Stochastic Simulation (MIE1613H) - Homework 2 (Solutions)

Due: Feb 19th, 2019

Problem 1. (Chapter 4, Exercise 8) In the simulation of the Asian option, the sample mean of 10,000 replications was 2.198270479 and the standard deviation was 4.770393202. *Approximately* how many replications would it take to decrease the reletive error to less than 1%?

Note: The relative error of a sample mean is the standard error divided by the mean.

(15 Points) Denote the expected value of the option by $E[\nu]$ and its standard deviaition by σ. We would like to set n such that

$$\frac{\sigma}{E[\nu]\sqrt{n}} < 0.01.$$

From the simulation experiments we have the estimates $E[\nu] \approx 2.198270479$, and $\sigma \approx 4.770393202$. Therefore, we need to find the smallest n such that

$$\frac{4.770393202}{2.198270479} < 0.01\sqrt{n},$$

which yields $n = 47092$.

Problem 2. (Down-and-in call option) Another variation of European options are barrier options. For a down-and-in call option, if we denote the stock price at time t by $X(t)$, the holder of the option receives payoff $(X(T) - K)^+$ at maturity time T only if the stock price has crossed below some barrier $b < X(0)$ before time T. Assume that the stock price evolves according to a Geometric Brownian Motion (GBM) with drift parameter $\mu = r = 0.05$ and volatility parameter $\sigma = 0.4$.

(a) Using simulation, estimate the expected payoff of a down-and-in call option for $T = 1$ year, assuming the initial price of $X(0) = 95$; strike price of $K = 100$ and barrier $b = 90$. Use 10,000 replications, and 64 steps when discretizing the GBM. Report a 95% confidence interval for the estimate.

(15 Points) The payoff of the barrier option is path dependent in that the payoff is $e^{-rT}(X(T) - K)^+$ only if the path has crossed the barrier b, and 0 otherwise, that is the expected payoff is given by

$$E[e^{-rT}(X(T) - K)^+ \mathbf{1}\{\min_{0\le t\le T} X(t) < b\}].$$

To estimate the expected payoff, we define a binary variable 'active' initialized with value 0 and check whether the price has crossed the barrier everytime we generate a new point on the sample path. If the barrier is crossed, we set the value of the variable 'active' to 1. When all the points are generated we compute the payoff depending on the value of 'active'. The estimated cost is the sample average of the payoff across all replications. The estimated 95% CI is [7.62,8.40].

(b) Compare the estimated expected payoff with that of a standard European call option (for the same parameters). Provide an intuitive explanation for your observation.

(5 Points) For the European option the payoff is simply $e^{-rT}(X(T) - K)^+$ regardless of the path. The estimated 95% CI for the average payoff is [14.32,15.44]. The barrier option has a lower estimated expected payoff (or is cheaper) since it can only generate a positive payoff if the price has gone below a certain value which in turn reduces the probability of having a positive final payoff.

```python
import SimRNG
import math
import pandas
import numpy as np

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]

ZRNG = SimRNG.InitializeRNSeed()
Replications = 10000
Maturity = 1.0
Steps = 64
Sigma = 0.4
InterestRate = 0.05
InitialValue = 95.0
StrikePrice = 100.0
Interval = Maturity / Steps
Sigma2 = Sigma * Sigma / 2
# barrier threshold
b = 90

TotalValue = []
TotalValue_European = []

for i in range(0,Replications,1):
    X = InitialValue
    # binary variable; set to 1 if barrier is crossed
    active = 0
    for j in range(0,Steps,1):
        Z = SimRNG.Normal(0,1,12)
        X = X * math.exp((InterestRate - Sigma2) * Interval + Sigma * math.sqrt(Interval) * Z)
        if X < b:
                active = 1
    Value = math.exp(-InterestRate * Maturity) * max(X - StrikePrice, 0)
    if active == 1:
        TotalValue.append(Value)
    else:
        TotalValue.append(0)
    TotalValue_European.append(Value)

print ("Barrier option:",CI_95(TotalValue))
print ("European option:",CI_95(TotalValue_European))
```

```
Barrier option:  (8.009622998674828, [7.616950885697123, 8.402295111652533])
European option:  (14.881710245205332, [14.31974006547141, 15.443680424939252])
```

Problem 3. (Chapter 4, Exercise 4) Beginning with the PythonSim event-based $M/G/1$ simulation, implement the changes necessary to make it an $M/G/s$ simulation (a single queue with any number of servers). Keeping $\lambda = 1$ and $\tau/s = 0.8$, simulate the system for $s = 1, 2, 3$ and (a) report the estimated expected number of customers in the system (including customers in the queue and service), expected system time, and expected number of busy servers in each case. (b) Compare the results and state clearly what you observe. What you're doing is comparing queues with the same service capacity, but with 1 fast server as compared to 2 or more slow servers.

The modifications are as follows: The number of servers is set using variable 'ServerNum'. Upon arrival of a customer we schedule a departure if there is an idle server (Server.Busy < ServerNum) in which case the customer can immediatly start service. When an EndofService event occurs, we schedule the next EndofService if (Queue.NumQueue() >= ServerNum) i.e. if there is a customer waiting. Note that Queue here includes both the waiting customers and those in service.

(a) **(15 Points)** The estimates are summarized below:

	$s = 1$	$s = 2$	$s = 3$
Expected Average wait	2.93	3.51	4.15
Expected Average queue-length	2.93	3.51	4.15
Expected Average # of servers busy	0.80	1.60	2.40

(b) **(5 Points)** We observe that as the number of servers increases and the service rate decreases, the expected number of customers and the total time in system increases. The average number of busy servers is equal to $0.8s$ so the average utilization of the servers remains constant. Note that with $s = 1$ the output rate of the system is $1/0.8s = 1/0.8$ as long as there are customers in the system, while when $s > 1$ the maximum output rate of $1/0.8$ is only achieved when all servers are busy. In other words, when there are less than s customers in the system, the system *slows down* and hence on average there are more customers in the system.

```
import SimFunctions
import SimRNG
import SimClasses
import numpy as np
import scipy.stats as stats

def t_mean_confidence_interval(data,alpha):
    a = 1.0*np.array(data)
    n = len(a)
    m, se = np.mean(a), stats.sem(a)
    h = stats.t.ppf(1-alpha/2, n-1)*se
    return m, m-h, m+h

ZSimRNG = SimRNG.InitializeRNSeed()

Queue = SimClasses.FIFOQueue()
Wait = SimClasses.DTStat()
Server = SimClasses.Resource()
```

```python
Calendar = SimClasses.EventCalendar()

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

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

ServerNum = 2
Server.SetUnits(ServerNum)
MeanTBA = 1.0
MeanST = 0.8 * ServerNum
Phases = 3
RunLength = 55000.0
WarmUp = 5000.0

AllWaitMean = []
AllQueueMean = []
AllQueueNum = []
AllServerBusyMean = []
print ("Rep", "Average Wait", "Average Number in Queue", "Number Remaining in Queue", "Average Server Busy")

def Arrival():
    SimFunctions.Schedule(Calendar,"Arrival",SimRNG.Expon(MeanTBA, 1))
    Customer = SimClasses.Entity()
    Queue.Add(Customer)

    if Server.Busy < ServerNum:
        Server.Seize(1)
        SimFunctions.Schedule(Calendar,"EndOfService",SimRNG.Erlang(Phases,MeanST,2))

def EndOfService():
    DepartingCustomer = Queue.Remove()
    Wait.Record(SimClasses.Clock - DepartingCustomer.CreateTime)
    # if there are customers waiting
    if Queue.NumQueue() > ServerNum:
        SimFunctions.Schedule(Calendar,"EndOfService",SimRNG.Erlang(Phases,MeanST,2))
    else:
        Server.Free(1)

for reps in range(0,10,1):

    SimFunctions.SimFunctionsInit(Calendar,TheQueues,TheCTStats,TheDTStats,TheResources)
    SimFunctions.Schedule(Calendar,"Arrival",SimRNG.Expon(MeanTBA, 1))
    SimFunctions.Schedule(Calendar,"EndSimulation",RunLength)
    SimFunctions.Schedule(Calendar,"ClearIt",WarmUp)

```

```
        NextEvent = Calendar.Remove()
        SimClasses.Clock = NextEvent.EventTime
        if NextEvent.EventType == "Arrival":
            Arrival()
        elif NextEvent.EventType == "EndOfService":
            EndOfService()
        elif NextEvent.EventType == "ClearIt":
            SimFunctions.ClearStats(TheCTStats,TheDTStats)

        while NextEvent.EventType != "EndSimulation":
            NextEvent = Calendar.Remove()
            SimClasses.Clock = NextEvent.EventTime
            if NextEvent.EventType == "Arrival":
                Arrival()
            elif NextEvent.EventType == "EndOfService":
                EndOfService()
            elif NextEvent.EventType == "ClearIt":
                SimFunctions.ClearStats(TheCTStats,TheDTStats)


        AllWaitMean.append(Wait.Mean())
        AllQueueMean.append(Queue.Mean())
        AllQueueNum.append(Queue.NumQueue())
        AllServerBusyMean.append(Server.Mean())
        print (reps+1, Wait.Mean(), Queue.Mean(), Queue.NumQueue(), Server.Mean())

# output results
print("Estimated Expected Average wait:",t_mean_confidence_interval(
    AllWaitMean,0.05))
print("Estimated Expected Average queue-length:", t_mean_confidence_interval(
    AllQueueMean,0.05))
print("Estimated Expected Average # of servers busy:",
    t_mean_confidence_interval(AllServerBusyMean,0.05))
```

Problem 4. (Chapter 4, Exercise 5) Modify the PythonSim event-based simulation of the $M/G/1$ queue to simulate a $M/G/1/c$ retrial queue. This means that customers who arrive to find c customers in the system (including the customer in service) leave immediately, but arrive again after an exponentially distributed amount of time with mean MeanTR. (You do not need to report any outputs for this problem.)

(15 Points) To model retrials we create another event called "Retrial" with logic similar to the "Arrival" event except that unlike in the Arrival event we do not schedule the next arrival everytime a Retrial event occurs. Arriving customers who find the system with c or more customers, regardless of whether they are original arrivals or retrials, schedule a Retrial event in the calendar.

```
# M/G/1/c queue
import SimFunctions
import SimRNG
import SimClasses
import numpy as np
import scipy.stats as stats

```

```
def t_mean_confidence_interval(data,alpha):
    a = 1.0*np.array(data)
    n = len(a)
    m, se = np.mean(a), stats.sem(a)
    h = stats.t.ppf(1-alpha/2, n-1)*se
    return m, m-h, m+h

Clock = 0.0
ZSimRNG = SimRNG.InitializeRNSeed()

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)

Server.SetUnits (1)
MeanTBA = 1.0
MeanST = 0.8
Phases = 3
MeanTR = 2
RunLength = 55000.0
WarmUp = 5000.0
c = 3

AllWaitMean = []
AllQueueMean = []
AllServerMean = []

def Arrival():
    SimFunctions.Schedule(Calendar,"Arrival",SimRNG.Expon(MeanTBA, 1))
    Customer = SimClasses.Entity()
    if Queue.NumQueue() == c:
        SimFunctions.Schedule(Calendar, "Retrial", SimRNG.Expon(MeanTR, 1))
    elif Server.Busy == 0:
        Queue.Add(Customer)
        Server.Seize(1)
        SimFunctions.Schedule(Calendar,"EndOfService",SimRNG.Erlang(Phases,MeanST,2))
    else:
        Queue.Add(Customer)

def Retrial():
    Customer = SimClasses.Entity()
    if Queue.NumQueue() == c:
        SimFunctions.Schedule(Calendar, "Retrial", SimRNG.Expon(MeanTR, 1))
```

```python
    elif Server.Busy == 0:
        Queue.Add(Customer)
        Server.Seize(1)
        SimFunctions.Schedule(Calendar,"EndOfService",SimRNG.Erlang(Phases,MeanST,2))
    else:
        Queue.Add(Customer)

def EndOfService():
    DepartingCustomer = Queue.Remove()
    Wait.Record(SimClasses.Clock - DepartingCustomer.CreateTime)
    if Queue.NumQueue() > 0:
        SimFunctions.Schedule(Calendar,"EndOfService",SimRNG.Erlang(Phases,MeanST,2))
    else:
        Server.Free(1)

for reps in range(0,10,1):
    SimFunctions.SimFunctionsInit(Calendar,TheQueues,TheCTStats,TheDTStats,TheResources)

    SimFunctions.Schedule(Calendar,"Arrival",SimRNG.Expon(MeanTBA, 1))
    SimFunctions.Schedule(Calendar,"EndSimulation",RunLength)
    SimFunctions.Schedule(Calendar,"ClearIt",WarmUp)

    NextEvent = Calendar.Remove()
    SimClasses.Clock = NextEvent.EventTime
    if NextEvent.EventType == "Arrival":
        Arrival()
    elif NextEvent.EventType == "EndOfService":
        EndOfService()
    elif NextEvent.EventType == "ClearIt":
        SimFunctions.ClearStats(TheCTStats,TheDTStats)

    while NextEvent.EventType != "EndSimulation":
        NextEvent = Calendar.Remove()
        SimClasses.Clock = NextEvent.EventTime
        if NextEvent.EventType == "Arrival":
            Arrival()
        if NextEvent.EventType == "Retrial":
            Retrial()
        elif NextEvent.EventType == "EndOfService":
            EndOfService()
        elif NextEvent.EventType == "ClearIt":
            SimFunctions.ClearStats(TheCTStats,TheDTStats)

    AllWaitMean.append(Wait.Mean())
    AllQueueMean.append(Queue.Mean())
    AllServerMean.append(Server.Mean())
    print (reps+1, Wait.Mean(), Queue.Mean(), Server.Mean())

# output results
print("Estimated Expected Average wait:",t_mean_confidence_interval(AllWaitMean,0.05))
```

```
print("Estimated Expected Average queue-length:", t_mean_confidence_interval(
    AllQueueMean,0.05))
print("Estimated Expected Average # of servers busy:",
    t_mean_confidence_interval(AllServerMean,0.05))
```

Problem 5. (Two queues in tandem) Customers arrive at a two-stage service system according to a stationary Poisson process with rate $\lambda = 1$. There is only 1 server working at each stage. Each customer has to go through both stages (first stage 1 and then stage 2) before departing the system. Service times are exponentially distributed with mean $\tau_1 = 0.8$ at stage 1 and $\tau_2 = 0.7$ at stage 2. Develop a simulation model of the system using PythonSim and estimate the quantities specified below in steady-state. Simulate the system for 50,000 time units with a warmup period of 5000 time units. Use 20 replications to obtain the estimates.

(30 Points) There are three events: arrival of a new customer, service completion at stage 1, and service completion at stage 2. Service completion at stage one initiates an arrival to stage 2. (You may choose to write a seperate function for arrivals to stage 2). To collect the total system time in stage 2 we define an attribute for the entity object named `(EndService1Time):`

```
class Entity():
    # This is the generic Entity that has a single attribute CreateTime
    def __init__(self):
        # Executes with the Entity object is created to initialize variables
        # Add additional problem-specific attributes here
        self.CreateTime = Clock
        self.EndService1Time = 0
```

The value of the atrribute is set to (Clock) for each entity at the instance when the service at stage 1 is completed. We use this attribute to compute the time spent in stage 2 for each entity. Note that queue 1 and 2 contain customers that are waiting and do not include the customer in service. The outputs are reported below.

(a) The expected number of customers waiting at each stage (not including the customer in service).

Estimated Total Time 1 = 4.02

Estimated Total Time 2 = 2.33

(b) The expected total time customers spend at each stage (including both the waiting time and time spent in service).

Estimated Queue 1 length = 3.22

Estimated Queue 2 length = 1.63

(c) Expected utilization of each server.

Estimated Server 1 Util. = 0.80

Estimated Server 2 Util. = 0.70

```
import SimFunctions
import SimRNG
import SimClasses
import numpy as np
```

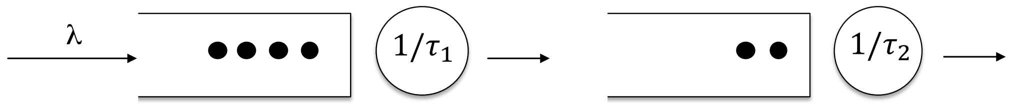

Figure 1: Tandem queue (problem 5).

```python

ZSimRNG = SimRNG.InitializeRNSeed()

Queue1 = SimClasses.FIFOQueue()
Queue2 = SimClasses.FIFOQueue()
Server1 = SimClasses.Resource()
Server2 = SimClasses.Resource()
TotalTime1 = SimClasses.DTStat()
TotalTime2 = SimClasses.DTStat()
Calendar = SimClasses.EventCalendar()

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

TheDTStats.append(TotalTime1)
TheDTStats.append(TotalTime2)
TheQueues.append(Queue1)
TheQueues.append(Queue2)
TheResources.append(Server1)
TheResources.append(Server2)

# lists to collect across replication outputs
AllQueue1 = []
AllQueue2 = []
AllTotalTime1 = []
AllTotalTime2 = []
AllUtil1 = []
AllUtil2 = []

Server1.SetUnits (1)
Server2.SetUnits (1)

MeanTBA1 = 1/(1.0)
MeanPT1 = 0.8
MeanPT2 = 0.7
WarmUp = 5000.0
RunLength = 50000.0

def Arrival():
    SimFunctions.Schedule(Calendar,"Arrival",SimRNG.Expon(MeanTBA1, 1))
    NewCustomer = SimClasses.Entity()

    if Server1.Busy == 0:
```

```
            Server1.Seize(1)
            SimFunctions.SchedulePlus(Calendar,"EndOfService1",SimRNG.Expon(MeanPT1,3), NewCustomer)
        else:
            Queue1.Add(NewCustomer)

def EndOfService1(DepartingCustomer):
    # record total time in stage 1
    TotalTime1.Record(SimClasses.Clock - DepartingCustomer.CreateTime)
    # assign the time as an attribute to the entity
    DepartingCustomer.EndService1Time = SimClasses.Clock
    # if server 2 is idle start service there
    if Server2.Busy == 0:
        Server2.Seize(1)
        SimFunctions.SchedulePlus(Calendar, "EndOfService2", SimRNG.Expon(MeanPT2,3), DepartingCustomer)
    else:
    # else add the customer to queue 2
        Queue2.Add(DepartingCustomer)
    # if queue 1 > 0 start the service of the next customer
    if Queue1.NumQueue() > 0:
        NextCustomer = Queue1.Remove()
        SimFunctions.SchedulePlus(Calendar, "EndOfService1", SimRNG.Expon(MeanPT1,3), NextCustomer)
    # else idle the server
    else:
        Server1.Free(1)

def EndOfService2(DepartingCustomer):
    # record stage 2 time
    TotalTime2.Record(SimClasses.Clock - DepartingCustomer.EndService1Time)
    if Queue2.NumQueue() > 0:
        NextCustomer = Queue2.Remove()
        SimFunctions.SchedulePlus(Calendar,"EndOfService2", SimRNG.Expon(MeanPT2,5), NextCustomer)
    else:
        Server2.Free(1)

# replication loop
for reps in range(0,10,1):
    print reps+1
    SimFunctions.SimFunctionsInit(Calendar,TheQueues,TheCTStats,TheDTStats,TheResources)

    SimFunctions.Schedule(Calendar,"Arrival",SimRNG.Expon(MeanTBA1, 1))
    SimFunctions.Schedule(Calendar,"EndSimulation",RunLength)
    SimFunctions.Schedule(Calendar,"ClearIt",WarmUp)

    NextEvent = Calendar.Remove()
    SimClasses.Clock = NextEvent.EventTime
    if NextEvent.EventType == "Arrival":
        Arrival()

    # main simulation loop
```

```python
    while NextEvent.EventType != "EndSimulation":
        NextEvent = Calendar.Remove()
        SimClasses.Clock = NextEvent.EventTime
        if NextEvent.EventType == "Arrival":
            Arrival()
        elif NextEvent.EventType == "EndOfService1":
            EndOfService1(NextEvent.WhichObject)
        elif NextEvent.EventType == "EndOfService2":
            EndOfService2(NextEvent.WhichObject)
        elif NextEvent.EventType == "ClearIt":
            SimFunctions.ClearStats(TheCTStats,TheDTStats)

    # save the output for each replication
    AllQueue1.append(Queue1.Mean())
    AllQueue2.append(Queue2.Mean())
    AllTotalTime1.append(TotalTime1.Mean())
    AllTotalTime2.append(TotalTime2.Mean())
    AllUtil1.append(Server1.Mean())
    AllUtil2.append(Server2.Mean())

# print the estimates
print ("Estimated Total Time 1 = ",np.mean(AllTotalTime1))
print ("Estimated Total Time 2 = ",np.mean(AllTotalTime2))
print ("Estimated Queue 1 length = ", np.mean(AllQueue1))
print ("Estimated Queue 2 length = " , np.mean(AllQueue2))
print ("Estimated Server 1 Util. = ", np.mean(AllUtil1))
print ("Estimated Server 2 Util. = " , np.mean(AllUtil2))
```