J.P.Morgan Quant Mentorship Program 2022

Name: Mahek Vora
Institute : IIT Guwahati
Branch: Computer Science and Engineering
Year: B.Tech(2020-2024), Second year

Table of Contents

CASE STUDY - A

Q.1 Continuous compounding

1: Formulae :

$A = P(1 + r/n)^{nt}$
where,
A = amount
P = principal
r = rate (not in percentage, ranging from 0 to 1)
t = number of years
n = number of times interest is compounded per year

Compound Interest = $A - P = P(1 + r/n)^{nt} - P = P((1 + r/n)^{nt} - 1)$

2: Given :
P=$10000
r=0.05
t=10
n is varying in the 4 sub parts.

3: Assumptions :
There are no leap years

4: Solution :
We are successively increasing the value of n to notice the results

n=2
Compounded semi-annually implies twice a year, hence n=2
Substituting the values in the formula,
$A = 10000(1 + (0.05/2))^{2*10}$
$= 10000(1.025)^{20}$
$= 16386.1644$
A = $16386.1644

n=52
There are 52 weeks in every year,n=52
$A = 10000(1 + (0.05/52))^{52*10}$
$= 16483.25245$
A = $16483.25245

n=365

There are 365 days in every year,n=365

$A = 10000(1 + (0.05/365))^{365*10}$

$= 16486.64814$

A = $16486.64814

Computing in terms of parameters, taking P, r, t as constants and n as a variable $A = P(1 + r/n)^{nt}$.

The limit of the function as n → infinity is shown (Figure 01 under point 5, page number 5)

Calculating its limit as n tends to infinity

$$\lim_{n\to\infty} P(1+\frac{r}{n})^{nt})$$
$$= P.(e^{(1+\frac{r}{n})-1)nt})$$
$$= P.(e^{rt})$$

Using this formula :

$$\lim_{n\to\infty}\left(1+\frac{x}{n}\right)^{n} = e^{\left(1+\frac{x}{n}\right)-1)n}$$
$$= e^{x}$$

Substituting P, r, t we get

$A = P.e^{rt}$ → (equation 1)

$= 10000*e^{0.05*10}$

= **$ 16487.21271**

5: Graphical representation :

The graph of Amount vs n is given in ‘Figure 01’

$Amount = 10000(1 + 0.05/n)^{n*10}$

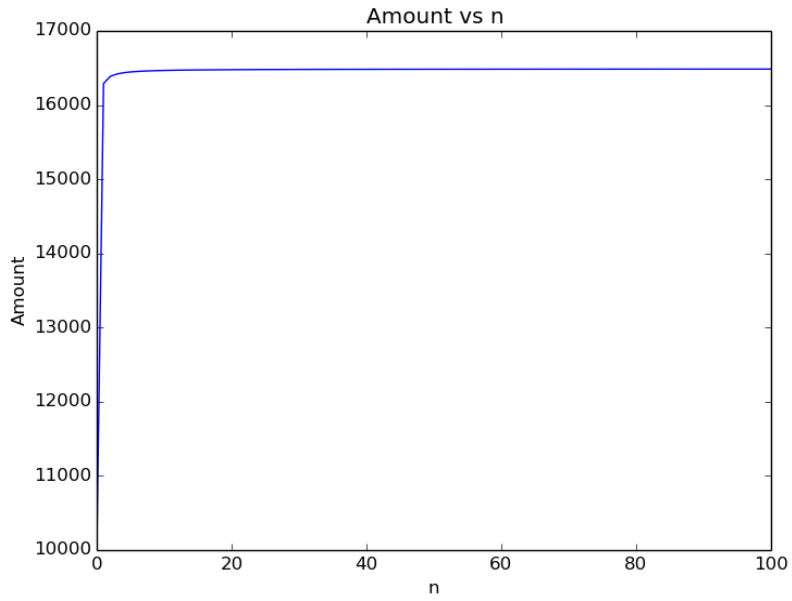


Figure 01

Source code:'MAHEK_VORA_CASE_STUDY_A_Q1_MAIN'

```
import matplotlib.pyplot as plt
import numpy as np
n= np.linspace(0,100,100)
y=10000*(1 + 0.05/n)**(n*10)
plt.plot(n,y)
plt.title('Amount vs n')
plt.xlabel('n')
plt.ylabel('Amount')
plt.show()
```

Q.2 Call / Put options

Q2.a:

1: Assumptions

The prices of the stocks at time t and T are given by S_t and S_T respectively

2: Solution

In order to compare the different amounts we must consider all of them in one time frame

The table depicts prices at present vs 't' years from now and 'T' years from now, where t is any arbitrary value between 0 and T, i.e. 0 <= t <= T

Table 01

	n = 0	n = t	n = T
Stock Price (in $)	S	S_t	S_T
Strike price (in $)	$K.e^{-rT}$	$K.e^{-r(T-t))}$	K
Premium to be paid (in $)	Pre	$Pre.e^{rt}$	$Pre.e^{rT}$

Explanation of the table :

The values marked in blue are the given values
Using Q.1 part d, (equation 1) we notice that moving 't' units forward on the time-axis will increase the value/price of an entity undergoing continuous compounding by a factor of e^{rt} where r is the rate of continuous compounding

Thus moving 't' units *backwards* must *reduce* the value by the same amount.

A new variable Pre has been declared to represent the amount of price the user is paying at present. It denotes 'premium'

Comparing the values at a fixed point of time :
Consider all the prices T years from now, i.e. column 4 in the table

Case 1: $S_T >= K$
In case of a call option profit will be attained when (depicted by the graph from the study material, attached here as Figure 02)

strike price + premium <= stock price

This is because a call option gives you the 'right to buy a stock at a pre-determined strike price at a date in the future,' → if stock price increases, you are paying less for a more valuable entity and the payoff attained is the difference in whatever one received and whatever one invested, (in this case invested = premium + strike price, received = something as valuable as stock price)

Substituting values,

$$K + Pre.e^{rT} <= S_T$$

$\Rightarrow$ Pre <= $(S_T - K).e^{-rT}$

This should be the upper bound on the price the user is willing to pay for a call option at t=0, i.e. present

Case 2: $S_T < K$

This means the strike price(after T years) is greater than the stock price(after T years). In this case the user would not consider buying a call option as it defeats the purpose of flooring the loss, so here the price the user is willing to pay is $0

3: Conclusion

For call option if(S_T >= K) : Premium <= (S_T - K).e^{-rT} else : Premium = 0, does not buy the option

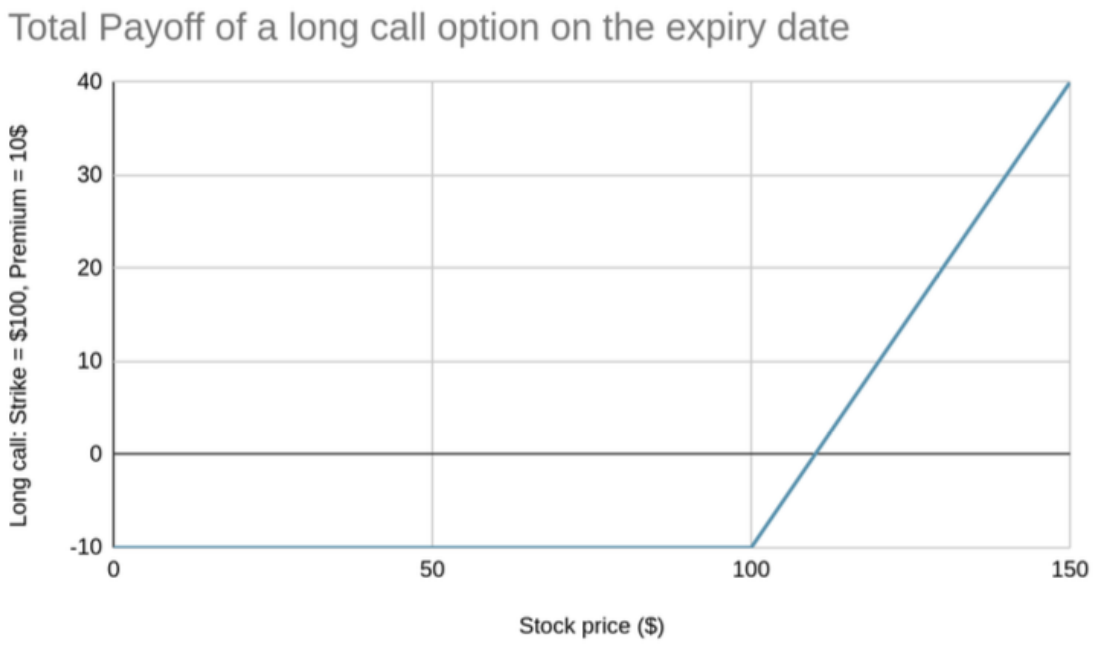


Figure 02

The point at which payoff becomes positive is when stock price crosses the sum of strike price and premium, i.e. strike + premium <= stock price

Q2.b:

1: Assumptions :

The prices of the stocks at time t and T are given by S_t and S_T respectively

2: Solution :

Following the same approach as above, all the values are to be compared at the same time, here we are comparing at time T
The table depicts prices at present vs 't' years from now and 'T' years from now, where t is any arbitrary value between 0 and T, i.e. 0 <= t <= T

	n = 0	n = t	n = T
Stock Price (in $)	S	S_t	S_T
Strike price (in $)	$K.e^{-rT}$	$K.e^{-r(T-t))}$	K
Premium to be paid (in $)	Pre	$Pre.e^{rt}$	$Pre.e^{rT}$

Comparing the values at a fixed point of time :
Consider all the prices T years from now, i.e. column 4 in the table

Case 1: $K>=S_T$
In case of a put option profit(positive payoff) will be attained when (depicted by the graph from the study material, attached here as Figure 03)

stock price + premium <= strike price

This is because a put option gives you 'the right to sell a stock at a pre determined strike price at a date in the future' → if stock price decreases, you are selling a more valuable entity at a lesser price and the payoff attained is the difference in whatever one received and whatever one invested, (in this case invested = premium, received = strike price - stock price)
Substituting values

$$S^T + Pre.e^{rT} <= K$$

$$\Rightarrow Pre <= (K - S_T).e^{-rT}$$

This should be the upper bound on the price the user is willing to pay for a call option at present

Case 2: $K<S_T$
This means the stock price (T years later) is greater than the strike price (T years later). In this case the user would not consider selling the stock at the predetermined price
So the user would be willing to pay $0, (i.e not buying the put option)

3: Conclusion

For put option
if(S_T<=K) : Premium <= (K- S_T).e^{-rT}
else : Premium = 0, does not buy the option

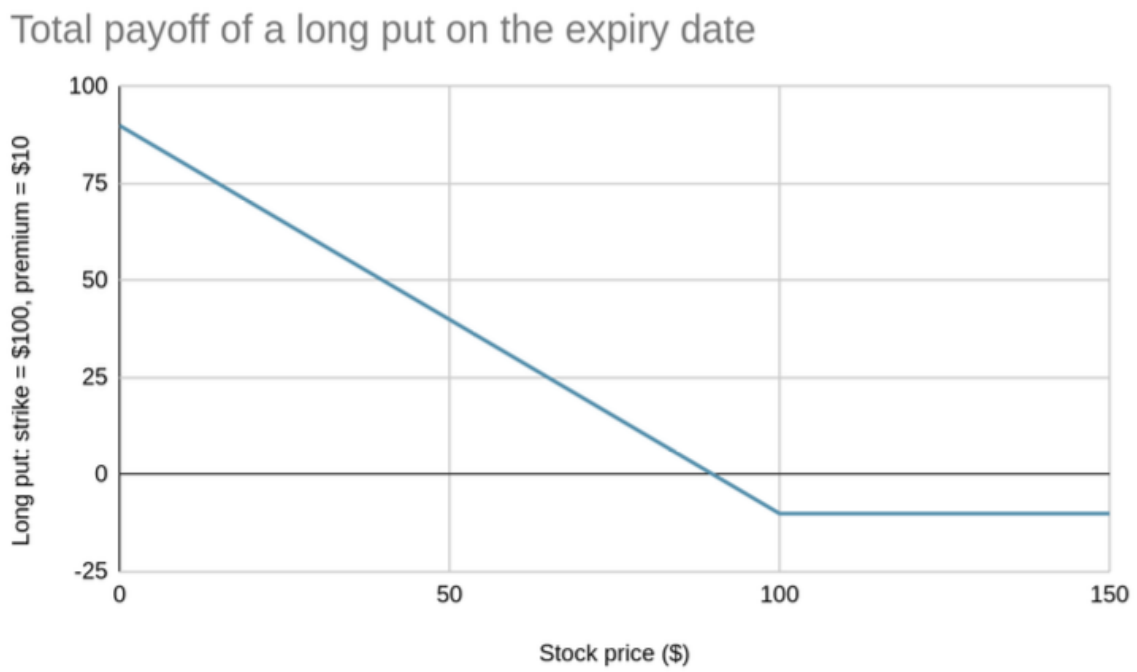


Figure 03

The point at which payoff is positive is before the stock price crosses the difference of strike price and premium, i.e. strike - premium >= stock price

Q.3 BSM Pricing

Q3.a:

Mathematical calculations when t → T

- Calculations for d_1, d_2 when t → T
- Calculations for N(d) when d goes to infinity, -infinity, and 0

As t $\to T$

$d_2 = d_1 - \sigma\sqrt{T-t}$

$d_2 \to d_1$

Considering $d_2 = d_1 = d$ for calculations of part (a),

$$d = \frac{1}{\sigma\sqrt{(T-t)}}\left(\ln\left(\frac{s_t}{k}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right)$$

$$d = \lim_{t\to T}\left(\frac{1}{\sigma\sqrt{(T-t)}}\left[\ln\left(\frac{S_t}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right]\right)$$

