Computational Finance Assignment 3

Misho Yanakiev, Max Meijer, Diogo Franquinho

March 22, 2021

1 Part I: Background of PDE Approach

1.1 Transformation Equations

The Black-Scholes equation reads:

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{\sigma^2}{2}S^2\frac{\partial^2 V}{\partial S^2} = rV$$

We let $X = \ln(S)$ and as such have $\frac{\partial X}{\partial S} = \frac{1}{S}$ and $\frac{\partial^2 X}{\partial S^2} = -\frac{1}{S^2}$. Using the chain rule, we can then express:

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

and the second derivative using the chain and product rule as:

$$\begin{aligned}\frac{\partial^2 V}{\partial S^2} &= \frac{\partial}{\partial S}(\frac{\partial V}{\partial X}\frac{1}{S}) \\ &= \frac{\partial V}{\partial X}\frac{\partial V}{\partial X}\frac{\partial X}{\partial S}\frac{1}{S} + \frac{\partial V}{\partial X}(-\frac{1}{S^2}) \\ &= \frac{\partial^2 V}{\partial X^2}\frac{1}{S^2} - \frac{\partial V}{\partial X}\frac{1}{S^2}\end{aligned}$$

Substituting this now into the LHS of the Black-Scholes equation, we obtain:

$$\begin{aligned}\frac{\partial S}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2} &= \frac{\partial S}{\partial t} + r\frac{\partial V}{\partial X} + \frac{1}{2}\sigma^2 S^2(\frac{\partial^2 V}{\partial X^2}\frac{1}{S^2} - \frac{\partial V}{\partial X}\frac{1}{S^2}) \\ &= \frac{\partial S}{\partial t} + (r - \frac{\sigma^2}{2})\frac{\partial V}{\partial X} + \frac{1}{2}\sigma^2\frac{\partial^2 V}{\partial X^2}\end{aligned}$$

Making use of the transformation $\frac{\partial V}{\partial t} = -\frac{\partial V}{\partial \tau}$, we get:

$$\begin{aligned}-\frac{\partial V}{\partial \tau} + (r - \frac{\sigma^2}{2})\frac{\partial V}{\partial X} + \frac{\sigma^2}{2}\frac{\partial^2 V}{\partial X^2} &= rV \\ \frac{\partial V}{\partial r} = (r - \frac{\sigma^2}{2})\frac{\partial V}{\partial X} + \frac{\sigma^2}{2}\frac{\partial^2 V}{\partial X^2} - rV \qquad (1)\end{aligned}$$

1.2 FD Approximation of FTCS

We use the first order Taylor expansion to express the time derivative as:

$$\begin{aligned}V(X, \tau + \Delta\tau) &\approx V(X, \tau) + \Delta\tau\frac{\partial V}{\partial \tau} \\ \frac{\partial V}{\partial \tau} &\approx \frac{V(X, \tau + \Delta\tau) - V(X, \tau)}{\Delta\tau}\end{aligned}$$

which can be expressed in incremental step notation with the lower indeces going over space and the upper over time as:

$$\Big(\frac{\partial V}{\partial \tau}\Big)_i^n \approx \frac{V_i^{n+1} - V_i^n}{\Delta \tau}$$

Now we expand $V(X + \Delta X, \tau)$ in a Taylor series keeping n fixed and thus omitting it from writing to obtain the derivative formula:

$$V_{i+1} = V_i + \Delta x (V_x)_i + \frac{\Delta x^2}{2}(V_{xx})_i + \mathcal{O}(\Delta x^3)$$

$$V_{i-1} = V_i - \Delta x (V_x)_i + \frac{\Delta x^2}{2}(V_{xx})_i + \mathcal{O}(\Delta x^3)$$

Subtracting the two terms $V_{i+1} - V_{i-1}$ we obtain:

$$V_{i+1} - V_{i-1} = 2\Delta x (V_x)_i + \mathcal{O}(\Delta x^3)$$

and thus

$$\Big(\frac{\partial V}{\partial X}\Big)_i^n \approx \frac{V_{i+1}^n - V_{i-1}^n}{2\Delta x}$$

On the other hand, adding the terms gives:

$$V_{i+1} + V_{i-1} = 2V_i + \Delta x^2 (V_{xx})_i + \mathcal{O}(\Delta x^3)$$

gives the formula

$$\Big(\frac{\partial^2 V}{\partial X^2}\Big)_i^n \approx \frac{V_{i+1}^n - 2V_i^n + V_{i-1}^n}{(\Delta x)^2}$$

From (1) we have that:

$$\frac{\partial V}{\partial r} = (r - \frac{\sigma^2}{2})\frac{\partial V}{\partial X} + \frac{\sigma^2}{2}\frac{\partial^2 V}{\partial X^2} - rV$$

Making the substitutions from the approximations defined above we have that:

$$\frac{V_i^{n+1} - V_i^n}{\Delta \tau} = (r - \frac{\sigma^2}{2})\frac{(V_{i+1}^n - V_{i-1}^n)}{2\Delta X} + \frac{\sigma^2}{2}\frac{(V_{i+1}^n - 2V_i^n + V_{i-1}^n)}{(\Delta x)^2} - rV_i^n$$

Rearranging to get V_i^{n+1} on the LHS of the equality we get the desired expression

$$V_i^{n+1} = V_i^n + (r - \frac{1}{2}\sigma^2)\frac{\Delta \tau}{2\Delta X}(V_{i+1}^n - V_{i-1}^n) + \frac{\sigma^2}{2}\frac{\Delta \tau}{(\Delta X)^2}(V_{i+1}^n - 2V_i^n + V_{i-1}^n) - r\Delta \tau V_i^n \tag{2}$$

1.3 Crank-Nicolson-Scheme

We first note that the Cranck-Nicholson Scheme is given by the average of the forward and backwards discretizations, so we make the following substitutions:

$$\Big(\frac{\partial V}{\partial \tau}\Big)_i^n \approx \frac{V_i^{n+1} - V_i^n}{\Delta \tau}$$

$$\Big(\frac{\partial V}{\partial X}\Big)_i^n \approx \frac{1}{2}\frac{V_{i+1}^n - V_{i-1}^n + V_{i+1}^{n+1} - V_{i-1}^{n+1}}{2\Delta X}$$

$$\Big(\frac{\partial^2 V}{\partial X^2}\Big)_i^n \approx \frac{V_{i+1}^n - 2V_i^n + V_{i-1}^n + V_{i+1}^{n+1} - 2V_i^{n+1} + V_{i-1}^{n+1}}{2(\Delta x)^2}$$

Substituting these approximations in equation (1), and rearranging to have V_i^{n+1} on the LHS we get the desired expression

$$\begin{aligned} V_i^{n+1} &= V_i^n + \Big(r - \frac{\sigma^2}{2}\Big)\frac{\Delta\tau}{4\Delta x}(V_{i+1}^n - V_{i-1}^n + V_{i+1}^{n+1} - V_{i-1}^{n+1}) \\ &+ \frac{1}{4}\frac{\sigma^2\Delta\tau}{(\Delta x)^2}(V_{i+1}^n - 2V_i^n + V_{i-1}^n + V_{i+1}^{n+1} - 2V_i^{n+1} + V_{i-1}^{n+1}) - \frac{r\Delta\tau}{2}(V_i^n + V_i^{n+1}) \end{aligned} \tag{3}$$

To show that the Crank-Nicolson BS is $\mathcal{O}(\Delta x^2)$ we write out the Taylor expansion of each term in the Crank-Nicolson derivative approximation according to:

$$f(x \pm \Delta x, \tau \pm \Delta\tau) = \sum_{0 \leqslant l,m \leqslant n,\ l+m \leqslant n} \frac{1}{l!m!}\frac{\partial^{m+l} f(x,\tau)}{\partial x^l \partial t^m}(\pm\Delta x)^l(\pm\Delta\tau)^m$$

This gives the following Taylor expressions for the Crank-Nicolson derivatives:

First order time

$$V_i^{n+1} = V_i^n + \Delta\tau(V_\tau)_i^n + \frac{\Delta\tau^2}{2}(V_{\tau\tau})_i^n + \mathcal{O}(\Delta\tau^3)$$

Rearranging we get the following:

$$\frac{V_i^{n+1} - V_i^n}{\Delta\tau} = (V_\tau)_i^n + \frac{\Delta\tau}{2}(V_{\tau\tau})_i^n + \mathcal{O}(\Delta\tau^2)$$

Therefore the approximation is going to be

$$\Big(\frac{\partial V}{\partial\tau}\Big)_i^n = (V_\tau)_i^n + \frac{\Delta\tau}{2}(V_{\tau\tau})_i^n + \mathcal{O}(\Delta\tau^2)$$

