Homework 2

[redacted]

Problem 1

In [16]:
```
import matplotlib.pyplot as plt
import numpy as np
```

In [81]:
```
np.random.seed(1)

def CI_95(data):
    a = np.array(data)
    n = len(a)
    m = np.mean(a)
    std = np.std(a)
    #se = std/math.sqrt(Replications)
    var = (std**2)*(n/(n-1))
    hw = 1.96*np.sqrt(var/n)
    print "Mean and CI:", round(m,4), [round(m-hw,4),round(m+hw,4)]
    #print "relative error:", se/m

def re(data, i):
    a = np.array(data)
    n = len(a)
    m = np.mean(a)
    var = (np.std(a)**2)*(n/(n-1))
    #std = np.std(a)
    se = np.sqrt(var)/np.sqrt(i)
    re = se/m
    return re

Maturity = 1.0
InterestRate = 0.05
Sigma = 0.3
InitialValue = 50.0
StrikePrice = 55.0
Steps = 32
Interval = Maturity / Steps
Sigma2 = Sigma * Sigma / 2

np.random.seed(1)
Replications = 10000

ValueList = [] # List to keep the option value for each sample path

for i in range(0,Replications):
    Sum = 0.0
    X = InitialValue
    for j in range(0,Steps):
        Z = np.random.standard_normal(1)
        X = X * np.exp((InterestRate - Sigma2) * Interval +
                       Sigma * np.sqrt(Interval) * Z)
        Sum = Sum + X
    Value = np.exp(-InterestRate * Maturity) * max(Sum/Steps - StrikePrice, 0)
    ValueList.append(Value)

print "{0:.4%}".format(re(ValueList, Replications))
```

```
2.1349%
```

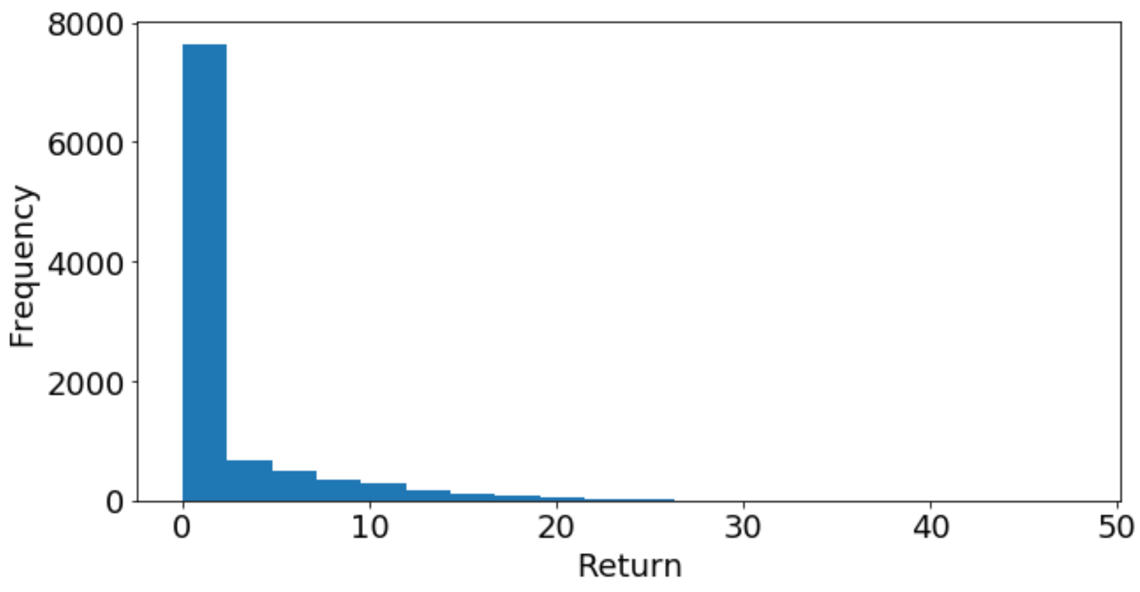
In [73]:
```
while re(ValueList, Replications) >= 0.01:
    Sum1 = 0.0
    X1 = InitialValue
    for j in range(0,Steps):
        Z = np.random.standard_normal(1)
        X1 = X1 * np.exp((InterestRate - Sigma2) * Interval +
                        Sigma * np.sqrt(Interval) * Z)
        Sum1 += X1
    Value1 = np.exp(-InterestRate * Maturity) * max(Sum1/Steps - StrikePrice, 0)
    ValueList.append(Value1)
    Replications += 1
```

