COMP 312 Assignment 7

Daniel Braithwaite

May 2, 2016

1 Python

1.1 Program A

1.1.1 Code

```
"""(q3.py)␣M/M/c␣queueing␣system␣with␣monitor
␣␣␣and␣multiple␣replications"""

from SimPy.Simulation import *
import random
import numpy
import math

## Useful extras ----------
def conf(L):
    """confidence␣interval"""
    lower = numpy.mean(L) - 1.96*numpy.std(L)/math.sqrt(len(L))
    upper = numpy.mean(L) + 1.96*numpy.std(L)/math.sqrt(len(L))
    return (lower,upper)

## Model ----------
class Source(Process):
    """generate␣random␣arrivals"""
    def run(self, N, lamb, mu):
        for i in range(N):
            a = Arrival(str(i))
            activate(a, a.run(mu))
            t = random.expovariate(lamb)
            yield hold, self, t

class Arrival(Process):
    """an␣arrival"""
    def run(self, mu):
        # Check to see if arival is rejected
        if len(G.server.waitQ) >= 5:
            G.rejectmon.observe(1)
            return
        G.rejectmon.observe(0)

        # See if the arrival will balk
        pBalk = 0.2 * len(G.server.waitQ)
        if random.random() <= pBalk:
            G.balkmon.observe(1)
            return
        G.balkmon.observe(0)

        arrivetime = now()
        yield request, self, G.server
        t = random.expovariate(mu)
        yield hold, self, t
        yield release, self, G.server
        delay = now()-arrivetime
        G.delaymon.observe(delay)



class G:
    server = 'dummy'
    delaymon = 'Monitor'
    balkmon = 'Monitor'
    rejectmon = 'Monitor'

def model(c, N, lamb, mu, maxtime, rvseed):
    # setup
    initialize()
    random.seed(rvseed)
    G.server = Resource(c)
    G.delaymon = Monitor()
    G.balkmon = Monitor()
    G.rejectmon = Monitor()

    # simulate
    s = Source('Source')
    activate(s, s.run(N, lamb, mu))
    simulate(until=maxtime)

    # gather performance measures
    W = G.delaymon.mean()
    B = G.balkmon.mean()
    R = G.rejectmon.mean()
    return(W, B, R)

## Experiment ------------
allW = []
allBalk = []
allReject = []
for k in range(50):
    print k
    seed = 123*k
    result = model(c=1, N=10000, lamb=2.0, mu=(1.0/0.5), maxtime=2000000, rvseed
    allW.append(result[0])
    allBalk.append(result[1])
    allReject.append(result[2])
#print allW
print ""
print "Estimate␣of␣W:",numpy.mean(allW)
print "Conf␣int␣of␣W:",conf(allW)
print ""
print "Estimate␣of␣Balk:",numpy.mean(allBalk)
print "Conf␣int␣of␣Balk:",conf(allBalk)
print ""
print "Estimate␣of␣Reject:",numpy.mean(allReject)
print "Conf␣int␣of␣Reject:",conf(allReject)
```

1.1.2 Output

```
Estimate of W: 1.20972369916
Conf int of W: (1.2029828999724295, 1.2164644983404909)

Estimate of Balk: 0.214441575607
Conf int of Balk: (0.21298878230402357, 0.21589436890972064)

Estimate of Reject: 0.008358
Conf int of Reject: (0.0079898917534854726, 0.0087261082465145256)
```

1.2 Program B

1.2.1 Code

```
"""(q3.py)␣M/M/c␣queueing␣system␣with␣monitor
␣␣␣and␣multiple␣replications"""

from SimPy.Simulation import *
import random
import numpy
import math

## Useful extras ----------
def conf(L):
    """confidence␣interval"""
    lower = numpy.mean(L) - 1.96*numpy.std(L)/math.sqrt(len(L))
    upper = numpy.mean(L) + 1.96*numpy.std(L)/math.sqrt(len(L))
    return (lower,upper)

## Model ----------
class Source(Process):
    """generate␣random␣arrivals"""
    def run(self, N, lamb):
        for i in range(N):
            a = Arrival(str(i))
            activate(a, a.run())
            t = random.expovariate(lamb)
            yield hold, self, t

class Arrival(Process):
    """an␣arrival"""
    def run(self):
        arrivetime = now()
        # Before we can move into the river we need
        # to be assigned a dock
        yield request, self, G.dock

        # Then we must wait for a tug boat to take us to the dock
        yield request, self, G.tug_boat

        # Once given a tug boat it takes 0.2 days to get there
        yield hold, self, 0.2

        # Now we can release the tug boat
        yield release, self, G.tug_boat

        # Now it takes a day to unload the boat
        yield hold, self, 1.0

        # Now we can release the dock
        yield release, self, G.dock
        #t = random.expovariate(mu)
        #yield hold, self, t
        #yield release, self, G.server
        delay = now()-arrivetime
        G.delaymon.observe(delay)



class G:
    dock = 'dummy'
    tug_boat = 'dummy'
    delaymon = 'Monitor'

def model(c, N, lamb, maxtime, rvseed):
    # setup
    initialize()
    random.seed(rvseed)
    G.dock = Resource(2)
    G.tug_boat = Resource(c, monitored=True)
    G.delaymon = Monitor()

    # simulate
    s = Source('Source')
    activate(s, s.run(N, lamb))
    simulate(until=maxtime)

    # gather performance measures
    U = 1 - G.tug_boat.waitMon.timeAverage()
    W = G.delaymon.mean()
    return(W, U)

## Experiment ----------------

for i in range(1, 3):
    print "\n" + str(i) + " Tug Boats"
    allW = []
    allU = []
    for k in range(50):
        seed = 123*k
        result = model(c=i, N=10000, lamb=1, maxtime=2000000, rvseed=seed)
        allW.append(result[0])
        allU.append(result[1])
    #print allW
    print ""
    print "Estimate of W:",numpy.mean(allW)
    print "Conf int of W:",conf(allW)
    print ""
    print "Estimate of U:",numpy.mean(allU)
    print "Conf int of U:",conf(allU)
```

1.2.2 Output

```
1 Tug Boats

Estimate of W: 1.5617122988
Conf int of W: (1.5568093809484571, 1.5666152166434895)

Estimate of U: 0.994934336555
Conf int of U: (0.99485610117267298, 0.99501257193779857)

2 Tug Boats

Estimate of W: 1.55240355925 Conf int of W: (1.547537739331946, 1.5572693791732504)
Estimate of U: 1.0
Conf int of U: (1.0, 1.0)

1 Tug Boats

Estimate of W: 1.5617122988
Conf int of W: (1.5568093809484571, 1.5666152166434895)

Estimate of U: 0.994934336555
Conf int of U: (0.99485610117267298, 0.99501257193779857)

2 Tug Boats

Estimate of W: 1.55240355925
Conf int of W: (1.547537739331946, 1.5572693791732504)

Estimate of U: 1.0
Conf int of U: (1.0, 1.0)
```