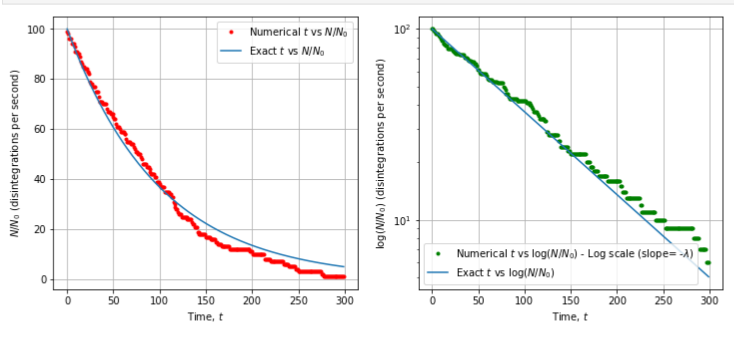
Assignment on Monte-Carlo techniques

Question 1

Part (a)

$N_0 = 100$, $\alpha = 0.01\ s^{-1}$, $\Delta t = 1$ sec

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

#part (a)
#N_0 = 100, a = 0.01 s^{-1}, \Delta t = 1 sec

alph = 0.01     # it is the decay constant
p = alph        # decay probablity for delta t =1
q = 1-p         # survival probablity
N_ini = 100
T = 300


def decay(N):
    population = []
    for t in range(T):
        r = np.random.random(N)
        survive = np.sum(r<q)          # number of atoms survived
        population.append(survive)
        N = survive
    return population


plt.figure(figsize =(12,5))



# Plots


plt.subplot(1,2,1)
plt.plot(range(T),decay(N_ini),'r.', label = 'Numerical $t$ vs $N/N_0$')

analyticals = []
for i in range(T):
    analyticals.append(N_ini*np.exp(-alph*i))
plt.plot(range(T),analyticals,label = 'Exact $t$ vs $N/N_0$' )

plt.xlabel('Time, $t$')
plt.ylabel('$N/N_0$ (disintegrations per second)')
plt.legend()
plt.grid()

plt.subplot(1,2,2)

plt.semilogy()

# Linear
plt.plot(range(T),decay(N_ini),'g.',label = 'Numerical $t$ vs $\log(N/N_0)$ - Log scale (slope= -$\lambda$)')
plt.semilogy()
plt.plot(range(T),analyticals,label = 'Exact $t$ vs $\log(N/N_0)$' )
plt.xlabel('Time, $t$')
plt.ylabel('$\log(N/N_0)$ (disintegrations per second)')
plt.legend()
plt.grid()
```

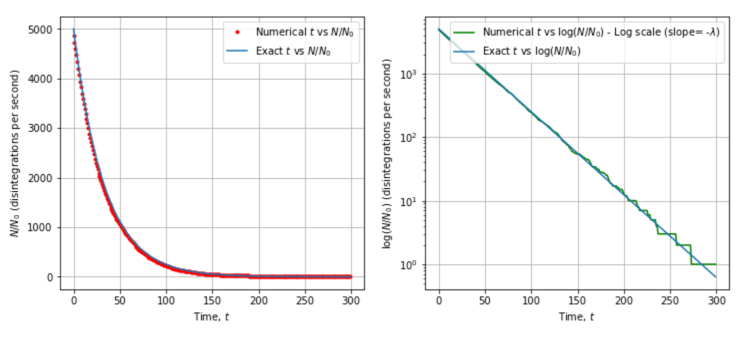


Part (b)

$N_0 = 5000$, $\alpha = 0.03\ s^{-1}$, $\Delta t = 1$ sec

In [ ]:
```python
#part (b)
#N_0 = 5000, a = 0.03 s^{-1}, \Delta_t = 1 sec

alpha2 = 0.03
p2 = alpha2          # decay probablity for delta t =1
q2 = 1-p2      # survival probablity
N_ini =5000
T=300

##########

def decay(N):
    population = []
    for t in range(T):
        r = np.random.random(N)
        survive = np.sum(r<q2)          # number of atoms survived
        population.append(survive)
        N = survive
    return population



plt.figure(figsize =(12,5))

plt.subplot(1,2,1)
plt.plot(range(T),decay(N_ini),'r.',label = 'Numerical $t$ vs $N/N_0$')

analyticals = []
for i in range(T):
    analyticals.append(N_ini*np.exp(-alpha2*i))
plt.plot(range(T),analyticals,label = 'Exact $t$ vs $N/N_0$' )

plt.xlabel('Time, $t$')
plt.ylabel('$N/N_0$ (disintegrations per second)')
plt.legend()
plt.grid()

plt.subplot(1,2,2)

plt.semilogy()