In [74]:
```
print "Approximately %d replications would it take to decrease the relative error
to less than 0.01." %Replications
```

```
Approximately 46550 replications would it take to decrease the reletive error to
less than 0.01.
```

In [80]:
```
plt.hist(ValueList, bins=20)
plt.xlabel('Return')
plt.ylabel('Frequency')
plt.rcParams['figure.figsize'] = (10, 5)
plt.rcParams.update({'font.size': 18})

CI_95(ValueList)

#print "Mean, Standard deviation, Standard error and CI:", CI_95(ValueList)
plt.show()
```

```
Mean and CI: 2.1593 [2.0689, 2.2496]
```



In [ ]:

Problem 2

(a)

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

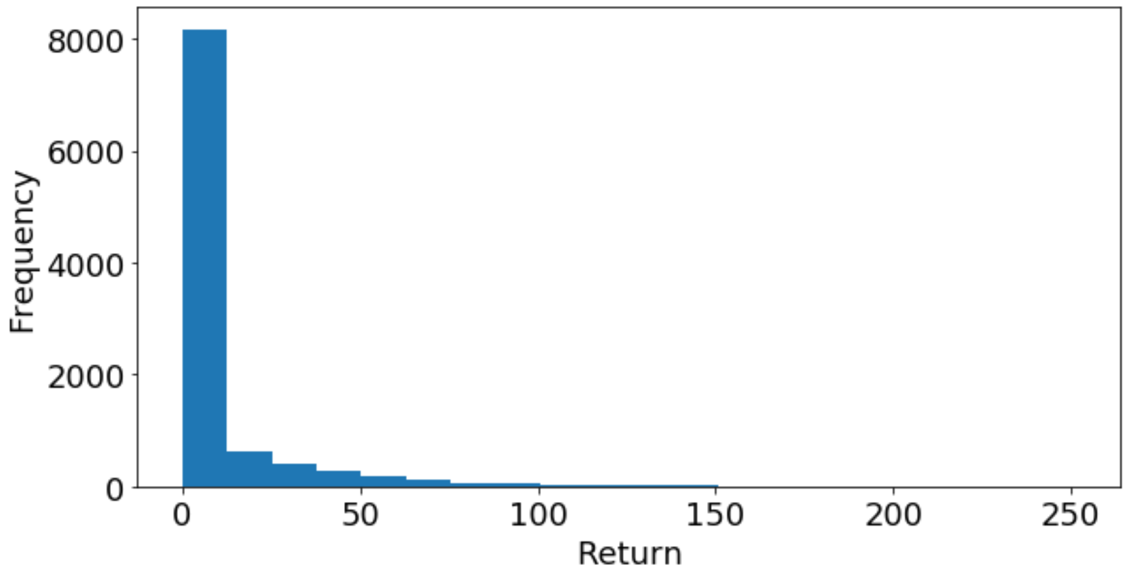
In [3]:
```
np.random.seed(1)

Maturity = 1.0
InterestRate = 0.05
Sigma = 0.4
InitialValue = 95.0
K = 100
Steps = 64
Interval = Maturity / Steps
Sigma2 = Sigma * Sigma / 2
b = 90

Replications = 10000

ValueList = [] # List to keep the option value for each sample path

for i in range(0,Replications):
    PriceList = [] # a list to store stock prices at different steps
    X = InitialValue
    for j in range(0,Steps):
        Z = np.random.standard_normal(1)
        X = X * np.exp((InterestRate - Sigma2) * Interval +
                       Sigma * np.sqrt(Interval) * Z)
        PriceList.append(X)
    Indicator = 0 # Initialize Indicator function
    for k in range(0, Steps):
        if PriceList[k] <= b:
            Indicator = 1
            break
    Value = Indicator * np.exp(-InterestRate * Maturity) * max(X - K, 0)
    ValueList.append(Value)
```

