Black-Scholes Finite Difference Solver

This repository contains a Python implementation of a numerical solution to the **Black-Scholes PDE** for a **European call option** using the **Forward Time Central Space (FTCS)** finite difference method. The model is applied to a forward contract, where the underlying asset follows a stochastic process defined by the given stochastic differential equations.

Case Scenario

Let us suppose that the price of an asset at time t, S_t^1, evolves, under the risk-neutral measure $\mathbb{Q}$, according to the stochastic differential equation (SDE):

$$dS_t^1 \; = \; \mu \, S_t^1 \, dt \; + \; \sigma \, S_t^1 \, dB_t.$$

With:

- μ: drift term of the underlying asset,
- σ_i: volatility of the asset,
- dB_t: increment of a Brownian motion under the risk-neutral measure.

The bank account follows $dS_t^0 \; = \; r \, S_t^0 \, dt \; ; \; S_0^0 = 1$

Now, the European call option expiring at time T with strike price K on this forward contract satisfies the following partial differential equation (PDE):

$$-rC(t,F) + \frac{1}{2}\sigma^2 F^2 \frac{\partial^2 C(t,F)}{\partial F^2} = -\frac{\partial C(t,F)}{\partial t}$$

where F is the forward price at time t, and the terminal condition is given by:

$$C(T, F(T,T)) \; = \; (F(T,T) - K)^+$$

The goal of our project is to solve this PDE numerically using the finite difference method

Project Highlights

1. Change of Variables

In this project, we first aim to simplify our PDE by introducing a change of variable. We define $\tau = T - t$, which implies that as t increases, τ decreases from T to 0.

Thus, the transformed PDE becomes:

$$-rC(\tau, \mathrm{F}) + \frac{1}{2}\sigma^2 F^2 \frac{\partial^2 C(\tau, \mathrm{F})}{\partial F^2} = \frac{\partial C(\tau, \mathrm{F})}{\partial \tau}$$

Initial Conditions:

- At $\tau = 0$, $C(0, F) = max(F - K, 0)$

Boundary Conditions:

- As $F \to 0$, $C(\tau, \mathrm{F}) \to 0$ since the option value is 0 when the forward price is zero.
- As $F \to \infty$, $C(\tau, \mathrm{F}) \to C(\tau, \mathrm{F_{max}}) = \mathrm{F_{max}} - Ke^{-r\tau}$, we will use this as we know it is the exact solution, but the discounting factor can be considered negligeable in our case.

Grid Set-up:

We use a grid of points $(j\Delta F, i\Delta\tau)$ where:

- $j = 0, \ldots, \mathrm{N_F}$ (asset price grid),
- $i = 0, \ldots, \mathrm{N_\tau}$ (time steps).

We must discretize the PDE using the finite difference approach to approximate the derivatives.

2. Deriving the FTCS Finite Difference Approximation

We derive the **Forward Time, Central Space (FTCS)** finite difference approximation.

Finite difference approximation:

- For the time derivative we use a forward difference: $\frac{\partial C}{\partial \tau} \approx \frac{C_j^{i+1} - C_j^i}{\Delta\tau}$ where $C_j^i \approx C(i\Delta\tau, j\Delta F)$
- For the second spatial derivative, we will use the central difference: $\frac{\partial^2 C}{\partial F^2} \approx \frac{C_{j+1}^i - 2C_j^i + C_{j-1}^i}{(\Delta F)^2}$

Then, we substitute our approximations into the PDE, and get:

$$-rC_j^i + \frac{1}{2}\sigma^2 j^2 \frac{C_{j+1}^i - 2C_j^i + C_{j-1}^i}{(\Delta F)^2} = \frac{C_j^{i+1} - C_j^i}{\Delta\tau}$$

3. Algorithm Explanation

Grid Initialization: Create a grid for F and τ.

Initial Condition: Set the initial values for the option price at $\tau = 0$

Time-stepping Loop: Iterate through time steps and update the option prices using the FTCS approximation for each asset price F.

Boundary Conditions: Apply boundary conditions at $F = 0$ and $F = F_{max}$

4. Plot generation and Stability Investigation

To investigate the stability of our results, we will plot the graph on a 3D axis and systematically vary N_τ to observe its impact. This will be done using the parameters specified below.

Key Parameters

- **Initial asset price (S_0^1)**: 100
- **Volatility (σ)**: 0.25
- **Space steps (N_F)**: 25
- **Maximum Value of the Forward contract (F_{max})**: 500
- **Maturity (T)**: 1

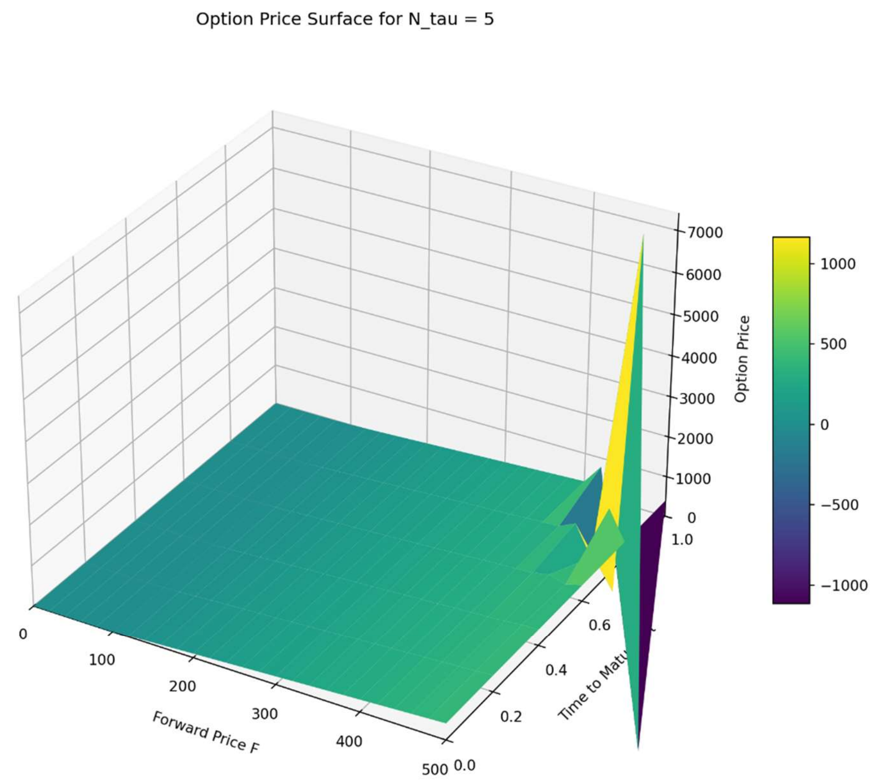
Code

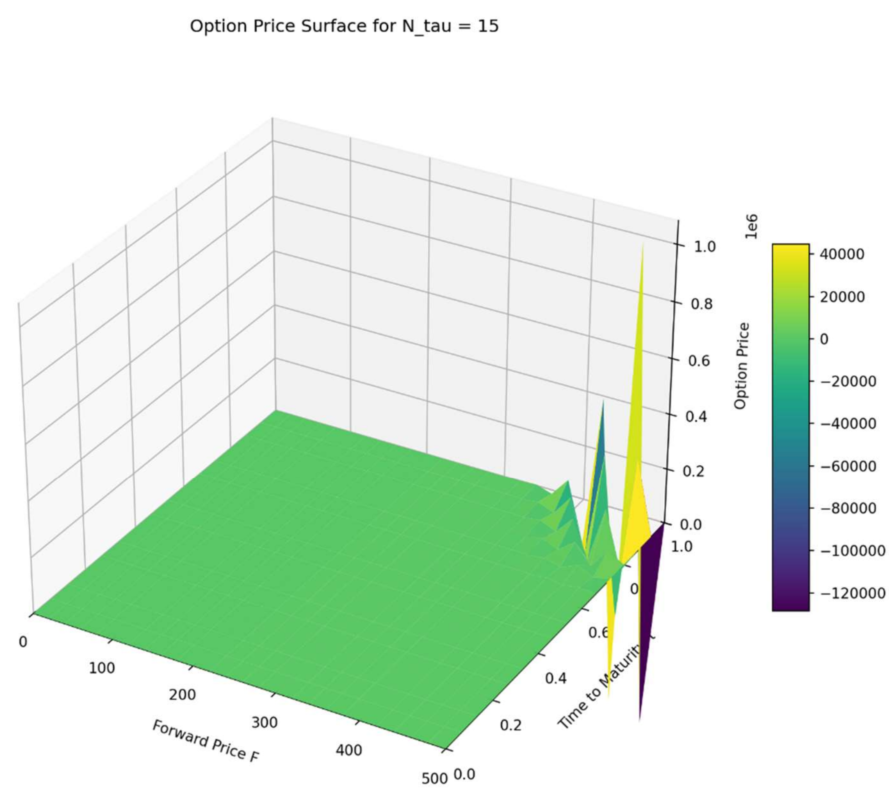
```
import numpy as np

import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D


# Parameters

S0 = 100

K = S0 * np.exp(r * T)

sigma = 0.25

r = 0.03

T = 1

NF = 25   # Number of space steps

F_max = 5 * S0   # Assume F_max is five x the initial price for boundary condi
tions

Delta_F = F_max / NF


# Grid of F and τ

F_grid = np.linspace(0, F_max, NF+1)


# FTCS Scheme Implementation

def FTCS_scheme(N_tau, Delta_tau, Delta_F, sigma, r, F_grid, K):

    C = np.maximum(F_grid - K, 0)   # Initial condition at τ=0 (maturity)
```

```
    C_all = np.zeros((N_tau+1, NF+1))  # To store the option prices at all ti
mes and space points

    # Store the initial option prices at τ=0
    C_all[0, :] = C

    for i in range(1, N_tau+1):
        C_new = C.copy()
        for j in range(1, NF):
            F_j = F_grid[j]
            C_new[j] = C[j] + Delta_tau * (0.5 * sigma**2 * j**2 * (C[j+1] -
2*C[j] + C[j-1]) - r * C[j])
        # Boundary conditions
        C_new[0] = 0
        C_new[-1] = F_grid[-1] - K * np.exp(-r * (T - (i) * Delta_tau))
        C_all[i, :] = C_new
        C = C_new

    return C_all

# Stability Investigation for Various N_tau Values
def investigate_stability(NF, Delta_F, sigma, r, F_grid, K):
    # Range of N_tau values for stability investigation
    N_tau_values = [5,15,20,25, 26,27,28,29,50,75,80,87,88,89,90,100]  # Numb
er of time steps

    for i, N_tau in enumerate(N_tau_values):
        # we adjust Delta_tau according to the number of time steps
        Delta_tau = T / N_tau

        #we will compute the option prices for the given N_tau
        option_prices = FTCS_scheme(N_tau, Delta_tau, Delta_F, sigma, r, F_gr
id, K)

        # we the mesh grid for 3D plotting
        tau_values = np.linspace(0, T, N_tau+1)
```

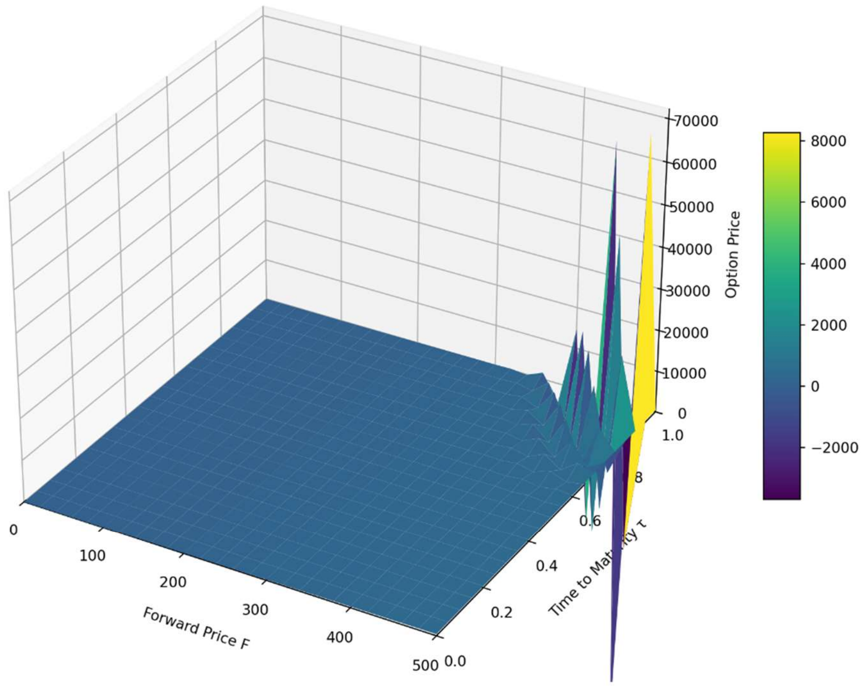
```
        F_grid_3d, tau_values_3d = np.meshgrid(F_grid, tau_values)

        # Create the 3D plot in a new figure for each N_tau
        fig = plt.figure(figsize=(12, 8), dpi=100)  # Increased size and resolution
        ax = fig.add_subplot(111, projection='3d')
        surface = ax.plot_surface(F_grid_3d, tau_values_3d, option_prices, cmap='viridis')

        # Explicitly set axis limits for clarity
        ax.set_xlim(0, F_max)
        ax.set_ylim(0, T)
        ax.set_zlim(0, np.max(option_prices) * 1.1)  # Leave a margin above the highest value

        # Labels and title
        ax.set_xlabel('Forward Price F', labelpad=10)
        ax.set_ylabel('Time to Maturity τ', labelpad=10)
        ax.set_zlabel('Option Price', labelpad=10)
        ax.set_title(f'Option Price Surface for N_tau = {N_tau}', pad=20)

        # Add a color bar for reference
        fig.colorbar(surface, ax=ax, shrink=0.5, aspect=10)

        # Optimize layout
        plt.tight_layout()
        plt.subplots_adjust(top=0.9)

        # Show the plot for this particular N_tau
        plt.show()

# Run the stability investigation
```

```
investigate_stability(NF, Delta_F, sigma, r, F_grid, K)
```

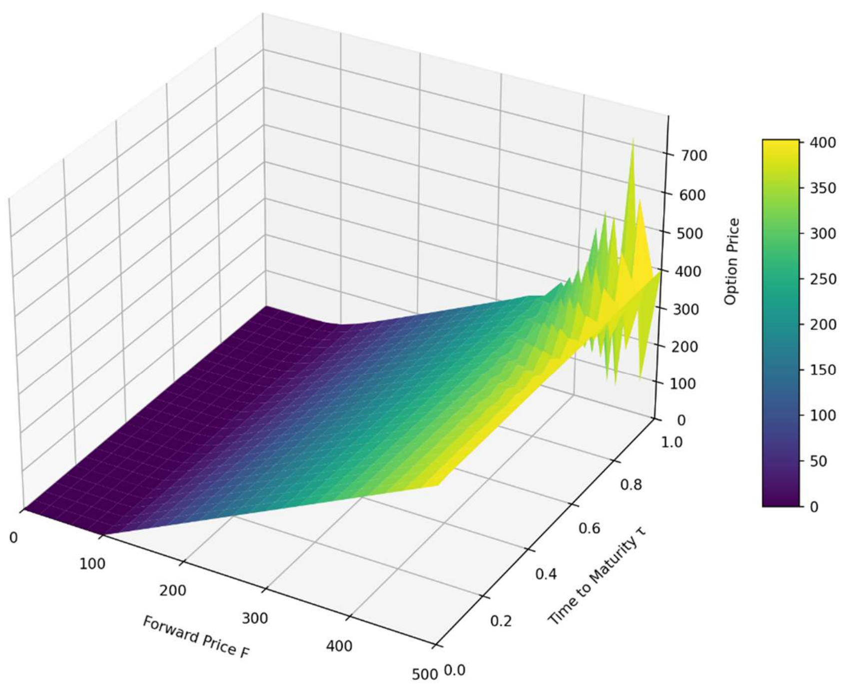




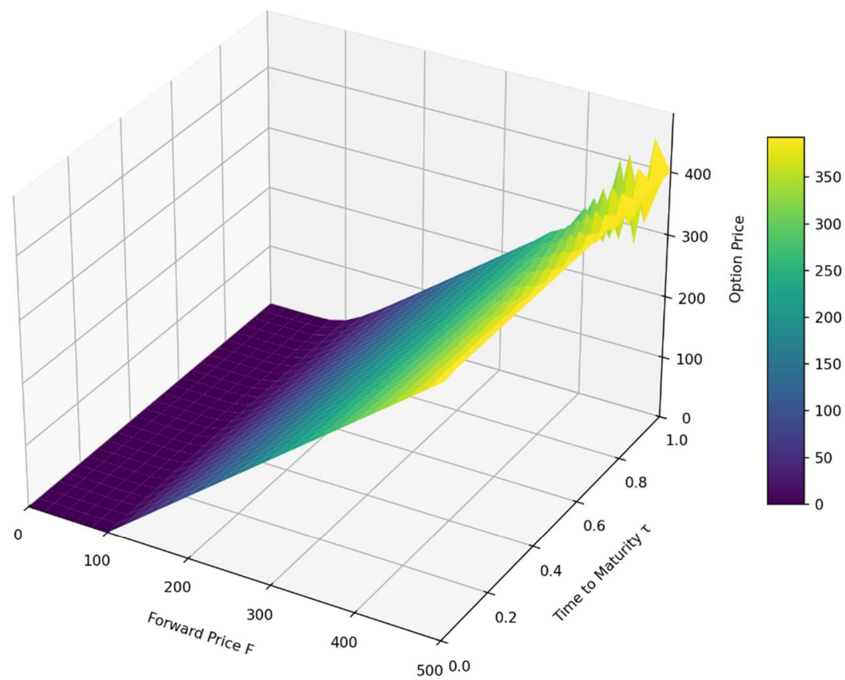
Option Price Surface for N_tau = 20



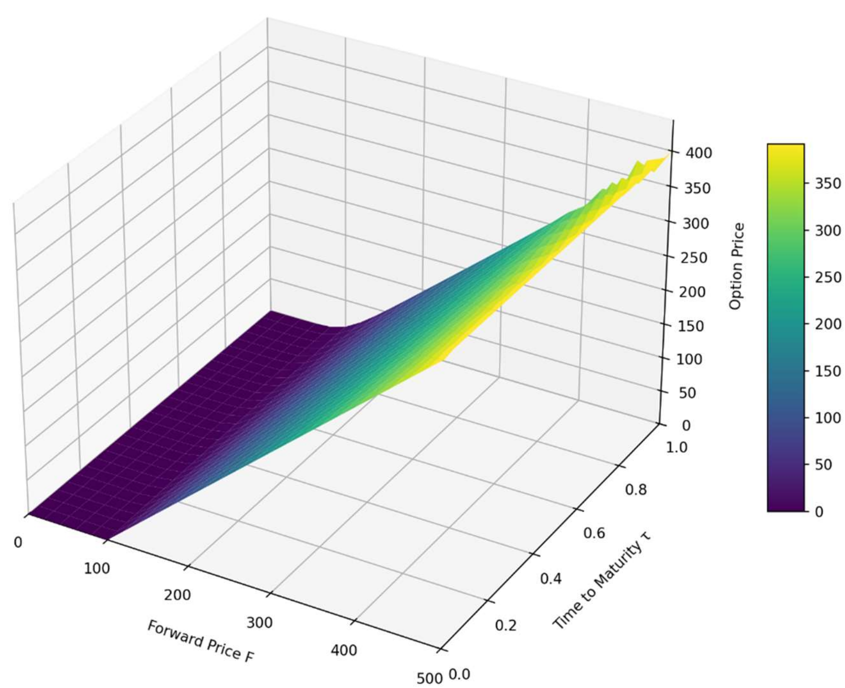
Option Price Surface for N_tau = 25



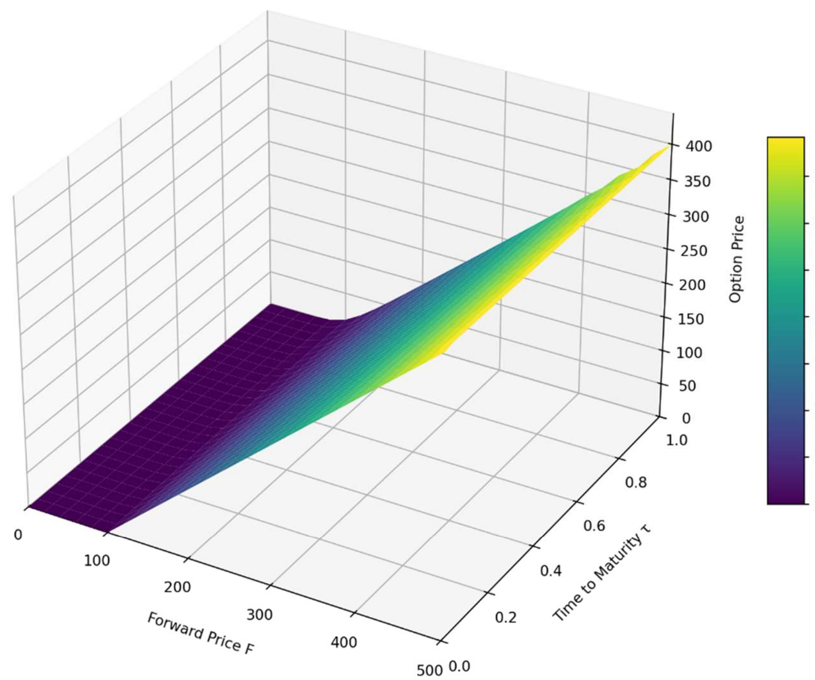
Option Price Surface for N_tau = 26



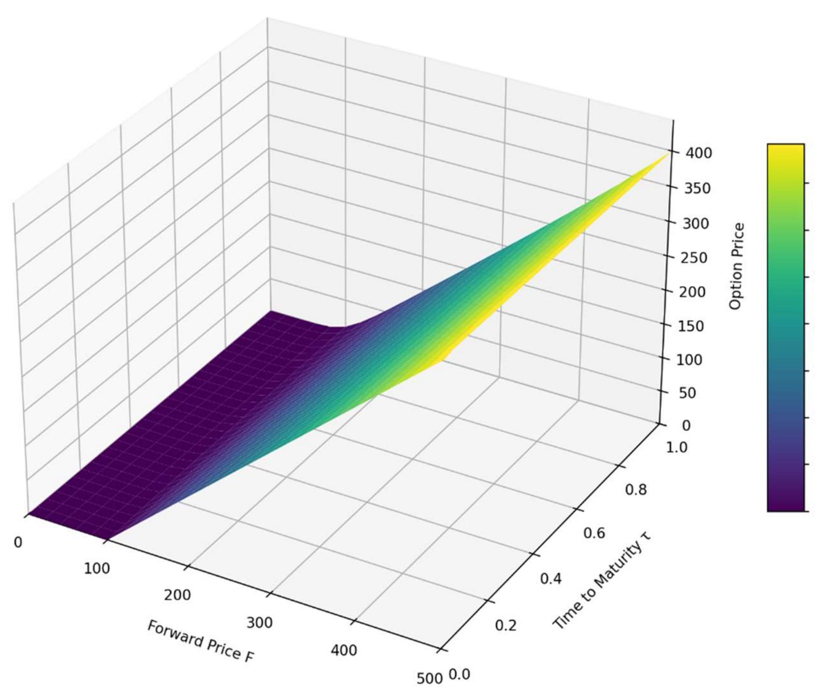
Option Price Surface for N_tau = 27



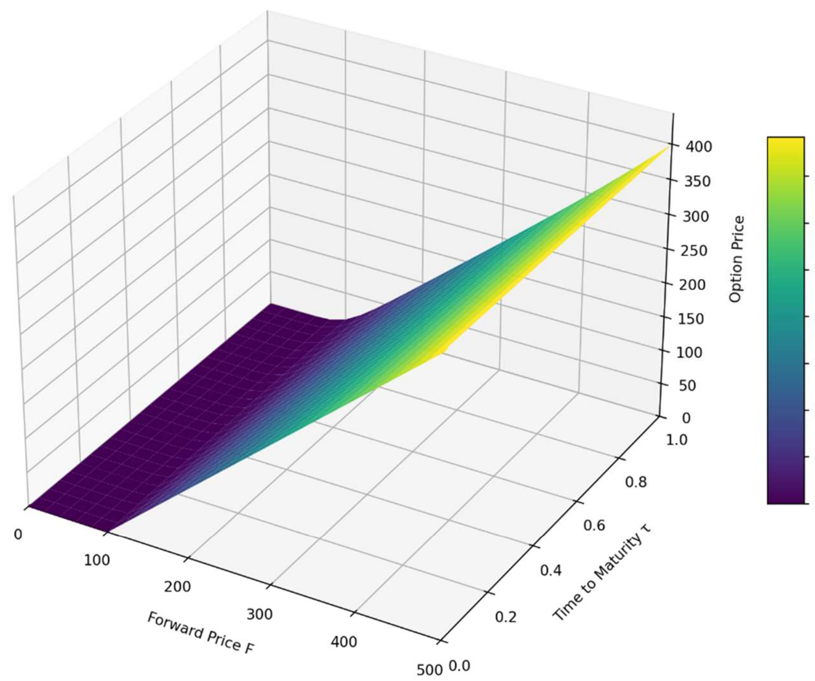
Option Price Surface for N_tau = 28



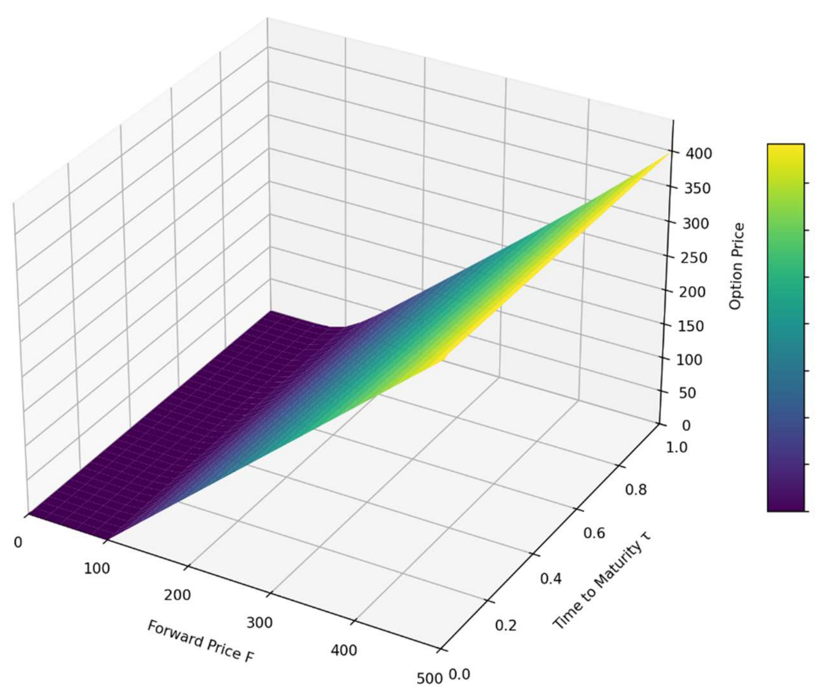
Option Price Surface for N_tau = 29



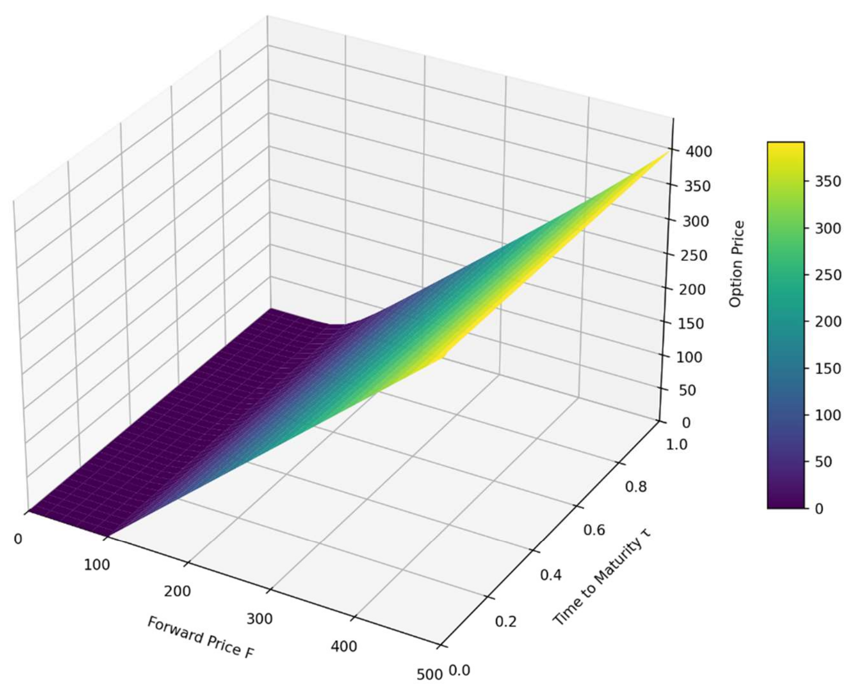
Option Price Surface for N_tau = 50



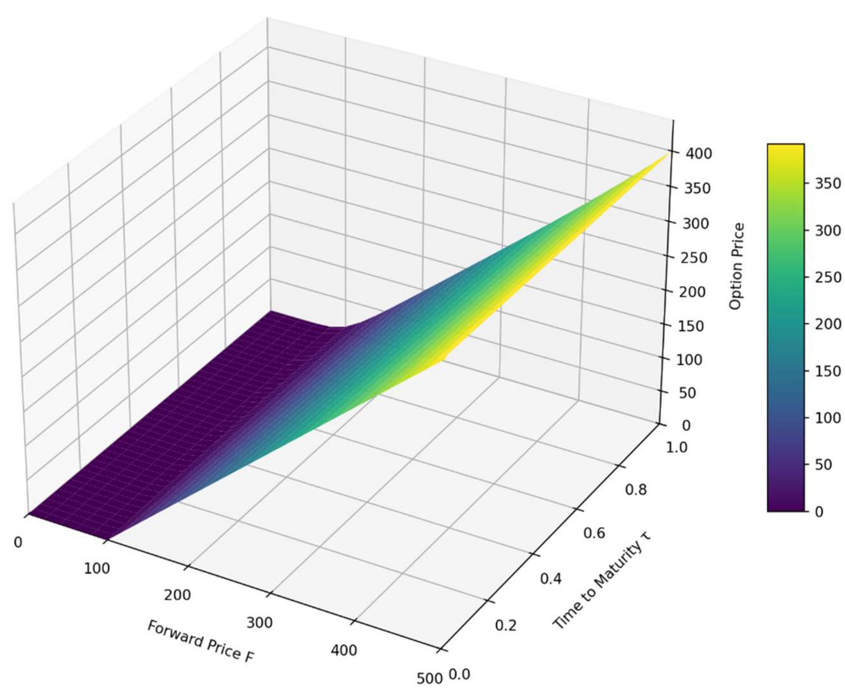
Option Price Surface for N_tau = 75



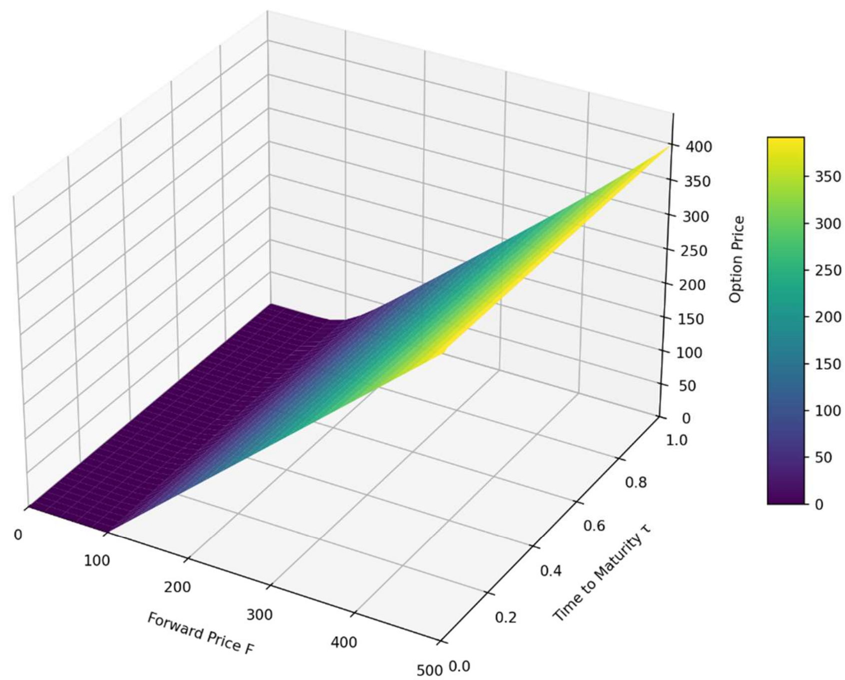
Option Price Surface for N_tau = 80



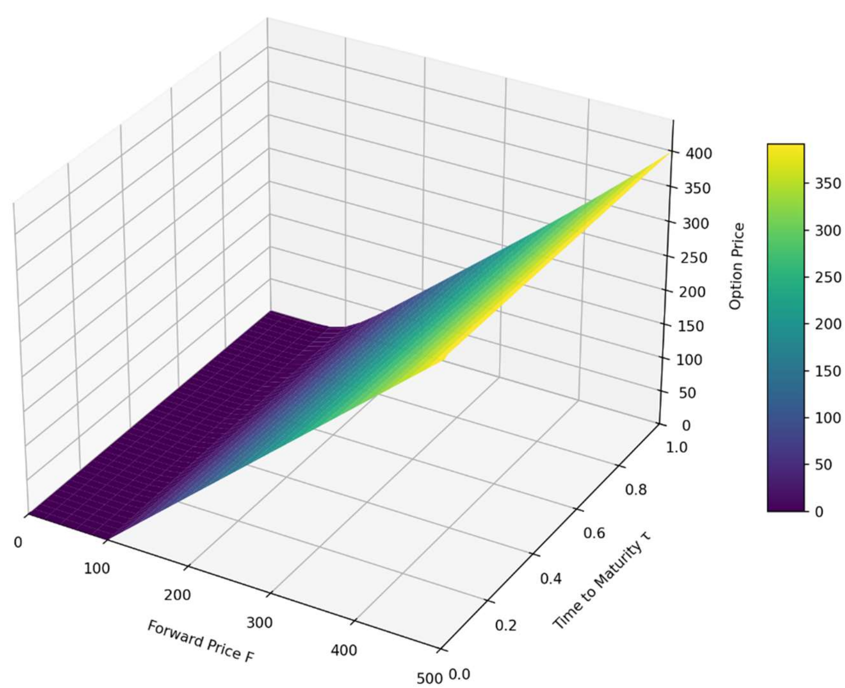
Option Price Surface for N_tau = 87

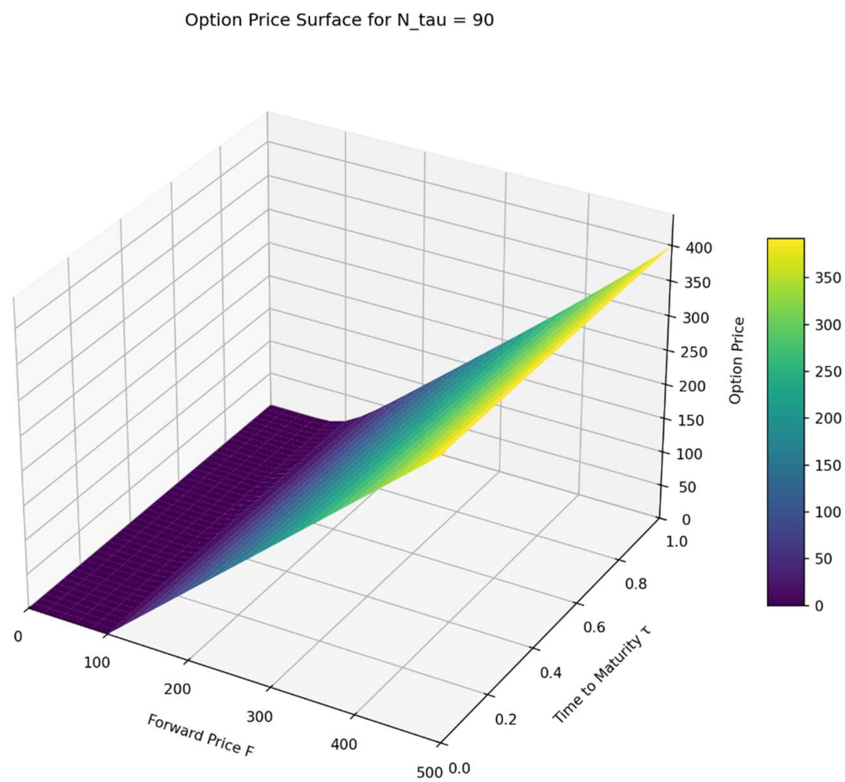


Option Price Surface for N_tau = 88



Option Price Surface for N_tau = 89





Results

In the code provided above, we observe that our algorithm stabilizes when $N_\tau \geq 28$, though it is difficult to definitively conclude this solely based on the plots. For a clearer understanding, the various graphs shown above can be referenced.

Furthermore, by using $N_F = 100$ and $N_\tau = 1000$, we find that the approximation for the price of the at-the-money futures call option, $C(0, F(0, T))$, yields a value of 9.943228. Comparing this to the true solution derived using Black's formula, $C_0 = 9.947645$, we observe that the approximation is quite accurate.

Project improvement

- To enhance this project further, one potential improvement involves transforming the partial differential equation (PDE) into a canonical form. This can be achieved by applying the following change of variables:

  $\tau = (T - t)\frac{\sigma^2}{2}, y = log\left(\frac{S}{K}\right)$

  We could further do transformations to have a canonical heat equation. This will serve as a better basis for stability and error analysis especially since we have known inequalities.