Student name: Steven Xie

Student number: 998979627

Program & Department: MEng MIE

Homework 1

Problem 1.

(a) Given X is uniformly distributed in [2,8]

$\theta = E[(X - 4)^+] = P(2 \le X \le 4) \times 0 + P(4 < X \le 8) \times E[(X-4)]$

$= \frac{4-2}{8-2} \times 0 + \frac{8-4}{8-2} \times (E(X) - 4)$

$= \frac{2}{3} \times (\int_4^8 \frac{x}{8-4} - 4)$

$= \frac{4}{3}$

(b) Shown in code

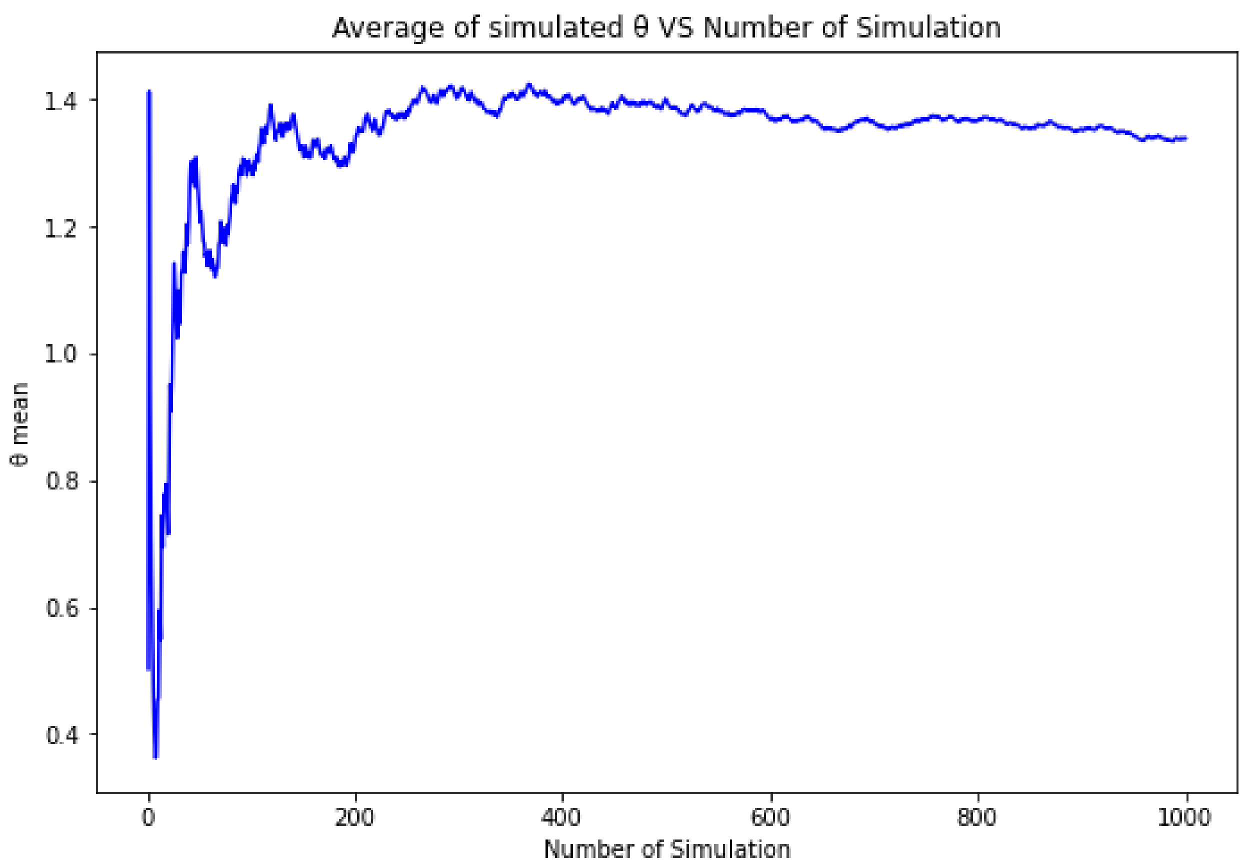
```
import numpy as np
# Set number of replications
N = 1000

# fix random number seed
np.random.seed(1)

# Define lists to keep samples of the outputs across replications
T_list = []
Ave_list = []

# Run the simulation for N times
for i in range(1,N+1):
  average = 0
  X = np.random.uniform(2,8) # Define the domain and distribution of X
  theta = max(0,X-4) # Define θ function
  T_list.append(theta) # Update each simulation result
  average = sum(T_list)/i # Take the average of all simulated θ
  Ave_list.append(average) # Update the average of to list
```

```
print('Estimated expected θ:', np.mean(T_list))
print ('95% CI for θ:', np.mean(T_list),'+/-',1.96*np.std(T_list, ddof = 1)/np.sqrt(N))
```

```
Estimated expected θ: 1.3383673235881042
95% CI for θ: 1.3383673235881042 +/- 0.08217292686292378
```

(c) In the plot we can see that the average of simulated θ converge to the computed E(X) = 1.33

```python
import matplotlib.pyplot as plt
X1 = range(1,1001)
Y1 = Ave_list
plt.figure(figsize=(9,6))
plt.plot(X1, Y1, 'b')
plt.title('Average of simulated θ VS Number of Simulation')
plt.xlabel('Number of Simulation')
plt.ylabel('θ mean')
plt.show()
```



Problem 2

(a). Shown in code. With 1 replication before T = 1000 the average number of functional components is 1.156.

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

# Set number of replications
N = 1
# Define lists to keep samples of the outputs across replications
TTF_list = []
Ave_list = []

# fix random number seed
np.random.seed(1)

for rep in range (0,N):
    # start with 2 functioning components at time 0
    clock = 0
    S = 2
    # initialize the time of events
    NextRepair = float('inf')
    NextFailure = np.ceil(6*np.random.random())
    EventTimes = [0]
    States = [2]
    # Define variables to keep the area under the sample path
    # and the time and state of the last event
    Area = 0.0
    Tlast = 0
    Slast = 2

    while clock <1000:  # Change the condition to T = 1000
        # advance the time
        clock = min(NextRepair, NextFailure)

        if NextRepair < NextFailure:
            # next event is completion of a repair
            S = S + 1
            if S==2:
              NextRepair = float('inf')  # no machine under repair
```

```python
        if NextRepair < NextFailure:
            # next event is completion of a repair
            S = S + 1
            if S==2:
              NextRepair = float('inf')  # no machine under repair
            else:
              NextRepair = clock + 2.5 # start repairing the second machine
        else:
            # next event is a failure, States cannot be below 0
            S = max(0,S - 1)
            # Next failture happends with at least 1 machine is running + the failure time
            if S==0:
              NextRepair = clock + 2.5
              NextFailure = max(NextRepair + np.ceil(6*np.random.random()), clock + np.ceil(6*np.random.random()))
            if S==1:
              NextRepair = clock + 2.5
              NextFailure = clock + np.ceil(6*np.random.random())

        # Update the area under the sample path and the
        # time and state of the last event
        Area = Area + (min(1000,clock) - Tlast)* Slast
        Tlast = min(1000,clock) # Clock within 1000
        Slast = S

    # The Average of S within time T = 1000
    Ave_list.append(Area/1000)

print('Estimated expected ave. # of func. comp. till T=1000:', np.mean(Ave_list))
```

```
Estimated expected ave. # of func. comp. till T=1000: 1.156
```

(b). Model is almost identical as in (a) except that the Slast is replaced with A_t which is a binary variable. The expected average system availability time till T=1000 is 82.5%.

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

# Set number of replications
N = 1
# Define lists to keep samples of the outputs across replications
TTF_list = []
Ave_list = []

# fix random number seed
np.random.seed(1)

for rep in range (0,N):
    # start with 2 functioning components at time 0
    clock = 0
    S = 2
    # initialize the time of events
    NextRepair = float('inf')
    NextFailure = np.ceil(6*np.random.random())
    EventTimes = [0]
    States = [2]
    # Define variables to keep the area under the sample path
    # and the time and state of the last event
    Area = 0.0
    Tlast = 0
    A_t = 1 # A_t = 1 if at least 1 machine is running, = 0 OW

    while clock <1000:  # Change the condition to T = 1000
        # advance the time
        clock = min(NextRepair, NextFailure)

        if NextRepair < NextFailure:
            # next event is completion of a repair
            S = S + 1
            if S==2:
              NextRepair = float('inf')  # no machine under repair
            else:
              NextRepair = clock + 2.5 # start repairing the second machine
        else:
            # next event is a failure, States cannot be below 0
            S = max(0,S - 1)
            # Next failture happends with at least 1 machine is running + the failure time
            if S==0:
              NextRepair = clock + 2.5
              NextFailure = max(NextRepair + np.ceil(6*np.random.random()), clock + np.ceil(6*np.random.random()))
            if S==1:
              NextRepair = clock + 2.5
              NextFailure = clock + np.ceil(6*np.random.random())

        # Update the area under the sample path and the
        # time and state of the last event
        Area = Area + (min(1000,clock) - Tlast)* A_t
        Tlast = min(1000,clock) # Clock within 1000
        # if S>0 system is availbe A(t) = 1, if S=0 A(t)=0 system is down
        if S== 0:
          A_t = 0
        else:
          A_t = 1

    # The Average of A_t within time T = 1000
    Ave_list.append(Area/1000)

print('Estimated expected ave. system availibility till T=1000:', np.mean(Ave_list))
```

```
Estimated expected ave. # of func. comp. till T=1000: 0.825
```

(c). The estimates for T=1000 and T=3000 are very close (1.156 vs 1.161 and 82.5% vs 83.4 %). Therefore, when we simulate a system with a relatively large number, the estimations of the average of the simulations will converge.

```
Estimated expected ave. # of func. comp. till T=3000: 1.1611666666666667
```

```
Estimated expected ave. system availibility till T=3000: 0.8341666666666666
```

Problem 3

Code Shown below.

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

# Set number of replications
R = 1000
# Define lists to keep samples of the outputs across replications
TTF_list = []
Ave_list = []

# fix random number seed
np.random.seed(1)

# Set the number of components N
N = int(input('enter the number of components in the system:'))
for rep in range (0,R):
    # start with 2 functioning components at time 0
    clock = 0
    S = N
    # initialize the time of events
    NextRepair = float('inf')
    NextFailure = np.ceil(6*np.random.random())
    EventTimes = [0]
    States = [S]
    # Define variables to keep the area under the sample path
    # and the time and state of the last event
    Area = 0.0
    Tlast = 0
    Slast = S

    while S > 0:
        # advance the time
        clock = min(NextRepair, NextFailure)

        if NextRepair < NextFailure:
            # next event is completion of a repair
            S = S + 1
            NextRepair = float('inf')
        else:
```

```
        else:
            # next event is a failure
            S = S - 1
            if S > 0:
                NextRepair = clock + 2.5
                NextFailure = clock + np.ceil(6*np.random.random())
        # Update the area under the sample path and the
        # time and state of the last event
        Area = Area + (clock - Tlast)* Slast
        Tlast = clock
        Slast = S

    # save the TTF and average # of func. components
    TTF_list.append(clock)
    Ave_list.append(Area/clock)

print('Estimated expected TTF:', np.mean(TTF_list))
print('Estimated expected ave. # of func. comp. till failure:', np.mean(Ave_list))

print ('95% CI for TTF:', np.mean(TTF_list), "+/-",
       1.96*np.std(TTF_list, ddof = 1)/np.sqrt(R))
print ('95% CI for ave. # of func. comp.:', np.mean(Ave_list), "+/-",
       1.96*np.std(Ave_list, ddof = 1)/np.sqrt(R))
```

```
enter the number of components in the system:
```

Result shown below:

N = 2: TTF = 13.927 +/- 0.727

```
enter the number of components in the system:2
Estimated expected TTF: 13.927
Estimated expected ave. # of func. comp. till failure: 1.568840332841451
95% CI for TTF: 13.927 +/- 0.7274410509424089
95% CI for ave. # of func. comp.: 1.568840332841451 +/- 0.008120648766690015
```

N = 3: TTF =24.527 +/- 0.989

```
enter the number of components in the system:3
Estimated expected TTF: 24.527
Estimated expected ave. # of func. comp. till failure: 2.055022982889395
95% CI for TTF: 24.527 +/- 0.9892156990124918
95% CI for ave. # of func. comp.: 2.055022982889395 +/- 0.01786110134802987
```

N = 4 TTF= 34.725 +/- 1.197

```
enter the number of components in the system:4
Estimated expected TTF: 34.725
Estimated expected ave. # of func. comp. till failure: 2.567804419903376
95% CI for TTF: 34.725 +/- 1.1969074875418457
95% CI for ave. # of func. comp.: 2.567804419903376 +/- 0.024892310549548383
```

Problem 4

standard error of $\bar{X}_n$ given X_i's are iid, std of X is σ

$$\text{std of } \bar{X}_n = \sqrt{Var(\bar{X}_n)}$$

$$= \sqrt{Var \frac{1}{n}\sum_{i=1}^{n} X_i}$$

$$= \sqrt{\frac{1}{n^2} Var[X_1+X_2+\cdots+X_n]} \quad \text{given } X_i\text{'s are iid}$$

$$= \frac{1}{n}\sqrt{n \, Var X_1} \quad \text{give std } X=\sigma \quad Var X = \sigma^2$$

$$= \frac{1}{n}\sqrt{n\sigma^2}$$

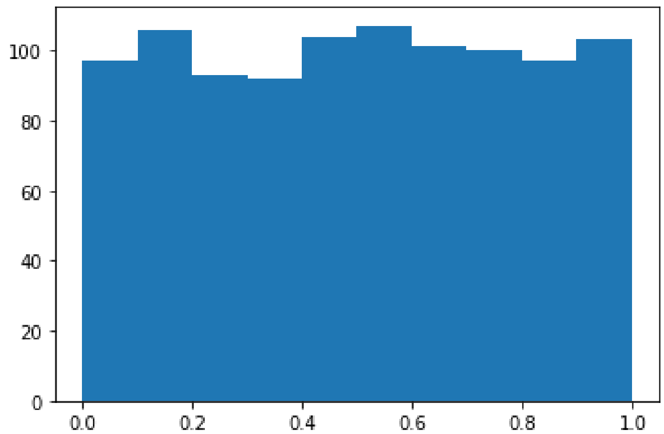
$$= \frac{\sigma}{\sqrt{n}}$$

Problem 5

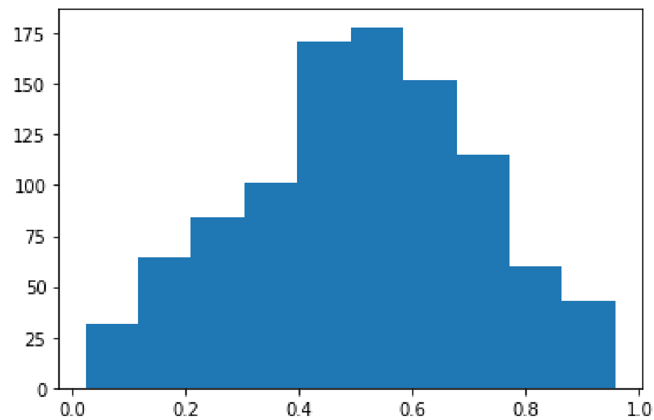
```python
import numpy as np
import matplotlib.pyplot as plt

# Set number of replications
N = 1000

# input the number of samples
n = int(input('Enter the number of samples: '))

# fix random number seed
np.random.seed(1)

Xn = []
Ave_Xn = []
for i in range(1,N+1):
  for j in range(n):
    X = np.random.uniform(0,1) # Define the domain and distribution of X
    Xn.append(X)
  Ave_Xn.append(np.mean(Xn))
  Xn = []
plt.hist(Ave_Xn)
```

```
Enter the number of samples:
```

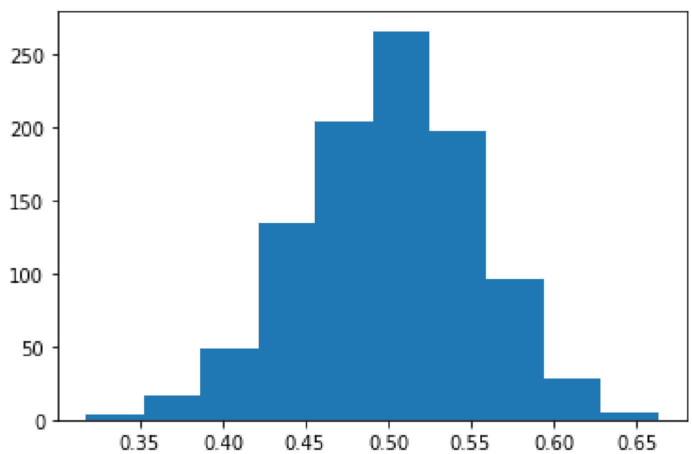
n = 1



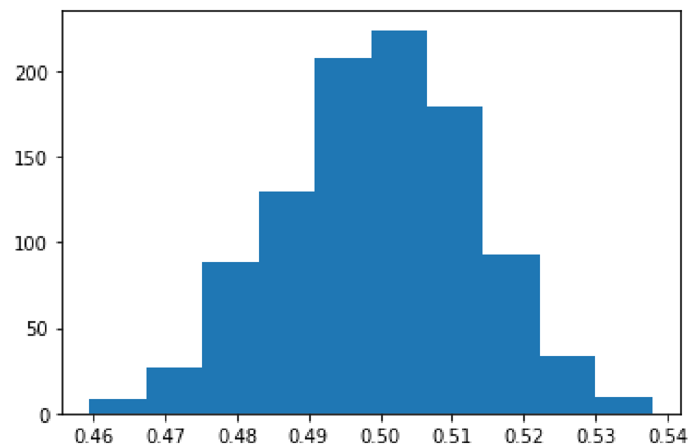
n= 2



n = 30



n=500



As n increases, the means of samples become closer to the expected value of mean = 0.5, the variances become smaller. The distribution of means of samples become more likely to be a normal distribution with decreasing variance as n increases.