Partial Differential Equations

QF607 Numerical Methods

Zhenke Guan
zhenkeguan@smu.edu.sg

Outline

- Derivation of Black-Scholes Partial Differential Equation
- Finite Difference Method
- Discretizing Spot Dimension
- ODE
- Discretizing Time Dimension

Section 1

Derivation of Black-Scholes Partial Differential Equation

Objectives

- We have derived the general Fokker Planck PDE that governs the risk neutral probability density function $p(x,t)$ for diffusion model $dX = A(X,t)dt + B(X,t)dW$:

$$\frac{\partial p(x,t)}{\partial t} = -\frac{\partial [p(x,t)a(x,t)]}{\partial x} + \frac{1}{2}\frac{\partial^2 [p(x,t)b^2(x,t)]}{\partial x^2}. \tag{1}$$

- And the Fokker Planck equation that helps us derive the local volatility surface for the local volatility model:

$$\frac{\partial p(s,t)}{\partial t} = -\frac{\partial [p(s,t)\mu(t)]}{\partial s} + \frac{1}{2}\frac{\partial^2 [s^2\sigma^2(s,t)p(s,t)]}{\partial s^2} \tag{2}$$

- The objective now is to be able to price derivative instruments using the local volatility model — analytic solution does not exist under local volatility model
- One way to price is through the partial differential equation that governs the price of the derivative instrument, denoted as $V(S,t)$ where S is spot and t is time

Objectives

- The local volatility model differs from Black-Scholes model only on that the parameters μ and σ are time and state dependent
- This difference does not affect the derivation of the partial differential equation because the two parameters are both deterministic
- We will derive Black-Scholes PDE first, and local volatility model's PDE will be straightforward
- When we say Black-Scholes PDE what we mean is the PDE that governs the derivative instrument's price $V(S, t)$ under Black-Scholes model

Black-Scholes PDE Derivation — Replicating Portfolio I

Under real world economy, the diffusion of a stock price S is:

$$\frac{dS}{S} = \bar{\mu}dt + \sigma dW, \quad \frac{dB}{B} = rdt \tag{3}$$

where S is the underlying asset and B is the money account, and $\bar{\mu}$ is the real world drift.

Constructing a replicating portfolio that holds δ amount of the asset and β amount of cash

$$\Pi = \delta S + \beta B \tag{4}$$

For the portfolio to replicate V, below equation needs to hold for all S and t

$$\Pi - V = 0 \tag{5}$$

Black-Scholes PDE Derivation — Replicating Portfolio II

so we can resolve β

$$\beta = \frac{V - \delta S}{B}$$

The increment of the replicating portfolio is

$$d\Pi = \delta(dS + qSdt) + \beta rBdt = \delta dS + (q\delta S + rV - r\delta S)dt$$

where q is the dividend yield or the foreign currency interest rate for FX. The increment of the derivative V is

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

Equating the two we can obtain the PDE that V needs to satisfy:

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

Note that we didn't mention risk neutrality at all.

Black-Scholes PDE Derivation - Through Risk Neutrality I

- Under risk neutral measure,

$$\frac{dS}{S} = (r - q)dt + \sigma dW, \quad \frac{dB}{B} = rdt \tag{6}$$

- The discounted value of the derivative, $\frac{V(t)}{B(t)}$, is a martingale under risk neutral measure.
- So the idea is to write down the diffusion and make it drift-less

Black-Scholes PDE Derivation - Through Risk Neutrality II

The diffusion of $\frac{V}{B}$:

$$d\left(\frac{V}{B}\right) = V \times d\left(\frac{1}{B}\right) + \frac{dV}{B} + \underbrace{dV \times d\left(\frac{1}{B}\right)}_{=0} \tag{7}$$

$$= V \times (-\frac{r}{B})dt + \frac{1}{B} \times \underbrace{(\frac{\partial V}{\partial t}dt + \frac{\partial V}{\partial S}dS + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}dS^2)}_{dV} \tag{8}$$

$$= \frac{-rV}{B}dt + \frac{1}{B}\underbrace{\left(\frac{\partial V}{\partial t}dt + \frac{\partial V}{\partial S}(S(r-q)dt + S\sigma dW) + \frac{\sigma^2 S^2}{2}\frac{\partial^2 V}{\partial S^2}dt\right)}_{dV} \tag{9}$$

$$= \frac{1}{B}\left(\underbrace{\frac{\partial V}{\partial t} + (r-q)S\frac{\partial V}{\partial S} + \frac{\sigma^2 S^2}{2}\frac{\partial^2 V}{\partial S^2} - rV}_{=0 \quad \leftarrow \textbf{BS PDE}}\right)dt + \frac{1}{B}\frac{\partial V}{\partial S}\sigma S dW \tag{10}$$

Verifying Black-Scholes Formula

It is easy to verify that the Black-Scholes formula is a solution to the Black-Scholes PDE:

$$V = Se^{-q(T-t)}N(d_+) - Ke^{-r(T-t)}N(d_-) \tag{11}$$

$$\frac{\partial V}{\partial S} = e^{-q(T-t)}N(d_+) \tag{12}$$

$$\frac{\partial^2 V}{\partial S^2} = \frac{e^{-q(T-t)}N'(d_+)}{\sigma S\sqrt{T-t}} = \frac{e^{-q(T-t)}\phi(d_+)}{\sigma S\sqrt{T-t}} \tag{13}$$

$$\frac{\partial V}{\partial t} = -\frac{Se^{-q(T-t)}\phi(d_+)\sigma}{2\sqrt{T}} - rKe^{-rT}N(d_-) + qSe^{-qt}N(d_+) \tag{14}$$

where $d_\pm = \frac{\ln\frac{F}{K} \pm \frac{1}{2}\sigma^2(T-t)}{\sigma\sqrt{T-t}}$, $F = Se^{(r-q)(T-t)}$.

Black-Scholes PDE and FPE (Fokker Planck Equation)

- It's worth putting the Black-Scholes PDE together with the Fokker Planck PDE:

$$\frac{\partial V}{\partial t} + (r-q)S\frac{\partial V}{\partial S} + \frac{\sigma^2 S^2}{2}\frac{\partial^2 V}{\partial S^2} - rV = 0 \tag{15}$$

$$\frac{\partial p}{\partial t} + (r-q)s\frac{\partial p}{\partial s} - \frac{\sigma^2 s^2}{2}\frac{\partial^2 p}{\partial s^2} + (r-q-\sigma^2)p = 0 \tag{16}$$

- Very similar in form
- When r and q become 0 the FPE becomes heat equation, BS PDE is inverse of heat equation
- BS PDE is backward and FPE is forward
- Second order in S, first order in t,
- Linear parabolic PDE (linear means the coefficients of the partial derivatives are not functions of the variables)
- Local volatility model for both cases only replace the σ by $\sigma(s,t)$

Section 2

Finite Difference Method

- To deal with the partial derivatives in Black-Scholes PDE, one popular way is to discretize it using **finite difference**.

- **Finite difference method**: a numerical method for solving differential equations by approximating them with difference equations.

Finite Difference — Order Of Convergence

- The definition of derivative is

$$\frac{df(x_0)}{dx} = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h} \tag{17}$$

  It's then natural to approximate the derivative using

$$\frac{df(x_0)}{dx} \approx \frac{f(x_0 + h) - f(x_0)}{h} \tag{18}$$

  for small h.
- The order of convergence can be analysed through Taylor expansion:

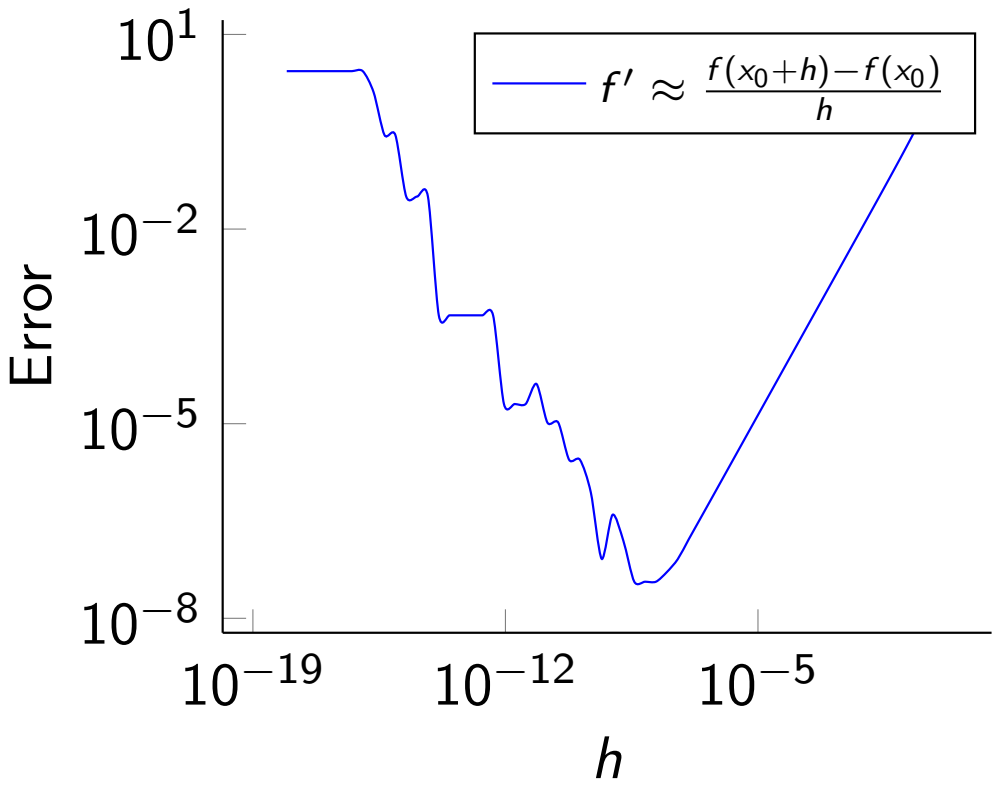
$$f(x_0 + h) = f(x_0) + f'(x_0)h + \frac{1}{2}f''(x_0)h^2 + O(h^3) \tag{19}$$

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} + O(h) \tag{20}$$

- The convergence of $f'(x_0) \approx \frac{f(x_0+h)-f(x_0)}{h}$ is thus $O(h)$
- The smaller h is, the more accurate the approximation is, but how small can h go?

Finite Difference — Errors

```
1  import math
2  import matplotlib.pyplot as plt
3
4  def testfinDiff():
5      x0 = 1
6      deriv = math.exp(x0)
7      h = 1
8      n = 60
9      hs = [h] * n
10     err = [0] * n
11     for i in range(0, n):
12         finDiff = (math.exp(x0+h) - math.
       exp(x0)) / h
13         hs[i] = h
14         h = h/2
15         err[i] = abs(finDiff - deriv)
16     plt.plot(hs, err, label="error")
17     plt.yscale('log')
18     plt.xscale('log')
19     plt.legend()
20     plt.show()
```



Finite difference error for $f = \exp(x)$

- The error does not reduce monotonically with h, it starts to increase at $h \approx 10^{-8}$

Type Of Numerical Errors Again

- Type of errors
  - Round-off error: computer represents numbers at limited precision, for `double` we can achieve machine precision at 10^{-16} to 10^{-15}.
  - Truncation error: the error due to the $O(h)$ term in the finite difference

- For $\frac{f(x_0+h)-f(x_0)}{h}$, round off error gets amplified by $\frac{1}{h}$, that's why we see at $h \approx 10^{-8}$, the error is around 10^{-7}

- Reducing h reduces the truncation error, but increases the round off error, so $h = 10^{-8}$ is the best we can achieve for the estimator $\frac{f(x_0+h)-f(x_0)}{h}$

Central Difference

- Just now we were using one-sided difference: using points $f(x_0)$ and $f(x_0 + h)$
- If we want to achieve higher order convergence, we can list the Taylor expansion at both $x_0 + h$ and $x_0 - h$:

$$f(x_0 + h) = f(x_0) + f'(x_0)h + \frac{1}{2}f''(x_0)h^2 + O(h^3) \tag{21}$$

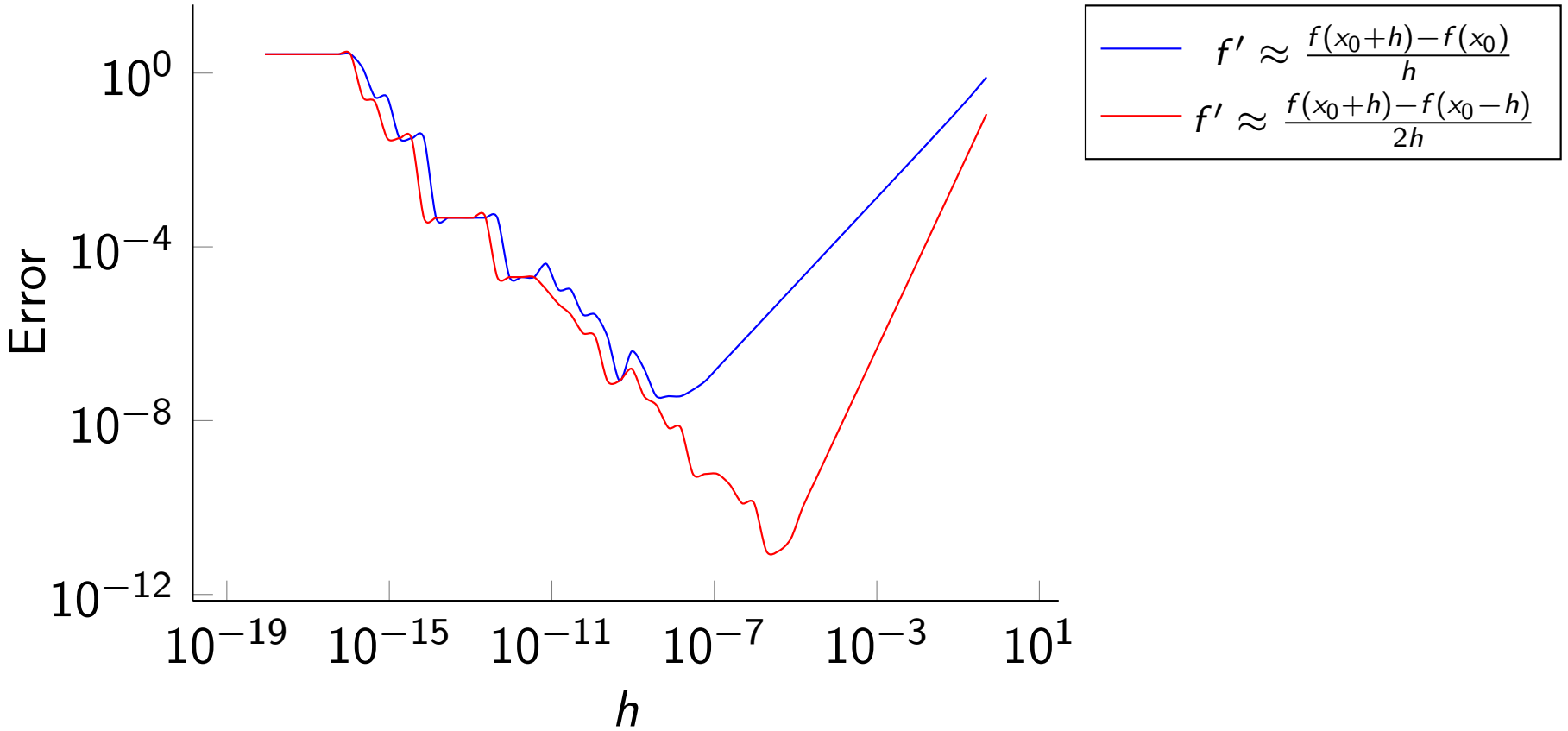
$$f(x_0 - h) = f(x_0) - f'(x_0)h + \frac{1}{2}f''(x_0)h^2 + O(h^3) \tag{22}$$

(21) minus (22) we can eliminate the h^2 term:

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} + O(h^2) \tag{23}$$

This is called **central difference**, whose order of convergence is second order in h.

Central Difference — Errors



Finite difference error for $f = \exp(x)$

- We see that central difference converges faster, and is able to achieve higher accuracy at around $h \approx 10^{-5}$ because when truncation error is small h is not too small such that round off error is still immaterial
- Also note that when h is less than 10^{-15} the error becomes flat — it's too small to be represented by `double` precision

Higher Order Finite Difference

In general, the more points we have, the more accurate we can achieve by listing the Taylor expansion and eliminate the low order terms. If we have four points at $x_0 + h_1$, $x_0 + h_2$, $x_0 + h_3$, $x_0 + h_4$, we can write down the Taylor expansion as

$$\begin{cases} c_1 f(x_0 + h_1) &= c_1[f(x_0) + f'(x_0)h_1 + \frac{1}{2}f''(x_0)h_1^2 + \frac{1}{6}f^{(3)}(x_0)h_1^3 + O(h_1^4)] \\ c_2 f(x_0 + h_2) &= c_2[f(x_0) + f'(x_0)h_2 + \frac{1}{2}f''(x_0)h_2^2 + \frac{1}{6}f^{(3)}(x_0)h_2^3 + O(h_2^4)] \\ c_3 f(x_0 + h_3) &= c_3[f(x_0) + f'(x_0)h_3 + \frac{1}{2}f''(x_0)h_3^2 + \frac{1}{6}f^{(3)}(x_0)h_3^3 + O(h_3^4)] \\ c_4 f(x_0 + h_4) &= c_4[f(x_0) + f'(x_0)h_4 + \frac{1}{2}f''(x_0)h_4^2 + \frac{1}{6}f^{(3)}(x_0)h_4^3 + O(h_4^4)] \end{cases} \tag{24}$$

It boils down to solving the linear system

$$\underbrace{\begin{pmatrix} 1 & 1 & 1 & 1 \\ h_1 & h_2 & h_3 & h_4 \\ \frac{h_1^2}{2} & \frac{h_2^2}{2} & \frac{h_3^2}{2} & \frac{h_4^2}{2} \\ \frac{h_1^3}{6} & \frac{h_2^3}{6} & \frac{h_3^3}{6} & \frac{h_4^3}{6} \end{pmatrix}}_{\mathbf{H}} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \tag{25}$$

Such that the sum of the equations in (24) give a equation for $f'(x_0)$:

$$f'(x_0) = \sum_{i=1}^{4} c_i f(x_0 + h_i) + O(\sum_{i=1}^{4} c_i h_i^4) \tag{26}$$

- The truncation error is of order $O(h^3)$ if all h_i are of the same magnitude: $\sum_{i=1}^{4} c_i h_i = 1$
- So in general with n points we can achieve truncation error at $O(h^{n-1})$.
- And we see that central difference is a spacial case: first and third order term degenerates and we achieve $O(h^2)$ with only two symmetric points.

Richardson Extrapolation

Richardson Extrapolation is built on a similar principal — manipulating the coefficients to eliminate the lower order terms. Suppose we have a function $g(h)$ that approximates the finite difference of $f(x)$, and $g(h)$ depends a step size h and is of order $O(h^n)$:

$$g(h) = f'(x_0) + Ch^n + O(h^{n+1}) \tag{27}$$

where C is a constant, we define a new estimator $r(h, k)$ as

$$r(h,k) = \frac{k^n g(h) - g(kh)}{k^n - 1} \quad \leftarrow \quad \textbf{Richardson Extrapolation} \tag{28}$$

$$= \frac{k^n(f'(x_0) + Ch^n + O(h^{n+1})) - (f'(x_0) + Ck^n h^n + O(h^{n+1}))}{k^n - 1} \tag{29}$$

$$= f'(x_0) + O(h^{n+1}) \tag{30}$$

Therefore, by evaluating twice the estimator with two step size, we can achieve a higher order estimator

We take the central difference for example,

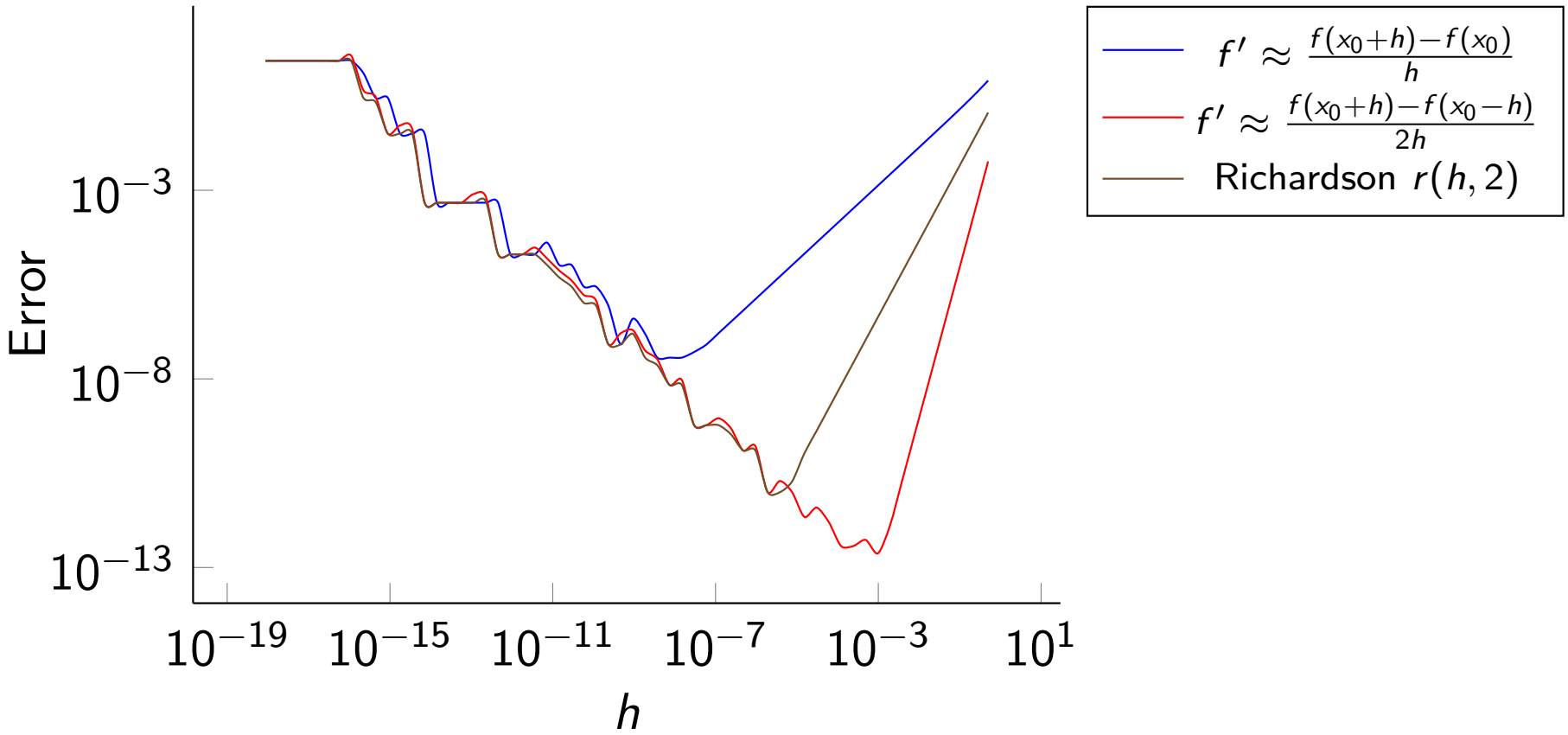
$$g(h) = \frac{f(x+h) - f(x-h)}{2h} \tag{31}$$

is order $O(h^2)$. We choose $k = 2$ we can have

$$r(h, 2) = \frac{4g(h) - g(2h)}{3} + O(h^3) \tag{32}$$

Effectively we need to evaluate 4 points: $\pm 2h, \pm h$ and the order is h^3 — same as Taylor expansion. It can be verified that the Taylor expansion method is generating the same formula if we go for the same four points.

Errors Of The Finite Difference Schemes



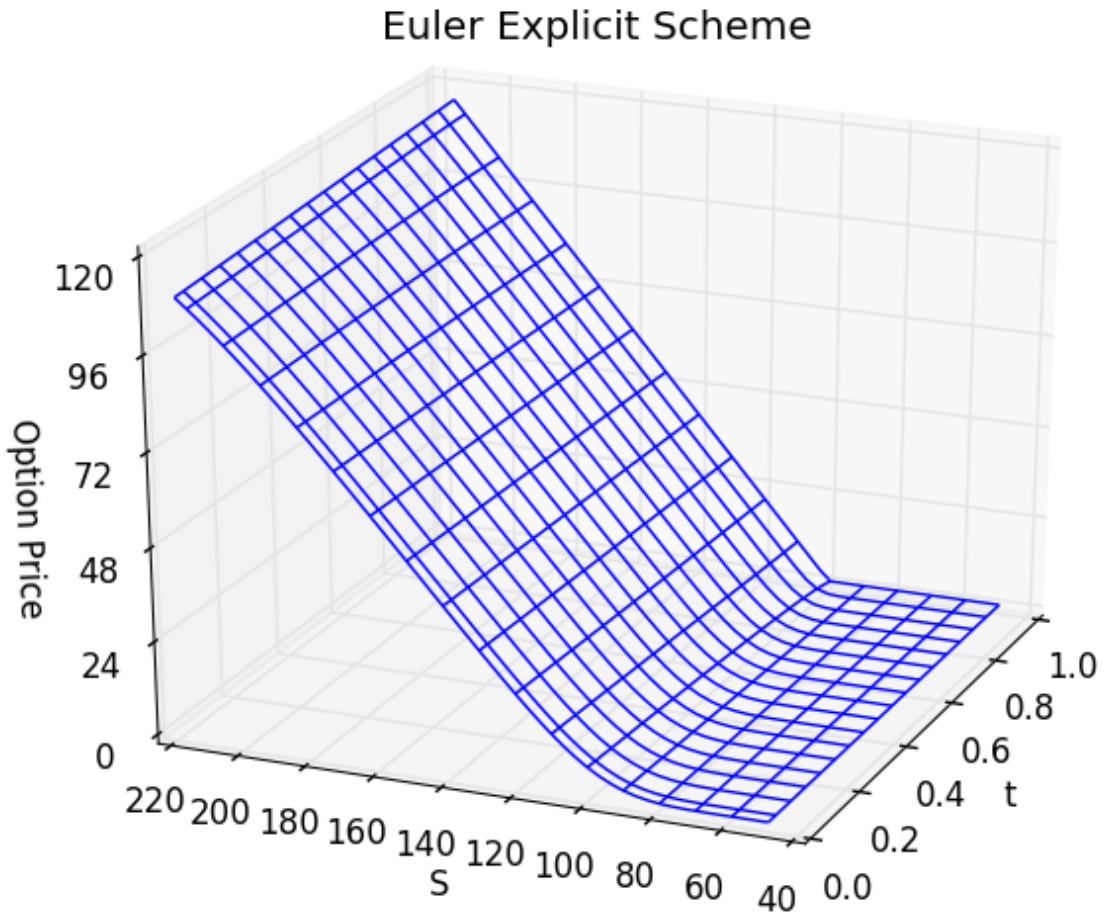
Finite difference error for $f = \exp(x)$

Finite Difference Method — Considerations

Finite difference method is very frequently used in finance

- Numerical pricer: PDE, Monte-Carlo
- Model construction, e.g, local volatility surface construction
- Estimating the greeks: when there is no analytic formula available (almost the case)
- The step size h has to be carefully chosen, too small — unstable, too large — bigger truncation error

Overview of Our Solution Framework



- Discretize the domain (S, t)
- From the final condition at $t = T$, solve backward the option price $V(S, t)$ at the grids
- For any point not on the grid, we can use interpolation

Section 3

Discretizing Spot Dimension

Back To Black-Schole Equation

The Black-Schole PDE is

$$\frac{\partial V}{\partial t} + (r-q)S\frac{\partial V}{\partial S} + \frac{\sigma^2 S^2}{2}\frac{\partial^2 V}{\partial S^2} - rV = 0 \tag{33}$$

For discussion simplicity and without loss of generality we divide the spot axis in equal space (from $S_{\min}$ to $S_{\max}$) by N_S points, and discretize the partial derivatives with respect to S using finite difference:

$$\frac{\partial V_i}{\partial S} = \frac{V_{i+1} - V_{i-1}}{S_{i+1} - S_{i-1}} \tag{34}$$

$$\frac{\partial^2 V_i}{\partial S^2} = \frac{V_{i+1} + V_{i-1} - 2V_i}{(S_{i+1} - S_i)^2} \tag{35}$$

where $i \in 0, 1, \ldots, N_S - 1$.

Substituting the finite differences back to the Black-Scholes PDE we have

$$\frac{\partial V_i}{\partial t} + (r-q)S\frac{V_{i+1} - V_{i-1}}{2\Delta S} + \frac{\sigma^2 S^2}{2}\frac{V_{i+1} + V_{i-1} - 2V_i}{\Delta S^2} - rV_i = 0 \quad (36)$$

Reorganize it we get

$$\frac{\partial V_i}{\partial t} + \frac{(r-q)S}{2\Delta S}V_{i+1} - \frac{(r-q)S}{2\Delta S}V_{i-1} + \frac{\sigma^2 S^2}{2\Delta S^2}V_{i+1} + \frac{\sigma^2 S^2}{2\Delta S^2}V_{i-1} - \frac{\sigma^2 S^2}{\Delta S^2}V_i - rV_i = 0 \quad (37)$$

$$\frac{\partial V_i}{\partial t} + \left(\frac{(r-q)S}{2\Delta S} + \frac{\sigma^2 S^2}{2\Delta S^2}\right)V_{i+1} - \left(\frac{\sigma^2 S^2}{\Delta S^2} + r\right)V_i + \left(\frac{\sigma^2 S^2}{2\Delta S^2} - \frac{(r-q)S}{2\Delta S}\right)V_{i-1} = 0 \quad (38)$$

Let

$$\begin{cases} A_i &= \frac{(r-q)S_i}{2\Delta S} - \frac{\sigma^2 S_i^2}{2\Delta S^2} \\ B_i &= r + \frac{\sigma^2 S_i^2}{\Delta S^2} \\ C_i &= -\frac{\sigma^2 S_i^2}{2\Delta S^2} - \frac{(r-q)S_i}{2\Delta S} \end{cases} \tag{39}$$

We have

$$\frac{\partial V_i}{\partial t} = \begin{pmatrix} A_i & B_i & C_i \end{pmatrix} \cdot \begin{pmatrix} V_{i-1} \\ V_i \\ V_{i+1} \end{pmatrix} \tag{40}$$

This is an ordinary differential equation of V with respect to t. And this ODE holds for $i \in [1, 2, \ldots, N_S - 2]$.

So we have a system of $N_S - 2$ ODE's:

$$\begin{pmatrix} \partial V_1/\partial t \\ \partial V_2/\partial t \\ \vdots \\ \partial V_{N_S-2}/\partial t \end{pmatrix} = \underbrace{\begin{pmatrix} A_1 & B_1 & C_1 & 0 & \dots & 0 \\ 0 & A_2 & B_2 & C_2 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & A_{N_S-2} & B_{N_S-2} & C_{N_S-2} \end{pmatrix}}_{=\mathbf{M}} \begin{pmatrix} V_0 \\ V_1 \\ V_2 \\ \vdots \\ V_{N_S-2} \\ V_{N_S-1} \end{pmatrix} \tag{41}$$

In total there are $N_S - 2$ equations, but N_S variables. V_0 and V_{N_S-1} requires **boundary conditions**.

Differential equations can have infinite amount of solutions, boundary conditions impose constraints such that the solution is unique. Note boundary condition could violate the differential equation itself so does not guarantee the existence of the solution.

Boundary Conditions

- There are three types of boundary conditions that are popular in finance
  - Dirichlet boundary condition (Constraint on V)
  - Neumann boundary condition (Constraint on V')
  - Linear boundary condition (Constraint on $V^{(2)}$)
- Which one to choose depends on what kind of option we are trying to price and what is the state variable of our discretization grid.
- We can also use a mixture of any of them

Dirichlet Boundary Condition

Specifies V at the boundary, e.g., $V(b) = e^{r(T-t)}\text{payoff}(b)$.

Rationale: when the spot is at the extreme wings, the intrinsic values determines the payoff, the volatility has little impact, or the payoff is really known at the boundary, e.g., knock-out options

$$\begin{pmatrix} e^{-r(T-t)}\text{payoff}(S_0) \\ \partial V_1/\partial t \\ \partial V_2/\partial t \\ \vdots \\ \partial V_{N_S-2}/\partial t \\ e^{-r(T-t)}\text{payoff}(S_{N_S-1}) \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ A_1 & B_1 & C_1 & 0 & \dots & 0 \\ 0 & A_2 & B_2 & C_2 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & A_{N_S-2} & B_{N_S-2} & C_{N_S-2} \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}}_{=\mathbf{M}} \begin{pmatrix} V_0 \\ V_1 \\ V_2 \\ \vdots \\ V_{N_S-2} \\ V_{N_S-1} \end{pmatrix} \tag{42}$$

Now our $\mathbf{M}$ is a square matrix. And the solution will be unique if exists.

Neumann Boundary Condition

Specifies the first derivative of V at the boundary: $\frac{\partial V}{\partial S} = c$.

Rationale: the volatility has little impact at the extreme cases and the payoff becomes linear at the wings and we known directly its derivative. With finite difference we have

$$V_1 - V_0 = c_0 \Delta S \quad \rightarrow \quad V_0 = V_1 - c_0 \Delta S \tag{43}$$

$$V_{N_S-1} - V_{N_S-2} = c_{N_S} \Delta S \quad \rightarrow \quad V_{N_S-1} = V_{N_S-2} + c_{N_S} \Delta S \tag{44}$$

Expressing them in the matrix form:

$$\begin{pmatrix} c_0 \Delta S \\ \partial V_1/\partial t \\ \partial V_2/\partial t \\ \vdots \\ \partial V_{N_S-2}/\partial t \\ c_{N_S} \Delta S \end{pmatrix} = \underbrace{\begin{pmatrix} -1 & 1 & 0 & 0 & \dots & 0 \\ A_1 & B_1 & C_1 & 0 & \dots & 0 \\ 0 & A_2 & B_2 & C_2 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & A_{N_S-2} & B_{N_S-2} & C_{N_S-2} \\ 0 & 0 & 0 & \dots & -1 & 1 \end{pmatrix}}_{=\mathbf{M}} \begin{pmatrix} V_0 \\ V_1 \\ V_2 \\ \vdots \\ V_{N_S-2} \\ V_{N_S-1} \end{pmatrix} \tag{45}$$

Again we have **M** in square form and the solution is unique if exists.

Linear Boundary Condition (Zero Curvature)

The second derivative is 0: $\frac{\partial^2 V}{\partial S^2} = 0$

Rationale: payoff becomes linear at the wings. But linear boundary condition leaves the calculation of the first derivative to the system, and allows the first derivative to be changing with respect to time.

Note that we cannot use central difference for V_0 since it's at the boundary, so we say

$$\frac{\partial^2 V_0}{\partial S^2} \approx \frac{\frac{\partial V_1}{\partial S} - \frac{\partial V_0}{\partial S}}{\Delta S} \approx \frac{\frac{V_2 - V_1}{\Delta S} - \frac{V_1 - V_0}{\Delta S}}{\Delta S} = \frac{V_2 + V_0 - 2V_1}{\Delta S^2} \tag{46}$$

Effectively this is the central difference for V_1. Similarly

$$\frac{\partial^2 V_{N_S}}{\partial S^2} \approx \frac{\frac{\partial V_{N_S-1}}{\partial S} - \frac{\partial V_{N_S-2}}{\partial S}}{\Delta S} \tag{47}$$

$$\approx \frac{\frac{V_{N_S-1} - V_{N_S-2}}{\Delta S} - \frac{V_{N_S-2} - V_{N_S-3}}{\Delta S}}{\Delta S} = \frac{V_{N_S-1} + V_{N_S-3} - 2V_{N_S-2}}{\Delta S^2} \tag{48}$$

Linear Boundary Condition

Our system of ODE now becomes

$$\begin{pmatrix} 0 \\ \partial V_1/\partial t \\ \partial V_2/\partial t \\ \vdots \\ \partial V_{N_S-2}/\partial t \\ 0 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & -2 & 1 & 0 & \dots & 0 \\ A_1 & B_1 & C_1 & 0 & \dots & 0 \\ 0 & A_2 & B_2 & C_2 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & A_{N_S-2} & B_{N_S-2} & C_{N_S-2} \\ 0 & 0 & \dots & 1 & -2 & 1 \end{pmatrix}}_{=\mathbf{M}} \begin{pmatrix} V_0 \\ V_1 \\ V_2 \\ \vdots \\ V_{N_S-2} \\ V_{N_S-1} \end{pmatrix} \tag{49}$$

Again we have $\mathbf{M}$ in square form and the solution is unique if exists.

Now let us focus on the solution to the ODE's

Section 4

Ordinary Differential Equations

- Ordinary differential equation involves derivative of only one variable.
- Some of the ODE's have closed form solution (e.g. (50)), some do not and require numerical methods.

$$y' = y \Rightarrow \frac{dy}{dx} = y \Rightarrow \frac{dy}{y} = dx \Rightarrow \int \frac{dy}{y} = \int dx \Rightarrow \ln y = x + C \tag{50}$$

$$y = Ke^x, \quad \text{where } K = e^C \tag{51}$$

- See ▸ Appendix) for a summary of the types of ODE's that have closed form solution (not particular useful for our PDE pricer, but good to know in general, and can be useful in more advanced model),
- We focus on discussion of the numerical methods (these will be useful for our PDE and MC pricers)

Numerical Methods For ODE — Euler Method

- The simplest method yet most popular one

$$y' = \frac{dy}{dx} = \frac{y(x+h) - y(x)}{h} + O(h) \tag{52}$$

$$y' = f(x, y) \tag{53}$$

$$\frac{y(x+h) - y(x)}{h} = f(x, y) + O(h) \tag{54}$$

$$y(x+h) = y(x) + hf(x, y) + O(h^2) \tag{55}$$

- Given $(x, y(x))$ we can calculate $y(x+h)$ — given initial value or final value, we can calculate the whole function y

- Euler's error is $O(h)$ — first order integration method.

Euler Method — Stability

- To analyse the stability of Euler method, we can take the simple ODE $y' = -ky$ as example, with $k > 0$
- Euler method calculates the solution iteratively as $\hat{y}_i$

$$\hat{y}_{i+1} = \hat{y}_i + h(-k\hat{y}_i) = (1 - hk)\hat{y}_i \tag{56}$$

- Note that $\hat{y}_i$ carries an error term e_i, if $|1 - hk| > 1$, the error term gets amplified at each iteration, and the numerical scheme becomes unstable
- So Euler's stability condition is $|1 - hk| < 1$ for this case — at each step the error term is reduced, the numerical method is then stable

Euler Method — Stability

- Now if instead of using the right difference we use the left difference:

$$\frac{y(x+h)-y(x)}{h} \approx f(x+h, y+h) \tag{57}$$

$$\hat{y}_{i+1} - \hat{y}_i = h(-k\hat{y}_{i+1}) \tag{58}$$

- For the same example we will have $\hat{y}_{i+1} = \frac{1}{1+hk}\hat{y}_i$. Since $|\frac{1}{1+hk}| < 1$ is always true ($h > 0$ and $k < 0$), this scheme is unconditionally stable, and this is called **implicit** scheme, as oppose to the previous **explicit** scheme — we will come back to them

Heun's Method (Runge Kutta 2nd order)

$$y'(x) = f(x, y)$$

$$y(x+h) = y(x) + hy'(x) + \frac{1}{2}y''(x)h^2 + O(h^3)$$

$$y''(x) = f'(x, y) = f_x(x, y) + f_y(x, y)f(x, y)$$

$$f(x+h, y+hf(x, y)) = f(x, y) + \underbrace{f_x(x, y)h + f(x, y)f_y(x, y)h}_{y''(x)h} + O(h^2)$$

$$y(x+h) = y(x) + f(x, y)h + \frac{h}{2}(f(x+h, y+hf(x, y)) - f(x, y)) + O(h^3)$$

$$\Downarrow$$

$$y(x+h) = y(x) + \frac{h}{2}(f(x, y(x)) + f(x+h, \underbrace{y(x) + hf(x, y(x))}_{\text{Euler method}}) + O(h^3)$$

- Heun's error is $O(h^3)/h$ — second order integration method.
- $\frac{1}{2}(f(x, y(x)) + f(x+h, y(x+h))$ is used to approximate the gradient, but uses Euler to estimate $f(x+h, y+hf(x, y))$ for $f(x+h, y(x+h))$ since the latter is unknown

Mid Point Method (Runge Kutta 2nd order)

$$y(x+h) = y(x) + hf(x + \frac{h}{2}, y + \frac{h}{2}f(x,y)) + O(h^3)$$

- The idea is to take derivative at the middle point $x + \frac{h}{2}$ and approximate $y(x + \frac{h}{2})$ using Euler estimate $y + \frac{h}{2}f(x,y)$.

Runge Kutta 4th Order Method

$$y_{i+i} = y_i + \frac{h}{6}(F_1 + 2F_2 + 2F_3 + F_4) + O(h^5)$$

where

$$F_1 = f(x, y_i), F_2 = f(x + \frac{h}{2}, y_i + \frac{h}{2}F_1),$$

$$F_3 = f(x + \frac{h}{2}, y_i + \frac{h}{2}F_2), F_4 = f(x + h, y_i + hF_3).$$

More Discretization Schemes

There are many more discretization schemes

- Adaptive Runge Kutta
- Adams (two-step) method
- Predictor corrector
- etc.

Normally lower order scheme is more robust. We will use Euler for PDE, Euler or at most second order scheme for Monte-Carlo.

Discretizing the time dimension

Now we need to deal with the temporal dimension of our system of ODEs. First we divide the temporal dimension into subdivisions (time steps): $t_0 = 0, t_1, \ldots, t_{N_T-1} = T$. And for simplicity again we take equal division so our time interval is Δt.

$$\begin{pmatrix} b_{(0,j)} \\ \partial V_{(1,j)}/\partial t \\ \partial V_{(2,j)}/\partial t \\ \vdots \\ \partial V_{(N_S-2,j)}/\partial t \\ b_{(N_S-1,j)} \end{pmatrix} = \mathbf{M} \begin{pmatrix} V_{(0,j)} \\ V_{(1,j)} \\ V_{(2,j)} \\ \vdots \\ V_{(N_S-2,j)} \\ V_{(N_S-1,j)} \end{pmatrix} \tag{59}$$

We denote the matrix $\mathbf{M}$ and the boundary condition's coefficients $b_{(0,j)}$ and $b_{(N_S-1,j)}$ because they are not of particular interest here.

To discretize the time derivatives, the simplest way is to use Euler scheme. But there are two ways to do that:

- Left difference

$$\frac{\partial V_{i,j}}{\partial t} \approx \frac{V_{i,j} - V_{i,j-1}}{\Delta t} \tag{60}$$

- Right difference:

$$\frac{\partial V_{i,j}}{\partial t} \approx \frac{V_{i,j+1} - V_{i,j}}{\Delta t} \tag{61}$$

Explicit Euler Scheme

Substituting the left difference to (59) we get the **explicit** Euler scheme:

$$\frac{1}{\Delta t}\begin{pmatrix} \Delta t \cdot b_{(0,j)} \\ V_{(1,j)} - V_{(1,j-1)} \\ V_{(2,j)} - V_{(2,j-1)} \\ \vdots \\ V_{(N_S-2,j)} - V_{(N_S-2,j-1)} \\ \Delta t \cdot b_{(N_S-1,j)} \end{pmatrix} = \mathbf{M}\begin{pmatrix} V_{(0,j)} \\ V_{(1,j)} \\ V_{(2,j)} \\ \vdots \\ V_{(N_S-2,j)} \\ V_{(N_S-1,j)} \end{pmatrix} \tag{62}$$

The reason it is explicit is that we know our V from the back. The linear system is an explicit function from $\mathbf{V}_j$ to $\mathbf{V}_{j-1}$.

The derivative instrument's payoff value at time T is the value of the derivative instrument, similar to the tree model. We call that our **final value**.

Denote $\mathbf{V_j}$ the solution at time slice t_j, i.e.,

$$\mathbf{V_j} = \begin{pmatrix} V_{(0,j)} \\ V_{(1,j)} \\ V_{(2,j)} \\ \vdots \\ V_{(N_S-2,j)} \\ V_{(N_S-1,j)} \end{pmatrix} \tag{63}$$

The explicit Euler scheme reads

$$\begin{pmatrix} -\Delta t \cdot b_{(0,j)} \\ V_{(1,j-1)} \\ V_{(2,j-1)} \\ \vdots \\ V_{(N_S-2,j-1)} \\ -\Delta t \cdot b_{(N_S-1,j)} \end{pmatrix} = (\mathbf{D} - \Delta t \mathbf{M})\mathbf{V}_j, \quad \mathbf{D} = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \end{pmatrix} \tag{64}$$

Note that (64) holds only for $V_{(1,j-1)}$ to $V_{(N_S-2,j-1)}$ in $\mathbf{V_{j-1}}$, the two boundary points $V_{0,j-1}$ and $V_{N_S-1,j-1}$ still rely on the boundary conditions, which are the first and last rows of the right hand side of (64).

Explicit Euler Scheme

- Stability condition of explicit scheme is

$$\begin{cases} \Delta t & \leq \left(\frac{\Delta S}{\sigma S_{\max}}\right)^2 \\ \Delta S & \leq \frac{\sigma^2 S_{\min}}{|r-q|} \end{cases} \tag{65}$$

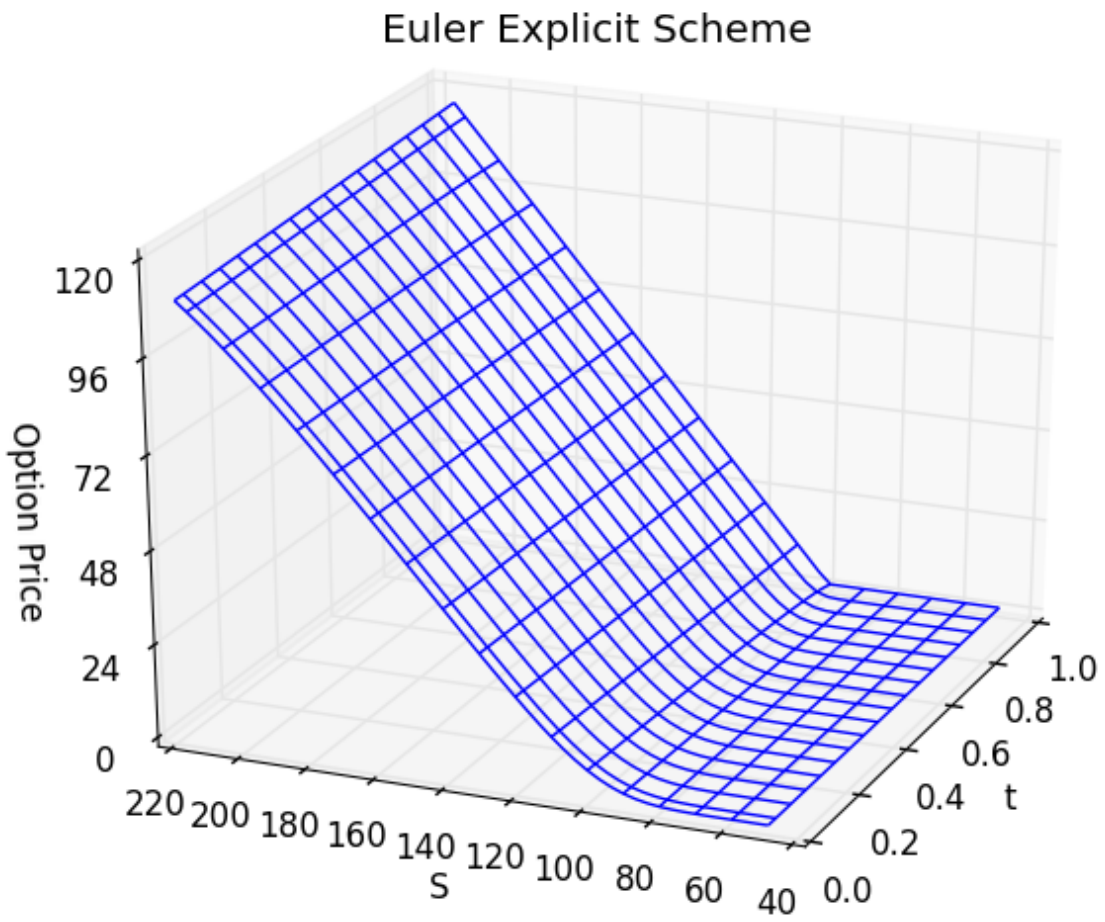
  The conditions are very restrictive — see the example code. And when they violates, the result simply blows up

- Rigorous stability analysis is non-trivial, but intuition is the same as the simple example given in the ODE section
- Error of explicity scheme is $O(\Delta t, \Delta S^2)$ — first order in time and second order in spot dimension.
- The times slice $\mathbf{V}_0$ gives us the option price for each spot at time 0. For any spot that does not lie on the points, we can use linear interpolation.

Euler Explicit Scheme — Implementation

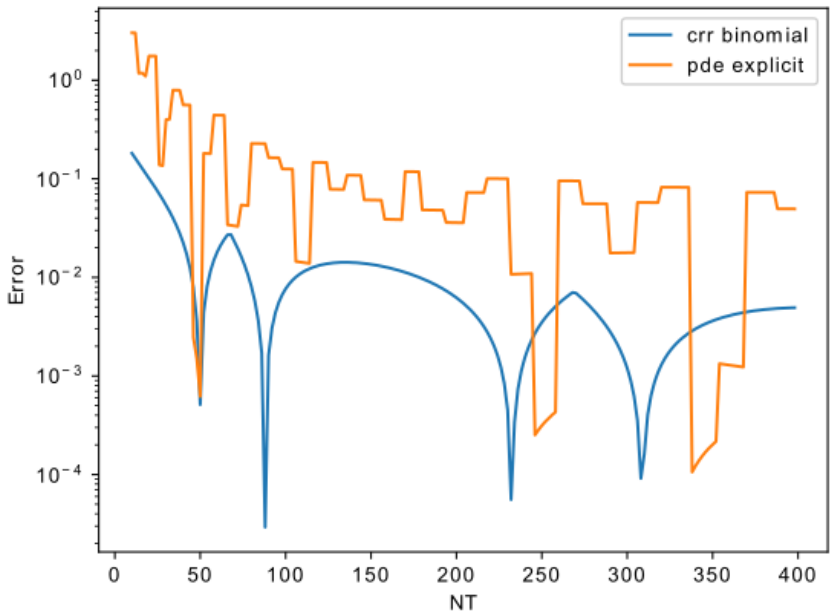
```
def pdeExplicitPricer(S0, r, q, vol, NS, NT, trade):
    # set up pde grid
    mu, T = r - q, trade.expiry
    srange = 5 * vol * math.sqrt(T)
    maxS = S0 * math.exp((mu - vol * vol * 0.5)*T + srange)
    minS = S0 * math.exp((mu - vol * vol * 0.5)*T - srange)
    dt, dS = T / (NT-1), (maxS - minS) / (NS-1)
    sGrid = np.array([minS + i*ds for i in range(NS)]) # set up spot grid
    ps = np.array([trade.payoff(s) for s in sGrid])     # initialize the payoff
    # set up the matrix, for BS the matrix does not change, for LV we need to update it for each iteration
    a, b = mu/2.0/ds, vol * vol / ds / ds
    M = np.zeros((NS, NS))
    D = np.zeros((NS, NS))
    for i in range (1, NS-1):
        M[i, i-1] = a*sGrid[i] - b*sGrid[i]*sGrid[i]/2.0
        M[i, i], D[i,i] = r + b * sGrid[i] * sGrid[i], 1.0
        M[i, i+1] = -a * sGrid[i] - b*sGrid[i]*sGrid[i]/2.0
    # the first row and last row depends on the boundary condition - we use Dirichlet here
    M[0,0], M[NS-1, NS-1] = 1.0, 1.0
    M = D - dt * M
    for j in range(1, NT): # backward induction
        ps = M.dot(ps)  # Euler explicit
        ps[0] = math.exp(-r*j*dt) * trade.payoff(sGrid[0]) # discounted payoff
        ps[NS-1] = math.exp(-r*j*dt) * trade.payoff(sGrid[NS-1])

    return np.interp(S0, sGrid, ps) # linear interpolate the price at S0
```

Euler Explicit Scheme — Result



- $K = 100, S = 100, r = 5\%, q = 2\%, T = 1, \sigma = 15\%, N_S = 50, N_T = 100$

