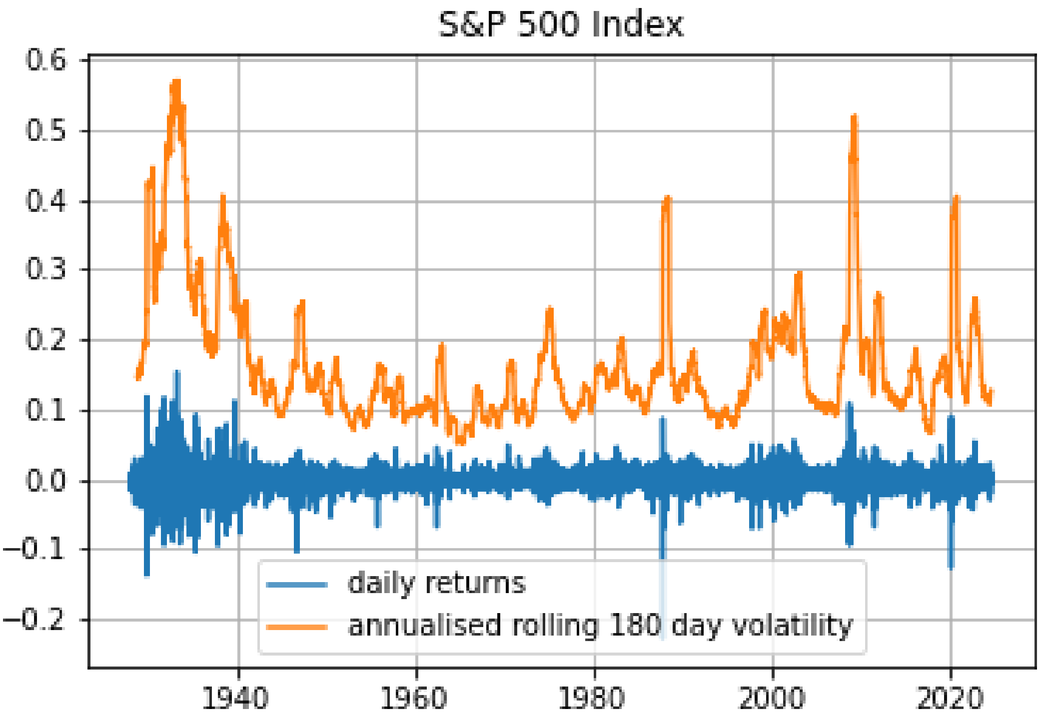
FINM3405 Derivatives and risk management

Tutorial Sheet 7: Options - Numerical methods (binomial and Monte Carlo)

Suggested solutions

September 22, 2024

Question 1. Download the historical daily returns of your favourite stock or share market index from yahoo!finance or some other data provider. Then using Excel or some other means, on the same graph plot the (i) time series of daily returns and (i) a rolling annualised 180 day historical volatility. What does the plot tell you about stock or share market index volatilities over time?



Volatility σ is not constant over time.

```python
import numpy as np, yfinance as yf
S = yf.download("^GSPC")["Adj Close"]
ret = np.log(S).diff(1).dropna()
sigma = ret.rolling(window=180).std()*np.sqrt(252)
plt.figure()
plt.plot(ret, label='daily returns')
plt.plot(sigma, label='annualised rolling 180 day volatility')
plt.grid(); plt.title("S&P 500 Index"); plt.legend()
```

In the following questions let $S = 50$, $K = 50$, $r = 5\%$, $T = \frac{1}{2}$, $\sigma = 25\%$ and dividend yield $y = 0$ unless otherwise stated.

Binomial model

Question 2. Calculate by hand the 1-step binomial prices of ATM European call options and compare them to the Black-Scholes prices. Do the same for strikes of $K = 47.5$ and $K = 52.5$. Use the CRR or JR schemes. I did the ATM options in the lecture. For the others, and using the Jarrow-Rudd scheme (since it most resembles geometric Brownian motion), we still have that

$$u = e^{(r-\sigma^2/2)T+\sigma\sqrt{T}} = 1.2046 \qquad \text{and} \qquad d = e^{(r-\sigma^2/2)T-\sigma\sqrt{T}} = 0.84586,$$

as well as

$$q = \frac{u^{rT} - d}{u - d} = 0.500231.$$

We calculate that

$$S_u = Su = 60.23 \qquad \text{and} \qquad S_d = Sd = 42.293.$$

For the strike of $K = 47.5$, the call option payoffs are

$$C_u = \max\{0, S_u - K\} = 12.73 \qquad \text{and} \qquad S_d = \max\{0, S_d - K\} = 0,$$

and the put payoffs are

$$P_u = \max\{0, K - S_u\} = 0 \qquad \text{and} \qquad P_d = \max\{0, K - S_d\} = 5.207.$$