First order space

The Taylor derivative term expansions are:

$$\begin{aligned} V_{i\pm1}^n &= V_i^n \pm \Delta x(V_x)_i^n + \frac{\Delta x^2}{2}(V_{xx})_i^n \pm \frac{\Delta x^3}{3!}(V_{xxx})_i^n + \mathcal{O}(\Delta x^4) \\ V_{i\pm1}^{n+1} &= V_i^n + \Delta\tau(V_\tau)_i^n \pm \Delta x(V_x)_i^n + \frac{1}{2}\Big(\pm 2\Delta\tau\Delta x(V_{x\tau})_i^n + \Delta x^2(V_{xx})_i^n + \Delta\tau^2(V_{\tau\tau})_i^n\Big) + \mathcal{O}(\Delta x^3, \Delta\tau^3) \end{aligned}$$

for the form:

This yields:

$$\frac{V_{i+1}^n - V_{i-1}^n}{2\Delta x} = \frac{2\Delta x(V_x)_i^n + \frac{2}{3}(\Delta x)^3(V_{xxx})_i^n}{2\Delta x} + \mathcal{O}(\Delta x^3) = (V_x)_i^n + \frac{(\Delta x)^2}{3}(V_{xxx})_i^n + \mathcal{O}(\Delta x^3)$$

$$\frac{V_{i+1}^{n+1} - V_{i-1}^{n+1}}{2\Delta x} = \frac{2\Delta x(V_x)_i^n + (\Delta x\Delta\tau)(V_{xt})_i^n}{2\Delta x} + \mathcal{O}(\Delta x^3, \Delta\tau^3) = (V_x)_i^n + (\Delta\tau)(V_{x\tau})_i^n + \mathcal{O}(\Delta x^3)$$

$$\begin{aligned} \Big(\frac{\partial V}{\partial X}\Big)_i^n &= \frac{1}{2}((V_x)_i^n + \frac{\Delta x^2}{3}(V_{xxx})_i^n + (V_x)_i^n + (\Delta\tau)(V_{x\tau})_i^n) + \mathcal{O}(\Delta x^2, \Delta\tau^2) \\ &= (V_x)_i^n + \frac{\Delta x^2}{6}(V_{xxx})_i^n + \frac{(\Delta\tau)}{2}(V_{x\tau})_i^n + \mathcal{O}(\Delta x^2, \Delta\tau^2) \end{aligned}$$

Second order space

The derivative we express as a Taylor expansion here is:

$$\Big(\frac{\partial^2 V}{\partial X^2}\Big)_i^n \approx \frac{V_{i+1}^n - 2V_i^n + V_{i-1}^n + V_{i+1}^{n+1} - 2V_i^{n+1} + V_{i-1}^{n+1}}{2(\Delta x)^2}$$
$$= \frac{V_{i+1}^n - 2V_i^n + V_{i-1}^n}{2\Delta x^2} + \frac{V_{i+1}^{n+1} - 2V_i^{n+1} + V_{i-1}^{n+1}}{2(\Delta x)^2}$$

The Taylor expansions are a degree higher than for the first derivative terms. We have:

$$V_{i\pm1}^n = V_i^n \pm \Delta x(V_x)_i^n + \frac{\Delta x^2}{2}(V_{xx})_i^n \pm \frac{\Delta x^3}{3!}(V_{xxx})_i^n + \frac{\Delta x^4}{4!}(V_{xxxx})_i^n + \mathcal{O}(\Delta x^5)$$
$$V_{i\pm1}^{n+1} = V_i^n + \Delta\tau(V_\tau)_i^n \pm \Delta x(V_x)_i^n + \frac{1}{2}\Big(\pm 2\Delta\tau\Delta x(V_{x\tau})_i^n + \Delta x^2(V_{xx})_i^n + \Delta\tau^2(V_{\tau\tau})_i^n\Big) +$$
$$+ \frac{1}{6}\Big(\pm\Delta x^3(V_{xxx})_i^n + 3(\Delta x)^2(\Delta\tau)(V_{xx\tau})_i^n \pm 3(\Delta x)(\Delta\tau^2)(V_{\tau\tau x})_i^n + \Delta\tau^3(V_{\tau\tau\tau})_i^n\Big) +$$
$$+ \frac{1}{24}\Big(\Delta x^4(V_{xxxx})_i^n \pm 4(\Delta x)^3(\Delta\tau)(V_{xxx\tau})_i^n + 4(\Delta x^2)(\Delta\tau^2)V_{xx\tau\tau})_i^n \pm 4(\Delta x)(\Delta\tau)^3V_{x\tau\tau\tau})_i^n +$$
$$+ \Delta\tau^4(V_{\tau\tau\tau\tau})_i^n\Big) + \mathcal{O}(\Delta x^5, \Delta\tau^5)$$

Taking the sum of the space only expansions we get:

$$V_{i+1}^n + V_{i-1}^n = 2V_i^n + (\Delta x)^2(V_{xx})_i^n + \frac{2(\Delta x^4)}{4!}(V_{xxxx})_i^n + \mathcal{O}(\Delta x^5)$$

Putting into the first part of the CN derivative expression:

$$\frac{V_{i+1}^n + V_{i-1}^n - 2V_i^n}{2(\Delta x)^2} = \frac{(V_{xx})_i^n}{2} + \frac{(\Delta x^2)}{4!}(V_{xxxx})_i^n + \mathcal{O}(\Delta x^3)$$

For the other term we get:

$$V_{i+1}^{n+1} + V_{i-1}^{n+1} = 2V_i^n + 2\Delta\tau(V_\tau)_i^n + \Delta x^2(V_{xx})_i^n + \Delta\tau^2(V_{\tau\tau})_i^n + \Delta x^2\Delta\tau^2(V_{xx\tau})_i^n$$
$$+ \frac{\Delta\tau^3}{3}(V_{\tau\tau\tau})_i^n + \mathcal{O}(\Delta x^4, \Delta\tau^4)$$
$$\frac{V_{i+1}^{n+1} + V_{i-1}^{n+1} - 2V_i^n}{2\Delta x^2} = \frac{\Delta\tau}{\Delta x^2}(V_\tau)_i^n + \frac{1}{2}(V_{xx})_i^n + \frac{\Delta\tau^2}{\Delta x^2}V_{\tau\tau i}^{\ n} + \frac{\Delta\tau^2}{\Delta x^2}(V_{xx\tau})_i^n + \frac{\Delta\tau^3}{6}(V_{\tau\tau\tau})_i^n +$$
$$+ \frac{\Delta\tau^2}{\Delta x}(V_{\tau\tau x})_i^n + \frac{1}{24}\Big(\Delta x^2(V_{xxxx})_i^n + 4(\Delta\tau^2)V_{xx\tau\tau})_i^n \pm$$
$$+ \frac{\Delta\tau^4}{\Delta x^2}(V_{\tau\tau\tau\tau})_i^n\Big) + \mathcal{O}(\Delta x^3, \Delta\tau^5)$$

Adding now the two derivative terms, we obtain:

$$\Big(\frac{\partial^2 V}{\partial X^2}\Big)_i^n \approx (V_{xx})_i^n + \frac{(\Delta x^2)}{4!}(V_{xxxx})_i^n + \frac{\Delta\tau}{\Delta x^2}(V_\tau)_i^n + \frac{\Delta\tau^2}{\Delta x^2}V_{\tau\tau i}^{\ n} + \frac{\Delta\tau^2}{\Delta x^2}(V_{xx\tau})_i^n + \frac{\Delta\tau^3}{6}(V_{\tau\tau\tau})_i^n +$$
$$+ \frac{\Delta\tau^2}{\Delta x}(V_{\tau\tau x})_i^n + \frac{1}{24}\Big(2\Delta x^2(V_{xxxx})_i^n + 4(\Delta\tau^2)V_{xx\tau\tau})_i^n +$$
$$+ 2\frac{\Delta\tau^4}{\Delta x^2}(V_{\tau\tau\tau\tau})_i^n\Big) + \mathcal{O}(\Delta x^3, \Delta\tau^5)$$

Black-Scholes

Having expressed all CN derivatives via the Taylor expansion, we now substitute them into the Black-Scholes equation according to:

$$\frac{\partial V}{\partial r} = (r - \frac{\sigma^2}{2})\frac{\partial V}{\partial X} + \frac{\sigma^2}{2}\frac{\partial^2 V}{\partial X^2} - rV \rightarrow \Big(\frac{\partial V}{\partial r}\Big)_i^n = (r - \frac{\sigma^2}{2})\Big(\frac{\partial V}{\partial X}\Big)_i^n + \frac{\sigma^2}{2}\Big(\frac{\partial^2 V}{\partial X^2}\Big)_i^n - rV_i^n$$

