# Case 4: Numerical Methods for European Option Pricing

Asha de Meij - i6254733

November 21, 2024

## Part 1: Fourier Transforms

**Question 1.** Derive the Fourier-transform of a (tilted) call-payoff. Note that the formulas in the slides assume $r = 0$, so re-derive the correct formulas for a (tilted) Fourier-transform of a call-option with $r \neq 0$.

We know that the option price for a risk-neutral process can be written as:

$$S_T = S_0 e^{\left(r-\frac{1}{2}\sigma^2\right)T+\sigma W_T}$$

Knowing this, we can update the characteristic function. Now the characteristic function for the Brownian motion with drift becomes:

$$\Phi^*(\xi, W_t) = \mathbb{E}\left[e^{i\xi W_T}\right] = e^{i\xi W_t-\left(r+\frac{1}{2}\xi^2\right)(T-t)}$$

Hence, the payoff for the option is:

$$f_\alpha(T, W_T) = e^{-\alpha W_T} \max\left\{S_0 e^{\left(r-\frac{1}{2}\sigma^2\right)T+(\sigma-\alpha)W_T} - Ke^{-\alpha W_T}, 0\right\}$$

Now we can also rewrite the formula of the tilted payoff $f_\alpha(T, W_T)$:

$$f_\alpha(T, W_T) = e^{-\alpha W_T} \max\left\{S_0 e^{\left(r-\frac{\sigma^2}{2}\right)T+(\sigma-\alpha)W_T} - K, 0\right\}$$

The Fourier transform of the tilted payoff is:

$$f_\alpha^*(\xi) = \int_{\underline{w}}^{\infty} \left[S_0 e^{i\xi w} e^{\left(r-\frac{\sigma^2}{2}\right)T+(\sigma-\alpha)w} - Ke^{-\alpha w}\right] dw$$

This can be split into two integrals:

$$f_\alpha^*(\xi) = S_0 e^{\left(r-\frac{\sigma^2}{2}\right)T} \int_{\underline{w}}^{\infty} e^{i\xi w} e^{(\sigma-\alpha)w} dw - K \int_{\underline{w}}^{\infty} e^{i\xi w} e^{-\alpha w} dw$$

Now we can solve each integral separately:
For the first term:

$$\int_{\underline{w}}^{\infty} e^{i\xi w} e^{(\sigma-\alpha)w} dw = 0 - \frac{e^{(i\xi+\sigma-\alpha)\underline{w}}}{i\xi+\sigma-\alpha}$$

For the second term:

$$\int_{\underline{w}}^{\infty} e^{i\xi w} e^{-\alpha w} dw = 0 - \frac{e^{(i\xi-\alpha)\underline{w}}}{i\xi-\alpha}$$

By substituting the results of the integrals we get:

$$f_\alpha^*(\xi) = -S_0 e^{\left(r-\frac{\sigma^2}{2}\right)T} \frac{e^{(i\xi+\sigma-\alpha)\underline{w}}}{i\xi+\sigma-\alpha} + K\frac{e^{(i\xi-\alpha)\underline{w}}}{i\xi-\alpha}$$

Hence, the final expression for the Fourier transform is:

$$f_\alpha^*(\xi) = -S_0 \frac{e^{\left(r-\frac{\sigma^2}{2}\right)T} e^{(i\xi+\sigma-\alpha)\underline{w}}}{i\xi+\sigma-\alpha} + K\frac{e^{(i\xi-\alpha)\underline{w}}}{i\xi-\alpha}$$

Where $\underline{w}$ takes the following value:

$$S_0 e^{\left(r-\frac{\sigma^2}{2}\right)T+\sigma\underline{w}} = K$$

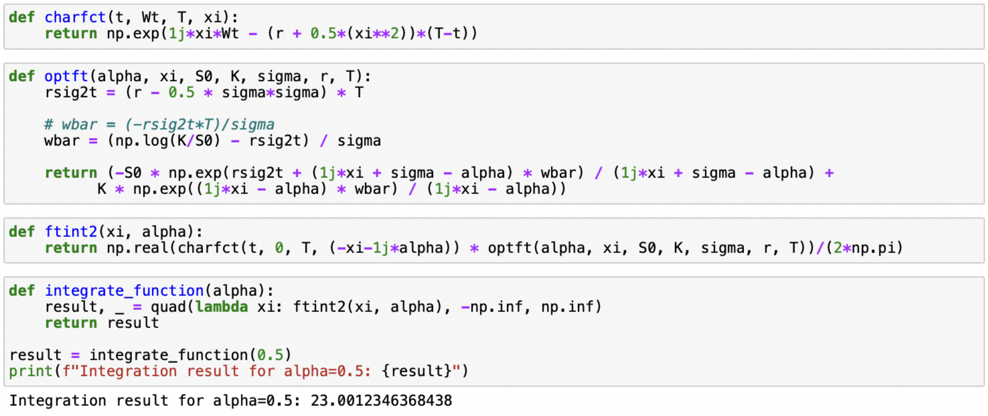
$$e^{\left(r-\frac{\sigma^2}{2}\right)T+\sigma\underline{w}} = K/S_0$$

Taking the natural logarithm of both sides, we get:

$$\left(r-\frac{\sigma^2}{2}\right)T + \sigma\underline{w} = \ln\left(\frac{K}{S_0}\right)$$

If we solve for $\underline{w}$:

$$\underline{w} = \frac{\ln\left(\frac{K}{S_0}\right) - \left(\frac{\sigma^2}{2} - r\right)T}{\sigma}$$



```
def charfct(t, Wt, T, xi):
    return np.exp(1j*xi*Wt - (r + 0.5*(xi**2))*(T-t))
```

```
def optft(alpha, xi, S0, K, sigma, r, T):
    rsig2t = (r - 0.5 * sigma*sigma) * T

    # wbar = (-rsig2t*T)/sigma
    wbar = (np.log(K/S0) - rsig2t) / sigma

    return (-S0 * np.exp(rsig2t + (1j*xi + sigma - alpha) * wbar) / (1j*xi + sigma - alpha) +
          K * np.exp((1j*xi - alpha) * wbar) / (1j*xi - alpha))
```

```
def ftint2(xi, alpha):
    return np.real(charfct(t, 0, T, (-xi-1j*alpha)) * optft(alpha, xi, S0, K, sigma, r, T))/(2*np.pi)
```

```
def integrate_function(alpha):
    result, _ = quad(lambda xi: ftint2(xi, alpha), -np.inf, np.inf)
    return result

result = integrate_function(0.5)
print(f"Integration result for alpha=0.5: {result}")
```

```
Integration result for alpha=0.5: 23.0012346368438
```

Figure 1: Tilted call-payoff implementation in Python

**Question 2.** Make a plot of the (real part of) integrand $f_\alpha^*(\xi) \cdot \phi^*(-\xi - i\alpha)$ for different values of $\alpha$ (with $\xi = 0$). Find the optimal value of $\alpha$ by identifying the minimum point in the graph. Use the Fourier inversion method to numerically integrate the price of a call-option. Use the model parameters stated above (and a "good" choice for $\alpha$).

From Figure 2, it is evident that the integrand reaches its minimum value around $\alpha = 0.5$. This is marked by the red point on the curve, which corresponds to the optimal value of $\alpha$. Deviating from this value (either increasing or decreasing $\alpha$) leads to a larger integrand.

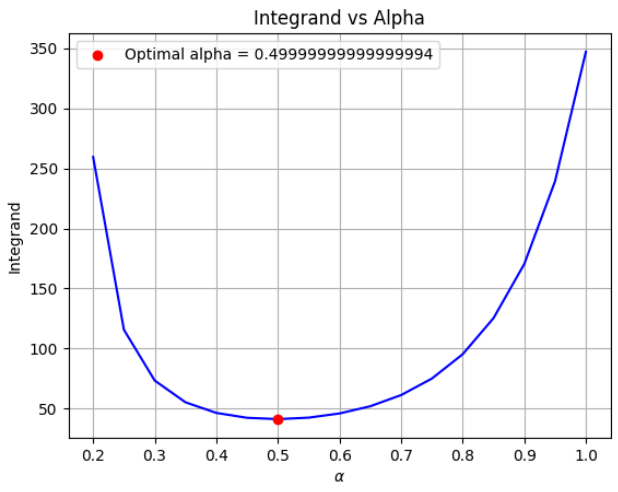


Figure 2: Integrand as a Function of Alpha

In Figure 3, the left plot shows how the integrand behaves for different values of $\alpha$. For values of $\alpha$ further from the optimal value (e.g., $\alpha = 0.2$ or $\alpha = 0.7$), the integrand exhibits more oscillations and "sharper" peaks (for smaller values of $\alpha$ the peak is wider spread out). On the right plot, which provides a zoomed-in view, we can clearly see that for values of $\alpha$ closer to the optimal $\alpha = 0.5$, the integrand is smoother and less oscillatory. This smoothness improves numerical integration performance.

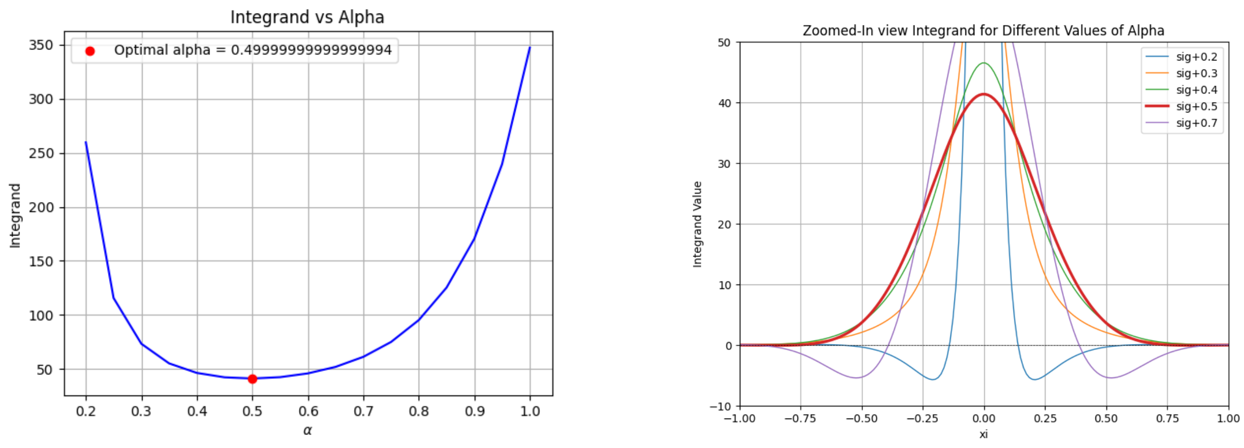


Figure 3: Integrand behavior for different values of alpha

**Question 3.** Is it necessary to use the "tilting trick" on a put-payoff $\max\{K - S_T, 0\}$? If yes, what is a necessary condition on $\alpha$ to make the Fourier integral converge?

The payoff of a put option is represented as:

$$f(T, W_T) = \max\{K - S_0 e^{(r-0.5\sigma^2)T+\sigma W_T}, 0\},$$

While the term $S_0 e^{(r-0.5\sigma^2)T+\sigma W_T}$ may grow unbounded as $W_T$ increases, the structure of the function ensures boundedness due to the *max* operator.Furthermore there is also a minus sign in front of the unbounded term. Specifically:

When $S_0 e^{(r-0.5\sigma^2)T+\sigma W_T}$ grows large, the payoff is capped at 0 because the value of $K - S_0 e^{(r-0.5\sigma^2)T+\sigma W_T}$ becomes negative, which the *max* function replaces with 0. Furthermore, the payoff cannot exceed $K$, thus, the function $f(T, W_T)$ is bounded within the range $[0, K]$.

Since $f(T, W_T)$ is not unbounded we do not need to use the "tilting trick" to make the integral diverge. The following fourier integral:

$$\int_{-\infty}^{\bar{w}} e^{i\xi w} \left(K - S_0 e^{(r-0.5\sigma^2)T+\sigma W_T}\right) dw,$$

converges naturally because the integrand, $f(T, W_T)$, is constrained and does not grow unbounded.

**Question 4.** Is it possible to use the "tilting trick" directly on the UL payoff $\max\{S_T, 100\}$?

The UL payoff $\max(S_T, 100)$ increases linearly for large values of $S_T$ and is fixed at 100 for small values of $S_T$. Because of this, the tilting trick cannot be directly applied to this payoff. The tilting trick adjusts the calculation using $e^{\alpha x}$, but:

- When $\alpha > 0$: It helps fix issues when $x \to -\infty$, but worsens divergence at $x \to \infty$ (large $S_T$).
- When $\alpha < 0$: It works for $x \to \infty$, but it can't handle the capped payoff at $x \to -\infty$.
- When $\alpha = 0$: Leaves divergence at both ends.

No single value of $\alpha$ can handle the issues at both ends of the payoff, therefore this trick isn't effective for $\max(S_T, 100)$. Breaking the payoff into different parts or using other transforms is a better way to calculate the Fourier transform.

## Part 2: Finite Difference

**Question 5.** Write a computer program to compute the price of a UL contract with an implicit FD solver. Pay careful attention to the boundary conditions for the FD grid. Not only do you need to supply values for the payoff at time $T$ (for all values of $x_T$), but you also must input boundary values for the “top” and “bottom” of the grid (for all values of $t$). The “top” and “bottom” are the largest and smallest values of the $x$-process in the grid. Hint: for $S_t \to \infty$ the value of the UL-contract approaches $S_t$, and for $S_t \to 0$ the value of the UL-contract approaches the discounted value of the guarantee.

```python
def lu_decomposition_tridiagonal(pu, pm, pd, m):

    lowerDiag = np.zeros(2 * m)
    mainDiag = np.zeros(2 * m + 1)
    upperDiag = np.zeros(2 * m)

    mainDiag[0] = pm
    for i in range(1, 2 * m + 1):
        lowerDiag[i - 1] = pd / mainDiag[i - 1]
        mainDiag[i] = pm - lowerDiag[i - 1] * pu
        if i < 2 * m:
            upperDiag[i - 1] = pu

    return lowerDiag, mainDiag, upperDiag
```

```python
def solve_lu_decomposition(lowerDiag, mainDiag, upperDiag, C, m, lambdaL, lambdaU):

    # Forward substitution (Ly = C)
    y = np.zeros(2 * m + 1)
    y[0] = C[0]
    for i in range(1, 2 * m + 1):
        y[i] = C[i] - lowerDiag[i - 1] * y[i - 1]

    y[0] -= lambdaL
    y[-1] += lambdaU

    # Back substitution (Ux = y)
    x = np.zeros(2 * m + 1)
    x[-1] = y[-1] / mainDiag[-1]
    for i in range(2 * m - 1, -1, -1):
        x[i] = (y[i] - upperDiag[i] * x[i + 1]) / mainDiag[i]

    return x
```

```python
def initialize_stock_grid(S0, dx, m):
    S = [S0 * np.exp(-m * dx)]  # from the lowest price
    for j in range(1, 2 * m + 1):
        S.append(S[j - 1] * np.exp(dx))
    return S

def initialize_payoff(S, K):
    return np.maximum(S, K)
```

```python
def implicit_FD(S0, K, T, sigma, r, q, N, m):

    dt = T / N
    dx = np.log(420)/m   # Spatial step

    # coefficients
    nu = r - q - 0.5 * sigma**2
    pu = -0.5 * dt * ((sigma / dx)**2 + nu / dx)
    pm = 1 + dt * (sigma / dx)**2 + r * dt
    pd = -0.5 * dt * ((sigma / dx)**2 - nu / dx)

    # Initialize the grid for option values and stock prices
    S = initialize_stock_grid(S0, dx, m)
    grid = initialize_payoff(S, K)

    # Boundary conditions
    lambdaU = S[2 * m] - S[2 * m - 1]  # Contract value at high prices
    lambdaL = K * np.exp(-r * dt) - S[0]  # Discounted guarantee at low prices

    # LU decomposition
    lowerDiag, mainDiag, upperDiag = lu_decomposition_tridiagonal(pu, pm, pd, m)

    # Backward induction
    for i in range(N):
        grid = solve_lu_decomposition(lowerDiag, mainDiag, upperDiag, grid, m, lambdaL, lambdaU)

    return grid[m]
```

Why is it not necessary to input ”top” and ”bottom” values for a binomial tree?

In a binomial tree model, it is unnecessary to explicitly specify the ”top” and ”bottom” values. the reason for this is that these values are already ”known” due to the defined by the up and down factors along with the number of time steps. The binomial model accounts for all potential price paths, eliminating the need to manually define the boundaries.

**Question 6.** Investigate the convergence behaviour for different choices of step-size $\Delta t$ and $\Delta x$. (Remember, with implicit FD you are free to choose any value of $\Delta t$ and $\Delta x$). For a fixed number of total nodes in the grid, is it better to choose a small $\Delta t$ or small $\Delta x$ for computing an accurate price?

To answer the question, I varied the parameters $N$ (number of time steps) and $m$ (number of spatial steps) to test the impact of different step sizes $\Delta t$ and $\Delta x$ on the accuracy of the computed option price.

```python
def convergenceAnalysis(S0, K, T, sigma, r, q, numPoints, contractPrice):
    results = []
    for N in [10, 100, 200, 500, 1000, 2000, 5000, 10000]:
        m = numPoints // N  # Keeps total grid points constant
        results.append((N, m, implicit_FD(S0, K, T, sigma, r, q, N, m),
                        abs(implicit_FD(S0, K, T, sigma, r, q, N, m) - contractPrice)))
    return results
```

The results of this experiment can be found in the table below (Table 1).

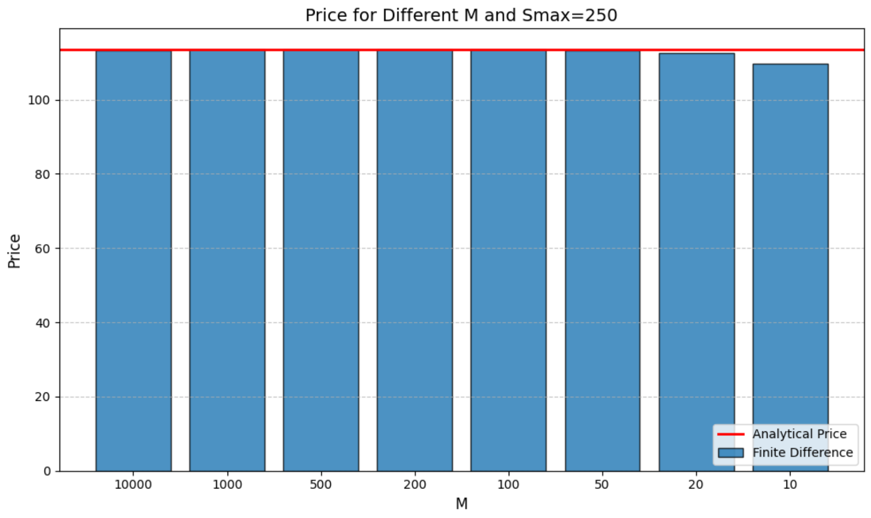
The true price of the unit-linked contract is: **113.482538**

| N | M | Price | Error |
|---|---|---|---|
| 10 | 10000 | 113.250683 | 0.234293 |
| 100 | 1000 | 113.461067 | 0.023909 |
| 200 | 500 | 113.471938 | 0.013038 |
| 500 | 200 | 113.472550 | 0.012427 |
| 1000 | 100 | 113.451735 | 0.033241 |
| 2000 | 50 | 113.359002 | 0.125974 |
| 5000 | 20 | 112.620472 | 0.864505 |
| 10000 | 10 | 109.707294 | 3.777683 |

Table 1: Results for the unit-linked contract pricing using different step sizes

Based on these results, it becomes clear that the accuracy is better when $\Delta x$ is small (i.e., when $M$ is large) compared to when $\Delta t$ is small (i.e., when $N$ is large). However, the table also shows that making $\Delta x$ excessively small can lead to an increase in error.

Hence, when the total number of grid points is fixed, the majority of resources should be allocated to making a smaller $\Delta x$ rather than unnecessarily reducing $\Delta t$ (Since a smaller $\Delta x$ improves the precision of the calculations).


Price for Different M and Smax=250
Price
M
10000
1000
500
200
100
50
20
10
0
20
40
60
80
100
Analytical Price
Finite Difference