Call prices are given by

$$\begin{aligned} C &= e^{-rT}\mathbb{E}^q[C_T] \\ &= e^{-rT}\big[qC_u + (1-q)C_d\big] \\ &= 6.211 \end{aligned}$$

and the put prices are

$$\begin{aligned} P &= e^{-rT}\mathbb{E}^q[P_T] \\ &= e^{-rT}\big[qP_u + (1-q)P_d\big] \\ &= 2.538. \end{aligned}$$

When $K = 52.5$ I get $C = 3.77$ and $P = 4.9752$.

```python
S = 50
K = 52.5
r = 0.05
T = 1/2
sigma = 0.25
u = np.exp((r - 0.5*sigma**2)*T + sigma*np.sqrt(T)) # JR
d = np.exp((r - 0.5*sigma**2)*T - sigma*np.sqrt(T)) # JR
q = (np.exp(r*T)-d)/(u-d)
Su = S*u
Sd = S*d
Cu = max(0, Su-K)
Cd = max(0, Sd-K)
Pu = max(0, K-Su)
Pd = max(0, K-Sd)
C = np.exp(-r*T)*(q*Cu + (1-q)*Cd)
P = np.exp(-r*T)*(q*Pu + (1-q)*Pd)
```

Question 3. Use excel to create a 7 layer binomial model asset price tree and calculate the binomial model prices of ATM European call and put options, and compare your prices to the Black-Scholes model prices. Do the same for strikes of $K = 47.5$ and $K = 52.5$. Use the CRR or JR schemes. See the spreadsheet provided, which gives the following asset, call and put price trees for ATM options:

Asset price tree							
							80.568
						75.260	
					70.302		70.490
				65.670		65.846	
			61.343		61.508		61.673
		57.302		57.456		57.610	
	53.527		53.670		53.814		53.958
50		50.134		50.269		50.403	
	46.831		46.957		47.083		47.209
		43.863		43.981		44.099	
			41.083		41.193		41.304
				38.479		38.582	
					36.040		36.137
						33.756	
							31.617

Call price tree							
							30.568
						25.438	
					20.658		20.490
				16.203		16.024	
			12.22		11.864		11.673
		8.882		8.3335		7.7879	
	6.237		5.603		4.8627		3.958
4.250		3.636		2.9123		1.9722	
	2.293		1.695		0.9826		0.000
		0.966		0.4896		0	
			0.244		0		0.000
				0		0	
					0		0.000
						0	
							0.000

Put price tree							
							0.000
						0	
					0		0.000
				0		0	
			0.172		0		0.000
		0.695		0.3451		0	
	1.65		1.224		0.6928		0.000
3.015		2.617		2.1109		1.3905	
	4.402		4.029		3.5441		2.791
		6.218		5.9761		5.7231	
			8.452		8.4509		8.696
				10.988		11.239	
					13.604		13.863
						16.065	
							18.383

But it's not feasible to do numerical option pricing in Excel due to the manual work involved and inflexibility in choosing parameters values such as the number N of layers or dates characterising the binomial model trees. It's more efficient and flexible to use coding languages such as Python, Java, C^{++}, etc. The below Python code calculates binomial option prices and produces the following trees (in which we ignore the NumPy NaNs):

	0	1	2	3	4	5	6	7
0	50	46.8311	43.863	41.083	38.4792	36.0405	33.7563	31.6169
1	nan	53.5265	50.1341	46.9567	43.9806	41.1932	38.5824	36.1372
2	nan	nan	57.3018	53.6701	50.2686	47.0826	44.0986	41.3037
3	nan	nan	nan	61.3434	57.4555	53.8141	50.4034	47.2089
4	nan	nan	nan	nan	65.6699	61.5079	57.6096	53.9584
5	nan	nan	nan	nan	nan	70.3017	65.8461	61.6729
6	nan	nan	nan	nan	nan	nan	75.2601	70.4903
7	nan	nan	nan	nan	nan	nan	nan	80.5683

	0	1	2	3	4	5	6	7
0	4.2496	2.29278	0.965963	0.243915	0	0	0	0
1	nan	6.23673	3.63594	1.69489	0.489563	0	0	0
2	nan	nan	8.88202	5.60291	2.91228	0.982605	0	0
3	nan	nan	nan	12.2245	8.3335	4.8627	1.97219	0
4	nan	nan	nan	nan	16.2028	11.8638	7.78786	3.9584
5	nan	nan	nan	nan	nan	20.6576	16.0243	11.6729
6	nan	nan	nan	nan	nan	nan	25.4384	20.4903
7	nan	nan	nan	nan	nan	nan	nan	30.5683

	0	1	2	3	4	5	6	7
0	3.0151	4.40168	6.21804	8.45169	10.9879	13.6036	16.0655	18.3831
1	nan	1.65016	2.6169	4.029	5.97607	8.45092	11.2393	13.8628
2	nan	nan	0.695273	1.2236	2.11085	3.54411	5.72314	8.6963
3	nan	nan	nan	0.171952	0.345142	0.692771	1.39053	2.79109
4	nan	nan	nan	nan	0	0	0	0
5	nan	nan	nan	nan	nan	0	0	0
6	nan	nan	nan	nan	nan	nan	0	0
7	nan	nan	nan	nan	nan	nan	nan	0

The differences here are that the Python code creates the trees "upside down" relative to how we may visualise them in our heads (the row number counts the number of up movements in the asset price), and computers like to work with rectangular matrices or data frames or arrays, hence the NaNs (not a number). But neither of these are major obstacles to conceptualise and work with.

```
import numpy as np
S = 50
K = 50
r = 0.05
T = 1/2
sigma = 0.25
N = 7; dt = T/N
u = np.exp((r - 0.5*sigma**2)*dt + sigma*np.sqrt(dt)) # JR
d = np.exp((r - 0.5*sigma**2)*dt - sigma*np.sqrt(dt)) # JR
q = (np.exp(r*dt)-d)/(u-d)
# asset price tree
St = np.zeros([N+1, N+1])*np.nan # initially create array of NaNs
for j in range(N+1):
    for i in range(j+1):
        St[i,j] = S*(u**i)*d**(j-i)
# option price trees
Ct = np.zeros([N+1, N+1])*np.nan # initially create array of NaNs
Pt = np.zeros([N+1, N+1])*np.nan # initially create array of NaNs
for i in range(N+1): # option payoffs at expiry
    ST = St[i,N]
    Ct[i,N] = max(0, ST-K)
    Pt[i,N] = max(0, K-ST)
for j in reversed(range(N)): # step backwards through asset tree to create option trees
   for i in range(j+1):
        Ct[i,j] = np.exp(-r*dt)*(q*Ct[i+1, j+1] + (1-q)*Ct[i, j+1])
        Pt[i,j] = np.exp(-r*dt)*(q*Pt[i+1, j+1] + (1-q)*Pt[i, j+1])
C = Ct[0,0]
P = Pt[0,0]
```

Question 4. In the derivation of the binomial model in the lecture notes, we suggestively defined and found an equation for the parameter Δ, namely

$$\Delta = \frac{C_u - C_d}{S(u-d)} = \frac{C_u - C_d}{S_u - S_d}.$$

The notation was not a coincidence since it does in fact define the delta of a call option as calculated by the binomial model. Calculate by hand the 1-period binomial model deltas of ATM calls and puts and compare them to the Black-Scholes model deltas. You can use the CRR or JR schemes. From the first question (or lecture notes), when $K = 50$ we had $S_u = 60.23$, $S_d = 42.293$, $C_u = 10.23$, $C_d = 0$, $P_u = 0$ and $P_d = 7.707$. This gives

$$\Delta_C = 0.57 \qquad \text{and} \qquad \Delta_P = -0.4297.$$

These compare to the Black-Scholes deltas of $\Delta_C = 0.59088$ and $\Delta_P = -0.40912$. We'd add the following lines to the code from Question 1 to calculate them:

```
deltaC = (Cu-Cd)/(Su-Sd)
deltaP = (Pu-Pd)/(Su-Sd)
```

Question 5. Following on from questions 3 and 4, calculate the ATM put and call deltas at each node of your 7 layer asset price tree (not including the final expiry date nodes). Then compare the deltas at the time $t_0 = 0$ node to the Black-Scholes model deltas. See the Excel spreadsheet provided. But it's much better to just add a few lines to the Python code. The below code calculates $\Delta_C = 0.589$ and $\Delta_P = -0.41095$, and creates the following delta trees:

	0	1	2	3	4	5	6
0	0.589048	0.425758	0.24703	0.08898…	0	0	0
1	nan	0.731905	0.582121	0.385296	0.166842	0	0
2	nan	nan	0.862947	0.754317	0.576415	0.312808	0
3	nan	nan	nan	0.957984	0.909958	0.807036	0.586474
4	nan	nan	nan	nan	1	1	1
5	nan	nan	nan	nan	nan	1	1
6	nan	nan	nan	nan	nan	nan	1
7	nan	nan	nan	nan	nan	nan	nan

	0	1	2	3	4	5	6
0	−0.4109…	−0.5742…	−0.75297	−0.9110…	−1	−1	−1
1	nan	−0.2680…	−0.4178…	−0.6147…	−0.8331…	−1	−1
2	nan	nan	−0.1370…	−0.2456…	−0.4235…	−0.6871…	−1
3	nan	nan	nan	−0.0420…	−0.0900…	−0.1929…	−0.4135…
4	nan	nan	nan	nan	0	0	0
5	nan	nan	nan	nan	nan	0	0
6	nan	nan	nan	nan	nan	nan	0
7	nan	nan	nan	nan	nan	nan	nan

```
import numpy as np
S = 50
K = 50
r = 0.05
T = 1/2
sigma = 0.25
N = 7; dt = T/N
u = np.exp((r - 0.5*sigma**2)*dt + sigma*np.sqrt(dt)) # JR
d = np.exp((r - 0.5*sigma**2)*dt - sigma*np.sqrt(dt)) # JR
q = (np.exp(r*dt)-d)/(u-d)
# asset price tree
St = np.zeros([N+1, N+1])*np.nan
for j in range(N+1):
    for i in range(j+1):
        St[i,j] = S*(u**i)*d**(j-i)
# option premiums and deltas
Ct = np.zeros([N+1, N+1])*np.nan
Pt = np.zeros([N+1, N+1])*np.nan
deltaCt = np.zeros([N+1, N])*np.nan
deltaPt = np.zeros([N+1, N])*np.nan
```

```
for i in range(N+1): # option payoffs at expiry
    ST = St[i,N]
    Ct[i,N] = max(0, ST-K)
    Pt[i,N] = max(0, K-ST)
for j in reversed(range(N)): # step backwards through the tree
   for i in range(j+1):
        Ct[i,j] = np.exp(-r*dt)*(q*Ct[i+1, j+1] + (1-q)*Ct[i, j+1])
        Pt[i,j] = np.exp(-r*dt)*(q*Pt[i+1, j+1] + (1-q)*Pt[i, j+1])
        deltaCt[i,j] = (Ct[i+1, j+1] - Ct[i, j+1])/(St[i+1, j+1] - St[i, j+1])
        deltaPt[i,j] = (Pt[i+1, j+1] - Pt[i, j+1])/(St[i+1, j+1] - St[i, j+1])
C = Ct[0,0]
P = Pt[0,0]
deltaC = deltaCt[0,0]
deltaP = deltaPt[0,0]
```

Question 6. A benefit of numerical option pricing methods is they can handle a very wide variety of payoffs at expiry, and hence price a wide variety of exotic options. Use the 7 layer binomial model to price the cash-or-nothing and asset-or-nothing binary options presented in a previous tutorial sheet and compare the results to their Black-Scholes prices. Use the CRR or JR schemes. For pricing the binary options, all we need to do here in either the Excel spreadsheets or the Python code is simply change the terminal payoffs at expiry. For example, the below Python code does it for the cash-or-nothing options and gives $C = 0.4877$ and $P = 0.4876$. The Black-Scholes prices are $C = 0.5083$ and $P = 0.467$. When I set $N = 10,000$ in the code I get $C = 0.5071$ and $P = 0.4682$.

```
import numpy as np
S = 50
K = 50
r = 0.05
T = 1/2
sigma = 0.25
N = 10000; dt = T/N
u = np.exp((r - 0.5*sigma**2)*dt + sigma*np.sqrt(dt)) # JR
d = np.exp((r - 0.5*sigma**2)*dt - sigma*np.sqrt(dt)) # JR
q = (np.exp(r*dt)-d)/(u-d)
# asset price tree
St = np.zeros([N+1, N+1])*np.nan
for j in range(N+1):
    for i in range(j+1):
        St[i,j] = S*(u**i)*d**(j-i)
# option premiums and deltas
Ct = np.zeros([N+1, N+1])*np.nan
Pt = np.zeros([N+1, N+1])*np.nan
for i in range(N+1): # option payoffs at expiry
    ST = St[i,N]
    Ct[i,N] = 1 if ST>K else 0 # cash-or-nothing call payoff
    Pt[i,N] = 1 if ST<K else 0 # cash-or-nothing put payoff
for j in reversed(range(N)): # step backwards through the tree
   for i in range(j+1):
        Ct[i,j] = np.exp(-r*dt)*(q*Ct[i+1, j+1] + (1-q)*Ct[i, j+1])
        Pt[i,j] = np.exp(-r*dt)*(q*Pt[i+1, j+1] + (1-q)*Pt[i, j+1])
C = Ct[0,0]
P = Pt[0,0]
```

Monte Carlo method

Question 7. When pricing European options by the Monte Carlo method, we actually only need to calculate the N final asset price prices S_i at expiry for $i = 1, \ldots, N$. We can use the formula in the lecture notes to simulate them via

$$S_i = Se^{(r-\frac{1}{2}\sigma^2)T+\sigma\sqrt{T}Z_i} \qquad \text{for } i = 1, \ldots, N,$$

where each Z_i is an independent, identically distributed standard normal random variable. The option prices are then simply given by

$$C = e^{-rT}\frac{1}{N}\sum_{i=1}^{N}\max\{0, S_i - K\} \qquad \text{and} \qquad P = e^{-rT}\frac{1}{N}\sum_{i=1}^{N}\max\{0, K - S_i\}.$$

Use Excel, Python, R or Matlab, etc (whatever is your software of choice - I use these and Java and C^{++}), to price ATM call and put options via this "simplified" Monte Carlo method that simulates only the final asset prices at expiry. *Remark*: It's particularly easy in Excel by simulating 1 final asset price at expiry and then dragging this cell down for say another 499 cells to get 500 simulated final asset prices at expiry. Compare your prices to the Black-Scholes and binomial model prices. Do the same for strikes of $K = 47.5$ and $K = 52.5$. I posted the R code for this on Blackboard, upon request from some students. When $K = 50$, the Python code below gives $C = 4.131$ and $P = 2.8904$.

```
import numpy as np
from scipy.stats import norm
S = 50; K = 50; r = 0.05; T = 1/2; sigma = 0.25; N = 1000000
CT = np.zeros(N); PT = np.zeros(N)
for i in range(N):
    ST = S*np.exp((r - 0.5*sigma**2)*T + sigma*np.sqrt(T)*norm.rvs())
    CT[i] = max(0, ST-K)
    PT[i] = max(0, K-ST)
C = np.exp(-r*T)*np.mean(CT)
P = np.exp(-r*T)*np.mean(PT)
```

Question 8. Now use this simplified Monte Carlo method to price the ATM cash-or-nothing and asset-or-nothing binary options and compare your prices to the Black-Scholes and binomial model prices. Again, for binary options we need only modify the option payoffs in the above code. The below code does this for the cash-or-nothing options and gives $C = 0.50885$ and $P = 0.4665$.

```
import numpy as np
from scipy.stats import norm
S = 50; K = 50; r = 0.05; T = 1/2; sigma = 0.25; N = 1000000
CT = np.zeros(N); PT = np.zeros(N)
for i in range(N):
    ST = S*np.exp((r - 0.5*sigma**2)*T + sigma*np.sqrt(T)*norm.rvs())
    CT[i] = 1 if ST>K else 0
    PT[i] = 1 if ST<K else 0
C = np.exp(-r*T)*np.mean(CT)
P = np.exp(-r*T)*np.mean(PT)
```