Let $a = (r - \frac{\sigma^2}{2}),\ b = \frac{\sigma^2}{2}$ and substitute in the Taylor expressions:

$$\begin{aligned}(V_\tau)_i^n + \frac{\Delta\tau}{2}(V_{\tau\tau})_i^n + \mathcal{O}(\Delta\tau^2) = a\Bigg((V_x)_i^n + \frac{\Delta\tau}{2}(V_{xx})_i^n + \frac{\Delta x^2}{3!}(V_{xxx})_i^n + \mathcal{O}(\Delta x^2, \Delta\tau^2)\Bigg) + \\ + b\Bigg((V_{xx})_i^n + \frac{(\Delta x^2)}{4!}(V_{xxxx})_i^n + \frac{\Delta\tau}{\Delta x^2}(V_\tau)_i^n + \\ + \frac{\Delta\tau^2}{\Delta x^2}V_{\tau\tau}{}_i^n + \frac{\Delta\tau^2}{\Delta x^2}(V_{xx\tau})_i^n + \frac{\Delta\tau^3}{6}(V_{\tau\tau\tau})_i^n + \\ + \frac{\Delta\tau^2}{\Delta x}(V_{\tau\tau x})_i^n + \frac{1}{24}\Big(2\Delta x^2(V_{xxxx})_i^n + 4(\Delta\tau^2)V_{xx\tau\tau})_i^n + \\ + 2\frac{\Delta\tau^4}{\Delta x^2}(V_{\tau\tau\tau\tau})_i^n\Big) + \mathcal{O}(\Delta x^3, \Delta\tau^5)\Bigg) - \\ - rV_i^n\end{aligned}$$

We recover:

$$\begin{aligned}(V_\tau)_i^n = a(V_x)_i^n + b(V_{xx})_i^n - rV_i^n + \\ \overset{(1)}{+}\ a\Bigg(\frac{\Delta\tau}{2}(V_{xx})_i^n + \frac{\Delta x^2}{3!}(V_{xxx})_i^n + \mathcal{O}(\Delta x^3, \Delta\tau^2)\Bigg) + \\ \overset{(2)}{+}\ b\Bigg(\frac{(\Delta x^2)}{4!}(V_{xxxx})_i^n + \frac{\Delta\tau}{\Delta x^2}(V_\tau)_i^n + \\ + \frac{\Delta\tau^2}{\Delta x^2}V_{\tau\tau}{}_i^n + \frac{\Delta\tau^2}{\Delta x^2}(V_{xx\tau})_i^n + \frac{\Delta\tau^3}{6}(V_{\tau\tau\tau})_i^n + \\ + \frac{\Delta\tau^2}{\Delta x}(V_{\tau\tau x})_i^n + \frac{1}{24}\Big(2\Delta x^2(V_{xxxx})_i^n + 4(\Delta\tau^2)V_{xx\tau\tau})_i^n + \\ + 2\frac{\Delta\tau^4}{\Delta x^2}(V_{\tau\tau\tau\tau})_i^n\Big) + \mathcal{O}(\Delta x^3, \Delta\tau^5)\Bigg) - \\ \overset{(3)}{-}\ (V_\tau)_i^n - \frac{\Delta\tau}{2}(V_{\tau\tau})_i^n + \mathcal{O}(\Delta\tau^2)\end{aligned}$$

We observe that in line (1), the terms are $\mathcal{O}(\Delta x^2)$. Similarly, in line (2) we observe the same orders $\mathcal{O}(\Delta x^2)$ and finally in line (3) they are independent of Δx. Therefore, we can conclude that the Crank-Nicolson applied to the Black-Scholes equation is second order in space.

2 Part II: FD Scheme for European Call

2.1 Matrix form of FTCS

We first take the equation (2)

$$V_i^{n+1} = V_i^n + \Big(r - \frac{\sigma^2}{2}\Big)\frac{\Delta\tau}{2\Delta x}(V_{i+1}^n - V_{i-1}^n) + \frac{1}{2}\frac{\sigma^2\Delta\tau}{(\Delta x)^2}(V_{i+1}^n - 2V_i^n + V_{i-1}^n) - r\Delta\tau V_i^n$$

To simplify the calculations we take $\alpha = \Big(r - \frac{\sigma^2}{2}\Big)\frac{\Delta\tau}{2\Delta x}$, $\beta = \frac{1}{2}\frac{\sigma^2\Delta\tau}{(\Delta x)^2}$, $\rho = r\Delta\tau$.
We then get the following:

$$\begin{aligned}V_i^{n+1} &= V_i^n + \alpha(V_{i+1}^n - V_{i-1}^n) + \beta(V_{i+1}^n - 2V_i^n + V_{i-1}^n) - \rho V_i^n \\ &= (-\alpha + \beta)V_{i-1}^n + (1 - 2\beta - \rho)V_i^n + (\alpha + \beta)V_{i+1}^n\end{aligned}$$

So we get that the coefficients a_{-1}, a_0, a_1 are going to be the following:

$$\begin{aligned} a_{-1} &= (-\alpha + \beta) \\ a_0 &= (1 - 2\beta - \rho) \\ a_1 &= (\alpha + \beta) \end{aligned}$$

Everywhere apart from in the first and last row where we will evaluate the boundary conditions:

The most important boundary condition is the one at $\tau = 0$ because there the price of the option is known to be equal to the payoff. This will be the initial state for our algorithm.

We have two boundary conditions that we need to include in the matrix. First, we have the Dirichlet boundary condition where we set $V(0, \tau) = 0$ for all τ. This simply means that $V_0^n = 0$ for all n, so we set the top row of the matrix to zeros. Secondly, we have the Neumann boundary condition. This condition says something about the derivative of the option price with respect to the stock price. Since we expect that $\lim_{x \to x'} V(x, \tau) = e^{x'}$ for large x', we expect the derivative to satisfy:

$$\frac{\partial V}{\partial X}|_{x = x_{\max}} = e^{x_{\max}}$$

Replacing the partial derivative by its discrete approximation, we obtain:

$$\frac{V_{N+1}^n - V_{N-1}^n}{2\Delta x} = e^{x_{\max}}$$

Therefore we find that:

$$V_{N+1}^n = 2\Delta x e^{x_{\max}} + V_{N-1}^n$$

Now we can substitute that in the equation we found earlier:

$$\begin{aligned} V_N^{n+1} &= (-\alpha + \beta)V_{N-1}^n + (1 - 2\beta - \rho)V_N^n + (\alpha + \beta)V_{N+1}^n \\ &= (-\alpha + \beta)V_{N-1}^n + (1 - 2\beta - \rho)V_N^n + (\alpha + \beta)(2\Delta x e^{x_{\max}} + V_{N-1}^n) \\ &= (-\alpha + \beta + \alpha + \beta)V_{N-1}^n + (1 - 2\beta - \rho)V_N^n + (\alpha + \beta) \cdot 2\Delta x e^{x_{\max}} \\ &= 2\beta V_{N-1}^n + (1 - 2\beta - \rho)V_N^n + (\alpha + \beta) \cdot 2\Delta x e^{x_{\max}} \end{aligned}$$

so the entries of the matrix at the bottom row are 2β and $1 - 2\beta - \rho$ and the constant vector contains the coefficient $(\alpha + \beta) \cdot 2\Delta x e^{x_{\max}}$ at the bottom row.

We note that the matrix B is the Identity matrix, so the coefficients b_{-1}, b_0, b_1 are going to be the following:

$$\begin{aligned} b_{-1} &= 0 \\ b_0 &= 1 \\ b_1 &= 0 \end{aligned}$$

So the FTCS scheme can be represented in matrix form as

$$B\vec{V}^{n+1} = A\vec{V}^n + \vec{k}$$

Where

$$A = \begin{pmatrix} 0 & 0 & & & \\ \beta-\alpha & 1-\beta-\rho & \beta+\alpha & & \\ & \ddots & \ddots & \ddots & \\ & & \beta-\alpha & 1-2\beta-\rho & \beta+\alpha \\ & & & 2\beta & 1-2\beta-\rho \end{pmatrix}$$

$$B = \begin{pmatrix} 1 & 0 & & & \\ 0 & 1 & 0 & & \\ & \ddots & \ddots & \ddots & \\ & & 0 & 1 & 0 \\ & & & 0 & 1 \end{pmatrix}$$

$$\vec{k} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ (\alpha+\beta)2\Delta x e^{x_{max}} \end{pmatrix}$$

2.2 Matrix form of Crank-Nicolson Method

We take equation (3)

$$V_i^{n+1} = V_i^n + \left(r - \frac{\sigma^2}{2}\right)\frac{\Delta\tau}{4\Delta x}(V_{i+1}^n - V_{i-1}^n + V_{i+1}^{n+1} - V_{i-1}^{n+1}) + \frac{1}{4}\frac{\sigma^2\Delta\tau}{(\Delta x)^2}(V_{i+1}^n - 2V_i^n + V_{i-1}^n + V_{i+1}^{n+1} - 2V_i^{n+1} + V_{i-1}^{n+1}) - \frac{r\Delta\tau}{2}(V_i^n + V_i^{n+1})$$