Let t be T-h where $h \to 0$

$$d = \lim_{h\to 0}\left(\frac{1}{\sigma\sqrt{h}}\left[\ln\left(\frac{S_T}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)h\right]\right)$$

consider the cases where

(1) $S_t > K : \ln\left(\frac{s_t}{k}\right) > 0 \to$ Numerator is finite, positive, denominator tending to 0

$$d \to \infty$$

(2) $S_t < K : \ln\left(\frac{s_t}{k}\right) < 0 \to$ Numerator is finite, negative, denominator tending to 0

$$d \to -\infty$$

(3) $S_t = K : \ln\left(\frac{s_t}{k}\right) = 0 \to$ Numerator is perfectly 0, denominator is tending to 0

$$d \to 0$$

$N(d) =$ CDF of the normal distribution. with mean $= 0, \sigma = 1$

$$= \int_{-\infty}^{d} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx$$

Computing $N(d)$ for the previous 3 cases:

$$d \to -\infty : \quad N(d) = 0$$

Proof: The upper & lower limits of the integral are equal

$$d \to \infty : \quad N(d) = 1$$

Proof: Let $I = \int_{-\infty}^{\infty} \frac{e^{-x^2/2}}{\sqrt{2}} dx$
putting $\frac{x}{\sqrt{2}} = z$,

$$I = \int_{-\infty}^{\infty} e^{-z^2} dz$$

$$I^2 = \left(\int_{-\infty}^{\infty} e^{-z^2} dz\right)^2$$
$$= \left(\int_{-\infty}^{\infty} e^{-x^2} dx\right)\left(\int_{-\infty}^{\infty} e^{-y^2} dy\right)$$
$$= \left(\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-(x^2+y^2)} dy\right)$$

Using polar co-ordinars,

$$I^2 = \int_0^{2\pi}\int_0^{\infty} e^{-r^2} r dr d\theta$$
$$= (2\pi)\left(\frac{e^{-r^2}\Big|_0^{\infty}}{-2}\right)$$
$$= \pi$$
$$I = \sqrt{\pi}$$
$$\lim_{d\to\infty} N(d) = 1$$

$$d \to 0: \quad \int_{-\infty}^{0} \frac{e^{-x^2/2}}{\sqrt{2\pi}} dx = 1/2$$

Proof:

$$\int_{-\infty}^{0} \frac{e^{-x^2/2}}{\sqrt{2\pi}} + \int_{0}^{\infty} \frac{e^{-x^2/2}}{\sqrt{2\pi}} = 1$$

Also, putthing $x = -x$, in $2nd$ integral,

$$2\int_{-\infty}^{0} \frac{e^{-x^2/2}}{\sqrt{2\pi}} dx = 1$$
$$\int_{-\infty}^{0} \frac{e^{-x^2/2}}{\sqrt{2\pi}} = 1/2$$

a.1 Call option, Case 1: ST > K

Calculations for premium at time T using BSM

When $t \to T$,

As $d_2 = d_1 - \sigma(T - t)^{1/2}$ and $t \to T$

$d_2 = d_1 = d$

The values for d as $t \to T$ have been computed for the cases where $S_T > K$ and $S_T < K$ individually

Here $S_T > K$ so $d \to \infty$, So $N(d) \to 1$,

$C(S_t, t) = N(d)[S_t - K.e^{-r(T-t)}]$

$= S_T - K$ → (equation 2)

This is the value of the premium at time T

From 'Table 01' and the conclusions derived in question 2

We have established that

Premium after time T = (Premium at present).e^{rT}

Premium for call options at time 0 is $(S_T-K).e^{-rT}$ at time t=0

Hence at time t=T it is S_T-K → (equation 3)

From equation 2 and 3 we have successfully confirmed the derived answers

a.2 Call option, Case 2: $S_T < K$

Calculations for premium at time T using BSM

When t → T,

As $d_2 = d_1 - \sigma(T - t)^{1/2}$ and t → T

$d_2 = d_1 = d$

The values for d as t → T have been computed for the cases where $S_T > K$ and $S_T < K$ individually

Here $S_T < K$ so d → -∞, So N(d) → 0,

$C(S_t,t) = N(d)[S_t - K.e^{-r(T-t)}]$

$= 0$ → (equation 4)

This is the value of the premium at time T

From 'Table 01' and the conclusions derived in question 2

We have established that

Premium after time T = (Premium at present).e^{rT}

Premium for call options at time 0 is 0 at time t=0

Hence at time t=T it is $0.e^{rT} = 0$ → (equation 5)

From equation 4 and 5 we have successfully confirmed the derived answers

a.3 Put option, Case 1: $K > S_T$

Calculations for premium at time T using BSM

When t → T,

As $d_2 = d_1 - \sigma(T - t)^{1/2}$ and t → T

$d_2 = d_1 = d$

The values for d as t → T have been computed for the cases where $S_T > K$ and $S_T < K$ individually

Here ST < K so d → -∞, So N(d) → 0,

$P(S_t, t) = K.e^{-r(T-t)} - S_t + C(S_t, t)$

$= K.e^{-r(T-t)} - S_t + N(d)[S_t - K.e^{-r(T-t)}]$

As t → T

$= K - S_T + 0$

$= K - S_T$

→ (equation 6)

This is the value of the premium at time T

From ‘Table 01’ and the conclusions derived in question 2
We have established that
Premium after time T = (Premium at present).e^{rT}
Premium for put options at time 0 is (K - ST).e^{-rT} at time t=0
Hence at time t=T it is K-S_T → (equation 7)
From equation 6 and 7 we have successfully confirmed the derived answers

a.4 Put option, Case 2: K < S_T

Calculations for premium at time T using BSM
When t → T,
As $d_2 = d_1 - \sigma(T - t)^{1/2}$ and t → T
$d_2 = d_1 = d$
The values for d as t → T have been computed for the cases where $S_T > K$ and $S_T < K$ individually
Here $S_T > K$ so d → ∞, So N(d) → 1,
$P(S_t, t) = K.e^{-r(T-t)} - S_t + C(S_t, t)$
$= K.e^{-r(T-t)} - S_t + N(d)[S_t - K.e^{-r(T-t)}]$
As t → T
$= K - S_T + S_T - K$
$= 0$
→ (equation 8)
This is the value of the premium at time T
From ‘Table 01’ and the conclusions derived in question 3
We have established that
Premium after time T = (Premium at present).e^{rT}
Premium for put options at time 0 is 0 at time t=0
Hence at time t=T it is $0.e^{rT}$ =0 → (equation 9)
From equation 8 and 9 we have successfully confirmed the derived answers

a.5 Call and Put option, Case 3: K = S_T

Here d → 0 so N(d) → ½
$C(S_t, t) = N(d)[S_t - K.e^{-r(T-t)}]$
As t → T
$= N(d)[S_T-K]$
$= 0$

$P(S_t, t) = K.e^{-r(T-t)} - S_t + C(S_t, t)$
$= K.e^{-r(T-t)} - S_t + N(d)[S_t - K.e^{-r(T-t)}]$
As t → T
$= K - S_T + 1/2 (S_T - K)$
$= 0$ as $S_T = K$
From the conclusions of Q2, when ST = K

$C(S_t, t) = (S_T - K) = 0$
$P(S_t, t) = (S_T - K) = 0$

So, have successfully confirmed the derived answers

b.
Plotting graphs for put options, plotted using Python
Source code: **'MAHEK_VORA_CASE_STUDY_A_Q3_BC_MAIN'**

[Note: this code is written for part b and c combined, code is provided in this pdf after the graphs too]

The graphs :
At t=0 :

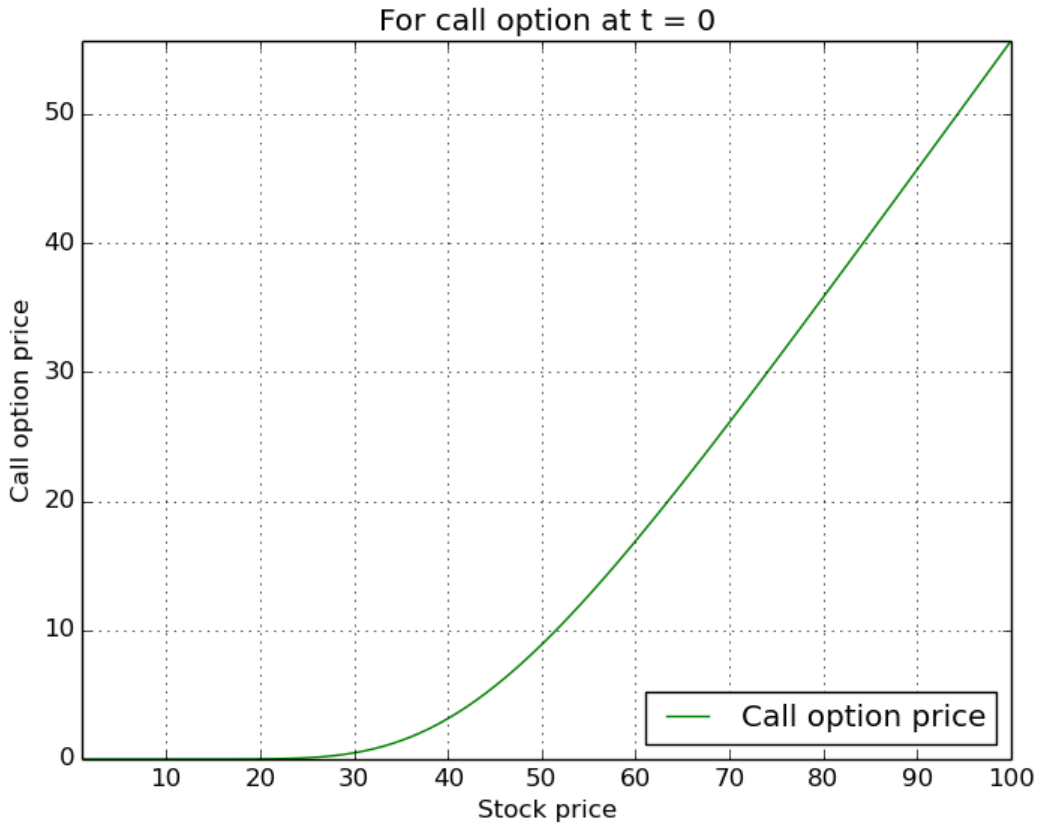


Figure 04

At t=0.5:

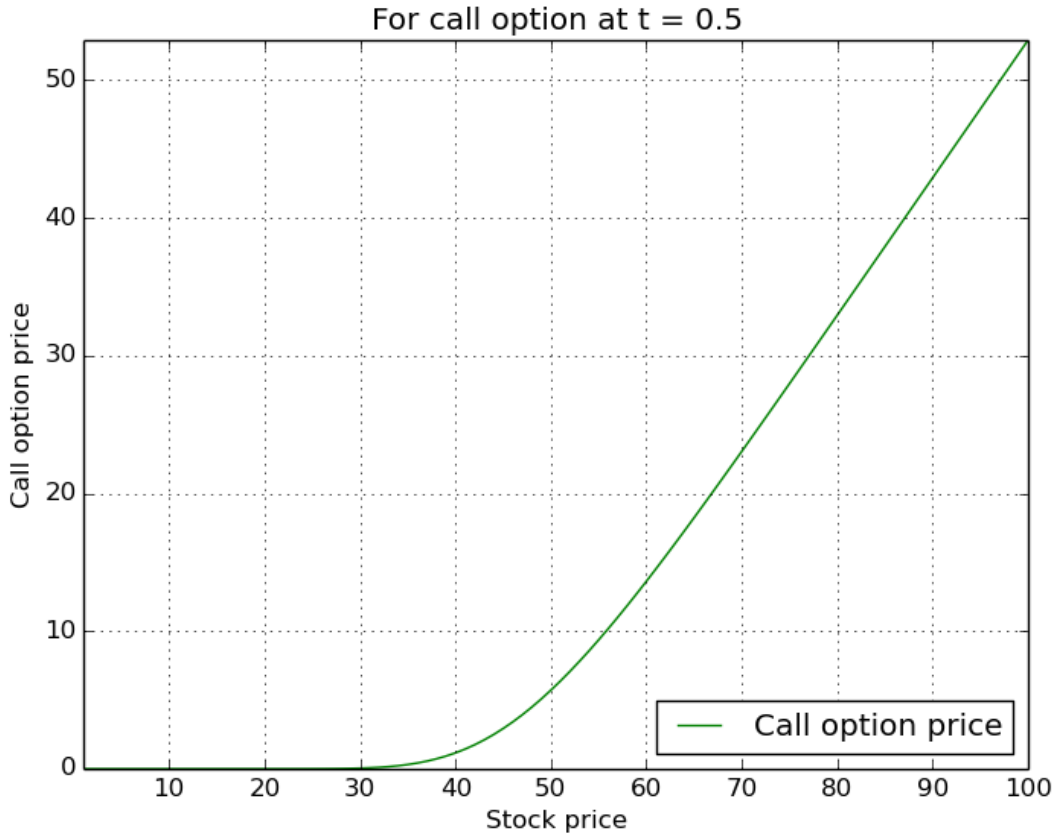


Figure 05

At t=1:

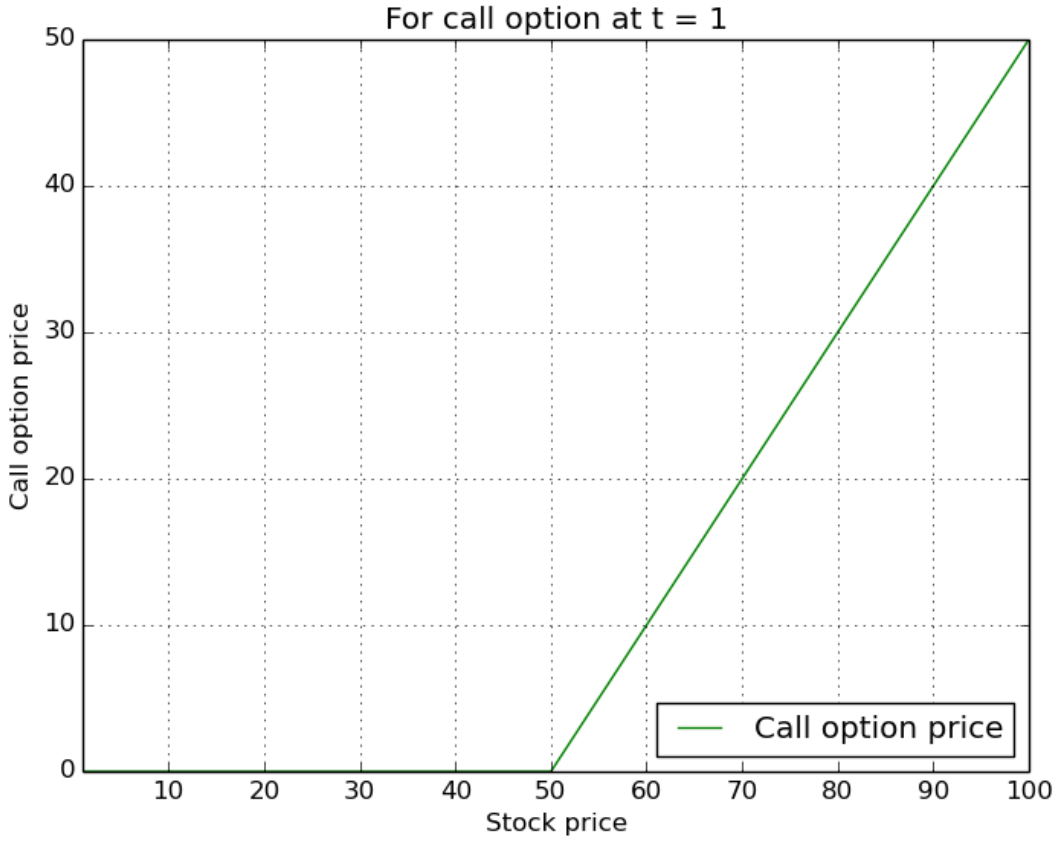


Figure 06

c.

Plotting graphs for put options, plotted using Python

The graphs :

At t=0:

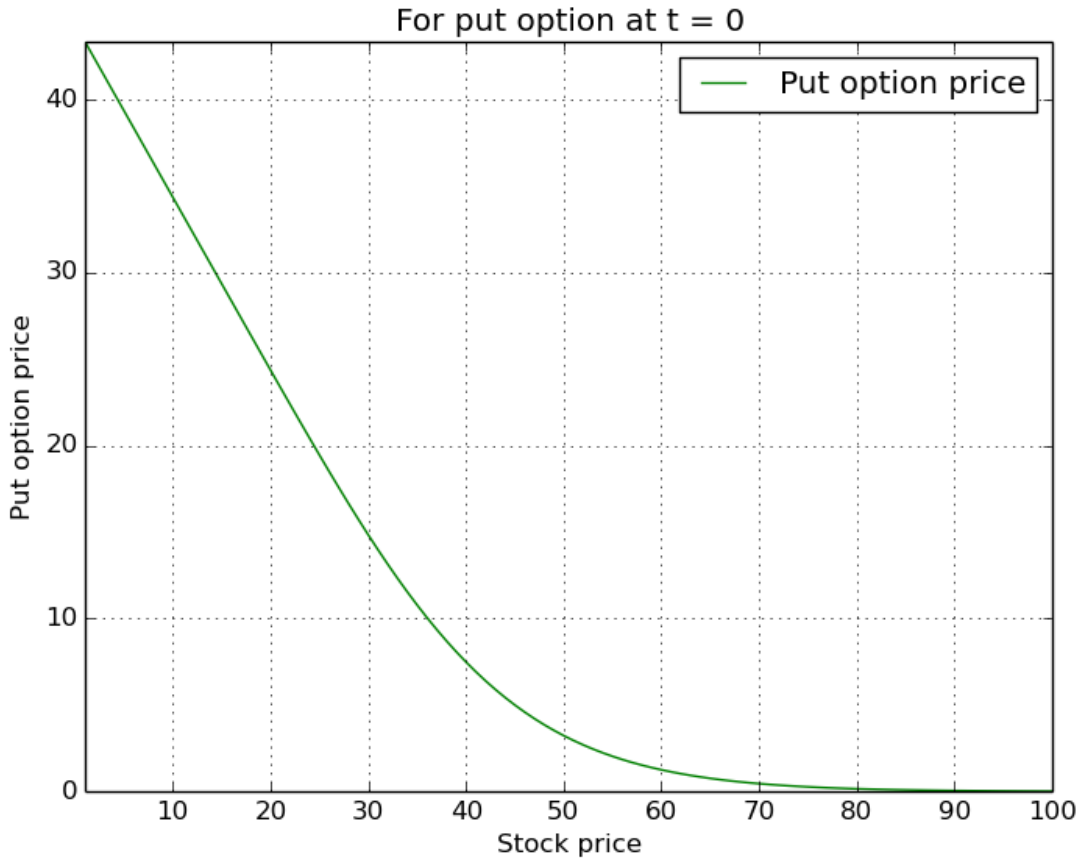


Figure 07

At t=0.5:

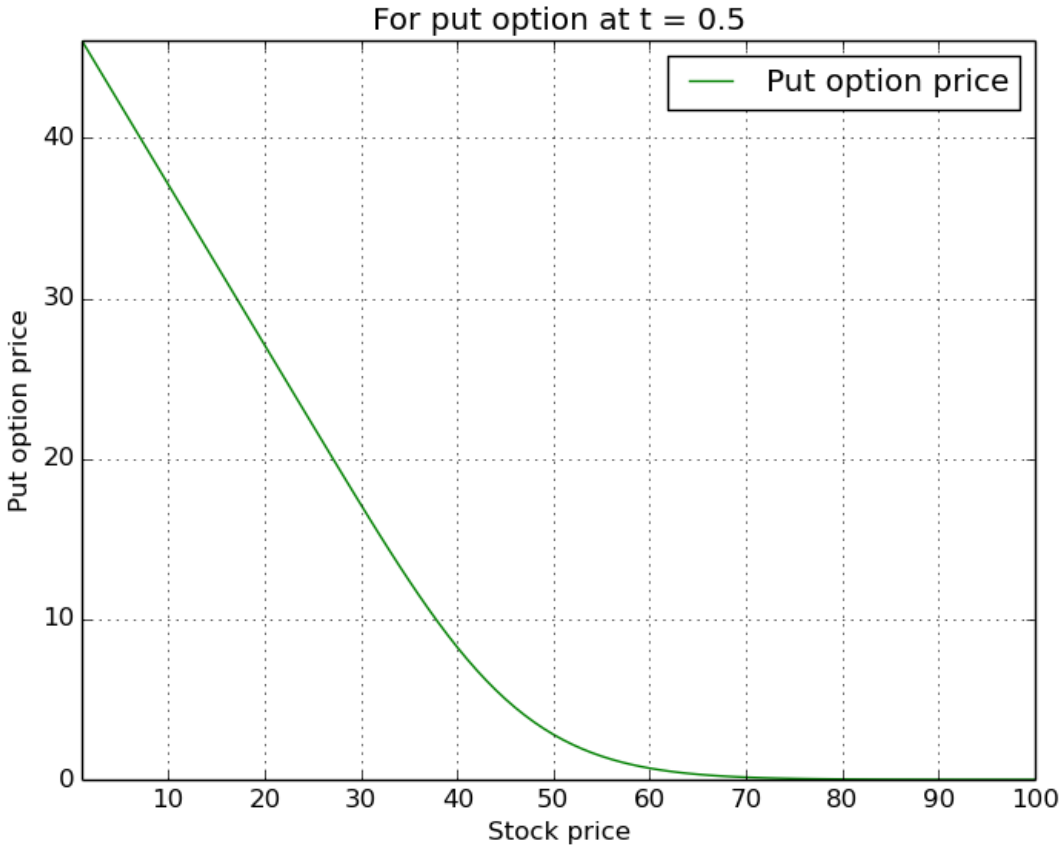


Figure 08

At t=1:

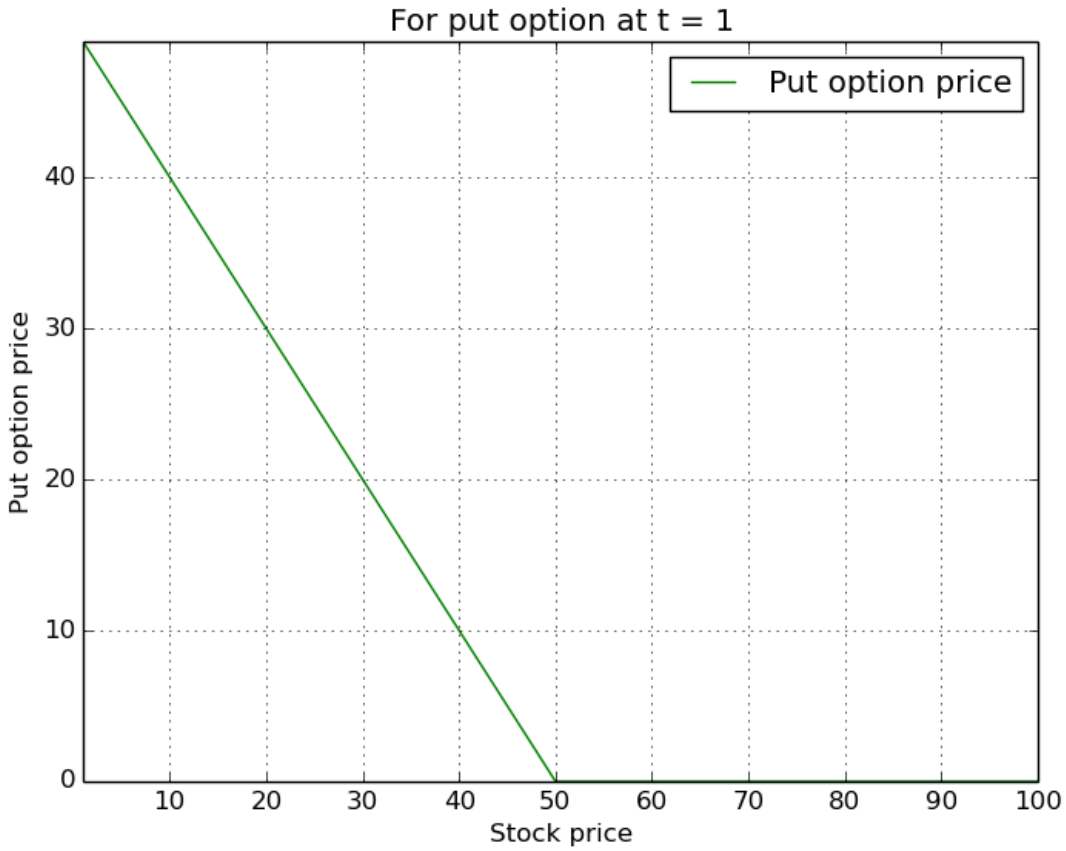


Figure 09

Source code: **'MAHEK_VORA_CASE_STUDY_A_3BC_MAIN.py'**

```
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
#plotting option price vs stock price
S= np.linspace(0.0,1,100.0)
#using bsm formulae
def calculate(T,S,K,sigma,t,r):
   T=float(T)
   S=float(S)
   K=float(K)
   sigma=float(sigma)
   t=float(t)
   r=float(r)
   if t!=T:
       d1=(1/(sigma*np.sqrt(T-t)))*(np.log(S/K)+((r+(sigma*sigma/2))*(T-t)))
       d2=d1-sigma*(np.sqrt(T-t))
       C=S*(norm.cdf(d1))-norm.cdf(d2)*K*np.exp(-1*r*(T-t))
       P=norm.cdf(-d2)*K*np.exp(-1*r*(T-t))-S*(norm.cdf(-d1))
   if t==T:
       C=max(S-K,0)
       P=max(K-S,0)
   return C,P
T=1
K=50
sigma=0.3
r=0.12
def plot_graphs(t):
  X=[]
  Y1=[]
  Y2=[]
   for i in range(100):
      C,P=calculate(T,i+1,K,sigma,t,r)
      Y1.append(C)
      Y2.append(P)
      X.append(i+1)

  #graphs for call
  plt.figure()
  plt.xticks([0,10,20,30,40,50,60,70,80,90,100])
  plt.yticks([0,10,20,30,40,50,60,70,80,90,100])
  plt.plot(X,Y1,linestyle = 'solid',color='green')
  plt.grid('on')
  plt.xlabel('Stock price')
  plt.ylabel('Call option price')
  plt.title('For call option at t = '+str(t))
  plt.legend(['Call option price'],loc=4)
  plt.show()
  #graphs for put
  plt.figure()
```

```
  plt.xticks([0,10,20,30,40,50,60,70,80,90,100])
  plt.yticks([0,10,20,30,40,50,60,70,80,90,100])
  plt.plot(X,Y2,linestyle = 'solid',color='green')
  plt.grid('on')
  plt.xlabel('Stock price')
  plt.ylabel('Put option price')
  plt.title('For put option at t = '+str(t))
  plt.legend(['Put option price'])
  plt.show()
#plotting all the cases by calling the main function
if __name__=='__main__':
  plot_graphs(0)
  plot_graphs(0.5)
  plot_graphs(1)
```

Q.4 Delta Calculations

Delta (Δ) of a stock option, is the ratio of the change in the price of the stock option to the change in the price of the underlying stock

That is, we can compute it in the following two ways:

1. Differentiation of the function of the 'price of stock option' with respect to 'price of stock'.(Analytical approach)
2. Slope of the graph with Y axis measuring 'price of stock option' and X axis measuring 'price of stock'.(Numerical approach)

Analytical:

For call option:

$$\text{Let } T-t=\tau$$

$$\frac{\partial C}{\partial S}=\frac{\partial}{\partial S}\left\{SN\left(d_1\right)-Ke^{-r\tau}N\left(d_2\right)\right\}$$

$$=\left\{N\left(d_1\right)\frac{\partial S}{\partial S}+S\frac{\partial}{\partial S}\left(N\left(d_1\right)\right)\right\}-\left\{Ke^{-r\tau}\frac{\partial}{\partial S}\left(N\left(d_1-\sigma\sqrt{\tau}\right)\right)\right\}$$

$$\text{Taking } \frac{\partial}{\partial S}N\left(d_1\right)=n\left(d_1\right)\frac{\partial\left(d_1\right)}{\partial S}$$

$$\text{where } n\left(d_1\right)=\frac{\partial\left(N\left(d_1\right)\right)}{\partial d_1}$$

$$\text{And similarly for } d_2, n\left(d_2\right)=\frac{\partial\left(N\left(d_2\right)\right)}{\partial d_2}$$

$$= N(d_1) + \left(\frac{\partial d_1}{\partial S}\right)[S.n(d_1) - Ke^{-r\tau}n(d_1 - \sigma\sqrt{\tau})]$$

$$n(d_1) = \frac{1}{\sqrt{2\pi}}e^{-d_1^2/2}$$

$$n(d_2) = n\left(d_1 - \sigma\sqrt{2}\right)$$

$$= \frac{1}{\sqrt{2\pi}}e^{-\left(d_1^2/2 + \sigma^2\tau/2 - d_1\sigma\sqrt{\tau}\right)}$$

$$= n(d_1)\, e^{-\frac{\sigma^2\tau}{2} + \ln\left(\frac{S_t}{K}\right) + r\tau + \frac{\sigma^2}{2}\tau}$$

$$= \frac{S.n(d_1)}{K}e^{r\tau}$$

...Putting the value of d_1

$$= N(d_1) + \frac{\partial d_1}{\partial s}(0)$$

$$= N(d_1)$$

For put option:

Let $T - t = \tau$

$$\frac{\partial P}{\partial S} = \frac{\partial}{\partial S}\left\{Ke^{-r\tau}N(-d_2) - SN(-d_1)\right\}$$

$$= Ke^{-r\tau}\frac{\partial}{\partial S}N(-d_2) - \left[N(-d_1) + \frac{\partial}{\partial S}N(-d_1)\right]$$

$$= -N(-d_1) + \frac{\partial}{\partial S}(d_1)\left[S \cdot n(-d_1) - Ke^{-r\tau}n\left(-d_1 + \sigma\sqrt{\tau}\right)\right]$$

$$= -N(-d_1)$$

$$\because n\left(-d_1 + \sigma\sqrt{\tau}\right) = \frac{S}{K}e^{r\tau}n(-d_1)$$

Using the erf function to represent N(d1) where erf of z is given by:

$$\operatorname{erf} z = \frac{2}{\sqrt{\pi}}\int_0^z e^{-t^2}\, dt.$$

$$= N(d_1) + \left(\frac{\partial d_1}{\partial S}\right)[S.n(d_1) - Ke^{-r\tau}n(d_1 - \sigma\sqrt{\tau})]$$

$$N(d_1) = \int_{-\infty}^{d_1} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx$$

$$= \int_0^{d_1} \frac{e^{-x^2/2}dx}{\sqrt{2\pi}} + \int_{-\infty}^{0} \frac{e^{-x^2/2}dx}{\sqrt{2\pi}}$$

putting $x \to -x$, $\int_0^{\infty} \frac{e^{-x^2/2}}{\sqrt{2\pi}} = \int_{-\infty}^{0} \frac{e^{-x^2/2}}{\sqrt{2\pi}}$

$$= \frac{1}{2} + \int_0^{d_1} \frac{e^{-x^2/2}dx}{\sqrt{2\pi}} dx$$

$$= \frac{1}{2}\left[1 + \frac{2}{\sqrt{\pi}} \int_0^{d_1/\sqrt{2}} e^{-y^2/2} dy\right] = \frac{1}{2}[1 + \text{erf}(d_1/\sqrt{2})]$$

Hence for ST = $ 125, T =1 year, K = $ 50, r = 0.12, σ = 0.30
Substituting in the formula,

$$d_1 = \frac{1}{\sigma\sqrt{T-t}}\left[\ln\left(\frac{S_t}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right]$$

Conclusions of Analytical calculation of delta (using differentiation):

Table 02

t (years)	d_1	$N(d_1)$, delta for call option	$-N(-d_1)$, delta for put option
0	3.60430243958	0.999843503789	-0.00015649621068
0.5	4.7083446633	0.999998751317	-1.24868286084e-06
1	∞	1.00	0.00

Source code: 'MAHEK_VORA_CASE_STUDY_A_Q4_A_MAIN.py'

```
import numpy as np
from scipy.stats import norm
def compute_slope(T,S,K,r,sigma,t,print_output):

T=float(T)
S=float(S)
K=float(K)
sigma=float(sigma)
t=float(t)
r=float(r)
if t!=T:
```

```
    d1=(1/(sigma*np.sqrt(T-t)))*(np.log(S/K)+((r+(sigma*sigma/2))*(T-t)))
    nd1_call=norm.cdf(d1)
    neg_nd1_put=(-1)*norm.cdf(-1*d1)
    if print_output==True:
        print('t = '+str(t))
        print('d1      :'+str(d1))
        print('N(d1)   :'+str(nd1_call))
        print('-N(-d1) :'+str(neg_nd1_put))

if t==T:
    d1=np.Inf
    nd1_call=norm.cdf(d1)
    neg_nd1_put=(-1)*norm.cdf(-1*d1)
    if print_output==True:
        print('t = 1')
        print('d1      :'+str(d1))
        print('N(d1)   :'+str(nd1_call))
        print('-N(-d1) :'+str(neg_nd1_put))
return nd1_call,neg_nd1_put
if __name__ == "__main__":
# Q4
#  compute_slope(T,S,K,r,sigma,t,print_output)
 compute_slope(1,125,50,0.12,0.3,0,True)
 compute_slope(1,125,50,0.12,0.3,0.5,True)
 compute_slope(1,125,50,0.12,0.3,1,True)
```

Numerical:

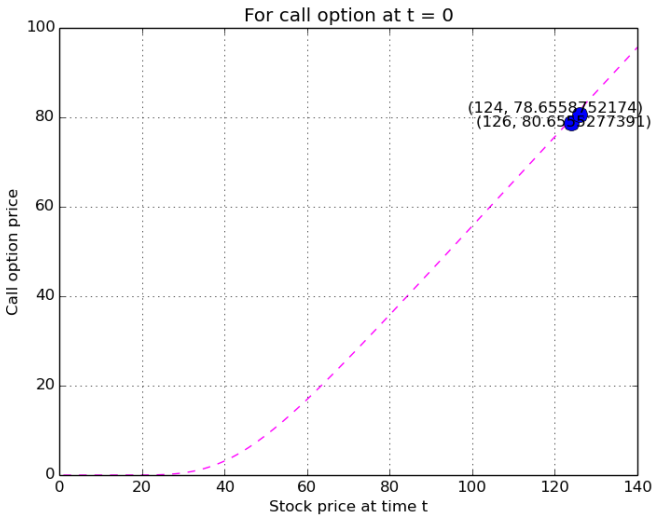
Consider the y values at S = 124 and S = 126 in order to calculate the slope using plotted graphs i.e. $y_1=f(124)$, $y_2=f(126)$

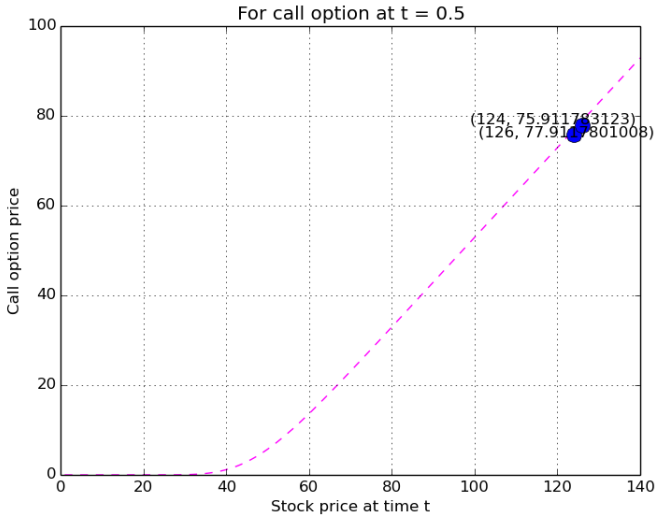
Then the delta can be calculated as (change in y)/ (change in x)

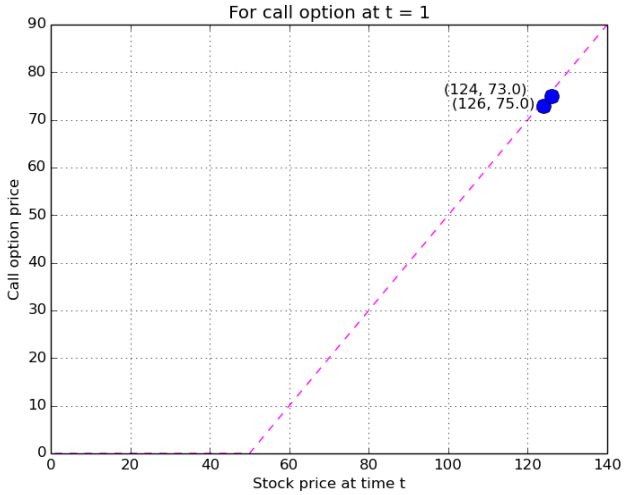
Here delta = (f(126) - f(124)) / (126-124)

= $(y_2 - y_1)/2$

Graphs for call options







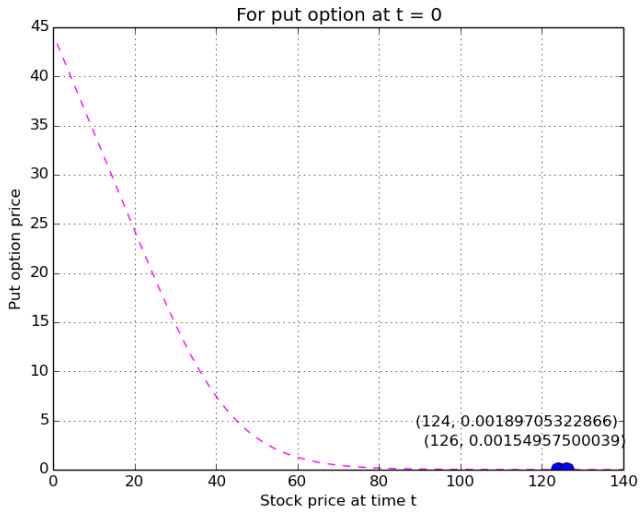
Delta:

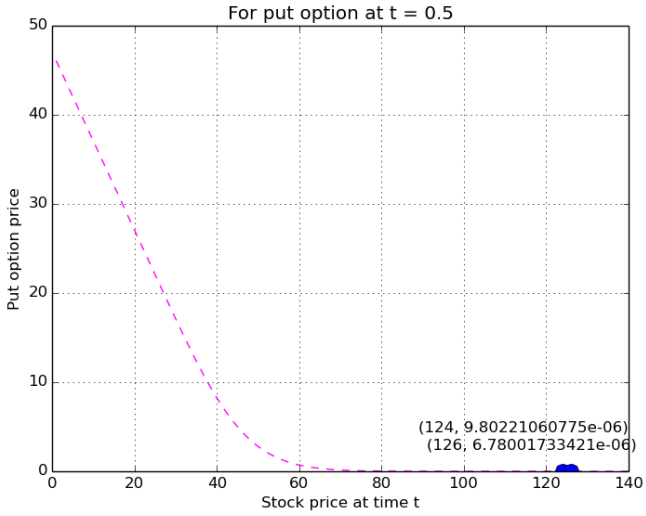
t=0: (f(126) - f(124)) / 2
=(80.6555277391-78.6558752174)/2
=0.99982

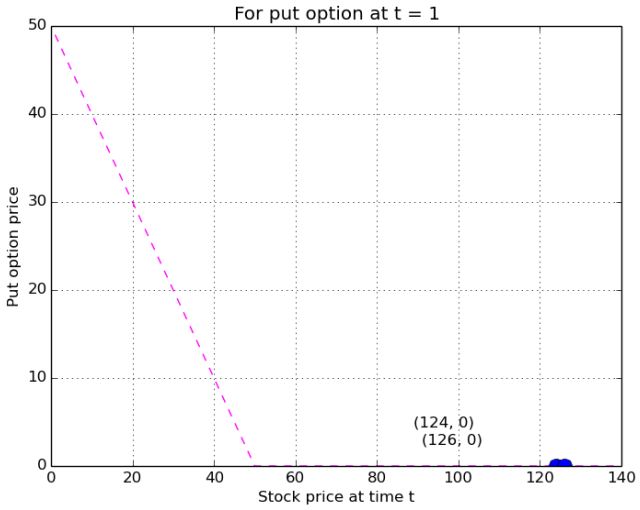
t=0.5:(f(126) - f(124)) / 2
=(77.9117801008-75.911783123)/2
=0.99999

t=1:(f(126) - f(124)) / 2
=(75.00-73.00)/2
=1.00

Graphs for put options







Put option:

t=0: (f(126) - f(124)) / 2

=(0.00154957500039-0.00189705322866)/2

=-0.00017374

t=0.5:(f(126) - f(124)) / 2

=(6.7800173342e-06-9.80221060775e-06)/2

$=-1.5110966367714673*10^{-6}$

t=1:(f(126) - f(124)) / 2

=(0-0)/2

=0.00

Conclusions of of numerical calculation of delta (using graphs):

Table 03

t (years)	**Delta for call option**	**Delta for put option**
0	0.99982	-0.00017374
0.5	0.99999	$-1.5110966367714673*10^{-6}$
1	1.00	0.00

Comparison:

Table 04

Delta call (Analytical)	**Delta call (Numerical)**	**Delta put (Analytical)**	**Delta put (Numerical)**
0.999843503789	0.9998	-0.00015649621068	-0.00017374

0.999998751317	1.00	-1.24868286084e-06	$-1.5110966367714673*10^{-6}$
1.00	1.00	0.00	0.00

Thus, Comparing the 2 tables, we see that our calculations are quite accurate

Source code: (for plotted graphs)

'MAHEK_VORA_CASE_STUDY_A_Q4_B_MAIN.py'

```
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
#increasing the x axis dimensions, to calculate slope at S=125
S= np.linspace(1.0,1,140.0)
def calculate(T,S,K,sigma,t,r):
  T=float(T)
  S=float(S)
  K=float(K)
  sigma=float(sigma)
  t=float(t)
  r=float(r)
  if t!=T:
      d1=(1/(sigma*np.sqrt(T-t)))*(np.log(S/K)+((r+(sigma*sigma/2))*(T-t)))
      d2=d1-sigma*(np.sqrt(T-t))
      C=S*(norm.cdf(d1))-norm.cdf(d2)*K*np.exp(-1*r*(T-t))
      P=norm.cdf(-d2)*K*np.exp(-1*r*(T-t))-S*(norm.cdf(-d1))
  if t==T:
      C=max(S-K,0)
      P=max(K-S,0)
  return C,P
T=1
S=0
K=50
sigma=0.3
r=0.12
def plot_graphs(t):
 X=[]
 Y1=[]
 Y2=[]
 for i in range(140):
   #store coordinates for these points, S=124 and 126
     if((i+1)==124):
          call_124=C
          put_124=P
     if((i+1)==126):
          call_126=C
          put_126=P
     C,P=calculate(T,i+1,K,sigma,t,r)
     Y1.append(C)
     Y2.append(P)
     X.append(i+1)
 plt.figure()
```

```
 plt.plot(X,Y1,linestyle='dashed',color='magenta')
 x = [124,126]
 y = [call_124,call_126]
 plt.text(124-25, call_124+2.5, '({}, {})'.format(124, call_124))
 plt.text(126-25, call_126-2.5, '({}, {})'.format(126, call_126))
 plt.plot(x, y, color='green', linestyle='dashed', linewidth = 3,
        marker='o', markerfacecolor='blue', markersize=12)
 plt.grid('on')
 plt.xlabel('Stock price at time t')
 plt.ylabel('Call option price')
 plt.title('For call option at t = '+str(t))
 plt.show()
 plt.figure()
 plt.plot(X,Y2,linestyle='dashed',color='magenta')
 x2 = [124,126]
 y2 = [put_124,put_126]
 #plot points and coordinates
 plt.text(124-35, put_124+4.5, '({}, {})'.format(124, put_124))
 plt.text(126-35, put_126+2.5, '({}, {})'.format(126, put_126))
 print(put_124,put_126,(put_124-put_126)/2)
 plt.plot(x2, y2, color='green', linestyle='dashed', linewidth = 3,
        marker='o', markerfacecolor='blue', markersize=12)
 plt.grid('on')
 plt.xlabel('Stock price at time t')
 plt.ylabel('Put option price')
 plt.title('For put option at t = '+str(t))
 plt.show()
if __name__=='__main__':
 plot_graphs(0)
 plot_graphs(0.5)
 plot_graphs(1)
```

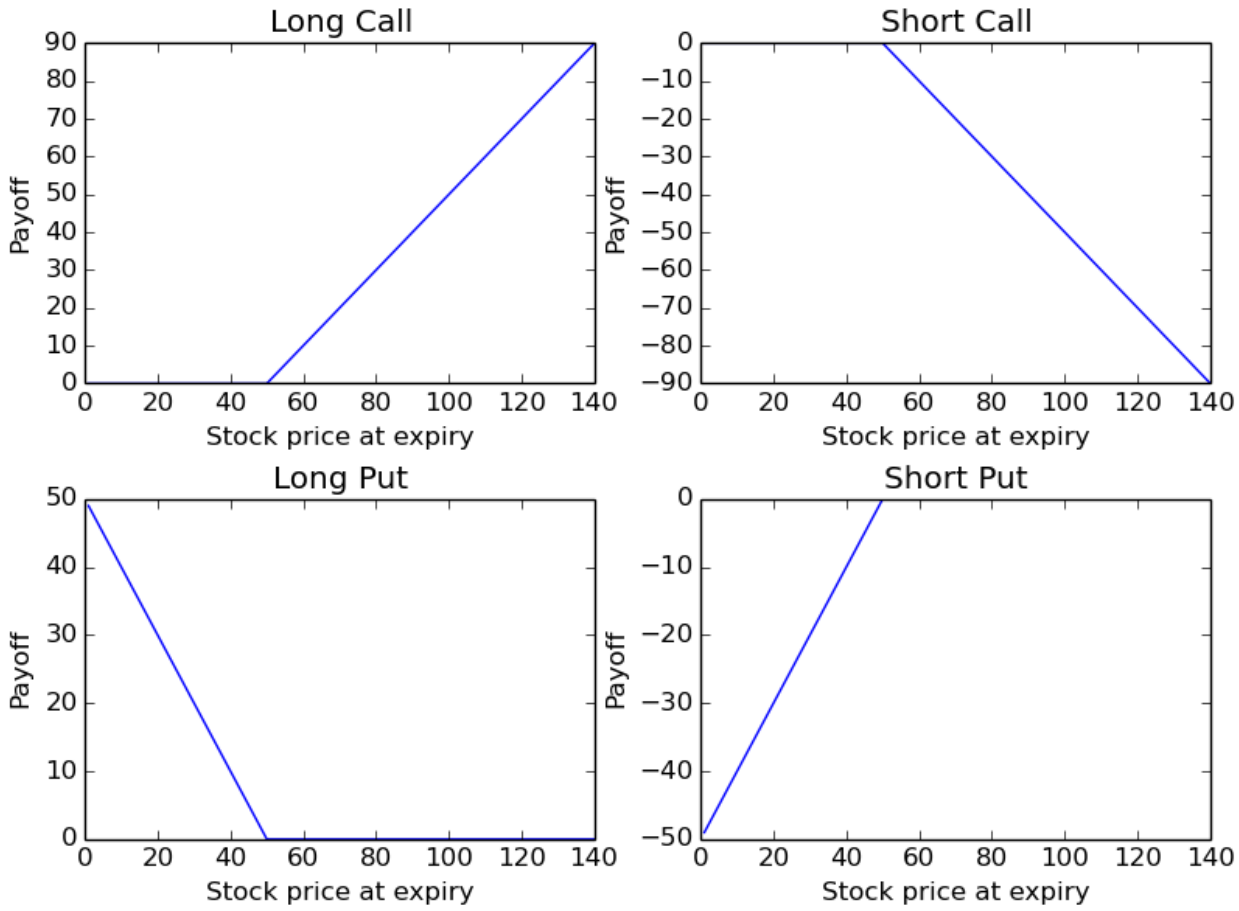
Q.5 Delta Hedging

1: Solution:

This question has 2 main parts:

1. Breaking down the given payoff into an **equivalent linear combination of call and put options**
2. Perform **delta hedging** in each case and create riskless portfolios

For **parts a and b** I have used the following 4 payoff graphs with when premium is 0, and these graphs can be **shifted up or down according to the premium**. Note that these are **payoffs plotted at expiry**



The following are the strategies commonly adopted in delta hedging:
We are only using hedging through stocks

Table 05

To Reduce Payoff	**To Increase Payoff**
Short Stock Long Puts Short Calls	Long Stock Long Calls Short Puts

Delta contributions:

Table 06

Option	**Delta Contribution**
Long Call	(Delta of call option)=$N(d_1)$
Short Call	-(Delta of call option)=-$N(d_1)$
Long Put	(Delta of put option)=-$N(-d_1)$
Short Put	-(Delta of put option)=$N(-d_1)$

Q5.a:

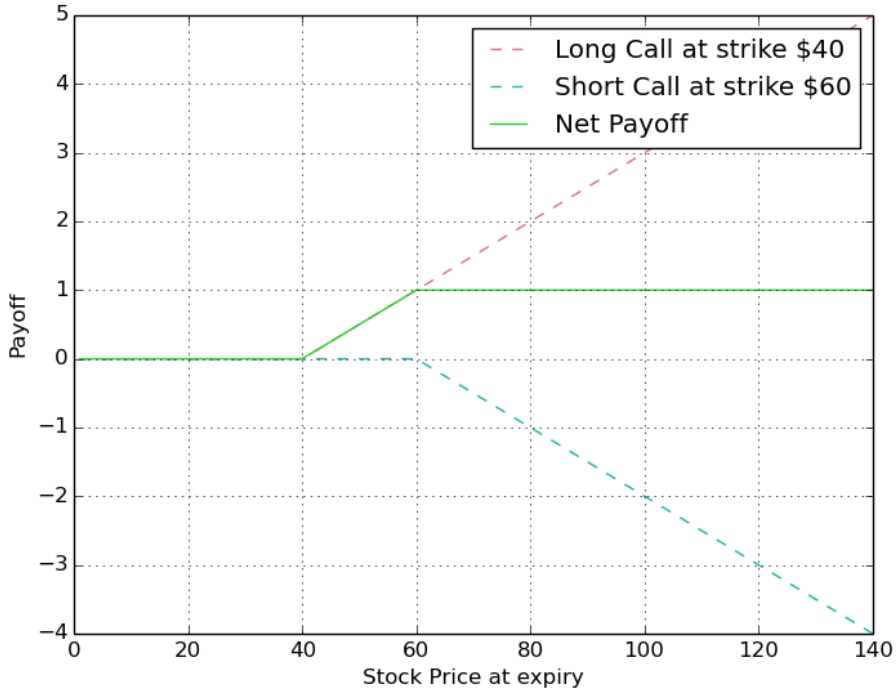
1: Answer:

The graph represents a 'Bull call spread' (it can also be made using bull put spread when premiums are high)

n such options can be achieved using a linear combination of

1. n*0.05 long calls at strike $40
2. n*0.05 short calls at strike $60

2: Explanation:



For now let there be **'x' long calls and 'y' short calls** for **1 option**
For different ranges of the Stock price at expiry, contribution of the options to the payoff at expiry is shown here:

Table 07

	0-40	**40-60**	**60+**
Long Call, $40 **max(S-K,0)** **= max(S-40,0)**	x*max(0,S-40) =0	x*max(0,S-40) =x*(S-40)	max(0,S-40) =x*(S-40)
Short Call, $60 **min(K-S,0)** **= min(60-S,0)**	y*min(60-S,0) =0	y*min(60-S,0) =0	min(60-S,0) =y*(60-S)

Total Payoff (at expiry)	0	x*(S-40)	x*(S-40)+y*(60-S)
Given total payoff	0	0.05*(S-40)	20

Solving the last 2 rows by comparing payoffs we get

0.05*(S-40)=x*(S-40)
So, **x=0.05**

0.05*(S-40)+y*(60-S)=20
(0.05-y)*S + 20 = 20 for all S>60
So, **y=0.05**

Hence for n options we have n*0.05 long calls at a strike price $40 and n*0.05 short calls at a strike price $60

Riskless portfolio:

i: S_t=$30

Using Table 06 for delta contributions,
We have **n*0.05** sets of these options. Calculating net delta & delta contributions for **1** such set:
Long Call: [$N(d_1)$]
Here,
S_t=30, K=40, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 = -0.408940241506
N(d1)=**0.341291758139**

Short Call: [$-N(d_1)$]
Here,
S_t=30, K=60, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 = -1.76049060187
-N(d1)=**-0.0391623297991**

Net Delta: 0.341291758139-0.0391623297991
=**0.30212942834**

Since the net delta is positive we must short the underlying stocks

The number of stocks shorted=(delta*number of options)
So,we must short 0.30212942834*n*(0.05) stocks
=0.015106471417*n stocks

The riskless portfolio is thus =
[-n*0.015106471417 stocks]

ii: S_t=$50
Using Table 06 for delta contributions,
We have **n*0.05** sets of these options. Calculating net delta & delta contributions for **1** such set:
Long Call: [$N(d_1)$]
Here,
S_t=50, K=40, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 = 1.29381183771
N(d1)=**0.902134788623**

Short Call: [$-N(d_1)$]
Here,
S_t=50, K=60, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 = -0.0577385226465
-N(d1)=**-0.476978454115**

Net Delta: 0.902134788623-0.476978454115
=0.425156334508

Since the net delta is positive we must short the underlying stocks
The number of stocks shorted=(delta*number of options)
So,we must short 0.425156334508*n*(0.05) stocks
=0.0212578167254*n stocks

The riskless portfolio is thus =
[-n*0.0212578167254 stocks]

iii: S_t=$70
Using Table 06 for delta contributions,
We have **n*0.05** sets of these options. Calculating net delta & delta contributions for **1** such set:
Long Call: $N(d_1)$

Here,
S_t=70, K=40, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 = 2.41538595978
N(d1)=**0.992140728093**

Short Call: -N(d_1)
Here,
S_t=70, K=60, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula, d1 = $(1/(\sigma*(T-t)^{1/2}))*(\log(S/K)+(r + \sigma^2/2)*(T-t))$
d1 = 1.06383559942
-N(d1)=**-0.856298409355**

Net Delta: 0.992140728093-0.856298409355
=0.135842318739

Since the net delta is positive we must short the underlying stocks
The number of stocks shorted=(delta*number of options)
So,we must short 0.135842318739*n*(0.05) stocks
=0.00679211593694*n stocks

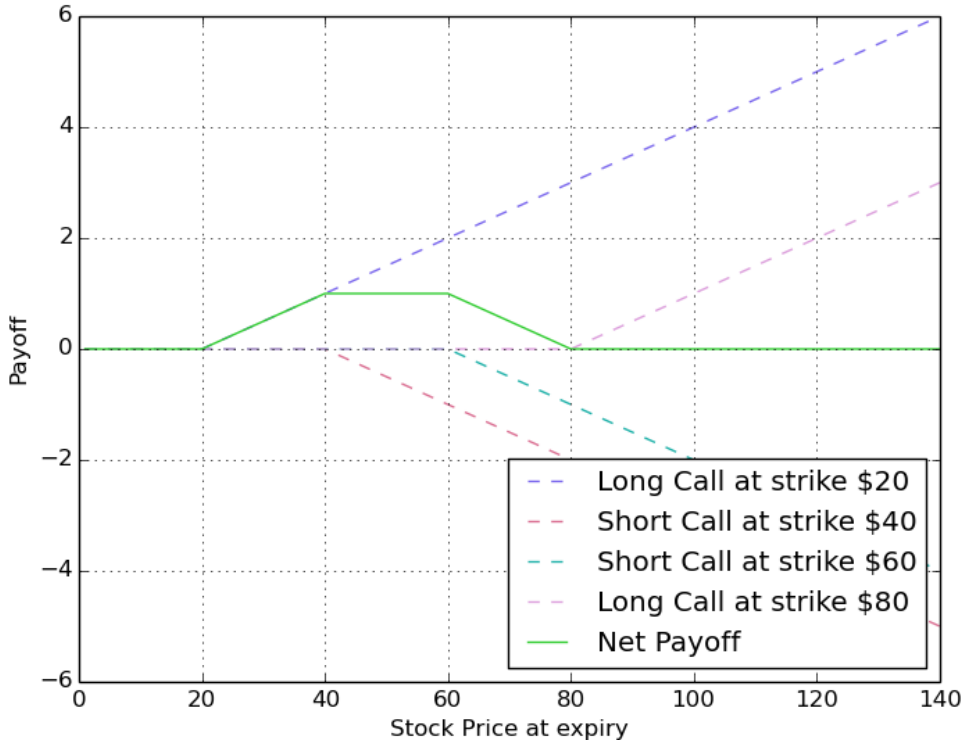
The riskless portfolio is thus =
[-n*0.00679211593694 stocks]
Q5.b:

1: Answer:
This payoff graph represents the 'Iron condor strategy-(Long Condor)'
It can be achieved as a combination of 'Bull call spread' and 'Bear call spread'

n such options can be achieved using a linear combination of

1. n*(0.05) long calls at strike price $20 and premium $0
2. n*(0.05) short calls at strike price $40 and premium $0
3. n*(0.05) short calls at strike price $60 and premium $0
4. n*(0.05) long calls at strike price $80 and premium $0

2: Explanation:



For now let there be **'x' long calls at strike 20, 'y' short calls at strike 40,'z' short calls at strike 60, 'x' long calls at strike 80** for **1 option**

For different ranges of the Stock price at expiry, contribution of the options to the payoff at expiry is shown here:

Table 08

	0-20	**20-40**	**40-60**	**60-80**	**80+**
Long Call, $20 **max(S-K,0) = max(S-20,0)**	x*max(S-20,0) =0	x*max(S-20,0) =x*(S-20)	x*max(S-20,0) =x*(S-20)	x*max(S-20,0) =x*(S-20)	x*max(S-20,0) =x*(S-20)
Short Call, $40 **min(K-S,0) = min(40-S,0) +10**	y*[min(40-S,0)] =0	y*[min(40-S,0)] =0	y*[min(40-S,0)] =y*(40-S)	y*[min(40-S,0)] =y*(40-S)	y*[min(40-S,0)] =y*(40-S)
Short Call, $60 **min(0,K-S)**	z*[min(0,60-S)] =0	z*[min(0,60-S)] =0	z*[min(0,60-S)] =0	z*[min(0,60-S)] =z*(60-S)	z*[min(0,60-S)] =z*(60-S)

= min(0,60-S) +10					
Long Call, $80 **max(0,S-K) = max(0,S-80)**	w*max(0,S-80) =0	w*max(0,S-80) =0	w*max(0,S-80) =0	w*max(0,S-80) =0	w*max(0,S-80) =w*(S-80)
Total Payoff (at expiry)	0	x*(S-20)	x*(S-20)+y*(40-S)	x*(S-20)+y*(40-S)+z*(60-S)	x*(S-20)+y*(40-S)+z*(60-S)+w*(S-80)
Given Payoff	0	0.05*(S-20)	1	0.05*(80-S)	0

Comparing the last 2 rows,
x*(S-20)=0.05*(S-20)==>x=0.05
0.05*(S-20)+y*(40-S)=1==>y=0.05
x*(S-20)+y*(40-S)+z*(60-S)=0.05*(80-S)==>z=0.05
x*(S-20)+y*(40-S)+z*(60-S)+w*(S-80)=0==>w=0.05
Solving these equations, **x=y=z=w=0.05**

Hence for n options, we have n*0.05 n*(0.05) long calls at strike price $20,$80 and n*(0.05) short calls at strike price $40,$60
Riskless portfolio:

i: S_t=$10
Using Table 06 for delta contributions,
We have **n*0.05** sets of these options. Calculating net delta & delta contributions for **1** such set:
Long call at 20: [$N(d_1)$]
Here,
S_t=10, K=20, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 =-1.76049060187
$N(d_1)$=**0.0391623297991**
Short call at 40: -[$N(d_1)$]
Here,
S_t=10, K=40, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 = -4.07098120373
-N(d1)=**$-2.34077636308*10^{-5}$**

Short Call at 60: [$-N(d_1)$]

Here,

S_t=10, K=60, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,

d1 =-5.42253156409

-N(d1)=**$-2.93804171571*10^{-8}$**

Long Call at 80: [$N(d_1)$]

Here,

S_t=10, K=80, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,

d1 = -6.3814718056

N(d1)=**$8.76970087494*10^{-11}$**

Net Delta: =0.0391388927428

Since the net delta is positive we must short the underlying stocks

The number of stocks shorted=(delta*number of options)

So,we must short 0.0391388927428*n*(0.05) stocks

=0.00195694463714*n stocks

The riskless portfolio is thus =

[-n*0.00195694463714 stocks]

ii: S_t=$30

Using Table 06 for delta contributions,

We have **n*0.05** sets of these options. Calculating net delta & delta contributions for **1** such set:

Long call at 20: [$N(d_1)$]

Here,

S_t=10, K=20, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,

d1 =1.90155036036

$N(d_1)$=**0.971385018619**

Short call at 40: -[$N(d_1)$]

Here,

S_t=10, K=40, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,

d1 = -0.408940241506

-N(d1)= **-0.341291758139**

Short Call at 60: [$-N(d_1)$]

Here,
S_t=30, K=60, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 = -1.76049060187
-N(d1)=**-0.0391623297991**

Long Call at 80: [$N(d_1)$]
Here,
S_t=30, K=80, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 = -2.71943084337
N(d1)=**0.00326971839925**

Net Delta: =0.59420064908
Since the net delta is positive we must short the underlying stocks
The number of stocks=(delta*number of options)
So,we must short *n*(0.05) stocks
=0.59420064908*n stocks

The riskless portfolio is thus =
[-n*0.029710032454 stocks]

iii: S_t=$50
Using Table 06 for delta contributions,
We have **n*0.05** sets of these options. Calculating net delta & delta contributions for **1** such set:
Long call at 20: [$N(d_1)$]
Here,
S_t=10, K=20, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 =3.60430243958
$N(d_1)$=**0.999843503789**
Short call at 40: -[$N(d_1)$]
Here,
S_t=10, K=40, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 = 1.29381183771
-N(d1)=**-0.902134788623**

Short Call at 60: [-$N(d_1)$]
Here,

S_t=10, K=60, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 =-0.0577385226465
-N(d1)=**-0.476978454115**

Long Call at 80: [$N(d_1)$]
Here,
S_t=10, K=80, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 = -1.01667876415
N(d1)=**0.15465313558**

Net Delta: = -0.224616603369
Since the net delta is negative we must long the underlying stocks
The number of stocks=(delta*number of options)
So,we must long 0.224616603369*n*(0.05) stocks
=0.0112308301684*n stocks

The riskless portfolio is thus =
[n*0.0112308301684 stocks]

iv: S_t=$70
Using Table 06 for delta contributions,
We have **n*0.05** sets of these options. Calculating net delta & delta contributions for **1** such set:
Long call at 20: [$N(d_1)$]
Here,
S_t=10, K=20, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 =4.72587656165
$N(d_1)$=**0.999998854376**
Short call at 40: -[$N(d_1)$]
Here,
S_t=10, K=40, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 = 2.41538595978
-N(d1)= **-0.992140728093**

Short Call at 60: [-$N(d_1)$]
Here,
S_t=70, K=60, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,

d1 = 1.06383559942
-N(d1)=**-0.856298409355**

Long Call at 80: [$N(d_1)$]

Here,
S_t=70, K=80, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 = 0.104895357918
N(d1)=**0.541770578754**

Net Delta: =-0.306669704319

Since the net delta is negative we must long the underlying stocks
The number of stocks=(delta*number of options)
So,we must long 0.306669704319*n*(0.05) stocks
=0.0153334852159*n stocks

The riskless portfolio is thus =
[n*0.0153334852159 stocks]

v: S_t=$90
Using Table 06 for delta contributions,
We have **n*0.05** sets of these options. Calculating net delta & delta contributions for **1** such set:
Long call at 20: [$N(d_1)$]

Here,
S_t=10, K=20, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 =5.56359132259
$N(d_1)$=**0.999999986786**

Short call at 40: -[$N(d_1)$]

Here,
S_t=10, K=40, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 = 3.25310072072
-N(d1)= **-0.999429234864**

Short Call at 60: [-$N(d_1)$]

Here,
S_t=10, K=60, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,
d1 =1.90155036036

-N(d1)=**-0.971385018619**

Long Call at 80: [$N(d_1)$]

Here,

S_t=10, K=80, r=0.12, σ = 0.30, T=1, t=0 substituting in the formula,

d1 = 0.942610118855

N(d1)=**0.82705981892**

Net Delta: =-0.143754447778

Since the net delta is negative we must long the underlying stocks

The number of stocks=(delta*number of options)

So,we must long 0.143754447778*n*(0.05) stocks

=0.00718772238889*n stocks

The riskless portfolio is thus =

[n*0.00718772238889 stocks]

Q5.c:

This curve can be obtained by:

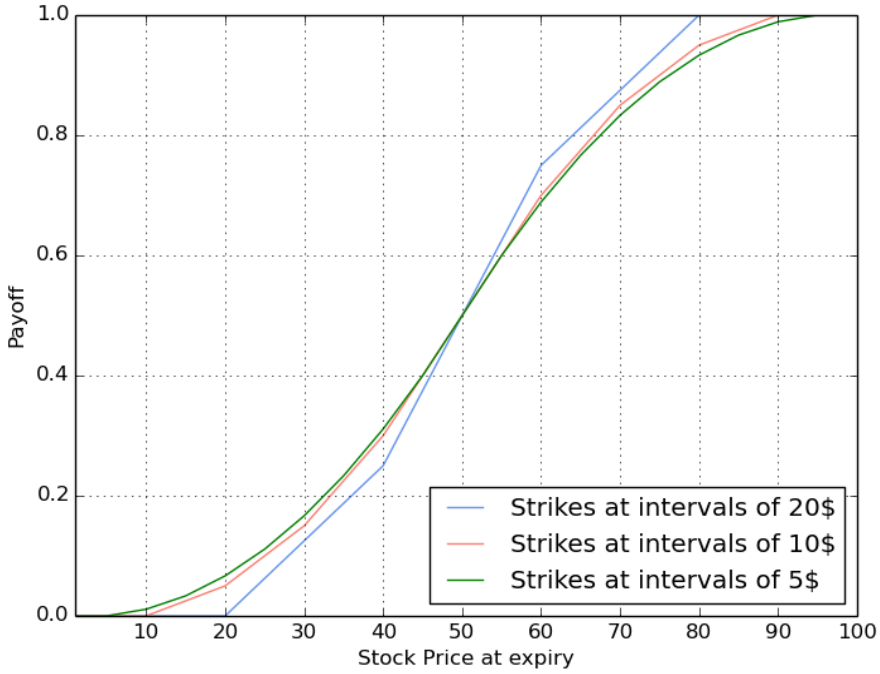
Answer:

Making an infinite amount of long call options with strike prices varying uniformly from $0 to $50 followed by an infinite amount of short call options with strike prices varying uniformly from $50 to $100.

Note that for 1 unit of this main option we divide the (number of options used) by the total payoff of our main option,so that the total payoff of the main option received is 1.00

Explanation:

As the interval between strike prices of 2 successive options decreases we get the graph resembles the required curve more



So we can take options separated by interval ε where ε –> 0
The delta for this curve will be obtained as follows:

The total payoff will be 50*((50/ε)-1)
The number of derivatives we use will thus be (1/(50*((50/ε)-1))), or (ε/(50*(50-ε))) to make the payoff 0.

$$\int_0^{50} N(d_1)\,dK + \int_{50}^{100} -N(d_1)\,dK$$

where,

$$N(d_1) = N\left(\frac{1}{\sigma\sqrt{T-t}}\left[\ln\left(\frac{S_t}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right]\right)$$

Putting
$\sigma = 0.3$
$r = 0.12$
$T = 1$
$t = 0$
$\Rightarrow \int_0^T N\left(\frac{\ln\left(\frac{S_t}{K}\right)+0.165}{0.3}\right) dK - \int_{50}^{100} N\left(\frac{\ln\left(\frac{S_t}{K}\right)+0.165}{0.3}\right) dK$

Using this equation,
For S_t=10:
Net delta=12.336776087
As net delta is positive we need to short (delta*no.of options) stocks
=12.336776087*(ε/(50*(50-ε)))
For S_t=30:
Net delta=34.8885822707

As net delta is positive we need to short (delta*no.of options) stocks
=34.8885822707*(ε/(50*(50-ε)))

For S_t=50:

Net delta=34.1151200351

As net delta is positive we need to short (delta*no.of options) stocks
=34.1151200351*(ε/(50*(50-ε)))

For S_t=70:

Net delta=18.6924919608

As net delta is positive we need to short (delta*no.of options) stocks
=18.6924919608*(ε/(50*(50-ε)))

For S_t=90:

Net delta=7.76320664497

As net delta is positive we need to short (delta*no.of options) stocks
=7.76320664497*(ε/(50*(50-ε)))

In all these cases as ε tends to 0 our answer for the required number of stocks to hedge are also 0

The calculations were made using this **source code:**

'MAHEK_VORA_CASE_STUDY_A_Q5_C_MAIN.py'

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm
from scipy.integrate import quad
def solve(S):
   sigma=0.3
   r=0.12
   T=1
   t=0
   T=float(T)
   S=float(S)
   sigma=float(sigma)
   t=float(t)
   r=float(r)
   def f(x):
       return norm.cdf((0.165+np.log(S/x))/0.3)
   return quad(f,0,50)[0]-quad(f,50,100)[0]

if __name__=="__main__":
   D1=solve(10)
   print(D1)
   D2=solve(30)
   print(D2)
   D3=solve(50)
   print(D3)
   D4=solve(70)
   print(D4)
   D5=solve(90)
   print(D5)
```

Source Code for graphs:**'MAHEK_VORA_CASE_STUDY_A_Q5_AB_MAIN.py'**

```
import numpy as np
```

```
import matplotlib.pyplot as plt
S= np.linspace(1.0,1,140.0)
def calculate(T,S,K,sigma,r):
  T=float(T)
  S=float(S)
  K=float(K)
  sigma=float(sigma)
  r=float(r)
  C=max(S-K,0)
  P=max(K-S,0)
  return C,P
T=1
S=0
sigma=0.3
r=0.12
def plot_graphs():
 X=[]
 Y0_1=[]
 Y0_2=[]
 Y0_3=[]
 Y1_1=[]
 Y1_2=[]
 Y1_3=[]
 Y1_4=[]

 Y=[]
 for i in range(140):
     C1,P1=calculate(T,i+1,20,sigma,r)
     C2,P2=calculate(T,i+1,40,sigma,r)
     C3,P3=calculate(T,i+1,60,sigma,r)
     C4,P4=calculate(T,i+1,80,sigma,r)
     Y0_1.append(C2/20)
     Y0_2.append(-1*C3/20)
     Y0_3.append((C2-C3)/20)
     Y1_1.append(C1/20)
     Y1_2.append(-C2/20)
     Y1_3.append(-C3/20)
     Y1_4.append(C4/20)
     Y.append((C1+C4-C2-C3)/20)
     X.append(i+1)
 plt.figure()
 plt.title('')
 plt.plot(X,Y0_1,linestyle='dashed',color='palevioletred')
 plt.plot(X,Y0_2,linestyle='dashed',color='lightseagreen')
 plt.plot(X,Y0_3,linestyle='solid',color='limegreen')
 plt.xlabel('Stock Price at expiry')
 plt.ylabel('Payoff')
 plt.grid('on')
 plt.legend(['Long Call at strike $40','Short Call at strike $60','Net
Payoff'])
 plt.show()

 plt.figure()
 plt.title('')
```

```
 plt.plot(X,Y1_1,linestyle='dashed',color='mediumslateblue')
 plt.plot(X,Y1_2,linestyle='dashed',color='palevioletred')
 plt.plot(X,Y1_3,linestyle='dashed',color='lightseagreen')
 plt.plot(X,Y1_4,linestyle='dashed',color='plum')
 plt.plot(X,Y,linestyle='solid',color='limegreen')
 plt.xlabel('Stock Price at expiry')
 plt.ylabel('Payoff')
 plt.grid('on')
 plt.legend(['Long Call at strike $20','Short Call at strike $40','Short Call
at strike $60','Long Call at strike $80','Net Payoff'],loc=4)
 plt.show()
if __name__=='__main__':
 plot_graphs()
```

CASE STUDY - B

Q.1 Circles?

Q1.1:

Answer:

Area ratio for n=3: **0.7074317238462213 = 0.70343 or 70.74317%**

Solution:

Answer is derived programmatically **the code for Q1.3 is used here**

Logic:

Take the bottom left corner of the rectangle as the origin

The equation of the diagonal (line passing through (0,0)and(6r,2r)):

$y=x*3$

The equation of the circle with centre (r,r) and radius r:

$(x-r)^2+(y-r)^2=r^2$

The lower half of the circle thus has this equation for y:

$y=r-(r^2-(x-r)^2)^{½}$

The area between curve representing the minimum of these 2 curves and x-axis is given by B

Also, A+B can be found as it is the area of the square with side r-area of ¼th circle with radius r

Hence $A+B=(1-\pi/4)r^2$

A=A+B-B

Ratio=A/(A+B)

As both A and A+B have a term of **r^2**, it will get cancelled out so for calculations, we can take r=1.

Source code: Given in the file '**MAHEK_VORA_CASE_STUDY_B_Q1_1_MAIN.py**'

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import quad

# defining a function as the line(diagonal) and circle.
# area between minimum of these 2 functions and x axis is B
```

```
def find_area_ratio(generate_plots, show_roundoff_answer):
 def f(x):
   return x/3
 def g(x):
   return ((-1)*(np.sqrt(1-(x-1)**2))+1)
 def h(x):
   return np.minimum(g(x),f(x))
  # if we wish to generate plots this can be passed as true, the generated
plot was used in Figure 10 by me
 if generate_plots:
   x = np.linspace(0,1,1000)

   plt.plot(x,f(x))
   plt.plot(x,g(x))

   plt.fill_between(x, f(x), color = "cyan",alpha = 0.3, hatch = '|')
   plt.fill_between(x, g(x), color = "pink", alpha = 0.2, hatch = '-')
   plt.show()

   plt.plot(x,h(x))
   plt.xlabel('X-axis')
   plt.ylabel('Y-axis')
   plt.fill_between(x, h(x), color = "magenta",alpha = 0.3, hatch = '|')
   plt.show()
  # Calculate B by integration
 B=quad(h,0,1)
 # Total is the area between circle and X axis from x=0 to x=r, here r=1
 Total=quad(g,0,1)

 # A/Total=(Total-B)/Total
 area_ratio=(Total[0]-B[0])/Total[0]
 if show_roundoff_answer:
   print(round(area_ratio,5))
 return area_ratio

if __name__ == "__main__":
 find_area_ratio(False,True)
```

Q1.2:

1: Assumptions:

Let n be a natural number as it is the number of circles arranged in the rectangle

The minimum value of n is thus 1, maximum value is infinity.

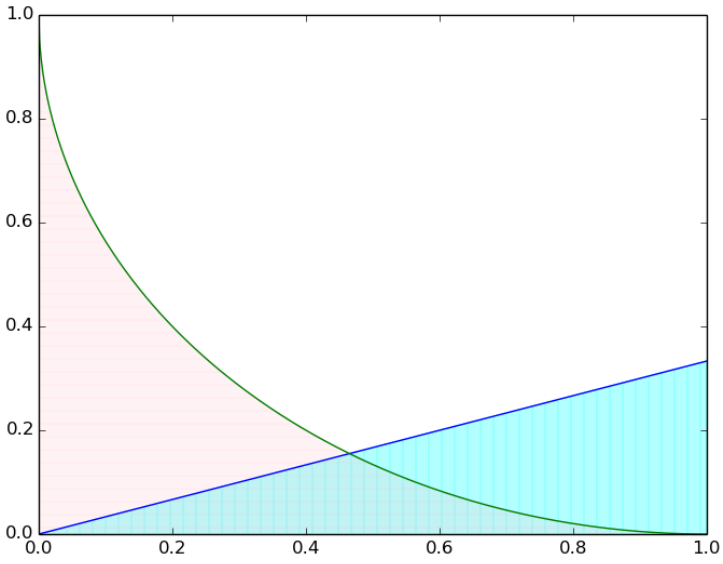
2: Answer:

Range of Area ratio for : [0.5, 1)

3: Explanation

The explanation can be split in 2 parts as follows
Part 1:
Claim:
The function of Area ratio with respect to the independent variable n is an increasing function

Figure 10



Proof:
Let T be the area denoted by the pink part and B is the area given by the intersection of the pink and blue regions

Area ratio = (T-B)/T
T does not vary with n, as it is the (area of square with side r) - (circle with radius r)

So, B is the only part that affects area ratio
Note that as slope of the blue line increases, the blue region increases, or (T-B) decreases.
This slope can be obtained in terms of n as it is the diagonal of the given rectangle, slope=$(y_2-y_1)/(x_2-x_1)$ = r/nr = 1/n
[Taking the 2 points on the diagonal as end points, (0,0) and (2nr,2r) where origin is the bottom left corner of the rectangle]

Hence the slope decreases and (T-B)/T increases as n increases
So, Area ratio is an increasing function in n
Part 2:

For an increasing function the minimum and maximum values in the given domain are at the end-points of the domain.
Here as our domain is natural numbers from 1 to infinity,

Minimum value : Area Ratio at n = 1, which is calculated to be 0.5 by code

Maximum value : Area Ratio as n → ∞, which is calculated to be 1 by code, This value is not included in the range of n, as n can never truly be infinity

This is further illustrated by this graph plotted using python:
The variation of n with area ratio:

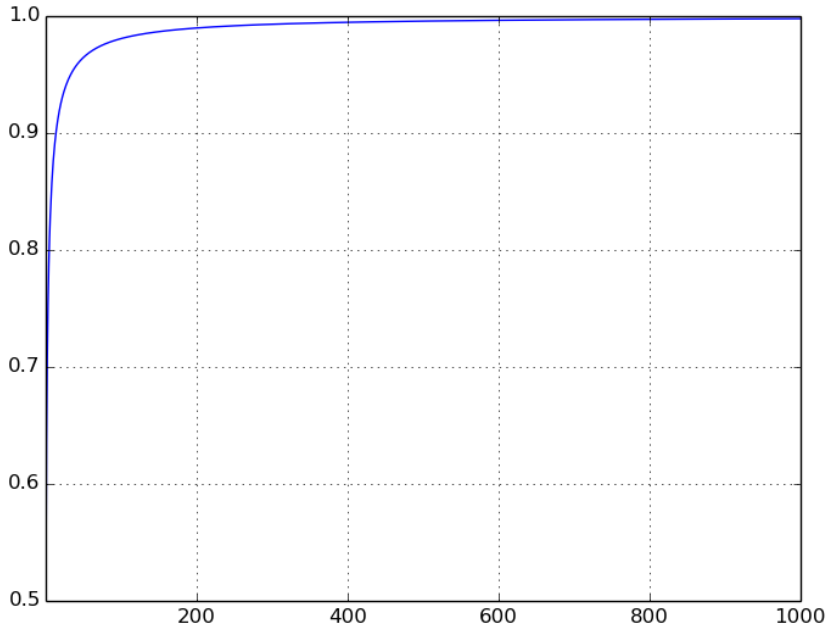


Figure 11
[Even though this plot reaches 1 at n=1000 it actually is asymptotically tending towards 1, and this is only due to the inaccuracies in plotting as the minimum distance here is 0.1 units]

Source code: (for the graph plot) It has been implemented using the source code pr Q1.3, 'MAHEK_VORA_CASE_STUDY_B_Q1_3_MAIN'

Q1.3:
Answer:
a. 50% : 1
b. 70% : 3
c. 90% : 15
d. 99.9% : 2240
e. 99.99%: 23012
f. 100% : infinity, i.e. no real number

Explanation:
Answer is derived programmatically

The 1st part is a generalisation of the code used in Q1.1, i.e. here the solution has a callable function where n is a variable

Take the bottom left corner of the rectangle as the origin

The equation of the diagonal (line passing through (0,0)and(2nr,2r)):

$y=x*n$

The equation of the circle with centre (r,r) and radius r:

$(x-r)^2+(y-r)^2=r^2$

The lower half of the circle thus has this equation for y:

$y=r-(r^2-(x-r)^2)^{½}$

The area between curve representing the minimum of these 2 curves and x-axis is given by B

Also, A+B can be found as it is the area of the square with side r-area of ¼th circle with radius r

Hence $A+B=(1-\pi/4)r^2$

A=A+B-B

Ratio=A/(A+B)

As both A and A+B have a term of r^2, it will get cancelled out so for calculations, we can take r=1.

The required answer for value[of area ratio] will be the first n where f(n)>=value [f(n) is area ratio for n='n']

So, I computed the values of area ratio from n=1 to the point where the area_ratio exceeds or is equal to value, as long as n<23020

[Done only once after program is run, not every time the function is called as it would be the same for every call]

I found the limit 23020 after some amount of trial and error and studying the graph in Figure 11 with different ranges of x.

Also for ratio=1, as proved in Q1.2, no real value of n exists for it.

This function also plots a graph of area ratio vs n and y=n to show the intersecting curves(required solution) (user can choose not to plot the graph as a boolean variable is passed as an argument to the function for this reason)

Source code : 'MAHEK_VORA_CASE_STUDY_B_Q1_3_MAIN'

(It takes about 2 minutes for execution as it is computing ratios many values)

Kindly install matplotlib, numpy and scipy for execution

```
import matplotlib.pyplot as plt
import numpy as np
# from scipy.optimize import fsolve
from scipy.integrate import quad
```

```
# 1st part- which is the same as a generalized version of the code for n=3
def find_area_ratio(n, generate_plots, show_answer,show_roundoff_answer):
 def f(x):
   return x/n
 def g(x):
   return ((-1)*(np.sqrt(1-(x-1)**2))+1)
 def h(x):
   return np.minimum(g(x),f(x))
  # can be passed as true if we want to see the figure as n varies
 if generate_plots:
   x = np.linspace(0,1,1000)

   plt.plot(x,f(x))
   plt.plot(x,g(x))

   plt.fill_between(x, f(x), color = "cyan",alpha = 0.3, hatch = '|')
   plt.fill_between(x, g(x), color = "pink", alpha = 0.2, hatch = '-')
   plt.show()

   plt.plot(x,h(x))
   plt.xlabel('X-axis')
   plt.ylabel('Y-axis')
   plt.fill_between(x, h(x), color = "magenta",alpha = 0.3, hatch = '|')
   plt.show()
  # area B and area Total are computed by integration
 B=quad(h,0,1)
 Total=quad(g,0,1)
 area_ratio=(Total[0]-B[0])/Total[0]
 if show_answer:
   print(area_ratio)
 if show_roundoff_answer:
   print(round(area_ratio))
 return area_ratio

# generating data for n=1 to n=23020,
# this data will be used to find the lowest n required for getting a
particular ratio.
# here(stored values from 1 to 23020) the values used are accurate, not
rounded off to 5 places
def generate_area_ratio_vs_n():
print('Started Execution, Please Wait...')
X=[]
Y=[]
for i in range(23020):
  Y.append(find_area_ratio(1+i,False,False,False))
  X.append(i+1)
plt.plot(X,Y)
return X,Y
X,Y=generate_area_ratio_vs_n()
def find_lower_bounds(val,generate_plots,show_answer):
 ans=np.Inf
 for i in range(len(Y)):
   if Y[i]>=val:
```

```
      ans=(i+1)
      break
 if generate_plots:
   plt.figure()
   plt.grid(True)
   plt.xlim(1,max(23020,ans*2))
   plt.plot(X,Y)
   plt.show()
 if show_answer:
   print('The minimum n required to get the area ratio ' +str(val)+ ' is :
'+str(ans))
 return ans

if __name__ == "__main__":
  find_lower_bounds(0.5,False,True)
  find_lower_bounds(0.7,False,True)
  find_lower_bounds(0.9,False,True)
  find_lower_bounds(0.999,False,True)
  find_lower_bounds(0.9999,False,True)
  find_lower_bounds(1.0,False,True)
  print('Done.')
```

Q.2 Keep Calm and Carry on

Q2.1:
Answer:
a. Cojelo Con Take It Easy : “S S E S E”: 0.00003
b. Con Take It Easy : “S E S E”: 0.00027
c. Easy Con : “E S”: 0.02520
d. Cojelo Easy Take It : “S E S E”: 0.00018

Explanation:
The logic used here to compute the answers is same as Q2.2, so explanation is combined for both the parts
Note that I have also made a code that verifies my method
Source Code: ‘MAHEK_VORA_CASE_STUDY_B_Q2_1_MAIN’

```
#given a sentence, break it into words, note down its emission probabilities
def make_input(input_sentence):
   val=[]
   input_sentence=input_sentence.strip()
   input_words=input_sentence.split()
   for i in input_words:
       i=i.lower()
       if i=='cojelo':
           val.append([0.1,0.3])
       if i=='con':
           val.append([0.2,0.3])
```

```
        if i=='take':
            val.append([0.3,0.15])
        if i=='it':
            val.append([0.2,0.15])
        if i=='easy':
            val.append([0.2,0.1])
    return len(input_words),val
#generate all possible transition strings given sentence length
def generate_possibilities(i,size,temp):
    if i==size:
        options.append(temp)
        return
    generate_possibilities(i+1, size, temp + ['S'])
    generate_possibilities(i+1, size, temp + ['E'])
#find max answer by checking all 2**n options for possible transition strings
def get_max_probability(n,options,val):
    for i in range(2**n):
        for j in range(n):
            if j==0:
                if options[i][j]=='E':
                    ans= 0.6*val[j][0]
                else:
                    ans=0.4*val[j][1]

            elif options[i][j]=='E' and options[i][j-1]=='E':
                ans*= 0.3*val[j][0]
            elif options[i][j]=='E' and options[i][j-1]=='S':
                ans*= 0.6*val[j][0]
            elif options[i][j]=='S' and options[i][j-1]=='S':
                ans*= 0.4*val[j][1]
            else:
                ans*=0.7*val[j][1]
        temp.append(ans)
    maxm = -float('inf')
    final_ans = []
    for i in range(len(temp)):
        if temp[i]>maxm:
            maxm = temp[i]
            final_ans = [options[i]]
        elif temp[i]==maxm:
            final_ans.append(options[i])
    for i in range(len(final_ans)):
        print('"'+" ".join(final_ans[i]) + '":'+str(round(maxm,5)))

#the main function outputting all answers
if __name__ =='__main__':
    n,val=make_input('cojelo con take it easy')
    options=[]
    temp=[]
    generate_possibilities(0,n,temp)
    get_max_probability(n,options,val)
    n,val=make_input('con take it easy')
    options=[]
    temp=[]
```

```
generate_possibilities(0,n,temp)
get_max_probability(n,options,val)
n,val=make_input('easy con')
options=[]
temp=[]
generate_possibilities(0,n,temp)
get_max_probability(n,options,val)
n,val=make_input('cojelo easy take it')
options=[]
temp=[]
generate_possibilities(0,n,temp)
get_max_probability(n,options,val)
```

Q2.2

Explanation:

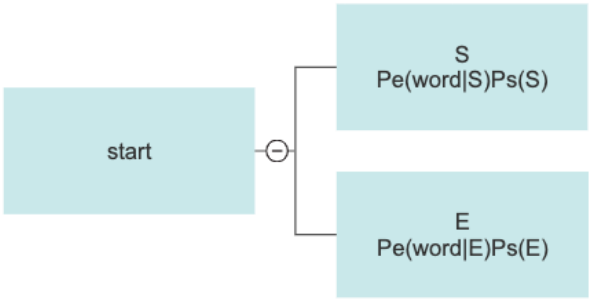
(Answer derived programmatically.)

- Transition probability(x|y)=Probability of the next word belonging to language ‘x’ given the language of the previous word is ‘y’

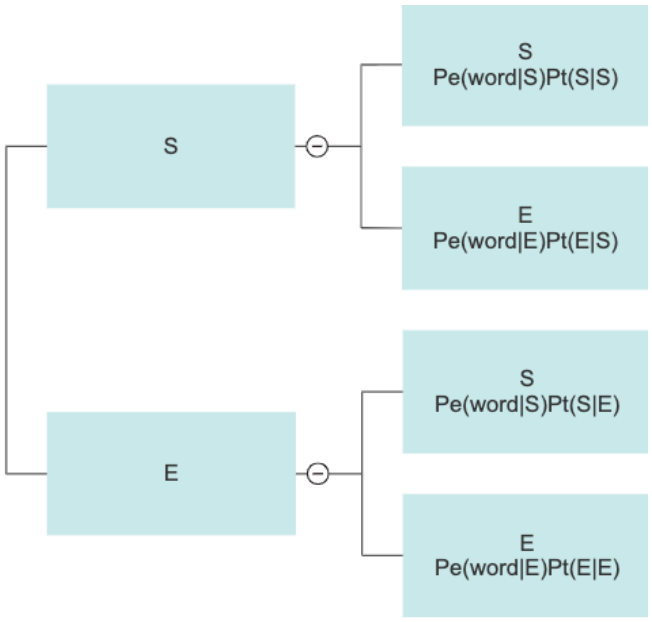
- Emission probability(string|x)=Probability of the word being ‘string’ given that the language expected is ‘x’

There are 2^n possible language transition strings
1st they have been computed recursively (and stored in the variable list ‘options’.)
Now for each possible transition string I have computed the probability as follows

For every word in the sentence, depending on its position we can make these 2 graphs for its probability contribution as follows
If position of the word is **1st in the sentence –> 2 cases**:



If position of the word is **not 1st in the sentence –> 4 cases based on previous letter of language transition string**:



Parent layer **current layer**

For each of the 2^n cases the probabilities of each of the words is multiplied, i.e. if the sentence is 'w1 w2 w3' and languages are l1,l2 Probability for the answer to be "l1 l2 l1" would be:
P=[Pe(w1|l1)*Ps(l1)]*[Pe(w2|l2)*Pt(l2|l1)]*[Pe(w3|l1)*Pt(l1|l2)]

The final answer would be the maximum probability among these 2^n cases.

As there are n words, the time complexity of the solution is $O(n.2^n)$

This has also been verified using a code which calculates the sum of probabilities for all transition strings, for all sentences, of a given input length. The sum of the probabilities comes out to be 1.

For example

	cojelo	**con**	**take**	**it**	**easy**
Pe(word|S)	0.3	0.3	0.15	0.15	0.1
Pe(word|E)	0.1	0.2	0.3	0.2	0.2

Sentence:

Example 01:
Cojelo con
There are 2^2 possibilities

1. SS
2. SE

3. ES
4. EE

SS:
probability=(P(cojelo|S)*Ps(S))*(P(con|S)*Pt(S|S))=(0.3*0.4)*(0.3*0.4)
=0.0144
SE
probability=(P(cojelo|S)*Ps(S))*(P(con|E)*Pt(E|S))=(0.3*0.4)*(0.2*0.6)
=0.0144
ES
probability=(P(cojelo|E)*Ps(E))*(P(con|S)*Pt(S|E))=(0.1*0.6)*(0.3*0.7)
=0.0126
EE
probability=(P(cojelo|E)*Ps(E))*(P(con|E)*Pt(E|E))=(0.1*0.6)*(0.2*0.3)
=0.0036

So there are 2 answers namely
S S:0.01440
S E:0.01440

Example 02:
Con take it easy
There are 2^4 possibilities

1.SSSS
2.SSSE
3.SSES
4.SSEE
5.SESS
6.SESE
7.SEES
8.SEEE
9.ESSS
10.ESSE
11.ESES
12.ESEE
13.EESS
14.EESE
15.EEES
16.EEEE

For example if we compute probability for SESE:
(P(con|S)*Ps(S))*(P(take|E)*Pt(E|S))*(P(it|S)*Pt(S|E))*(P(easy|E)*Pt(E
|S))

=(0.3*0.4)*(0.3*0.6)*(0.15*0.7)*(0.2*0.6)
=0.00027216
=0.00027
Similarly probabilities for other strings are computed. So we get:

SSSS:(0.3*0.4)*(0.15*0.4)*(0.15*0.4)*(0.1*0.4)=0.00002
SSSE:(0.3*0.4)*(0.15*0.4)*(0.15*0.4)*(0.2*0.6)=0.00005
SSES:(0.3*0.4)*(0.15*0.4)*(0.2*0.6)*(0.1*0.7)=0.00006
SSEE:(0.3*0.4)*(0.15*0.4)*(0.2*0.6)*(0.2*0.3)=0.00005
SESS:(0.3*0.4)*(0.3*0.6)*(0.15*0.7)*(0.1*0.4)=0.00009
SESE:(0.3*0.4)*(0.3*0.6)*(0.15*0.7)*(0.2*0.6)=0.00027
SEES:(0.3*0.4)*(0.3*0.6)*(0.2*0.3)*(0.1*0.7)=0.00009
SEEE:(0.3*0.4)*(0.3*0.6)*(0.2*0.3)*(0.2*0.3)=0.00008
ESSS:(0.2*0.6)*(0.15*0.7)*(0.15*0.4)*(0.1*0.4)=0.00003
ESSE:(0.2*0.6)*(0.15*0.7)*(0.15*0.4)*(0.2*0.6)=0.00009
ESES:(0.2*0.6)*(0.15*0.7)*(0.2*0.6)*(0.1*0.7)=0.00011
ESEE:(0.2*0.6)*(0.15*0.7)*(0.2*0.6)*(0.2*0.3)=0.00009
EESS:(0.2*0.6)*(0.3*0.3)*(0.15*0.7)*(0.1*0.4)=0.00005
EESE:(0.2*0.6)*(0.3*0.3)*(0.15*0.7)*(0.2*0.6)=0.00014
EEES:(0.2*0.6)*(0.3*0.3)*(0.2*0.3)*(0.1*0.7)=0.00005
EEEE:(0.2*0.6)*(0.3*0.3)*(0.2*0.3)*(0.2*0.3)=0.00004

Source code: 'MAHEK_VORA_CASE_STUDY_B_Q2_2_MAIN.py'

```
#taking sentence as input
def take_input():
   val=[]
   input_sentence=input()
   input_sentence=input_sentence.strip()
   input_words=input_sentence.split()
   for i in input_words:
       i=i.lower()
       if i=='cojelo':
           val.append([0.1,0.3])
       if i=='con':
           val.append([0.2,0.3])
       if i=='take':
           val.append([0.3,0.15])
       if i=='it':
           val.append([0.2,0.15])
       if i=='easy':
           val.append([0.2,0.1])
   return len(input_words),val
#generate all possible transition strings given length of word
def generate_possibilities(i,size,temp):
   if i==size:
       options.append(temp)
       return
   generate_possibilities(i+1, size, temp + ['S'])
```

```
    generate_possibilities(i+1, size, temp + ['E'])
#checking for each of the 2**n cases, compute probability
def solve(n,options,val):
    for i in range(2**n):
        for j in range(n):
            if j==0:
                if options[i][j]=='E':
                    ans= 0.6*val[j][0]
                else:
                    ans=0.4*val[j][1]

            elif options[i][j]=='E' and options[i][j-1]=='E':
                ans*= 0.3*val[j][0]
            elif options[i][j]=='E' and options[i][j-1]=='S':
                ans*= 0.6*val[j][0]
            elif options[i][j]=='S' and options[i][j-1]=='S':
                ans*= 0.4*val[j][1]
            else:
                ans*=0.7*val[j][1]
        temp.append(ans)
    maxm = -float('inf')
    final_ans = []
    for i in range(len(temp)):
        if temp[i]>maxm:
            maxm = temp[i]
            final_ans = [options[i]]
        elif temp[i]==maxm:
            final_ans.append(options[i])
    for i in range(len(final_ans)):
        print('"'+' '.join(final_ans[i]) + '":' +str(round(maxm,5)))

#main function which shows required outputs
if __name__ =='__main__':
    n,val=take_input()
    options=[]
    temp=[]
    generate_possibilities(0,n,temp)
    solve(n,options,val)
```

Source Code for verification:'MAHEK_VORA_CASE_STUDY_B_Q2_VERIFICATION_MAIN'

```
input_sentence=[]
#make all possible sentences of size n
def generate_input_sentences(i,size,temp):
    if i==size:
        input_sentence.append(temp)
        return
    generate_input_sentences(i+1, size, temp + 'cojelo ')
    generate_input_sentences(i+1, size, temp + 'con ')
    generate_input_sentences(i+1, size, temp + 'take ')
```

```
    generate_input_sentences(i+1, size, temp + 'it ')
    generate_input_sentences(i+1, size, temp + 'easy ')
#store their emission probabilities
def make_input(size):
    val=[]
    input_words=[]
    generate_input_sentences(0,size,'')
    for i in input_sentence:
         i=i.strip()
         l=(i.split())
         input_words.append(l)

    res=[]
    #print(input_words)
    for j in range(len(input_sentence)):
        for i in input_words[j]:
            i=i.lower()
            if i=='cojelo':
                val.append([0.1,0.3])
            if i=='con':
                val.append([0.2,0.3])
            if i=='take':
                val.append([0.3,0.15])
            if i=='it':
                val.append([0.2,0.15])
            if i=='easy':
                val.append([0.2,0.1])
       #print(val)
        res.append(val)
        val=[]
    return len(input_words[0]),res
#all possible transition strings given a strings length
def generate_possibilities(i,size,temp):
    if i==size:
        options.append(temp)
        return
    generate_possibilities(i+1, size, temp + ['S'])
    generate_possibilities(i+1, size, temp + ['E'])

def solve(n,options,res,x):
    temp=[]
    for i in range(2**n):
        #print(options[i])
        for j in range(n):
            if j==0:
                if options[i][j]=='E':
                    ans= 0.6*res[x][j][0]
                else:
                    ans=0.4*res[x][j][1]

            elif options[i][j]=='E' and options[i][j-1]=='E':
                ans*= 0.3*res[x][j][0]
            elif options[i][j]=='E' and options[i][j-1]=='S':
                ans*= 0.6*res[x][j][0]
```

```
            elif options[i][j]=='S' and options[i][j-1]=='S':
                ans*= 0.4*res[x][j][1]
            else:
                ans*=0.7*res[x][j][1]
            #print(ans)
        temp.append(ans)
    a=0
    for i in range(len(temp)):
        a=a+temp[i]
        #print(temp[i])
    return a

#main function, shows output
if __name__ =='__main__':
    n,res=make_input(2)
    options=[]
    temp=[]
    generate_possibilities(0,n,temp)
    ans=0
    for i in range(len(input_sentence)):
        ans=ans+solve(n,options,res,i)
    print('The sum of the probabilities for all possible strings and their
possible transition strings of a fixed length is :'+str(ans))
```