Euler Explicit Scheme – Errors



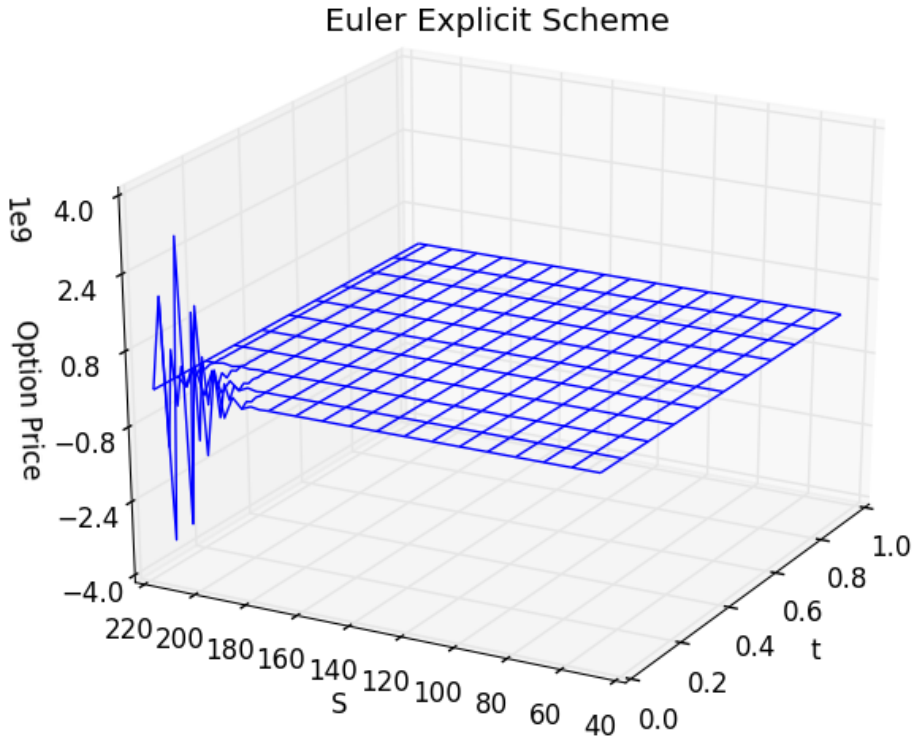
- Need to be careful when changing the number of time steps, one stability condition is

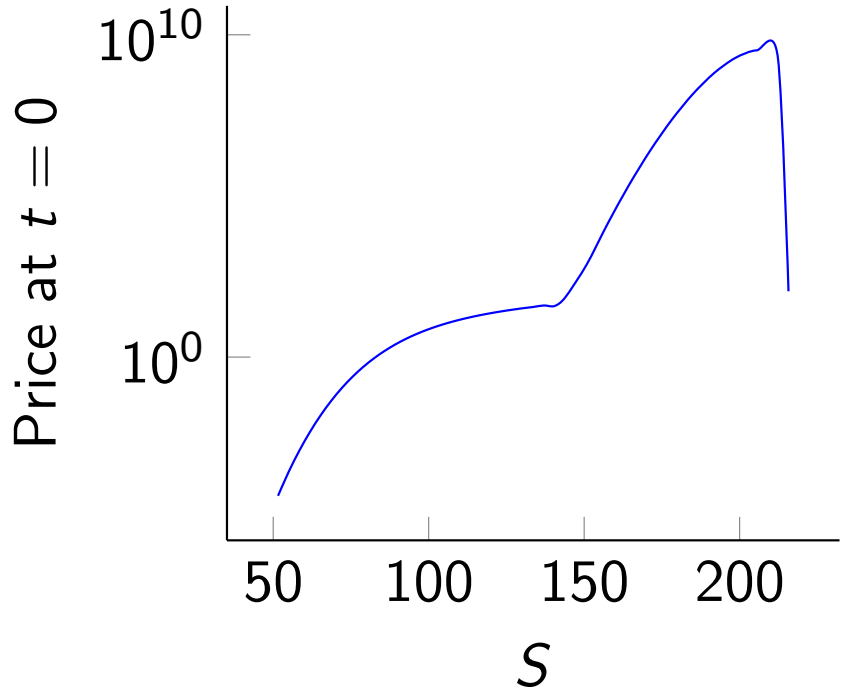
$$\frac{\Delta t}{\Delta S^2} \leq \frac{1}{\sigma^2 S_{\max}^2} \rightarrowtail \frac{N_S^2}{N_T} \leq \frac{1}{\sigma^2}$$

- When we increase N_t by 4 times, We can only increase N_S by 2 times, the comparison with tree is therefore unfair
- The error reduces linear in time — $O(\Delta t, \Delta S^2)$

When Stability Condition Get Violated

The result will be very obviously wrong:





$K = 100, S = 100, r = 5\%, q = 2\%, T = 1, \sigma = 15\%, N_S = 50, N_T = 60$

- Note that the central region is also oscillating but just not visible in the chart due to the corner

Implicit Euler Scheme

Substituting the right difference

$$\frac{\partial V_{i,j}}{\partial t} \approx \frac{V_{i,j+1} - V_{i,j}}{\Delta t} \tag{66}$$

to (59), we get the **implicit** Euler scheme:

$$\frac{1}{\Delta t} \begin{pmatrix} \Delta t \cdot b_{(0,j)} \\ V_{(1,j+1)} - V_{(1,j)} \\ V_{(2,j+1)} - V_{(2,j)} \\ \vdots \\ V_{(N_S-2,j+1)} - V_{(N_S-2,j)} \\ \Delta t \cdot b_{(N_S-1,j)} \end{pmatrix} = \mathbf{M} \begin{pmatrix} V_{(0,j)} \\ V_{(1,j)} \\ V_{(2,j)} \\ \vdots \\ V_{(N_S-2,j)} \\ V_{(N_S-1,j)} \end{pmatrix} \tag{67}$$

Now since we are calculating backward, we know $\mathbf{V}_{j+1}$ and want to solve for $\mathbf{V}_j$ so:

$$\begin{pmatrix} \Delta t \cdot b_{(0,j)} \\ V_{(1,j+1)} \\ V_{(2,j+1)} \\ \vdots \\ V_{(N_S-2,j+1)} \\ \Delta t \cdot b_{(N_S-1,j)} \end{pmatrix} = (\mathbf{D} + \Delta t \mathbf{M})\mathbf{V}_j, \qquad \mathbf{D} = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \end{pmatrix} \tag{68}$$

This is an **implicit** function of $\mathbf{V}_j$, therefore

$$\mathbf{V}_j = (\mathbf{D} + \Delta t \mathbf{M})^{-1} \begin{pmatrix} \Delta t \cdot b_{(0,j)} \\ V_{(1,j+1)} \\ V_{(2,j+1)} \\ \vdots \\ V_{(N_S-2,j+1)} \\ \Delta t \cdot b_{(N_S-1,j)} \end{pmatrix} \tag{69}$$

Implicit Euler Scheme

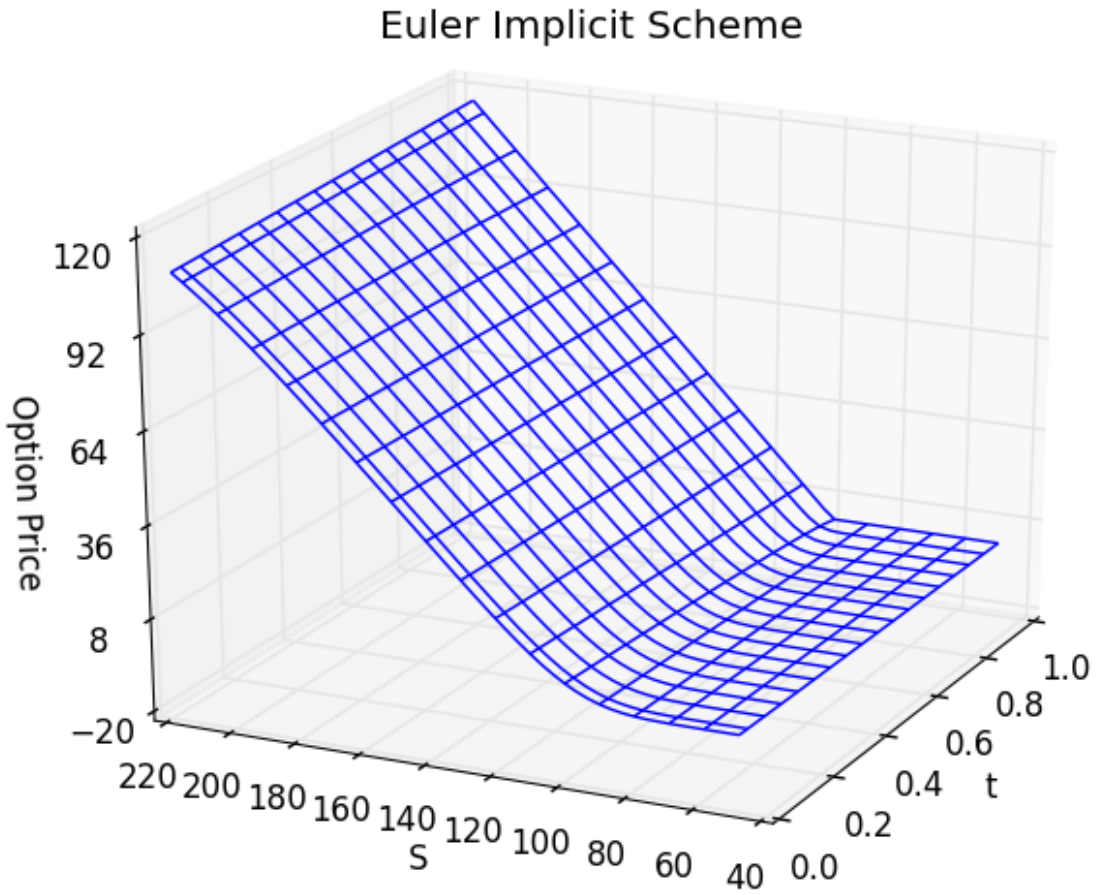
- Implicit Euler scheme is unconditionally stable
- Error of implicit scheme is $O(\Delta t, \Delta S^2)$ — same as explicit Euler
- The implementation involves an inverse of the tri-diagonal matrix — we can use a LU decomposition
- Side note: linear PDE after discretization becomes linear systems.

Euler Implicit Scheme — Implementation

Differs from the explicit scheme implementation only at the iteration part

```
    def pdeImplicitPricer(S0, r, q, vol, NS, NT, trade):
    # set up pde grid
    mu, T = r - q, trade.expiry
    srange = 5 * vol * math.sqrt(T)
    maxS = S0 * math.exp((mu - vol * vol * 0.5)*T + srange)
    minS = S0 * math.exp((mu - vol * vol * 0.5)*T - srange)
    dt, dS = T / (NT-1), (maxS - minS) / (NS-1)
    sGrid = np.array([minS + i*ds for i in range(NS)]) # set up spot grid
    ps = np.array([trade.payoff(s) for s in sGrid])     # initialize the payoff
    # set up the matrix, for BS the matrix does not change, for LV we need to update it for each iteration
    a, b = mu/2.0/ds, vol * vol / ds / ds
    M = np.zeros((NS, NS))
    D = np.zeros((NS, NS))
    for i in range (1, NS-1):
        M[i, i-1] = a*sGrid[i] - b*sGrid[i]*sGrid[i]/2.0
        M[i, i], D[i,i] = r + b * sGrid[i] * sGrid[i], 1.0
        M[i, i+1] = -a * sGrid[i] - b*sGrid[i]*sGrid[i]/2.0
    # the first row and last row depends on the boundary condition - we use Dirichlet here
    M[0,0], M[NS-1, NS-1] = 1.0, 1.0

    # only different from explicit at backward induction HERE
    M = numpy.linalg.inv(D + dt * M)
    for j in range(1, NT):
        ps[0] = dt*math.exp(-r*j*dt) * trade.payoff(sGrid[0]) # discounted payoff
        ps[NS-1] = dt*math.exp(-r*j*dt) * trade.payoff(sGrid[NS-1])
        ps = M.dot(ps)  # Euler implicit
    return np.interp(S0, sGrid, ps) # linear interpolate the price at S0
```

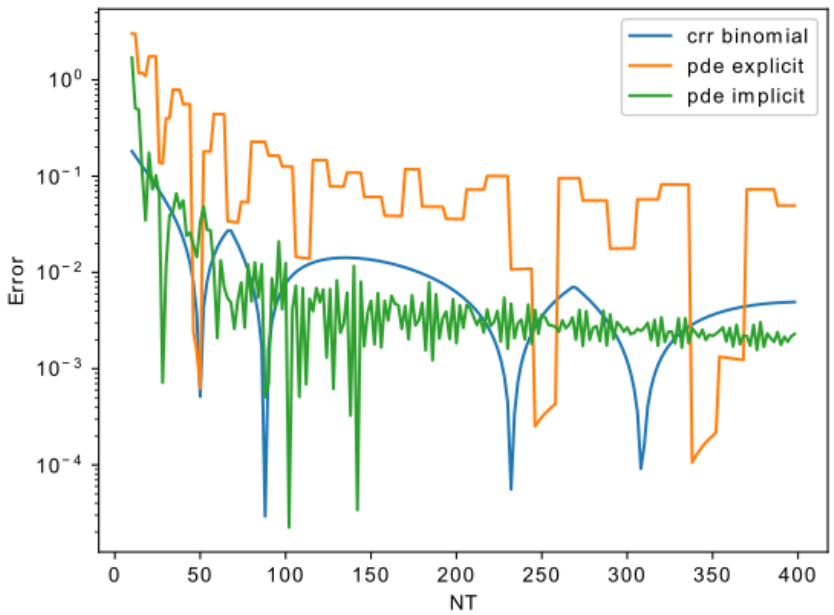
Euler Implicit Scheme — Result



- $K = 100, S = 100, r = 5\%, q = 2\%, T = 1, \sigma = 15\%, N_S = 50, N_T = 100$
- No much difference from explicit scheme when explicit scheme is stable

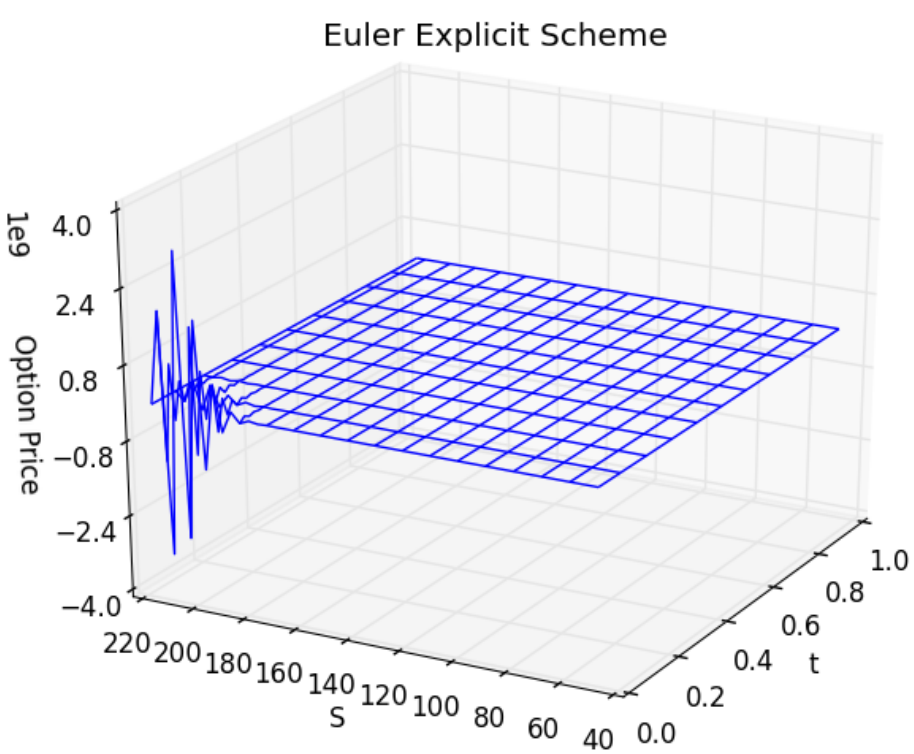
Euler Implicit Scheme – Errors

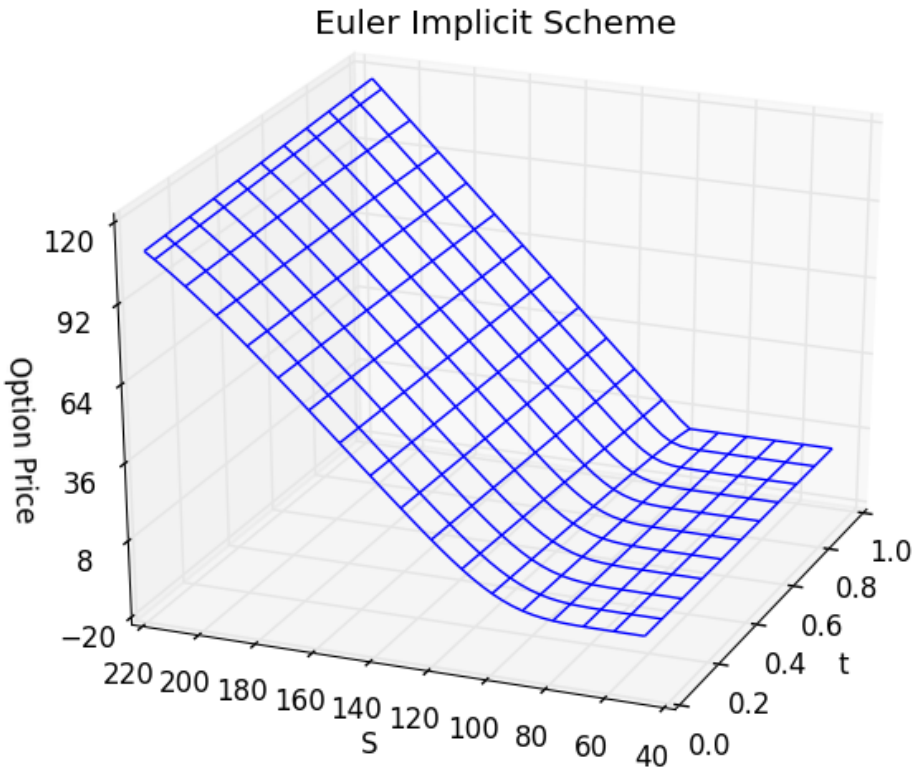
```
rg = range(10, 400, 2)
pdeErrsE = [(abs(pdeExplicitPricer(S,r,0.0,vol,int(math.sqrt(i/vol)),i,opt)-bsprc)) for i in rg]
pdeErrsI = [(abs(pdeImplicitPricer(S,r,0.0,vol,i,i,opt) - bsprc)) for i in rg]
crrErrs = [(abs(binomialPricer(S,r,vol,opt,i,crrCalib) - bsprc)) for i in rg]
```



- Unconditionally stable, so do not need to pay too much attention on the descritization
- When we increase N_t by 2 times, we can increase N_S by two times
- That's why the convergence appears faster than explicit
- It's still linear in time — $O(\Delta t, \Delta S^2)$, just allow us to eliminate the ΔS^2 term quickly

Euler Implicit Stability





$K = 100, S = 100, r = 5\%, q = 2\%, T = 1, \sigma = 15\%, N_S = 50, N_T = 60$

- Implicit scheme is still stable when explicit scheme is not

Douglas Scheme

Let's put together the implicit and explicit Euler scheme:

$$\mathbf{V}_{j-1} = (\mathbf{D} - \Delta t\mathbf{M})\mathbf{V}_j \tag{70}$$

$$(\mathbf{D} + \Delta t\mathbf{M})\mathbf{V}_{j-1} = \mathbf{V}_j \tag{71}$$

Here we omit the special case for the two boundary points — they can be derived once we get the middles points of $\mathbf{V}_{j-1}$ anyway.

Douglas scheme is to do a weighted sum of the two schemes:

$$w\mathbf{V}_{j-1} + (1-w)(\mathbf{D} + \Delta t\mathbf{M})\mathbf{V}_{j-1} = [w(\mathbf{D} - \Delta t\mathbf{M}) + (1-w)\mathbf{I}] \cdot \mathbf{V}_j \tag{72}$$

$$[\mathbf{I}_w + (1-w)\Delta t\mathbf{M}]\mathbf{V}_{j-1} = [\mathbf{I}_{1-w} - w\Delta t\mathbf{M}] \cdot \mathbf{V}_j \tag{73}$$

where $\mathbf{I}_w$ and $\mathbf{I}_{1-w}$ differs from the identity matrix only at the first and last row (replaced by w and $1-w$)

Douglas Scheme

Solution of Douglas scheme

$$\mathbf{V}_{j-1} = [\mathbf{I}_w + (1-w)\Delta t\mathbf{M}]^{-1}[\mathbf{I}_{1-w} - w\Delta t\mathbf{M}] \cdot \mathbf{V}_j \tag{74}$$

Why Douglas scheme?

The truncation error for Douglas scheme is

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

So if we choose $w = \frac{1}{2} - \frac{\Delta S^2}{12\Delta t}$ we can achieve $O(\Delta t^2, \Delta S^4)$

Crank-Nicholson Scheme

- And when the weight $w = 0.5$, it is the famous **Crank-Nicholson scheme**:

$$[\mathbf{I}_{0.5} + 0.5\Delta t\mathbf{M}]\mathbf{V}_{j-1} = [\mathbf{I}_{0.5} - 0.5\Delta t\mathbf{M}] \cdot \mathbf{V}_j \tag{75}$$

- Crank-Nicholson is unconditionally stable. Intuition with $y' = -ky$:

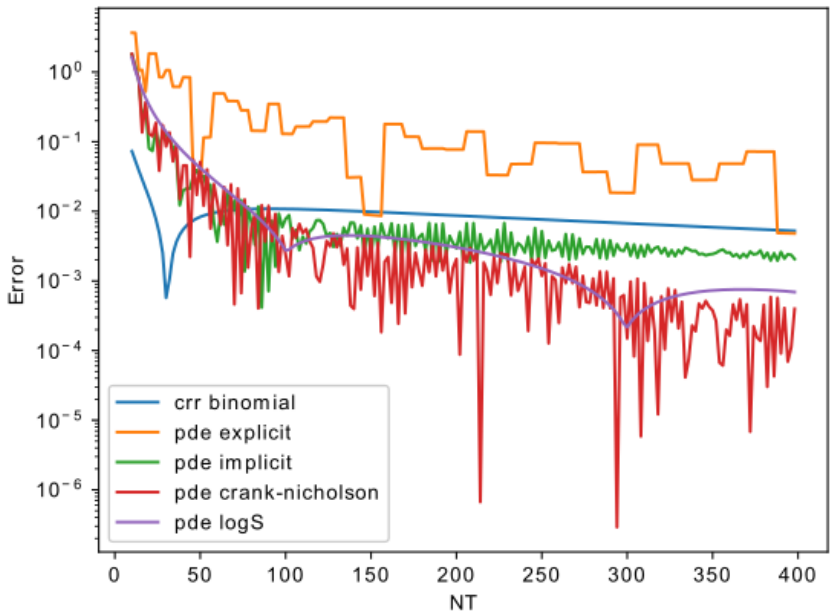
$$\hat{y}_{i+1} = \hat{y}_i - hk\hat{y}_i \quad \rightarrow \quad \hat{y}_{i+1} = (1 - hk)\hat{y}_i \quad \text{explicit} \tag{76}$$

$$\hat{y}_{i+1} = \hat{y}_i - hk\hat{y}_{i+1} \quad \rightarrow \quad \hat{y}_{i+1} = \frac{1}{1 + hk}\hat{y}_i \quad \text{implicit} \tag{77}$$

$$\hat{y}_{i+1} = \hat{y}_i - \frac{hk}{2}(\hat{y}_{i+1} + \hat{y}_i) \quad \rightarrow \quad \hat{y}_{i+1} = \frac{2 - hk}{2 + hk}\hat{y}_i \quad \text{Crank-N} \tag{78}$$

- Truncation error is $O(\Delta t^2, \Delta S^2)$: second order in time
- Very popular scheme in finance
- It does have drawbacks — oscillation when payoff is discontinuous. Solution is to use implicit scheme or other methods to smooth out the discontinuities, when volatility is low and drift is high — see the second stability condition of explicit scheme: $\Delta S \leq \frac{\sigma^2 S_{\min}}{|r-q|}$

Crank Nicholson Scheme – Errors



- Unconditionally stable, so do not need to pay too much attention on the discretization
- When we increase N_t by 2 times, we can increase N_S by two times
- The error is second order in time — $O(\Delta t^2, \Delta S^2)$

Richardson Extrapolation

- Similar to the idea of Richardson Extrapolation applied to finite difference, if we have two solutions V_1 and V_2 whose grid satisfies

$$\frac{\Delta t_1}{\Delta S_1^2} = \frac{\Delta t_2}{\Delta S_2^2}, \tag{79}$$

- We can achieve $O(\Delta t^2, \Delta S^3)$ by extrapolating V_1 and V_2 by

$$V = \frac{\Delta S_2^2 V_1 - \Delta S_1^2 V_2}{\Delta S_2^2 - \Delta S_1^2} \tag{80}$$

- Note that all these are trivial to implement when we have explicit and implicit Euler schemes implemented.

Change Of Variable

- So far we have been using equally spaced discretization in time and spot dimensions. Our derivation can be generalized to non-regular discretization.
- One general guideline to construct the PDE grid is to place more points at the regions where V is sensitive to the spot: e.g., near the current spot or option strike.
- The choice of the spot as state variable is not ideal when we use equal spacing: basically we need a lot more points at the center than at the wings, but with equal spacing spot discretization, if we need to achieve certain precision we need to add a lot of points at the wings as well
- A more natural choice of state variable is the log spot: $X = \log S$ — linear in X means exponential in S and our grid will be naturally denser at the center region and wider at the wings.

So we need to express our PDE in terms of X instead of S. And this is very straight-forward:

$$\frac{\partial V}{\partial S} = \frac{\partial V}{\partial X}\frac{\partial X}{\partial S} = \frac{1}{S}\frac{\partial V}{\partial X} \tag{81}$$

$$\frac{\partial^2 V}{\partial S^2} = \frac{\partial(\frac{1}{S}\frac{\partial V}{\partial X})}{\partial S} = \frac{1}{S^2}\frac{\partial^2 V}{\partial X^2} - \frac{1}{S^2}\frac{\partial V}{\partial X} \tag{82}$$

So the Black-Scholes PDE becomes:

$$\frac{\partial V}{\partial t} + \left(r - q - \frac{\sigma^2}{2}\right)\frac{\partial V}{\partial X} + \frac{\sigma^2}{2}\frac{\partial^2 V}{\partial X^2} - rV = 0 \tag{83}$$

Our coefficients are not even S dependent now. The rest is exactly the same as in the previous discussion, only coefficients changes:

$$A = \frac{r - q - \frac{\sigma^2}{2}}{2\Delta X} - \frac{\sigma^2}{2\Delta X^2} \tag{84}$$

$$B = r + \frac{\sigma^2}{\Delta X^2} \tag{85}$$

$$C = -\frac{r - q - \frac{\sigma^2}{2}}{2\Delta X} - \frac{\sigma^2}{2\Delta X^2} \tag{86}$$

Appendix: ODE's with Closed Form Solution

Seperable ODE

A separable ODE can be written as $\frac{dy}{dx} = g(x)h(y)$. The solution is therefore very simple: integrating both sides of

$$\frac{dy}{h(y)} = g(x)dx \quad \Rightarrow \quad \int \frac{dy}{h(y)} = \int g(x)dx$$

There are cases that an ODE appears non-separable, but is in fact separable after a change of variable. For example, $y' = \frac{x-y}{x+y}$. Apply change of variable $z = x + y$, the right hand side becomes $\frac{-z+2x}{z}$. The left hand side is $\frac{dy}{dx} = \frac{d(z-x)}{dx} = \frac{dz}{dx} - 1$. So we have

$$\frac{dz}{dx} = \frac{2x}{z} \quad \Rightarrow \quad \int z dz = \int 2x dz + c \quad \Rightarrow \quad \frac{1}{2}z^2 = x^2 + c$$

The solution is trivial: $y^2 + 2xy - x^2 = c$

First-order linear ODE

A first order linear ODE has the form $\frac{dy}{dx} + P(x)y = Q(x)$. Apparently it is non-separable. The standard approach to solve it is to find a suitable integrating factor $I(x)$, such that

$$I(x)(y' + P(x)y) = I(x)y' + I(x)P(x)y = (I(x)y)'. \tag{87}$$

If (87) holds, the original ODE becomes $(I(x)y)' = I(x)Q(x)$ and we have

$$I(x)y = \int (I(x)Q(x))dx \quad \Rightarrow \quad y(x) = \frac{\int I(x)Q(x)dx}{I(x)}.$$

For (87) to hold, we need to have

$$I(x)' = I(x)P(x) \quad \Rightarrow \quad \frac{dI(x)}{I(x)} = P(x)dx \quad \Rightarrow \quad I(x) = e^{\int P(x)dx}.$$

Applying the above technique we can derive easily that the solution of $y' + \frac{y}{x} = \frac{1}{x^2}$ is $\frac{\ln x + c}{x}$.

Homogeneous linear ODE

A homogeneous linear ODE is a **second order** linear ODE of the form

$$a(x)y'' + b(x)y' + c(x)y = 0$$

It is easy to show that if y_1 and y_2 are linearly independent solutions to the ODE, then any $y(x) = c_1 y_1(x) + c_2 y_2(x)$ for arbitrary constants $c_{1,2}$ is also a solution to the ODE.
When a, b, and c ($a \neq 0$) are constants, the homogeneous linear ODE

$$ay'' + by' + cy = 0, \tag{88}$$

has closed form solutions. Let r_1 and r_2 be the roots of the quadratic equation $ar^2 + br + c = 0$, typically $r_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ The closed form solution is

$$y(x) = \begin{cases} c_1 e^{r_1 x} + c_2 e^{r_2 x} & \text{if } \ b^2 - 4ac > 0 \\ c_1 e^{rx} + c_2 x e^{rx} & \text{if } \ b^2 - 4ac = 0, r_1 = r_2 = r = -\frac{b}{2a} \\ e^{\alpha x}(c_1 \cos \beta x + c_2 \sin \beta x) & \text{if } \ b^2 - 4ac < 0, \ r_{1,2} = \alpha \pm i\beta = -\frac{b}{2a} \pm i\frac{\sqrt{4ac - b^2}}{2a} \end{cases}$$

Non-homogeneous linear ODE

A non-homogeneous linear ODE is a **second order** linear ODE of the form

$$a(x)y'' + b(x)y' + c(x)y = d(x)$$

It does not have closed form solution even if a, b, c are constants.

However, for constant coefficients if we can find a particular solution $y_p(x)$ for

$$ay'' + by' + cy = d(x), \tag{89}$$

It can be shown that $y(x) = y_g(x) + y_p(x)$ is a general solution for (89), where $y_g(x)$ is a general solution for the homogeneous linear ODE (88). When $d(x)$ is a simple polynomial, the particular solution is often a polynomial of the same degree.

Non-homogeneous Linear ODE — Example

- For example, let us look at

$$y'' + y' + y = x$$

- A particular solution of it is of the form $y = mx + n$: Plugging it into the ODE we obtain $m + mx + n = x$ so $m = 1$ and $n = -1$, and $y_p(x) = x - 1$.
- The closed form solution for the corresponding homogeneous ODE is

$$y_g(x) = e^{\alpha x}(c_1 \cos \beta x + c_2 \sin \beta x)$$

where $\alpha = -\frac{1}{2}$ and $\beta = \frac{\sqrt{3}}{2}$.

- The general solution of the non-homogeneous linear ODE is then $y(x) = e^{\alpha x}(c_1 \cos \beta x + c_2 \sin \beta x) + x - 1$.