COMP 312 Assignment 7

Daniel Braithwaite

May 9, 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):
    n = 0

    """an arrival"""
    def run(self, mu):
        arrivetime = now()

        Arrival.n += 1
        G.nummon.observe(Arrival.n)

        currentStation = 1
        while not currentStation == -1:

            station = 'dummy'
            if currentStation == 1:
```

```
                station = G.station1
            elif currentStation == 2:
                station = G.station2
            elif currentStation == 3:
                station = G.station3

            yield request, self, station
            t = random.expovariate(mu)
            yield hold, self, t
            yield release, self, station

            r = random.random()
            if currentStation == 1:
                if r <= 0.1:
                    currentStation = 2
                else:
                    currentStation = 3
            elif currentStation == 2:
                if r <= 0.2:
                    currentStation = 1
                else:
                    currentStation = 3
            elif currentStation == 3:
                if r <= 0.1:
                    currentStation = 2
                else:
                    currentStation = -1


        Arrival.n -= 1
        G.nummon.observe(Arrival.n)

        delay = now()-arrivetime
        G.delaymon.observe(delay)



class G:
    station1 = 'dummy'
    station2 = 'dummy'
    station3 = 'dummy'
    delaymon = 'Monitor'
    nummon = 'Monitor'

def model(c, N, lamb, mu, maxtime, rvseed):
    # setup
```

```
    initialize()
    random.seed(rvseed)
    G.station1 = Resource(c)
    G.station2 = Resource(c)
    G.station3 = Resource(c)

    G.delaymon = Monitor()
    G.nummon = Monitor()

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

    # gather performance measures
    W = G.delaymon.mean()
    L = G.nummon.timeAverage()
    return(W, L)

## Experiment ----------
allW = []
allL = []
for k in range(50):
    print k
    seed = 123*k
    result = model(c=2, N=10000, lamb=1.4, mu=1.00,
                   maxtime=2000000, rvseed=seed)
    allW.append(result[0])
    allL.append(result[1])

print ""
print "Estimate of W:",numpy.mean(allW)
print "Conf int of W:",conf(allW)

print ""
print "Estimate of L:",numpy.mean(allL)
print "Conf int of L:",conf(allL)
```

1.1.2 Output

Lambda = 1
Estimate of W: 3.26472367151
Conf int of W: (3.2488630500389366, 3.280584292989257)

Estimate of L: 3.26905934434
Conf int of L: (3.2481626576521117, 3.289956031026612)

Lambda = 1.4
Estimate of W: 5.27502175978
Conf int of W: (5.2113909913800276, 5.3386525281750901)

Estimate of L: 7.39460960176
Conf int of L: (7.2956646964018264, 7.4935545071161238)

Lambda = 1.6
Estimate of W: 8.88431876814
Conf int of W: (8.6294892979862041, 9.1391482382906801)

Estimate of L: 14.2139801084
Conf int of L: (13.788264932464067, 14.639695284252635)

Lambda = 1.8
Estimate of W: 50.9382817297
Conf int of W: (44.339013047433241, 57.537550411953703)

Estimate of L: 90.5524817225
Conf int of L: (78.81803465053963, 102.28692879440669)

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)

class Job(Process):

    """an arrival"""
```

```
    def run(self, p):
        currentNode = 0

        while True:
            server = getattr(G, 'node' + str(currentNode))

            t = now()

            yield request, self, server[0]
            t = random.expovariate(server[1])
            yield hold, self, t
            yield release, self, server[0]

            server[2].observe(now() - t)

            if currentNode == 0:
                r = random.random()
                if r <= p:
                    currentNode = 1
                else:
                    currentNode = 2
            elif currentNode == 1:
                currentNode = 0
            elif currentNode == 2:
                currentNode = 1



class G:
    node0 = ['server', 1.0/20.0, 'Monitor']
    node1 = ['server', 1.0/10.0, 'Monitor']
    node2 = ['server', 1.0/30.0, 'Monitor']
    delaymon = 'Monitor'

def model(N, p, maxtime, rvseed):
    # setup
    initialize()

    # Create the customers in the sysyem
    for i in range(N):
        j = Job(name="Job" + str(i))
        activate(j, j.run(p))

    random.seed(rvseed)
    G.node0[0] = Resource(N, monitored=True) # Has enough servers to handler all
    G.node0[2] = Monitor()
```

```
    G.node1[0] = Resource(1, monitored=True)
    G.node1[2] = Monitor()

    G.node2[0] = Resource(1, monitored=True)
    G.node2[2] = Monitor()
    G.delaymon = Monitor()

    # simulate
    simulate(until=maxtime)

    L = []
    for i in range(3):
        server = getattr(G, 'node' + str(i))
        h = server[0].waitMon.timeAverage() + server[0].actMon.timeAverage()
        L.append(h)

    # gather performance measures
    W0 = G.node0[2].mean()
    W1 = G.node1[2].mean()
    W2 = G.node2[2].mean()
    return ([W0, W1, W2], L)

## Experiment ----------
allW0 = []
allW1 = []
allW2 = []
allL0 = []
allL1 = []
allL2 = []
for k in range(50):
    print k
    seed = 123*k
    result = model(N=5, p=0.5, maxtime=10000, rvseed=seed)

    W = result[0]
    L = result[1]

    allW0.append(W[0])
    allW1.append(W[1])
    allW2.append(W[2])
    allL0.append(L[0])
    allL1.append(L[1])
    allL2.append(L[2])

print ""
```

```
print "Estimate of W0:",numpy.mean(allW0)
print "Conf int of W0:",conf(allW0)

print ""
print "Estimate of W1:",numpy.mean(allW1)
print "Conf int of W1:",conf(allW1)

print ""
print "Estimate of W2:",numpy.mean(allW2)
print "Conf int of W2:",conf(allW2)

print ""
print "Estimate of L0:",numpy.mean(allL0)
print "Conf int of L0:",conf(allL0)

print ""
print "Estimate of L1:",numpy.mean(allL1)
print "Conf int of L1:",conf(allL1)

print ""
print "Estimate of L2:",numpy.mean(allL2)
print "Conf int of L2:",conf(allL2)
```

1.2.2 Output

Estimate of W0: 4943.04818104
Conf int of W0: (4898.3210316269033, 4987.7753304614289)

Estimate of W1: 4985.24543653
Conf int of W1: (4939.9264150329, 5030.5644580285843)

Estimate of W2: 4977.50494534
Conf int of W2: (4937.3688100317413, 5017.6410806387121)

Estimate of L0: 1.20182218568
Conf int of L0: (1.1806983072381694, 1.2229460641316552)

Estimate of L1: 1.17795497502
Conf int of L1: (1.146837330909527, 1.2090726191268364)

Estimate of L2: 2.6202228393
Conf int of L2: (2.5774359912704954, 2.6630096873233171)

2 Code Used For MVA

```
f = (100.0/321.0, 10.0/107.0, 100.0/321.0, 91.0/321.0)
es = (3, 6, 15, 20)

def lam(m, f, w):
    s = 0
    for i in range(len(f)):
        s += f[i] * w[i]

    return float(m)/float(s)


def w(l, es):
    return (1 + l) * es

def l(lam, f, w):
    return lam * f * w

res = [[3, 6, 15, 20]]
lmb = lam(1, f, es)
res[0].append(lmb)
res[0].append(l(lmb, f[0], es[0]))
res[0].append(l(lmb, f[1], es[1]))
res[0].append(l(lmb, f[2], es[2]))
res[0].append(l(lmb, f[3], es[3]))

for i in range(1, 5):
    m = i+1
    prev = res[i-1]

    c = []
    for j in range(len(es)-1):
        # print str(6 + j)
        c.append(w(prev[5 + j], es[j]))
    c.append(es[3])

    lmb = lam(m, f, c)
    c.append(lmb)

    for j in range(len(es)):
        c.append(l(lmb, f[j], c[j]))

    res.append(c)

for row in res:
    print row
```