MAP586 project

1D PDE solver

2022-01-27

Introduction

The price of a European option satitisfies the *Black Scholes equation*

$$0 = \frac{\partial f}{\partial t}(s,t) + \frac{1}{2}\sigma^2 s^2 \frac{\partial^2 f}{\partial s^2}(s,t) + rs\frac{\partial f}{\partial s}(s,t) - rf(s,t)$$

with the final condition $f(s,T) = V(s)$ where $V(s)$ is the payout.

Applying the variable change $x = log(s)$ yields

$$\frac{\partial f}{\partial t}(x,t) = -\frac{1}{2}\sigma^2 \frac{\partial^2 f}{\partial x^2}(x,t) + (\frac{1}{2}\sigma^2 - r)\frac{\partial f}{\partial x}(x,t) + rf(x,t)$$

with the final condition $f(x,T) = V(x) = V(e^s)$.

A common method to solve this PDE is to use finite differences to approximate the derivatives and then solve the equation on a discrete mesh.

Finite difference method

We first define a mesh over space and time, usually the time is discretized such that the time step dt is 1 day. The boundaries are 0 (today) and T (the maturity of the option). The space mesh is centered to the log of the current spot ($log(s_0)$)and the boundaries are set to $log(s_0) \pm 5stddev = log(s_0) \pm 5\sigma\sqrt{T}$. Let's call N the number of points in space and dx the space step.

We use the following approximations for the derivatives:

- $\frac{\partial^2 f}{\partial x^2}(x_i,t_n) = \frac{f^n_{i+1} - 2f^n_i + f^n_{i-1}}{dx^2}$
- $\frac{\partial f}{\partial x}(x_i,t_n) = \frac{f^n_{i+1} - f^n_{i-1}}{2dx}$
- $\frac{\partial f}{\partial t}(x_i,t_n) = \frac{f^{n+1}_i - f^n_i}{dt}$

where $f^n_i = f(x_i,t_n)$.

Let's define

$$L^n_i = -\frac{1}{2}\sigma^2 \frac{f^n_{i+1} - 2f^n_i + f^n_{i-1}}{dx^2} + (\frac{1}{2}\sigma^2 - r)\frac{f^n_{i+1} - f^n_{i-1}}{2dx} + rf^n_i$$

the *Black Scholes equation* then becomes a system of linear equations

$$\frac{f^{n+1}_i - f^n_i}{dt} = L^{n+1}_i, i \in [1, N-1] \iff f^n_i = f^{n+1}_i - L^{n+1}_i dt, i \in [1, N-1]$$

Since we are solving a backward equation (given the payoff at maturity T, we want to compute the price of the option today), evaluating L_i at t^{n+1} yields an explicit scheme, i.e we can direclty compute f^n_i given the price at the previous iteration f^{n+1}_i. While this scheme converges fast, it is highly instable.

Another solution is to evaluate L_i at t^n, we get

$$\frac{f_i^{n+1} - f_i^n}{dt} = L_i^n, i \in [1, N-1] \iff f_i^n + L_i^n dt = f_i^{n+1}, i \in [1, N-1]$$

Here we need to invert a linear system to compute f_i^n. While this is unconditionnally stable, this is slow to converge.

To get the benefits from both schemes, the solution is to use a linear combination of them, this is called a theta scheme:

$$\frac{f_i^{n+1} - f_i^n}{dt} = \theta L_i^n + (1-\theta) L_i^{n+1}$$

The closer θ is to 1, the more stable the scheme is. The closer θ is to 0 the faster the scheme converges. $\theta = \frac{1}{2}$ gives the best convergence speed under unconditionnal stability and is thus usually chosen. It is known as the Crank-Nicholson scheme.

Boundary conditions

At the boundaries of the space mesh ($i = 0$ and $i = N$) we cannot use centered finite difference

- $\frac{\partial^2 f}{\partial x^2}(x_0, t_n) = \frac{f_2^n - 2f_1^n + f_0^n}{dx^2}$
- $\frac{\partial f}{\partial x}(x_0, t_n) = \frac{f_1^n - f_0^n}{dx}$
- $\frac{\partial^2 f}{\partial x^2}(x_N, t_n) = \frac{f_N^n - 2f_{N-1}^n + f_{N-2}^n}{dx^2}$
- $\frac{\partial f}{\partial x}(x_N, t_n) = \frac{f_N^n - f_{N-1}^n}{dx}$

We then need to specify values of f at the boundaries (f_0 and f_N) to be able to solve the linear system. This is called Dirichlet boundary conditions. Another possibility is to impose the derivatives at the boundaries, these are the Neumann boundary conditions.

Work to be done

You have just joined the quantitative team of a bank, who needs to reimplement its PDE solver. Indeed, the first version was developed by a battalion of interns who were learning C++ during their break for lunch; as a surprising result, it is highly unstable and slow (but sometimes it gives a correct price). The villains have been punished: the interns have been fired and the manager responsible for this mess has been promoted, so at least you won't have to interact with him.

The new manager hired you for your expert skills in C++ and want you to fix the issue before mid January (or you will meet the same fate as the interns):

- Write the maths to get the exact expression of the linear system to solve
- Implement the solver in C++, keeping in mind the following constraints
  - the solver should be able to accept any kind of payoff
  - the solver must be flexible enough so the client can specify the boundary conditions
  - the solver must be flexible enough so the client can speficy different diffusion models (so σ and r might be more complicated than simple constants)
  - the client should be able to specify the mesh and the value of θ
- Test your implementation by rolling back the payoff of a call and comparing the result with the Back Scholes closed-form
- Add methods to compute the δ, γ and θ. These methods should add minimal overhead once the price is computed.
- Add a method to compute the vega.

The source code must be well documented and the repository of the project must contain several examples and tests. In the report, you must justify your choices and clearly explain the data structure and the algorithms.