To simplify the calculations we take the same variables as previously $\alpha = \left(r - \frac{\sigma^2}{2}\right)\frac{\Delta\tau}{2\Delta x}$, $\beta = \frac{1}{2}\frac{\sigma^2\Delta\tau}{(\Delta x)^2}$, $\rho = r\Delta\tau$.
We then get the following:

$$V_i^{n+1} = V_i^n + \frac{\alpha}{2}(V_{i+1}^n - V_{i-1}^n + V_{i+1}^{n+1} - V_{i-1}^{n+1}) + \frac{\beta}{2}(V_{i+1}^n - 2V_i^n + V_{i-1}^n + V_{i+1}^{n+1} - 2V_i^{n+1} + V_{i-1}^{n+1}) - \frac{\rho}{2}(V_i^n + V_i^{n+1})$$

Rearranging, we get

$$(\frac{\alpha}{2} - \frac{\beta}{2})V_{i-1}^{n+1} + (1 + \beta + \frac{\rho}{2})V_i^{n+1} + (-\frac{\alpha}{2} - \frac{\beta}{2})V_{i+1}^{n+1} = (-\frac{\alpha}{2} + \frac{\beta}{2})V_{i-1}^n + (1 - \beta - \frac{\rho}{2})V_i^n + (\frac{\alpha}{2} + \frac{\beta}{2})V_{i+1}^n \quad (4)$$

So we get that the coefficients a_{-1}, a_0, a_1 are going to be the following:

$$a_{-1} = (-\frac{\alpha}{2} + \frac{\beta}{2})$$
$$a_0 = (1 - \beta - \frac{\rho}{2})$$
$$a_1 = (\frac{\alpha}{2} + \frac{\beta}{2})$$

While the coefficients b_{-1}, b_0, b_1 are going to be the following:

$$b_{-1} = (\frac{\alpha}{2} - \frac{\beta}{2})$$
$$b_0 = (1 + \beta + \frac{\rho}{2})$$
$$b_1 = (-\frac{\alpha}{2} - \frac{\beta}{2})$$

For the boundary conditions, we use equation (4), for $i = N$

$$(\frac{\alpha}{2} - \frac{\beta}{2})V_{N-1}^{n+1} + (1 + \beta + \frac{\rho}{2})V_N^{n+1} + (-\frac{\alpha}{2} - \frac{\beta}{2})V_{N+1}^{n+1} = (-\frac{\alpha}{2} + \frac{\beta}{2})V_{N-1}^n + (1 - \beta - \frac{\rho}{2})V_N^n + (\frac{\alpha}{2} + \frac{\beta}{2})V_{N+1}^n$$

We can perform the same substitution as before, namely $V_{N+1}^n = 2\Delta x e^{x_{\max}} + V_{N-1}^n$:

$$\begin{aligned}&(\frac{\alpha}{2} - \frac{\beta}{2})V_{N-1}^{n+1} + (1 + \beta + \frac{\rho}{2})V_N^{n+1} + (-\frac{\alpha}{2} - \frac{\beta}{2})(2\Delta x e^{x_{\max}} + V_{N-1}^{n+1})\\ &= (-\frac{\alpha}{2} + \frac{\beta}{2})V_{N-1}^n + (1 - \beta - \frac{\rho}{2})V_N^n + (\frac{\alpha}{2} + \frac{\beta}{2})(2\Delta x e^{x_{\max}} + V_{N-1}^n)\end{aligned}$$

Rearranging we get:

$$-\beta V_{N-1}^{n+1} + (1 + \beta + \frac{\rho}{2})V_N^{n+1} = \beta V_{N-1}^n + (1 - \beta - \frac{\rho}{2})V_N^n + 2(\alpha + \beta)(\Delta x e^{x_{\max}})$$

So we get that for A the coefficients at the bottom row are β and $1 - \beta - \frac{1}{2}\rho$. Furthermore, the constant vector that is added will contain $2(\alpha + \beta)\Delta x e^{x_{\max}}$ at the bottom row. For B the coefficients on the bottom row will be $-\beta$ and $1 + \beta + \rho/2$.
So the Crank-Nicholson scheme can be represented in matrix form as

$$B\vec{V}^{n+1} = A\vec{V}^n + \vec{k}$$

Where

$$A = \begin{pmatrix} 0 & 0 & 0 & & \\ -\frac{\alpha}{2} + \frac{\beta}{2} & 1 - \beta - \rho/2 & \alpha/2 + \beta/2 & & \\ & \ddots & \ddots & \ddots & \\ & & -\frac{\alpha}{2} + \frac{\beta}{2} & 1 - \beta - \rho/2 & \alpha/2 + \beta/2 \\ & & 0 & \beta & 1 - \beta - \frac{1}{2}\rho \end{pmatrix}$$

$$B = \begin{pmatrix} 1 & 0 & 0 & & \\ \frac{\alpha}{2} - \frac{\beta}{2} & 1 + \beta + \rho/2 & -\alpha/2 - \beta/2 & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{\alpha}{2} - \frac{\beta}{2} & 1 + \beta + \rho/2 & -\alpha/2 - \beta/2 \\ & & 0 & -\beta & 1 + \beta + \frac{1}{2}\rho \end{pmatrix}$$

$$\vec{k} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 2(\alpha + \beta)\Delta x e^{x_{\max}} \end{pmatrix}$$

2.3 Stability analysis FTCS Method

From above we have that in the FTCS method the value of V_i^{n+1} is going to be given by:

$$V_i^{n+1} = (-\alpha + \beta)V_{i-1}^n + (1 - 2\beta - \rho)V_i^n + (\alpha + \beta)V_{i+1}^n$$

Where $\alpha = \left(r - \frac{\sigma^2}{2}\right)\frac{\Delta\tau}{2\Delta x}$, $\beta = \frac{1}{2}\frac{\sigma^2\Delta\tau}{(\Delta x)^2}$, $\rho = r\Delta\tau$, the same as above.

We note that the discretization error ϵ_i^n also follows the same equation such that

$$\epsilon_i^{n+1} = (-\alpha + \beta)\epsilon_{i-1}^n + (1 - 2\beta - \rho)\epsilon_i^n + (\alpha + \beta)\epsilon_{i+1}^n$$

The stability criteria is such that:

$$G = \frac{\epsilon_i^{n+1}}{\epsilon_i^n}, |G| \leqslant 1$$

Since the error tends to grow or decay exponentially with time, it is reasonable to assume that the amplitude varies exponentially with time; hence we make the substitution

$$\epsilon_m(x,t) = \epsilon_i^n = e^{at}e^{ik_m x}$$

substituting:

$$e^{a(t+\Delta t)}e^{ik_m\Delta x} = (-\alpha+\beta)e^{a(t)}e^{ik_m(x-\Delta x)} + (1-2\beta-\rho)e^{a(t)}e^{ik_m\Delta(x)} + (\alpha+\beta)e^{a(t)}e^{ik_m(x+\Delta x)}$$

Dividing by $e^{at}e^{ik_m\Delta x}$ on both sides of the equation we get

$$e^{a\Delta t} = (1-2\beta-\rho) + (-\alpha+\beta)e^{-ik_m\Delta x} + (\alpha+\beta)e^{ik_m\Delta x}$$

Rearranging

$$e^{a\Delta x} = (1-\rho) + \beta(e^{ik_m\Delta x} + e^{-ik_m\Delta x} - 2) + \alpha(e^{ik_m\Delta x} - e^{-iK_m\Delta x})$$

Making use of the following trigonometric equalities

$$2i\sin(k_m\Delta x) = e^{ik_m\Delta x} - e^{-ik_m\Delta x}$$

$$\sin^2(\frac{k_m\Delta x}{2}) = -\frac{e^{ik_m\Delta x} + e^{-ik_m\Delta x} - 2}{4}$$

Therefore we get as a final formula:

$$e^{a\Delta t} = 1 - \rho - 4\beta\sin^2(\frac{k_m\Delta x}{2}) + 2\alpha i\sin(k_m\Delta x)$$

We also note that, by assumption

$$G = \frac{\epsilon_i^{n+1}}{\epsilon_i^n} = e^{a\Delta t}$$

Thus

$$|G| \leqslant 1 \iff |1 - \rho - 4\beta\sin^2(\frac{k_m\Delta x}{2}) + 2\alpha i\sin(k_m\Delta x)| \leqslant 1$$

Making use of the absolute value for imaginary numbers we have that:

$$\sqrt{(1-\rho-4\beta\sin^2(\frac{k_m\Delta x}{2}))^2 + 4\alpha^2\sin^2(k_m\Delta x)} \leqslant 1 \iff$$
$$(1-\rho-4\beta\sin^2(\frac{k_m\Delta x}{2}))^2 + 4\alpha^2\sin^2(k_m\Delta x) \leqslant 1$$

Note that $\sin^2$ takes values in $[0,1]$ so a sufficient condition for the above is for the following three inequalities to hold:

$$(1-\rho-4\beta)^2 \leqslant 1$$
$$(1-\rho)^2 + 4\alpha^2 \leqslant 1$$
$$(1-\rho-4\beta)^2 + 4\alpha^2 \leqslant 1$$

For our choice of Δx and $\Delta\tau$, we evaluated the values on the left hand side numerically and we saw that they were all less than 1 so we can conclude that in that case the FTCS method should indeed be stable, which was indeed the case judging from the fact that we obtained accurate results. In the general case, this problem can be solved by means of Linear Programming as it is a set of 3 equations with 3 unknowns subjected to the same constraint.

2.4 Stability analysis Crank-Nicolson method

From above we have that in the Crank-Nicolson method the value of V_i^{n+1} is going to be given by:

$$V_i^{n+1} = V_i^n + \frac{\alpha}{2}(V_{i+1}^n - V_{i-1}^n + V_{i+1}^{n+1} - V_{i-1}^{n+1}) + \frac{\beta}{2}(V_{i+1}^n - 2V_i^n + V_{i-1}^n + V_{i+1}^{n+1} - 2V_i^{n+1} + V_{i-1}^{n+1}) - \frac{\rho}{2}(V_i^n + V_i^{n+1})$$

Where $\alpha = \left(r - \frac{\sigma^2}{2}\right)\frac{\Delta\tau}{2\Delta x}$, $\beta = \frac{1}{2}\frac{\sigma^2\Delta\tau}{(\Delta x)^2}$, $\rho = r\Delta\tau$, the same as above.

We note that the discretization error ϵ_i^n also follows the same equation such that

$$\epsilon_i^{n+1} = \epsilon_i^n + \frac{\alpha}{2}(\epsilon_{i+1}^n - \epsilon_{i-1}^n + \epsilon_{i+1}^{n+1} - \epsilon_{i-1}^{n+1}) + \frac{\beta}{2}(\epsilon_{i+1}^n - 2\epsilon_i^n + \epsilon_{i-1}^n + \epsilon_{i+1}^{n+1} - 2\epsilon_i^{n+1} + \epsilon_{i-1}^{n+1}) - \frac{\rho}{2}(\epsilon_i^n + \epsilon_i^{n+1})$$

The stability criteria is such that:

$$G = \frac{\epsilon_i^{n+1}}{\epsilon_i^n}, |G| \leqslant 1$$

Rearraging:

$$(\frac{\alpha}{2} - \frac{\beta}{2})\epsilon_{i-1}^{n+1} + (1 + \beta + \frac{\rho}{2})\epsilon_i^{n+1} + (-\frac{\alpha}{2} - \frac{\beta}{2})\epsilon_{i+1}^{n+1} = (-\frac{\alpha}{2} + \frac{\beta}{2})\epsilon_{i-1}^n + (1 - \beta - \frac{\rho}{2})\epsilon_i^n + (\frac{\alpha}{2} + \frac{\beta}{2})\epsilon_{i+1}^n$$

Since the error tends to grow or decay exponentially with time, it is reasonable to assume that the amplitude varies exponentially with time; hence we make the substitution

$$\epsilon_m(x,t) = \epsilon_i^n = e^{at}e^{ik_m x}$$

substituting:

$$(\frac{\alpha}{2} - \frac{\beta}{2})e^{a(t+\Delta t)}e^{ik_m(x-\delta x)} + (1 + \beta + \frac{\rho}{2})e^{a(t+\Delta t)}e^{ik_m(x)} + (-\frac{\alpha}{2} - \frac{\beta}{2})e^{a(t+\Delta t)}e^{ik_m(x+\delta x)} =$$
$$(-\frac{\alpha}{2} + \frac{\beta}{2})e^{a(t)}e^{ik_m(x-\Delta x)} + (1 - \beta - \frac{\rho}{2})e^{a(t)}e^{ik_m x} + (\frac{\alpha}{2} + \frac{\beta}{2})e^{a(t)}e^{ik_m(x+\Delta x)}$$

Dividing by $e^{at}e^{ik_m\Delta x}$ on both sides of the equation we get

$$(\frac{\alpha}{2} - \frac{\beta}{2})e^{a\Delta t}e^{-ik_m\Delta x} + (1 + \beta + \frac{\rho}{2})e^{a\Delta t} + (-\frac{\alpha}{2} - \frac{\beta}{2})e^{a\Delta t}e^{ik_m\Delta x} = (-\frac{\alpha}{2} + \frac{\beta}{2})e^{-ik_m\Delta x} + (1 - \beta - \frac{\rho}{2}) + (\frac{\alpha}{2} + \frac{\beta}{2})e^{ik_m\Delta x}$$

Solving for $e^{a\Delta t}$ we get the following:

$$e^{a\Delta t} = \frac{(-\frac{\alpha}{2} + \frac{\beta}{2})e^{-ik_m\Delta x} + (1 - \beta - \frac{\rho}{2}) + (\frac{\alpha}{2} + \frac{\beta}{2})e^{ik_m\Delta x}}{(\frac{\alpha}{2} - \frac{\beta}{2})e^{-ik_m\Delta x} + (1 + \beta + \frac{\rho}{2}) + (-\frac{\alpha}{2} - \frac{\beta}{2})e^{ik_m\Delta x}}$$

$$e^{a\Delta t} = \frac{(\frac{\alpha}{2})(e^{ik_m\Delta x} - e^{-ik_m\Delta x}) + (\frac{\beta}{2})(e^{ik_m\Delta x} + e^{-ik_m\Delta x} - 2) + (1 - \frac{\rho}{2})}{-(\frac{\alpha}{2})(e^{ik_m\Delta x} - e^{-ik_m\Delta x}) - (\frac{\beta}{2})(e^{ik_m\Delta x} + e^{-ik_m\Delta x} - 2) + (1 + \frac{\rho}{2})}$$

Making use of the following trigonometric equalities

$$2i\sin(k_m\Delta x) = e^{ik_m\Delta x} - e^{-ik_m\Delta x}$$

$$\sin^2(\frac{k_m\Delta x}{2}) = -\frac{e^{ik_m\Delta x} + e^{-ik_m\Delta x} - 2}{4}$$

We get the following:

$$e^{a\Delta t} = \frac{\alpha i\sin(k_m\Delta x) - 2\beta\sin^2(\frac{k_m\Delta x}{2}) + (1 - \frac{\rho}{2})}{-(\alpha i\sin(k_m\Delta x) - 2\beta\sin^2(\frac{k_m\Delta x}{2})) + (1 + \frac{\rho}{2})}$$

We note that the functions $\alpha i \sin(k_m \Delta x) - 2\beta \sin^2(\frac{k_m \Delta x}{2})$ and $-(\alpha i \sin(k_m \Delta x) - 2\beta \sin^2(\frac{k_m \Delta x}{2}))$ are bounded and symmetric therefore their variation will only determine the sign of $e^{a\Delta t}$. Therefore:

$$e^{a\Delta t} = \pm \frac{(1 - \frac{\rho}{2})}{(1 + \frac{\rho}{2})}$$

We note that ρ is always positive, therefore $(1 + \frac{\rho}{2}) \geqslant (1 - \frac{\rho}{2})$, therefore we have that

$$|G| = |e^{a\Delta t}| \leqslant 1$$

Therefore we conclude that the Crank-Nicolson method is unconditionally stable.

2.5 Computing Greeks

Delta

Note that (by the chain rule):

$$\Delta = \frac{\partial V}{\partial S} = \frac{\partial V}{\partial X}\frac{\partial X}{\partial S} = \frac{\partial V}{\partial X}\frac{\partial \ln(S)}{\partial S} = \frac{\partial V}{\partial X}\frac{1}{S}$$

And we can approximate this by:

$$\Delta_i^n \approx \Big(\frac{\partial V}{\partial X}\Big)_i^n \frac{1}{S_i} = \frac{V_{i+1}^n - V_{i-1}^n}{2\Delta x} e^{-x_i}$$

Gamma

The Γ is the derivative of the Δ with respect to the stock so we apply the same approximation to get:

$$\Gamma = \frac{\Delta_{i+1}^n - \Delta_{i-1}^n}{2\Delta x} e^{-x_i}$$

2.6 Algorithm

To do the simulation numerically, we made use of matrix multiplication. To solve the equation from part 1 of this question, we cannot simply use matrix multiplication. Inverting the matrix on the left hand side is not a goo option because that would be computationally expensive and involve storing a large matrix. Instead, we made use of the fact that we are dealing with tridiagonal matrices by implementing the tridiagonal matrix algorithm which can solve the $Ax = b$ for given vectors b and tridiagonal matrices A.

2.7 Results

2.7.1 Option price

S_0	FTCS	CN	BS
100	9.642348	9.642210	9.625358
110	15.146160	15.145984	15.128591
120	21.809036	21.808867	21.788808

Table 1: Estimated option prices

We ran simulations for the option price using three different methods, the Crank-Nicolson method (CN), the Black-Scholes formula (BS) and the Forward in Time, Centered in Space method (FTCS). The results of the option price simulation are displayed in fig. 3 for the three different methods. For this experiment,

we used $\Delta x = 0.01$ and $\Delta\tau = 0.0001$ with 922 space grid points centered around $S_0 = 100$ and time ranging from 0 to 1 (in years). Additionally we have computed the option price values for $S_0 = 100, 110, 120$ in table 1 with grids centered around the respective values of S_0 and the same values of Δx and $\Delta\tau$ as before. There is only one curve visible in the graph, because the estimations using the different methods are very close to each other. In the table it can be seen that the values are not exactly the same. The CN method is slightly closer to the BS method than the FTCS method in each of the three scenarios in the table.

From the fact that the graph does not fluctuate and it matches it very closely to the Black Scholes we can also see that our choice of Δx and $\Delta\tau$ does indeed yield a stable solution (for both the CN and the FTCS).

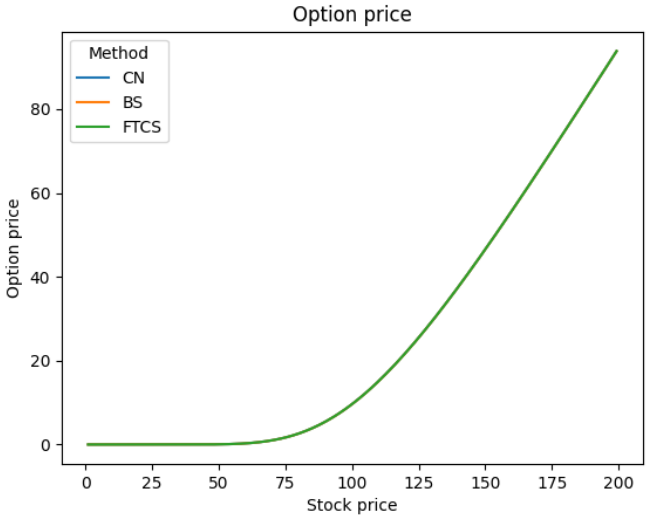

Figure 1: Option price simulated by different methods

We have also made 3D plots of the mesh grid for the CN and FTCS methods. Between the two methods the plots are indistinguishable, therefore we only present the plots of one of the methods, the Crank-Nicolson. Here we can see that at time 0 the price is equal to the payoff (zero up to the strike price and then increasing linearly with slope 1) and at maturity time. We can also see that when the stock price is close to zero the option price is also zero. The Neumann boundary condition is not as easy to check from the graph because it is about the derivative, but at least we have seen that the Dirichlet boundary condition and the initial state (i.e. the payoffs) have been incorporated correctly.

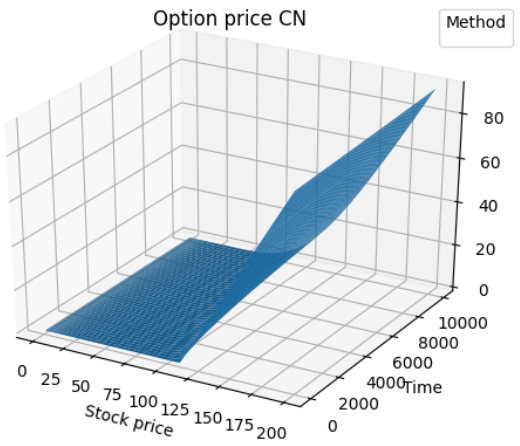

Figure 2: 3D plot of estimated option price CN method

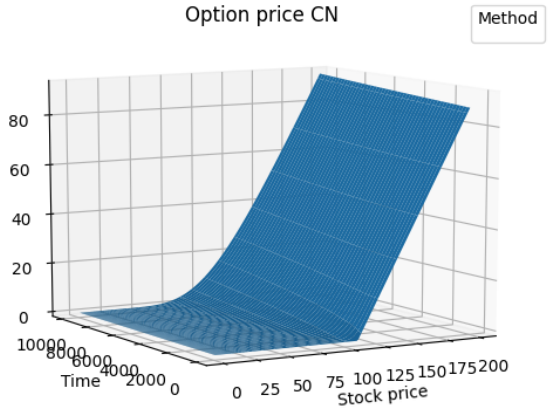

Figure 3: 3D plot of estimated option price Crank-Nicolson method

To better examine the error of the methods with respect to the Black Scholes model, we have made graphs of this as well. In fig. 4 we have plotted the error with respect to Black Scholes and in fig. 5 we have plotted the relative error with respect to Black Scholes on a log-log plot. In fig. 4 we only see a small difference between the CN and the FTCS around $S_0 = 125$ but on the remainder of the graph the difference cannot really be spotted with the naked eye. In fig. 5 we do see that the relative error is a little high for really low values of the stock price but this is also not the area we are most interested in.

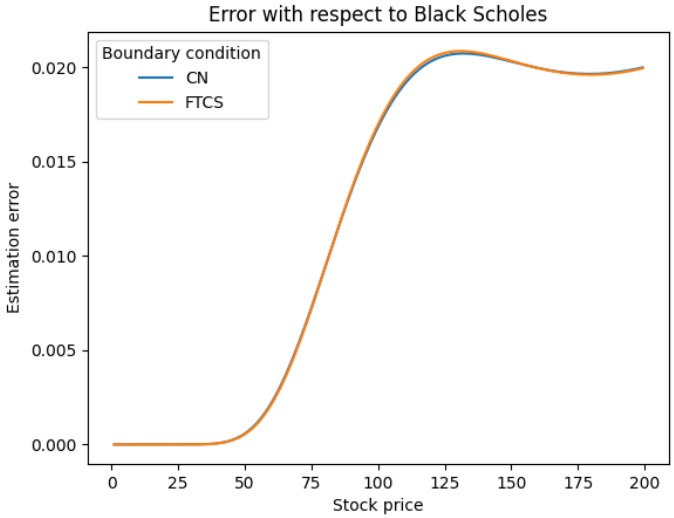

Figure 4: Error of option price estimation with respect to Black Scholes

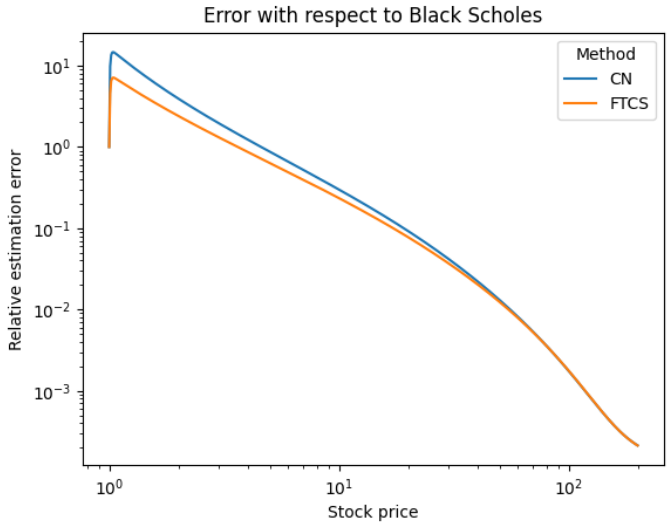

Figure 5: Relative error of option price estimation with respect to Black Scholes

2.7.2 Convergence

To examine the rate of convergence, we tried simulations with different numbers of space grid points. This is shown in fig. 6. On the x-axis we have plotted the square of the number of grid points while we kept the maximal value of the stock constant so that it can be examined whether the results are consistent with square convergence. Indeed, we see that this graph looks to be consistent with square convergence.

2.7.3 How we chose the mesh size

To choose our mesh size we tried several different values of Δx, $\Delta\tau$ and the number of grid points. We noticed that with large values of $\Delta\tau$ the FTCS method was unstable recognizable from fluctuating graphs that had large errors compared to the Black-Scholes. For CN we had stable results regardless of the value of $\Delta\tau$. However, decreasing $\Delta\tau$ did have a positive effect in reducing the error for both the FTCS and CN

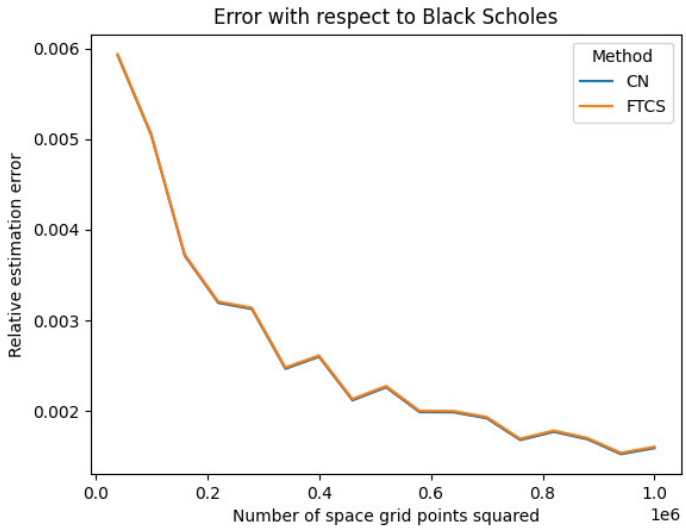

Figure 6: Relative error for different numbers of space grid points

method. The same can be seen when increasing Δx as discussed in the previous subsection.

2.7.4 Greeks

We also plotted the Delta and the Gamma based on our simulations. The results are shown in fig. 7 and fig. 8 respectively.

For the Δ we clearly see that for small values of the stock price, the Δ is also small. This is logical since when the stock price is low compared to the strike price there is only a small probability that the payoff will be nonzero and whether the stock becomes a little lower or higher will have little effect on the option price. For large stock prices it is natural that we hedge the risk of our claim by holding more stock in our portfolio, as small changes in the stock price can have a big impact on the price of the claim therefore we see that the Δ value will tend to 1 as the stock price increases. We conclude that for options with relative low strike prices, holding that option becomes approximately the same as holding the stock.

As for the Γ, we can see the expected shape since it reflects the amount of change in the Δ. As the stock price approaches the strike price, the Δ changes sharply and this change is reflected in the peak of the Γ. On the other hand, as the stock price goes well over the strike and the call option starts approximating the stock price, we can see that the Δ converges asymptotically in S to 1 and thus becomes approximately constant, resulting into Γ approaching 0 asymptotically.

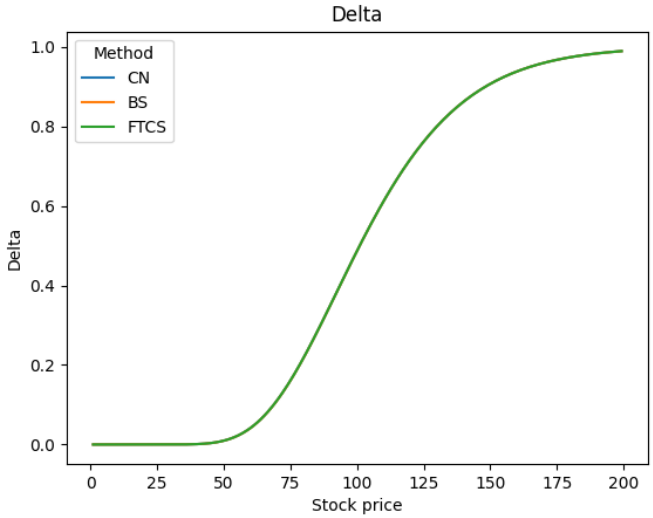

Figure 7: Estimated Δ of the option

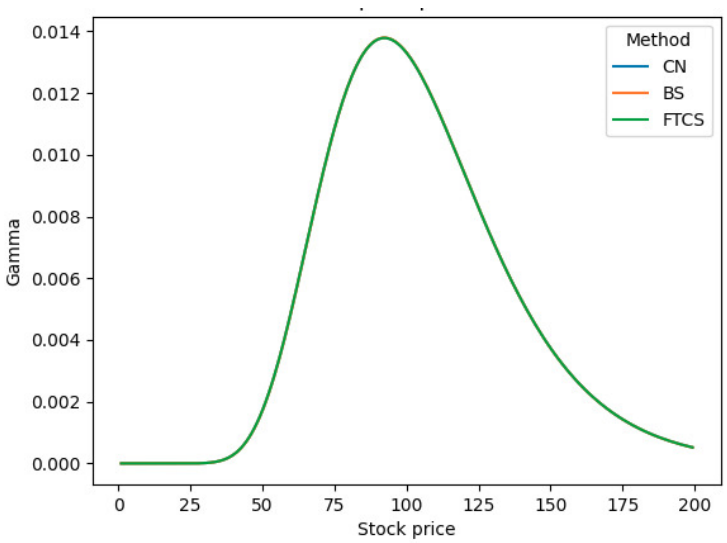

Figure 8: Estimated Γ of the option

Appendix

```
import numpy as np
from scipy.sparse import lil_matrix
import matplotlib.pyplot as plt
from math import exp, log, sqrt, sin
from scipy.stats import norm
import pandas as pd

def writeToFile(file, text):
  with open(file, 'w') as f:
    f.write(text)
```

```
def computeOptionPayoffs(stock_prices, K):
  return np.maximum(stock_prices - K, 0)

def simulate(alpha, beta, rho, X, K, T, dt, dx):
  N = len(X) - 1
  M = N+1
  V = np.zeros((T, M))

  # Stock prices
  S = np.exp(X)

  matrix = lil_matrix((M, M))
  constant = np.zeros(M)

  for i in np.arange(1,N):
    matrix[i, i-1] = beta - alpha
    matrix[i, i] = 1 - 2*beta - rho
    matrix[i, i+1] = beta + alpha

  # Boundary conditions
  k1 = 0
  k2 = 0
  k3 = matrix[-2, -3] + matrix[-2,-1]
  k4 = matrix[-2, -2]
  # First row
  matrix[0, 0] = k1
  matrix[0, 1] = k2
  # Last row
  matrix[-1, -2] = k3
  matrix[-1, -1] = k4
  # Last entry of vector gets constant
  constant[-1] = matrix[-2,-1] * 2 * dx * np.exp(X[-1])
  # At time zero the option price is the payoff
  V[0, :] = computeOptionPayoffs(S, K)
  for t in np.arange(1,T):
    V[t, :] = matrix.dot(V[t-1, :]) + constant

  return V

def BlackScholesCall(S, K, T, vol, r):
    d1=(np.log(S/K)+(r+ (vol**2 /2))*T)/ (vol*sqrt(T))
    d2=d1-vol*sqrt(T)
    Price = S*norm.cdf(d1)-K*exp(-r*(T))*norm.cdf(d2)
    return Price

def multiBlackScholes(S, K, r, T, vol):
  def forS(S):
    return BlackScholesCall(S=S, K=K, r=r, T=T, vol=vol)
  return list(map(forS, S))
def Tridiag(B, d):

  b_neg_1 = B.diagonal(-1)
  b_0 = B.diagonal(0)
```

```
    b_1 = B.diagonal(1)

    dim = len(d)
    b_neg_1_c, b_0_c, b_1_c, d_c = map(np.array, (b_neg_1, b_0, b_1, d))

    for it in range(1, dim):
        m = b_neg_1_c[it-1]/b_0_c[it-1]
        b_0_c[it] = b_0_c[it] - m*b_1_c[it-1]
        d_c[it] = d_c[it] - m*d_c[it-1]

    x = b_0_c
    x[-1] = d_c[-1]/b_0_c[-1]

    for il in range(dim-2, -1, -1):
        x[il] = (d_c[il]-b_1_c[il]*x[il+1])/b_0_c[il]

    return x


def crank_nicolson(alpha, beta, rho, X, K, T, dx):
    N = len(X) - 1
    V = np.zeros((T, N+1))
    A = lil_matrix((N+1,N+1))
    B = lil_matrix((N+1,N+1))

    # Stock prices
    S = np.exp(X)

    constant = np.zeros(N+1)

    constant[-1] = 2*(alpha + beta)*dx*np.exp(X[-1])

    for i in np.arange(1,N):
      A[i, i-1] = -0.5*(alpha - beta)
      A[i, i] = 1 - beta - rho/2
      A[i, i+1] = 0.5*(beta + alpha)
      B[i, i-1] = (alpha - beta)*0.5
      B[i, i] = (1 + beta + rho/2)
      B[i, i+1] = (-0.5)*(alpha + beta)

    #set BCs
    B[-1, -2] = - beta
    B[0, 0] = 1
    B[-1, -1] = 1 + beta + 0.5*rho
    B[0, 1] = 0
    A[0,0] = 0
    A[0, 1] = 0
    A[-1, -1] = 1 - beta - 0.5*rho
    A[-1, -2] = beta

    V[0, :] = computeOptionPayoffs(S, K)

    for t in np.arange(1,T):
      V[t, :] = Tridiag(B, (A@(V[t-1, :])) + constant)
```

```
  return V

def computeDelta(results, dx, S):
  deltas = np.zeros(results.shape)
  for t in np.arange(results.shape[0]):
    for i in np.arange(1, results.shape[1]-1):
      deltas[t,i] = (results[t,i+1]-results[t,i-1])/(2*dx)/S[i]
  return deltas

def computeGamma(results, dx, S):
  gamma = np.zeros(results.shape)
  for t in np.arange(results.shape[0]):
    for i in np.arange(1, results.shape[1]-1):
      gamma[t,i] = (results[t,i+1]-results[t,i-1])/(2*dx)
  return gamma

CN = "CN"
BS = "BS"
FTCS = "FTCS"
METHODS = [CN,BS,FTCS]
RETURN_GRID = "RETURN_GRID"
RETURN_S0 = "RETURN_S0"
RETURN_WITH_S = "RETURN_WITH_S"
RETURN_WITH_DELTA = "RETURN_WITH_DELTA"
def runSimulation(method, S_0, S_max, delta_x, delta_t, return_option =
    RETURN_GRID):
  T = int(1 / delta_t)
  N = int(x_max / delta_x)
  # Make sure N is even
  if N % 2 == 1:
    N += 1
  print(N,T)
  delta_x = x_max / (N+1)
  delta_t = 1 / T
  alpha = (r-sigma**2/2)*(delta_t/delta_x)/2
  beta = sigma**2 / 2 * delta_t / (delta_x ** 2)
  rho = r * delta_t
  X_0 = np.log(S_0)
  X_radius = np.log(S_max)-X_0
  X = np.linspace(X_0 - X_radius, X_0 + X_radius, N+1)
  S = np.exp(X)
  results = None
  if method == CN:
    results = crank_nicolson(alpha=alpha, beta=beta, rho=rho, X=X, K=K, T=T,
        dx=delta_x)
  elif method == FTCS:
    results = simulate(alpha=alpha, beta=beta, rho=rho, X=X, K=K, T=T, dt=
        delta_t, dx=delta_x)
  elif method == BS:
    results = np.tile(np.array(multiBlackScholes(S, K ,r, 1, sigma)),(T,1))
  if return_option == RETURN_GRID:
    return results
  elif return_option == RETURN_S0:
```

```
    return results[-1, N//2]
  elif return_option == RETURN_WITH_S:
    return results, S, computeDelta(results, delta_x, S), computeDelta(
        computeDelta(results, delta_x, S), delta_x, S)

sigma = 0.3
r = 0.04
S_0 = 100
K = 110
S_max = 10000
show_max = 200
delta_x = 0.01
delta_t = 0.0001
x_max = np.log(S_max)
T = int(1 / delta_t)
N = int(x_max / delta_x)
delta_x = x_max / (N+1)
delta_t = 1 / T
alpha = (r-sigma**2/2)*(delta_t/delta_x)/2
beta = sigma**2 / 2 * delta_t / (delta_x ** 2)
rho = r * delta_t
X = np.linspace(0, x_max, N+1)
S = np.exp(X)
results = runSimulation(method=FTCS, S_0=S_0, S_max=S_max, delta_x=delta_x,
    delta_t=delta_t)
results_Misho = runSimulation(method=CN, S_0=S_0, S_max=S_max, delta_x=delta_x
    , delta_t=delta_t)
results_Max = results_Misho
results_BS = runSimulation(method=BS, S_0=S_0, S_max=S_max, delta_x=delta_x,
    delta_t=delta_t)
fig = plt.figure()
ax = plt.axes(projection='3d')
W,Y = np.meshgrid(S[S < show_max], np.arange(T))
print(W.shape, Y.shape, results.shape)
ax.plot_surface(W,Y, results[:, :len(S[S < show_max])])
plt.legend(title="Method")
plt.title("Option price FTCS")
plt.xlabel("Stock price")
plt.ylabel("Time")
plt.savefig("option-price-ftcs-3d-100.png")
plt.close()
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(W,Y,results_Max[:, :len(S[S < show_max])])
plt.legend(title="Method")
plt.title("Option price CN")
plt.xlabel("Stock price")
plt.ylabel("Time")
plt.savefig("option-price-cn-3d-100.png")
plt.close()
results_dict = {}
for method in METHODS:
  results_dict[method] = runSimulation(method=method, S_0=S_0, S_max=S_max,
      delta_x=delta_x, delta_t=delta_t, return_option=RETURN_WITH_S)
```

```
for method in METHODS:
  S = results_dict[method][1]
  plt.plot(S[S < show_max], results_dict[method][0][-1, :len(S[S < show_max])
      ], label=method)
def addPricePlots():
  plt.legend(title="Method")
  plt.title("Option price")
  plt.xlabel("Stock price")
  plt.ylabel("Option price")
  plt.savefig("option-price.png")
  plt.close()
addPricePlots()
for method in METHODS:
  S = results_dict[method][1]
  plt.plot(S[S < show_max], results_dict[method][2][-1, :len(S[S < show_max])
      ], label=method)
plt.legend(title="Method")
plt.title("Delta")
plt.xlabel("Stock price")
plt.ylabel("Delta")
plt.savefig("delta.png")
plt.close()
for method in METHODS:
  S = results_dict[method][1]
  plt.plot(S[S < show_max], results_dict[method][3][-1, :len(S[S < show_max])
      ], label=method)
plt.legend(title="Method")
plt.title("Gamma")
plt.xlabel("Stock price")
plt.ylabel("Gamma")
plt.savefig("gamma.png")
plt.close()

S_0s = [100, 110, 120]
methods = [FTCS, CN, BS]
results_table = np.zeros((3,3))
for i, S_0 in enumerate(S_0s):
  for j, method in enumerate(methods):
    S_m = np.exp(np.log(S_max) - np.log(100) + np.log(S_0))
    results_table[i,j] = runSimulation(method=method, S_0=S_0, S_max=S_m,
        delta_x=delta_x, delta_t=delta_t, return_option=RETURN_S0)
writeToFile("results_table.tex", pd.DataFrame(results_table).to_latex())
for method in METHODS:
  if method != BS:
    S = results_dict[method][1]
    error = results_dict[method][0] - multiBlackScholes(S, K, r, 1, sigma)
    plt.plot(S[S < show_max], error[-1, :len(S[S < show_max])], label=method)
plt.title("Error with respect to Black Scholes")
plt.xlabel("Stock price")
plt.ylabel("Estimation error")
plt.legend(title="Boundary condition")
plt.savefig("error.png")
plt.close()
for method in METHODS:
```

```
  if method != BS:
    S = results_dict[method][1]
    results_BS =  multiBlackScholes(S, K, r, 1, sigma)
    error = results_dict[method][0] - results_BS
    relative_error = np.abs(error) / results_BS
    plt.plot(S[S < show_max], relative_error[-1, :len(S[S < show_max])], label
        =method)
plt.title("Error with respect to Black Scholes")
plt.xlabel("Stock price")
plt.xscale('log')
plt.yscale('log')
plt.ylabel("Relative estimation error")
plt.legend(title="Method")
plt.savefig("relative-error.png")
plt.close()
dxs = np.linspace(np.exp(-1)*2, (np.log(S_max)-np.log(S_0))/1000, 10)
num_gridpoints = np.linspace(200**2, 1000**2, 10)
relative_error_dict = {}
for method in METHODS:
  if method == BS:
    continue
  results = []
  for pt in num_gridpoints:
    results.append(runSimulation(method=method, S_0=S_0, S_max=S_max, delta_x
        =(x_max/np.sqrt(pt)), delta_t=delta_t, return_option=RETURN_S0))
  result_BS = BlackScholesCall(S=S_0, K=K, T=1, vol=sigma, r=r)
  error = results - result_BS
  relative_error = np.abs(error) / result_BS
  print(relative_error)
  relative_error_dict[method] = relative_error
for method in METHODS:
  if method == BS:
    continue
  plt.plot(np.sqrt(num_gridpoints), relative_error_dict[method], label=method)
plt.title("Error with respect to Black Scholes")
plt.xlabel("Number of space grid points")
plt.ylabel("Relative estimation error")
plt.legend(title="Method")
plt.savefig("relative-error-dx.png")
plt.close()
for method in METHODS:
  if method == BS:
    continue
  plt.plot(num_gridpoints, relative_error_dict[method], label=method)
plt.title("Error with respect to Black Scholes")
plt.xlabel("Number of space grid points squared")
plt.ylabel("Relative estimation error")
plt.legend(title="Method")
plt.savefig("relative-error-dx-squared.png")
plt.close()

def computeFunction():
  a = (1 - rho - 4 * beta)**2 + 4 * alpha ** 2
  b = (1- rho)**2 + 4 * alpha **2
```

```
    c = (1-rho - 4* beta)**2
    print(a,b,c)
computeFunction()
```