```python
"""
MATH942
Ruben Traicevski 6790021
"""

import numpy as np
import matplotlib.pyplot as plt
"""
# Bisection Method Alogrithm
"""

#Defining our suitable function which has an unknown root
def f(x):
    return
# For the below bisection method to work, we emphasis the word suitable


#Define our bisection method which will take
# f, our function
# [a,b], an interval
# acc, some pre specified level of tolerance on the error with the true root
def bisect(f, a, b, acc):

    #First check if the given interval contains a root
    if f(a)*f(b) > 0:
      #By intermediate value theorem, we need that f(a)f(b) < 0 at all times
      #This checks if the signs of f(a) or f(b) are the same

        raise Exception("No root within the interval")
        # Will throw an exception if there is no root in the interval given to
        # the bisection function

    m = a + (a-b)/2 #Obtain midpoint of the interval
    # we will make our bisection function
    # call itself automatically and give itself the appropriate interval
    # However, this will not give us a way to look at the preceeding intervals


    # This will stop the function when have found suitable approximation of the root
    # in accordance with our error
    if np.abs(f(m)) < acc:
        return m


    # Use elif keyword, as python will try these conditons if previous condition was not met
    elif np.sign(f(a)) == np.sign(f(m)):
        #Checking if root is between lower bound a and midpoint m
        # if IVT doesnt hold, then bisec func calls itself with new interval between m and b
        return bisect(f, m, b, acc)

    elif np.sign(f(b)) == np.sign(f(m)):
        #Checking if root is between midpoint m and uppwer bound b
        # if IVT doesnt hold, then bisec func calls itself with new interval between a and m
        return bisect(f, a, m, acc)

"""
# Newton Method Alogrithm
"""
# Define suitable function with unknown root/s
def f(x):
    return
```

```python
#Define the derivative function of f(x)
def df(x):
    return

# Define newton method function
def newt(f, df, Xo, acc):

    #Check if root is within tolerance, and print the value of f(x) at Xo
    if abs(f(Xo)) < acc:
        return Xo, print("Value of f(x) at the approx root ="),Xo

    #else condition triggers automatically if condition above is not met
    else:
        return newt(f, df, (Xo - f(Xo)/df(Xo)), acc)
    # This will enable the newt function to call itself automatically without needing
    # an external loop.
"""
# Current Algorithm does not exit after n iterations, thus it is advisable
# to ensure conditions for newton's method are met so it does not run indefinitely
#It is also needed that the first initial guess be within a suitable range from the actual root
"""




"""
 Question 3 / 4
 For an artificially designed test problem, we can arbitrarily select the inputs for
 a derivative to obtain the derivative price, then give the derivative formula to our
  numerical algorithms with IV missing for it to then approximate the suitable IV to obtain
  the same price.

  We shall make the following assumptions below which also apply to Question 4
  1. Will use standard Black Scholes Model for European Options
  2. Underlying instrument is non dividend paying
  3. Underlying instrument will be a stock

  The following variables will be defined as such
  S = Stock Price
  K = Strike Price
  T = Date of Maturity (As of now, the units on time are not important, only the magnitude)
  r = Interest Rate
  Sigma = Implied Volatility

  Also note that for this controlled test, we are assuming t = 0,

 """

 import numpy as np
 import matplotlib.pyplot as plt
 from scipy.stats import norm

 # We will then define the needed formula's

 N = norm.cdf

 def BS_CALL(S, K, T, r, sigma):
     d1 = (np.log(S/K) + (r + sigma**2/2)*T) / (sigma*np.sqrt(T))
     d2 = d1 - sigma * np.sqrt(T)
     return S * N(d1) - K * np.exp(-r*T)* N(d2)


 # We shall arbitrarily give the inputs
 S = 140
 K = 150
 T = 3
 r = 0.05
 IV_known = 0.3
 sigma = IV_known

 Call_Price = BS_CALL(S, K, T, r, sigma)
 """
 This will give a call price of 33.35....

 Then to demonstrate the ability of our numerical approximation algorithms, we will set sigma
 to be unknown. In short, we are saying we want to find the unknown root (IV) that will give the
 current call option price
 """

 """
 BISECTION METHOD
 The bisection method can only be used for finding roots, that is f(r)=0 and requires
 the Intermediate Value Theorem to be True. Thus, to make the bisection method work, we will
 shift BS_CALL(IV, .) by - Price to make it zero when the right IV is found

 To do this, we will chose an interval containing the actual IV.
 """
 a = 0.01 #Lower guess for IV
 b = 0.9 # Upper Guess for IV
 acc = 0.001 # accuracy / tolerance for between actual IV

 def bisect(BS_CALL, a, b, acc):


     if (BS_CALL(S, K, T, r, a)-Call_Price)*(BS_CALL(S, K, T, r, b)-Call_Price) > 0:


         raise Exception("No root within the interval")

     m = (a+b)/2


     if np.abs(BS_CALL(S, K, T, r, m) -Call_Price  ) < acc:
         return m ,print("Value of BS_CALL at the approx root using Bisection Method =", m)


     elif np.sign(BS_CALL(S, K, T, r, a) -Call_Price) == np.sign(BS_CALL(S, K, T, r, m)-Call_Pri

         return bisect(BS_CALL, m, b, acc)

     elif np.sign(BS_CALL(S, K, T, r, b)-Call_Price) == np.sign(BS_CALL(S, K, T, r, m)-Call_Pric

         return bisect(BS_CALL, a, m, acc)

 approx_IV = bisect(BS_CALL, a, b, acc)
 print(approx_IV)
 """
 Our approx_IV gives a value of 0.3000.... which is the value exact value of the IV given
 before
 """
```

```
 ##############################################################################################
 """
 Newton's Method

 For Newtons method to work, we will require the partial derivative of the BS_CALL with
 respect to volatility which is also known as vega.
 The Vega of a call option is also the same as
 a put option, by put-call parity
 """

def Vega_BS_Call(S, K, T, r, sigma):

    d1 = (np.log(S/K) + (r + sigma**2/2)*T) / (sigma*np.sqrt(T))
    return S*np.sqrt(T) * norm.pdf(d1)


 Xo = 0.4 #Inital guess for IV
 tol = 0.001


 def newt(BS_CALL, Vega_BS_Call, Xo, tol):

     #Check if root is within tolerance, and print the value of f(x) at Xo
     if abs(BS_CALL(S, K, T, r, Xo) - 16.4519157) < tol:
         return Xo, print("Value of BS_CALL at the approx root using Newton Method =", Xo)

     #else condition triggers automatically if condition above is not met
     else:
         return newt(BS_CALL, Vega_BS_Call , (Xo - (BS_CALL(S, K, T, r, Xo)-16.4519157)/Vega_BS_
     # This will enable the newt function to call itself automatically without needing
     # an external loop.

 approx_iv_newt = newt(BS_CALL, Vega_BS_Call, Xo, tol)
 print(approx_iv_newt)

 """
 We obtained the exact value for the IV of 0.30000
 """

"""
We can now use the Algorithms using real market price data

However, we are likely to see discrepencies based on the following

1. Real world market data is biased
2. Not enough trading volume to generate accurate prices
3. Underlying stock may pay dividends over the life of the contract
4. The BSM call option formula used in this Algorithm may be too simplistic
   compared to the BS models used by market makers
5. Incorrect interest rate being used, nor are interest rates constant in the real world.
6. Market makers also account for Volatiliy skew in their models,
   where as the one used below does not
7. Security Prices do not exactly follow a stationary log-normal random process
   leading to price discrepancies.
   For example,it has been long noted
   that the BS model will tend to underprice deep OTM options,
   and overprice deep ITM options.

"""
```

