The prices of the European put option

Find the prices of European option using (CRR - Jarrow-Rudd - Leisen-Reimer) methods:

First Method is: CRR Parameters

CRR model is: a simple but very important model for the price of a single risky security.

At N = 300 with CRR Method for European option

In [102]:
```
import pandas as pd          ## import from module scipy all functions
import numpy as np          ## import from module numpy all functions
def Eur_putprice(S0,E,T,N,r,sigma,cp):      ##  Defining the function Eur_putprice to calculate
        ## the price of a European put option
    dt=T/N
    U=np.exp(sigma*np.sqrt(dt))
    D=np.exp(-sigma*np.sqrt(dt))
    R=(np.exp(r*dt))
    q=(R-D)/(U-D)
    ## defining delta_t, parameters U, D, R and risk-neutrall probability q
    ## Here we use CRR parameters for U and D.
    V_old=np.zeros(N+1)
    V_new=np.zeros(N+1)
    ## initializing two N+1 dimensional vectors with zeros to store option prices
    for m in range(0,N+1):   ## cycle - m gets the values 0,1, ...,N
        V_old[m]=max(S0*U**m*D**(N-m)-E,0)
        ## first cycle - Computing the prices of the option (depending on the price S(T)=S0*U**D**(N-m)
        ##at time T=N*dt (see formula V(N,m) for put option). The prices of the option at time T
        ## equal to the payments at time T.
    for j in range(N-1,-1,-1):
        ## cycle for formula V(j,m), j=N-1, N-2,...,0
        for m in range(0,j+1):
            V_new[m]=(q*V_old[m+1]+(1-q)*V_old[m])/R
            ## computing the option prices at time j*dt if we know the prices at time (j+1)*dt.
            ## (see formula for V(j,m))
        for m in range(0,j+1):
            V_old[m]=V_new[m]
            ## after that the all option prices at time j*dt are known, we consider these prices as old
 prices.
    return(V_new[0])
    ## V_new[0] is the option price at t=0

print('The European put option price at N = 300 is %8.4f' % Eur_putprice(S0=100,E=98.5,T=2,N=300,r=0.03
,sigma=np.sqrt(0.15),cp=-1))
## finding and printing European put option with parameters
## S0=100,E=98.50,T=2,N=1000,r=0.03,sigma=np.sqrt(0.15),cp=-1.
```

```
The European put option price at N = 300 is  24.6155
```

At N = 400 with CRR Method for European option

In [103]:
```
import pandas as pd          ## import from module scipy all functions
import numpy as np          ## import from module numpy all functions
def Eur_putprice(S0,E,T,N,r,sigma,cp):      ##  Defining the function Eur_putprice to calculate
        ## the price of a European put option
    dt=T/N
    U=np.exp(sigma*np.sqrt(dt))
    D=np.exp(-sigma*np.sqrt(dt))
    R=(np.exp(r*dt))
    q=(R-D)/(U-D)
    ## defining delta_t, parameters U, D, R and risk-neutrall probability q
    ## Here we use CRR parameters for U and D.
    V_old=np.zeros(N+1)
    V_new=np.zeros(N+1)
    ## initializing two N+1 dimensional vectors with zeros to store option prices
    for m in range(0,N+1):   ## cycle - m gets the values 0,1, ...,N
        V_old[m]=max(S0*U**m*D**(N-m)-E,0)
        ## first cycle - Computing the prices of the option (depending on the price S(T)=S0*U**D**(N-m)
        ##at time T=N*dt (see formula V(N,m) for put option). The prices of the option at time T
        ## equal to the payments at time T.
    for j in range(N-1,-1,-1):
        ## cycle for formula V(j,m), j=N-1, N-2,...,0
        for m in range(0,j+1):
            V_new[m]=(q*V_old[m+1]+(1-q)*V_old[m])/R
            ## computing the option prices at time j*dt if we know the prices at time (j+1)*dt.
            ## (see formula for V(j,m))
        for m in range(0,j+1):
            V_old[m]=V_new[m]
            ## after that the all option prices at time j*dt are known, we consider these prices as old
 prices.
    return(V_new[0])
    ## V_new[0] is the option price at t=0

print('The European put option price is %8.4f' % Eur_putprice(S0=100,E=98.5,T=2,N=400,r=0.03,sigma=np.s
qrt(0.15),cp=-1))
## finding and printing European put option with parameters
## S0=100,E=98.50,T=2,N=1000,r=0.03,sigma=np.sqrt(0.15),cp=-1.
```

```
The European put option price is  24.6155
```

At N = 500 with CRR Method for European option

In [104]:
```
import pandas as pd          ## import from module scipy all functions
import numpy as np          ## import from module numpy all functions
def Eur_putprice(S0,E,T,N,r,sigma,cp):      ##  Defining the function Eur_putprice to calculate
        ## the price of a European put option
    dt=T/N
    U=np.exp(sigma*np.sqrt(dt))
    D=np.exp(-sigma*np.sqrt(dt))
    R=(np.exp(r*dt))
    q=(R-D)/(U-D)
    ## defining delta_t, parameters U, D, R and risk-neutrall probability q
    ## Here we use CRR parameters for U and D.
    V_old=np.zeros(N+1)
    V_new=np.zeros(N+1)
    ## initializing two N+1 dimensional vectors with zeros to store option prices
    for m in range(0,N+1):   ## cycle - m gets the values 0,1, ...,N
        V_old[m]=max(S0*U**m*D**(N-m)-E,0)
        ## first cycle - Computing the prices of the option (depending on the price S(T)=S0*U**D**(N-m)
        ##at time T=N*dt (see formula V(N,m) for put option). The prices of the option at time T
        ## equal to the payments at time T.
    for j in range(N-1,-1,-1):
        ## cycle for formula V(j,m), j=N-1, N-2,...,0
        for m in range(0,j+1):
            V_new[m]=(q*V_old[m+1]+(1-q)*V_old[m])/R
            ## computing the option prices at time j*dt if we know the prices at time (j+1)*dt.
            ## (see formula for V(j,m))
        for m in range(0,j+1):
            V_old[m]=V_new[m]
            ## after that the all option prices at time j*dt are known, we consider these prices as old
 prices.
    return(V_new[0])
    ## V_new[0] is the option price at t=0

print('The European put option price is %8.4f' % Eur_putprice(S0=100,E=98.5,T=2,N=500,r=0.03,sigma=np.s
qrt(0.15),cp=-1))
## finding and printing European put option with parameters
## S0=100,E=98.50,T=2,N=1000,r=0.03,sigma=np.sqrt(0.15),cp=-1.
```

```
The European put option price is  24.6152
```

