Stochastic Simulation (MIE1613H) - Homework 3 (Solutions)

Due: March 21, 2021

Problem 1. (20 Pts.) Exercise 16 from Chapter 4 of the textbook.

Solution. The current system is comprised of two independent $M/G/s$ queues. The queue for financial customers has Poisson arrivals with rate $\lambda_1 = 0.59$ per minute, Erlang-2 service times with mean $1/\mu_1 = 5$ minutes, and $s_1 = 4$ servers. The queue for contact management calls has Poisson arrivals with rate $\lambda_2 = 0.41$ per minute, Erlang-3 service times $1/\mu_2 = 5$ minutes and $s_2 = 3$. Simulating the system for $n = 1000$ replications with run length $T = 8$ hours (480 minutes) we have the following estimates for the expected waiting time of each customer class:

$$W_1 = 1.71 \pm 0.06$$
$$W_2 = 1.60 \pm 0.05$$

The conceptual system has s servers that can attend to both customer classes. The service times have the same distribution but the average is now 5.25 minutes. Note that to simulate this system we need to distinguish between the two customer classes when scheduling service completions. One way to do this is to define a new attribute for the Entity object, say Type, so that the customer types are known when removing the next customer from the queue. The estimated waiting times for each customer class based on $n = 1000$ replications and with $s = 7$ (same number of servers) are as follows:

$$W_1 = 0.83 \pm 0.03$$
$$W_2 = 0.83 \pm 0.03$$

With $s = 6$ server we get

$$W_1 = 3.01 \pm 0.13$$
$$W_2 = 3.00 \pm 0.13$$

Results suggest that the cross-trained system will have an improved performance with respect to the expected waiting time of both customer classes under the same number of servers (i.e., 7) but reducing the number of servers (e.g., to 6) will lead to a worse performance compared to the current system.

The code is available on Quercus.

Problem 2. (15 Pts.) Exercise 16 from Chapter 6 of the textbook. **Note:** Data file (Call-Counts.xls) is available on Quercus.

Solution. The arrival rates can be estimated by averaging the call counts for each hour :

t	1	2	3	4	5	6	7	8
$\hat{\lambda}(t)$	30.194	38.774	58.774	79.484	72.710	39.419	41.226	20.032

The piecewise-constant arrival rate function then can be used together with the thinning method to generate the NSPP. Note that the inter-arrival times are no longer exponentially distributed and hence generating exponential inter-arrival times with time-dependent rate is NOT a valid approach.

The current system has estimated performance (based on $n = 1000$ replications:)

$$W_1 = 2.66 \pm 0.14$$
$$W_2 = 2.14 \pm 0.12$$

For the system with $s = 7$ cross-trained servers the estimated waiting times are:

$$W_1 = 1.83 \pm 0.10$$
$$W_2 = 1.80 \pm 0.10$$

With $s = 6$ server we get

$$W_1 = 5.90 \pm 0.24$$
$$W_2 = 5.86 \pm 0.24$$

So the conclusion remains unaffected.

The code is available on Quercus.

Problem 3. (15 Pts.) Exercise 17 from Chapter 6 of the textbook. **Note:** For this question you may use Python, R, or Excel, but make sure to explain your computations and report the data used to fit the regression.

Solution. For this problem we first need to compute $C_j(t)$, the cumulative number of arrivals by time t for each day j and each hour $t = 1, 2, ..., 8$. Next compute the mean and variance of $C_j(t)$, $j = 1, 2, ..., 31$, for each t (see the table below). Finally, run a regression through the origin on these 8 values. That is, fit the model

$$S^2_{C(t)} = \beta \bar{C}(t) + \varepsilon(t),\ t = 1, 2, ..., 8.$$

The results is $\beta = 1.218987203$, which is reasonably close to 1, with an $R^2 = 0.93$. This supports an NSPP model.

$\bar{C}(t)$	$S^2_{C(t)}$
30.19354839	28.62795699
68.96774194	46.36559140
127.7419355	139.2645161
207.2258065	297.5139785
279.9354839	393.0623656
319.3548387	445.7032258
360.5806452	394.3849462
380.6129032	409.8451613

Problem 4. Consider the discrete-time stochastic process $S_n = \sum_{i=1}^{n} X_i$, for $n = 1, 2, 3, \ldots$, where X_i's are iid random variables with CDF,

$$F(x) = 1 - \left(\frac{2}{x}\right)^3.$$

(a) (5 Pts.) Propose an inversion algorithm to generate samples of X_i's.

Solution. We denote the inverse cdf by $F^{-1}(.)$. So,

$$F(x) = u \iff x = F^{-1}(u).$$

We have

$$1 - \left(\frac{2}{x}\right)^3 = u \implies \left(\frac{2}{x}\right)^3 = 1 - u \implies \frac{2}{x} = (1-u)^{1/3} \implies x = \frac{2}{(1-u)^{1/3}}$$

Therefore,

$$F^{-1}(u) = \frac{2}{(1-u)^{1/3}}$$

So, the inversion algorithm works as follows:

1. Generate $U \sim U[0, 1]$.
2. Set $X = \frac{2}{(1-U)^{1/3}}$ and return X.

(b) (5 Pts.) Propose an unbiased estimator for $P(S_{10} > 40)$. Provide an estimate and a 95% CI using 10^6 samples.

Solution. We generate $m = 10^6$ samples of S_{10} and propose $\frac{1}{m}\sum_{i=1}^{m} I(S_{10} > 40)$, i.e., sample mean, as the unbiased estimator of $P(S_{10} > 40)$. To generate a sample of S_{10}, we have to first generate 10 samples of X using the inversion algorithm given in part (a) and then add them up. The estimated value of $P(S_{10} > 40)$ is 0.0421 with the 95% CI of [0.0417, 0.0425].

```
1  import numpy as np
2
3  def CI_95(data):
4      a = np.array(data)
5      n = len(a)
6      m = np.mean(a)
7      var = ((np.std(a)) ** 2) * (n / (n - 1))
8      hw = 1.96 * np.sqrt(var / n)
9      return m, [m - hw, m + hw]
10
11 n = 10
12 Indicator = []  # Indicator function
13 np.random.seed(1)
14 for rep in range(1000000):
15     S = 0
16     for i in range(n):
17         U = np.random.random()
18         X = 2 / ((1 - U) ** (1 / 3))
```

```
        S += X
    if (S > 40):
        Indicator.append(1)
    else:
        Indicator.append(0)

print("Estimate and 95% CI:", CI_95(Indicator))
```

```
Estimate and 95% CI: (0.042135, [0.04174124150182047, 0.042528758498179525])
```

Problem 5. In Ontario, when a patient requires an MRI image, the physician refers the patient to an imaging facility. The facility then schedules an appointment according to the urgency level of the patient. Consider a facility with a single MRI machine. Appointments are given at the beginning of each hour, starting 7AM and over the course of a 12 hour working day until 7PM. That is, the first appointment is at 7AM, the second at 8AM and so on. Patients are however not always punctual and may arrive before or after their scheduled appointment. Imaging time for each patient takes on average around 50 minutes but involves some variability. If there are still patients in the facility at the end of the day, the facility remains open until all patients with appointments on that day are served. Assume all patients show up for their appointments. Assume patients are served in order of arrival.

The attached data file (Problem5_Data.xlsx) provides historical samples of imaging durations (time it took to conduct the imaging) and how much each patient's arrival deviated from the original appointment time. For example, a sample value of 3 in the "Deviation from appointment" column indicates that the patient was 3 minutes late for their appointment and a sample value of -5 indicates that they were 5 minutes early.

(a) (6 Pts.) Fit an appropriate distribution to the "Imaging Duration" and "Deviation from appointment" data. Justify your choice of the distributions.

Solution. Since we do not have a physical justification for the choice of distributions, we can speculate a few candidate distributions (e.g., by looking at the shape of the histogram) in each case and examine the goodness of fit for them.

For the "Imaging Duration" data, we pick Lognormal, Weibull, and Gamma. The KS test does not reject any of the distributions at $(1-\alpha) = 95\%$ confidence level. Examining the Q-Q plot however suggests that Gamma provides a better fit at the tail.

For the "Deviation from appointment" data, we pick Normal distribution by looking at the shape of the empirical distribution. The KS test does not reject the Normal distribution. Examining the Q-Q plot we observe that this fit provides an acceptable choice.

```
# This file uses the fitdistrplus package. You'll need to install it first
# by running the command: 'install.packages("fitdistrplus")' in R
require(fitdistrplus)

# import the data from .csv file
Data <- read.csv("Problem5_Data.csv")


plotdist(Data$Imaging.duration, histo = TRUE, demp = TRUE)
plotdist(Data$Deviation.from.appointment, histo = TRUE, demp = TRUE)

```

```r
# fit a Lognormal, Gamma, and Weibull to the "Imaging Duration" data
ImgDur_fln <- fitdist(Data$Imaging.duration, "lnorm")
ImgDur_fw <- fitdist(Data$Imaging.duration, "weibull")
ImgDur_fg <- fitdist(Data$Imaging.duration, "gamma")

# fit a Normal distribution to the "Deviation from appointment" data
Dev_fn <- fitdist(Data$Deviation.from.appointment, "norm")


# compare the fits for imaging duration
plot.legend <- c("lognormal", "weibull", "gamma")
qqcomp(list(ImgDur_fln, ImgDur_fw, ImgDur_fg), legendtext = plot.legend)

# fit for deviation
plot.legend <- c("normal")
qqcomp(list(Dev_fn), legendtext = plot.legend)


# Perform GofFit tests for imaging duration
gof_load <- gofstat(list(ImgDur_fln, ImgDur_fw, ImgDur_fg), fitnames = c("
    lognormal", "weibull", "gamma"))
gof_load
gof_load$kstest


# Perform GofFit tests for deviation
gof_load <- gofstat(Dev_fn, fitnames = c("normal"))
gof_load
gof_load$kstest

# Estimation of the parameters of Gamma
ImgDur_fg

# Estimation of the parameters of Normal
Dev_fn
```

Fitting of the distribution ' gamma ' by maximum likelihood Parameters:

	estimate	Std. Error
shape	3.1371857	0.188625853
rate	0.0612125	0.003990131

Fitting of the distribution ' norm ' by maximum likelihood Parameters:

	estimate	Std. Error
mean	4.931862	0.2188955
sd	4.894653	0.1547825

(b) (4 Pts.) Explain, in words, how you would simulate arrivals driven by appointments, but subject to early or late arrivals according to the fitted distribution in Part (a).

Solution. We know appointments are given at the beginning of each hour over the course of a 12-hour working day. So, we schedule the arrival events for 12 patients to the calendar at the beginning of each replication such that the (random) deviation from appointment for each patient is sampled from the fitted Normal distribution with mean 4.93 and standard deviation 4.89.

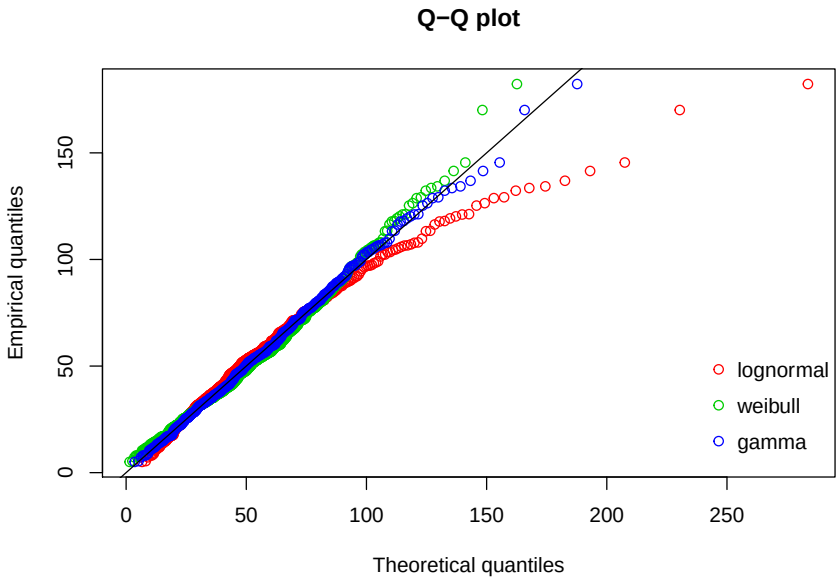

Figure 1: Q-Q plot for the fitted distributions to the "Imaging Duration" data.

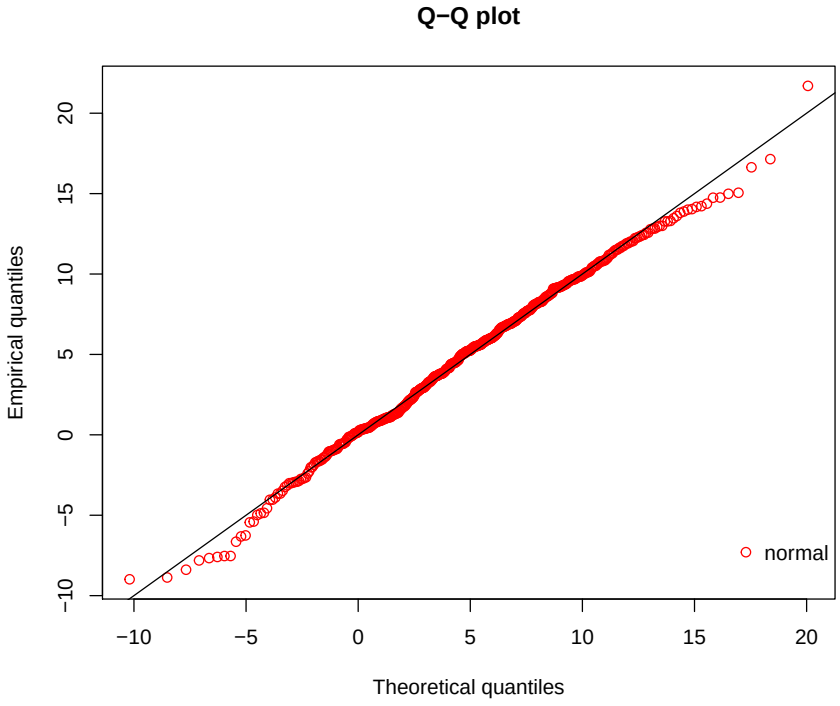

Figure 2: Q-Q plot for the Normal distribution fitted to the "Deviation from appointment" data.

(c) (10 Pts.) Develop a simulation model of the imaging facility using the distributions you obtained in part (a). Estimate (1) the expected average waiting time of patients during the day, and (2) the expected time it takes until all scheduled patients are served. Report a 95% CI for both estimates.

Solution. We first schedule the arrival events for all 12 patients to the calendar at the beginning of each replication given that the actual arrival times deviate from appointments at the beginning of each hour according to the fitted Normal distribution. We also create a variable (count) to count the number of patients served and use it to terminate our simulation. To compute the waiting times of patients in the imaging facility during the day, we first add patients to the queue when they arrive and then check the status of the MRI machine (i.e., idle or busy). If the MRI machine is idle, we seize it to serve the patient and schedule the end of the service using the fitted Gamma distribution. For the first arrival, if the patient arrives early before 7AM, we first set the clock to 7AM and then schedule the end of the service. We also create a list (TotalTime) to keep the time it takes until all scheduled patients are served (when simulation ends) in each replication. Using 1000 replications, the estimated expected average waiting time in the imaging facility is 1.17 hour with the 95% CI of [1.14, 1.20]. The estimated expected time it takes until all scheduled patients are served is 12.38 hours with the 95% CI of [12.33, 12.43].

```
# Imaging facility
import SimFunctions
import SimRNG
import SimClasses
import numpy as np
import scipy.stats as stats


def CI_95(data):
    a = np.array(data)
    n = len(a)
    m = np.mean(a)
    var = ((np.std(a))**2)*(n/(n-1))
    hw = 1.96*np.sqrt(var/n)
    return m, [m-hw,m+hw]


Clock = 0.0
ZSimRNG = SimRNG.InitializeRNSeed()
np.random.seed(1)

Queue = SimClasses.FIFOQueue()
Wait = SimClasses.DTStat()
Server = SimClasses.Resource()
Calendar = SimClasses.EventCalendar()

TheCTStats = []
TheDTStats = []
TheQueues = []
TheResources = []

TheDTStats.append(Wait)
TheQueues.append(Queue)
TheResources.append(Server)
```

```
35
36 Server.SetUnits(1)
37 MeanDev = 4.931862
38 VarDev = 4.894653**2
39 ShapeST = 3.1371857
40 ScaleST = 1/0.0612125
41 AllWaitMean = []
42 TotalTime = []
43
44
45 def Arrival():
46     Patient = SimClasses.Entity()
47     Queue.Add(Patient)
48     if Server.Busy == 0:
49         if SimClasses.Clock < 7:
50             SimClasses.Clock = 7
51             Server.Seize(1)
52             SimFunctions.Schedule(Calendar, "EndOfService", np.random.gamma(
                   shape=ShapeST, scale=ScaleST) / 60)
53         else:
54             Server.Seize(1)
55             SimFunctions.Schedule(Calendar, "EndOfService", np.random.gamma(
                   shape=ShapeST, scale=ScaleST) / 60)
56
57
58 def EndOfService():
59     global  count
60     count += 1
61     DepartingPatient = Queue.Remove()
62     Wait.Record(SimClasses.Clock - DepartingPatient.CreateTime)
63     if Queue.NumQueue() > 0:
64         SimFunctions.Schedule(Calendar, "EndOfService", np.random.gamma(shape
               = ShapeST, scale = ScaleST)/60)
65     else:
66         Server.Free(1)
67
68
69 for reps in range(1000):
70     SimFunctions.SimFunctionsInit(Calendar, TheQueues, TheCTStats, TheDTStats,
           TheResources)
71
72     # Schedule the arrival events for all 12 patients
73     for i in range(12):
74         SimFunctions.Schedule(Calendar, "Arrival", 7 + i + SimRNG.Normal(
               MeanDev, VarDev, 2)/60)
75
76     # To keep track of the number of patients served
77     count = 0
78     while count < 12:
79         NextEvent = Calendar.Remove()
80         SimClasses.Clock = NextEvent.EventTime
81         if NextEvent.EventType == "Arrival":
82             Arrival()
83         elif NextEvent.EventType == "EndOfService":
```

```
84                 EndOfService()
85
86         TotalTime.append(SimClasses.Clock - 7)
87         AllWaitMean.append(Wait.Mean())
88
89 # output results
90 print("Estimated expected average waiting time in the imaging facility:",
       CI_95(AllWaitMean))
91 print("Estimated expected time to serve all patients:", CI_95(TotalTime))
```

```
Estimated expected average waiting time in the imaging facility:
(1.1750341874352785, [1.1454852106163853, 1.2045831642541718])
Estimated expected time to serve all patients:
(12.384215255174333, [12.331849007923221, 12.436581502425444])
```

Problem 6. In your analysis of arrival data for a system that you are planning to simulate, you have estimated $Var(N(t))/E(N(t))$ to be approximately 1.6 for all $t \geq 0$. Since the estimate is significantly greater than 1, you have concluded that a non-homogeneous Poisson process is not a good choice and therefore you need a non-exponential base distribution with variance $\sigma_A^2 = 1.6$ to use with the inversion method for generating arrivals in your simulation model. As discussed in Exercise 35 of Chapter 6, one option for the base distribution is a balanced hyperexponential distribution; that is $\tilde{A}_n$ for $n = 2, 3, \ldots$, is exponentially distributed with rate λ_1 with probability p, and exponentially distributed with rate λ_2 with probability $(1-p)$. "Balanced" means that $p/\lambda_1 = (1-p)/\lambda_2$. Therefore, there are only two free parameters; λ_1 and p.

(a) (**5 Pts.**) Show that

$$p = \frac{1}{2}\left(1 + \sqrt{\frac{\sigma_A^2 - 1}{\sigma_A^2 + 1}}\right), \quad \lambda_1 = 2p,$$

achieves the desired arrival rate and variance. **HINT**: The first and second moments of the hyperexponential random variable are given by $p(1/\lambda_1) + (1-p)(1/\lambda_2)$ and $p(2/\lambda_1^2) + (1-p)(2/\lambda_2^2)$, respectivelly.

Solution. We need to verify that the mean and variance of the hyperexponential random variable with the above parameters, denoted by $\tilde{A}$, is respectivelly 1 and σ_A^2. The mean is the weighted sum of the mean of each exponential, that is,

$$E[\tilde{A}] = p(1/\lambda_1) + (1-p)(1/\lambda_2) = 2p/\lambda_1 = 2p/2p = 1.$$

Similarly, the second moment can be obtained as the weighted sum of the second moments of each exponential. Recalling that the second moment of an exponential random variable with rate λ is $2/\lambda^2$ and using $\lambda_1 = 2p$ we have,

$$\begin{aligned} E[\tilde{A}^2] &= \frac{2p}{\lambda_1^2} + \frac{2(1-p)}{\lambda_2^2} = \frac{2p}{4p^2} + \frac{2(1-p)}{\lambda_2}\frac{1}{\lambda_2} \\ &= \frac{1}{2p} + \frac{2p}{\lambda_1}\frac{p}{\lambda_1(1-p)} \\ &= \frac{1}{2p} + \frac{2p^2}{4p^2(1-p)} \\ &= \frac{1}{2p} + \frac{1}{2(1-p)} = \frac{1}{2p(1-p)}. \end{aligned}$$

Substituting p from above we have,

$$\begin{aligned} p(1-p) &= \frac{1}{2}\left(1+\sqrt{\frac{\sigma_A^2-1}{\sigma_A^2+1}}\right)\left(1-\frac{1}{2}\left(1+\sqrt{\frac{\sigma_A^2-1}{\sigma_A^2+1}}\right)\right) \\ &= \left(\frac{1}{2}+\frac{1}{2}\sqrt{\frac{\sigma_A^2-1}{\sigma_A^2+1}}\right)\left(\frac{1}{2}-\frac{1}{2}\sqrt{\frac{\sigma_A^2-1}{\sigma_A^2+1}}\right) \\ &= \frac{1}{4}\left(1-\frac{\sigma_A^2-1}{\sigma_A^2+1}\right) = \frac{1}{2}\left(\frac{1}{\sigma_A^2+1}\right), \end{aligned}$$

It follows that the variance is given by,

$$Var[\tilde{A}] = E[\tilde{A}^2] - E[\tilde{A}] = \sigma_A^2 + 1 - 1 = \sigma_A^2.$$

(b) (5 Pts.) Show that $G_e(t)$ or the distribution of the first inter-arrival time $\tilde{A}_1$ for the equilibrium renewal process is also hyperexponential but with $p = (1-p) = 1/2$, that is

$$G_e(t) = \frac{1}{2}(1-e^{-\lambda_1 t}) + \frac{1}{2}(1-e^{-\lambda_2 t}).$$

Solution. We have,

$$1 - G(t) = pe^{-\lambda_1 t} + (1-p)e^{-\lambda_2 t},$$

and therefore,

$$\begin{aligned} G_e(t) &= \tilde{\lambda}\int_0^t (1-G(s))ds \\ &= \int_0^t pe^{-\lambda_1 s}ds + \int_0^t (1-p)e^{-\lambda_2 s}ds \\ &= \frac{p}{\lambda_1}(1-e^{-\lambda_1 t}) + \frac{1-p}{\lambda_2}(1-e^{-\lambda_2 t}) \\ &= \frac{1}{2}(1-e^{-\lambda_1 t}) + \frac{1}{2}(1-e^{-\lambda_2 t}), \end{aligned}$$

where we have used $\tilde{\lambda} = 1$ and $p/\lambda_1 = (1-p)/\lambda_2$. Note that the above implies that the initial inter-arrival time is equally likely to be exponential with rate λ_1 or λ_2. Therefore, it has a hyperexponential distribution with $p = (1-p) = 1/2$.

(c) (5 Pts.) Propose a method for generating samples from the hyperexponential distribution. **HINT**: Remember the definition of the Hyperexponential distribution.

Solution. To generate a sample from a hyperexponential distribution, one can directly use the defenition. In the genertal case, consider a hyperexponential distribution with parameters p, λ_1, λ_2. That is, the random variable is exponentially distributed with rate λ_1, with probability p, and exponentially distributed with rate λ_2, with probability $(1-p)$. Then one can generate a sample X as follows:

- Generate $U_1 \sim Unif[0,1]$ and $U_2 \sim Unif[0,1]$
- If $U_1 \leq p$: Return $X = \frac{-1}{\lambda_1}\ln(U_2)$; else: Return $X = \frac{-1}{\lambda_2}\ln(U_2)$.

Note that the above returns an exponential with rate λ_1 with probability p, and an exponential with rate λ_2 with probability $(1-p)$.

(d) (5 Pts.) Assuming $\Lambda(t) = e^t$, write a program in Python that generates customer arrival times for 10 time units. Report the arrival times for one realization.

Solution. We apply the inversion method with the equilibrium renewal arrival process obtained above. The inter-arrival times $A_2, A_3, \ldots$ are generated from the balanced hyper-exponential specified in the question and with parameters:

$$\begin{aligned} p &= \frac{1}{2}\left(1 + \sqrt{\frac{1.6-1}{1.6+1}}\right) = 0.74, \\ \lambda_1 &= 2p = 1.48, \\ \lambda_2 &= (1-p)\lambda_1/p = 2(1-p) = 0.52. \end{aligned}$$

The first inter-arrival time A_1 is generated from a hyper-exponential with

$$\begin{aligned} p &= 0.5, \\ \lambda_1 &= 1.48, \\ \lambda_2 &= 0.52. \end{aligned}$$

If negative values are generated after taking the *log* of unit-rate arrival times, we can take the first arrival as the one after time 0.

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

# the function returns a random variate
# from hyperexponential(q,l1,l2)

def HypExp(q,l1,l2):
    U1 = np.random.random()
    U2 = np.random.random()
    if U1 < q:
        return (-1/l1)*np.log(U2)
    else:
        return (-1/l2)*np.log(U2)

np.random.seed(1)

# set parameters of the renewal process
p = 0.74
lambda1 = 2*p
lambda2 = 2*(1-p)

# list to keep arrival times
Arrival_times = []
# S denotes the actual arrival times
S = 0
# S_tilde denotes arrival times of the unit rate process
# generate the first arrival using Ge(t)
S_tilde = HypExp(1/2,lambda1,lambda2)
```

```
29 S = np.log(S_tilde)
30 Arrival_times.append(S)
31
32 # generate arrivals during [0,10]
33 while S < 10:
34     # generate the next arrival using G(t)
35     S_tilde = S_tilde + HypExp(p,lambda1,lambda2)
36     S = np.log(S_tilde)
37     Arrival_times.append(S)
38
39 print(Arrival_times)
```

```
[-1.5066207132402623, 0.029480812323000174, 0.9706187376724378, 1.2112089417814977, 1.3284971847790814,
1.393960326852811, 1.4155139318525725, 1.4790385806402864, 1.5648747950091304, 1.770872638819192,
1.7813730065545463, 1.8223617182802363, 1.856395975396509, 2.1507967845269054, 2.160964271603683,
2.226122158546919, 2.3488331098579542, 2.420613879634465, 2.4314098987799535, 2.448342766448462,
2.4954427820418723, 2.508541019814889, 2.551762252233947, 2.7204004384886074, 2.8072788359622742,
2.8229563414497334, 2.8748318736323255, 2.9807511447097483, 3.044491372747837,...]
```