Implied Volatility Calibration

Calibration

```
##########################################################################
########
#                           PART         A
##########################################################################
########



import numpy as np
from scipy.stats import norm

n_prime = 4
t_prime = 37/365
delta_t = t_prime/n_prime

# Volatility surface function
def implied_volatility(K):
    vol = 2.772 - 0.03797*K + 0.0002019*K**2 - (3.418 * 10**(-7))*K**3
    return vol * np.sqrt(delta_t)

# Define R as a regular function
def R(n, j, r):
    return np.exp(r*delta_t)

# Function to calculate the Black-Scholes price for a European put/call
def black_scholes(S, K, T, r, sigma, option_type="put"):
    d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(
T))
    d2 = d1 - sigma * np.sqrt(T)
    if option_type == "put":
        price = K * np.exp(-r * T) * norm.cdf(-d2) - S * norm.cdf(-d1)
```

```
    elif option_type == "call":
        price = S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    return price

# Function to compute Arrow-Debreu prices
def arrow_debreu_prob(p, R, lambda_prev):
    return p / R * lambda_prev

def construct_volatility_tree(S0=149.86 , r=0.03, steps=4):
    # Initialize stock prices, option prices, Arrow-Debreu prices, and probabilities
    stock_tree = np.ones((steps + 1, steps + 1))*0  # Stock prices
    lambda_tree = np.ones((steps + 1, steps + 1))*0 # Arrow-Debreu prices
    p_tree = np.ones((steps + 1, steps + 1))*0     # Risk-neutral probabilities

    # Set initial stock price and Arrow-Debreu price
    stock_tree[0, 0] = S0
    lambda_tree[0, 0] = 1  # Initial Arrow-Debreu price

    # Compute the steps including n = 1
    for n in range(1, steps + 1):
        if n == 1:
            # For n = 1, it is case 3
            K = stock_tree[n-1, 0]
            sigma_imp = implied_volatility(K)
            put_price_0 = black_scholes(stock_tree[n-1, 0], K, n*delta_t, r, sigma_imp, "put")

            u_0 = (stock_tree[n-1, 0] + put_price_0) / (stock_tree[n-1, 0] / R(0, 0, r) - put_price_0)
            d_0 = 1 / u_0

            stock_tree[1, 1] = u_0 * stock_tree[n-1, 0]  # S(1,1)
            stock_tree[1, 0] = d_0 * stock_tree[n-1, 0]  # S(1,0)
```

```
            if stock_tree[1, 0]>stock_tree[0, 0]*R(0,0,r):
                stocktree[1,0] = stock_tree[1, 1]*stock_tree[0, 0] / st
ock_tree[0, 0]
            else:
                if stock_tree[1, 1]<stock_tree[0, 0]*R(0,0,r):
                    stocktree[1,1] = stock_tree[1, 0]*stock_tree[0, 0]
/ stock_tree[0, 1]
                if stock_tree[1, 0]<stock_tree[0, 0]*R(0,0,r)< stock_tr
ee[1, 1]:
                    print("No adjustments necessary for n=1")




            p_00 = (R(0, 0, r) - d_0) / (u_0 - d_0)
            p_tree[0, 0] = p_00  # Store probability for S(1,1)


            lambda_tree[1, 1] = arrow_debreu_prob(p_00, R(0, 0, r), lam
bda_tree[0, 0])
            lambda_tree[1, 0] = arrow_debreu_prob(1 - p_00, R(0, 0, r),
lambda_tree[0, 0])



        elif n % 2 == 0:
            print(f"Processing even n = {n}")
            # Process j = n // 2 first -> known
            j = n // 2
            stock_tree[n, j] = stock_tree[0, 0]
            print(f"  j = {j}, stock_tree[n, j] set to {stock_tree[n, j
]}")


            # Process j >= (n // 2 + 1) and j <= n -> Case 2
            for j in range(n // 2 + 1, n + 1):
                if j > 0:  # Ensure j-1 is a valid index
                    # Set K(n) and calculate ado[n,j]
                    Kn = stock_tree[n-1, j-1]
