DUBLIN CITY UNIVERSITY

CA4011 Operations Research

C. Simulation

Contents

1. Simulation

1.1 Introductory Example: Application to Queues

The following exercises are posed, by way of introduction, in the first chapter of (Ross, 2002) (DCU library ref 519.2/ROS):

Exercise 1: "The following data yield the arrival times and service times[1] that each customer will require, for the first 13 customers of a single-server system. Upon arrival, a customer either enters service if the server is free or joins the waiting line. When the server completes work on a server, the next one in line (i.e. the one who has been waiting the longest) enters service.

Arrival Times	12	31	63	95	99	154	198	221	304	346	411	455	537
Service Times	40	32	55	48	18	50	47	18	28	54	40	72	12

(a) Determine the departure times of these 13 customers.
(b) Repeat (a) when there are two servers and a customer can be served by either one.
(c) Repeat (a) under the new assumption that when the server completes a service, the next customer to enter service is the one who has been waiting the least time."

Note: We can calculate the average service and inter-arrival times directly from the given data:

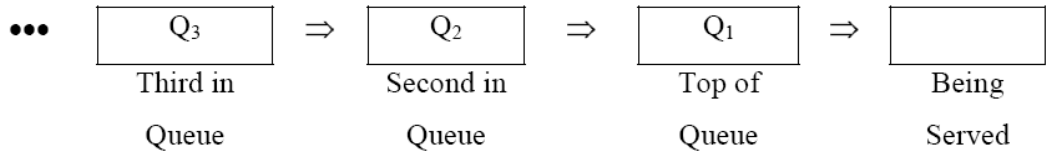
Average service time = $(40 + 32 + \ldots + 72 + 12)/13 = 39.54$
Average inter-arrival time = $(19 + 32 + \ldots + 82)/12 = 43.75$

Hence, we have

Arrival rate = $\lambda = 1/43.75$
Service rate = $\mu = 1/39.54$

In queue theory the ratio λ/μ is called the traffic intensity. The closer this ratio gets to 1, the longer the queue becomes. If $\lambda/\mu > 1$ then we will have queues that grow longer and longer – as in rush hour traffic.

Solution attempt for 1(a):
Let Cn denote customer n
Let the queue system be depicted as



A key step is to start with the notion of an **event** of which we have
(1) A customer arrives
(2) A customer joins the queue at position Q_n
(3) A customer enters service
(4) A customer departs service.
Hence, we can form a timeline of events:

[1]The word "times" is used here with two different meanings. Customer arrival time is when the customer arrives to the system; it is sometimes called the epoch and denote a point on the time axis. On the other hand, service time means the duration of a service.

Time	Event(s)	Departure Times
12	C1 arrives; C1 enter service	⇒ C1 departs at 52
31	C2 arrives; C2 joins at Q1	
52	C1 departs; C2 enter service	⇒ C2 departs at 84
63	C3 arrives; C3 joins at Q1	
84	C2 departs; C3 enter service	⇒ C3 departs at 139
95	C4 arrives; C4 joins at Q1	
99	C5 arrives; C5 joins at Q2	
139	C3 departs; C4 enter service; C5 goes to Q1	⇒ C4 departs at 187
154	C6 arrives; C6 joins at Q2	
187	C4 departs; C5 enter service; C6 goes to Q1	⇒ C5 departs at 205
198	C7 arrives; C7 joins at Q2	
205	C5 departs; C6 enter service; C7 goes to Q1	⇒ C6 departs at 255
221	C8 arrives; C8 joins at Q2	
255	C6 departs; C7 enter service; C8 goes to Q1	⇒ C7 departs at 302
302	C7 departs; C8 enter service	⇒ C8 departs at 320
304	C9 arrives; C9 joins at Q1	
320	C8 departs; C9 enter service	⇒ C9 departs at 348
346	C10 arrives; C10 joins at Q1	
348	C9 departs;C10 enters service	⇒ C10 departs at 402
402	C10 departs	
411	C11 arrives; C11 enter service	⇒ C11 departs at 451
451	C11 departs	
455	C12 arrives; C12 enter service	⇒ C12 departs at 527
527	C12 departs	
537	C13 arrives; C13 enter service	⇒ C13 departs at 549
549	C13 departs	

The rightmost column above provides the answer to part (a).
Note: Average time in System = Average time in Queue + Average Service Time

Note: It may also be of interest to calculate the time each customer spends in the system and queueing. From the timeline above we obtain the following table based on the observation that "(time in system) = (time of departure) – (time of arrival)":

Customer	Time in system	Service time	Queue Time
1	40	40	0
2	53	32	21
3	76	55	21
4	92	48	44
5	106	18	88
6	101	50	51
7	104	47	57
8	99	18	81
9	34	28	6
10	54	54	0
11	40	40	0
12	72	72	0
13	12	12	0
Averages	67.92	39.54	28.38

Note on comparison with mathematical models of simplified systems: Often when simulating queue systems, we use known mathematical models of simplified systems as a check on our simulation results.
For example, in Part B: section 2 of this module, we have results for a steady-state queue with a single server with exponentially distributed inter-arrival and service times (denoted M/M/1). For such a system,
Average No. in system = $L = \lambda / (\mu - \lambda) = 9.39$ (for our example)
There is also a (general) relationship (Little's relationship) that
Average time in system = $W = L/\lambda$
For our example, this gives 410.73 seconds which is far greater than 67.92. Clearly, our queue is far from being M/M/1.
Average queue length (estimate): There is also a Little relationship $Wq = Lq/\lambda$, relating the average time queueing (Wq) to the average number in the queue (Lq). Applying this to our system we find $28.38 = Lq\ (43.75) \Rightarrow Lq = 28.38/43.75 = 0.6$
Problems: Answer 1(b) and 1(c).

Exercise 2: "Consider a service station where customers arrive and are served in their order of arrival. Let A_n, S_n and D_n denote, respectively, the arrival time, the service time (i.e. duration), and the departure time of customer n. Suppose there is a single server and that the system is initially empty of customers.
(a) With $D_0 = 0$, argue that for $n > 0$

$$D_n - S_n = \text{Maximum}(A_n, D_{n-1})$$

(b) Determine the corresponding recursion formula when there are two servers.
(c) Determine the corresponding recursion formula when there are k servers.
(d) Write a computer program to determine the departure times as a function of the arrival and service times and use it to check your answers to parts (a) and (b) of Exercise 1."

Solution 2(a): We have two cases:

- If the queue is empty when customer n arrives then that customer goes straight into service. Hence,
Departure time = Arrival time + Service time i.e. **Dn = Sn + An**
- If the queue is not empty then the customer cannot enter service until the customer immediately before (i.e. sustomer n-1) departs. Hence,
Departure time = Dep. time of customer n-1 + Service time i.e. **Dn = Sn + Dn-1**
Combining these formulae gives **Dn = Sn + Maximum{An, Dn-1}**

Applying this answer to **Exercise 1(a)** we get

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13
D_n	0	52	84	139	187	205	255	302	320	348	402	451	527	549

Problems: Answer 2(b), 2(c) and 2(d).

1.2 Another application to queues (using simulated data)

The following example is taken from section 5.4 of (Cox & Smith, 1961). It introduces some notions that are basic to statistical simulation, including

–generating random numbers from different probability distributions

–idea of replicating "experiments" several times in order to obtain error estimates or confidence intervals for parameters of interest.

Suppose that there are two (identical) servers, that the distribution of service-time is exponential with mean 1.5 units (so that service rate = $\mu = 2/3$) and that the arrival pattern is as follows:

Arrivals are scheduled regularly at times 1, 2, 3, … but the actual arrival points are independently normally distributed around the scheduled arrival points as mean and with standard distribution ½ unit.
We can see that the average arrival rate (no. of arrivals per unit time) is $\lambda = 1$.
Hence, $\lambda/\mu = 3/2$ so that traffic intensity (see Part B: section 2 of this module) $\rho = \lambda/(2\mu) = ¾$.

Note: If there were random arrivals then this would be an M/M/2 system and, using the results of Part B: section 2 of this module, we could calculate that the average number in the system = $L = 2\rho/(1 - \rho^2) = 24/7$.
Then, the average waiting time in the system would be $W = L/\lambda = 24/7$.
Hence, the average waiting time in the queue would be $W - 1/\mu = 27/14 = 1.93$.
These results for an "idealized" model will be useful as a check on the simulation results presented below.

Simulation steps:
(1) Generate appropriately distributed random quantities. (*In section 2.1 the data were given so this step was not required*). We will define how this can be done for different probability distributions (including normal and exponential) shortly.

[Aside: Example of simulation of arrival times in R and in Matlab:
(a) In the R language, we could simulate the actual arrival times by,

```
arrive<-rnorm(1,1,.5)
```

```
for (i in 2:200)arrive[i]<-rnorm(1,i,0.5)
round(arrive,2)
```

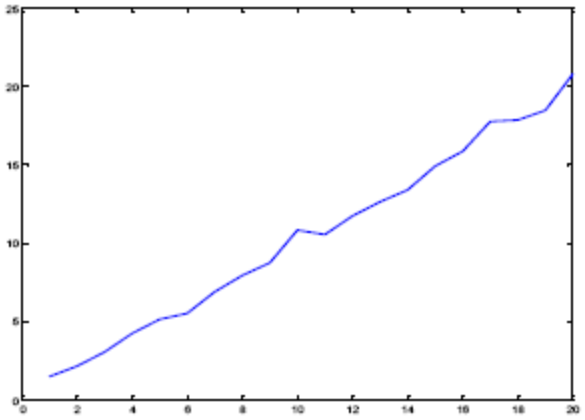
(b) In Matlab, we could have (taking 20 rather than 200 for brevity):

```
arrive=1+0.5*randn(1); % Sets first entry of 20
for i=2:20
arrive=[arrive,i+0.5*randn(1)];
end
plot(arrive)
arrive
```

This yields the data

```
arrive =1.4746  2.1535  3.0676  4.2576  5.1307  5.5293  6.9188
7.9270  8.7340  10.8411  10.5621  11.7581  12.6440  13.4129
14.9039 15.8630 17.7650 17.8755 18.4679 20.8017
```

and the plot



End of Aside]

(2) Define an initial condition. Here, the system is assumed empty at the start.
(3) Carry out the simulation (similar to Exercise 1 of section 2.1 except here there are two servers). The following table (from (Cox & Smith, 1961)), which shows the queueing-time for the first 12 customers, illustrates the nature of the calculations.

Outline of simulation calculation sheet for estimating mean queueing-time.

Customer number	Arrival deviation	Actual arrival time	Service-time	Admitted to server 1	Departs from server 1	Admitted to server 2	Departs from server 2	Queueing-time
1	0·4	1·4	0·2	1·4	1·6			0
2	−0·3	1·7	2·7	1·7	4·4			0
3	−0·2	2·8	1·7			2·8	4·5	0
4	0·4	4·4	4·3	4·4	8·7			0
5	0·4	5·4	1·7			5·4	7·1	0
6	−0·3	5·7	0·6			7·1	7·7	1·4
7	−0·3	6·7	2·7			7·7	10·4	1·0
8	−0·4	7·6	1·0	8·7	9·7			1·1
9	−0·5	8·5	0·6	9·7	10·3			1·2
10	0·6	10·6	3·6	10·6	14·2			0
11	−0·1	10·9	0·7			10·9	11·6	0
12	−0·1	11·9	0·9			11·9	12·8	0
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

Queueing time = Departure time – arrival time – Service time
–E.g. **customer 6**: 7.7 – 5.7 – 0.6 = 1.4

The simulation was continued for 200 customers, with the following results (presented in batches of 20):

TABLE 5.3

Properties of queueing-time estimated by simulation.

	Propn. of zero queueing-times	Mean queueing-time	Maximum queueing-time
1st 20 customers	0·90	0·02	0·2
2nd " "	0·85	0·07	0·8
3rd " "	0·60	0·35	1·8
4th " "	0·65	0·32	1·8
5th " "	0·65	0·23	1·2
6th " "	0·60	0·39	1·6
7th " "	0·30	1·18	3·1
8th " "	0·90	0·04	0·6
9th " "	0·70	0·13	1·1
10th " "	0·10	2·72	5·4
Combined	0·62	0·55	5·4

Note: There may be an error in the above table but that does not take from the principle.

Note 1: The mean queueing time of 0.55 may be compared with the result of 1.93 under M/M/2 assumptions (see note above on "idealized" model). We can see that the estimated queueing time value for the queue with regular arrivals is much lower than that for random arrivals which is probably to be expected (?).

Note 2: We won't do it but one can analyse the queue with regular arrivals mathematically and come up with the value of 0.71 for the "true" mean queueing time. So, our estimate of 0.55 is somewhat low and this error is due to estimation errors. This leads on to the consideration of "**variability of results**" from simulations.

Note on "Variability of Results" from simulations:
For our example, we want to estimate the standard error of the final mean queueing time. Cox & Smith suggest two methods:
(1) (**Preferred where practicable**) Repeat the whole procedure a number of times (M, say) independently from the beginning, calculating a mean queueing time for each run. This results in a set of queueing means. The standard error of the pooled mean is then

$$s/\sqrt{M}$$

where s is the standard deviation of the set of means.
One must make sure that
(a) each individual run is long enough, and
(b) that M is large enough
for the purposes of the work.

(2) This is based on a single run where the results are divided into sections (batches) each of which is sufficiently long for the correlation between the mean queueing times in adjacent sections to be negligible (may involve trial and error!). In Table 5.3 of (Cox & Smith, 1961) (above) the customers were divided into 10 sections of 20 customers and a mean queueing time (say, $T_1 = 0.02$, … $T_{10} = 2.72$) found for each section (as shown).
The mean of these is then calculated as

$T = 0.55.$

Then, one can calculate the standard deviation as

$s = \sqrt{[(1/9)\sum(T - T_i)^2]} = 0.83.$

Finally, the standard error is calculated as

$s/\sqrt{10} = 0.26$

This allows one to give a (rough) estimate of the mean queueing error as

$0.55 \pm 0.26.$

If the standard error is too large (as here probably) then one has an approximate means of predicting how many more customers, sections etc are needed to obtain a sufficiently small value.

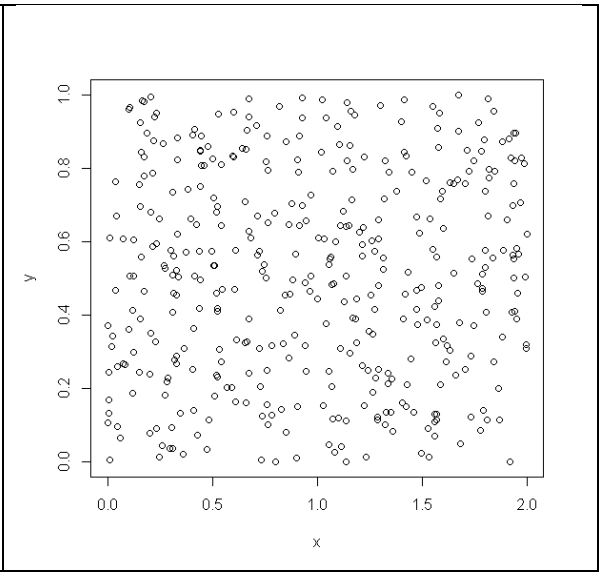
1.3 Estimate π/2 by simulation

We show the basic idea using some R code and then more detailed calculations using Mathematica. Languages such as R, Matlab, Mathematica etc are convenient for simulation work but general purpose languages like Java or C++ can also be used of course.

(a) R code:

```
x<-runif(400,0,2)
y<-runif(400,0,1)
plot(x,y)
```

results in a plot like that opposite.



Next, how many points lie in semi-circle?

```
count<-0
for (i in (1:400))(if
((x[i]-1)^2+y[i]^2<=1)
count<-count+1)
count
```

302

Finally, find

```
(count/400)*2
```

1.51

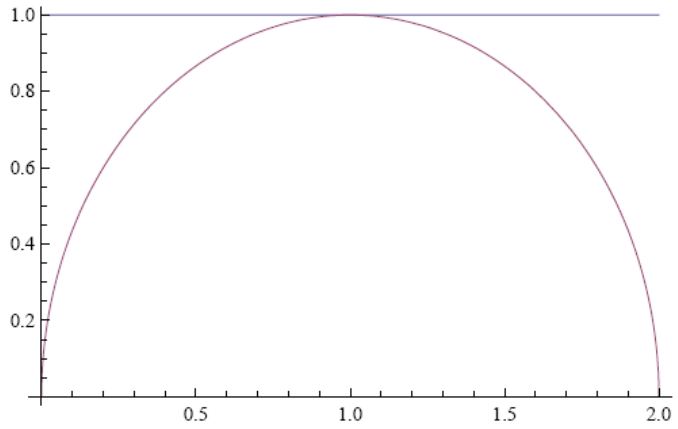
This is the estimated area of the semi-circle, that is π/2.

Note: We used that ratio 400:2 = count:(π/2) approximately.

(b) Mathematica code:

Plot of area concerned, embedded in a 2 X 1 rectangle

```
Plot[{1, Sqrt[1 - (1 - x)^2]}, {x, 0, 2}, AxesOrigin → {0, 0}]
```



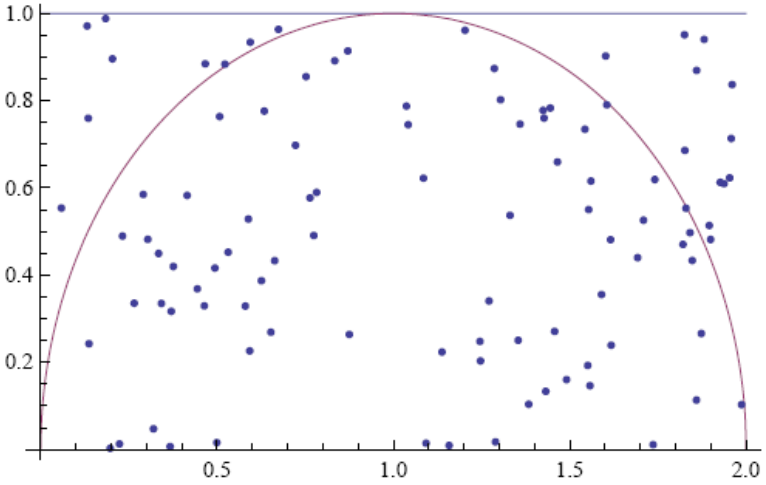
Note: Clearly, x is in [0, 2], y is in [0, 1]

Sample calculation of 400 random pairs of (x, y) values:

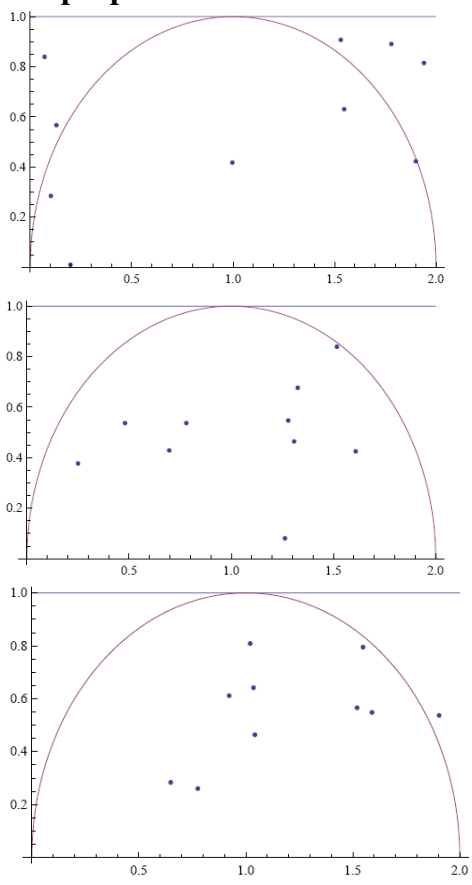
```
In[16]:= data = Table[{Random[Real, {0, 2}], Random[]}, {i, 1, 400}]

Out[16]= {{0.7143, 0.190783}, {1.9287, 0.754654}, {1.69409, 0.382084}, {1.00644, 0.1432
  {1.79096, 0.845446}, {1.41667, 0.595071}, {0.754852, 0.203811}, {1.8966, 0.02
  {0.105761, 0.919837}, {1.12103, 0.768346}, {1.06342, 0.456087}, {1.57327, 0.9
  {0.349115, 0.265303}, {1.64458, 0.218696}, {0.655025, 0.883219}, {0.638132, 0
  {0.864064, 0.0377735}, {1.22146, 0.480355}, {0.109212, 0.833963}, {1.32486, 0
  {0.00345055, 0.914125}, {0.203826, 0.68298}, {0.940035, 0.458038}, {0.630551,
  {0.590919, 0.192735}, {0.985975, 0.490935}, {1.93589, 0.309516}, {0.347843, 0
  {1.07183, 0.271742}, {1.12638, 0.935154}, {0.962619, 0.43778}, {1.80152, 0.48
  {0.959169, 0.523654}, {1.5977, 0.800848}, {0.0191337, 0.0656158}, {0.967144,
  {1.42821, 0.872881}, {1.98117, 0.600282}, {1.49232, 0.563365}, {1.63333, 0.18
  {0.420489, 0.291622}, {0.506944, 0.249619}, {1.45787, 0.853843}, {0.705422, 0
```

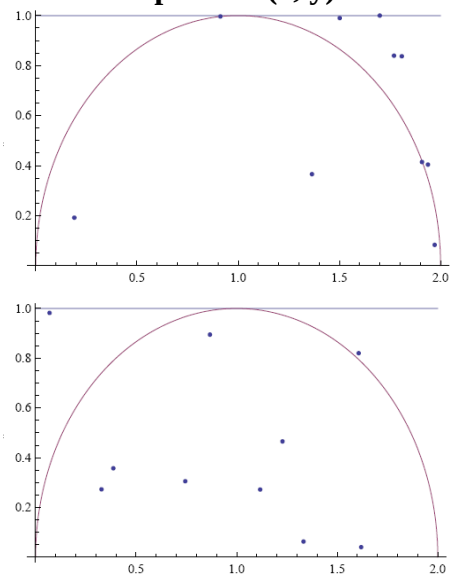
Sample plot of a set of 100 random pairs of (x, y) values:

```
Show[Plot[{1, Sqrt[1 - (1 - x)^2]}, {x, 0, 2}, AxesOrigin → {0, 0}],
 ListPlot[Table[{Random[Real, {0, 2}], Random[]}, {i, 1, 100}]]]
```



Sample plots & area estimation: 5 sets of 10 random pairs of (x, y) values:





Processing the no. of points in the semi - circle in the 5 cases:

```
internals = {5, 5, 10, 8, 9} / 10

{1/2, 1/2, 1, 4/5, 9/10}

areaEstimates = internals * 2

{1, 1, 2, 8/5, 9/5}

semiCircleAreaEst = Mean[areaEstimates]

1.48
```

1.4 Elements of Simulation

1.4.1 Key aspects

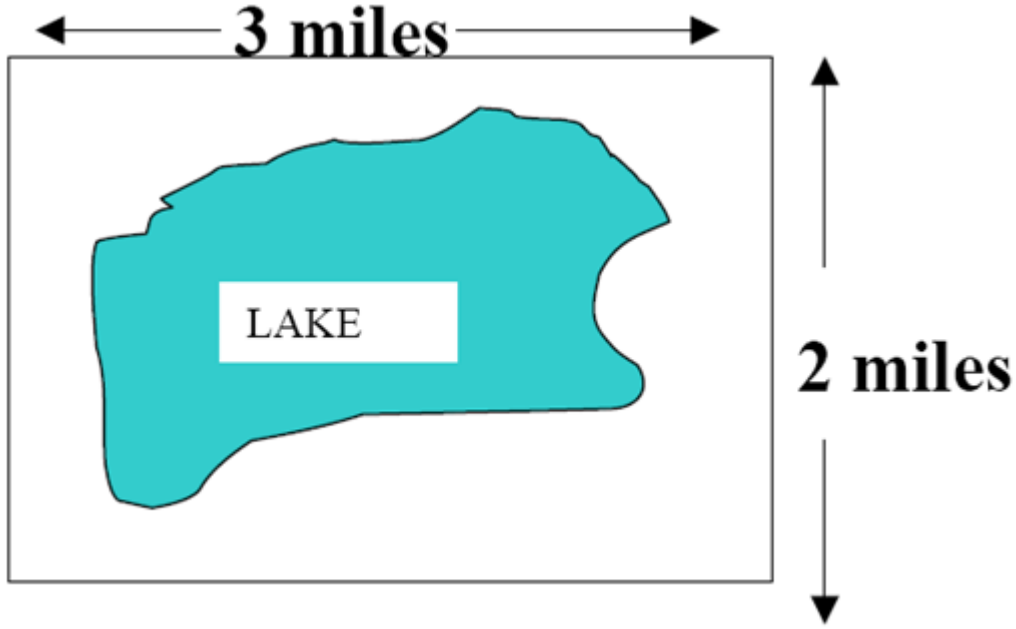
- Conducting experiments using logical relationships to describe a real world system.
- Analytical solution difficult or impossible
- "Method of last resort"

The following quotation (due to Smith) from (Boccara, 2004) in distinguishing between modelling and simulation may be of interest. The focus is on ecology but similar comments are valid for other areas.

"*If, for example, one wished to know how many fur seals can be culled annually from a population without threatening its future survival, it would be necessary to have a description of that population, in its particular environment, which includes as much relevant detail as possible. At a minimum, one would require age-specific birth and death rates, and knowledge of how these rates varied with the density of the population, and with other features of the environment likely to alter in the future. Such information could be built into a simulation of the population, which could be used to predict the effects of particular management policies.*

The value of such simulations is obvious, but their utility lies mainly in analysing particular cases. A theory of ecology must make statements about ecosystems as a whole, as well as about particular species at particular times, and it must make statements which are true for many different species and not just one. Any actual ecosystem contains far too many species, which interact in far too many ways, for simulation to be a practical approach. The better a simulation is for its own purposes, by the inclusion of all relevant details, the more difficult it is to generalize its conclusions to other species. For the discovery of general ideas in ecology, therefore, different kinds of mathematical descriptions, which may be called models, are called for. Whereas a good simulation should include as much detail as possible, a good model should include as little as possible."

1.4.2 Outline of another "area" example (lake)



Obviously, Area of rectangle is known (3 x 2) = 6 square miles
Method: pick sample points at random – uniformly distributed in the rectangle
Question: For each point decide whether or not it is located in lake?

So you would end up with perhaps a table such as

	Yes	No
Point 1	……..	……..
Point 2	……..	……..
Point 3		
Point 4		
Point 5		
Point 6		
Point 7		
……..		
……		

Estimate of lake area = 6(no. points in lake)/(total no. of points) in square miles

Note: The more points chosen (i.e. the greater the sample size), the greater the accuracy of estimate of lake area.

N.B. sampling error – some attempt should be made to quantify this, and this is commonly done through **replications**.

- The experiment could be repeated, to give further estimates of lake size. Each repetition is called a **replication**.
- Final estimate of lake size = average of estimate for each replication.
- Also, as we have seen for a queueing example and for our $\pi/2$ example, we may also estimate the standard error from the set of replication estimates.

1.4.3 Terminology

Static simulation model – fixed point in time.

Dynamic simulation model – evolves over time.

- deterministic (no random variables)
- stochastic (at least one **random** variable)

System: collection of entities which interact to accomplish some logical end.

State: status of system, described by state variables.

Entities (e.g. customers) can have attributes.

Discrete system – state variables change at countable points in time (e.g. when customer arrives/departs).

Continuous system – e.g. chemical process.

Discrete stochastic models are called *Discrete Event Simulation Models.*

Event – situation which causes state of system to change (e.g. arrival, departure).

Event list – schedule of events.

Clock time – time in a simulation.

1.4.4 Monte Carlo Sampling

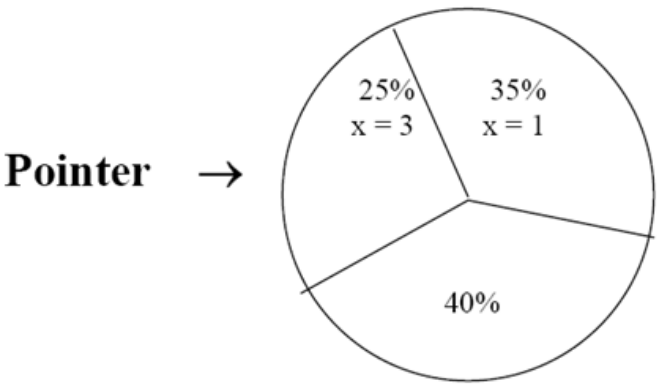
Times of events are generated by sampling from probability distributions.

Example: service times are either 1, 2 or 3 minutes.

Service time	Probability
1	0.35
2	0.40
3	0.25

Approach: We can generate random service times by spins of a roulette wheel:
e.g. 100 numbers on roulette wheel –

00 – 34 → 1 minute
35 – 74 → 2 minutes
75 – 99 → 3 minutes



For example, using Matlab in which the function **`rand(n,m)`** generates an nxm matrix of (pseudo-) uniformly distributed random numbers in [0, 1], we have

```
numbers=round(100*rand(1,100))
```

giving, for example, the set of numbers

64 38 81 53 35 94 88 55 62 59 21 30 47 23 84 19 23
17 23 44 31 92 43 18 90 98 44 11 26 41 59 26 60 71
22 12 30 32 42 51 9 26 80 3 93 73 49 58 24 46 96
55 52 23 49 62 68 40 37 99 4 89 91 80 10 26 34 68

14 72 11 65 49 78 72 90 89 33 70 20 3 74 50 48 90 61 62 86 81 58 18 24 89 3 49 17 98 71 50 47

Thus, the first will be 2 minutes service, the second 2 minutes service, the third 3 minutes service, and so on.

We can easily check the totals in the three categories for this sample as follows.

(A)

The Matlab function **`find(…)`** allows us to find the indices (1 to 100) of those numbers that are less than 35:

```
index1=find(numbers<35)
```

index1 = 11 12 14 16 17 18 19 21 24 28 29 32 35 36 37 38 41 42 44 49 54 61 65 66 67 69 71 78 80 81 91 92 94 96

We then check the size (no. or rows and columns) of this index vector to see how many numbers are less than 35:

```
>> size(index1)
```

ans = 1 34

So, we have 34 values less than 35 in this sample set.

(B)

We exclude the numbers less than 35 using

```
numbers(index1)=[]
```

This leaves

numbers = 64 38 81 53 35 94 88 55 62 59 47 84 44 92 43 90 98 44 41 59 60 71 42 51 80 93 73 49 58 46 96 55 52 49 62 68 40 37 99 89 91 80 68 72 65 49 78 72 90 89 70 74 50 48 90 61 62 86 81 58 89 49 98 71 50 47

(C) Next we do step (A) for values less than 75:

```
>> index2=find(numbers<75)
```

index2 = 1 2 4 5 8 9 10 11 13 15 18 19 20 21 22 23 24 27 28 29 30 32 33 34 35 36 37 38 43 44 45 46 48 51 52 53 54 56 57 60 62 64 65 66

We check size to find

```
>> size(index2)
```

ans = 1 44

So, we have 44 values greater than 34 and less than 75 in this sample.

So, we conclude that for this sample of random number, that there will be 34 "1-minute" services, 44 "2-minute" services and 22 "3-minute" services.

If we were to repeat (***replicate***) with a different set of random numbers, we would get somewhat different sub-totals.

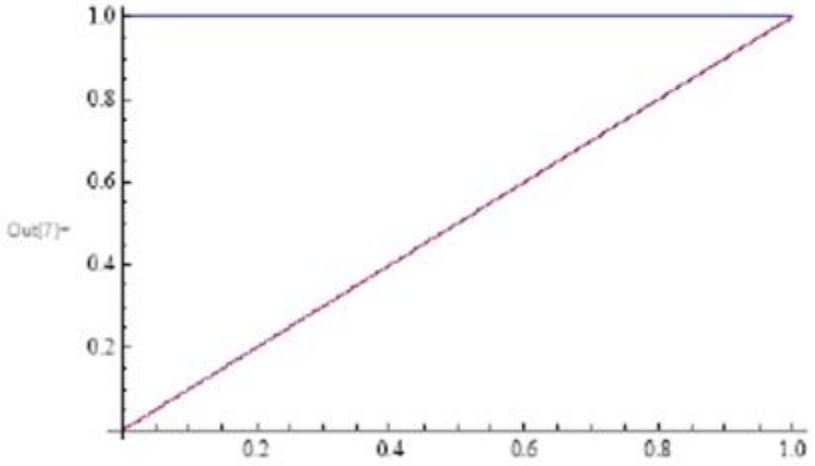
1.4.5 Some common probability distributions

We recall a few of the most commonly used probability distributions, displaying their probability density functions (pdfs) and cumulative distribution functions (cdfs). We have used Mathematica to depict the curves in this case.

■ 1. Uniform distribution (in [0, 1])

```
In[5]:= uPDF[x_] := 1;
        uCDF[X_] := x;

In[7]:= Plot[{uPDF[x], uCDF[x]}, {x, 0, 1}]
```

Out[7]=



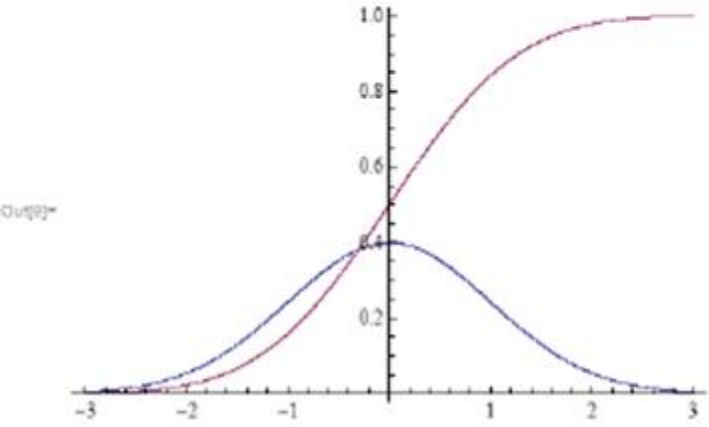
The **pdf** is in blue and **cdf** in red.

Note: For future reference note that the cdf is a non-decreasing function with values between 0 and 1.

■ 2. Normal distribution (μ=0, σ=1) [Built - in to *Mathematica*]

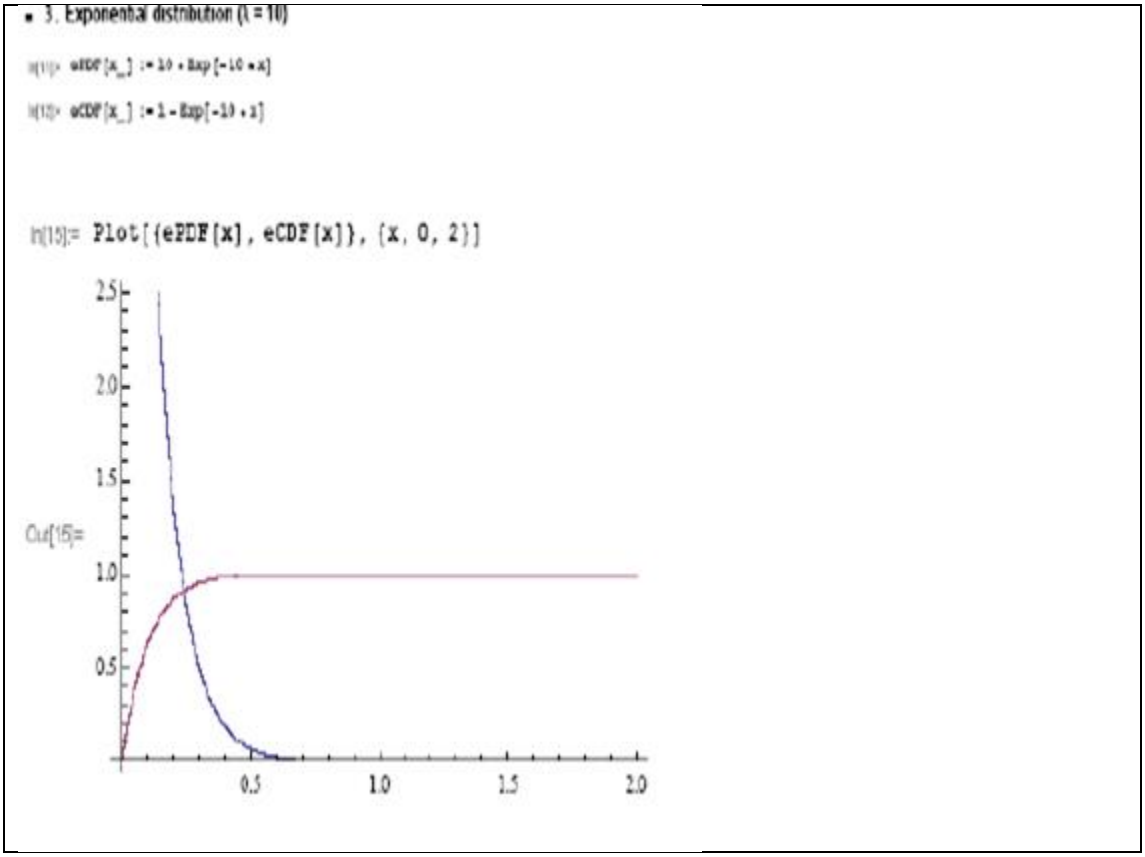
```
In[4]:= nPDF[x_] := PDF[NormalDistribution[0, 1], x]

In[8]:= nCDF[x_] := CDF[NormalDistribution[0, 1], x]

In[9]:= Plot[{nPDF[x], nCDF[x]}, {x, -3, 3}]
```

Out[9]=



This is the standard normal with mean 0 and standard deviation 1.

The **pdf** is again in blue and **cdf** in red. As previously, note that the cdf is a non-decreasing function with values between between 0 and 1.

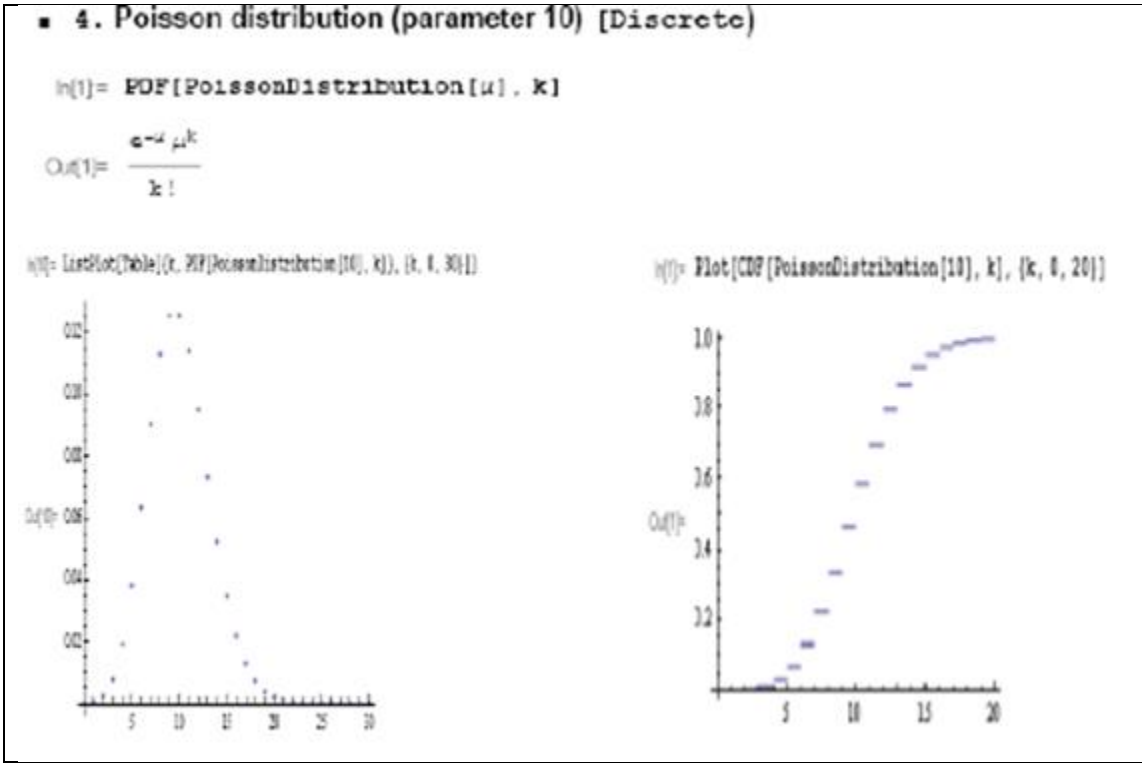


In general, the pdf and cdf of this distribution are, respectively,

$f(x;\lambda) = \lambda e^{-\lambda x}$ if $x \geq 0$, $= 0$ if $x < 0$
$F(x;\lambda) = 1 - e^{-\lambda x}$ if $x \geq 0$, $= 0$ if $x < 0$

Above, we have depicted the case of $\lambda = 10$.

The **pdf** is again in blue and **cdf** in red. As previously (and always), note that the cdf is a non-decreasing function with values between between 0 and 1.



The diagram is not very clear so we restate that the pdf (or probability distribution as this is a discrete distribution) is

$\lambda^k e^{-\lambda}/k!$ = Probability of k events in an interval.
Above, we have the case of $\lambda = 10$.

The formula for the CDF is a bit messier (a summation).
As previously (and always), note that the cdf is a non-decreasing function with values between between 0 and 1.

Note: There are various other distributions that may arise in (i.e. be appropriate for) simulations.

1.4.6 Generating pseudo-random numbers for Uniform distribution

In general there are two aspects to consider if using random deviates for a simulation:

- Select probability distributions that adequately describe the physical process(es) being simulated
- Generate random deviates from the chosen distribution(s)

Most software languages used for simulation, and most general purpose programming languages, have built-in pseudo-random number generators. For the purpose of this module, these generators will be fine. However, for more exact work, it would be important to make sure that the generators used have appropriate properties.

In this section, we give a flavour of what is involved in random number generators (for the **uniform distribution**) and identify some desirable properties. The uniform

distribution is particularly important as random deviates for other distributions are most often calculated from uniformly distributed random numbers. Thus,

Steps involved:

1. Generate random numbers from uniform distribution.
2. Perform **transformation** on the uniform random numbers (of step 1) to obtain deviates of required distribution (see section 1.4.7).
3. Use deviates (of step 2) to experiment.

Many programming languages provide functions to calculate random deviates for the more commonly used probability distributions directly so that, in practice, we may not have to perform Step 2 explicitly (i.e. it is done within the provided functions).

Uniform distribution & pseudo-random generation

Recap

Probability Density function of the uniform distribution:

Over interval [a , b]

$$f(x) = 1 / (b - a)$$

Over interval [0 , 1]

$f(x_0)$ = probability $x = x_0$

$f(x_0) = 1$ $0 \leq x_0 \leq 1$

$= 0$ otherwise

Probability Distribution function of the uniform distribution (cumulative probability)

Over interval [0, 1]:

$F(x_0)$ = probability $x \leq x_0$

$F(x_0) = x_0$ $0 \leq x_0 \leq 1$

Random number generators:

Algorithmic/recursive type formulae, called random number generators, are used to generate uniform pseudo-random numbers. We outline some considerations for these.

Required properties of random number generator:

1. Numbers distributed as uniformly as possible
2. Fast
3. No large memory requirement
4. Long cycle (i.e. very long time for number pattern to repeat)
5. Does not degenerate to a constant
6. Different set of numbers from different starting point.

Some Methods for generating pseudo random numbers:

Fibanocci method: (*between 0 and 100*)

$X_{n+1} = (X_n + X_{n-1}) \bmod (M)$

For example: $X_0 = 22 \quad X_1 = 55 \quad M = 100$ *(parameters)*

$X_2 = (X_1 + X_0) \bmod (100)$

$= (55 + 22) \bmod (100) = 77$

$X_3 = (77 + 55) \bmod (100) = 32$

etc.

Multiplicative Congruential methods (most common):

$u_n = (b\, u_{n-1} + c) \bmod (m)$ $\quad n = 1, 2, \ldots\ldots$

$R_n = u_n / m$ $\quad n = 1, 2, \ldots\ldots$

u_0 , b, c and m are parameters

u_0 = "seed"

For example: $b = 9,\ c = 5,\ u_0 = 11,\ m = 12$

$u_1 = (9 \times 11 + 5) \bmod (12) = 8 \quad R_1 = .6667$

$u_2 = (9 \times 8 + 5) \bmod (12) = 5 \quad R_2 = .4167$

$u_3 = (9 \times 5 + 5) \bmod (12) = 2 \quad R_3 = .1667$

1.4.7 Generating "deviates" from non-Uniform prob. distributions

1.4.7.1 Inverse Transformation Method

Explanation of the method:

1. Given the density function $f(x)$ for x the distribution (cumulative distribution) function $F(x)$ is $\int_{-\infty}^{x} f(t)dt$

2. Generate a uniform random number r *in [0,1], that is* $r \sim U(0, 1)$

3. Set $F(x) = r$ and solve for x.

Example:

$f(x) = 3x^2 \quad 0 \le x \le 1$

$= 0 \quad$ otherwise

Hence,

$$F(x) = \int_{-\infty}^{x} 3t^2 dt = \int_{0}^{x} 3t^2 dt = x^3$$

Generate random number, r, from uniform distribution [0 , 1]

Set $F(x) = r$ and solve for x, that is $x^3 = r \Rightarrow x = F^{-1}(r) = r^{1/3}$

For instance,

r	0.07	0.80	0.86	0.82	0.40	0.25	0.67	0.56	0.49
x	0.40	0.93	0.95	0.94	0.74	0.63	0.88	0.82	0.79

Example: Exponential distribution (*method results in a quite simple formula*):

$f(x) = \lambda e^{-\lambda x} \quad x \ge 0 \quad (\lambda \ge 0)$

$= 0 \quad$ otherwise

Hence,

$F(x = 0 \quad x < 0$

$= 1 - e^{-\lambda x} \quad x \ge 0$

Generate random number, r, from uniform distribution [0 , 1]

Set $F(x) = r$ and solve for x, that is $1 - e^{-\lambda x} = r \Rightarrow -\lambda x = \ln(1 - r)$

$\Rightarrow x = -1/\lambda \ln(1 - r) \Rightarrow x = -1/\lambda \ln(r)$ (*as r is in [0, 1]*)

For instance, for $\lambda = 2$,

r	0.33	0.88	0.62	0.60	0.53	0.79	0.77	0.34	0.53
x	0.55	0.07	0.24	0.25	0.32	0.12	0.13	0.54	0.32

1.4.7.2 Other Method

There are other methods – e.g. Acceptance-Rejection method but we will ***not*** go into these.

1.4.7.3 Specific Methods for the Normal Distribution

Convolution method (approximation) – uses Central Limit Theorem:

Given $r_1 r_2 \ldots\ldots r_n \sim U(0, 1)$ [*Reminder:* mean = 0.5, variance= *1/12*]

Then, $Z = \{\sum_{i=1}^{n} r_i - 0.5n\}/(n/12)^{1/2}$ is approximately N(0, 1)

When n = 12 *(often the value taken)* $Z = \sum_{i=1}^{n} r_i - 6$

Note: For $X \sim N(\mu, \sigma^2)$ simply apply the transformation $X = \mu + Z\sigma$

Box-Muller method: This is another (exact) method for generating Normal deviates.

1.4.7.4 Concluding note on sampling from probability distributions

We have just given a flavour of the topic so that much more can be read about it, as the need arises.

It is worth repeating that there are many library functions now available that can be made use of. Of course, one should make sure as far as possible that any functions used do indeed have the properties claimed. There are various **statistical tests** that can be applied to test whether a given set of numbers follow (to sufficient accuracy) a specified distribution but we will not go into these.

We have seen that other distributions can be derived from uniform numbers. In fact, sometimes one can apply a similar kind of reasoning to generate deviates from a required distribution given some "known" deviates. For example, the Rice (or Rician) distribution is the probability distribution of the absolute value of a circular bivariate normal random variable with (possibly) non-zero mean. Thus,

$R \sim \text{Rice}(\nu, \sigma)$ has a Rice distribution if $R = \sqrt{X^2 + Y^2}$ where $X \sim N(\nu\cos\theta, \sigma^2)$ and $Y \sim N(\nu\sin\theta, \sigma^2)$ are statistically independent normal random variables and θ is any real number. One way of generating a Rician deviate R is to generate normal deviates X and Y and then just use the above formula for R.

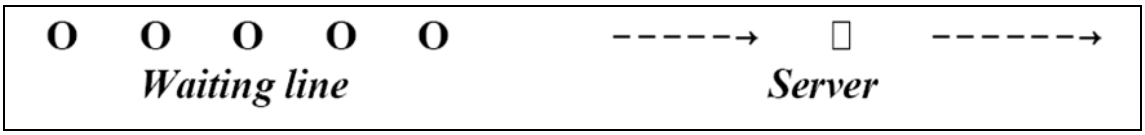
1.4.8 Simulation models, implementation aspects, etc

1.4.8.1 Two types of time-advanced simulation models

(A) Next Event Time-advanced Simulation Models- State variables moved by events (e.g. arrival, departure). Uneven time gaps between events.

(B) Fixed Increment Time-advanced Simulation Models – Clock advanced in fixed time increments (less efficient).

Example of (B): Single server queueing system (see earlier examples):



Here, there are two events: arrival and departure (though one might consider inclusion of an "entering service" event also, as discussed previously).

Notation/Key parameters:

TM = clock time	WL = length of waiting line
AT = next scheduled arrival time	MX = length of time of simulation run
DT = next scheduled departure time	IT = inter-arrival time
SS = status of server (1 = busy, 0 = idle)	ST = service time

N.B. *IT* and *ST* must be independent of each other.

1.4.8.2 To program a simulation:

- Any general-purpose programming language – C++, Java, etc.
- Languages with strong mathematical/statistical/visualisation capabilities such as Matlab, R and Mathematica
- Special purpose simulation languages of which many have been developed over the years. Two categories can be distinguished:
  - Event scheduling
  - Process oriented
- Event scheduling – SIMSCRIPT, SLAM, SIMAN etc (some just historical now)
- Process oriented – use *blocks* to represent processes (e.g. source, queue, facility), linked together to describe *transactions (entities)*. 'Black box' approach – easy to use. Examples are GPSS, SIMNET.
- Spreadsheets particularly Excel

1.4.8.3 Other reading

There is a good deal of interesting material available on simulation and software development (especially object-oriented). See, for example, (Steadman, 2003) the text of which is available on the internet. This reference contains a useful discussion and definition of various types of simulation as well as, from the software design perspective, various class diagrams etc.

1.5 Simulation examples

1.5.1 A Traffic Simulation [based on (Burkardt, 2009) in LOOP]

See (Burkardt, 2009), pages 2 to 10, for background and build-up to a "A Traffic Simulation", for which he presents an outline Matlab model:

```
for cycle = 1 : cycle_num
        r = rand ( cycle_length, 1 );
        cars_new = sum ( r < p );
        cars = cars + cars_new;
        cars_in = cars_in + cars_new;
        if ( light == 'g' )
                [ cars, cars_out, light, green_timer ] = go ( ...
                green_cycles, cars, cars_out, light, green_timer );
        else
                [ cars, light, red_timer ] = stop ( red_cycles, ...
                cars, light, red_timer );
        end
        car_wait_cycles = car_wait_cycles + cars;
end
```

where

cycle_length	= time unit (10 seconds)
cycle	= counter of cycles
cycle_num	= number of cycles
cars	= number of cars waiting at the intersection (initially zero)
p	= probability that a new car will arrive at the light every second
cars_new	= number of car that arrive in a given cycle
red_cycles	= number of cycles that red light lasts for
green_cycles	= number of cycles that green light lasts for
red_timer	= counter to keep track of how long red light has been on
green_timer	= counter to keep track of how long green light has been on

Note: Results presented by (Burkardt, 2009), pages 12-18, are suggestive but details of input and output parameters are insufficient to examine in detail.

Based on, but extending the (Burkardt, 2009) model, we present in the following a software implementation of a single-lane, one direction only, traffic lights model.

It is composed of two functions[2], a "driver" function:

[2] In Matlab returned parameters (if any) are contained in square brackets between the word function and the function name. The input parameters are in parentheses after the function name.

function **RunTrafficSimulation**(cycle_num, cycle_length, p, cars_initial, initial_light, red_cycles, green_cycles, green_rate, prt)

and a function to carry out the actual simulation:

function [carQueue, cars_in, cars_out, lambda, MeanQueueLength, MeanQueueTime] = **SingleOneWayLane**(cycle_num, cycle_length, p, cars_initial, initial_light, red_cycles, green_cycles, green_rate,prt)

We give the details of these functions below, followed by some indicative "simulation runs".

```
function RunTrafficSimulation( cycle_num, cycle_length, p, cars_initial, initial_light, red_cycles, green_cycles, green_rate, prt)
fprintf('INPUT ARGUMENTS:\n')
fprintf('%5d = Total number of cycles simulated \n',cycle_num)
fprintf('%5d = Length of a cycle in time units (typically seconds) \n',cycle_length)
fprintf('%5.2f = prob. that a new car will arrive at light every time unit (second) \n',p)
fprintf('%5s = Initial light colour (assume just turned this colour) \n',initial_light)
fprintf('%5d = total number of cycles per green light episode \n',green_cycles)
fprintf('%5d = total number of cycles per red light episode \n',red_cycles)
fprintf('%5d = No. of cars leaving per green cycle (maximum) \n', green_rate)
fprintf('%5d = Output controller = 1 for detail \n', prt)
[carQueue,cars_in, cars_out, lambda,MeanQueueLength,MeanQueueTime] = SingleOneWayLane( cycle_num,cycle_length,p,cars_initial, initial_light, red_cycles, green_cycles, green_rate, prt);
fprintf('\n \n OUPUT ARGUMENTS:\n\n')
if prt==1
   fprintf('List of cars waiting at end of each cycle \n')
   carQueue
end
fprintf('\n *** Plot generated of no. waiting at end of each cycle *** \n\n')
plot(carQueue)
fprintf('%5d = total number of cars that arrived at lights (includes initial) \n', cars_in)
fprintf('%5d = Total number of cars that left lights \n', cars_out)
fprintf('%5.2f = Throughput (out/in) \n',cars_out/cars_in)
fprintf('%5d = Mean no. of arrivals per cycle = arrival rate per cycle \n', lambda)
fprintf('%5.2f = Average length of queue over period \n', MeanQueueLength)
fprintf('%5.2f = Average queue waiting time (assuming ok to use Little) \n', MeanQueueTime)
```

```
function [carQueue, cars_in, cars_out, lambda, MeanQueueLength, MeanQueueTime] = SingleOneWayLane( cycle_num, cycle_length, p, cars_initial, initial_light, red_cycles, green_cycles, green_rate,prt)
% INPUT ARGUMENTS:
```

```
% cycle_num = Total number of cycles simulated
% cycle_length = Length of a cycle in time units (typically seconds)
% p = prob. that a new car will arrive at light every time unit (second)
% initial_light = Initial light colour (assume just turned this colour)
% green_cycles = total number of cycles per green light episode
% reds_cycles = total number of cycles per red light episode
% green_rate = No. of cars leaving per green cycle (maximum)
% prt = output flag, if 1 then detailed cycle output produced
%
% OUTPUT PARAMETERS:
% carQueue = list of cars waiting at end of each cycle
% cars_in = total number of cars that arrived at lights (includes initial)
% cars_out = Total number of cars that left lights
% lambda = Mean no. of arrivals per cycle = arrival rate per cycle.
% MeanQueueLength = Average length of queue over period
% MeanQueueTime = Average queue waiting time (assuming ok to use Little)
cars=cars_initial;
carQueue=[];
cars_in=cars_initial;
cars_out=0;
lambda=p*cycle_length;
light=initial_light;
if light == 'g'
   green_timer=0;
elseif light == 'r'
   red_timer=0;
end
if prt == 1
   fprintf('      Cycle Light Cars_Waiting New_Batch\n')
end
for cycle = 1 : cycle_num
   r = rand ( cycle_length, 1 );
   cars_new = sum ( r < p );
   cars = cars + cars_new;
```

```
    if prt == 1
        fprintf('START %5d %5s %5d     %5d     (incl. new arrival batch)\n', cycle, light,
cars, cars_new)
    end
    cars_in = cars_in + cars_new;
if  light == 'g'
        if cars <= green_rate
            cars_out=cars_out+cars;
            cars=0;
        else
            cars_out=cars_out+green_rate;
            cars=cars-green_rate;
        end
        green_timer=green_timer+1;
        if green_timer >= green_cycles
            red_timer=0;
            light='r';
        end
elseif light == 'r'
        red_timer=red_timer+1;
        if red_timer >= red_cycles
            green_timer=0;
            light='g';
        end
 end
carQueue=[carQueue cars];
    if prt == 1
        fprintf('END      %5d  %5s  %5d                    (ready  for  next  cycle)\n
\n',cycle,light,cars)
    end
end
MeanQueueLength=sum(carQueue)/cycle_num;
MeanQueueTime=MeanQueueLength/lambda;   % Using Little relation
```

end

Sample results (1) [Short simulation duration]

INPUT ARGUMENTS:

10 = Total number of cycles simulated

10 = Length of a cycle in time units (typically seconds)

0.40 = prob. that a new car will arrive at light every time unit (second)

r = Initial light colour (assume just turned this colour)

4 = total number of cycles per green light episode

4 = total number of cycles per red light episode

10 = No. of cars leaving per green cycle (maximum)

0 = Output controller = 1 for detail

OUPUT ARGUMENTS:

*** Plot generated of no. waiting at end of each cycle *** [***see below***]

44 = total number of cars that arrived at lights (includes initial)

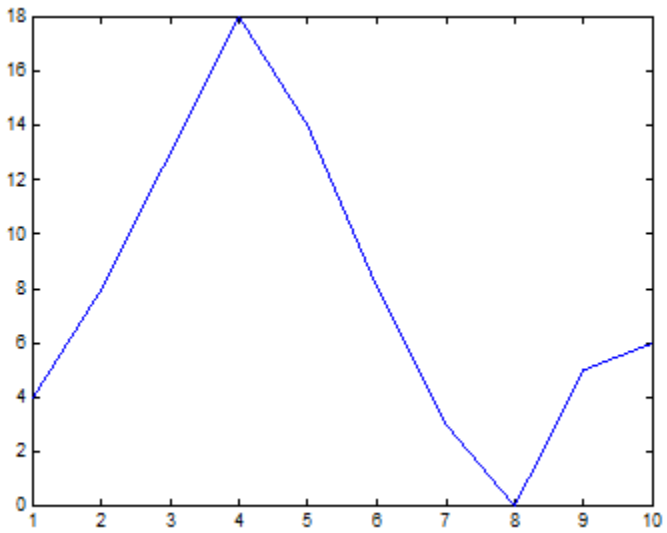
38 = Total number of cars that left lights

0.86 = Throughput (out/in)

4 = Mean no. of arrivals per cycle = arrival rate per cycle

7.90 = Average length of queue over period

1.98 = Average queue waiting time (assuming ok to use Little)



Sample results (2) [Long(er) simulation duration]

INPUT ARGUMENTS:

180 = Total number of cycles simulated

10 = Length of a cycle in time units (typically seconds)

0.40 = prob. that a new car will arrive at light every time unit (second)

r = Initial light colour (assume just turned this colour)

4 = total number of cycles per green light episode

4 = total number of cycles per red light episode

10 = No. of cars leaving per green cycle (maximum)

0 = Output controller = 1 for detail

OUPUT ARGUMENTS:

*** Plot generated of no. waiting at end of each cycle *** [***see below***]

713 = total number of cars that arrived at lights (includes initial)

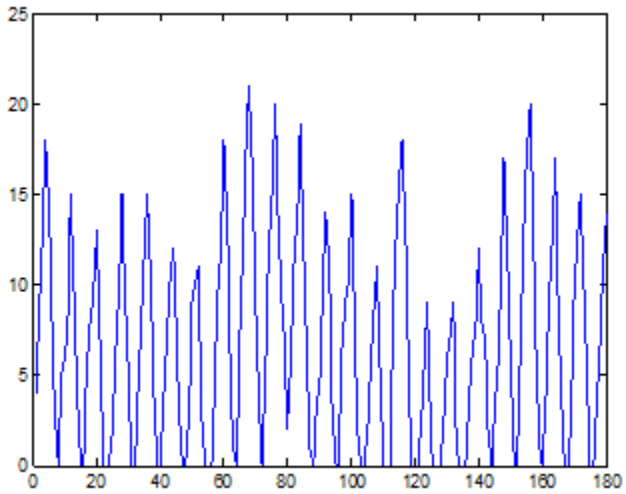
699 = Total number of cars that left lights

0.98 = Throughput (out/in)

4 = Mean no. of arrivals per cycle = arrival rate per cycle

6.70 = Average length of queue over period

1.68 = Average queue waiting time (assuming ok to use Little)



Sample results (3) [Same as 2 but p=0.5]

INPUT ARGUMENTS:

180 = Total number of cycles simulated

10 = Length of a cycle in time units (typically seconds)

0.50 = prob. that a new car will arrive at light every time unit (second)

r = Initial light colour (assume just turned this colour)

4 = total number of cycles per green light episode

4 = total number of cycles per red light episode

10 = No. of cars leaving per green cycle (maximum)

0 = Output controller = 1 for detail

OUPUT ARGUMENTS:

*** Plot generated of no. waiting at end of each cycle *** [***see below***]

867 = total number of cars that arrived at lights (includes initial)

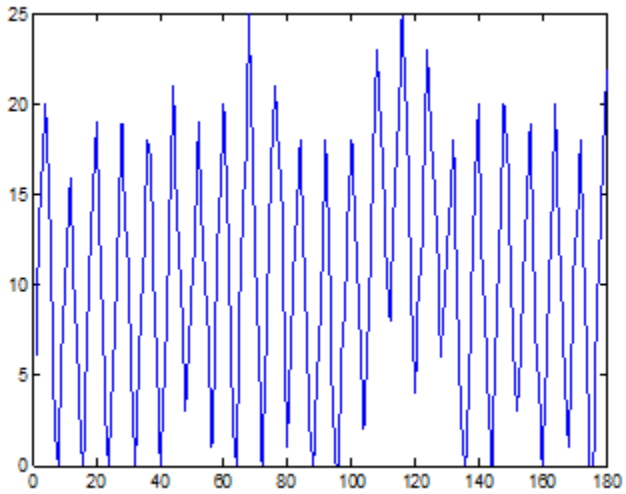
845 = Total number of cars that left lights

0.97 = Throughput (out/in)

5 = Mean no. of arrivals per cycle = arrival rate per cycle

10.43 = Average length of queue over period

2.09 = Average queue waiting time (assuming ok to use Little)



Sample results (4) [Same as 3 but 8 leave per green cy.]

INPUT ARGUMENTS:

180 = Total number of cycles simulated

10 = Length of a cycle in time units (typically seconds)

0.50 = prob. that a new car will arrive at light every time unit (second)

r = Initial light colour (assume just turned this colour)

4 = total number of cycles per green light episode

4 = total number of cycles per red light episode

8 = No. of cars leaving per green cycle (maximum)

0 = Output controller = 1 for detail

OUPUT ARGUMENTS:

*** Plot generated of no. waiting at end of each cycle *** [***see below***]

897 = total number of cars that arrived at lights (includes initial)

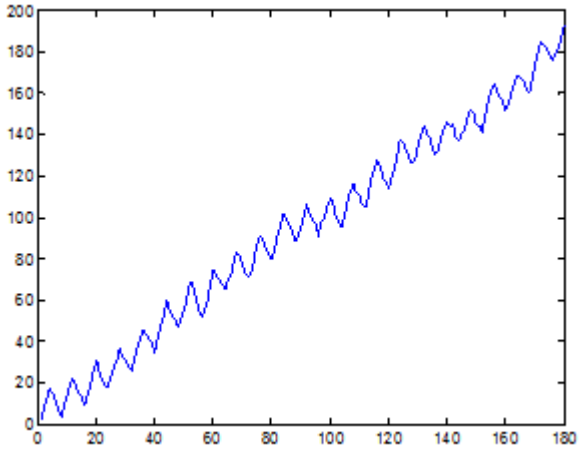
704 = Total number of cars that left lights

0.78 = Throughput (out/in)

5 = Mean no. of arrivals per cycle = arrival rate per cycle

93.98 = Average length of queue over period

18.80 = Average queue waiting time (assuming ok to use Little



Sample results (5) [Same as 4 but green on for 6 cycles]

INPUT ARGUMENTS:

180 = Total number of cycles simulated

10 = Length of a cycle in time units (typically seconds)

0.50 = prob. that a new car will arrive at light every time unit (second)

r = Initial light colour (assume just turned this colour)

6 = total number of cycles per green light episode

4 = total number of cycles per red light episode

8 = No. of cars leaving per green cycle (maximum)

0 = Output controller = 1 for detail

OUPUT ARGUMENTS:

*** Plot generated of no. waiting at end of each cycle *** [***see below***]

914 = total number of cars that arrived at lights (includes initial)

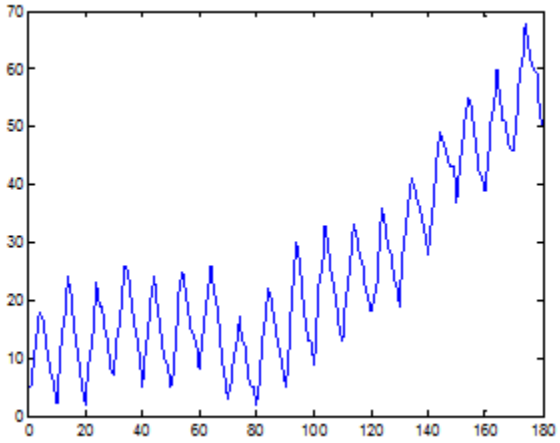
864 = Total number of cars that left lights

0.95 = Throughput (out/in)

5 = Mean no. of arrivals per cycle = arrival rate per cycle

24.79 = Average length of queue over period

4.96 = Average queue waiting time (assuming ok to use Little



Proviso:

(1) Check that above simulation is implemented correctly!

(2) Also, does it meet the criteria, in words of (Burkardt, 2009):

"*Computer simulation is intended to provide a simple model of a complex process. The model is made up of simplifications, arbitrary choices, and rough measurements. We hope that there is still enough relationship to the real world that we can study the model and get an idea of how the original process works, predict outputs for given inputs, and how we can improve it by changing some of the parameters.*"

1.5.2 Outline of other applications, from (Burkardt, 2009)

1.5.2.1 Duels

Another simulation presented by (Burkardt, 2009) (page 19 etc) is that of

- A basic example from Game Theory: Duels

or, more generally,

- Coding a Three-Way Duel

As before, the simulation is implemented in Matlab. Below we present

- function **duel_once**(…) to simulate one instance of a two-person duel
- A script (*essentially a main program in this context*) to call **duel_once**(..) multiple times in order to estimate average statistics etc.

```
function [survivor,turn_num]=duel_once ( p )
turn_num = 0;
while ( 1 )
   turn_num = turn_num + 1; % Player 1 fires.
```

```
    r = rand ( );
    if ( r <= p(1) )
        survivor = 1;
        break
    end
    turn_num = turn_num + 1; % Player 2 fires.
    r = rand ( );
    if ( r <= p(2) )
        survivor = 2;
        break
    end
end
end
```

Following is the calling script:

```
s = zeros(2,1);
turn_average = 0;
p=[4/6,5/6];
duel_num=100;
for duel = 1 : duel_num
        [ survivor, turn_num ] = duel_once ( p );
        s(survivor) = s(survivor) + 1;
        turn_average = turn_average + turn_num;
end
s = s / duel_num
turn_average = turn_average / duel_num
```

This resulted in the following output:

```
>> Duels2
s =
    0.6700          [Proportion of wins for Player 1]
    0.3300          [Proportion of wins for Player 2]
turn_average =
    1.4300      [Average number of turns taken]
```

1.5.2.2 Other examples (refer to (Burkardt, 2009))

GAMBLER'S RUIN: This is another fairly easy simulation, discussed and implemented by (Burkardt, 2009).

(Burkardt, 2009) also discusses and presents some results for simulations of

- **Brownian Motion**
- **Random Walks**
- **A Model of Epidemics**

These are just the beginning of study of whole classes of related problems. Other approaches besides simulation are also used, including Markov Chains for epidemic modelling.

Bibliography

Boccara, N., 2004. *Modeling Complex Systems.* s.l.:Springer.
Burkardt, J., 2009. *The Monte Carlo Method: SIMULATION.* [Online]
Available at: http://people.sc.fsu.edu/~jburkardt/presentations/
[Accessed 2015].
Cox, D. R. & Smith, W. L., 1961. *Queues.* s.l.:Chapman and Hall.
Ross, S. M., 2002. *Simulation.* 3 ed. s.l.:Academic Press.
Steadman, P., 2003. *Overview of Modeling and Simulation and Object,* s.l.: s.n.