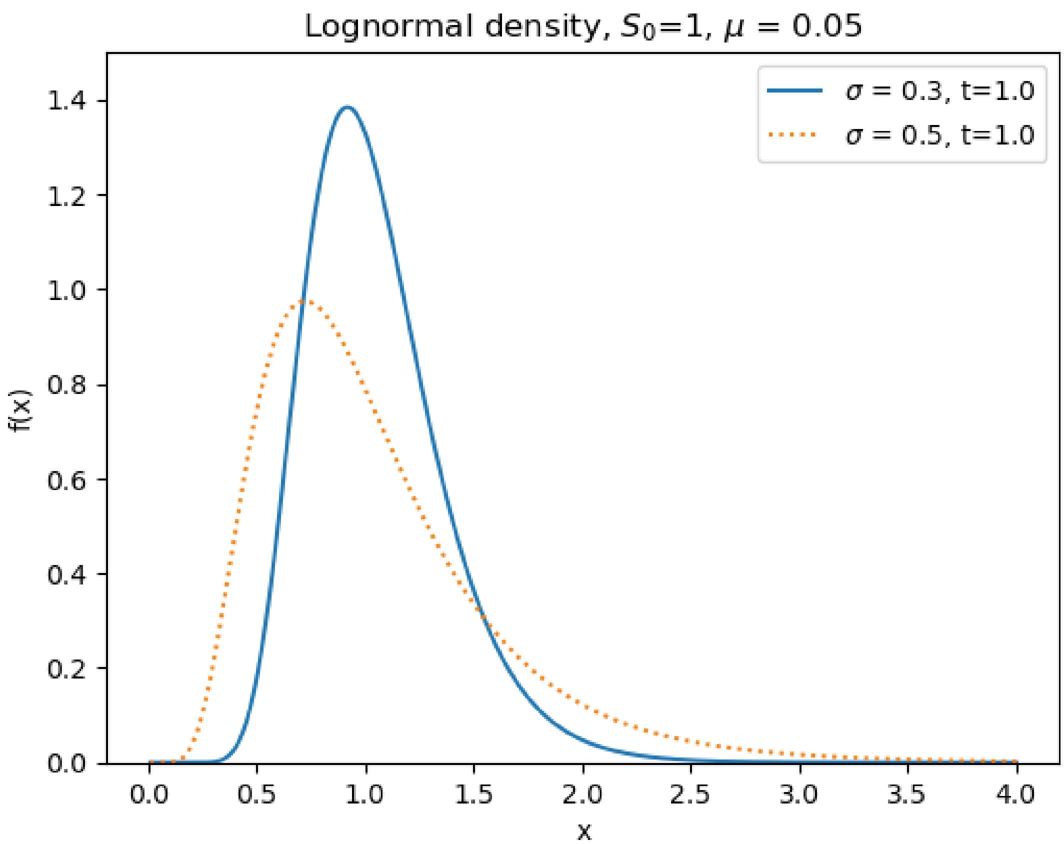
Practical codes-Feng

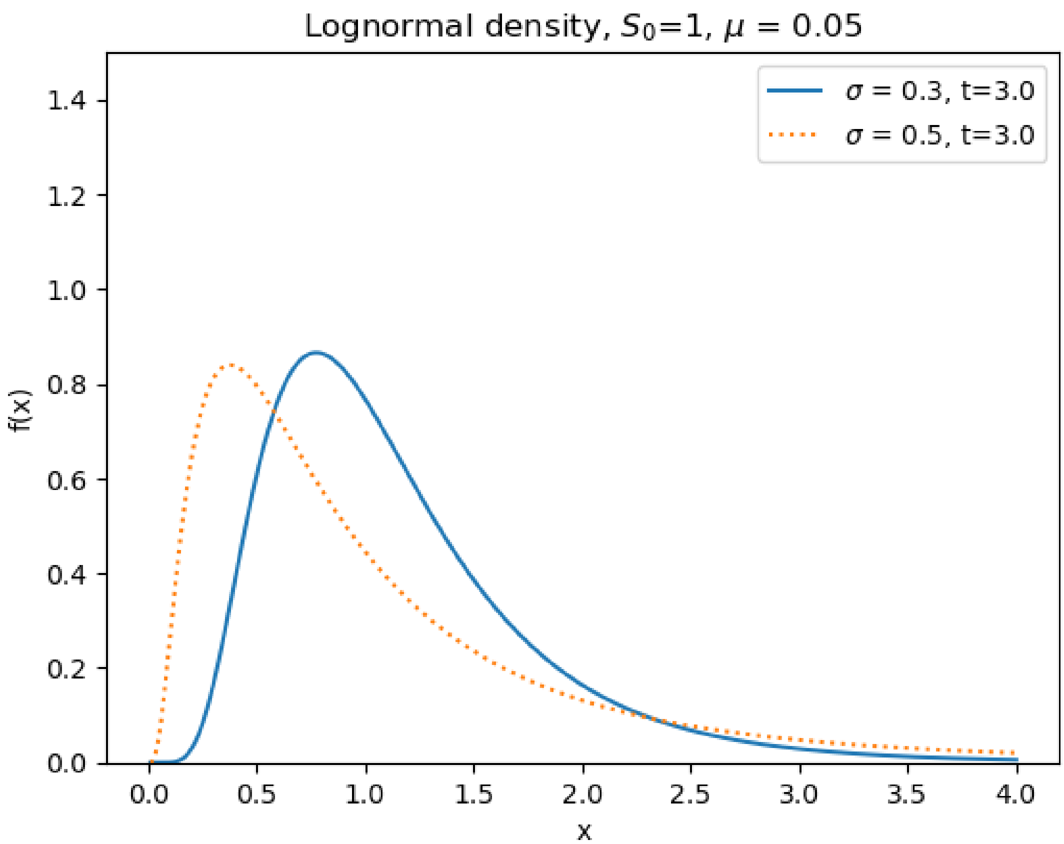
March 1, 2024

[2]:

```
import numpy as np
import matplotlib.pyplot as plt

# P1  Simulating the stock price process
# Parameters
mu = 0.05
S = 1
sigmas = [0.3, 0.5]

# (a)
times = [1, 3]
x = np.linspace(.01,4,500)
plt.figure()
for t in times:
    tempa = ((np.log(x / S) - (mu - 0.5 * sigmas[0] ** 2) * t) ** 2) / (2 * t *␣
 ↪sigmas[0] ** 2)
    tempb = x * sigmas[0] * np.sqrt(2 * np.pi * t)
    y1 = np.exp(-tempa)/tempb
    plt.plot(x,y1,'-',label='$\sigma$ = 0.3, t=%.1f' %t)
    plt.ylim([0,1.5])
    tempa = ((np.log(x / S) - (mu - 0.5 * sigmas[1] ** 2) * t) ** 2) / (2 * t *␣
 ↪sigmas[1] ** 2)
    tempb = x * sigmas[1] * np.sqrt(2 * np.pi * t)
    y2 = np.exp(-tempa)/tempb
    plt.plot(x,y2,':',label='$\sigma$ = 0.5, t=%.1f' %t)
    plt.legend()
    plt.title('Lognormal density, $S_0$=1, $\mu$ = 0.05')
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.show()
```

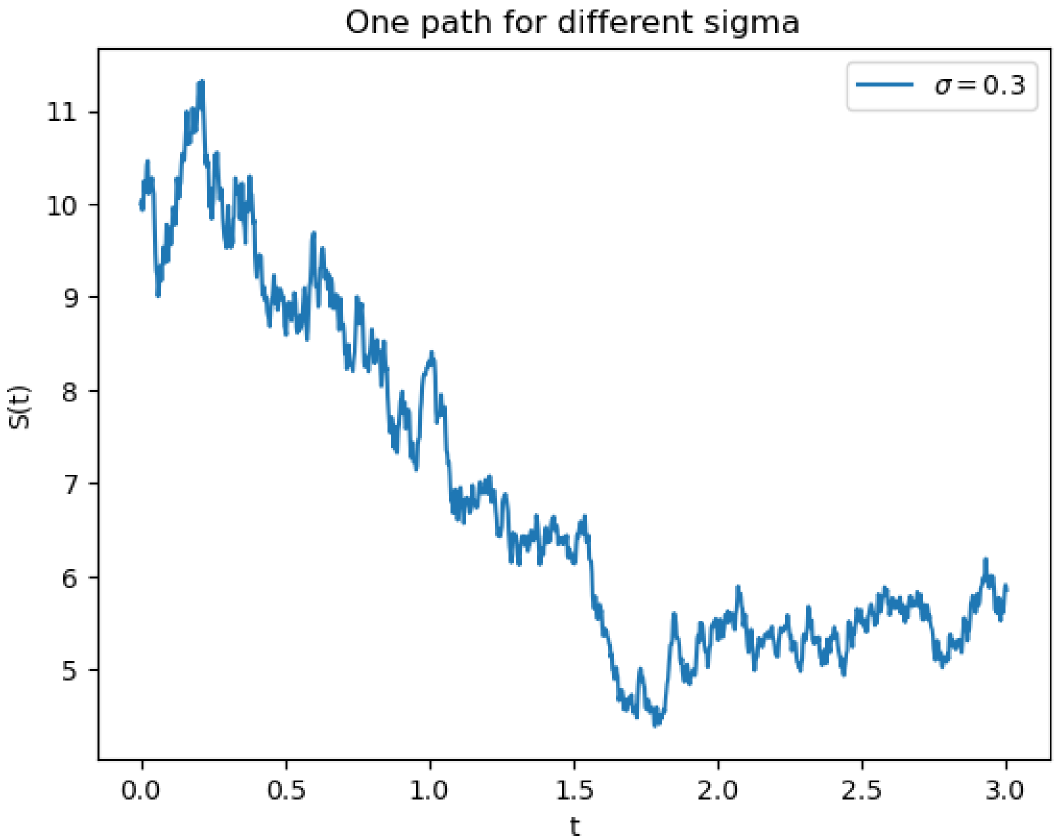


Lognormal density, S_0=1, μ = 0.05
σ = 0.3, t=1.0
σ = 0.5, t=1.0
f(x)
x
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

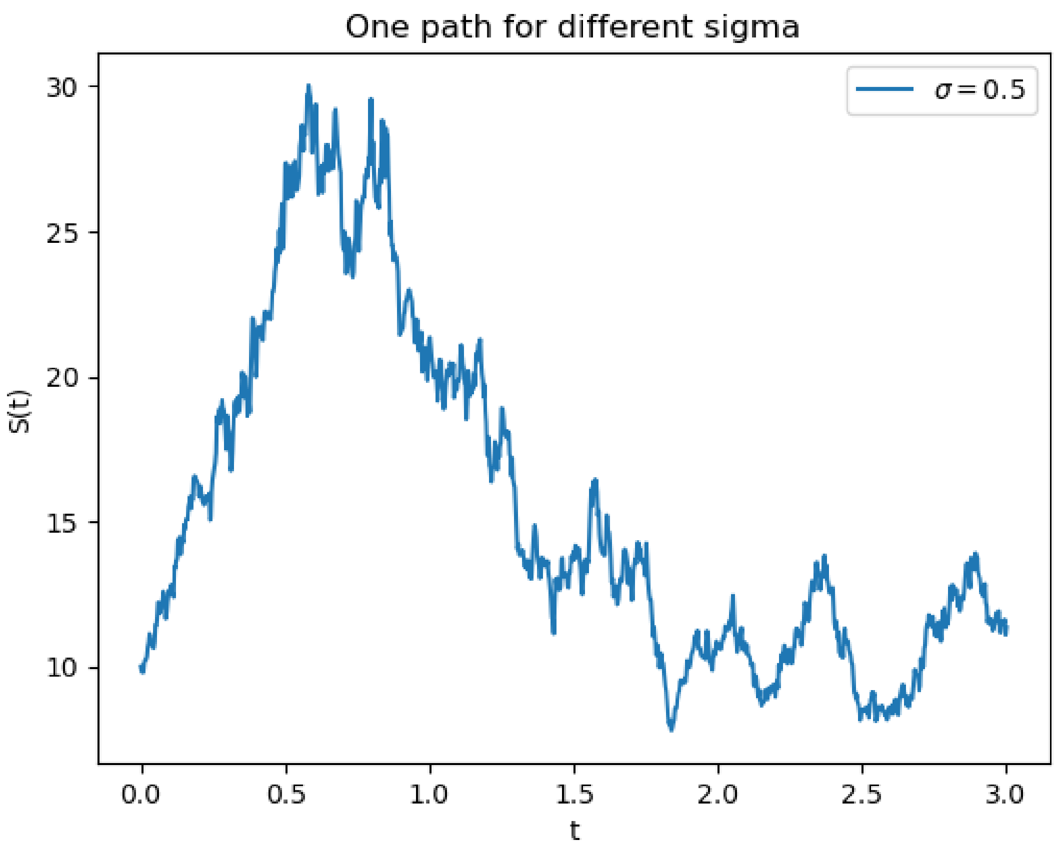


[28]:

```
# (b)
L = 1000
T = 3
dt = T/L
tvals = np.linspace(0,T,L+1)
plt.figure()
for sigma in sigmas:
    tvals = np.linspace(0, T, L + 1)
    Svals = S * np.cumprod(np.exp((mu - 0.5 * sigma ** 2) * dt + sigma * np.
 ↪sqrt(dt) * np.random.randn(L)))
    Svals = np.insert(Svals, 0, S)  # add initial asset price
    plt.plot(tvals, Svals.transpose(),label = '$\sigma=%.1f$' %sigma)
    plt.title('One path for different sigma')
    plt.legend()
    plt.xlabel('t')
    plt.ylabel('S(t)')
    plt.show()
```



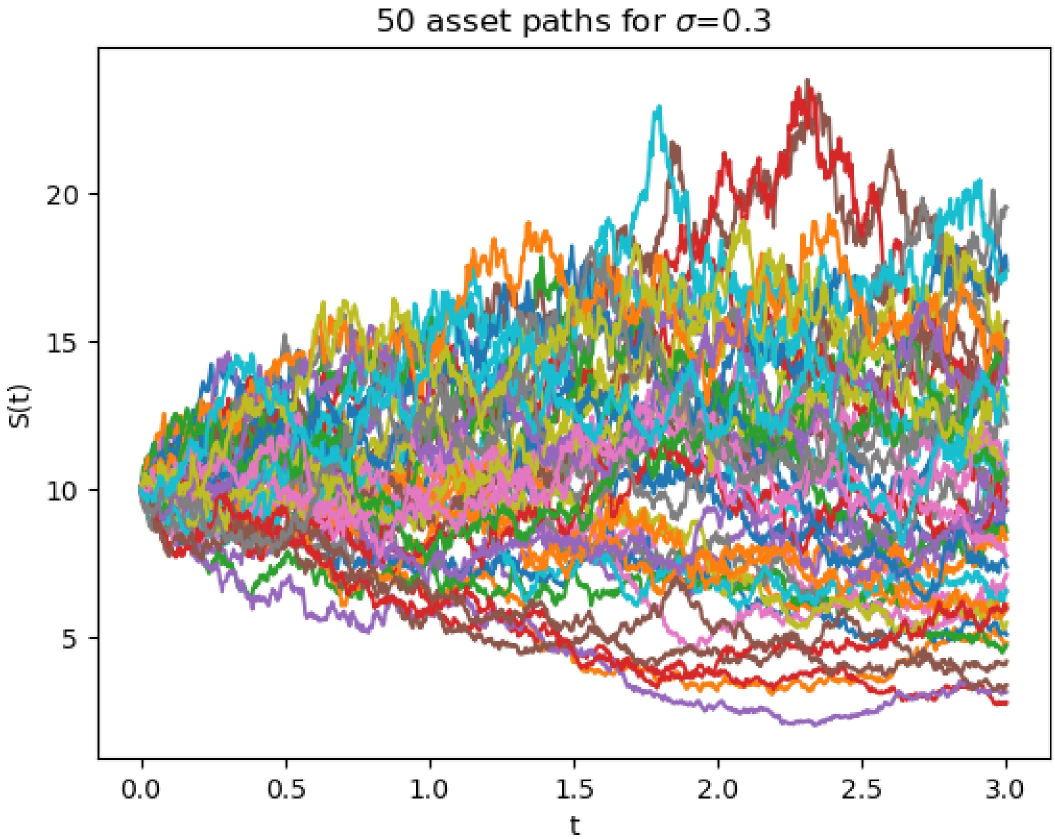
One path for different sigma
$\sigma = 0.3$
11
10
9
8
7
6
5
S(t)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
t

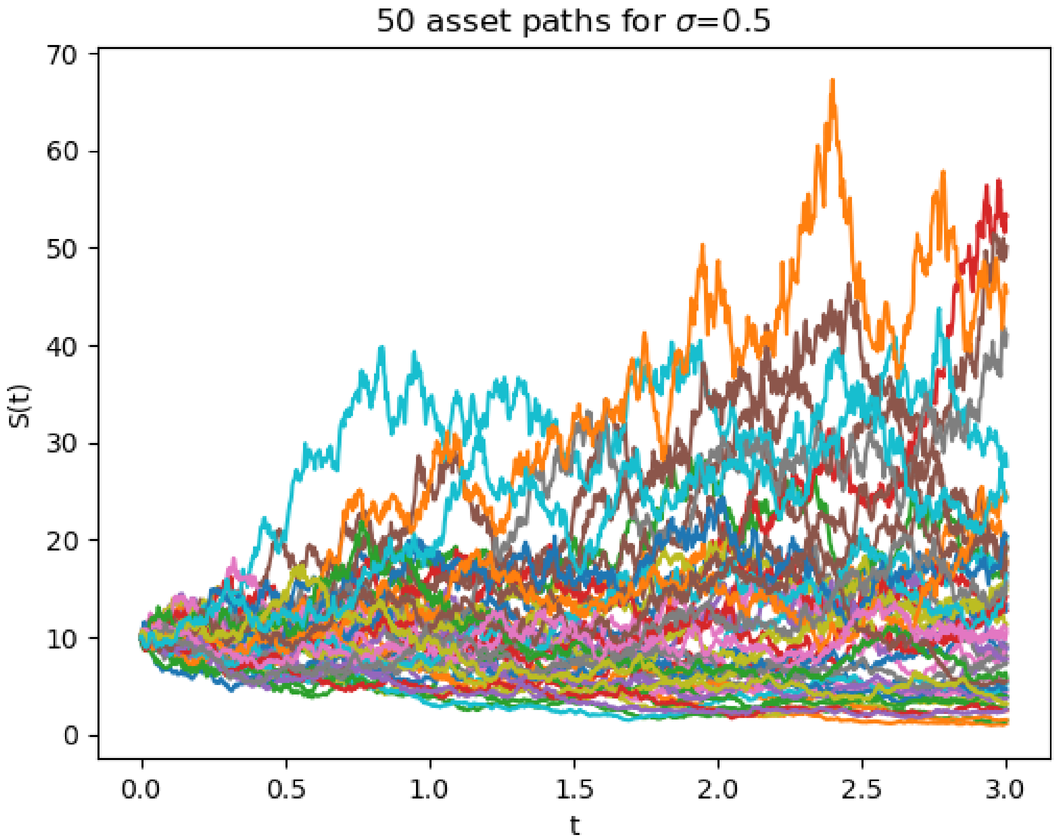


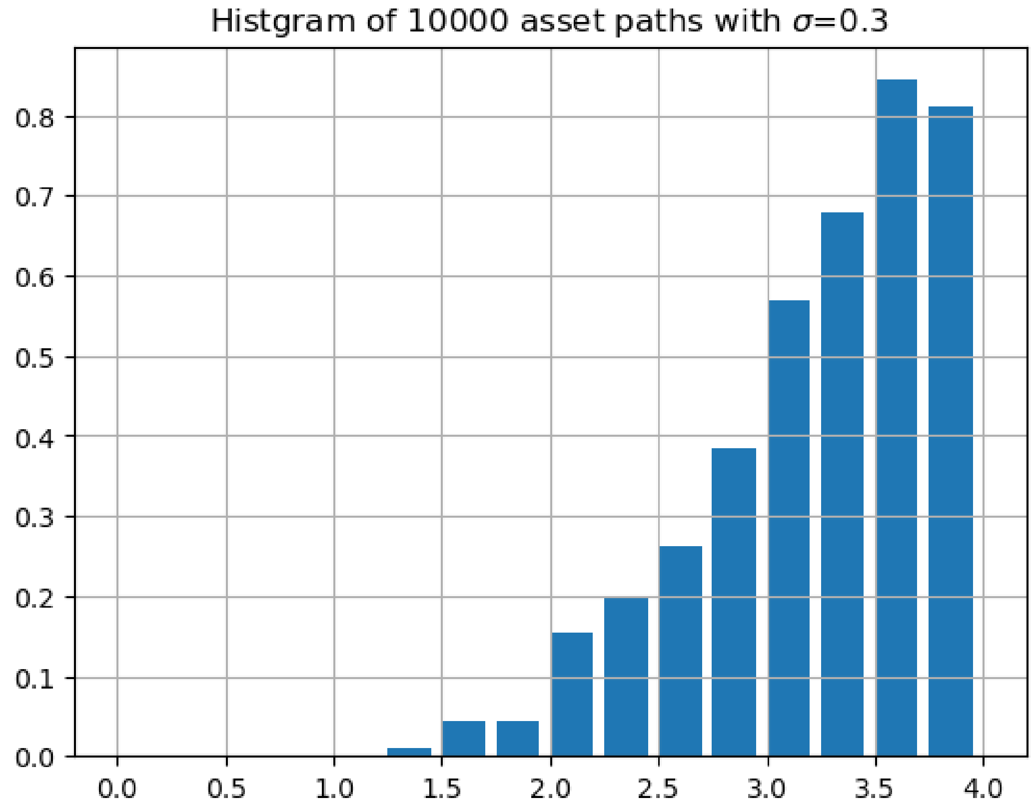
[29]:
```
# (c)

for sigma in sigmas:
    M = 50
    plt.figure()
    tvals = np.linspace(0,T,L+1)
    Svals = S*np.cumprod(np.exp((mu-0.5*sigma**2)*dt + sigma*np.sqrt(dt)*np.
 ↪random.randn(M,L)),axis=1)
    Svals = np.insert(Svals,0,S*np.ones(M),axis=1) # add initial asset price
    plt.plot(tvals,Svals.transpose())
    plt.title('50 asset paths for $\sigma$=%.1f' %sigma)
    plt.xlabel('t')
    plt.ylabel('S(t)')
    # plt.show()

for sigma in sigmas:
    plt.figure()
    M = 10000
    Svals4 = S*np.cumprod(np.exp((mu-0.5*sigma**2)*dt + sigma*np.sqrt(dt)*np.
 ↪random.randn(M,L)),axis=1)
```

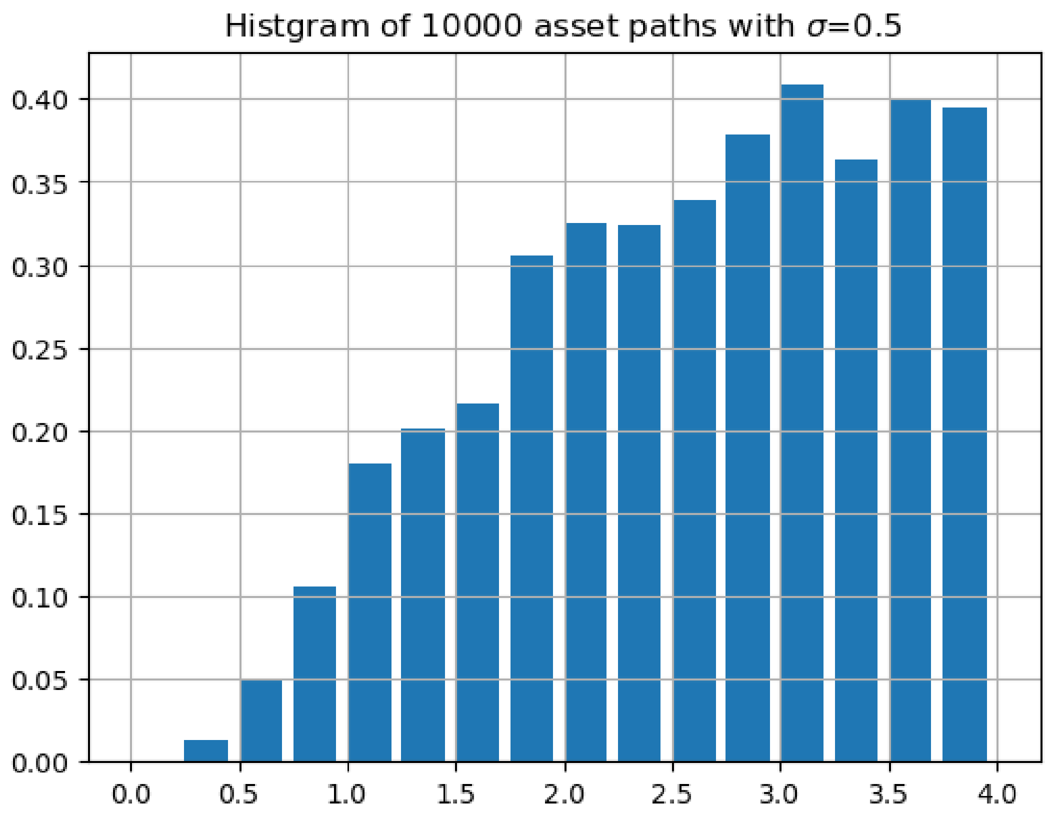
```
    Samples = Svals4[:,-1]
    centers = np.linspace(0, 4, 17)
    tempb = x * sigma * np.sqrt(2 * np.pi * t)
    tempa = ((np.log(x / S) - (mu - 0.5 * sigma ** 2) * t) ** 2) / (2 * t *␣
↪sigma ** 2)
    y = np.exp(-tempa) / tempb
    N = plt.hist(Samples, bins=centers, width=0.2, density=True, label='Sample␣
↪data')
    plt.plot(x, y, '-', label='$\sigma$ = %.1f' % sigma)
    plt.title('Histgram of %d asset paths with $\sigma$=%.1f' %(M,sigma))
    #plt.legend()
    plt.grid()
    plt.show()
```

```
/var/folders/bt/m5bfjjn55k785tq8l5t_gdlh0000gq/T/ipykernel_12508/1251824529.py:2
2: RuntimeWarning: divide by zero encountered in log
  tempa = ((np.log(x / S) - (mu - 0.5 * sigma ** 2) * t) ** 2) / (2 * t * sigma
** 2)
/var/folders/bt/m5bfjjn55k785tq8l5t_gdlh0000gq/T/ipykernel_12508/1251824529.py:2
2: RuntimeWarning: divide by zero encountered in divide
  tempa = ((np.log(x / S) - (mu - 0.5 * sigma ** 2) * t) ** 2) / (2 * t * sigma
** 2)
/var/folders/bt/m5bfjjn55k785tq8l5t_gdlh0000gq/T/ipykernel_12508/1251824529.py:2
3: RuntimeWarning: invalid value encountered in divide
  y = np.exp(-tempa) / tempb
```


50 asset paths for σ=0.3
S(t)
20
15
10
5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
t


50 asset paths for σ=0.5
S(t)
t
0
10
20
30
40
50
60
70
0.0
0.5
1.0
1.5
2.0
2.5
3.0


Histgram of 10000 asset paths with σ=0.3
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

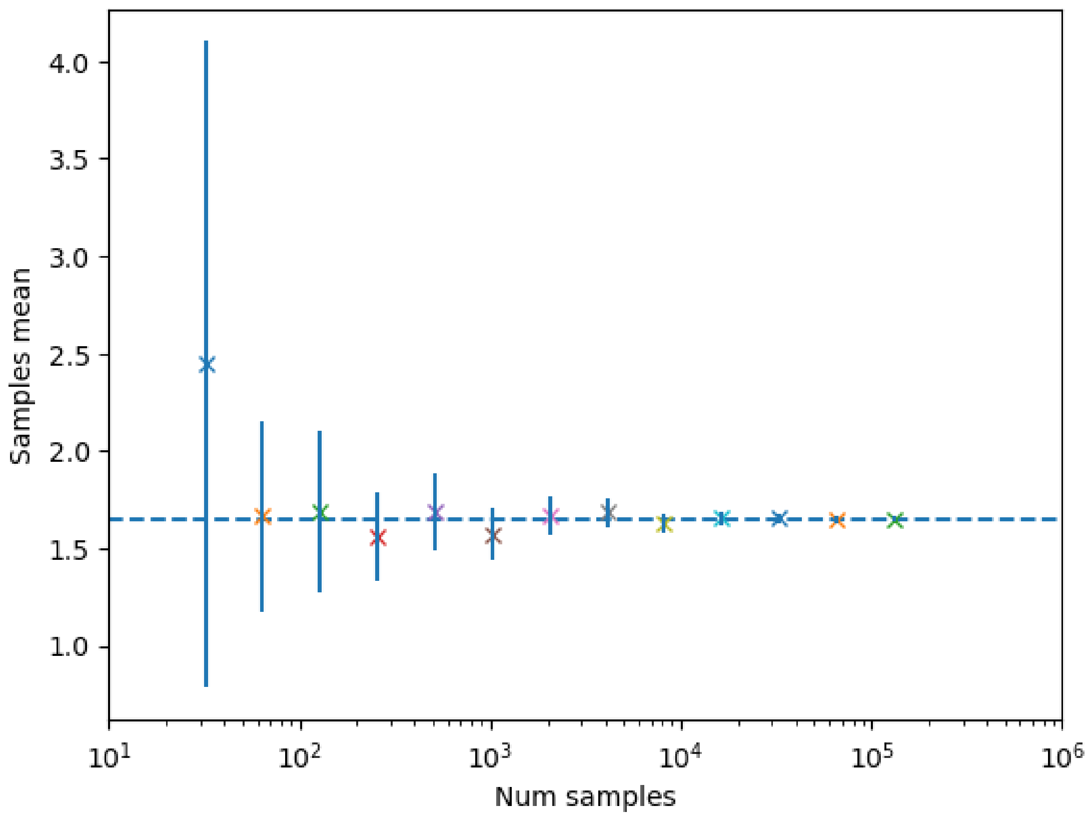


[9]:
```
import scipy
import numpy as np

# Standard Black-Scholes formula P2
def BS_call(t,S,r,sigma,K,T):
    tau = T-t
    if tau > 0:
        d1 = (np.log(S/K) + (r + 0.5*sigma**2)*(tau))/(sigma*np.sqrt(tau))
        d2 = d1 - sigma*np.sqrt(tau)
        N1 = 0.5*(1+scipy.special.erf(d1/np.sqrt(2)))
        N2 = 0.5*(1+scipy.special.erf(d2/np.sqrt(2)))
        C = S*N1-K*np.exp(-r*(tau))*N2
    else:
        C = max(S-K,0)
    return(C)
res = BS_call(0,10,0.06,0.1,9,1)
res1=round(res,4)
print(res1)
```

```
1.5429
```

[5]:

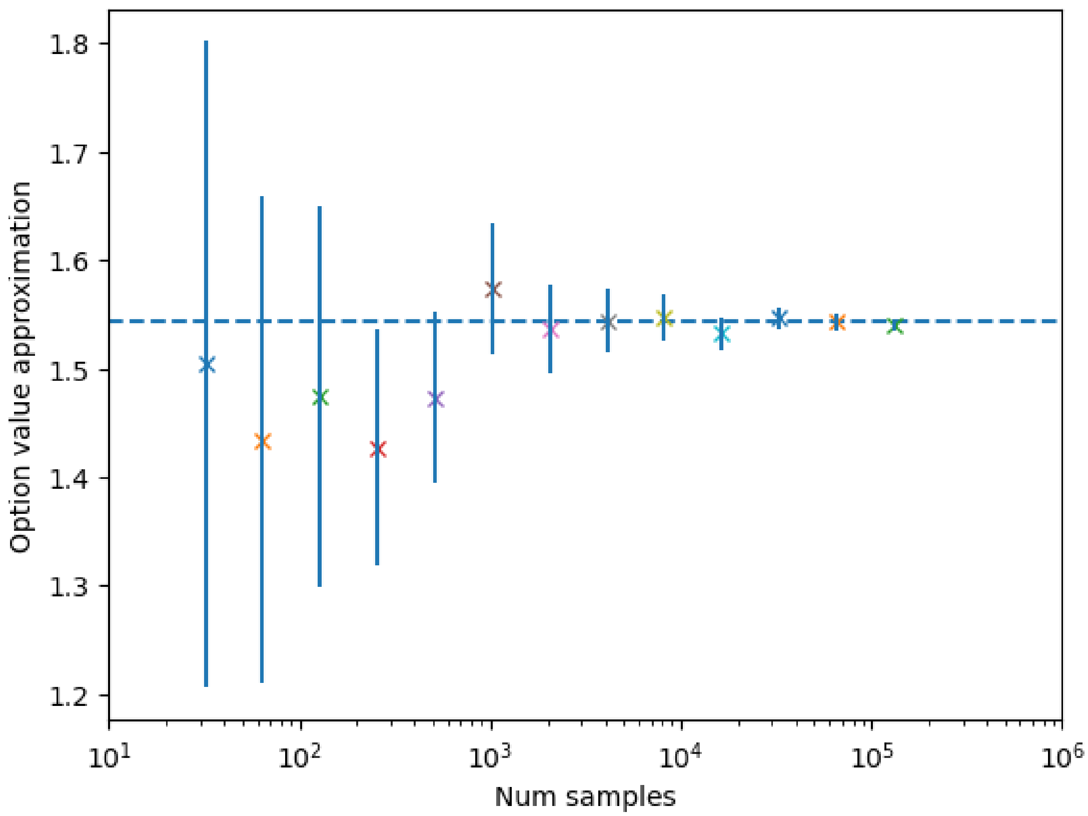
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm
from scipy.special import erfinv, erf
# For P3
#
# Monte Carlo for e^Z

### Problem and method parameters ###

conf_level = 0.97
c_p = 1-(1-conf_level)/2
criterion = np.sqrt(np.exp(1))

# The result may vary as the random numbers generated by "np.random.randn()".
# One could fixed it by the choosing a random seed: (similar to randn('state',␣
↪100) as in Matlab)
# np.random.seed(567)

plt.xscale('log')
plt.hlines(criterion,10,1000000,linestyles='dashed')
plt.xlim([10,1000000])
for k in range(13):
    M = 2**(k+5)
    V = np.zeros(M)
    for i in range(M):
        V[i] = np.exp(np.random.randn())
    aM = np.mean(V)
    bM = np.std(V)
    #c = norm.ppf(c_p)
    c=np.sqrt(2)*erfinv(2*c_p-1)
    conf_lb = aM - c*bM/np.sqrt(M)
    conf_ub = aM + c*bM/np.sqrt(M)
    plt.plot(M,aM,'x')
    plt.vlines(M,conf_lb,conf_ub)
    plt.xlabel('Num samples')
    plt.ylabel('Samples mean')
    # print('The exact value of E(e^Z) is',np.sqrt(np.exp(1)))
    # print('The Monte Carlo scheme give %.d%% confident interval with %.d␣
↪samples:'%(100*conf_level,M),conf)
plt.show()
```



[11]:
```
import numpy as np
from scipy.stats import norm
# import p2
import matplotlib.pyplot as plt

#P4 Standard Monte Carlo to price European call option
t = 0
S = 10
K = 9
sigma = 0.1
r = 0.06
T = 1
Dt = 1e-3
N = T/Dt
M = 10000


V = np.zeros(M)
for i in range(M):
    Sfinal = S*np.exp((r-0.5*sigma**2)*T+sigma*np.sqrt(T)*np.random.randn())
    V[i] = np.exp(-r*T)*max(Sfinal-K,0)
```

```
aM = np.mean(V)
bM = np.std(V)
conf = [aM - 1.96*bM/np.sqrt(M), aM + 1.96*bM/np.sqrt(M)]
print(conf)
res = BS_call(t,S,r,sigma,K,T)
plt.xscale('log')
plt.hlines(res,10,1000000,linestyles='dashed')
plt.xlim([10,1000000])
for k in range(13):
    M = 2**(k+5)
    V = np.zeros(M)
    for i in range(M):
        Sfinal = S * np.exp((r - 0.5 * sigma ** 2) * T + sigma * np.sqrt(T) *␣
 ↪np.random.randn())
        V[i] = np.exp(-r * T) * max(Sfinal - K, 0)
    aM = np.mean(V)
    bM = np.std(V)
    conf_lb = aM - 1.96*bM/np.sqrt(M)
    conf_ub = aM + 1.96*bM/np.sqrt(M)
    plt.plot(M,aM,'x')
    plt.vlines(M,conf_lb,conf_ub)
    plt.xlabel('Num samples')
    plt.ylabel('Option value approximation')
plt.show()
```

```
[1.520845484496328, 1.558961485577926]
```



[49]:
```python
import numpy as np
import scipy

#P5 up-and-in-European call option by standard Monte Carlo
S = 5
K = 6
sigma = 0.3
r = 0.05
T = 1
B = 8

N= 10000
Dt = T/N
M= 1000


V = np.zeros(M)
for i in range(M):
    Svals = S*np.cumprod([np.exp((r-0.5*sigma**2)*Dt+sigma*np.sqrt(Dt)*np.
↪random.randn()) for i in range(N)]);
    Smax = max(Svals)
```

```
    if Smax > B:
        V[i] = np.exp(-r*T)*max(Svals[-1]-K,0)

aM = np.mean(V)
bM = np.std(V)
conf1 = [aM - 1.96*bM/np.sqrt(M), aM + 1.96*bM/np.sqrt(M)]
print(conf1)
```

```
[0.18680677576336074, 0.28783659134192896]
```

[50]:

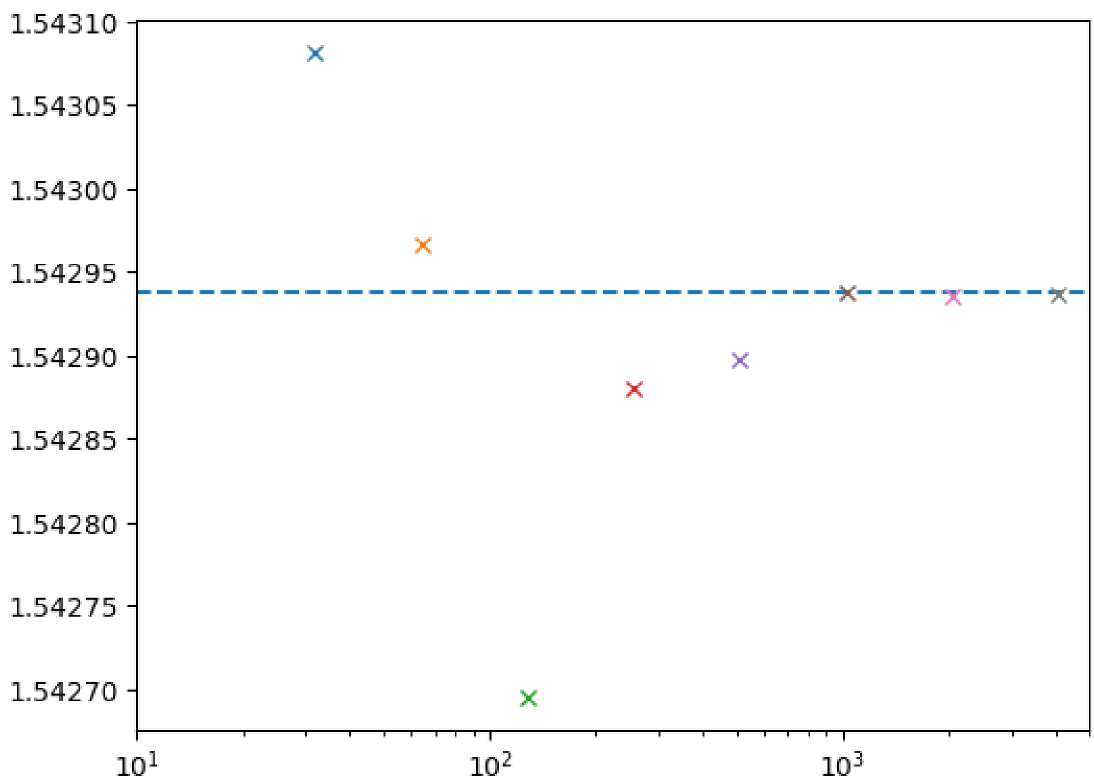
```
import numpy as np
import scipy

#P5 up-and-in-European call option by antithetic variate
S = 5
K = 6
sigma = 0.3
r = 0.05
T = 1
B = 8

N= 10000
Dt = T/N
M= 1000


V = np.zeros(M)
V1 = np.zeros(M)
V2 = np.zeros(M)
for i in range(M):
    samples = np.random.randn(N)
    Svals = S*np.cumprod([np.exp((r-0.5*sigma**2)*Dt+sigma*np.sqrt(Dt)*sam) for␣
 ↪sam in samples])
    Smax = max(Svals)
    if Smax > B:
        V[i] = np.exp(-r*T)*max(Svals[-1]-K,0)
    Svals1 = S*np.cumprod([np.exp((r-0.5*sigma**2)*Dt-sigma*np.sqrt(Dt)*sam)␣
 ↪for sam in samples])
    Smax1 = max(Svals1)
    if Smax1 > B:
        V1[i] = np.exp(-r*T)*max(Svals1[-1]-K,0)
for i in range(M):
    V2[i]=0.5*V[i]+0.5*V1[i]
aM = np.mean(V2)
bM = np.std(V2)
conf = [aM - 1.96*bM/np.sqrt(M), aM + 1.96*bM/np.sqrt(M)]
print(conf)
```

```
[0.21916754760096424, 0.28703238845479884]
```

[55]:
```
import numpy as np
import matplotlib.pyplot as plt
# import p2

#P6 Binomial method for European call option
t = 0
S = 10
K = 9
sigma = 0.1
r = 0.06
T = 1

M = 400
dt = T/M
p = 0.5
u = np.exp(sigma*np.sqrt(dt) + (r-0.5*sigma**2)*dt)
d = np.exp(-sigma*np.sqrt(dt) + (r-0.5*sigma**2)*dt)

# Time T option values
W = [max(S*d**(M-i)*u**(i)-K,0) for i in range(0,M+1)]
# Work back to option value at time zero
for i in range(M,0,-1):
    W = [np.exp(-r*dt)*(p*a + (1-p)*b) for (a,b) in zip(W[0:-1],W[1:])]
print('Option value is',W)

# Convergence to the B-S values as M increasing
res = BS_call(t,S,r,sigma,K,T)
print('The result from B-S formula is', res)
plt.xscale('log')
plt.hlines(res,10,5000,linestyles='dashed')
plt.xlim([10,5000])
for k in range(8):
    M = 2**(k+5)
    dt = T / M
    u = np.exp(sigma * np.sqrt(dt) + (r - 0.5 * sigma ** 2) * dt)
    d = np.exp(-sigma * np.sqrt(dt) + (r - 0.5 * sigma ** 2) * dt)
    W = [max(S * d ** (M - i) * u ** (i) - K, 0) for i in range(0, M + 1)]
    for i in range(M, 0, -1):
        W = [np.exp(-r * dt) * (p * a + (1 - p) * b) for (a, b) in zip(W[0:-1],
 ↪W[1:])]
    print('Option value is', W)
    plt.plot(M,W,'x')
plt.show()
```

```
Option value is [1.5428470397443415]
The result from B-S formula is 1.5429374445144521
```

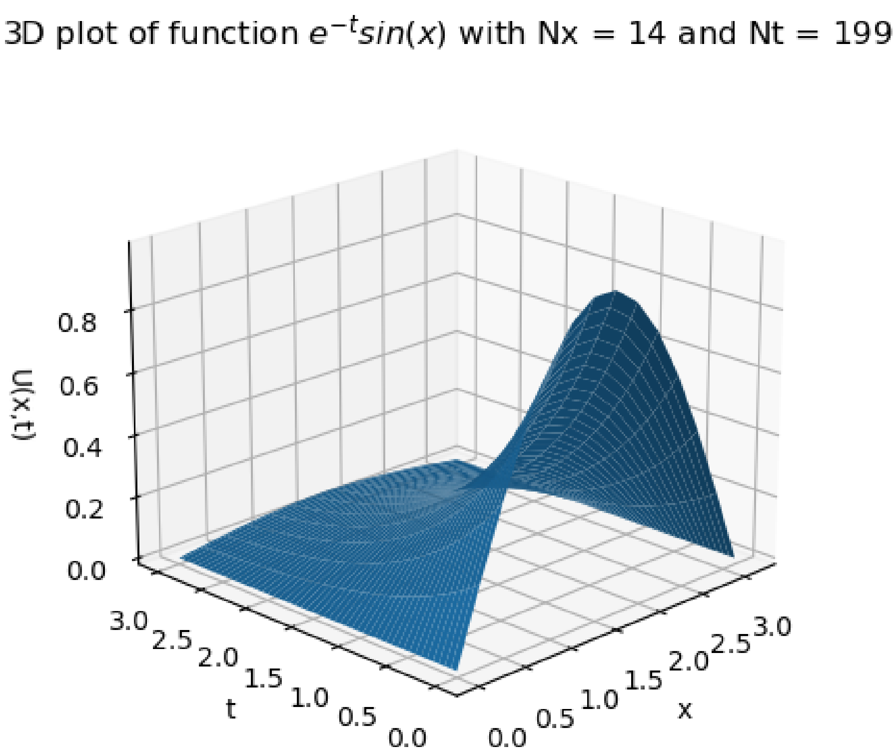
```
Option value is [1.5430817586966676]
Option value is [1.5429667165057488]
Option value is [1.5426952559493257]
Option value is [1.5428804226390112]
Option value is [1.542897915635234]
Option value is [1.5429376713992933]
Option value is [1.5429352090505495]
Option value is [1.5429369068685188]
```

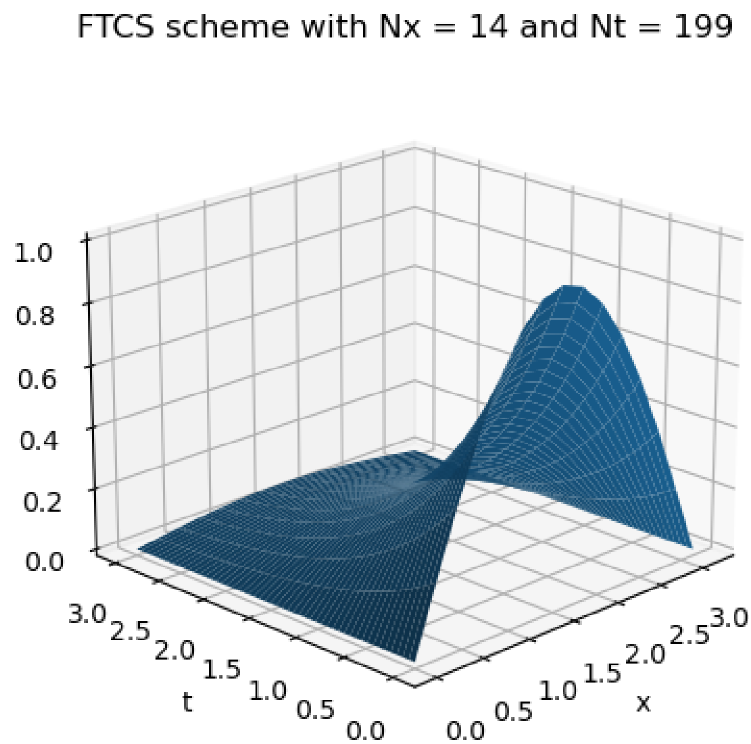


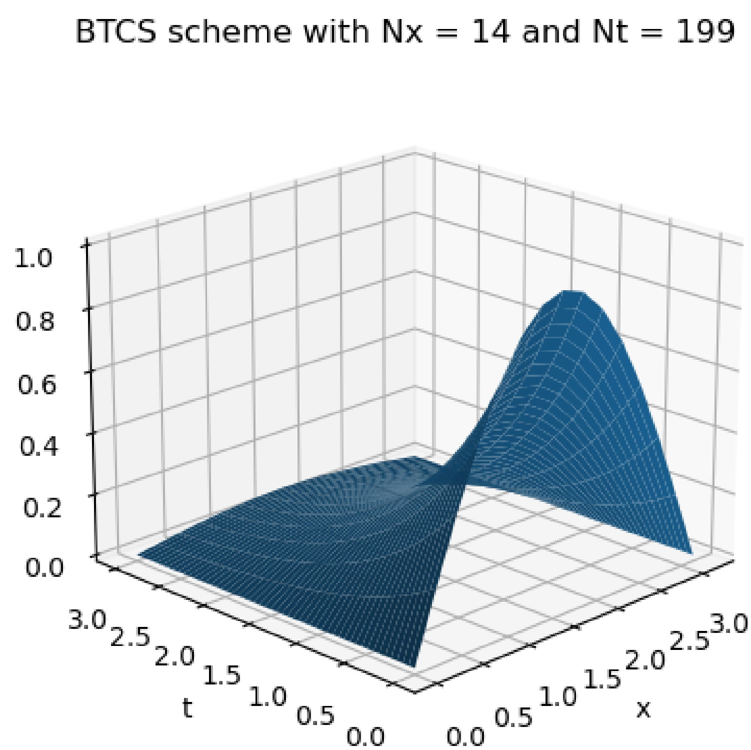
[23]:
```
import numpy as np
import matplotlib.pyplot as plt

#P7 Finite difference method for heat equation

L = np.pi
Nx = 14
T = 3
Nt = 199
# (a)
xvals = np.linspace(0,L,Nx)
tvals = np.linspace(0,T,Nt)
xmat,tmat = np.meshgrid(xvals,tvals)
```

```
C = np.exp(-tmat)*np.sin(xmat)
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
ax.plot_surface(xmat,tmat,C)
ax.view_init(elev=20, azim=-135, roll=0)
plt.title('3D plot of function $e^{-t}sin(x)$ with Nx = %d and Nt = %d'␣
  ↪%(Nx,Nt))
plt.ylabel('t')
plt.xlabel('x')
ax.set_zlabel('U(x,t)')
plt.show()

# (b)
def ftcs(L,Nx,T,Nt):
    dt = T / Nt
    dx = L / Nx
    nu = dt / dx ** 2
    F = (1-2*nu)*np.eye(Nx-1,Nx-1) + nu*np.diag(np.ones(Nx-2),1) + nu*np.
  ↪diag(np.ones(Nx-2),-1)
    U = np.zeros((Nx-1,Nt+1))
    U[:,0] = np.sin(np.arange(dx,L,dx))
    for i in range(Nt):
        x = np.dot(F,U[:,i])
        U[:,i+1] = x
    bc = np.zeros(Nt+1)
    U =np.r_[[bc],U,[bc]]
    t = np.linspace(0,T,Nt+1)
    x = np.linspace(0,L,Nx+1)
    t,x = np.meshgrid(t,x)
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    ax.plot_surface(x,t,U)
    ax.view_init(elev=20, azim=-135, roll=0)
    plt.title('FTCS scheme with Nx = %d and Nt = %d' %(Nx,Nt))
    plt.xlabel('x')
    plt.ylabel('t')
    plt.show()

def btcs(L,Nx,T,Nt):
    dt = T / Nt
    dx = L / Nx
    nu = dt / dx ** 2
    B = (1 + 2 * nu) * np.eye(Nx - 1, Nx - 1) - nu * np.diag(np.ones(Nx - 2),␣
  ↪1) - nu * np.diag(np.ones(Nx - 2), -1)
    U = np.zeros((Nx - 1, Nt + 1))
    U[:, 0] = np.sin(np.arange(dx, L, dx))
    for i in range(Nt):
        x = np.dot(np.linalg.pinv(B), U[:, i])
        U[:, i + 1] = x
```

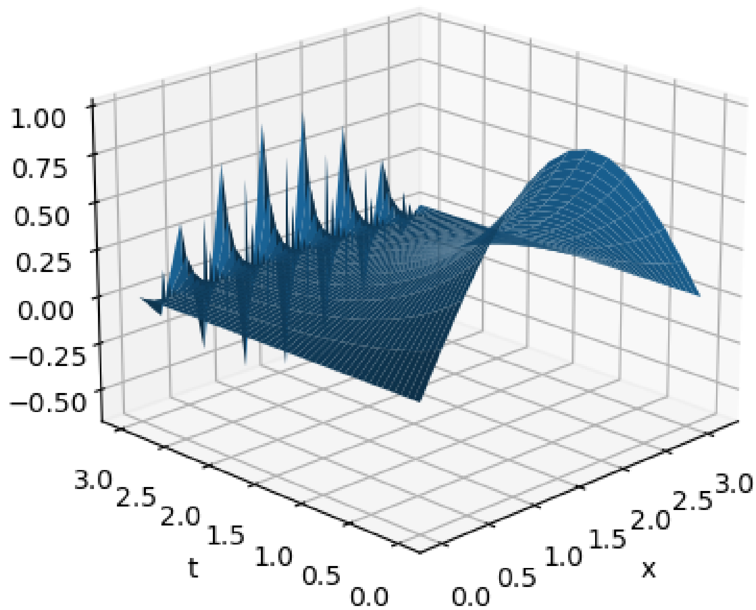
```python
    bc = np.zeros(Nt + 1)
    U = np.r_[[bc], U, [bc]]
    t = np.linspace(0, T, Nt + 1)
    x = np.linspace(0, L, Nx + 1)
    t, x = np.meshgrid(t, x)
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    ax.plot_surface(x, t, U)
    ax.view_init(elev=20, azim=-135, roll=0)
    plt.title('BTCS scheme with Nx = %d and Nt = %d' % (Nx, Nt))
    plt.xlabel('x')
    plt.ylabel('t')
    plt.show()

ftcs(L,Nx,T,Nt)
btcs(L,Nx,T,Nt)

# (c)
ftcs(L,Nx,T,94)
btcs(L,Nx,T,94)
```

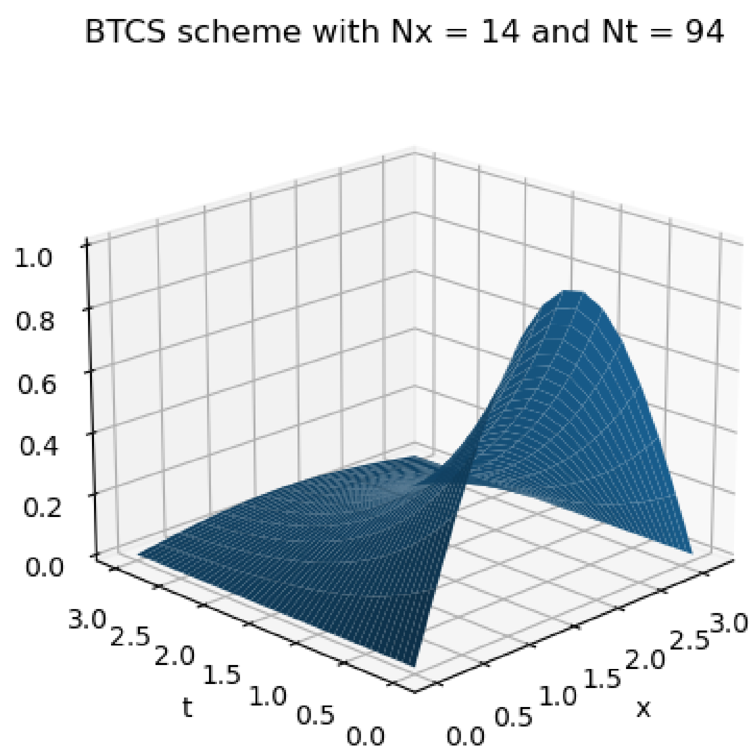



FTCS scheme with Nx = 14 and Nt = 199
1.0
0.8
0.6
0.4
0.2
0.0
3.0
2.5
2.0
1.5
1.0
0.5
0.0
t
0.0
0.5
1.0
1.5
2.0
2.5
3.0
x


BTCS scheme with Nx = 14 and Nt = 199
1.0
0.8
0.6
0.4
0.2
0.0
3.0
2.5
2.0
1.5
1.0
0.5
0.0
t
0.0
0.5
1.0
1.5
2.0
2.5
3.0
x

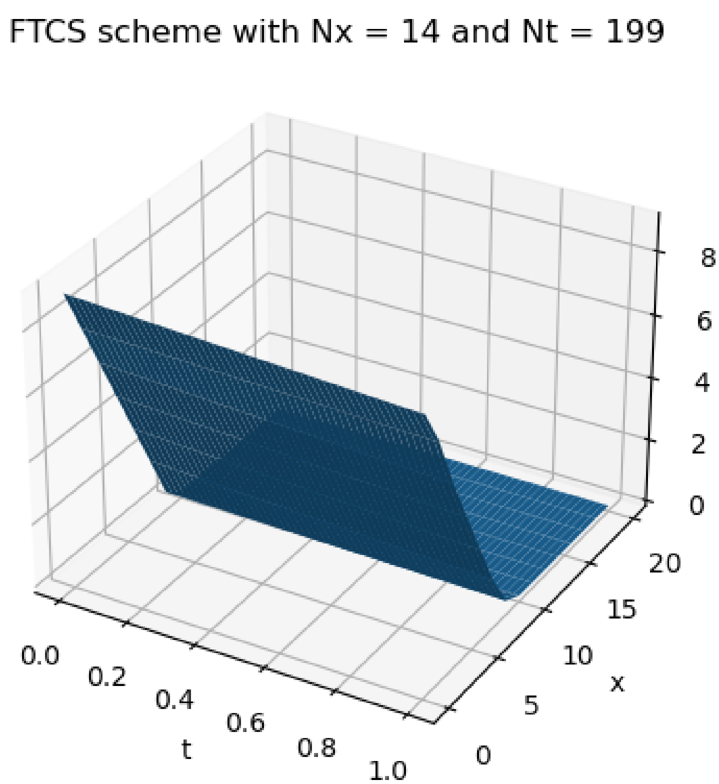
FTCS scheme with Nx = 14 and Nt = 94






```
[19]: import numpy as np
import matplotlib.pyplot as plt

#P8 Finite difference method for Black-Scholes PDE
t = 0
L = 20
K = 9
sigma = 0.1
r = 0.06
T = 1
Nt = 199
Nx = 14

k = T / Nt
h = L / Nx
T1 = np.diag(np.ones(Nx - 2), 1) - np.diag(np.ones(Nx - 2), -1)
T2 = -2 * np.eye(Nx - 1, Nx - 1) + np.diag(np.ones(Nx - 2), 1) + np.diag(np.
 ↪ones(Nx - 2), -1)
mvec = np.arange(1, Nx, 1)
D1 = np.diag(mvec)
D2 = np.diag(mvec ** 2)
```

```
F = (1 - r * k) * np.eye(Nx - 1, Nx - 1) + 0.5 * k * sigma ** 2 * np.dot(D2,␣
↪T2) + 0.5 * k * r * np.dot(D1, T1)
U = np.zeros((Nx-1,Nt+1))
U[:,0] = [max(K-i,0) for i in np.arange(h,L,h)]
for i in range(Nt):
    x = np.dot(F,U[:,i])
    U[:,i+1] = x
bca = K*np.exp(-r*np.arange(0,T+k,k))
bcb = np.zeros(Nt+1)
U = np.r_[[bca],U,[bcb]]
x = np.linspace(0,T,Nt+1)
y = np.linspace(0,L,Nx+1)
x,y = np.meshgrid(x,y)
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
ax.plot_surface(x,y,U)
plt.title('FTCS scheme with Nx = %d and Nt = %d' %(Nx,Nt))
plt.xlabel('t')
plt.ylabel('x')
plt.show()
```




```
[ ]:
```