Numerical Project 2:
The Constant Elasticity Variance model

Amie Fall, Dani González

March 15, 2024

Contents

1 Part 1

1.1 Introduction

In finance, we use a lot of different models to describe the behaviour of an asset's price within a given time frame, one of those models is the Constant Elasticity of Variance (CEV) model.

In the CEV, the price of a stock option with maturity $T > 0$ and payoff $g(x) = g(S(T))$ is given by $\Pi_Y(t) = e^{r(t-t)}u(t, S(t))$. Here $S(t)$ is the price of the stock at the time t and u is the solution to the following differential equation.

$$\frac{d}{dt}u(t, S(t)) + rx\frac{d}{dx}u(t, S(t)) + \frac{\sigma^2}{2}x^{2\delta}\frac{d^2}{dx^2}u(t, S(t)) = 0$$

$x > 0$, $t \in (0, T)$ and the terminal condition $u(T, x) = g(x)$, $x > 0$.

The CEV have several applications, but one of the most essential ones is the price-setting option, which is then used to calculate the implied volatility of an option. If an option does not present linear patterns, this type of approach can be very useful, as well as for assets that present a high degree of volatility.

Other than that, CEV is used in financial risk, to model different financial instruments, such as swaps and futures contracts.

Since CEV is applicable in risk management, it is also directly used as a way to manage risk and therefore optimize portfolios. For example, by tailoring portfolios to an investor's risk aversion and investment goals.

As importantly, CEV can be used as a forecasting tool to forecast different assets in a given time frame. Investors can then make well-educated decisions when it comes to their investment goals. The insights into risk and returns, that CEV gives, are obtained by modelling the connection between an asset's price and its volatility.

Another model used to describe an asset's price is the Black-Scholes-Merton model, which uses mathematical equations that consider time and other factors to estimate the theoretical value of derivatives, hence pricing options.

The main difference between the CEV model and the Black-Scholes- Merton model is how they approach the volatility of an asset's price. In the CEV model, the volatility will be depicted by a function that will increase or decrease, depending on the price level of the asset. If the asset's price is low/high, then the volatility of the returns for that specific asset can be expected to be lower/higher. Instead, the Black-Scholes-Merton model will assume that the volatility of an asset is constant within the given time frame, hence independent of the price of the asset. Assuming that the volatility is constant may work out for some assets, however, it can generate inaccurate pricing of assets that have big volatility changes.

Another difference, between the two models, is the impact that changes in the stock price have on option prices. For the CEV model, the impact will be non-linear and dependent on the level of the stock price, on account of the power-law relationship. As for the BSM model, the impact will be linear and independent of the level of the stock price.

The mathematical formulation of the models is another difference that we can point out, starting with the volatility assumptions that we talked about earlier. Other than that, the BSM

model is based on a closed-form solution, while the CEV model is based on partial differential equations and is often calculated numerically due to its complexity.

Last but not least, the models are limited in different ways and are therefore also applied differently. The BSM model does not take into account that the volatility may change during the option's life and is consequently limited in that aspect. Whereas, the CEV model is more flexible and more suitable for when volatility fluctuates over time. The BSM model is simpler in its application and is mainly used for European option pricing, though the CEV model is applied when the BSM model reaches its limits and one needs to take into account fluctuating volatility.

1.2 Exercise 6.27

Given σ, r, and $\delta \neq 1$ define, $a = 2r(\delta - 1)$,$b = \frac{\sigma^2}{2r}(2\delta - 1)$, $c = -2\sigma(\delta - 1)$, and $\theta = -\frac{1}{2(\delta - 1)}$

Let $\{X(t)\}_{t\geq 0}$ be the CIR process

$$dX(t) = a(b - X(t))dt + c\sqrt{X(t)}d\tilde{W}(t), X(0) = x > 0.$$

Show that $S(t) = X(t)^\theta$ solves $dS(t) = rS(t)dt + \sigma S(t)\delta d\tilde{W}(t)$, $S(0) = S_0 > 0$ with $S(0) = x^\theta$

Solution:

We will start by computing $dX(t)^\theta$

$$dX(t)^\theta = \theta X(t)^{\theta-1}dX(t) + \frac{1}{2}\theta(\theta - 1)X(t)^{\theta-2}dX(t)dX(t).$$

We know that $dX(t) = a(b - X(t))dt + c\sqrt{X(t)}d\tilde{W}(t)$, and if we use that information we get the following equation

$$dX(t)^\theta = \theta X(t)^{\theta-1}[a(b - X(t))dt + c\sqrt{X(t)}d\tilde{W}(t)] + \frac{1}{2}\theta(\theta - 1)X(t)^{\theta-2}[c^2X(t)dt]$$

$$\Leftrightarrow dX(t)^\theta = [a(b - X(t))\theta X(t)^{\theta-1} + \frac{1}{2}\theta(\theta - 1)X(t)^{\theta-2}c^2X(t)]dt + \theta X(t)^{\theta-1}c\sqrt{X(t)}d\tilde{W}(t)$$

$$\Leftrightarrow dX(t)^\theta = [ab\theta X(t)^{\theta-1} - a\theta X(t)^\theta + \frac{1}{2}\theta(\theta - 1)X(t)^{\theta-1}c^2]dt + c\theta X(t)^{\theta-\frac{1}{2}}d\tilde{W}(t)$$

$$= \left[\frac{\sigma^2(\delta - 1)(2\delta - 1)}{2(\delta - 1)}X(t)^{\theta-1} + rX(t)^\theta + \frac{1}{2}\left(\frac{-1}{2(\delta - 1)}\right)\left(\frac{-(2\delta - 1)}{2(\delta - 1)}\right)X(t)^{\theta-1}4\sigma^2(\delta - 1)^2\right]dt$$
$$+ \sigma X(t)^{-\frac{\delta}{2(\delta-1)}}d\tilde{W}(t)$$

$$= \left[-\frac{\sigma^2}{2}(2\delta - 1)X^{\theta-1}(t) + rX(t)^\theta + \frac{\sigma^2}{2}(2\delta - 1)X(t)^{\theta-1}\right]dt + \sigma(X(t)^\theta)^\delta d\tilde{W}(t)$$

$$\Leftrightarrow dX(t)^\theta = rX(t)^\theta dt + \sigma(X(t)^\theta)^\delta d\tilde{W}(t) \text{ and } X(0)^\theta = u^\theta > 0.$$

Hence, $S(t) = X(t)^\theta$ solves our equation, with $S_0 = u^\theta$.

1.3 Finite Difference Scheme: Crank-Nicolson Method

We now want to write a finite difference scheme for the following Partial Differential Equation (PDE):

$$-\partial_t u(t,x) + rx\partial_x u(t,x) + \frac{\sigma^2}{2}x^{2\delta}\partial_x^2 u(t,x) = 0$$

on the domain $(t, x) \in (0, T) \times (0, X)$ with initial data and boundary conditions corresponding to European call or put options. To do this, we use the Crank-Nicholson Method.

Solution:

We are given the following PDE on the domain $(t, x) \in (0, T) \times (0, X)$:

$$-\partial_t u(t, x) + rx\partial_x u(t, x) + \frac{\sigma^2}{2} x^{2\delta} \partial_x^2 u(t, x) = 0$$

$u(0, x) = max(x - K, 0)$

Boundary conditions: $u(t, 0) = 0$ and $u(t, X) = X - Ke^{-r(T-t)}$

As seen in class, the Crank-Nicholson method is the average of two methods:

- Forward in time, centered in space
- Backward in time, centered in space

We start with Forward in time, centered in space method:

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

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

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

$$\frac{u(t + \Delta t, x) - u(t, x)}{\Delta t} = rx\left(\frac{u(t, x + \Delta x) - u(t, x - \Delta x)}{\Delta x}\right) + \frac{\sigma^2}{2} x^{2\delta} \frac{u(t, x + \Delta x) - 2u(t, x) + u(t, x - \Delta x)}{\Delta x^2}$$

$$u(t+\Delta t, x) = u(t, x) + rx\Delta t\left(\frac{u(t, x + \Delta x) - u(t, x - \Delta x)}{\Delta x}\right) + \frac{\sigma^2}{2} x^{2\delta} \Delta t\left(\frac{u(t, x + \Delta x) - 2u(t, x) + u(t, x - \Delta x)}{\Delta x^2}\right)$$

Here the $\Delta t = \frac{n}{t}$, with t being the number of steps and T the time of maturity. As for $\Delta x = \frac{m}{2X}$, where m is the number of steps and x is the space partition in our case.

Where $d = \frac{\Delta t}{\Delta x^2}$ represents the discrete partition of the grid.

We can now discretize the equation so that it will depend on previous values,

$$u_{i+1,j} = u_{i,j} + rx\Delta x(d)(u_{i,j+1} - u_{i,j-1}) + \frac{\sigma^2}{2} x^{2\delta}(d)(u_{i,j+1} - 2u_{i,j} + u_{i,j-1})$$

Backward in time, centered in space:

Here we take the equation previously found and apply the

$$u_{i+1,j} = u_{i,j} + rx\Delta x(d)(u_{i+1,j+1} - u_{i+1,j-1}) + \frac{\sigma^2}{2}x^{2\delta}(d)(u_{i+1,j+1} - 2u_{i+1,j} + u_{i+1,j-1})$$

Now that we have the two methods, we can take the average and get the Crank-Nicholson method directly.

$$u_{i+1,j} = \frac{1}{2}\left(u_{i+1,j}^{Forward} + u_{i+1,j}^{Backward}\right)$$

Other solution:

Let u_i^n represent $u(t_n, x_i)$, where $t_n = n\Delta t$, $x_i = i\Delta x$.

For the PDE that we have presented above the Crank-Nicholson formula is the following:

$$u_i^{n+1} = \frac{\Delta t}{4}\left[rx_i\left(\frac{u_{i+1}^{n+1} - u_{i-1}^{n+1}}{2\Delta x} + \frac{u_{i+1}^{n} - u_{i-1}^{n}}{2\Delta x}\right) + \frac{\sigma^2 x_i^{2\delta}}{(\Delta x)^2}\left(\frac{u_{i+1}^{n+1} - 2u_{n+1}^{i} + u_{i-1}^{n+1}}{2} + \frac{u_{i+1}^{n} - -2u_{n}^{i} + u_{i-1}^{n}}{2}\right)\right] + u_i^n$$

2 Part 2: implementation of the code

2.1 Implementing finite difference scheme

Our first task is to implement the finite difference scheme presented in the first part of our report, the Crank-Nicholson Method, in the case of put/call options.

The parameters $S_0, r, K, \sigma, \delta, X, T$ will be implemented as input variables in our function.

To implement this in Python, we defined a function that returns a grid that represents the solutions to u(t, z). That grid, denoted A is a $(n + 1)x(m + 1)$ matrix.

Our function will have the following parameters:

- Stock Price: S_0
- Time to maturity: T Number of spatial grid points: M
- Number of time steps: N
- The risk-free interest rate: r
- Volatility: σ
- Strike Price: K

And it will return:

- Option price matrix
- The grid containing the stock price
- Time Grid

You can find the code in the Appendix 3.1

2.2 Comparison with Black-Scholes Model

We now want to compare the result we get from implementing the finite difference scheme, setting $\delta = 1$ and comparing it to the Black-Scholes solution.

We will implement another function in our code (see 3.2) that calculates the Black-Scholes solution. Then we will compare the two solutions using the function *max _ error*. This function computes the number of discrepancies between the values found with the Crank-Nicholson and Black-Scholes-Merton methods for each time step and returns the number of unmatched elements.

There are several possible sources of errors within the two methods.

Firstly, there are a few numerical errors that could be avoided while derivative of the finite difference method. The numerical discretization of the PDE can introduce errors due to finite differences, to correct this, one could reduce the spatial and temporal grid size, for example, you could try the code with the values $M = 10$ and $N = 10$, and in that case we get a lower error value. Another source of error, due to numerical handling, could be due to the handling of the boundary conditions. This can be solved by simply being more accurate in our calculations, and maybe even using other methods to find them. The choice of δ can also introduce errors, although to compare the methods we have already set $\delta = 1$, otherwise to get rid of these types of numerical errors we could test different δ, then choose the one giving the most accurate values.

Secondly, the Crank-Nicholson method can be, for certain parameter combinations, unstable, therefore it can sometimes help to choose smaller time steps.

Last, but not least, the use of the exact solution of the Black-Sholes can present numerical integration errors when it is directly implemented. Here, you can choose other numerical methods or maybe other analytical solutions. Thinking about the source of error can help with the accuracy of the finite difference solution and bring it closer to the exact Black-Scholes solution.

Results

Setting $M = 100$ and $N = 1000$ we get the result that our error count is ≈ 45, in other words, 45 out of the 1000 time steps give different results. As seen before we could diminish the grid parameters, to get more accurate results. Setting $M = 10$ and $N = 20$ we get the error count ≈ 37.

3 Appendix

3.1 A1

```
   import numpy as np
from scipy.stats import norm
import scipy.stats as stats


def crank_nicolson_european_option(T, X, r, sigma, delta, K, M, N):
    dt = T / N
    dx = X / M
```

```
    # Create grid
    x_values = np.linspace(0, X, M + 1)
    t_values = np.linspace(0, T, N + 1)

    # Initialize matrix for option prices
    u = np.zeros((M + 1, N + 1))

    # Set initial condition for European call option
    u[:, 0] = np.maximum(x_values - K, 0)

    # Crank-Nicolson update loop
    for n in range(0, N):
        for i in range(1, M):
            # Coefficients for the update formula
            alpha = 0.25 * dt * r * x_values[i]
            beta = 0.5 * dt * sigma**2 * x_values[i]**(2 * delta) / dx**2

            # Update formula
            u[i, n+1] = (
                (dt / 4) * (alpha * (u[i+1, n+1] - u[i-1, n+1] +

                u[i+1, n] - u[i-1, n]) + beta * (u[i+1, n+1] -

                2*u[i, n+1] + u[i-1, n+1] + u[i+1, n] - 2*u[i, n] + u[i-1, n]))
                + u[i, n]
            )

    return u, x_values, t_values

# Example usage:
T = 1.0        # Time to expiration
X = 100.0      # Maximum spatial coordinate
r = 0.05       # Risk-free interest rate
sigma = 0.2    # Volatility
delta = 1.0    # Delta parameter
K = 50.0       # Option strike price
M = 100        # Number of spatial steps
N = 1000       # Number of time steps

option_prices, x_values, t_values
= crank_nicolson_european_option(T, X, r, sigma, delta, K, M, N)
```

3.2 A2

```
    # Black-Scholes solution for European call option
def black_scholes_call(S, K, T, r, sigma):
```

```
    d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    return S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)

# Parameters
S=50
T = 1.0
X = 100.0
r = 0.05
sigma = 0.2
delta = 1.0
K = 50.0
M = 100
N = 1000

# Finite difference solution
fd_solution, x_values, t_values = crank_nicolson_european_option(T, X, r, sigma, delta, K, M,

# Black-Scholes solution
bs_solution = black_scholes_call(x_values, K, T, r, sigma)

# Calculate and print the maximum absolute error
max_error = np.max(np.abs(fd_solution[:, -1] - bs_solution))

print("Maximum Absolute Error:", max_error)
```

References

https://pythonnumericalmethods.berkeley.edu/notebooks/chapter23.03-Finite-Difference-Method.html

https://georg.io/2013/12/03/Crank_Nicolson

https://colab.research.google.com/github/waltherg/georgio_fastpages/blob/master/_notebooks/2013-12-03-Crank_Nicolson.ipynb