finm320_24_hw2

April 12, 2024

1 Problem 1

[ ]:
```
import numpy as np
```

[2]:
```
class localvolDynamics:

    def __init__(self, S0, r, q, maxvol, localvol):
        self.S0 = S0
        self.r = r
        self.q = q
        self.maxvol = maxvol
        self.localvol = localvol
```

[3]:
```
hw2dynamics = localvolDynamics(S0 = 100, r = 0.06, q = 0.01, maxvol = 0.6,
                      localvol = lambda S,t: np.minimum(0.2+5*np.log(S/100)**2+0.
 ↪1*np.exp(-t), 0.6))

# Note that hw2dynamics.localvol is a function
# that may be invoked in the usual way, for example:
# hw2dynamics.localvol( exchangerate , time )
```

[4]:
```
class CallOnAmericanPut:

    def __init__(self, putexpiry, putstrike, callexpiry, callstrike):
        self.putexpiry = putexpiry
        self.putstrike = putstrike
        self.callexpiry = callexpiry
        self.callstrike = callstrike
```

[5]:
```
hw2contract = CallOnAmericanPut(putexpiry=0.75, putstrike=95, callexpiry=0.25,␣
 ↪callstrike=10)
```

[13]:
```
class TreeEngine:
    def __init__(self, N):
        """
        Initialize the TreeEngine with the number of time steps, N.
        """
        self.N = N
```

```
    # You complete the coding of this function

    def price_compound_localvol(self, contract:CallOnAmericanPut, dynamics:
↪localvolDynamics):
        avgvol = dynamics.localvol(dynamics.S0, 0)
        deltat = contract.putexpiry / self.N
        deltax = np.maximum(avgvol*np.sqrt(3* deltat),dynamics.maxvol *np.
↪sqrt(deltat))

        # Ensure that the call expiry can be represented in the tree grid.
        numTimestepsLeft = contract.callexpiry/contract.putexpiry * self.N
        if abs(numTimestepsLeft-round(numTimestepsLeft)) > 1e-8:
            raise ValueError("This value of N fails to enable call expiry date␣
↪to be represented in the tree")

        # Generate the grid of stock prices using exponential spacing.
        Sgrid = dynamics.S0*np.exp(np.linspace(self.N, -self.N, num=2*self.N+1,␣
↪endpoint=True)*deltax)

        # Initialize put option prices at maturity.
        optionprice_put = np.maximum(contract.putstrike-Sgrid,0)

        # Backward induction to calculate option prices at earlier times.
        for t in np.linspace(self.N-1, 0, num=self.N, endpoint=True)*deltat:
            # Adjust stock prices and recalculate local volatility and drift.
            Sgrid = Sgrid[1:-1]
            localvol = dynamics.localvol(Sgrid,t)
            nu = (dynamics.r-dynamics.q) - localvol**2/2

            # Calculate transition probabilities for up, down, and middle␣
↪movements.
            Pu = 0.5 * (((localvol**2 * deltat + (nu*deltat)**2)/deltax**2) +␣
↪(nu*deltat)/deltax)
            Pd = 0.5 * (((localvol**2 * deltat + (nu*deltat)**2)/deltax**2) -␣
↪(nu*deltat)/deltax)
            Pm = 1 - ((localvol**2 * deltat + (nu*deltat)**2)/deltax**2)

            # Update put prices considering early exercise.
            optionprice_put = np.exp(-dynamics.r*deltat) * (Pu*optionprice_put[:
↪-2] + Pd*optionprice_put[2:] + Pm*optionprice_put[1:-1])
            optionprice_boundary = np.maximum(contract.putstrike-Sgrid,0)
            optionprice_put = np.
↪where(optionprice_boundary>optionprice_put,optionprice_boundary,optionprice_put)

            # Calculation of the call on the put option.
```

```
            if abs(t-contract.callexpiry)<1e-8:
                optionprice_call_on_put = np.maximum(optionprice_put-contract.
↪callstrike,0)    #an array of time-T option prices.
            elif t < contract.callexpiry:
                optionprice_call_on_put = np.exp(-dynamics.r*deltat) *␣
↪(Pu*optionprice_call_on_put[:-2] + Pd*optionprice_call_on_put[2:] +␣
↪Pm*optionprice_call_on_put[1:-1])

        price_of_put = optionprice_put[0] #write code to compute this
        price_of_call_on_put = optionprice_call_on_put[0] #write code to␣
↪compute this

        return (price_of_put, price_of_call_on_put)
```

[14]:
```
hw2tree = TreeEngine(N=3000)  #change if necessary to get $0.01 accuracy, in␣
↪your judgment
```

[15]:
```
(answer_part_a, answer_part_b) = hw2tree.
↪price_compound_localvol(hw2contract,hw2dynamics)
```

[16]:
```
(answer_part_a, answer_part_b)
```

[16]:
```
(7.007352296510587, 1.5925531282199283)
```

2 Problem 2

2.1 Part a

Black-Scholes call price is $S_0 N(d_1) - Ke^{-rT} N(d_2)$

where

$$d_{1,2} := \frac{log(S_0 e^{rT}/K)}{\sigma\sqrt{T}} \pm \frac{\sigma\sqrt{T}}{2}$$

Since K = S0,

$$d_{1,2} := \frac{r\sqrt{T}}{\sigma} \pm \frac{\sigma\sqrt{T}}{2}$$

Using first order Taylor Expansion: $N(x) = N(0) + xN'(0)$, where $N(0) = 0.5$ and $N'(0) = \frac{1}{\sqrt{2\pi}}$

$$\Delta = N(d_1) = 0.5 + \frac{1}{\sqrt{2\pi}}(\frac{r\sqrt{T}}{\sigma} + \frac{\sigma\sqrt{T}}{2})$$

Plug in $\sigma = 0.2$, $T = 0.25$, $r = 0.01$ we get

$$\Delta \approx 0.5 + 0.075 \times \frac{1}{\sqrt{2\pi}} \approx 0.5 + 0.075 \times 0.3989422804014337 = 0.5 + 0.029921$$

Therefore, the approximate delta of the at-the-money call option under the specified parameters is approximately 0.53.

2.2 Part b

[17]:
```
# Given values
S_0 = 4
C_0 = 5
dollar_delta = 3
dollar_gamma = 0.02
S = 3.6

# Convert dollar delta to delta
Delta = dollar_delta / S_0

# Convert dollar gamma to gamma
Gamma = (dollar_gamma * 100) / S_0**2

# Apply the second-order Taylor expansion to approximate the new option price␣
 ↪C(S)
C_S = C_0 + (S - S_0) * Delta + 0.5 * (S - S_0)**2 * Gamma

print(f"Time-0 value of the contract using a second-order Taylor expansion is␣
 ↪{C_S:.2f}")
```

```
Time-0 value of the contract using a second-order Taylor expansion is 4.71
```

[ ]:
```

```