Exercise 2 - Solutions

2.1

The code should be very easy to produce in Matlab. Here is one version:

```
function d=queue1(n,lambda,mu)

%Simulate delays in queue for single server queuing model (M/M/1)
%n the number of customers
%lambda the arrival rate of customers
%mu the service rate

t=exprnd(lambda*ones(n-1,1)); %inter-arrival times
s=exprnd(mu*ones(n-1,1));     %service timese

d(1)=0; %waiting time for customer 1 equals 0

for i=2:n
      d(i)=max(d(i-1)+s(i-1)-t(i-1),0);
end

d=d';
```

Let $\bar{D}_k$ be the average delay in the k^{th} independent *simulation replication*. I.e., you might write (we don't yet need to worry about management of random number streams):

```
for i=1:100; d=queue1(100,1,0.9); D(i)=mean(d); end;
```

The 1-*α* confidence interval for average delay is:

$$\frac{1}{m}\sum_{k=1}^{m}\bar{D}_k \pm t_{m-1,1-\alpha/2}\sqrt{\frac{\hat{Var}(\bar{D}_k)}{m}}$$

where $t_{m-1,1-\alpha/2}$ is the 1-*α*/2 percentile of Student's t-distribution with m-1 degrees of freedom. $\hat{Var}(\bar{D}_k)$ is the estimated variance of $\bar{D}_k$. In Matlab:

```
mean(D)+tinv(0.975,99)*sqrt(var(D)/100).
```

2.2

A Matlab-implementation of the inventory simulation is given below. Note that the inventory is controlled at fixed time intervals (of one month). It should not, however, be too difficult to modify the code so that continuous control is enabled.

```
function C=inventory(s,S)
%The (s,S)-inventory simulation - order-up-to -version
%Inventory is controlled once a month

y(1) = 60;                %Initial inventory
Tmax = 120;               %Total simulation time

held=0;                   %Number of held items
back=0;                   %Number of backlogged items

c_f=32;                   %Fixed order costs
c_v=3;                    %Order cost per item

order_cost=0;             %Total order costs during simulation

hold_cost=1;              %Holding cost of items
short_cost=5;             %Shortage cost of items

mtd = 0.1;                %Mean time between demand
dp = [1/6 3/6 5/6 1];     %Probabilities for size of demand

t(1)=0;      %simulation time
tp=0;        %time of previous event
D=0;         %demand of items
td=0;        %time, when the next demand is realized
tc=1;        %time, when inventory is next controlled
ts=0;        %time, when new items are next supplied
oa=0;        %order amount


%Time of first supply is first set to a large number
ts = Tmax+1;

%Time of first demand
td = exprnd(mtd);


while t(end) < Tmax

    if td <= ts && td <=tc          %A delivery is made

        %Simulation time
        t(end+1) = td;



        %Update counters for held or backlogged items
        if y(end) > 0
            held = held+y(end)*(t(end)-tp);
        elseif y(end) < 0
            back = back-y(end)*(t(end)-tp);
```

```
    end

    %Time of previous event
    tp = t(end);

    %Demand
    D = rand;
    for i=1:length(dp)
        if D<dp(i)
            D=i;break;
        end
    end

    %Inventory level
    y(end+1) = y(end) - D;

    %Time of next demand
    td = t(end) + exprnd(mtd);

elseif tc < td && tc < ts      %The inventory is controlled

    %Simulation time
    t(end+1) = tc;

    %Update counters for held or backlogged items
    if y(end) > 0
        held = held+y(end)*(t(end)-tp);
    elseif y(end) < 0
        back = back-y(end)*(t(end)-tp);
    end

    %Time of previous event
    tp = t(end);

    %Check, if inventory level is under s
    if y(end) < s
        ts = t(end) + unifrnd(0.5,1);    %Order delay
        oa = S-y(end);                   %Order amount
        order_cost =  order_cost + c_f + c_v*oa;
    end

    %Inventory level is not changed
    y(end+1) = y(end);

    %Time of next control - at the start of next month
    tc = t(end)+1;

elseif ts < td && ts < tc      %The inventory is replenished

    %Simulation time
    t(end+1) = ts;




    %Update counters for held or backlogged items
    if y(end) > 0
        held = held+y(end)*(t(end)-tp);
    elseif y(end) < 0
        back = back-y(end)*(t(end)-tp);
```

```
        end

        %Time of previous event
        tp = t(end);

        %Inventory level
        y(end+1) = y(end) + oa;
        ts = Tmax+1;
        oa = 0;

    end

    %Update counters from the remaining simulation time
    %in case all events are scheduled after Tmax
    if min(td,min(ts,tc)) >= Tmax

        t(end) = Tmax;

        if y(end) > 0
            held = held+y(end)*(t(end)-tp);
        elseif y < 0
            back = back-y(end)*(t(end)-tp);
        end
    end
end

%Calculate average monthly cost
C = held*hold_cost + back*short_cost + order_cost;
C = C/Tmax;
```

Results

10 simulation replications were made for both of the policies (20,40) and (20,80). The average monthly costs are presented in the following table.

(20,40)	(20,80)	Difference
129.19	122.32	6.86
125.70	122.02	3.68
127.89	122.13	5.76
129.97	124.35	5.61
132.39	122.95	9.43
139.12	124.06	15.07
120.80	121.79	-0.99
123.19	124.02	-0.83
125.67	124.64	1.03
123.99	120.53	3.47

The average difference between the systems across the 10 replications is 4.91 and the variance of differences 23.9. A paired $t_{1-0.025,9}$ confidence interval for the difference is [1.41,8.41] suggesting that the difference is statistically significant. The policy (20,80) could be expected to produce a lower cost of operation.

For more accuracy, simply add number of replications. Note, that 10 replications is not too much. The results you will obtain may therefore vary from the ones presented here.

2.3

Analytic solution

There is an analytic solution to this exercise, which is presented first.

The computer center forms a queuing system with exponential customer (jobs) arrivals, exponential service times (execution of jobs) and parallel servers. Such a system is commonly denoted as $M/M/s$-queue, where s is the number of servers. Jobs enter the system with constant intensity λ. The execution rate of jobs for all computers is equal and denoted with μ. Given that there are i jobs currently in the system, the overall execution rate of jobs for the computer center is

$$\mu_i = \begin{cases} i\mu & i=1,\dots,s \\ s\mu & i>s \end{cases}$$

In the equilibrium, the probability that there are i jobs in the system at a randomly chosen point in time is

$$\pi_i = \theta_i \pi_0, \text{ where}$$

$$\theta_i = \begin{cases} \frac{1}{i!}\left(\frac{\lambda}{\mu}\right)^i & i=1,\dots,s \\ \frac{1}{s!}\left(\frac{\lambda}{\mu}\right)^s \left(\frac{\lambda}{s\mu}\right)^{i-s} & i>s \end{cases}$$

Now, there are 25 users, each submitting a job every 15 minutes on the average. The overall arrival rate of jobs is therefore 100 jobs in an hour. Each computer executes a job in 2 minutes on the average, so the execution rate is 30 jobs in an hour for one computer and 120 jobs in total.

a)

The probability that a job can not be executed immediately as it is submitted equals the probability that all computers are busy. This probability is calculated as

$$P = \sum_{j=s}^{\infty} \pi_j = \frac{1}{1-\frac{\lambda}{s\mu}} \left(\frac{\lambda}{\mu}\right)^s \pi_0 = \frac{2500}{3801} \approx 0.66, \text{ where } \pi_0 = \frac{1}{\sum_{j=0}^{\infty} \theta_j}$$

b)

The average time W a job spends in the system is calculated (using Little's formula)

$$W=\frac{1}{\lambda}\sum_{j=0}^{\infty} j\pi_j=\frac{839}{12670}\approx 3 \text{ min } 58 \text{ sec.}$$

c)

The average number of jobs waiting to be executed L_Q is

$$L_q=\sum_{j=s}^{\infty}(j-s)\pi_j=\frac{\lambda}{s\mu-\lambda}\frac{1}{1-\frac{\lambda}{s\mu}}\frac{\left(\frac{\lambda}{\mu}\right)^s}{s!}\pi_0=\frac{12500}{3801}\approx 3.28 \text{ jobs}$$

d)

The percentage of time the computer center is idle equals the probability of observing an empty system

$$\sum_{j=0}^{\infty}\pi_j=1\rightarrow\sum_{j=0}^{\infty}\theta_j\pi_0=1\rightarrow\pi_0=\frac{1}{\sum_{j=0}^{\infty}\theta_j}$$

We get $\pi_0 = 27/1267$, which is approximately 2% of the time.

e)

The average number of idle computers is

$$E[\#idle]=\sum_{j=0}^{s}(s-j)\pi_j=\frac{2}{3}.$$

Simulation

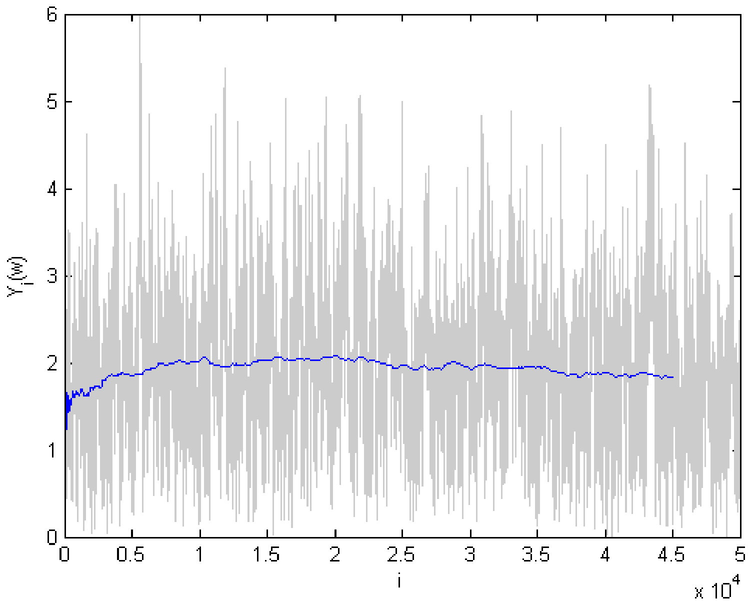
A Matlab-inplementation of the model could look like this:

```
function
[d,busy_time,total_time,ninq,idle_time,nidle]=cc(n,warmup,k,lambda,mu)
%Simulate a computer center with k computers, i.e., a (M/M/k)-queue
%n          the number of customers
%warmup     the lenght of warmup period (#customers)
%lambda     the mean inter-arrival time of jobs
%mu         the mean completion time of jobs


t=0;                    %Simulation clock
```

```
ta=exprnd(lambda);   %The time of next job arrival
td=inf;              %Departure time of jobs (set to a dummy value)

warmup_t=0;          %Length of the warmup in simulated time

served=[];           %Jobs being completed
queue=[];            %Queued jobs

n_in=0;              %Number of job arrivals
n_out=0;             %Number of job departures

d=[];                %Queueing delays of jobs in the system
busy_time=0;         %a) The fraction of time all computers are busy
total_time=0;        %b) Mean total time of jobs in the system
ninq=0;              %c) Average number of jobs in queue
idle_time=0;         %d) The fraction of time all computers are idle
nidle=0;             %e) Average number of idle computers


%The main simulation loop
while n_out<n

    if ta<td                        %Next event is arrival

        %Update statistics
        n_in = n_in+1;

        if n_in > warmup

            if length(served)==k
                busy_time = busy_time+(ta-t);             %a)
            end

            total_time = total_time+(ta-t)*(length(queue)+length(served));
%b)

            ninq = ninq+(ta-t)*length(queue);             %c)

            if length(served)==0
                idle_time = idle_time+(ta-t);             %d)
            end

            nidle = nidle+(ta-t)*(k-length(served));      %e)

        end

        %Add either to server or queue
        if length(served)<k
            served = sort([served ta+exprnd(mu)]);
            d(end+1)= 0;
        else
            queue = [queue ta];
        end

        %Update simulation clock
        t = ta;
        ta = ta+exprnd(lambda);
        td = served(1);


    else                            %Next event is departure
```

```
        %Update statistics
        n_out = n_out+1;

        if n_in > warmup

            if length(served)==k
                busy_time = busy_time+(td-t);            %a)
            end

            total_time = total_time+(td-t)*(length(queue)+length(served));
%b)

            ninq = ninq+(td-t)*length(queue);            %c)

            if length(served)==0
                idle_time = idle_time+(td-t);            %d)
            end

            nidle = nidle+(td-t)*(k-length(served));     %e)

        end

        %Departure
        served=served(2:end);

        %Take next customer from queue, if queue is not empty
        if length(queue)>0
            served = sort([served td+exprnd(mu)]);
            d(end+1)= td-queue(1);
            queue = queue(2:end);
        end

        %Update simulation clock
        t = td;
        if isempty(served)
            td = inf;
        else
            td = served(1);
        end

    end

    %Record length of warmup in time
    if n_in > warmup && warmup_t==0
        warmup_t=t;
    end

end

%Return exactly n delays
d=d(1:n);

busy_time = busy_time/(t-warmup_t);
total_time = total_time/(n_in-warmup);
ninq = ninq/(t-warmup_t);
idle_time = idle_time/(t-warmup_t);
nidle = nidle/(t-warmup_t);
```

The objective is to study the steady-state performance measures of the queuing system. To do this, we first determine the length of the warm-up period with Welch's procedure. The waiting time in queue is selected as the performance measure to determine the length of the transient. 10 independent replications of the simulation were conducted, each with a length of 50000 jobs. The resulting queuing times were first averaged across replications. The resulting process was further smoothed by calculating its moving average with a window of 5000 jobs. According to the figure, the system seems to settle to a covariance stationary state after approximately 10000 jobs. This period should be around 100 hours in length with arrival rate of 100 jobs an hour.



The performance measures of interest can now be determined by replicating the simulation and by discarding the statistics for the first 10000 jobs at each replication. You should get results that are very close to the analytic solutions.

2.4

Here is an attempt:

```
function [d,u,nq]=queue2(n,k,lambda,mu)
      %Simulate delays in queue for queuing model (M/M/k)
      %n the number of customers
      %k the number of servers
      %lambda the arrival rate of customers
      %mu the service rate for all servers


      t=0;                    %Simulation clock
      ta=exprnd(lambda);      %The time of next customer arrival
      td=ta+1;                %Departure time of customers (set to a dummy
      value)

      served=[];              %Customers being served
      queue=[];               %Queued customers

      n_in=0;                 %Number of customer arrivals
      n_out=0;                %Number of customer departures

      d=[];                   %Service delays
      u=0;                    %Utilization of server
      nq=0;                    %Number in queue statistic

      %The main simulation loop
      while n_out<n

          if ta<td                    %Next event is arrival

              %Update statistics
              n_in = n_in+1;
              u = u+(ta-t)*length(served)/k;
              nq = nq+(ta-t)*length(queue);

              %Add either to server or queue
              if length(served)<k
                  served = sort([served ta+exprnd(mu)]);
                  d(end+1)= 0;
              else
                  queue = [queue ta];
              end

              %Update simulation clock
              t = ta;
              ta = ta+exprnd(lambda);
              td = served(1);


          else                        %Next event is departure

              %Update statistics
              n_out = n_out+1;
              u = u+(td-t)*length(served)/k;
              nq = nq+(td-t)*length(queue);

              %Departure
              served=served(2:end);
```

```
            %Take next customer from queue, if queue is not empty
            if length(queue)>0
                served = sort([served td+exprnd(mu)]);
                d(end+1)= td-queue(1);
                queue = queue(2:end);
            end

            %Update simulation clock
            t = td;
            if isempty(served)
                td = ta+1;
            else
                td = served(1);
            end

        end
    end

    u=u/t;
    nq=nq/t;
```

Calculation of the performance indicators should be easy. Just construct a confidence interval individually for each indicator. Utilization for two servers is calculated as the time-weighted fraction of servers in use. Results can look something like:

1. Average queueing delay: 4.3±0.62
   Number in queue: 4.5±0.67
   Server utilization: 0.83±0.02

2. Average queueing delay: 1.89±0.32
   Number in queue: 3.98±0.70
   Server utilization: 0.84±0.02