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm



# Call Option Values
r = 0.035
#Looking at the RBA's cash rate history, we the cash rate has stayed below 7.5% since
# 1992, with it staying around 5%.

Cp_Euro = 0.330 #Quoted Call Option Price on Qantas
S = 6.52 # Adjusted Close Price of Qantas
K = 6.5 #Strike Price of Contract
T = 3 #Start 19th may 2023, Expire 17th Aug 2023 ~ 3 Months to expiry


N = norm.cdf

def BS_CALL(S, K, T, r, sigma):
    d1 = (np.log(S/K) + (r + sigma**2/2)*T) / (sigma*np.sqrt(T))
    d2 = d1 - (sigma * np.sqrt(T))
    return S * N(d2) - K * np.exp(-r*T)* N(d2)

def Vega_BS_Call(S, K, T, r, sigma):

    d1 = (np.log(S/K) + (r + sigma**2/2)*T) / (sigma*np.sqrt(T))
    return S*np.sqrt(T) * norm.pdf(d1)

a = 0.01 #Lower guess for IV
b = 5 # Upper Guess for IV
acc = 0.01 # accuracy / tolerance for between actual IV

def bisect(BS_CALL, a, b, acc):


    if (BS_CALL(S, K, T, r, a)-Cp_Euro)*(BS_CALL(S, K, T, r, b)-Cp_Euro) > 0:


        raise Exception("No root within the interval")

    m = (a+b)/2


    if np.abs(BS_CALL(S, K, T, r, m) - Cp_Euro  ) < acc:
        return m ,print("IV of BS_CALL at the approx root using Bisection Method =", m)


    elif np.sign(BS_CALL(S, K, T, r, a) -Cp_Euro) == np.sign(BS_CALL(S, K, T, r, m)- Cp_Euro):

        return bisect(BS_CALL, m, b, acc)

    elif np.sign(BS_CALL(S, K, T, r, b)- Cp_Euro) == np.sign(BS_CALL(S, K, T, r, m)- Cp_Euro):

        return bisect(BS_CALL, a, m, acc)

approx_IV = bisect(BS_CALL, a, b, acc)
print(approx_IV)
"""
This gave an approximate value of 0.282.....
"""
```

```
"""
Newtons Method

We have adjusted the previous Newton's method to account for
how Mmany iterations we would like to do to stop
an infinite loop occuring for bad initial guess

"""

def Vega_BS_Call(S, K, T, r, sigma):

    d1 = (np.log(S/K) + (r + (sigma**2)/2)*T) / (sigma*np.sqrt(T))
    return S*np.sqrt(T) * norm.pdf(d1)


Xo = 1.9 #Inital guess for IV
tol = 0.01

MAX_ITERATIONS = 10000
def newt(BS_CALL, Vega_BS_Call, Xo, tol):
    #For loop added to stop infinite loop

    for i in range(MAX_ITERATIONS):
        diff = (BS_CALL(S, K, T, r, Xo)-Cp_Euro)

    #Check if root is within tolerance, and print the value of f(x) at Xo
        if abs(diff) < tol:

            return Xo, print("IV of BS_CALL at the approx root using Newton Method =", Xo)

        Xo +=diff/Vega_BS_Call(S, K, T, r, Xo)



approx_iv_newt = newt(BS_CALL, Vega_BS_Call, Xo, tol)
print(approx_iv_newt)
"""
This gave an approximate value of 0.298....

Then looking at the IV quoted by the vendor, it is ~ 0.206
which will give a call price of 0.3209... indicating our
model has slightly underpriced the call option

"""
```