                    sigma_imp = implied_volatility(Kn)
```

```
                    print(f"  Calculating for j = {j}, Kn = {Kn}, sigma_imp = {sigma_imp}")
                    # Calculate Arrow-Debreu price using Black-Scholes and lambda_tree
                    if j == n:
                        ado_d = black_scholes(S0, Kn, n*delta_t, r, sigma_imp, "call") / lambda_tree[n-1, j-1]
                        print(f"    ado_d (for j == n): {ado_d}")
                    else:
                        sum_c = lambda_tree[n-1, j] / R(n-1, j, r) * (stock_tree[n-1, j] * R(n-1, j, r) - stock_tree[n-1, j-1])
                        ado_d = (black_scholes(S0, Kn, n*delta_t, r, sigma_imp, "call") - sum_c) / lambda_tree[n-1, j-1]
                        print(f"    sum_c: {sum_c}, ado_d (for j != n): {ado_d}")

                    R_n1_j_1 = R(n-1, j-1, r)

                    # Compute stock price S(n, j) using ado and previous stock prices
                    if R_n1_j_1 != 0:
                        dumnum = ado_d * stock_tree[n, j-1] + stock_tree[n-1, j-1] * (stock_tree[n, j-1] / R_n1_j_1 - stock_tree[n-1, j-1])
                        dumden = ado_d + stock_tree[n, j-1] / R_n1_j_1 - stock_tree[n-1, j-1]
                        print(f"    dumnum: {dumnum}, dumden: {dumden}")
                        if dumden != 0:
                            stock_tree[n, j] = dumnum / dumden
                            print(f"    stock_tree[{n}, {j}] updated to: {stock_tree[n, j]}")

            # Process j < (n // 2) last -> Case 1
            for j in range(n // 2 -1 , -1 , -1):
                Kn = stock_tree[n-1, j]
                sigma_imp = implied_volatility(Kn)
                print(f"  Processing j < n // 2: j = {j}, Kn = {Kn}, sigma_imp = {sigma_imp}")
                if j == 0:
```

```
                    ado = black_scholes(S0, Kn, n*delta_t, r, sigma_imp, "put") / lambda_tree[n-1, j]
                    print(f"    ado (for j == 0): {ado}")
                else:
                    print(f"lambda_tree[{n-1}, {j-1}] = {lambda_tree[n-1, j-1]}")
                    print(f"R({n-1}, {j-1}, {r}) = {R(n-1, j-1, r)}")
                    print(f"stock_tree[{n-1}, {j}] = {stock_tree[n-1, j]}")
                    print(f"stock_tree[{n-1}, {j-1}] = {stock_tree[n-1, j-1]}")
                    sum_p = lambda_tree[n-1, j-1] / R(n-1, j-1, r) * (stock_tree[n-1, j] - stock_tree[n-1, j-1]* R(n-1, j-1, r))
                    ado = (black_scholes(S0, Kn, n*delta_t, r, sigma_imp, "put") - sum_p) / lambda_tree[n-1, j]
                    print(f"    sum_p: {sum_p}, ado (for j != 0): {ado}")

                R_n1_j = R(n-1, j, r)
                if j + 1 < n + 1:  # Ensure j+1 is a valid index
                    stock_next = stock_tree[n, j+1]
                    stock_prev_n1 = stock_tree[n-1, j]
                    dumnum = ado * stock_next + stock_prev_n1 * (stock_prev_n1 - stock_next / R_n1_j)
                    dumden = ado + stock_prev_n1 - stock_next / R_n1_j
                    print(f"    dumnum: {dumnum}, dumden: {dumden}")
                    if dumden != 0:
                        stock_tree[n, j] = dumnum / dumden
                        print(f"stock_tree[{n}, {j}] updated to: {stock_tree[n, j]}")

            for j in range(1,n):
              if stock_tree[n, j]<stock_tree[n-1, j]*R(n-1,j,r)< stock_tree[n, j+1]:
                  print(f"No adjustments made for n={n} , j={j},{j-1},{j+1}")
              else:
                  if stock_tree[n,j]>stock_tree[n-1,j]*R(n-1,j,r):
```

```
                      stock_tree[n,j]=stock_tree[n,j+1]*stock_tree[n-1,j]/stock_tree[n-1,j+1]
