In [26]:

```
import numpy as np
import matplotlib.pyplot as plt
from statistics import mean
from statistics import variance
import scipy.integrate as sc
```

In [36]:

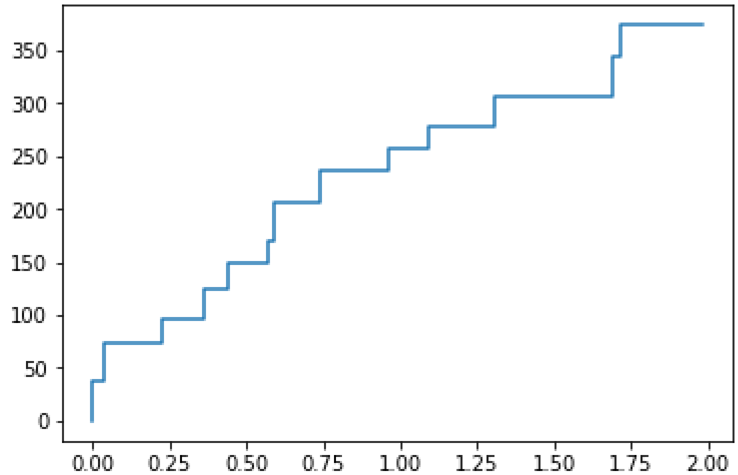
```
ret=np.zeros(50)
reti=np.zeros(50000)
n=50
w=200
b=0
for J in range(0,50000):
    ret=np.zeros(50)
    n=50
    w=200
    b=0
    for i in range(len(ret)):
        if i==0:
            ret[i]=np.random.uniform(0,2*w/n)
        if i>=1 and i <=48:
            b=(2/(50-i))*(200-np.sum(ret))

            ret[i]=np.random.uniform(0,b)


        if i==49:
            ret[i]=w-np.sum(ret)
    reti[J]=ret[2]

print(reti.mean())
print(variance(reti))
```

```
4.003091605611965
5.333268618594164
```

In [8]:

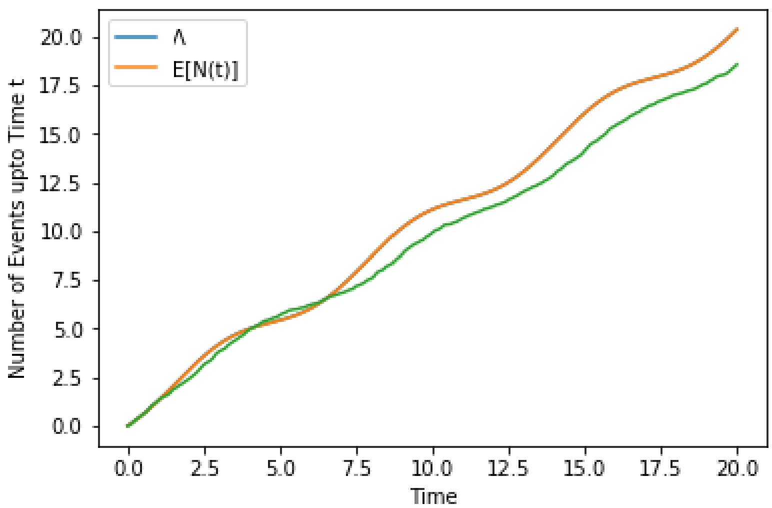
```
#2
rate = 5
t = [0]
n = [0]
while ( t[-1] < 2 ):
 next_t = t[-1] + (-1/rate)*np.log(np.random.uniform())
 t.append(next_t)
 next_batch = n[-1] + np.random.randint(20,40)
 n.append(next_batch)
```

In [9]:
```
plt.step(t[:-1], n[:-1])
plt.show()
```



In [41]:
```
import matplotlib.pyplot as plt
import numpy as np
import scipy.integrate as sc
def NHPPa(T):#naive algorithm
    t=0
    S=[]
    while(t<=T):
        S.append(t)# appending all values of t including 0
        l=1+0.6*np.sin(t)
        U=np.random.uniform(0,1)
        t=t-np.log(U)/l
    S.pop(0)
    return(S)

def NHPPb(T):
    t=0
    S=[]
    while(t<=T):
        p=(1+0.6*np.sin(t))/1.6#p function with 1.6 majorising lambda
        U1=np.random.uniform(0,1)
        t=t-np.log(U1)/(1.6)
        U2=np.random.uniform(0,1)
        if U2<=p:
            S.append(t)
    S.pop()

    return(S)

Exp=[]
I=[]
Inte=[]
f= lambda x: 1+0.6*np.sin(x)#function
t=np.linspace(0,20,201)#small finite breaks
for i in t:
    I.append(sc.quad(f,0,i))#Integral to calculate Λ for plotting
Inte.append([x[0] for x in I])
Inte=Inte[0]
#Values of Integral over the time horizon bin size =1
plt.plot(t,Inte)# Integral over 0 to 20
# 100 iterations of naive algorithm
for j in range (0,100):
    x=NHPPa(20)
    Ex=[]
    for m in t:
        Ex.append(sum(i<=m for i in x))
    Exp.append(Ex)
testa=np.array(Exp)
#E[N(t)] and Var[N(t)] for naive
np.mean(testa,axis=0)# showing values for bigger bins to save space bin size=1

np.var(testa,axis=0)

#plotting naive E[N(t)] against Λ with bin size=0.1
plt.plot(t,Inte)
plt.plot(t,np.mean(testa,axis=0))
plt.xlabel('Time')
```

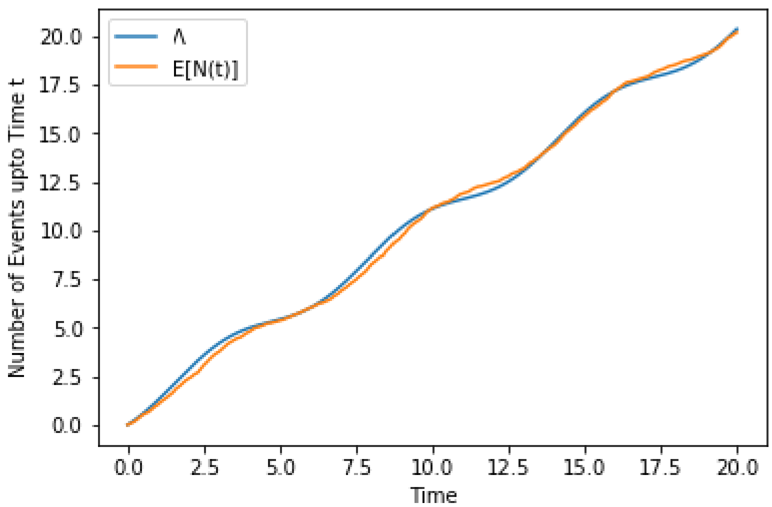
```
plt.ylabel('Number of Events upto Time t')
plt.legend(('Λ','E[N(t)]'))
plt.show()


Exp=[]
#100 interations for thinning algorithm
for j in range (0,100):
    x=NHPPb(20)
    Ex=[]
    for m in t:
        Ex.append(sum(i<=m for i in x))
    Exp.append(Ex)
testb=np.array(Exp)
#E[N(t)] and Var[N(t)] for naive
np.mean(testb,axis=0)# showing values for bigger bins to save space bin size=1

np.var(testb,axis=0)
```




```
Out[41]: array([ 0.    ,  0.0979,  0.2275,  0.31  ,  0.3704,  0.5291,  0.6051,
                 0.7011,  0.7624,  0.8276,  1.0196,  1.2936,  1.39  ,  1.5379,
                 1.7451,  1.8244,  2.0171,  2.3651,  2.5384,  2.7659,  2.8411,
                 3.1875,  3.5971,  3.5204,  3.5899,  3.7739,  3.8539,  3.8851,
                 4.1475,  4.4171,  4.3544,  4.3984,  4.5056,  4.6739,  4.9376,
                 4.8619,  5.2264,  5.1011,  5.2059,  5.2059,  5.1019,  5.0691,
                 5.1936,  5.2204,  5.32  ,  5.2824,  5.2844,  5.28  ,  5.2499,
                 5.2491,  5.2419,  5.2244,  5.25  ,  5.2424,  5.2011,  5.2019,
                 5.2984,  5.4299,  5.4819,  5.6059,  5.6875,  5.6659,  5.5504,
                 5.5684,  5.7884,  5.5475,  5.6539,  5.6816,  5.9744,  6.1384,
                 6.4491,  6.3531,  6.35  ,  6.2851,  6.4091,  6.3331,  6.74  ,
                 6.6291,  6.6275,  6.8324,  7.2836,  7.2896,  7.2976,  7.2476,
                 7.3384,  7.6656,  8.1816,  8.6884,  9.3411,  9.1859,  9.3675,
                 9.8051,  9.7876,  9.9259,  9.8036,  9.9491, 10.1211,  9.8916,
                 9.63  ,  9.7396,  9.6744,  9.7075, 10.0611, 10.08  , 10.31  ,
                10.4664, 10.4731, 10.6459, 10.4275, 10.4675, 10.1736, 10.16  ,
                10.0251,  9.9056, 10.0404,  9.9859,  9.9124, 10.0939, 10.0131,
                10.2419, 10.1084, 10.0899, 10.2036, 10.2075, 10.0259, 10.3139,
                10.2331, 10.4736, 10.5699, 11.0136, 11.0116, 11.4376, 11.4219,
```

```
11.5891, 11.5419, 11.3176, 11.4259, 11.3204, 11.4499, 11.8531,
11.7171, 11.5275, 11.7291, 11.5011, 11.9136, 12.2675, 12.2859,
12.4576, 13.2696, 13.1836, 13.4624, 14.1651, 14.1675, 14.2716,
14.6675, 14.6636, 14.0116, 14.07  , 13.7164, 13.3944, 13.7611,
13.7264, 13.8496, 13.8956, 13.6939, 13.8571, 13.7476, 13.7964,
13.4596, 13.2784, 13.3819, 13.6004, 14.2571, 14.3971, 14.49  ,
14.4844, 14.5979, 14.5051, 14.2899, 14.2219, 14.4844, 14.81  ,
15.2275, 15.6076, 15.5804, 15.4136, 15.3396, 15.3796, 14.81  ,
14.6811, 14.7859, 14.5816, 14.5356, 14.7491, 14.9579, 14.85  ,
15.2539, 15.6675, 15.5979, 16.1075, 16.5731])
```

In [40]:
```
#plotting naive E[N(t)] against Λ with bin size=0.1
plt.plot(t,Inte)
plt.plot(t,np.mean(testb,axis=0))
plt.xlabel('Time')
plt.ylabel('Number of Events upto Time t')
plt.legend(('Λ','E[N(t)]'))
plt.show()
```



In [ ]:
```
# we can see that naive way is clearly incorrect by seeing the plot
```

In [21]:
```
#4
P=[[1/2,1/3,1/6],[0,1/3,2/3],[1/2,0,1/2]]
state0=1
state1=0
state2=0
newstate=0
for i in range(0,999):

    z=np.random.uniform(0,1)
    if z<P[newstate][0]:
        state0=state0+1
        newstate=0

    elif z<P[newstate][0]+P[newstate][1]:
        state1=state1+1
        newstate=1

    else:
        state2=state2+1
        newstate=2

print("the long-run proportion of time the DTMC is in state 0 is,", state0/1000)
print("the long-run proportion of time the DTMC is in state 1 is,", state1/1000)
print("the long-run proportion of time the DTMC is in state 2 is,", state2/1000)
print("Long run average of DTMC is ",(state0*0 + state1*1 + state2*2)/1000)
```

```
the long-run proportion of time the DTMC is in state 0 is, 0.397
the long-run proportion of time the DTMC is in state 1 is, 0.21
the long-run proportion of time the DTMC is in state 2 is, 0.393
Long run average of DTMC is  0.996
```

In [ ]: