```python
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm_notebook
import scipy.stats as st
```

QUESTION 1

$Var(X) = E(X^2) - E(X)^2$

Date No.

Q1) To show whether $Var(\hat{\beta}_1)$ or $Var(\hat{\beta}_2)$ is larger.
An estimator is more efficient if the variance of the estimator is smaller.

$$\hat{\beta}_1 = \left(\frac{1}{R}\sum_{i=1}^{R} z'_i\right)\left(\frac{1}{R}\sum_{i=1}^{R} z''_i\right)$$
$$= \frac{1}{R^2}\left(\sum_{i=1}^{R} z'_i\right)\left(\sum_{i=1}^{R} z''_i\right)$$

$$Var(\hat{\beta}_1) = \frac{1}{R^4} Var\left[\left(\sum_{i=1}^{R} z'_i\right)\left(\sum_{i=1}^{R} z''_i\right)\right]$$
$$= \left(Var(z'z'') + (R-1)\left[E(z')^2 Var(z'') + E(z'')^2 Var(z')\right]\right)/R^2$$

$$Var(\hat{\beta}_2) = Var\left(\frac{1}{R}\sum_{i=1}^{R} z'_i z''_i\right)$$
$$= R\left[Var(z'z'')\right]/R^2$$

To show $Var(\hat{\beta}_1) < Var(\beta_2)$, we need to show (1):

$$R[Var(z'z'')] \geq Var(z'z'') + (R-1)[E(z')^2 Var(z'') + E(z'')^2 Var(z')]$$

which is also

$$\cancel{(R-1)}[Var(z'z'')] \geq \cancel{(R-1)}[E(z')^2 Var(z'') + E(z'')^2 Var(z')]$$

(let $X = z'$, $Y = z''$)

$$Var(XY) = E[(XY)^2] - E[XY]^2$$
$$= E[X^2]E[Y^2] - E[X]^2 E[Y]^2$$
$$= E[X^2]E[Y^2] - E[X]^2E[Y^2] + E[X]^2E[Y^2] - E[X]^2E[Y]^2$$
$$= E[X]^2(E[Y^2] - E[Y]^2) + E[Y^2](E[X^2] - E[X]^2)$$
$$= E[X]^2 Var[Y] + E[Y^2] Var[X]$$
$$\geq E[X]^2 Var[Y] + E[Y]^2 Var[X]$$

(as $E[Y^2] \geq E[Y]^2$)

Therefore (1) is shown, and thus $Var(\beta_1) < Var(\beta_2)$ and β_1 is a better estimator of α

//

QUESTION 2

We define a function that simulate how the company capital changes with the next claim.

Each duration to the successive claim has exponentially distributed.

```
def generate_claim(rate, increase, mean_claim):
    time = np.random.exponential(1/rate)
    claim = np.random.exponential(mean_claim)
    change = time*increase - claim
    return change, time
```

We now simulate the company over the days. We also record the predicted probability and its recorded time step.

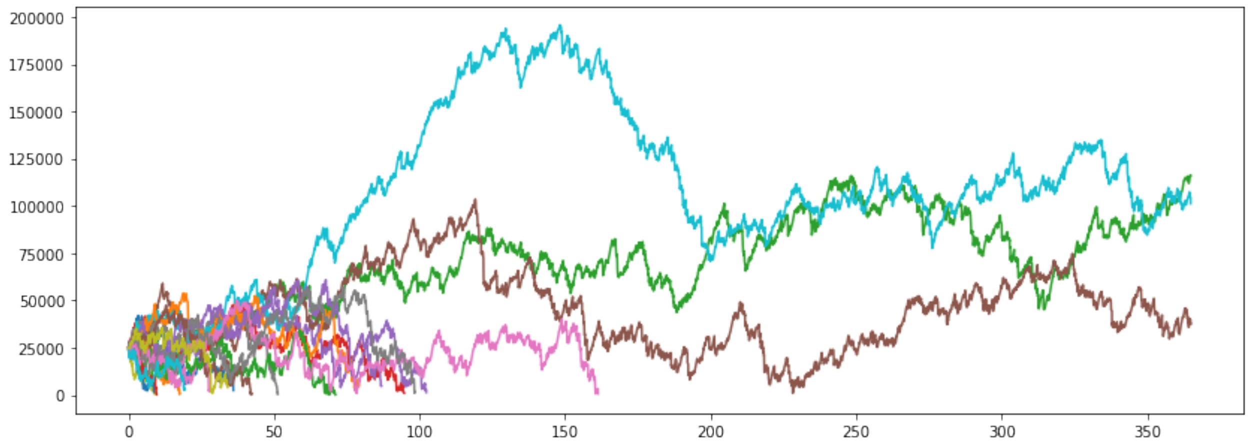
```
alpha = 0.05

history_capitals = []
history_times = []
passing = 0
failing = 0
sample_means = []
confidence_intervals = []


for i in tqdm_notebook(range(2000)):
    current_capital = 25000
    current_time = 0
    history_capital = []
    history_time = []

    sample_mean = []
    confidence_upper = []
    confidence_lower = []

    while True:
        change, time = generate_claim(rate=10, increase=10000,
mean_claim=1000)
        current_time += time
        current_capital += change
        if current_time > 365:
            passing += 1
            break
        if current_capital < 0:
            failing += 1
            break
        history_capital.append(current_capital)
```

```
        history_time.append(current_time)

    history_capitals.append(history_capital)
    history_times.append(history_time)

    k = i+1
    sample_mean = passing/k
    interval = st.norm.ppf(1-alpha/2)*np.sqrt(sample_mean*(1-sample_mean)/k)
    sample_means.append(sample_mean)
    confidence_intervals.append(interval)
```

```
HBox(children=(IntProgress(value=0, max=2000), HTML(value='')))
```

We inspect a few samples to ensure that we are generating the correct thing.

```
plt.figure(figsize=(14,5))
for x,y in zip(history_times[:20], history_capitals[:20]):
    plt.plot(x, y)
plt.show()
print("In the simultations, {} passed, {} failed".format(passing, failing))
```




```
In the simultations, 472 passed, 1528 failed
```

```
plt.figure(figsize=(14,5))
plt.plot(sample_means)
plt.plot([xx+yy for (xx,yy) in zip(sample_means,confidence_intervals)])
plt.plot([xx-yy for (xx,yy) in zip(sample_means,confidence_intervals)])
plt.ylim(sample_means[-1] - 0.1, sample_means[-1] + 0.1)
plt.xlabel("Number of simulations")
plt.ylabel("Estimate and confidence interval")
plt.title("Running estimate and confidence interval for first x simulations")
plt.show()
```


```
print("The estimated probability that the firm capital is positive",
      "\nfor the first 365 days is {}".format(sample_means[-1]),
      "\nwith a 95% confidence interval of",
       "({:.4f},{:.4f})".format(sample_means[-1]-confidence_intervals[-1],
                                sample_means[-1]+confidence_intervals[-1]))
```

```
The estimated probability that the firm capital is positive
for the first 365 days is 0.236
with a 95% confidence interval of (0.2174,0.2546)
```

```
for index,confidence_interval in enumerate(confidence_intervals):
    if index < 100: continue
    if confidence_interval < 0.1:
        break
```

```
 print("If the analysis has stopped at {}-th iteration,".format(index),
       "\nwe can stop the simulation and estimate a probability of {:.4f}"
       .format(sample_means[index]),
       "\nwith a 95% confidence interval of",
       "({:.4f},{:.4f})".format(sample_means[index]-
confidence_intervals[index],

 sample_means[index]+confidence_intervals[index]))
```

```
If the analysis has stopped at 100-th iteration,
we can stop the simulation and estimate a probability of 0.2574
with a 95% confidence interval of (0.1722,0.3427)
```

Question 3

We define reusable functions for question 3.

```
%reset -sf
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm_notebook
import scipy.stats as st

def mean_confidence_interval(data, confidence=0.95, n=None):
    a = 1.0 * np.array(data)
    if n == None: n = len(a)
    m, se = np.mean(a), st.sem(a)
    h = se * st.t.ppf((1 + confidence) / 2., n-1)
    return m, h

def queuing_simulation(unif_arr, unif_svc, lambda_arr, lambda_svc,
                       alpha=0.05, start_index_for_mean=0):
    assert unif_arr.shape == unif_svc.shape
    num_sims, num_cust = unif_arr.shape

    arrvial_seed=unif_arr
    service_seed=unif_svc

    system_times_sims = []
    system_times_means = []

    for sim in range(num_sims):
        arrivals = np.cumsum(-np.log(arrvial_seed[sim]))*lambda_arr
        services = -np.log(service_seed[sim])*lambda_svc

        current_time = 0
        departures = []
        for arrival,service in zip(arrivals,services):
            current_time = max(current_time,arrival)
            current_time += service
            departures.append(current_time)

        system_times = departures - arrivals
        system_times = system_times[start_index_for_mean:]
        system_times_sims.append(system_times)
        system_times_means.append(np.mean(system_times))

    req_system_times_confidence = []
    req_system_times_means = []
```

```
    for i,mean in enumerate(system_times_means[:-2]):
        req_sample_mean, interval =
mean_confidence_interval(system_times_means[:i+1])
        req_system_times_means.append(req_sample_mean)
        req_system_times_confidence.append(interval)

    return (np.array(req_system_times_means),
            np.array(req_system_times_confidence),
            np.array(system_times_means),
            np.array(system_times_sims))
```

QUESTION 3a

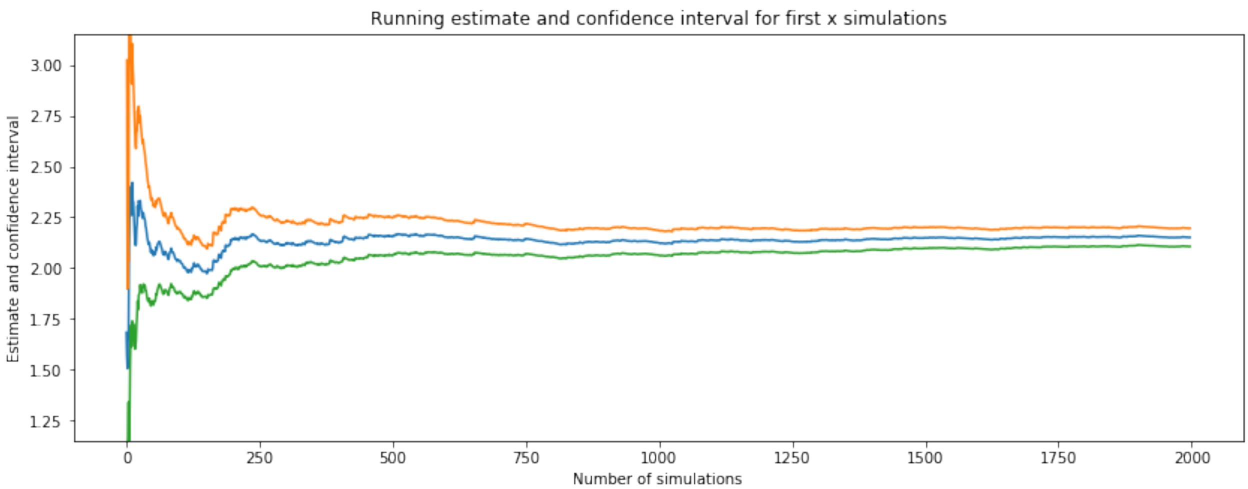
```
unif_arr_3a = np.random.uniform(size=(2000,100))
unif_svc_3a = np.random.uniform(size=(2000,100))
```

```
req_system_times_means, req_system_times_confidence, _, _ = \
queuing_simulation(unif_arr_3a, unif_svc_3a, lambda_arr=1, lambda_svc=0.7)
```

```
/usr/local/anaconda3/lib/python3.7/site-packages/numpy/core/_methods.py:140:
RuntimeWarning: Degrees of freedom <= 0 for slice
  keepdims=keepdims)
/usr/local/anaconda3/lib/python3.7/site-packages/numpy/core/_methods.py:132:
RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```

We can ignore the error. This is because the confidence interval of one value is undefined.

```
plt.figure(figsize=(14,5))
plt.plot(req_system_times_means)
plt.plot([xx+yy for (xx,yy) in
zip(req_system_times_means,req_system_times_confidence)])
plt.plot([xx-yy for (xx,yy) in
zip(req_system_times_means,req_system_times_confidence)])
plt.ylim(req_system_times_means[-1] - 1, req_system_times_means[-1] + 1)
plt.xlabel("Number of simulations")
plt.ylabel("Estimate and confidence interval")
plt.title("Running estimate and confidence interval for first x simulations")
plt.show()
```



```
print("The average time spent by the first 100 customers is",
      "\n{:.5f} \u00B1 {:.5f} (95%)".format(req_system_times_means[-1],
                                             req_system_times_confidence[-1]))
```

```
The average time spent by the first 100 customers is
2.15135 ± 0.04451 (95%)
```

```
index = np.argwhere(req_system_times_confidence[1:]<0.1).min() + 1
print("To attained our required accuracy, we will just need to stop at" +
      "the {}-th iteration for this simulation set.".format(index),
      "\nThe average time customers spent in the long run is",
      "\n{:.5f} \u00B1 {:.5f} (95%)".format(req_system_times_means[index],

 req_system_times_confidence[index]))
```

```
To attained our required accuracy, we will just need to stop atthe 362-th
iteration for this simulation set.
The average time customers spent in the long run is
2.11705 ± 0.09984 (95%)
```

QUESTION 3b

```
unif_arr_3b = np.random.uniform(size=(2000,500))
unif_svc_3b = np.random.uniform(size=(2000,500))
```

```
_, _, _ ,system_times_sims = \
queuing_simulation(unif_arr_3b, unif_svc_3b, lambda_arr=1, lambda_svc=0.7)
```

```
plt.figure(figsize=(14,4))
plt.plot(np.mean(system_times_sims, axis=0)[:500])
plt.title("Average waiting time of n-th customer across different
simulations")
plt.xlabel("Customer number")
plt.ylabel("Average waiting time")
plt.show()
```

By inspection, we should drop the system time of the first 100 customers.

```
req_system_times_means, req_system_times_confidence, system_times_means_base,
_ = \
queuing_simulation(unif_arr_3b, unif_svc_3b, lambda_arr=1, lambda_svc=0.7,
                   start_index_for_mean=100)
```

```python
plt.figure(figsize=(14,5))
plt.plot(req_system_times_means)
plt.plot([xx+yy for (xx,yy) in
zip(req_system_times_means,req_system_times_confidence)])
plt.plot([xx-yy for (xx,yy) in
zip(req_system_times_means,req_system_times_confidence)])
plt.ylim(req_system_times_means[-1] - 0.1, req_system_times_means[-1] + 0.1)
plt.xlabel("Number of simulations")
plt.ylabel("Estimate and confidence interval")
plt.title("Running estimate and confidence interval for first x simulations")
plt.show()
```


```python
print("The average time customers spent in the long run is",
      "\n{:.5f} \u00B1 {:.5f} (95%)".format(req_system_times_means[-1],
                                             req_system_times_confidence[-1]))
```

```
The average time customers spent in the long run is
2.34528 ± 0.02994 (95%)
```

```python
index = np.argwhere(req_system_times_confidence[1:]<0.1).min() + 1
print("To attained our required accuracy, we will just need to stop at" +
      "the {}-th iteration for this simulation set.".format(index),
      "\nThe average time customers spent in the long run is",
      "\n{:.5f} \u00B1 {:.5f} (95%)".format(req_system_times_means[index],

 req_system_times_confidence[index]))
```

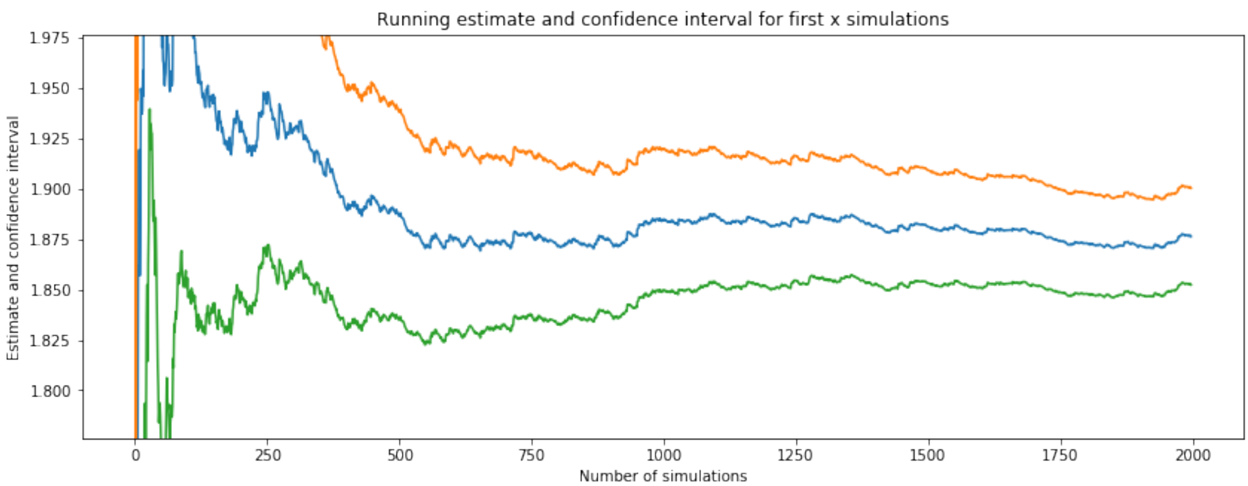
```
To attained our required accuracy, we will just need to stop atthe 215-th
iteration for this simulation set.
The average time customers spent in the long run is
2.40065 ± 0.09978 (95%)
```

Question 3c

We use the same common random numbers that was used for 3b for 3c, and show that one configuration is better than the other.

```
req_system_times_means, req_system_times_confidence, system_times_means_alt, _
= \
queuing_simulation(unif_arr_3b, unif_svc_3b, lambda_arr=1/1.25,
lambda_svc=0.56,
                   start_index_for_mean=100)
```

```
plt.figure(figsize=(14,5))
plt.plot(req_system_times_means)
plt.plot([xx+yy for (xx,yy) in
zip(req_system_times_means,req_system_times_confidence)])
plt.plot([xx-yy for (xx,yy) in
zip(req_system_times_means,req_system_times_confidence)])
plt.ylim(req_system_times_means[-1] - 0.1, req_system_times_means[-1] + 0.1)
plt.xlabel("Number of simulations")
plt.ylabel("Estimate and confidence interval")
plt.title("Running estimate and confidence interval for first x simulations")
plt.show()
```




```
print("The average time customers spent in the long run is",
      "\n{:.5f} \u00B1 {:.5f} (95%)".format(req_system_times_means[-1],
                                            req_system_times_confidence[-1]))
```

```
The average time customers spent in the long run is
1.87623 ± 0.02395 (95%)
```

```
index = np.argwhere(req_system_times_confidence[1:]<0.1).min() + 1
print("To attain our required accuracy, we will just need to stop at" +
      "the {}-th iteration for this simulation set.".format(index),
      "\nThe average time customers spent in the long run is",
      "\n{:.5f} \u00B1 {:.5f} (95%)".format(req_system_times_means[index],

 req_system_times_confidence[index]))
```

```
To attain our required accuracy, we will just need to stop atthe 155-th
iteration for this simulation set.
The average time customers spent in the long run is
1.93363 ± 0.09958 (95%)
```

```
differences = system_times_means_base -  system_times_means_alt
for index, diff in enumerate(differences[:-2]):
    difference, interval = mean_confidence_interval(differences[:index+2])
    if abs(difference) - abs(interval) > 0 and index+1 > 100:
        break
```

```
print("The new configuration is faster than the old configuration by" +
      "\n{:.5f} \u00B1 {:.5f} (95%).".format(difference, interval),
      "{} simulations were used.".format(index))
```

```
The new configuration is faster than the old configuration by
0.49653 ± 0.03506 (95%). 100 simulations were used.
```

As the confidence interval does not contain zero, we are at least 95% confident the second configuration is better than the original. This is the same conclusion with when we use CRN.

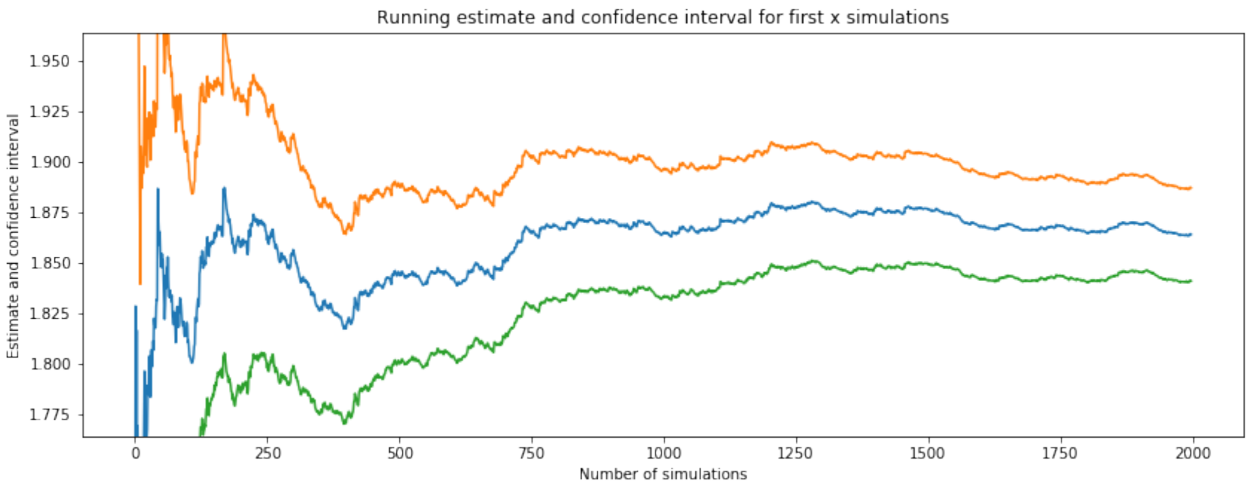
To compare systems, we require at least 100 simulations.

Question 3d

```
unif_arr_3b_idp = np.random.uniform(size=(2000,500))
unif_svc_3b_idp = np.random.uniform(size=(2000,500))
```

```
req_system_times_means, req_system_times_confidence, system_times_means_alt, _
= \
queuing_simulation(unif_arr_3b_idp, unif_svc_3b_idp, lambda_arr=1/1.25,
lambda_svc=0.56,
                   start_index_for_mean=100)
```

```
plt.figure(figsize=(14,5))
plt.plot(req_system_times_means)
plt.plot([xx+yy for (xx,yy) in
zip(req_system_times_means,req_system_times_confidence)])
plt.plot([xx-yy for (xx,yy) in
zip(req_system_times_means,req_system_times_confidence)])
plt.ylim(req_system_times_means[-1] - 0.1, req_system_times_means[-1] + 0.1)
plt.xlabel("Number of simulations")
plt.ylabel("Estimate and confidence interval")
plt.title("Running estimate and confidence interval for first x simulations")
plt.show()
```




```
print("The average time customers spent in the long run is",
      "\n{:.5f} \u00B1 {:.5f} (95%)".format(req_system_times_means[-1],
                                            req_system_times_confidence[-1]))
```

```
The average time customers spent in the long run is
1.86400 ± 0.02302 (95%)
```

```python
index = np.argwhere(req_system_times_confidence[1:]<0.1).min() + 1
print("To attain our required accuracy, we will just need to stop at" +
      "the {}-th iteration for this simulation set.".format(index),
      "\nThe average time customers spent in the long run is",
      "\n{:.5f} \u00B1 {:.5f} (95%)".format(req_system_times_means[index],

 req_system_times_confidence[index]))
```

```
To attain our required accuracy, we will just need to stop atthe 39-th
iteration for this simulation set.
The average time customers spent in the long run is
1.81773 ± 0.09953 (95%)
```

```python
differences = system_times_means_base -  system_times_means_alt
for i, diff in enumerate(differences[:-2]):
    difference, interval = mean_confidence_interval(differences[:i+2])
    if abs(difference) - abs(interval) > 0 and i+1 > 100:
        break
```

```python
print("The new configuration is faster than the old configuration by" +
      "\n{:.5f} \u00B1 {:.5f} (95%).".format(difference, interval),
      "{} simulations were used.".format(i))
```

```
The new configuration is faster than the old configuration by
0.67123 ± 0.19325 (95%). 100 simulations were used.
```

As the confidence interval does not contain zero, we are at least 95% confident the second configuration is better than the original. This is the same conclusion with when we use CRN.

To compare systems, we require at least 100 simulations. While the number of simulations are the same, we can see that the number of confidence interval for using CRN is smaller than not using CRN. The result is very similar and the two confidence intervals largely overlap.

We also have to note that the number of simulations used is another random variable. However, this is not important here we need to exceed the minimum quota of 100 simulations for reliable comparison of system configuration.

QUESTION 4

Tong Hui Kang Date No.

4a) To show $Z = e^{U^2}(1+e^{1-2U})/2$ is an unbiased estimator of θ.

$$Z = (e^{U^2} + e^{1-2U+U^2})/2$$
$$= (e^{U^2} + e^{(1-U)^2})/2$$
$$= \frac{1}{2}\int_0^1 e^{x^2} + e^{(1-x)^2}\,dx$$
$$= \frac{1}{2}\int_0^1 e^{x^2}\,dx + \frac{1}{2}\int_0^1 e^{(1-x)^2}\,dx$$
$$= \frac{1}{2}\int_0^1 e^{x^2}\,dx + \frac{1}{2}\int_{-1}^0 e^{x^2}\,dx$$
$$= \frac{1}{2}\int_0^1 e^{x^2}\,dx + \frac{1}{2}\int_0^1 e^{x^2}\,dx$$
$$= \int_0^1 e^{x^2}\,dx = \theta \quad \text{(shown)}$$

b) To show $Var(Z) < Var(Y)$

Y is already assumed to be an estimator of θ

$$Var(Z) = Var\left(e^{U^2} + e^{(1-U)^2}/2\right)$$
$$= \frac{1}{4} Var\left(e^{U^2} + e^{(1-U)^2}\right)$$
$$= \frac{1}{4}\left[Var(e^{U^2}) + Var(e^{(1-U)^2}) + \underbrace{2cov(e^{U^2}, e^{(1-U)^2})}_{\text{negative}}\right]$$

$$Var(Y) = Var\left(e^{U_1^2} + e^{U_2^2}/2\right)$$
$$= \frac{1}{4}\left[Var(e^{U_1^2} + e^{U_2^2})\right]$$
$$= \frac{1}{4}\left[Var(e^{U_1^2}) + Var(e^{U_2^2}) + \underbrace{2cov(e^{U_1^2}, e^{U_2^2})}_{\text{zero}}\right]$$

As $cov(e^{U_1^2}, e^{U_2^2}) = 0 \geq cov(e^{U^2}, e^{(1-U)^2})$
(because independent) as e^{U^2} and $e^{(1-U)^2}$ is negatively correlated.

$\therefore Var(Z) < Var(Y)$ and Z is a more efficient estimator than Y.

QUESTION 5

A better control variate will minimise $Var(X_c)$ and its value is given by

$$Var(X_c) = Var(X) - \frac{Cov(X,Y)^2}{Var(Y)}$$

We want a $(Cor(X,Y))^2$ with a larger absolute value.

Objective: Determine which is higher - $(Cor(\theta,U^2))^2$ or $(Cor(\theta,U))^2$

We will show that $(Cor(\theta,U^2))^2 - (Cor(\theta,U))^2$ is positive with more than 95% confidence.

```
cov_u1_t_arr = []
cov_u2_t_arr = []

for _ in range(100):
    unif_arr = np.random.uniform(size=(1000))
    unif_arr_2 = unif_arr**2
    theta_arr = np.sqrt(1-unif_arr_2)

    cov_u1_t_arr.append(np.cov(unif_arr, theta_arr)[0][1])
    cov_u2_t_arr.append(np.cov(unif_arr_2, theta_arr)[0][1])

cov2_u1_t_arr = np.array(cov_u1_t_arr)**2
cov2_u2_t_arr = np.array(cov_u2_t_arr)**2
cov2_diff = cov2_u2_t_arr - cov2_u1_t_arr
```

```
cov2_diff_mean, cov2_diff_conf = mean_confidence_interval(cov2_diff)
print("Interval : ({},{})".format(cov2_diff_mean - cov2_diff_conf,
                                  cov2_diff_mean + cov2_diff_conf))
```

```
Interval : (0.0007573445149785855,0.0007989088706777195)
```

$(cov(\theta,U^2))^2 - (cov(\theta,U))^2$ is positive with more than 95% confidence.

Therefore it is shown that $(cov(\theta,U^2))^2 > (cov(\theta,U))^2$

It is better to use U^2 as a control variate than U.