1 \times m+1$ matrix.

The i^{th} row contains the value of the approximate solution at each point of the spatial mesh at the fixed time t_i.

For instance, the zeroth row is the initial datum: $u_{0,j} = u_0(x_j)$, $i = 0, \dots m$.

The columns of the matrix $u_{i,j}$ contain the values of the approximate solution at one spatial point for different times.

For instance, the column $u_{i,0}$ are the values of the approximate solution at $x_0 = 0$ for different times t_i, while $u_{i,m}$ contains the values at $x_m = X$.

By the given boundary conditions we then have

$$u_{i,0} = u_{i,m} = 0, \quad i = 0, \dots, n.$$

We define (for a uniform spatial partition)

$$d = \frac{\Delta t}{\Delta x^2} = \frac{T}{X^2}\frac{m^2}{n}. \tag{22}$$

Method 1: Forward in time, centered in space

In this method we use a forward difference approximation for the time derivative and a centered difference approximation for the second spatial derivative:

$$\partial_t u(t,x) = \frac{u(t+\Delta t, x) - u(t,x)}{\Delta t} + O(\Delta t),$$

$$\partial_x^2 u(t,x) = \frac{u(t, x+\Delta x) - 2u(t,x) + u(t, x-\Delta x)}{\Delta x^2} + O(\Delta x).$$

We find

$$u(t+\Delta t, x) = u(t,x) + d(u(t, x+\Delta x) - 2u(t,x) + u(t, x-\Delta x)).$$

Hence we obtain the following iterative equation

$$u_{i+1,j} = u_{i,j} + d(u_{i,j+1} - 2u_{i,j} + u_{i,j-1}), \quad i = 0, \dots, n-1,\ j = 1, \dots, m-1, \tag{23}$$

where we recall that $u_{0,j} = u_0(x_j)$, $u_{i,0} = u_{i,m+1} = 0$, $i = 0, \dots, n$, $j = 0, \dots m$. Let

$$\mathbf{u_i} = (u_{i,1}\ u_{i,1}\ \ldots\ u_{i,m-1})^T$$

be the *column* vector containing the approximate solution at time t_i; note that we do not need to include $u_{i,0}$, $u_{i,m}$ in the vector $\mathbf{u}_i$, as these components are fixed equal to zero by the boundary conditions.

We can rewrite (23) in matrix form as follows:

$$\mathbf{u_{i+1}} = A(d)\mathbf{u_i}, \tag{24}$$

where $A(z)$ is the $(m-1)\times(m-1)$ matrix with non-zero entries given by

$$A_{k,k}(z) = 1-2z, \quad k = 1,\ldots,m-1, \qquad A_{q,q+1}(z) = A_{q+1,q}(z) = z, \quad q = 1,\ldots,m-2. \tag{25}$$

This method is completely explicit, as the solution at the time step $i+1$ is explicitly given in terms of the solution at the time step i.

A Matlab function solving the iteration (24) with the initial datum $u_0(x) = \exp(X^2/4) - \exp((x-X/2)^2)$ is the following.

```
function [time,space,sol]=heatexp(T,X,n,m)
dt=T/n; dx=X/m;
d=dt/dx^2;
sol=zeros(n+1,m+1);
time=zeros(1,n+1);
space=zeros(1,m+1);
for i=2:n+1
time(i)=time(i-1)+dt;
end
for j=2:m+1
space(j)=space(j-1)+dx;
end
for j=1:m+1
sol(1,j)=exp(X^2/4)-exp((space(j)-X/2)^2);
end
sol(:,1)=0; sol(:,m+1)=0;
A=zeros(m-1,m-1);
for k=1:m-1
A(k,k)=1-2*d;
```

```
end
for q=1:m-2
A(q,q+1)=d;
A(q+1,q)=d;
end
for i=2:n+1
sol(i,2:m)=sol(i-1,2:m)*transpose(A);
end
```

To visualize the result it is convenient to employ an animation which plots the approximate solution at each point on the spatial mesh for some increasing sequence of times in the partition $\{t_0, t_1, \ldots, t_n\}$. This visualization can be achieved with the following simple Matlab function:

```
function anim(r,F)
N=length(F(:,1));
Max=max(max(F));
for i=1:N
plot(r,F(i,:));
axis([0 1 0 Max]);
drawnow;
pause(0.01);
end
```

Upon running the command `anim(space,sol)`, the previous function will plot the approximate solutions at different increasing times.

Let us try the following: `[time,space,sol]=heatexp(1,1,2500,50)`. Hence we solve the problem on the unit square $(t, x) \in (0, 1)^2$ on a mesh of $(n, m) = 2500 \times 50$ points. The value of the parameter (22) is

$$d = 1.$$

If we now try to visualize the solution by running `anim(space,sol,0.1)`, we find that the approximate solution behaves very strangely (it produces just random oscillations). However by increasing the number of time steps with `[time,space,sol]=heatexp(1,1,5000,50)`, so that

$$d = 0.5,$$

and visualize the solution, we shall find that the approximate solution converges quickly and smoothly to $u \equiv 0$, which is the equilibrium of our problem (i.e., the time independent solution of (20)). In fact, this is not a coincidence, as it can be shown that *the forward-centered method for the heat equation is unstable if* $d > 0.5$ *and stable for* $d \leq 0.5$. The term unstable here refers to the fact that numerical errors, due for instance to the truncation

and round-off of the initial datum on the spatial grid, will increase in time. On the other hand, stability of a finite difference method means that the error will remain small at all times. The stability condition $d \leq 0.5$ for the forward-centered method applied to the heat equation is very restrictive: it forces us to choose a very high number of points on the time partition. To avoid such a restriction, which could be very costly in terms of computation time, implicit methods are preferred, such as the one we present next.

Method 2: Backward in time, centered in space

In this method we employ the backward finite difference approximation for the time derivative and the central difference for the second spatial derivative (same as before). This results in the following iterative equation:

$$u_{i+1,j} = u_{i,j} + d(u_{i+1,j+1} - 2u_{i+1,j} + u_{i+1,j-1}), \quad i = 0, \ldots, n-1,\ j = 1, \ldots, m-1, \quad (26)$$

where we recall that $u_{0,j} = u_0(x_j)$, $u_{i,0} = u_{i,m+1} = 0$, $i = 0, \ldots, n$, $j = 0, \ldots m$. This method is implicit and we need therefore to solve for the solution at time $i+1$ in terms of the solution at time i. To this purpose we let, as before,

$$\mathbf{u_i} = (u_{i,1}\ u_{i,1}\ \ldots\ u_{i,m-1})^T$$

and rewrite (26) in matrix form as follows:

$$A(-d)\mathbf{u_{i+1}} = \mathbf{u_i}, \quad (27)$$

where $A(z)$ is the matrix with non-zero entries (25). The matrix A is invertible, hence we can express $\mathbf{u_{i+1}}$ in terms of $\mathbf{u_i}$ as

$$\mathbf{u_{i+1}} = A(-d)^{-1}\mathbf{u_i}. \quad (28)$$

This method is unconditionally stable, i.e., it is stable for all values of the parameter d. We can test this property by using the following Matlab function, which solves the iterative equation (28):

```
function [time,space,sol]=heatimp(T,X,n,m)
dt=T/n; dx=X/m;
d=dt/dx^2;
sol=zeros(n+1,m+1);
time=zeros(1,n+1);
space=zeros(1,m+1);
for i=2:n+1
time(i)=time(i-1)+dt;
end
```

```
for j=2:m+1
space(j)=space(j-1)+dx;
end
for j=1:m+1
sol(1,j)=exp(X^2/4)-exp((space(j)-X/2)^2);
end
sol(:,1)=0; sol(:,m+1)=0;
A=zeros(m-1,m-1);
for k=1:m-1
A(k,k)=1+2*d;
end
for q=1:m-2
A(q,q+1)=-d;
A(q+1,q)=-d;
end
for i=2:n+1
sol(i,2:m)=sol(i-1,2:m)*transpose(inv(A));
end
```

If we now run `[time,space,sol]=heatexp(1,1,500,50)`, for which $d = 5$, and visualize the solution we shall obtain that the approximate solution behaves smoothly as expected, indicating that the instability problem of the forward-centered method has been solved.

0.0.1 Method 3: The θ-method

This is an implicit method with higher order of accuracy than the backward-centered method. It is obtained by simply averaging between methods 1 and 2 above, as follows

$$u_{i+1,j} = \theta u^{\text{back}}_{i+1,j} + (1-\theta)u^{\text{forw}}_{i+1,j}, \quad \theta \in (0,1),$$

where the first term in the right hand side is computed with method 1 and the second term with method 2. Thus we obtain the following iterative equation

$$u_{i+1,j} = u_{i,j} + d[(1-\theta)(u_{i,j+1} - 2u_{i,j} + u_{i,j-1}) + \theta(u_{i+1,j+1} - 2u_{i+1,j} + u_{i+1,j-1})], \qquad (29)$$

or, in matrix form

$$A(-d\theta)\mathbf{u}_{i+1} = A(d(1-\theta))\mathbf{u}_i$$

Remark 1. Note carefully that the solution obtained with the θ-method is *not* the average of the solutions obtained with methods 1 and 2, but rather the solution obtained by averaging the two methods.

Remark 2. *For $\theta = 1/2$, the θ-method is also called* **Crank-Nicolson method**.