In [25]:
```
def CI_95(data):
    a = np.array(data)
    n = len(a)
    m = np.mean(a)
    std = np.std(a)
    var = (std**2)*(n/(n-1))
    hw = 1.96*np.sqrt(var/n)
    return round(m,4), [round(m-hw,4),round(m+hw,4)]
```

In [26]:
```
print "Mean and CI:", CI_95(ValueList)
```

```
Mean and CI: (8.1495, [7.7541, 8.5448])
```

In [14]:
```
plt.hist(ValueList, bins=20)
plt.xlabel('Return')
plt.ylabel('Frequency')
plt.rcParams['figure.figsize'] = (14, 7)
plt.rcParams.update({'font.size': 18})
```



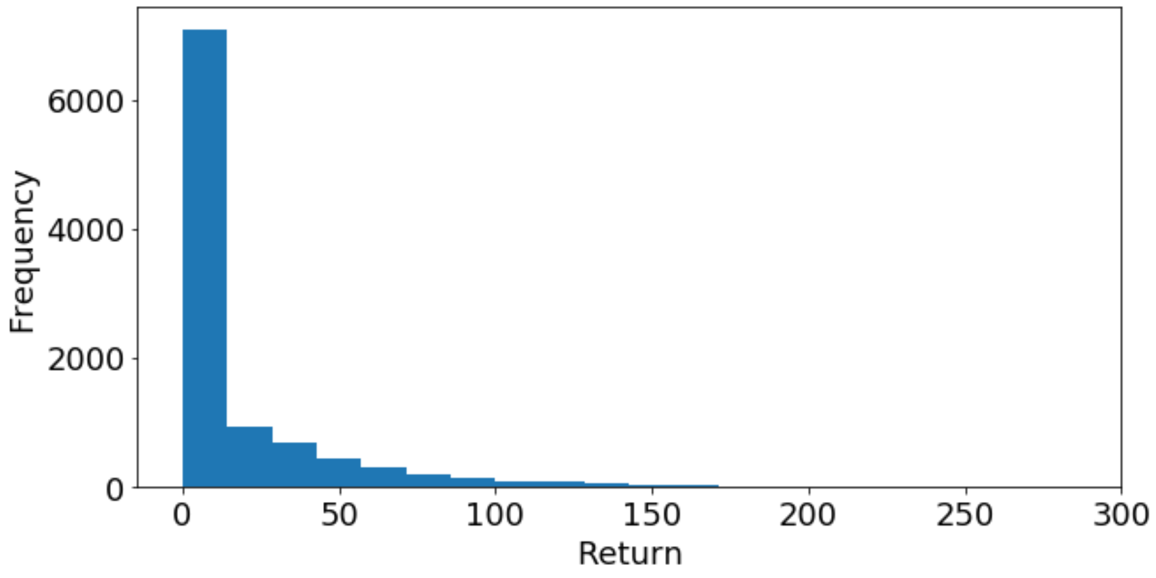
(b)

In [18]:
```
np.random.seed(1)

ValueList1 = [] # List to keep the option value for each sample path

for i in range(0,Replications):
    #PriceList = [] # a list to store stock prices at different steps
    X = InitialValue
    for j in range(0,Steps):
        Z = np.random.standard_normal(1)
        X = X * np.exp((InterestRate - Sigma2) * Interval +
                       Sigma * np.sqrt(Interval) * Z)
        #PriceList.append(X)
    Value1 = np.exp(-InterestRate * Maturity) * max(X - K, 0)
    ValueList1.append(Value1)
```

In [27]:
```
print "Mean and CI:", CI_95(ValueList1)
```

```
Mean and CI: (15.3769, [14.8082, 15.9457])
```

In [20]:
```
plt.hist(ValueList1, bins=20)
plt.xlabel('Return')
plt.ylabel('Frequency')
plt.rcParams['figure.figsize'] = (10, 5)
plt.rcParams.update({'font.size': 18})
```



Based on the result of (a) and (b), we can see the value of Down-and-In call option is much lower than the value of the standard European call option. Because in Down-and-in option, only if the price of stock reaches the barrier before expiry, then the option is said to have knocked in. Due to that, there is some possibility that the option doesn't pay off, which is shown by the "Indicator" in the code.

In [ ]:

Problem 3

In [9]:
```
import SimFunctions
import SimRNG
import SimClasses
import numpy as np
import pandas as pd
```

In [19]:
```
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)

MeanTBA = 1.0
Phases = 3
RunLength = 55000.0
WarmUp = 5000.0
s = 3 # number of servers, s = 1, 2, 3
```

In [20]:
```
for s in range (1,s+1,1):
    print "\nWhen s = %d" %s
    print "----------------------------"
    Server.SetUnits(s)
    MeanST = 0.8 * s

    AllWaitMean = []
    AllQueueMean = []
    AllQueueNum = []
    AllServerMean = []
    print "Rep", "Expected System Time ", "expected Number of Customers ", "Expec
ted Number of Busy Servers"

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

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

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

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

        SimFunctions.SimFunctionsInit(Calendar,TheQueues,TheCTStats,TheDTStats,Th
eResources)
        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())
        AllServerMean.append(Server.Mean())
        print reps+1, round(Wait.Mean(), 4), round(Queue.Mean(), 4), round(Server
.Mean(), 4)

    print "Estimated Expected System Time:",round(np.mean(AllWaitMean), 4)
    print "Estimated Expected Number of Customers:", round(np.mean(AllQueueMean),
4)
```

```
    print "Estimated Expected Number of Busy Servers:",round(np.mean(AllServerMea
n), 4)
```

```
When s = 1
---------------------------
Rep Expected System Time  expected Number of Customers  Expected Number of Busy
Servers
1 2.9988 2.9745 0.7945
2 3.0966 3.1348 0.8111
3 2.9602 2.9749 0.8063
4 2.9349 2.9354 0.8025
5 2.895 2.8745 0.7931
6 2.9419 2.9507 0.8034
7 2.89 2.8836 0.7991
8 3.0527 3.0744 0.8074
9 2.8528 2.8445 0.7965
10 2.7794 2.7788 0.803
Estimated Expected System Time: 2.9402
Estimated Expected Number of Customers: 2.9426
Estimated Expected Number of Busy Servers: 0.8017

When s = 2
---------------------------
Rep Expected System Time  expected Number of Customers  Expected Number of Busy
Servers
1 3.5443 3.5402 1.6089
2 3.5648 3.5841 1.6108
3 3.4884 3.4961 1.6004
4 3.5318 3.5322 1.5948
5 3.6218 3.6468 1.6138
6 3.3736 3.359 1.5853
7 3.3981 3.379 1.5877
8 3.4659 3.4758 1.6098
9 3.4801 3.4847 1.6027
10 3.722 3.7252 1.6068
Estimated Expected System Time: 3.5191
Estimated Expected Number of Customers: 3.5223
Estimated Expected Number of Busy Servers: 1.6021

When s = 3
---------------------------
Rep Expected System Time  expected Number of Customers  Expected Number of Busy
Servers
1 4.1484 4.167 2.414
2 4.1019 4.0901 2.3927
3 4.0478 4.0547 2.3915
4 4.1249 4.1127 2.3869
5 4.0128 3.9947 2.3826
6 4.2622 4.266 2.4071
7 4.2032 4.1901 2.3979
8 4.1216 4.1254 2.4048
9 4.3264 4.3457 2.4178
10 4.0595 4.0481 2.3829
Estimated Expected System Time: 4.1409
Estimated Expected Number of Customers: 4.1395
Estimated Expected Number of Busy Servers: 2.3978
```

(a)

Based on the results above, when there is 1 server, the estimated expected number of customers in the system is 2.9426; expected system time is 2.9402; expected number of busy servers is 0.8017.
When there is 2 servers, the estimated expected number of customers in the system is 3.5223; expected system time is 3.5191; expected number of busy servers is 1.6021.
When there is 3 servers, the estimated expected number of customers in the system is 4.1395; expected system time is 4.1409; expected number of busy servers is 2.3978.

(b)

With the number of servers increasing (When there are 2 or more slow servers), the estimated expected system time and estimated expected number of customers increases (Because they are slower servers).
At the meantime, the estimated expected number of busy servers doesn't change (because the output should be devided by the number s)

In [ ]:

```

```

Problem 4

```
In [6]: import SimFunctions
        import SimRNG
        import SimClasses
        import numpy as np
        import pandas
```

```
In [ ]: 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
        MeanTR = 1.0 # Randomly select number, retrial customer return time ~ exp(MeanTR)
        MeanST = 0.8
        Phases = 3
        RunLength = 55000.0
        WarmUp = 5000.0
        c = 5 # Randomly selected number, if there are c customers in the system

        AllWaitMean = []
        AllQueueMean = []
        AllQueueNum = []
        AllServerMean = []
```

```
In [ ]: def Arrival():
            SimFunctions.Schedule(Calendar,"Arrival",SimRNG.Expon(MeanTBA, 1))
            if Queue.NumQueue() >= c: # if there are c or more customers in the system,
                SimFunctions.Schedule(Calendar,"Arrival",SimRNG.Expon(MeanTR,1)) # the ne
        w customer will return after an exponentiallly distributed time
            else:
                Customer = SimClasses.Entity()
                Queue.Add(Customer)

                if Server.Busy == 0:
                    Server.Seize(1)
                    SimFunctions.Schedule(Calendar,"EndOfService",SimRNG.Erlang(Phases,Me
        anST,2))
```

```
In [ ]: 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)
```

In [ ]:
```
for reps in range(0,10,1):

    SimFunctions.SimFunctionsInit(Calendar,TheQueues,TheCTStats,TheDTStats,TheRes
ources)
    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())
    AllServerMean.append(Server.Mean())
    print (reps+1, Wait.Mean(), Queue.Mean(), Queue.NumQueue(), Server.Mean())
```

In [ ]:
```
print ("Rep", "Average Wait", "Average Number in Queue", "Number Remaining in Que
ue", "Server Utilization")
# output results

print("Estimated Expected Average wait:",np.mean(AllWaitMean))
print("Estimated Expected Average queue-length:", np.mean(AllQueueMean))
print("Estimated Expected Average utilization:",np.mean(AllServerMean))
```

In [ ]:
```
# convert the output to a dataframe and write to a .csv file
output = {"AllWaitMean" : AllWaitMean, "AllQueueMean": AllQueueMean, "AllQueueNu
m" : AllQueueNum, "AllServerMean": AllServerMean}
output = pandas.DataFrame(output)
output.to_csv("MG1_output.csv", sep=",")
```

Problem 5

In [54]:
```
import SimFunctions
import SimRNG
import SimClasses
import numpy as np
```

In [98]:
```
ZSimRNG = SimRNG.InitializeRNSeed()

Queue1 = SimClasses.FIFOQueue()
Queue2 = SimClasses.FIFOQueue()
Wait1 = SimClasses.DTStat()
Wait2 = SimClasses.DTStat()
Server1 = SimClasses.Resource()
Server2 = SimClasses.Resource()
Calendar = SimClasses.EventCalendar()

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

TheDTStats.append(Wait1)
TheDTStats.append(Wait2)
TheQueues.append(Queue1)
TheQueues.append(Queue2)
TheResources.append(Server1)
TheResources.append(Server2)

AllQueue1 = []
AllQueue2 = []
AllWait1 = []
AllWait2 = []
AllUtil1 = []
AllUtil2 = []

Server1.SetUnits (1)
Server2.SetUnits (2)

MeanTBA = 1.0
MeanST1 = 0.8
MeanST2 = 0.7
RunLength = 50000.0
WarmUp = 5000.0
```

In [99]:
```
def Arrival():
    SimFunctions.Schedule(Calendar,"Arrival",SimRNG.Expon(MeanTBA, 1))
    Customer = SimClasses.Entity()
    Customer.ClassNum = 1
    #Queue1.Add(Customer)

    if Server1.Busy == 0:
        Server1.Seize(1)
        SimFunctions.SchedulePlus(Calendar,"EndOfService1",SimRNG.Expon(MeanST1,3
), Customer)

    else:
        Queue1.Add(Customer)

def EndOfService1(DepartingCustomer): # Customer job departing stage 1

    Wait1.Record(SimClasses.Clock - DepartingCustomer.CreateTime)
    if Queue1.NumQueue() > 0:
        NextCustomer = Queue1.Remove()
        SimFunctions.SchedulePlus(Calendar,"EndOfService1",SimRNG.Expon(MeanST1,3
), NextCustomer)
        Queue2.Add(NextCustomer)
        if Server2.Busy == 0:
            Server2.Seize(1)
            SimFunctions.SchedulePlus(Calendar,"EndOfService2",SimRNG.Expon(MeanS
T2,4), NextCustomer)
    else:
        Server1.Free(1)

def EndOfService2(DepartingCustomer):
    #DepartingCustomer = Queue2.Remove()
    #CreateTime = SimClasses.Clock
    Wait2.Record(SimClasses.Clock - DepartingCustomer.CreateTime)
    if Queue2.NumQueue() > 0:
        NextCustomer = Queue2.Remove()
        SimFunctions.SchedulePlus(Calendar,"EndOfService2", SimRNG.Expon(MeanST2,
4), NextCustomer)
    else:
        Server2.Free(1)
```

In [100]:
```
for reps in range(0,20,1):
    SimFunctions.SimFunctionsInit(Calendar,TheQueues,TheCTStats,TheDTStats,TheRes
ources)

    SimFunctions.Schedule(Calendar,"Arrival",SimRNG.Expon(MeanTBA, 1))
    #SimFunctions.Schedule(Calendar,"Arrival2",SimRNG.Expon(MeanTBA2, 2))
    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 == "EndOfService1":
        EndOfService1(NextEvent.WhichObject)
    elif NextEvent.EventType == "EndOfService2":
        EndOfService2(NextEvent.WhichObject)
    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 == "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())
    AllWait1.append(Wait1.Mean())
    AllWait2.append(Wait2.Mean()-Wait1.Mean()) # Use the total time to minus the
 queue 1 time.
    AllUtil1.append(Server1.Mean())
    AllUtil2.append(Server2.Mean())
```

In [101]:
```
print "Estimated Wait Time 1 = ",np.mean(AllWait1)
print "Estimated Wait Time 2 = ",np.mean(AllWait2)
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)
```

```
Estimated Wait Time 1 =  4.027434554901267
Estimated Wait Time 2 =  2.3086152591281417
Estimated Queue 1 length =  3.2258605361762234
Estimated Queue 2 length =  1.673499699064695
Estimated Server 1 Util. =  0.8001293937661591
Estimated Server 2 Util. =  0.6840385307509437
```

(a) The expected number of customers waiting at stage 1 is 3.23; The expected number of customers waiting at stage 2 is 1.67.
(b) The expected total time customers spend at stage 1 is 4.03; The expected total time customers spend at stage 2 is 2.31.
(c) Expected utilization of server 1 is 0.80; Expected utilization of server 2 is 0.68.

In [ ]: