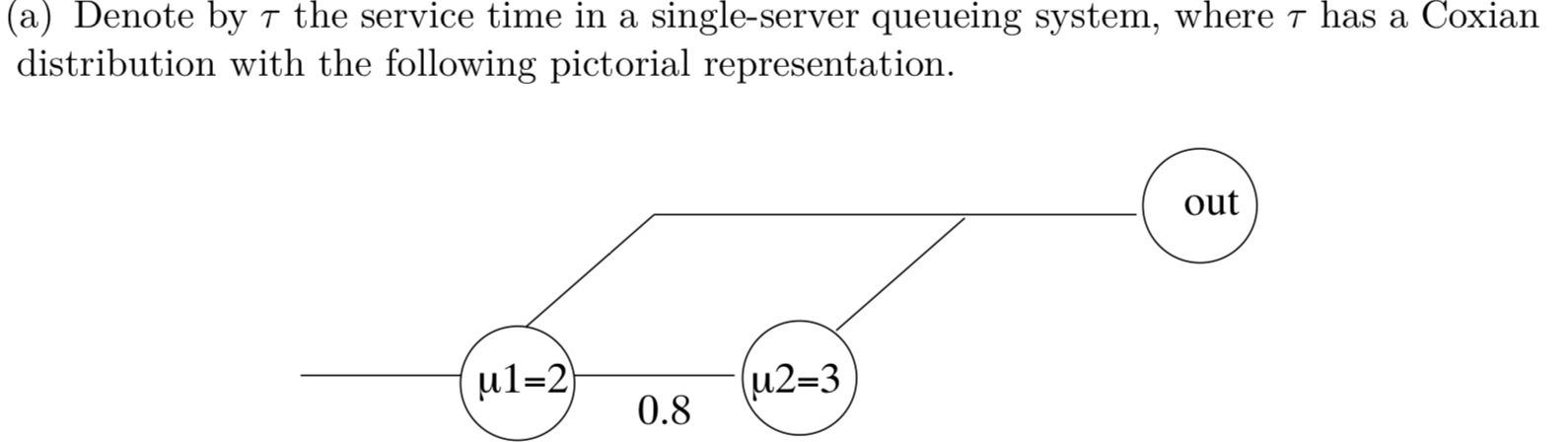
COMP312 Assignment06

Part1- Python

1. (a)(i)



import random

def tablelookup(P):

"""Sample from i=0..n-1 with probabilities P[i]"""

u = random.random()

sumP = 0.0

for i in range(len(P)):

sumP += P[i]

if u < sumP:

return i

def phase(beta,mu,T):

"""Random variate from a phase-type distribution

beta= entry probabilities

mu = rates

T = transition matrix

"""

i = tablelookup(beta) # entry node

result = 0.0

n = len(mu)

while i != n:

result += random.expovariate(mu[i])

i = tablelookup(T[i])

return result

beta= [1,0]

mu= [2,3]

T= [[0, 0.8, 0.2],

[0, 0.0, 1.0]]

print(phase(beta, mu, T)). #given a coxian-type data input

(a)(ii)

Simulate 10000 random variates from this particular Coxian distribution and give a 95% confidence interval for E[τ] and E[τ2]. Compare your results to the exact values of E(τ) =23/30 and E(τ2) =17/18and comment on your findings.

(I utilised phase.py provided in Bites of SimPy to get all random variate from phase-type since coxian distribution is just one of the specific type of phase-type)

""" (phase.py) simulating from a phase-type distribution"""

import random

def tablelookup(P):

"""Sample from i=0..n-1 with probabilities P[i]"""

u = random.random()

sumP = 0.0

for i in range(len(P)):

sumP += P[i]

if u < sumP:

return i

def phase(beta,mu,T):

"""Random variate from a phase-type distribution

beta= entry probabilities

mu = rates

T = transition matrix

"""

i = tablelookup(beta) # entry node

result = 0.0

n = len(mu)

while i != n:

result += random.expovariate(mu[i])

i = tablelookup(T[i])

return result

import math

import random, phase, numpy as np

beta= [1,0]

mu= [2,3]

T=[[0, 0.8, 0.2],

[0, 0.0, 1.0]]

def conf(L):

"""Give confidence interval"""

lower= np.mean(L)- 1.96\*np.std(L)/math.sqrt(len(L))

upper= np.mean(L)+ 1.96\*np.std(L)/math.sqrt(len(L))

return (lower, upper)

def coxian():

W=[]

W2=[]

for i in range (10000):

x= phase.phase(beta, mu, T)

W.append(x)

W2.append(x\*\*2)

return (np.mean(W), np.mean(W2), conf(W), conf(W2))

random.seed(123)

r= coxian()

print "estimate of E[tau]: ", r[0]

print "estimate of E[tau^2]: ",r[1]

print "95% Confidence Interval for E[tau]= {0}".format(r[2])

print "95% Confidence Interval for E[tau^2]= {0}".format(r[3])

print "V(tau)= ", r[1]-(r[0]\*\*2)

output:

estimate of E[tau]: 0.7717677798157517

estimate of E[tau^2]: 0.964190927389069

95% Confidence Interval for E[tau]= (0.7598687003431402, 0.7836668592883632)

95% Confidence Interval for E[tau^2]= (0.9326127615923999, 0.995769093185738)

V(tau)= 0.3685654214273343

Comment: the computed output from all x1, x2, x3……x10000 random variates gives 0.771767 which is close enough with the exact value of 23/30= 0.766666667 (it lies within the 95%CI range) same as E[tau\*\*2]

2.

## 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)

## Erlang distribution ----------

def erlangvariate(k,mu):

"""random variate from an Erlang(k,mu) distribution"""

result = 0.0

for i in range(k):

result += random.expovariate(mu)

return result

## Model ----------

class Source(Process):

"""generate random arrivals"""

def run(self, N, lamb, k, mu):

for i in range(N):

a = Arrival(str(i))

activate(a, a.run(k, mu))

t = random.expovariate(lamb)

yield hold, self, t

class Arrival(Process):

"""an arrival"""

n = 0 # class variable (number in system)

def run(self, k, mu):

Arrival.n += 1 # number in system

arrivetime = now()

G.numbermon.observe(Arrival.n)

if (Arrival.n>0):

G.busymon.observe(1)

else:

G.busymon.observe(0)

yield request, self, G.server

t = erlangvariate(k,mu)

G.servicemon.observe(t)

G.servicesquaredmon.observe(t\*\*2)

yield hold, self, t

yield release, self, G.server

Arrival.n -= 1

G.numbermon.observe(Arrival.n)

if (Arrival.n>0):

G.busymon.observe(1)

else:

G.busymon.observe(0)

delay = now()-arrivetime

G.delaymon.observe(delay)

class G:

server = 'dummy'

delaymon = 'Monitor'

numbermon = 'Monitor'

busymon = 'Monitor'

servicemon = 'Monitor'

servicesquaredmon = 'Monitor'

def model(c, N, lamb, k, mu, maxtime, rvseed):

# setup

initialize()

random.seed(rvseed)

G.server = Resource(c,monitored=True)

G.delaymon = Monitor()

G.numbermon = Monitor()

G.busymon = Monitor()

G.servicemon = Monitor()

G.servicesquaredmon = Monitor()

# simulate

s = Source('Source')

activate(s, s.run(N, lamb, k, mu))

simulate(until=maxtime)

# gather performance measures

W = G.delaymon.mean()

L = G.numbermon.timeAverage()

B = G.busymon.timeAverage()

S = G.servicemon.mean()

S2 = G.servicesquaredmon.mean()

LQ = G.server.waitMon.timeAverage()

lambeff = L/W

wQ = LQ/lambeff

Y = wQ - lambeff\*S2/(2.0\*(1-B))

return(Y)

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

allY = []

for i in range(50):

seed = 123\*i

result = model(c=1, N=10000, lamb=3, k=4, mu=20,

maxtime=2000000, rvseed=seed)

allY.append(result)

print "Estimate of Y:",numpy.mean(allY)

print "Conf int of Y:",conf(allY)