In [140]:
```
import numpy as np
import math
from math import factorial
import operator as op
from functools import reduce
import matplotlib.pyplot as plt
import itertools
import decimal
```

Question 1)

a)

In [24]:
```
def ncr(n, r):
    return factorial(n)/((factorial(n-r))*(factorial(r)))
```

In [25]:
```
n_list = [10, 20, 40, 80, 100, 200, 500]
```

In [95]:
```
s0, k, r, sigma, T = 32, 30, 0.05, 0.24, 0.5
def binomial_price_a(s0, k, r, sigma, T, n):
    delta = T/n
    c = 0.5*(np.exp(-r*delta) + np.exp((r + sigma**2)*delta))
    d = c - np.sqrt(c**2 - 1)
    u = 1/d
    p = (np.exp(r * delta) - d)/(u - d)
    sum_price = 0
    for i in range(n):
        sum_price += (ncr(n, i) * p**i * (1-p)**(n-i) * max(0, s0*u**i*d**(
    return np.exp(-r*n*delta)*sum_price
```

In [57]:
```
a_prices = [binomial_price_a(s0, k, r, sigma, T, i) for i in n_list]
```

b)

In [43]:
```
s0, k, r, sigma, T = 32, 30, 0.05, 0.24, 0.5
def binomial_price_b(s0, k, r, sigma, T, n):
    delta = T/n
    u = np.exp(r*delta)*(1 + np.sqrt(np.exp(sigma**2*delta) - 1))
    d = np.exp(r*delta)*(1 - np.sqrt(np.exp(sigma**2*delta) - 1))
    p = 1/2
    sum_price = 0
    for i in range(n):
        sum_price += (ncr(n, i) * p**i * (1-p)**(n-i) * max(0, s0*u**i*d**(
    return np.exp(-r*n*delta)*sum_price
```

In [56]:
```
b_prices = [binomial_price_b(s0, k, r, sigma, T, i) for i in n_list]
```

c)

In [45]:
```
s0, k, r, sigma, T = 32, 30, 0.05, 0.24, 0.5
def binomial_price_c(s0, k, r, sigma, T, n):
    delta = T/n
    u = np.exp((r - (sigma**2)/2)*delta + (sigma*np.sqrt(delta)))
    d = np.exp((r - (sigma**2)/2)*delta - (sigma*np.sqrt(delta)))
    p = 1/2
    sum_price = 0
    for i in range(n):
        sum_price += (ncr(n, i) * p**i * (1-p)**(n-i) * max(0, s0*u**i*d**(
    return np.exp(-r*n*delta)*sum_price
```

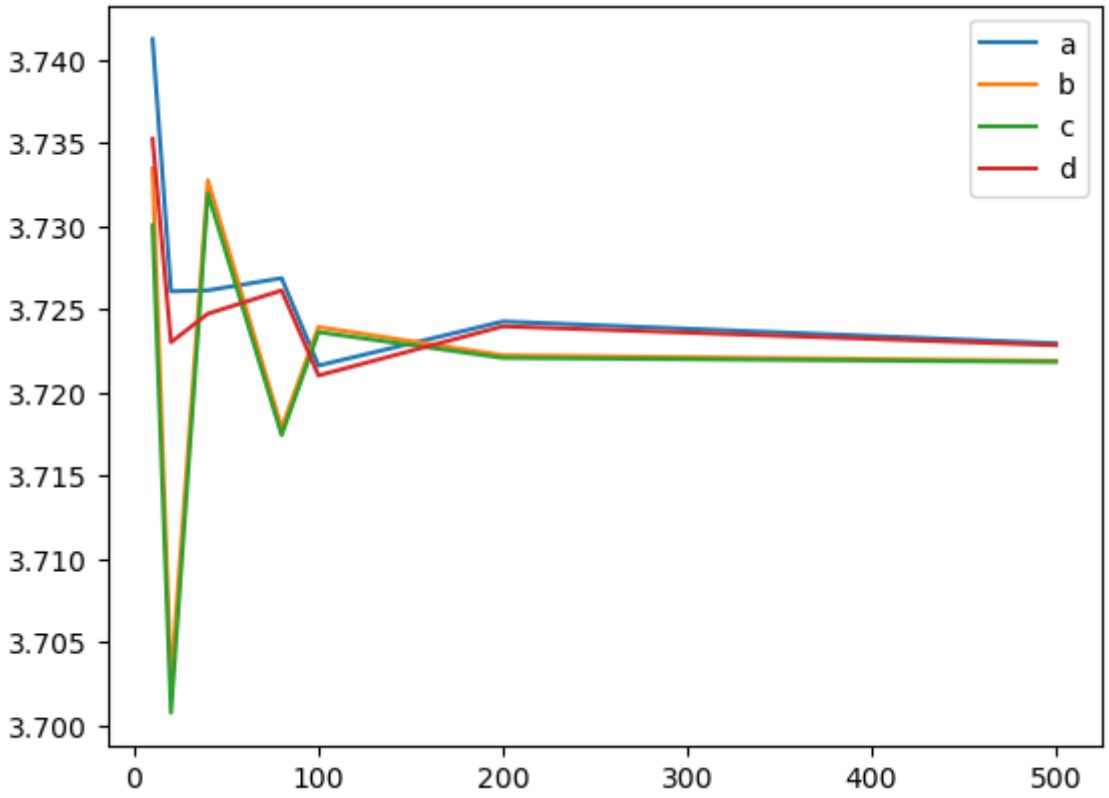
In [55]:
```
c_prices = [binomial_price_c(s0, k, r, sigma, T, i) for i in n_list]
```

d)

In [47]:
```
s0, k, r, sigma, T = 32, 30, 0.05, 0.24, 0.5
def binomial_price_d(s0, k, r, sigma, T, n):
    delta = T/n
    u = np.exp(sigma*np.sqrt(delta))
    d = np.exp(-sigma*np.sqrt(delta))
    p = 0.5 + 0.5*(((r - (sigma**2)/2)*np.sqrt(delta))/sigma)
    sum_price = 0
    for i in range(n):
        sum_price += (ncr(n, i) * p**i * (1-p)**(n-i) * max(0, s0*u**i*d**(
    return np.exp(-r*n*delta)*sum_price
```

In [54]:
```
d_prices = [binomial_price_d(s0, k, r, sigma, T, i) for i in n_list]
```

In [80]:
```
plt.plot(n_list, a_prices, label='a')
plt.plot(n_list, b_prices, label='b')
plt.plot(n_list, c_prices, label='c')
plt.plot(n_list, d_prices, label='d')
plt.legend()
plt.show()
```

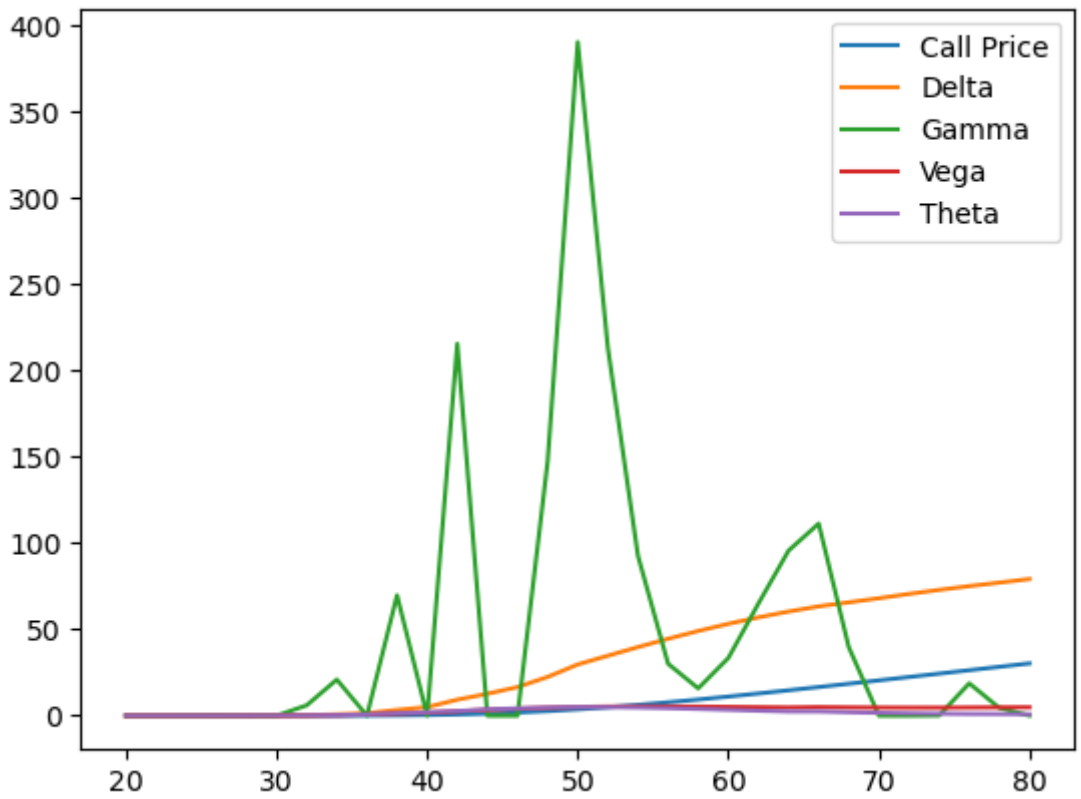


Question 2)

In [246]:
```
def american_option(s0, K, r, sigma, T, mu, N):
    delta = T/N
    u = np.exp(sigma*np.sqrt(delta))
    d = np.exp(-sigma*np.sqrt(delta))
    vector = np.zeros(N+1)
    S_T = s0 * u**np.arange(0, N+1, 1) * d**np.arange(N, -1, -1)

    p = (np.exp(r * delta) - d)/ (u - d)
    q = 1.0 - p

    vector[:] = np.maximum(S_T-K, 0)

    for i in range(N-1, -1, -1):
        vector[:-1] = np.exp(-r*delta) * (q * vector[1:] + p * vector[:-1])
        S_T = S_T * u
        vector = np.maximum(vector, S_T-K )
    return vector[0]
```

In [247]:
```
def delta(S0, K, r, sigma, T, mu, N):
        return (american_option(S0*1.01, K, r, sigma, T, mu, N) - american_

def gamma(S0, K, r, sigma, T, mu, N):
    return (american_option(S0*1.01, K, r, sigma, T, mu, N) - (2*american_o

def vega(S0, K, r, sigma, T, mu, N):
    return (american_option(S0, K, r, sigma*1.01, T, mu, N) - american_opti

def theta(S0, K, r, sigma, T, mu, N):
    return (american_option(S0, K, r, sigma, T+0.004, mu, N) - american_opt
```

In [248]:
```
K, r, sigma, T, mu, N = 50, 0.05, 0.28, 0.3846, 0.14, 100
stock_increments = np.arange(20, 82, 2)
call_price = [american_option(i, K, r, sigma, T, mu, N) for i in stock_incr
deltas = [delta(stock_increments[i], K, r, sigma, T, mu, N) for i in range(
gammas = [gamma(stock_increments[i], K, r, sigma, T, mu, N) for i in range(
vegas = [vega(stock_increments[i], K, r, sigma, T, mu, N) for i in range(le
thetas = [theta(stock_increments[i], K, r, sigma, T, mu, N) for i in range(
plt.plot(stock_increments, call_price, label='Call Price')
plt.plot(stock_increments, deltas, label='Delta')
plt.plot(stock_increments, gammas, label='Gamma')
plt.plot(stock_increments, vegas, label='Vega')
plt.plot(stock_increments, thetas, label='Theta')
plt.legend()
plt.show()
```



Question 3

a)

In [156]:
```
def multinomial(n, u, m , d):
    return factorial(n)/((factorial(u))*(factorial(m))*(factorial(d)))
```

In [161]:
```
def trinomial_price_a(s0, k, r, sigma, T, n):
    delta = T/n
    d = np.exp(-sigma*np.sqrt(3*delta))
    u = 1/d
    m = 1
    p_d = (r*delta*(1-u) + (r*delta)**2 + (sigma**2 * delta))/((u-d)*(1-d))
    p_u = (r*delta*(1-d) + (r*delta)**2 + (sigma**2 * delta))/((u-d)*(u-1))
    p_m = 1 - p_u - p_d
    sum_price = 0
    for i in range(n):
        for j in range(n-i):
            sum_price += multinomial(n, i, j, n-i-j) * p_u**i * p_m**j * p_
    return (np.exp(-r*T))*sum_price
```

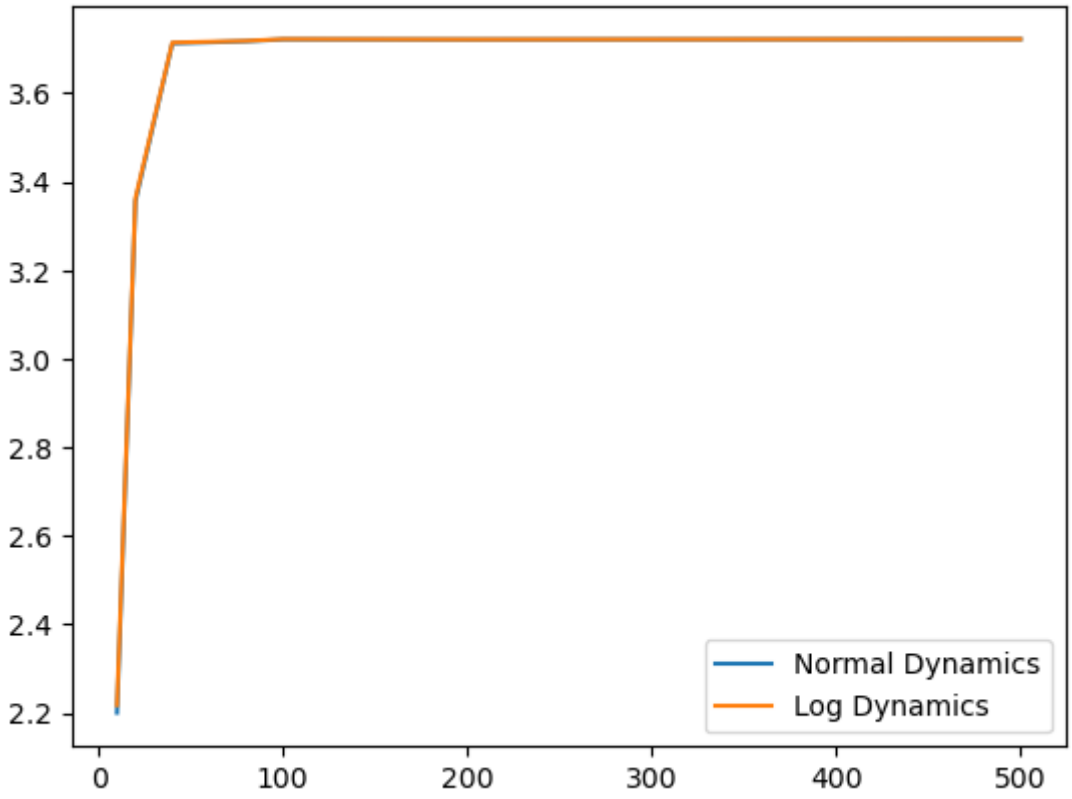
In [175]:
```
s0, K, r, sigma, T = 32, 30, 0.05, 0.24, 0.5
n_list = [10, 20, 40, 80, 100, 200, 500]
trinomial_prices_a = [trinomial_price_a(s0, K, r, sigma, T, i) for i in n_l
```

b)

In [213]:
```
def trinomial_price_b(s0, k, r, sigma, T, n):
    sd = sigma
    dt = T/n

    dXu = sd*np.sqrt(3*dt)
    dXd = -sd*np.sqrt(3*dt)
    dXm = 0

    pu = (1/2)*(((r - 0.5*sd**2)**2 *(dt)**2 + sd**2 *dt)/(dXu)**2 + ((r -
    pd = (1/2)*(((r - 0.5*sd**2)**2 *(dt)**2 + sd**2 *dt)/(dXu)**2 - ((r -
    pm = 1 - pu - pd

    sum_price = 0
    for i in range(n):
        for j in range(n-i):
            payoff = max(0, np.exp(np.log(s0) + (i*dXu) + (j*dXm) + ((n-i-j
            sum_price += multinomial(n, i, j, n-i-j) * pu**i * pm**j * pd**
    return (np.exp(-r*T))*sum_price
```

In [214]:
```
s0, K, r, sigma, T = 32, 30, 0.05, 0.24, 0.5
n_list = [10, 20, 40, 80, 100, 200, 500]
trinomial_prices_b = [trinomial_price_b(s0, K, r, sigma, T, i) for i in n_l
```

In [217]:
```
plt.plot(n_list, trinomial_prices_a, label='Normal Dynamics')
plt.plot(n_list, trinomial_prices_b, label='Log Dynamics')
plt.legend()
plt.show()
```



Question 4

In [218]:
```
def getHalton(HowMany, Base):
    Seq = np.zeros(HowMany) # Column vector
    NumBits = 1 + math.ceil(np.log(HowMany)/np.log(Base))
    VetBase = 1/(Base**((np.arange(1,NumBits+1))))
    WorkVet = np.zeros(NumBits) # row vector
    for i in range(1, HowMany+1):
        j = 1
        ok = 0
        while ok == 0:
            WorkVet[j] = WorkVet[j] + 1
            if WorkVet[j] < Base:
                ok = 1
            else:
                WorkVet[j] = 0
                j += 1
        Seq[i-1] = np.dot(WorkVet, VetBase)
    return Seq
def box_muller(u1, u2):
    z1 = np.sqrt(-2 * np.log(u1)) * math.cos(2 * np.pi * u2)
    z2 = np.sqrt(-2 * np.log(u1)) * math.sin(2 * np.pi * u2)
    return [z1, z2]
```

In [234]:
```
halton_1 = getHalton(500, 2)
halton_2 = getHalton(500, 7)
stand_norm = [box_muller(halton_1[i], halton_2[i]) for i in range(len(halto
stand_norm = np.concatenate(stand_norm)
```

In [235]:
```
def euro_call(S0, K, r, sigma, T, stand_norm):
      return np.mean([max(0, S0*np.exp((r-(sigma**2)/2)*T + (sigma*np.sqrt(
```

In [240]:
```
print("Price of European call option:", "$", euro_call(100, 100, 0.05, 0.24
```

```
Price of European call option: $ 23.449094604417134
```

In [ ]: