Name: Somya Khandelwal

Roll Number: 200123056

Department: Mathematics and Computing

Course: MA 323 - Monte Carlo Simulation

Lab: 02

In [45]:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
```

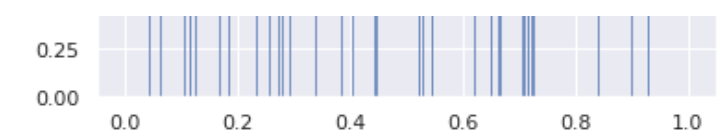
Task 1:

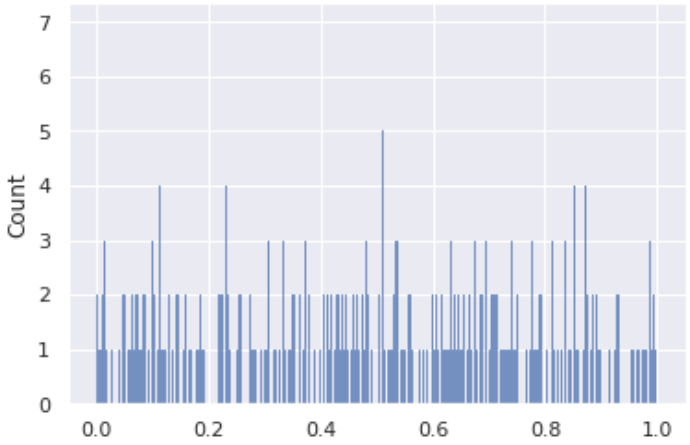
$$U_{i+1} = (U_{i-17} - U_{i-5}).$$

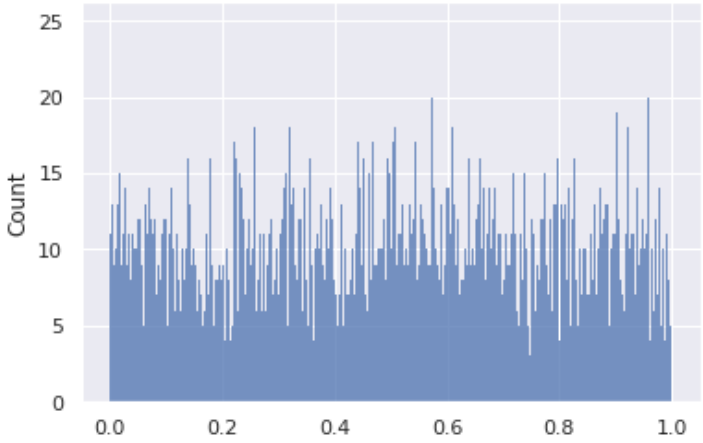
In the event that $U_i < 0$, set $U_i = U_i + 1$.

In [60]:

```
a=1741
b=2731
m=12960
xt=2
D = np.array([])
sns.set_theme(); np.random.seed(0)
xi=xt
N=[1000,10000,100000]
for i in range(1,18):
  xi=(xi*a+b)%m
  ui= xi/m
  D=np.insert(D,len(D),ui)

for n in N:
  d=D
  for i in range(18,n+1):
    ui = d[i-18]-d[i-6]
    if(ui<0):
      ui+=1
    d=np.insert(d,len(d),ui)
  sns.histplot(data=d,bins = 10000)
  plt.show()
```



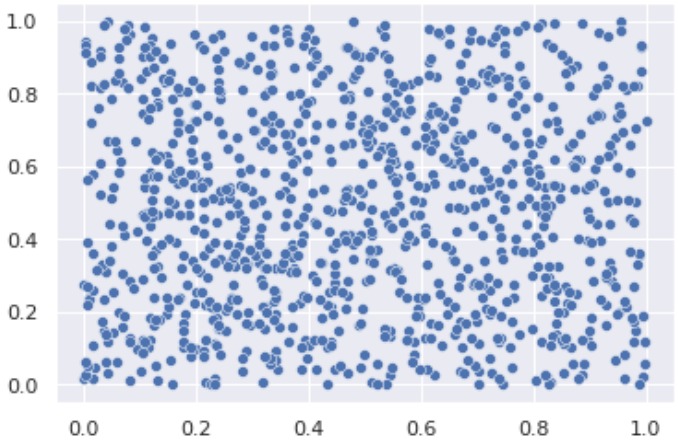


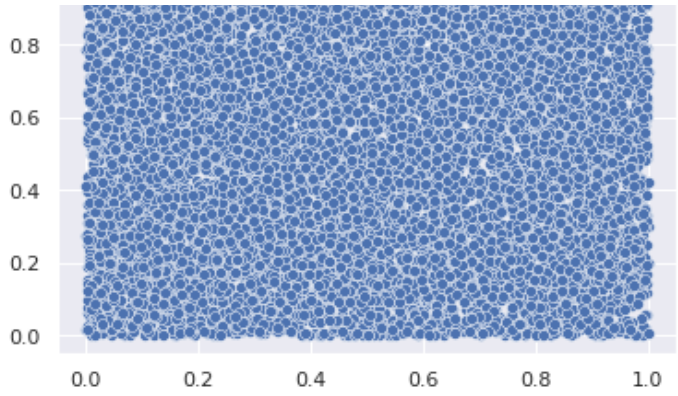


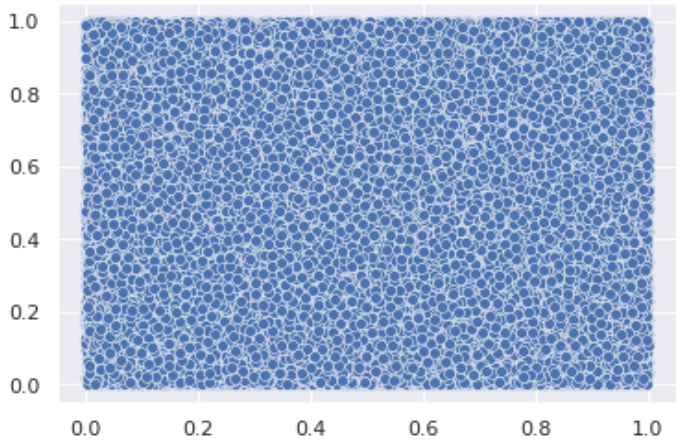
From above plots, we can observe that our distribution is fairly distributed even with a smaller bin size and large N.

In [61]:

```
for n in N:
  d=D
  for i in range(18,n+1):
    ui = d[i-18]-d[i-6]
    if(ui<0):
      ui+=1
    d=np.insert(d,len(d),ui)
  d1 = np.delete(d, 0)
  d2 = np.delete(d,len(d)-1)
  sns.scatterplot(x=d2, y=d1)
  plt.show()
```







From above Scatter plots, we can observe that the recursive relation of Un is fairly random, hence while plotting U(i-1),U(i), we get the points uniformly spaced out.

Task 2

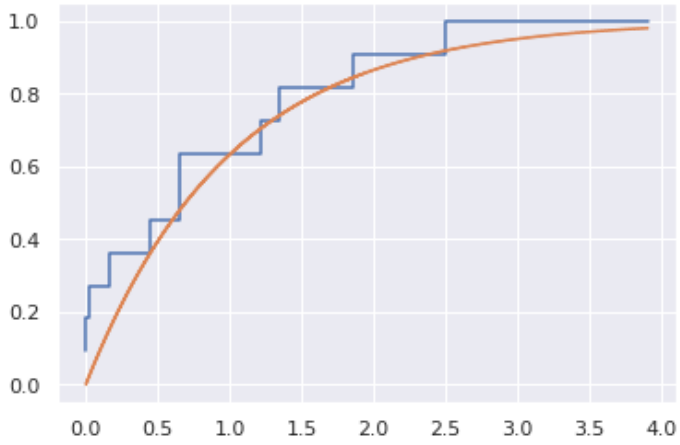
$$F(x) = 1 - e^{-x/\theta}\ ,\ x \geq 0,$$

where $\theta > 0$.

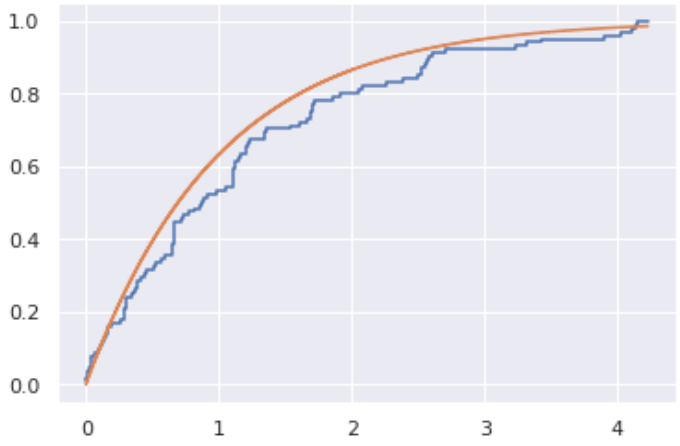
In [57]:

```
a=1741
b=2731
m=12960
xi=2
theta=1
D = np.array([])
sns.set_theme(); np.random.seed(0)

N=[10,100,1000,10000,100000]
for n in N:
  D = np.array([])
  X=np.array([])
  xi=2
  for i in range(n+1):
    xi=(xi*a+b)%m
    ui= xi/m
    val = -theta*(math.log(1-ui))
    D=np.insert(D,len(D),val)
    X=np.insert(X,len(X),ui)

  xt, counts = np.unique(D, return_counts=True)
  cusum = np.cumsum(counts)
  cusum = cusum / cusum[-1]
  plt.step(x=xt, y=cusum)
  xf = np.linspace(0, np.amax(xt), 1000)
```

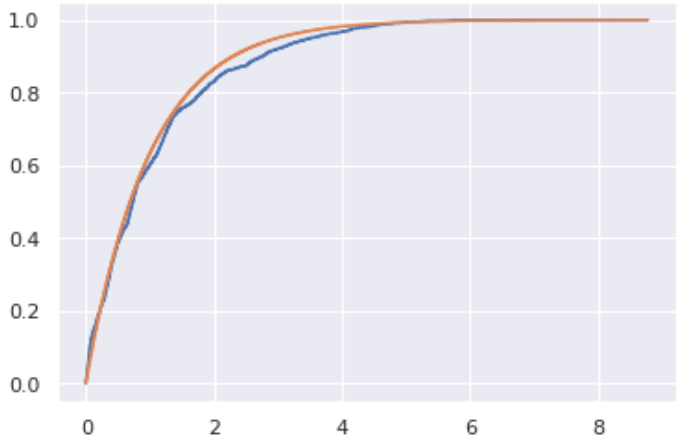
```
  yf=np.array([])
  for xtt in xf:
    yf=np.insert(yf,len(yf),1- math.exp(-xtt/theta))
  plt.step(x=xf,y=yf)
  plt.show()
  print('For n = %d'%n + ': Experimental Mean = %f'%(np.average(xt)) + ', Theoritical Me
an = %f'%(theta))
  print('For n = %d'%n + ': Experimental Variance = %f'%(np.var(xt)) + ', Theoritical Va
riance = %f'%(theta*theta))
```



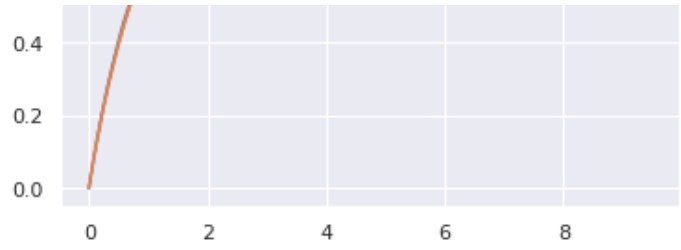

```
For n = 10: Experimental Mean = 1.158288, Theoritical Mean = 1.000000
For n = 10: Experimental Variance = 1.318390, Theoritical Variance = 1.000000
```



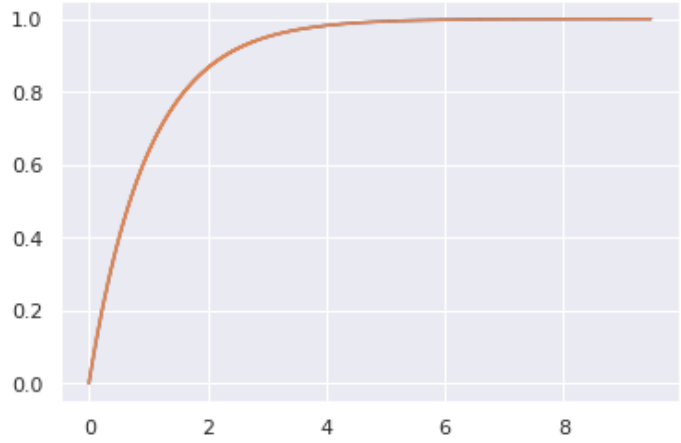

```
For n = 100: Experimental Mean = 1.219845, Theoritical Mean = 1.000000
For n = 100: Experimental Variance = 1.210073, Theoritical Variance = 1.000000
```




```
For n = 1000: Experimental Mean = 1.102956, Theoritical Mean = 1.000000
For n = 1000: Experimental Variance = 1.295442, Theoritical Variance = 1.000000
```




```
For n = 10000: Experimental Mean = 0.980423, Theoritical Mean = 1.000000
For n = 10000: Experimental Variance = 1.012048, Theoritical Variance = 1.000000
```




```
For n = 100000: Experimental Mean = 0.999564, Theoritical Mean = 1.000000
For n = 100000: Experimental Variance = 0.995915, Theoritical Variance = 1.000000
```

From above plots, we can observe that our plot approaches actual function as we increase N, hence the error in mean and variance also decreases.

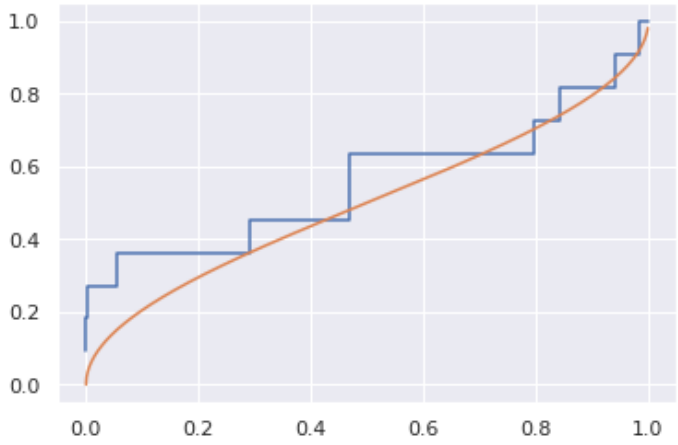
Task 3:

$$F(x) = \frac{2}{\pi} \text{ arc sin } \sqrt{x} \ , \ 0 \leq x \leq 1.$$

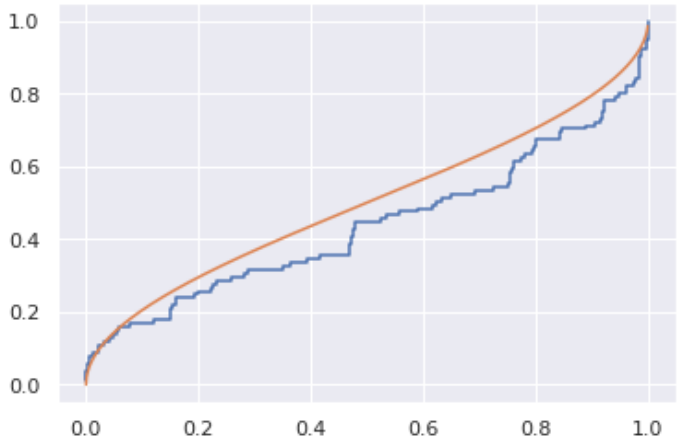
In [62]:

```
a=1741
b=2731
m=12960
xi=2
theta=1
D = np.array([])
sns.set_theme(); np.random.seed(0)

N=[10,100,1000,10000,100000]
for n in N:
  D = np.array([])
  X=np.array([])
  xi=2
  for i in range(n+1):
    xi=(xi*a+b)%m
    ui= xi/m
    val = np.sin(math.pi*ui/2)*np.sin(math.pi*ui/2)
    D=np.insert(D,len(D),val)
    X=np.insert(X,len(X),ui)

  xt, counts = np.unique(D, return_counts=True)
  cusum = np.cumsum(counts)
  cusum = cusum / cusum[-1]
  # print(cusum)
  plt.step(x=xt, y=cusum)
  xf = np.linspace(0, np.amax(xt), 1000)
```

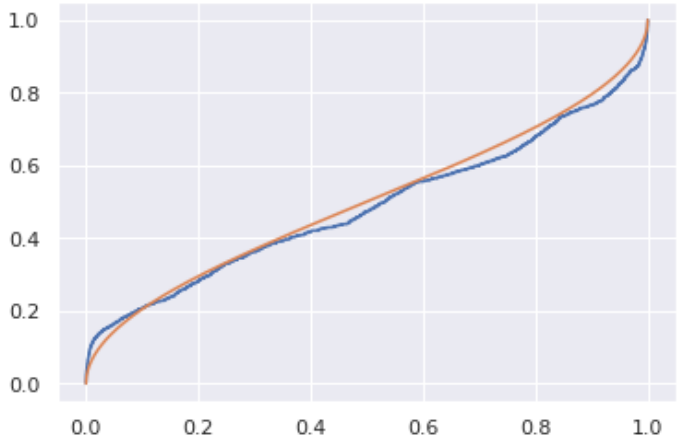
```
  yf=np.array([])
  for xtt in xf:
    yf=np.insert(yf,len(yf),(2/(math.pi))*np.arcsin(math.sqrt(xtt)))
  sns.lineplot(x=xf,y=yf)
  plt.show()
  print('For n = %d'%n + ': Experimental Mean = %f'%(np.average(xt)) + ', Theoritical Me
an = 0.5')
  print('For n = %d'%n + ': Experimental Variance = %f'%(np.var(xt)) + ', Theoritical Va
riance = %f'%(1/8))
```



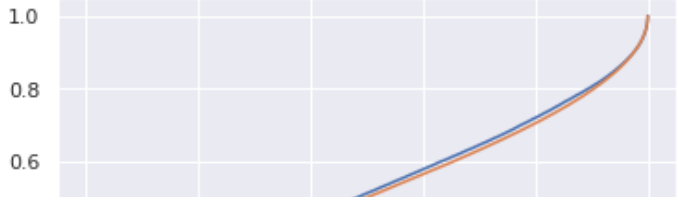

```
For n = 10: Experimental Mean = 0.530974, Theoritical Mean = 0.5
For n = 10: Experimental Variance = 0.146830, Theoritical Variance = 0.125000
```

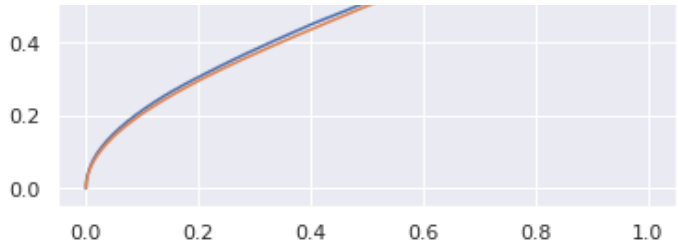



```
For n = 100: Experimental Mean = 0.571871, Theoritical Mean = 0.5
For n = 100: Experimental Variance = 0.127361, Theoritical Variance = 0.125000
```

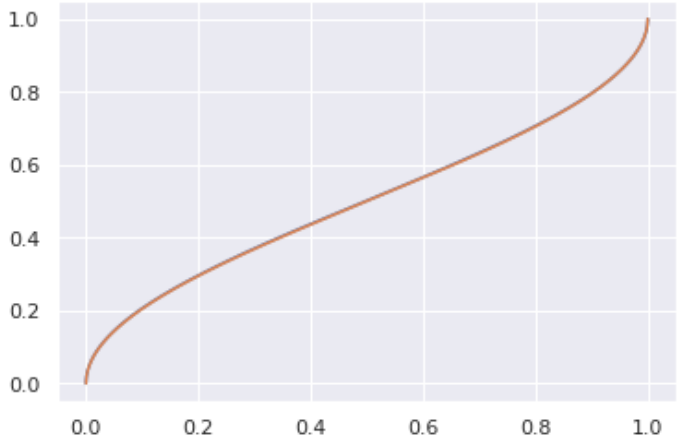



```
For n = 1000: Experimental Mean = 0.516472, Theoritical Mean = 0.5
For n = 1000: Experimental Variance = 0.131919, Theoritical Variance = 0.125000
```






```
For n = 10000: Experimental Mean = 0.488865, Theoritical Mean = 0.5
For n = 10000: Experimental Variance = 0.124693, Theoritical Variance = 0.125000
```



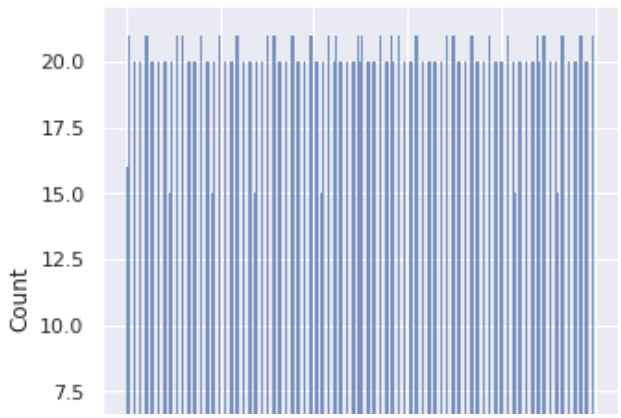

```
For n = 100000: Experimental Mean = 0.499961, Theoritical Mean = 0.5
For n = 100000: Experimental Variance = 0.125000, Theoritical Variance = 0.125000
```

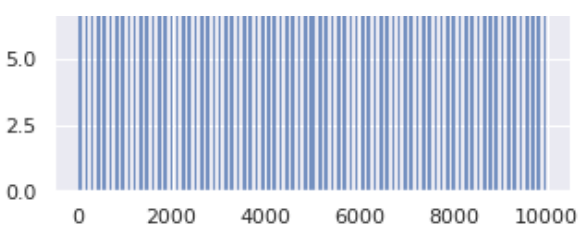
From above plots, we can observe that our plot approaches actual function as we increase N, hence the error in mean and variance also decreases.

Task 4:

In [59]:

```
X=[]
a=1
b=19568
m=19647
xi=2
theta=1
U=np.array([])
for i in range(100000+1):
  xi=(xi*a+b)%m
  ui= xi/m
  U=np.insert(U,len(U),ui)
  temp=math.floor(U[i]*5000)
  X.append(2*temp+1)

sns.displot(X,bins=10000)
plt.show()
```




5.0
2.5
0.0
0
2000
4000
6000
8000
10000