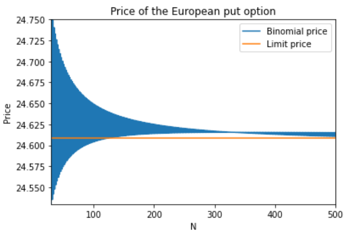
Find the price of the European option using limit price for BS formula:

In [105]:
```
from scipy.stats import norm
import math

S0=100
E=98.5
T=2
r=0.03
sigma=np.sqrt(.15)
d1=(math.log(S0/E)+(r+0.5*sigma**2)*T)/(sigma*np.sqrt(T))
d2=d1-sigma*np.sqrt(T)
Vput=S0*norm.cdf(d1)-E*np.exp(-r*T)*norm.cdf(d2)
## function norm.cdf computes the value of the cumulative distribution function of the standardized nor
mal distribution N(0,1),
print('The Limit Price is %8.4f' %Vput)
```

```
The Limit Price is  24.6085
```

Plot the graph of the European option and the limit price in one graph:

In [106]:
```
import matplotlib.pyplot as plt

# Assign a short name to a module to use it when referring to a function in the module
## for example, instead matplotlib.pyplot.xlim we use plt.xlim.
V=np.zeros(501)
Vc=Vput*np.ones(501)
##initializing vector V with zeros to store option prices
for k in range(30,501):
    V[k]=Eur_putprice(S0=100,E=98.5,T=2,N=k,r=0.03,sigma=np.sqrt(0.15),cp=-1)
    ## computing the prices
n = np.linspace(0, 500, 501)
#defining vector of x-values (number of the period)
plt.plot(n, V)
plt.plot(n, Vc)
## definining the plot, the x-values are in vector n, y-values are in vector V.
plt.xlim((30, 500))
## limits of x-axis
plt.ylim((24.53,24.75))
## limits of y-axis (limits of option prices)
plt.ylabel('Price')
plt.xlabel('N')
plt.legend(['Binomial price','Limit price'], loc='best')
plt.title('Price of the European put option')
plt.show()
## show plot
```



Second Method is: Jarrow-Rudd method

At N = 300 with Jarrow-Rudd Method for European option

In [107]:

```
import pandas as pd           ## import from module scipy all functions
import numpy as np           ## import from module numpy all functions
def Eur_putprice_Jarrow(S0,E,T,N,r,sigma,cp):      ##  Defining the function Eur_putprice to calculate
        ## the price of a European put option
    dt=T/N
    p=r-(0.5*sigma**2)
    U=np.exp(p*dt+sigma*np.sqrt(dt))
    D=np.exp(p*dt-sigma*np.sqrt(dt))
    R=(np.exp(r*dt))
    q=(R-D)/(U-D)
    ## defining delta_t, parameters U, D, R and risk-neutrall probability q
    ## Here we use Jarrow-Rudd parameters for U and D.
    V_old=np.zeros(N+1)
    V_new=np.zeros(N+1)
    ## initializing two N+1 dimensional vectors with zeros to store option prices
    for m in range(0,N+1):   ## cycle - m gets the values 0,1, ...,N
        V_old[m]=max(S0*U**m*D**(N-m)-E,0)
        ## first cycle - Computing the prices of the option (depending on the price S(T)=S0*U**D**(N-m)
        ##at time T=N*dt (see formula V(N,m) for put option). The prices of the option at time T
        ## equal to the payments at time T.
    for j in range(N-1,-1,-1):
        ## cycle for formula V(j,m), j=N-1, N-2,...,0
        for m in range(0,j+1):
            V_new[m]=(q*V_old[m+1]+(1-q)*V_old[m])/R
            ## computing the option prices at time j*dt if we know the prices at time (j+1)*dt.
            ## (see formula for V(j,m))
        for m in range(0,j+1):
            V_old[m]=V_new[m]
            ## after that the all option prices at time j*dt are known, we consider these prices as old
 prices.
    return(V_new[0])
    ## V_new[0] is the option price at t=0

print('The European put option price using Jarrow-Rudd Parameters is %8.4f' % Eur_putprice_Jarrow(S0=10
0,E=98.5,T=2,N=300,r=0.03,sigma=np.sqrt(0.15),cp=-1))
## finding and printing European put option with parameters
## S0=100,E=98.50,T=2,N=1000,r=0.03,sigma=np.sqrt(0.15),cp=-1.
```

```
The European put option price using Jarrow-Rudd Parameters is  24.6097
```

At N = 400 with Jarrow-Rudd Method for European option

In [108]:

```
import pandas as pd           ## import from module scipy all functions
import numpy as np           ## import from module numpy all functions
def Eur_putprice_Jarrow(S0,E,T,N,r,sigma,cp):      ##  Defining the function Eur_putprice to calculate
        ## the price of a European put option
    dt=T/N
    p=r-(0.5*sigma**2)
    U=np.exp(p*dt+sigma*np.sqrt(dt))
    D=np.exp(p*dt-sigma*np.sqrt(dt))
    R=(np.exp(r*dt))
    q=(R-D)/(U-D)
    ## defining delta_t, parameters U, D, R and risk-neutrall probability q
    ## Here we use Jarrow-Rudd parameters for U and D.
    V_old=np.zeros(N+1)
    V_new=np.zeros(N+1)
    ## initializing two N+1 dimensional vectors with zeros to store option prices
    for m in range(0,N+1):   ## cycle - m gets the values 0,1, ...,N
        V_old[m]=max(S0*U**m*D**(N-m)-E,0)
        ## first cycle - Computing the prices of the option (depending on the price S(T)=S0*U**D**(N-m)
        ##at time T=N*dt (see formula V(N,m) for put option). The prices of the option at time T
        ## equal to the payments at time T.
    for j in range(N-1,-1,-1):
        ## cycle for formula V(j,m), j=N-1, N-2,...,0
        for m in range(0,j+1):
            V_new[m]=(q*V_old[m+1]+(1-q)*V_old[m])/R
            ## computing the option prices at time j*dt if we know the prices at time (j+1)*dt.
            ## (see formula for V(j,m))
        for m in range(0,j+1):
            V_old[m]=V_new[m]
            ## after that the all option prices at time j*dt are known, we consider these prices as old
 prices.
    return(V_new[0])
    ## V_new[0] is the option price at t=0

print('The European put option price using Jarrow-Rudd Parameters is %8.4f' % Eur_putprice_Jarrow(S0=10
0,E=98.5,T=2,N=400,r=0.03,sigma=np.sqrt(0.15),cp=-1))
## finding and printing European put option with parameters
## S0=100,E=98.50,T=2,N=1000,r=0.03,sigma=np.sqrt(0.15),cp=-1.
```

```
The European put option price using Jarrow-Rudd Parameters is  24.6177
```

At N = 500 with Jarrow-Rudd Method for European option

In [109]:

```
import pandas as pd           ## import from module scipy all functions
import numpy as np           ## import from module numpy all functions
def Eur_putprice_Jarrow(S0,E,T,N,r,sigma,cp):      ##  Defining the function Eur_putprice to calculate
        ## the price of a European put option
    dt=T/N
    p=r-(0.5*sigma**2)
    U=np.exp(p*dt+sigma*np.sqrt(dt))
    D=np.exp(p*dt-sigma*np.sqrt(dt))
    R=(np.exp(r*dt))
    q=(R-D)/(U-D)
    ## defining delta_t, parameters U, D, R and risk-neutrall probability q
    ## Here we use Jarrow-Rudd parameters for U and D.
    V_old=np.zeros(N+1)
    V_new=np.zeros(N+1)
    ## initializing two N+1 dimensional vectors with zeros to store option prices
    for m in range(0,N+1):   ## cycle - m gets the values 0,1, ...,N
        V_old[m]=max(S0*U**m*D**(N-m)-E,0)
        ## first cycle - Computing the prices of the option (depending on the price S(T)=S0*U**D**(N-m)
        ##at time T=N*dt (see formula V(N,m) for put option). The prices of the option at time T
        ## equal to the payments at time T.
    for j in range(N-1,-1,-1):
        ## cycle for formula V(j,m), j=N-1, N-2,...,0
        for m in range(0,j+1):
            V_new[m]=(q*V_old[m+1]+(1-q)*V_old[m])/R
            ## computing the option prices at time j*dt if we know the prices at time (j+1)*dt.
            ## (see formula for V(j,m))
        for m in range(0,j+1):
            V_old[m]=V_new[m]
            ## after that the all option prices at time j*dt are known, we consider these prices as old
 prices.
    return(V_new[0])
    ## V_new[0] is the option price at t=0

print('The European put option price using Jarrow-Rudd Parameters is %8.4f' % Eur_putprice_Jarrow(S0=10
0,E=98.5,T=2,N=500,r=0.03,sigma=np.sqrt(0.15),cp=-1))
## finding and printing European put option with parameters
## S0=100,E=98.50,T=2,N=1000,r=0.03,sigma=np.sqrt(0.15),cp=-1.
```

```
The European put option price using Jarrow-Rudd Parameters is  24.6172
```

Third Method is: Leisen-Reimer method

Definying sign(z) which is equal to 1

That's because h(z) is a discrete approximation to the cummulative distribution function for a normal distribution. so when I use the numpy.sign(z) to validate my answer. I find that output is approximately is 0.7950. So sign(z) equal 1..

At N = 300 with Leisen-Reimer Method for European option

In [110]:

```
# Try to find the value of sign(z)


# Assume that sign(z) equal is 1
sign_z = np.sqrt(1-np.exp(-1))
print(sign_z)

# Assume that sign(z) equal is 0
sign_z0 = np.sqrt(1-np.exp(0))
print(sign_z0)
```

```
0.7950600976206501
0.0
```

In [111]:

```
import pandas as pd           ## import from module scipy all functions
import numpy as np    ## import from module numpy all functions
import math
def Eur_putprice_Leisen(S0,E,T,N,r,sigma,cp):      ##  Defining the function Eur_putprice to calculate
        ## the price of a European put option
    dt=T/N
    R=(np.exp(r*dt))
    z=1
    N=N+1
    a=(z**2/(N+1/3+0.1/(N+1))**2)*(N+1/6)
    h=0.5+0.5*z*np.sqrt(1-np.exp(-a))
    d1=(math.log(S0/E)+(r+0.5*sigma**2)*T)/sigma*np.sqrt(T)
    d2=d1-(sigma*np.sqrt(T))
    p=r-(0.5*sigma**2)
    q_quote=0.5+0.5*z*np.sqrt(1-np.exp(-d1))
    q=0.5+0.5*z*np.sqrt(1-np.exp(-d2))
    U=(R*q_quote)/q
    D=(R-q*U)/(1-q)
    ## defining delta_t, parameters U, D, R and risk-neutrall probability q
    ## Here we use Leisen-Reimer parameters for U and D.
    V_old=np.zeros(N+1)
    V_new=np.zeros(N+1)
    ## initializing two N+1 dimensional vectors with zeros to store option prices
    for m in range(0,N+1):   ## cycle - m gets the values 0,1, ...,N
        V_old[m]=max(S0*U**m*D**(N-m)-E,0)
        ## first cycle - Computing the prices of the option (depending on the price S(T)=S0*U**D**(N-m)
        ##at time T=N*dt (see formula V(N,m) for put option). The prices of the option at time T
        ## equal to the payments at time T.
    for j in range(N-1,-1,-1):
        ## cycle for formula V(j,m), j=N-1, N-2,...,0
        for m in range(0,j+1):
            V_new[m]=(q*V_old[m+1]+(1-q)*V_old[m])/R
            ## computing the option prices at time j*dt if we know the prices at time (j+1)*dt.
            ## (see formula for V(j,m))
        for m in range(0,j+1):
            V_old[m]=V_new[m]
            ## after that the all option prices at time j*dt are known, we consider these prices as old
 prices.
    return(V_new[0])
    ## V_new[0] is the option price at t=0

print('The European put option price using Leisen-Reimer Parameters is %8.4f' % Eur_putprice_Leisen(S0=
100,E=98.5,T=2,N=300,r=0.03,sigma=np.sqrt(0.15),cp=-1))
## finding and printing European put option with parameters
## S0=100,E=98.50,T=2,N=1000,r=0.03,sigma=np.sqrt(0.15),cp=-1.
```

```
The European put option price using Leisen-Reimer Parameters is  99.6543
```

At N = 400 with Leisen-Reimer Method for European option

In [112]:

```
import pandas as pd          ## import from module scipy all functions
import numpy as np   ## import from module numpy all functions
import math
def Eur_putprice_Leisen(S0,E,T,N,r,sigma,cp):      ##  Defining the function Eur_putprice to calculate
        ## the price of a European put option
    dt=T/N
    R=(np.exp(r*dt))
    z=1
    N=N+1
    a=(z**2/(N+1/3+0.1/(N+1))**2)*(N+1/6)
    h=0.5+0.5*z*np.sqrt(1-np.exp(-a))
    d1=(math.log(S0/E)+(r+0.5*sigma**2)*T)/sigma*np.sqrt(T)
    d2=d1-(sigma*np.sqrt(T))
    p=r-(0.5*sigma**2)
    q_quote=0.5+0.5*z*np.sqrt(1-np.exp(-d1))
    q=0.5+0.5*z*np.sqrt(1-np.exp(-d2))
    U=(R*q_quote)/q
    D=(R-q*U)/(1-q)
    ## defining delta_t, parameters U, D, R and risk-neutrall probability q
    ## Here we use Leisen-Reimer parameters for U and D.
    V_old=np.zeros(N+1)
    V_new=np.zeros(N+1)
    ## initializing two N+1 dimensional vectors with zeros to store option prices
    for m in range(0,N+1):   ## cycle - m gets the values 0,1, ...,N
        V_old[m]=max(S0*U**m*D**(N-m)-E,0)
        ## first cycle - Computing the prices of the option (depending on the price S(T)=S0*U**D**(N-m)
        ##at time T=N*dt (see formula V(N,m) for put option). The prices of the option at time T
        ## equal to the payments at time T.
    for j in range(N-1,-1,-1):
        ## cycle for formula V(j,m), j=N-1, N-2,...,0
        for m in range(0,j+1):
            V_new[m]=(q*V_old[m+1]+(1-q)*V_old[m])/R
            ## computing the option prices at time j*dt if we know the prices at time (j+1)*dt.
            ## (see formula for V(j,m))
        for m in range(0,j+1):
            V_old[m]=V_new[m]
            ## after that the all option prices at time j*dt are known, we consider these prices as old
 prices.
    return(V_new[0])
    ## V_new[0] is the option price at t=0

print('The European put option price using Leisen-Reimer Parameters is %8.4f' % Eur_putprice_Leisen(S0=
100,E=98.5,T=2,N=400,r=0.03,sigma=np.sqrt(0.15),cp=-1))
## finding and printing European put option with parameters
## S0=100,E=98.50,T=2,N=1000,r=0.03,sigma=np.sqrt(0.15),cp=-1.
```

```
The European put option price using Leisen-Reimer Parameters is  99.9238
```

At N = 500 with Leisen-Reimer Method for European option

In [113]:

```
import pandas as pd          ## import from module scipy all functions
import numpy as np   ## import from module numpy all functions
import math
def Eur_putprice_Leisen(S0,E,T,N,r,sigma,cp):      ##  Defining the function Eur_putprice to calculate
        ## the price of a European put option
    dt=T/N
    R=(np.exp(r*dt))
    z=1
    N=N+1
    a=(z**2/(N+1/3+0.1/(N+1))**2)*(N+1/6)
    h=0.5+0.5*z*np.sqrt(1-np.exp(-a))
    d1=(math.log(S0/E)+(r+0.5*sigma**2)*T)/sigma*np.sqrt(T)
    d2=d1-(sigma*np.sqrt(T))
    p=r-(0.5*sigma**2)
    q_quote=0.5+0.5*z*np.sqrt(1-np.exp(-d1))
    q=0.5+0.5*z*np.sqrt(1-np.exp(-d2))
    U=(R*q_quote)/q
    D=(R-q*U)/(1-q)
    ## defining delta_t, parameters U, D, R and risk-neutrall probability q
    ## Here we use Leisen-Reimer parameters for U and D.
    V_old=np.zeros(N+1)
    V_new=np.zeros(N+1)
    ## initializing two N+1 dimensional vectors with zeros to store option prices
    for m in range(0,N+1):   ## cycle - m gets the values 0,1, ...,N
        V_old[m]=max(S0*U**m*D**(N-m)-E,0)
        ## first cycle - Computing the prices of the option (depending on the price S(T)=S0*U**D**(N-m)
        ##at time T=N*dt (see formula V(N,m) for put option). The prices of the option at time T
        ## equal to the payments at time T.
    for j in range(N-1,-1,-1):
        ## cycle for formula V(j,m), j=N-1, N-2,...,0
        for m in range(0,j+1):
            V_new[m]=(q*V_old[m+1]+(1-q)*V_old[m])/R
            ## computing the option prices at time j*dt if we know the prices at time (j+1)*dt.
            ## (see formula for V(j,m))
        for m in range(0,j+1):
            V_old[m]=V_new[m]
            ## after that the all option prices at time j*dt are known, we consider these prices as old
 prices.
    return(V_new[0])
    ## V_new[0] is the option price at t=0

print('The European put option price using Leisen-Reimer Parameters is %8.4f' % Eur_putprice_Leisen(S0=
100,E=98.5,T=2,N=500,r=0.03,sigma=np.sqrt(0.15),cp=-1))
## finding and printing European put option with parameters
## S0=100,E=98.50,T=2,N=1000,r=0.03,sigma=np.sqrt(0.15),cp=-1.
```

```
The European put option price using Leisen-Reimer Parameters is  99.9833
```

Compute the errors of European option in case N=300, N=400 and N=500 for CRR Method:

In [114]:

```
error_CRR300 = ((Eur_putprice(S0=100,E=98.5,T=2,N=300,r=0.03,sigma=np.sqrt(0.15),cp=-1)) - Vput)
print('The European put option price at N = 300 - CRR Method is %8.4f' % error_CRR300)

error_CRR400 = ((Eur_putprice(S0=100,E=98.5,T=2,N=400,r=0.03,sigma=np.sqrt(0.15),cp=-1)) - Vput)
print('The European put option price at N = 400 - CRR Method is %8.4f' % error_CRR400)

error_CRR500 = ((Eur_putprice(S0=100,E=98.5,T=2,N=500,r=0.03,sigma=np.sqrt(0.15),cp=-1)) - Vput)
print('The European put option price at N = 500 - CRR Method Method is %8.4f' % error_CRR500)
```

```
The European put option price at N = 300 - CRR Method is   0.0070
The European put option price at N = 400 - CRR Method is   0.0070
The European put option price at N = 500 - CRR Method Method is   0.0067
```

Compute the errors of European option in case N=300, N=400 and N=500 for Jarrow-Rudd Method:

In [115]:

```
error_Jarrow300 = (Eur_putprice_Jarrow(S0=100,E=98.5,T=2,N=300,r=0.03,sigma=np.sqrt(0.15),cp=-1) - Vput
)
print('The European put option price at N = 300 - Jarrow-Rudd Method is %8.4f' % error_Jarrow300)

error_Jarrow400 = (Eur_putprice_Jarrow(S0=100,E=98.5,T=2,N=400,r=0.03,sigma=np.sqrt(0.15),cp=-1) - Vput
)
print('The European put option price at N = 400 - Jarrow-Rudd Method is %8.4f' % error_Jarrow400)

error_Jarrow500 = (Eur_putprice_Jarrow(S0=100,E=98.5,T=2,N=500,r=0.03,sigma=np.sqrt(0.15),cp=-1) - Vput
)
print('The European put option price at N = 500 - Jarrow-Rudd Method is %8.4f' % error_Jarrow500)
```

```
The European put option price at N = 300 - Jarrow-Rudd Method is   0.0012
The European put option price at N = 400 - Jarrow-Rudd Method is   0.0092
The European put option price at N = 500 - Jarrow-Rudd Method is   0.0087
```

Compute the errors of European option in case N=300, N=400 and N=500 for Leisen-Reimer Method:

In [116]:

```
error_Leisen300 = ((Eur_putprice_Leisen(S0=100,E=98.5,T=2,N=300,r=0.03,sigma=np.sqrt(0.15),cp=-1)) - Vp
ut)
print('The European put option price at N = 300 - Leisen-Reimer Method is %8.4f' % error_Leisen300)

error_Leisen400 = ((Eur_putprice_Leisen(S0=100,E=98.5,T=2,N=400,r=0.03,sigma=np.sqrt(0.15),cp=-1)) - Vp
ut)
print('The European put option price at N = 400 - Leisen-Reimer Method is %8.4f' % error_Leisen400)

error_Leisen500 = ((Eur_putprice_Leisen(S0=100,E=98.5,T=2,N=500,r=0.03,sigma=np.sqrt(0.15),cp=-1)) - Vp
ut)
print('The European put option price at N = 500 - Leisen-Reimer Method is %8.4f' % error_Leisen500)
```

```
The European put option price at N = 300 - Leisen-Reimer Method is  75.0458
The European put option price at N = 400 - Leisen-Reimer Method is  75.3153
The European put option price at N = 500 - Leisen-Reimer Method is  75.3748
```

Comment on the European option using(CRR - Jarrow-Rudd - Leisen-Reimer) Method at N=300, 400, 500:

- For CRR Method: We can see that the European option price at N = 300, 400, 500 are 24.6155, 24.6155, 24.6152 respectively.
- The limit price for European option is: 24.6085, which we are going to use it later to compute the values of errors for each method at different values of N.
- N is an important factor in our model as it express the number of time periods.
- For Jarrow-Rudd Method: We can see that the European option price at N = 300, 400, 500 are 24.6097, 24.6177, 24.6172 respectively.
- For Leisen-Reimer Method: We can see that the European option price at N = 300, 400, 500 are 99.6543, 99.9238, 99.9833 respectively. (Based on the interesting note in the page 97, the model parameters for this method helps in order to achieve faster convergence).
- Finally, when it comes to the calculation of the errors wether for CRR & Jarrow-Rudd Methods. We witnessed quite similar values. Not like the Leisen-Reimer Method.

Find the prices of American option:

At N = 300 with CRR Method

In [117]:

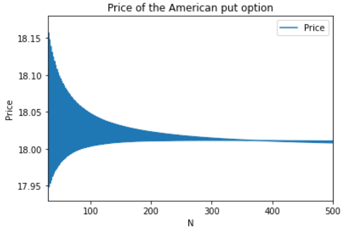
```
def Amer_price(S0,E,T,N,r,sigma,cp):
    dt=T/N
    U=np.exp(sigma*np.sqrt(dt))
    D=np.exp(-sigma*np.sqrt(dt))
    R=(np.exp(r*dt))
    q=(R-D)/(U-D)
    V_old=np.zeros(N+1)
    V_new=np.zeros(N+1)
    for m in range(0,N+1):
        V_old[m]=max(cp*(S0*U**m*D**(N-m)-E),0)
    for j in range(N-1,-1,-1):
        for m in range(0,j+1):
            V_new[m]=(q*V_old[m+1]+(1-q)*V_old[m])/R
            V_new[m]=max(V_new[m],max(cp*(S0*U**m*D**(j-m)-E),0))
        for m in range(0,j+1):
            V_old[m]=V_new[m]
    return(V_new[0])
print('The American put option price is %8.4f.' % Amer_price(S0=100,E=98.5,T=2,N=300,r=0.03,sigma=np.sq
rt(0.15),cp=-1))
```

```
The American put option price is  18.0115.
```

Plot the graph of the American option and the limit price in one graph:

In [118]:

```
import matplotlib.pyplot as plt
## Assign a short name to a module to use it when referring to a function in the module
## for example, instead matplotlib.pyplot.xlim we use plt.xlim.
V=np.zeros(501)
##initializing vector V with zeros to store option prices
for k in range(30,501):
    V[k]=Amer_price(S0=100,E=98.5,T=2,N=k,r=0.03,sigma=np.sqrt(0.15),cp=-1)
    ## computing the prices
n =np.linspace(0,500,501)
#defining vector of x-values (number of the period)
plt.plot(n, V)
## definining the plot, the x-values are in vector n, y-values are in vector V.
plt.xlim((30, 500))
## limits of x-axis
plt.ylim((17.93,18.18))
## limits of y-axis (limits of option prices)
plt.ylabel('Price')
plt.xlabel('N')
plt.legend(['Price'], loc='best')
plt.title('Price of the American put option')
plt.show()
## show plot
```



At N = 400 with CRR Method for American option

In [119]:

```
def Amer_price(S0,E,T,N,r,sigma,cp):
    dt=T/N
    U=np.exp(sigma*np.sqrt(dt))
    D=np.exp(-sigma*np.sqrt(dt))
    R=(np.exp(r*dt))
    q=(R-D)/(U-D)
    V_old=np.zeros(N+1)
    V_new=np.zeros(N+1)
    for m in range(0,N+1):
        V_old[m]=max(cp*(S0*U**m*D**(N-m)-E),0)
    for j in range(N-1,-1,-1):
        for m in range(0,j+1):
            V_new[m]=(q*V_old[m+1]+(1-q)*V_old[m])/R
            V_new[m]=max(V_new[m],max(cp*(S0*U**m*D**(j-m)-E),0))
        for m in range(0,j+1):
            V_old[m]=V_new[m]
    return(V_new[0])
print('The American put option price is %8.4f.' % Amer_price(S0=100,E=98.5,T=2,N=400,r=0.03,sigma=np.sq
rt(0.15),cp=-1))
```

```
The American put option price is  18.0112.
```

At N = 500 with CRR Method for American option

In [120]:

```
def Amer_price(S0,E,T,N,r,sigma,cp):
    dt=T/N
    U=np.exp(sigma*np.sqrt(dt))
    D=np.exp(-sigma*np.sqrt(dt))
    R=(np.exp(r*dt))
    q=(R-D)/(U-D)
    V_old=np.zeros(N+1)
    V_new=np.zeros(N+1)
    for m in range(0,N+1):
        V_old[m]=max(cp*(S0*U**m*D**(N-m)-E),0)
    for j in range(N-1,-1,-1):
        for m in range(0,j+1):
            V_new[m]=(q*V_old[m+1]+(1-q)*V_old[m])/R
            V_new[m]=max(V_new[m],max(cp*(S0*U**m*D**(j-m)-E),0))
        for m in range(0,j+1):
            V_old[m]=V_new[m]
    return(V_new[0])
print('The American put option price is %8.4f.' % Amer_price(S0=100,E=98.5,T=2,N=500,r=0.03,sigma=np.sq
rt(0.15),cp=-1))
```

```
The American put option price is  18.0107.
```

At N = 300 with Jarrow-Rudd Method for American option

In [133]:

```
import pandas as pd          ## import from module scipy all functions
import numpy as np          ## import from module numpy all functions
def Amer_price(S0,E,T,N,r,sigma,cp):      ##  Defining the function AMR_putprice to calculate
        ## the price of a American put option
    dt=T/N
    p=r-(0.5*sigma**2)
    U=np.exp(p*dt+sigma*np.sqrt(dt))
    D=np.exp(p*dt-sigma*np.sqrt(dt))
    R=(np.exp(r*dt))
    q=(R-D)/(U-D)
    ## defining delta_t, parameters U, D, R and risk-neutrall probability q
    ## Here we use Jarrow-Rudd parameters for U and D.
    V_old=np.zeros(N+1)
    V_new=np.zeros(N+1)
    ## initializing two N+1 dimensional vectors with zeros to store option prices
    for m in range(0,N+1):
        V_old[m]=max(cp*(S0*U**m*D**(N-m)-E),0)
    for j in range(N-1,-1,-1):
        for m in range(0,j+1):
            V_new[m]=(q*V_old[m+1]+(1-q)*V_old[m])/R
            V_new[m]=max(V_new[m],max(cp*(S0*U**m*D**(j-m)-E),0))
        for m in range(0,j+1):
            V_old[m]=V_new[m]
    return(V_new[0])
    ## V_new[0] is the option price at t=0

print('The American put option price using Jarrow-Rudd Parameters is %8.4f' % Amer_price(S0=100,E=98.5,
T=2,N=300,r=0.03,sigma=np.sqrt(0.15),cp=-1))
## finding and printing American put option with parameters
## S0=100,E=98.50,T=2,N=1000,r=0.03,sigma=np.sqrt(0.15),cp=-1.
```

```
The American put option price using Jarrow-Rudd Parameters is  18.0096
```

At N = 400 with Jarrow-Rudd Method for American option

In [132]:

```
import pandas as pd          ## import from module scipy all functions
import numpy as np          ## import from module numpy all functions
def Amer_price(S0,E,T,N,r,sigma,cp):      ##  Defining the function AMR_putprice to calculate
        ## the price of a American put option
    dt=T/N
    p=r-(0.5*sigma**2)
    U=np.exp(p*dt+sigma*np.sqrt(dt))
    D=np.exp(p*dt-sigma*np.sqrt(dt))
    R=(np.exp(r*dt))
    q=(R-D)/(U-D)
    ## defining delta_t, parameters U, D, R and risk-neutrall probability q
    ## Here we use Jarrow-Rudd parameters for U and D.
    V_old=np.zeros(N+1)
    V_new=np.zeros(N+1)
    ## initializing two N+1 dimensional vectors with zeros to store option prices
    for m in range(0,N+1):
        V_old[m]=max(cp*(S0*U**m*D**(N-m)-E),0)
    for j in range(N-1,-1,-1):
        for m in range(0,j+1):
            V_new[m]=(q*V_old[m+1]+(1-q)*V_old[m])/R
            V_new[m]=max(V_new[m],max(cp*(S0*U**m*D**(j-m)-E),0))
        for m in range(0,j+1):
            V_old[m]=V_new[m]
    return(V_new[0])
    ## V_new[0] is the option price at t=0

print('The American put option price using Jarrow-Rudd Parameters is %8.4f' % Amer_price(S0=100,E=98.5,
T=2,N=400,r=0.03,sigma=np.sqrt(0.15),cp=-1))
## finding and printing American put option with parameters
## S0=100,E=98.50,T=2,N=1000,r=0.03,sigma=np.sqrt(0.15),cp=-1.
```

```
The American put option price using Jarrow-Rudd Parameters is  18.0144
```

At N = 500 with Jarrow-Rudd Method for American option

In [131]:

```
import pandas as pd          ## import from module scipy all functions
import numpy as np          ## import from module numpy all functions
def Amer_price(S0,E,T,N,r,sigma,cp):      ##  Defining the function AMR_putprice to calculate
        ## the price of a American put option
    dt=T/N
    p=r-(0.5*sigma**2)
    U=np.exp(p*dt+sigma*np.sqrt(dt))
    D=np.exp(p*dt-sigma*np.sqrt(dt))
    R=(np.exp(r*dt))
    q=(R-D)/(U-D)
    ## defining delta_t, parameters U, D, R and risk-neutrall probability q
    ## Here we use Jarrow-Rudd parameters for U and D.
    V_old=np.zeros(N+1)
    V_new=np.zeros(N+1)
    ## initializing two N+1 dimensional vectors with zeros to store option prices
    for m in range(0,N+1):
        V_old[m]=max(cp*(S0*U**m*D**(N-m)-E),0)
    for j in range(N-1,-1,-1):
        for m in range(0,j+1):
            V_new[m]=(q*V_old[m+1]+(1-q)*V_old[m])/R
            V_new[m]=max(V_new[m],max(cp*(S0*U**m*D**(j-m)-E),0))
        for m in range(0,j+1):
            V_old[m]=V_new[m]
    return(V_new[0])
    ## V_new[0] is the option price at t=0

print('The American put option price using Jarrow-Rudd Parameters is %8.4f' % Amer_price(S0=100,E=98.5,
T=2,N=500,r=0.03,sigma=np.sqrt(0.15),cp=-1))
## finding and printing American put option with parameters
## S0=100,E=98.50,T=2,N=1000,r=0.03,sigma=np.sqrt(0.15),cp=-1.
```

```
The American put option price using Jarrow-Rudd Parameters is  18.0130
```

At N = 300 with Leisen-Reimer Method for American option

In [130]:
```python
import pandas as pd          ## import from module scipy all functions
import numpy as np   ## import from module numpy all functions
import math
def AMR_putprice_Leisen(S0,E,T,N,r,sigma,cp):      ##  Defining the function AMR_putprice to calculate
        ## the price of a American put option
    dt=T/N
    R=(np.exp(r*dt))
    z=1
    N=N+1
    a=(z**2/(N+1/3+0.1/(N+1))**2)*(N+1/6)
    h=0.5+0.5*z*np.sqrt(1-np.exp(-a))
    d1=(math.log(S0/E)+(r+0.5*sigma**2)*T)/sigma*np.sqrt(T)
    d2=d1-(sigma*np.sqrt(T))
    p=r-(0.5*sigma**2)
    q_quote=0.5+0.5*z*np.sqrt(1-np.exp(-d1))
    q=0.5+0.5*z*np.sqrt(1-np.exp(-d2))
    U=(R*q_quote)/q
    D=(R-q*U)/(1-q)
    ## defining delta_t, parameters U, D, R and risk-neutrall probability q
    ## Here we use Leisen-Reimer parameters for U and D.
    V_old=np.zeros(N+1)
    V_new=np.zeros(N+1)
    ## initializing two N+1 dimensional vectors with zeros to store option prices
    for m in range(0,N+1):
        V_old[m]=max(cp*(S0*U**m*D**(N-m)-E),0)
    for j in range(N-1,-1,-1):
        for m in range(0,j+1):
            V_new[m]=(q*V_old[m+1]+(1-q)*V_old[m])/R
            V_new[m]=max(V_new[m],max(cp*(S0*U**m*D**(j-m)-E),0))
        for m in range(0,j+1):
            V_old[m]=V_new[m]
    return(V_new[0])
    ## V_new[0] is the option price at t=0

print('The American put option price using Leisen-Reimer Parameters is %8.4f' % AMR_putprice_Leisen(S0=
100,E=98.5,T=2,N=300,r=0.03,sigma=np.sqrt(0.15),cp=-1))
## finding and printing American put option with parameters
## S0=100,E=98.50,T=2,N=1000,r=0.03,sigma=np.sqrt(0.15),cp=-1.
```

```
The American put option price using Leisen-Reimer Parameters is  96.0968
```

At N = 400 with Leisen-Reimer Method for American option

In [129]:
```python
import pandas as pd          ## import from module scipy all functions
import numpy as np   ## import from module numpy all functions
import math
def AMR_putprice_Leisen(S0,E,T,N,r,sigma,cp):      ##  Defining the function AMR_putprice to calculate
        ## the price of a American put option
    dt=T/N
    R=(np.exp(r*dt))
    z=1
    N=N+1
    a=(z**2/(N+1/3+0.1/(N+1))**2)*(N+1/6)
    h=0.5+0.5*z*np.sqrt(1-np.exp(-a))
    d1=(math.log(S0/E)+(r+0.5*sigma**2)*T)/sigma*np.sqrt(T)
    d2=d1-(sigma*np.sqrt(T))
    p=r-(0.5*sigma**2)
    q_quote=0.5+0.5*z*np.sqrt(1-np.exp(-d1))
    q=0.5+0.5*z*np.sqrt(1-np.exp(-d2))
    U=(R*q_quote)/q
    D=(R-q*U)/(1-q)
    ## defining delta_t, parameters U, D, R and risk-neutrall probability q
    ## Here we use Leisen-Reimer parameters for U and D.
    V_old=np.zeros(N+1)
    V_new=np.zeros(N+1)
    ## initializing two N+1 dimensional vectors with zeros to store option prices
    for m in range(0,N+1):
        V_old[m]=max(cp*(S0*U**m*D**(N-m)-E),0)
    for j in range(N-1,-1,-1):
        for m in range(0,j+1):
            V_new[m]=(q*V_old[m+1]+(1-q)*V_old[m])/R
            V_new[m]=max(V_new[m],max(cp*(S0*U**m*D**(j-m)-E),0))
        for m in range(0,j+1):
            V_old[m]=V_new[m]
    return(V_new[0])
    ## V_new[0] is the option price at t=0

print('The American put option price using Leisen-Reimer Parameters is %8.4f' % AMR_putprice_Leisen(S0=
100,E=98.5,T=2,N=400,r=0.03,sigma=np.sqrt(0.15),cp=-1))
## finding and printing American put option with parameters
## S0=100,E=98.50,T=2,N=1000,r=0.03,sigma=np.sqrt(0.15),cp=-1.
```

```
The American put option price using Leisen-Reimer Parameters is  96.7626
```

At N = 500 with Leisen-Reimer Method for American option

In [127]:
```python
import pandas as pd          ## import from module scipy all functions
import numpy as np   ## import from module numpy all functions
import math
def AMR_putprice_Leisen(S0,E,T,N,r,sigma,cp):      ##  Defining the function AMR_putprice to calculate
        ## the price of a American put option
    dt=T/N
    R=(np.exp(r*dt))
    z=1
    N=N+1
    a=(z**2/(N+1/3+0.1/(N+1))**2)*(N+1/6)
    h=0.5+0.5*z*np.sqrt(1-np.exp(-a))
    d1=(math.log(S0/E)+(r+0.5*sigma**2)*T)/sigma*np.sqrt(T)
    d2=d1-(sigma*np.sqrt(T))
    p=r-(0.5*sigma**2)
    q_quote=0.5+0.5*z*np.sqrt(1-np.exp(-d1))
    q=0.5+0.5*z*np.sqrt(1-np.exp(-d2))
    U=(R*q_quote)/q
    D=(R-q*U)/(1-q)
    ## defining delta_t, parameters U, D, R and risk-neutrall probability q
    ## Here we use Leisen-Reimer parameters for U and D.
    V_old=np.zeros(N+1)
    V_new=np.zeros(N+1)
    ## initializing two N+1 dimensional vectors with zeros to store option prices
    for m in range(0,N+1):
        V_old[m]=max(cp*(S0*U**m*D**(N-m)-E),0)
    for j in range(N-1,-1,-1):
        for m in range(0,j+1):
            V_new[m]=(q*V_old[m+1]+(1-q)*V_old[m])/R
            V_new[m]=max(V_new[m],max(cp*(S0*U**m*D**(j-m)-E),0))
        for m in range(0,j+1):
            V_old[m]=V_new[m]
    return(V_new[0])
    ## V_new[0] is the option price at t=0

print('The American put option price using Leisen-Reimer Parameters is %8.4f' % AMR_putprice_Leisen(S0=
100,E=98.5,T=2,N=500,r=0.03,sigma=np.sqrt(0.15),cp=-1))
## finding and printing American put option with parameters
## S0=100,E=98.50,T=2,N=1000,r=0.03,sigma=np.sqrt(0.15),cp=-1.
```

```
The American put option price using Leisen-Reimer Parameters is  97.0969
```

Comment on the American option using(CRR - Jarrow-Rudd - Leisen-Reimer) Method at N=300, 400, 500:

- For CRR Method: We can see that the American option price at N = 300, 400, 500 are 18.0115, 18.0114, 18.0107 respectively.
- For Jarrow-Rudd Method: We can see that the American option price at N = 300, 400, 500 are 18.0096, 18.0144, 18.0130 respectively.
- For Leisen-Reimer Method: We can see that the American option price at N = 300, 400, 500 are 96.0968, 96.7626, 97.0969 respectively. (Based on the interesting note in the page 97, the model parameters for this method helps in order to achieve faster convergence).