# Linear
plt.plot(range(T),decay(N_ini),'g-',label = 'Numerical $t$ vs $\log(N/N_0)$ - Log scale (slope= -$\lambda$)')
plt.semilogy()
plt.plot(range(T),analyticals,label = 'Exact $t$ vs $\log(N/N_0)$' )
plt.grid()
plt.xlabel('Time, $t$')
plt.ylabel('$\log(N/N_0)$ (disintegrations per second)')
plt.legend()
```

Out[ ]: `<matplotlib.legend.Legend at 0x2078c920880>`



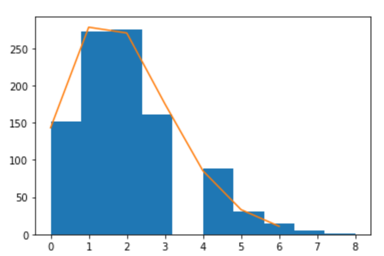
Question 2

Part (a)

$N_0 = 500$, $\alpha = 4 \times 10^{-5}\ s^{-1}$, $\Delta t = 10$ sec, $T = 100$ sec

In [ ]:
```python
distri = []
N = 500
def decays(N):
    alphas = 4*10**(-5)   # it is the decay constant
    delta_t = 10
    p = delta_t*alphas           # decay probablity for delta t =1
    q = 1-p      # survival probablity

    T=100
    jump = int(T/delta_t)
    populations = []
    for t in range(jump):
        r = np.random.random(N)
        survive = np.sum(r<q)          # number of atoms survived
        populations.append(survive)
        N = survive
    return populations




for i in range(1000):
    p = decays(N)
    distri.append(N-p[-1])


plt.hist(distri)
avg_distri = np.mean(distri)
from scipy.stats import poisson
x = np.arange(0,7,1)
y = 1000*poisson.pmf(x, mu =avg_distri)
plt.plot(x,y)
```

Out[ ]: `[<matplotlib.lines.Line2D at 0x2078e596e50>]`

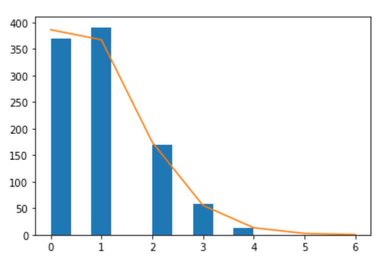


Part (b)

$N_0 = 500$, $\alpha = 2 \times 10^{-5}\ s^{-1}$, $\Delta t = 10$ sec, $T = 100$ sec

In [ ]:
```python
distri = []

def decays(N):
    alphas = 2*10**(-5)   # it is the decay constant
    delta_t = 10
    p = delta_t*alphas           # decay probablity for delta t =1
    q = 1-p      # survival probablity

    T=100
    jump = int(T/delta_t)
    populations = []
    for t in range(jump):
        r = np.random.random(N)
        survive = np.sum(r<q)          # number of atoms survived
        populations.append(survive)
        N = survive
    return populations




for i in range(1000):
    p = decays(N)
    distri.append(N-p[-1])




plt.hist(distri)
avg_distri = np.mean(distri)
from scipy.stats import poisson
x = np.arange(0,7,1)
y = 1000*poisson.pmf(x, mu =avg_distri)
plt.plot(x,y)
```

Out[ ]: `[<matplotlib.lines.Line2D at 0x2078e60a700>]`



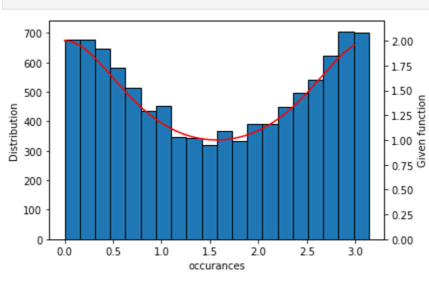
Question 3

Part (a) for $a = 0.5$

In [ ]:
```python
def DistFunction(u):
    return(np.arctan(np.sqrt(0.5)*np.tan((u-1)*np.pi)))

DistFunction(0)
#DistFunction(1)
N=np.random.rand(10000)
#print(Y)
def Func1(th):
    return(1/(np.sin(th)**2+0.5*np.cos(th)**2))
```

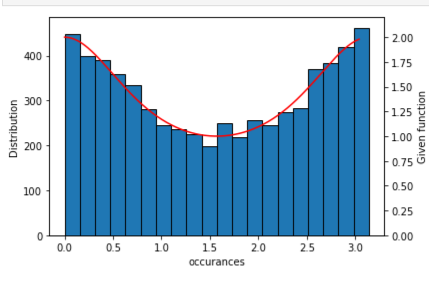
By Inversion technique

In [ ]:
```python
fig,ax1=plt.subplots()
Y=DistFunction(np.random.rand(10000))
for i in range(10000):
    if Y[i]<0:
        Y[i]=Y[i]+np.pi
#print(Y)
X=np.arange(0,3,0.01)
Curve=Func1(X)
#print(Curve)
#plt.plot(X,PD,"o-","red")
ax1.hist(Y,edgecolor='black',bins=20)

ax1.set_xlabel('occurances')
ax1.set_ylabel('Distribution')
ax2=ax1.twinx()
ax2.plot(X,Curve,linestyle='-',color='red')
ax2.set_ylim(0,2.2)
ax2.set_ylabel('Given function')
    #plt.grid()
plt.show()
```



By Acception-Rejection technique

In [ ]:
```python
U1= np.random.rand(10000)
U2= np.random.rand(10000)
Y=[]
for i in range(10000):
    if Func1(np.pi*U1[i])>=U2[i]*2.25:
        Y.append(np.pi*U1[i])
fig,ax1=plt.subplots()
ax1.hist(Y,edgecolor='black',bins=20)
ax1.set_xlabel('occurances')
ax1.set_ylabel('Distribution')
ax2=ax1.twinx()
X=np.arange(0,3.05,0.01)
Curve=Func1(X)
ax2.plot(X,Curve,linestyle='-',color='red')
ax2.set_ylim(0,2.2)
ax2.set_ylabel('Given function')
plt.show()
```

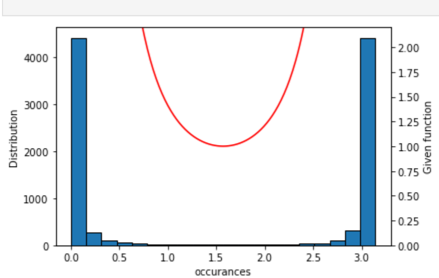


Part (b) for $a = 0.001$

In [ ]:
```python
def DistFunction2(u):
    return(np.arctan(np.sqrt(0.001)*np.tan((u-1)*np.pi)))

DistFunction2(0)
#DistFunction(1)
N=np.random.rand(10000)
#print(Y)
def Func2(th):
    return(1/(np.sin(th)**2+0.001*np.cos(th)**2))
```

By Inversion technique

In [ ]:
```python
fig,ax1=plt.subplots()
Y=DistFunction2(np.random.rand(10000))
for i in range(10000):
    if Y[i]<0:
        Y[i]=Y[i]+np.pi
#print(Y)
X=np.arange(0,3,0.01)
Curve=Func2(X)
#print(Curve)
#plt.plot(X,PD,"o-","red")
ax1.hist(Y,edgecolor='black',bins=20)

ax1.set_xlabel('occurances')
ax1.set_ylabel('Distribution')
ax2=ax1.twinx()
ax2.plot(X,Curve,linestyle='-',color='red')
ax2.set_ylim(0,2.2)
ax2.set_ylabel('Given function')
    #plt.grid()
plt.show()
```



By Acception-Rejection technique

In [ ]:
```python
U1= np.random.rand(10000)
U2= np.random.rand(10000)
Y=[]
for i in range(10000):
    if Func2(np.pi*U1[i])>=U2[i]*2.25:
        Y.append(np.pi*U1[i])
fig,ax1=plt.subplots()
ax1.hist(Y,edgecolor='black',bins=20)
ax1.set_xlabel('occurances')
ax1.set_ylabel('Distribution')
ax2=ax1.twinx()
X=np.arange(0,3.05,0.01)
Curve=Func2(X)
ax2.plot(X,Curve,linestyle='-',color='red')
ax2.set_ylim(0,2.2)
ax2.set_ylabel('Given function')
plt.show()
```