                  if stock_tree[n,j+1]<stock_tree[n-1,j]*R(n-1,j,r):
                      stock_tree[n,j+1]=stock_tree[n,j]*stock_tree[n-1,j]/stock_tree[n-1,j-1]



            for j in range(1, n+1):
              p_tree[n-1, j-1] = (R(n-1, j-1, r) * stock_tree[n-1, j-1] - stock_tree[n, j-1]) / (stock_tree[n, j] - stock_tree[n, j-1])


            for j in range(0, n+1):


              if j == 0:
                lambda_tree[n, j] = (1 - p_tree[n-1, j]) / R(n-1, j, r) * lambda_tree[n-1, j]
              elif j == n:
                lambda_tree[n, j] = p_tree[n-1, j-1] / R(n-1, j-1, r) * lambda_tree[n-1, j-1]
              else:
                lambda_tree[n, j] = (1 - p_tree[n-1, j]) / R(n-1, j, r) * lambda_tree[n-1, j] + p_tree[n-1, j-1] / R(n-1, j-1, r) * lambda_tree[n-1, j-1]


        elif n % 2 == 1:
            print(f"  Odd step n={n}")


            # Process j = (n-1) // 2 + 1 first -> Case 3


            j = (n-1) // 2 + 1
            print(f"    Processing j={j}")


            K = stock_tree[n-1, j-1]
            sigma_imp = implied_volatility(K)
            put_price_0 = black_scholes(stock_tree[0, 0], K, n*delta_t, r, sigma_imp, "put")
            print(f"    K: {K}, sigma_imp: {sigma_imp}, put_price_0: {put_price_0}")
```

```
            Sum_p = lambda_tree[n-1, j-2] * 1 / R(n-1, j-2, r) * (stock
_tree[n-1, j-1] - stock_tree[n-1, j-2] * R(n-1, j-2, r))
            print(f"    Sum_p: {Sum_p}")

            V_put_n_min1__jmin1 = (put_price_0 - Sum_p) / lambda_tree[n
-1, j-1]
            print(f"    V_put_n_min1__jmin1: {V_put_n_min1__jmin1}")

            u_n_min1__jmin1 = (stock_tree[n-1, j-1] + V_put_n_min1__jmi
n1) / (stock_tree[n-1, j-1] / R(n-1, j-1, r) - V_put_n_min1__jmin1)
            print(f"    u_n_min1__jmin1: {u_n_min1__jmin1}")

            stock_tree[n, j] = u_n_min1__jmin1 * stock_tree[n-1, j-1]
            stock_tree[n, j-1] = stock_tree[n-1, j-1] / u_n_min1__jmin1
            print(f"    Stock Tree Updated at [{n}, {j}]: {stock_tree[n
, j]}")
            print(f"    Stock Tree Updated at [{n}, {j-1}]: {stock_tree
[n, j-1]}")

            for j in range((n-1) // 2 + 2, n + 1):
               Kn = stock_tree[n-1, j-1]
               sigma_imp = implied_volatility(Kn)

               print(f"Debugging n={n}, j={j}")
               print(f"Kn (stock_tree[{n-1}, {j-1}]): {Kn}")
               print(f"Implied volatility for Kn: {sigma_imp}")

               if j == n:
                 ado_d = black_scholes(S0, Kn, n*delta_t, r, sigma_imp,
"call") / lambda_tree[n-1, j-1]
                 print(f"ado_d (for j == n): {ado_d}")
               else:
                 sum_c = lambda_tree[n-1, j] / R(n-1, j, r) * (stock_tr
ee[n-1, j] * R(n-1, j, r) - stock_tree[n-1, j-1])
                 print(f"sum_c: {sum_c}")
                 ado_d = (black_scholes(S0, Kn, n*delta_t, r, sigma_imp
, "call") / lambda_tree[n-1, j] - sum_c) / lambda_tree[n-1, j-1]
```

```
                 print(f"ado_d (for j != n): {ado_d}")

               dumnum = ado_d * stock_tree[n, j-1] + stock_tree[n-1, j-1] * (stock_tree[n, j-1] / R(n-1, j-1, r) - stock_tree[n-1, j-1])
               dumden = ado_d + stock_tree[n, j-1] / R(n-1, j-1, r) - stock_tree[n-1, j-1]

               print(f"dumnum: {dumnum}")
               print(f"dumden: {dumden}")

               if dumden != 0:
                 stock_tree[n, j] = dumnum / dumden
                 print(f"stock_tree[{n}, {j}] updated to: {stock_tree[n, j]}")
               else:
                 print(f"dumden is zero, stock_tree[{n}, {j}] not updated.")

            for j in range((n-1)//2 -1, -1, -1):
              Kn = stock_tree[n-1, j]
              sigma_imp = implied_volatility(Kn)

              print(f"Debugging n={n}, j={j}")
              print(f"Kn (stock_tree[{n-1}, {j}]): {Kn}")
              print(f"Implied volatility for Kn: {sigma_imp}")

              if j == 0:
                if lambda_tree[n-1, j] != 0:  # Ensure no division by zero
                  ado_d = black_scholes(S0, Kn, n*delta_t, r, sigma_imp, "put") / lambda_tree[n-1, j]
                  print(f"ado_d (for j == 0): {ado_d}")
              else:
                if lambda_tree[n-1, j-1] != 0 and lambda_tree[n-1, j] != 0:  # Ensure no division by zero
                  sum_p = lambda_tree[n-1, j-1] / R(n-1, j-1, r) * (stock_tree[n-1, j] - stock_tree[n-1, j-1] * R(n-1, j-1, r))
```

```
         ado_d = (black_scholes(S0, Kn, n*delta_t, r, sigma_imp, "put") - sum_p) / lambda_tree[n-1, j]
         print(f"sum_p: {sum_p}")
         print(f"ado_d (for j != 0): {ado_d}")

       # Check bounds for j+1
       if j + 1 < stock_tree.shape[1]:
        dumnum = ado_d * stock_tree[n, j+1] + stock_tree[n-1, j] * (stock_tree[n-1, j] - stock_tree[n, j+1] / R(n-1, j, r))
        dumden = ado_d + stock_tree[n-1, j] - stock_tree[n, j+1] / R(n-1, j, r)


        print(f"dumnum: {dumnum}")
        print(f"dumden: {dumden}")
        stock_tree[n, j] = dumnum / dumden
        print(f"    Stock Tree Updated at [{n}, {j}]: {stock_tree[n, j]}")


      for j in range(1,n):
        if stock_tree[n, j]<stock_tree[n-1, j]*R(n-1,j,r)< stock_tree[n, j+1]:
          print(f"No adjustments made for n={n} , j={j},{j-1},{j+1}")
        else:
          if stock_tree[n,j]>stock_tree[n-1,j]*R(n-1,j,r):
            stock_tree[n,j]=stock_tree[n,j+1]*stock_tree[n-1,j]/stock_tree[n-1,j+1]
          if stock_tree[n,j+1]<stock_tree[n-1,j]*R(n-1,j,r):
            stock_tree[n,j+1]=stock_tree[n,j]*stock_tree[n-1,j]/stock_tree[n-1,j-1]


      for j in range(1, n+1):
       p_tree[n-1, j-1] = (R(n-1, j-1, r) * stock_tree[n-1, j-1] - stock_tree[n, j-1]) / (stock_tree[n, j] - stock_tree[n, j-1])


      for j in range(0, n+1):
       if j == 0:
```

```
                lambda_tree[n, j] = (1 - p_tree[n-1, j]) / R(n-1, j, r) * lambda_tree[n-1, j]
              elif j == n:
                lambda_tree[n, j] = p_tree[n-1, j-1] / R(n-1, j-1, r) * lambda_tree[n-1, j-1]
              else:
                lambda_tree[n, j] = (1 - p_tree[n-1, j]) / R(n-1, j, r) * lambda_tree[n-1, j] + p_tree[n-1, j-1] / R(n-1, j-1, r) * lambda_tree[n-1, j-1]


    return stock_tree, lambda_tree, p_tree


# Run the volatility tree construction
stock_tree, lambda_tree, p_tree = construct_volatility_tree()
# Display the results
print("\nStock Prices Tree:")
```

```
##
## Stock Prices Tree:
```

```
print(stock_tree)
```

```
## [[149.86         0.           0.           0.           0.        ]
##  [148.45242088 151.28092535   0.           0.           0.        ]
##  [145.92325053 149.86       153.65158944   0.           0.        ]
##  [144.78684484 148.23593653 151.50185661 155.20235145   0.        ]
##  [142.37654117 146.37725459 149.86       153.39264581 157.51288667]]
```

```
print("\nArrow-Debreu Prices Tree:")
```

```
##
## Arrow-Debreu Prices Tree:
```

```
print(lambda_tree)
```

```
## [[1.         0.         0.         0.         0.        ]
##  [0.46171188 0.53752814 0.         0.         0.        ]
##  [0.15172695 0.62916491 0.21758874 0.         0.        ]
##  [0.09678017 0.34894677 0.41882058 0.13317425 0.        ]
##  [0.03578208 0.21223324 0.4077186  0.2789684  0.0622612 ]]
```

```
print("\nProbability Tree:")
```

```
##
## Probability Tree:
```

```
print(p_tree)
```

```
## [[0.53793696 0.         0.         0.         0.         ]
##  [0.6711318  0.40510294 0.         0.         0.         ]
##  [0.3616574  0.53217513 0.61251121 0.         0.         ]
##  [0.62999344 0.56605479 0.49738451 0.46787236 0.         ]
##  [0.         0.         0.         0.         0.         ]]
```

American Put: application

```
################################################################################
#                           PART         B
################################################################################
def american_put(K=150, r=0.03, steps=4):
    # Initialize the option price tree (a_put_tree)
    a_put_tree = np.zeros((steps + 1, steps + 1))  # Option prices

    # Iterate backward from maturity to step 0
    for n in range(steps, -1, -1):
        if n == steps:
            # Calculate option prices at maturity
            for j in range(n + 1):
                a_put_tree[n, j] = max(K - stock_tree[n, j], 0)
                if a_put_tree[n, j] == K - stock_tree[n, j]:
                    print(f"\na_put_tree[{n}, {j}] = K - stock_tree[{n}, {j}] = {a_put_tree[n, j]}")
        else:
            # Calculate option prices at earlier steps
            for j in range(n + 1):
                expected_value = 1 / R(n, j, r) * (p_tree[n, j] * a_put_tree[n + 1, j + 1] + (1 - p_tree[n, j]) * a_put_tree[n + 1, j])
                a_put_tree[n, j] = max(K - stock_tree[n, j], expected_value)
                if a_put_tree[n, j] == K - stock_tree[n, j]:
```

```python
                    print(f"a_put_tree[{n}, {j}] = K - stock_tree[{n}, {j}] = {a_put_tree[n, j]}")

    # Return the option price tree
    return a_put_tree

# Assuming stock_tree and p_tree are already defined in your environment

# Call the function to get the result
result = american_put()
```

```
##
## a_put_tree[4, 0] = K - stock_tree[4, 0] = 7.623458827826198
##
## a_put_tree[4, 1] = K - stock_tree[4, 1] = 3.622745408810971
##
## a_put_tree[4, 2] = K - stock_tree[4, 2] = 0.13999999999998636
## a_put_tree[3, 0] = K - stock_tree[3, 0] = 5.213155156423198
## a_put_tree[3, 1] = K - stock_tree[3, 1] = 1.7640634716194938
## a_put_tree[2, 0] = K - stock_tree[2, 0] = 4.076749467715672
```

```python
# Print the result
print("\nAmerican Put :")
```

```
##
## American Put :
```

```python
print(result)
```

```
## [[1.16683924 0.         0.         0.         0.        ]
##  [1.91779428 0.52345324 0.         0.         0.        ]
##  [4.07674947 0.86203579 0.02722467 0.         0.        ]
##  [5.21315516 1.76406347 0.07031269 0.         0.        ]
##  [7.62345883 3.62274541 0.14       0.         0.        ]]
```