PROBABILITY AND SIMULATION WITH MAPLE

ADI BEN-ISRAEL

Presented at the 12th Annual International Conference on Technology in Collegiate Mathematics, San Francisco, November 4-7, 1999

ABSTRACT. Several well-known examples in Probability and Operations Research are analyzed using simulation with MAPLE.

1. INTRODUCTION

1.1. **Notation and terminology.** *Random variables* (abbreviated **RV**'s) are denoted by capital letters, e.g. X, and their *values* (also *observations*) are denoted by low-case letters. Probabilities associated with a RV X are denoted by $\text{Prob}\{X \in (-2,5]\}$, $\text{Prob}\{X \geq 7\}$, etc.
Example: X is the value returned by a fair die,

$$\text{Prob}\{X = x\} = \begin{cases} \frac{1}{6} & , x \in \{1,2,3,4,5,6\} \\ 0 & , \text{otherwise} \end{cases}$$

The *(cumulative) distribution function* (abbreviated **DF**) of a RV X is the function

$$F(x) = \text{Prob}\{X \leq x\} \tag{1}$$

If X is a continuous RV it has a *density function*

$$f(x) = \lim_{dx \to 0} \frac{\text{Prob}\{x \leq X \leq x + dx\}}{dx} \text{ , such that } F(x) = \int_{-\infty}^{x} f(t)\, dt \,. \tag{2}$$

In the discrete case the integral is replaced by a sum

$$F(x) = \sum_{t=-\infty}^{x} f(t) \text{ , where } f(t) = \text{Prob}\{X = t\} \tag{3}$$

In either case, the DF is a nondecreasing function, rising from $F(-\infty) = 0$ to $F(\infty) = 1$.
The *expected value* $\text{E}\{X\}$ and *variance* $\text{Var}\{X\}$ of a RV X are computed by

$$\begin{aligned} \text{E}\{X\} &= \int_{-\infty}^{\infty} t\, f(t)\, dt \text{ , continuous case ,} \\ &= \sum_{-\infty}^{\infty} t\, f(t) \text{ , discrete case ,} \\ \text{Var}\{X\} &= E\,(X - E\{X\})^2 \,. \end{aligned}$$

The *standard deviation* of X is $\sqrt{\text{Var}\{X\}}$.

The *support* of a RV is the largest set where the probability (density in the continuous case) is positive.

A *random sample* is a set of independent observations $\{x_1, x_2, \cdots, x_N\}$. Independent RV's with the same DF are called *independent, identically distributed* (abbreviated **iid**).

Date: November 5, 1999.
Key words and phrases. Probability, simulation, Monte-Carlo, Maple, birthday problem, 3 doors puzzle, St Petersburg paradox, Buffon needle problem, newsboy problem, investment problem, queuing system.

1.2. **Simulation.** Given an interval $[a,b]$, a RV with the density function

$$f(x)=\begin{cases}\frac{1}{b-a} & ,\ x\in[a,b]\\ 0 & ,\ x\notin[a,b]\end{cases} \tag{4}$$

is said to be *uniformly distributed* (abbreviated **uniform**) on $[a,b]$.

Random numbers (abbreviated **RN**'s) are uniform on $[0,1]$ and iid.
If X is a RV with DF $F(x)$ and if $\{r_1,r_2,\cdots,r_N\}$ are RN's we can (artificially) generate a random sample $\{x_1,x_2,\cdots,r_N\}$ of X by solving

$$F(x)=r_i\ , \tag{5}$$

for $i=1,2,\cdots,N$. If the DF $F(x)$ has an inverse (this happens if X is continuous, $F(x)$ is monotone increasing in an interval I and the density $f(x)$ is zero outside I) then (5) returns a unique value x_i,

$$x_i=F^{-1}(r_i) \tag{6}$$

In the remaining cases, ambiguities in (5) are very unlikely (have zero probability) and can be resolved by the rule,

$$x_i \text{ is smallest } x \text{ such that } F(x)=r_i\ .$$

Simulation (also *Monte-Carlo simulation*) is a process of generating samples of a RV X by using RN's. It is one of the most important tools for analyzing and solving problems involving uncertainty, arising in many fields, from Nuclear Physics to Business. Often such problems are too complicated for precise analysis, and simulation is the only option. Consequently every programming language includes a *random number generator* (abbreviated **RNG**) giving *pseudo random numbers*, which are good approximations of RN's.

Outline of this paper:

§ 2: Maple functions used in the sequel, § 3: simulation using Maple,
§ 4: the birthday problem, § 5: the St Petersburg paradox,
§ 6: the 3 doors puzzle, § 7: the Buffon needle problem,
§ 8: the newsboy problem, § 9: an investment problem,
§ 10: queuing systems.

All new Maple functions in this paper have names beginning with capital letters, e.g. `DiscreteSimulations(f,x,L,U,n)`. Some Maple functions occur in singular/plural pairs, e.g. `Rand()` and `Rands(n)`, `Simulation(x,p)` and `Simulations(x,p,n)`, where the singular form returns a scalar, and the plural form returns an n–dimensional vector. All Maple programs in this paper can be downloaded from `ftp://rutcor.rutgers.edu/pub/bisrael/SIMULATION.mws`

2. Some Maple functions

The Maple functions collected in this section are used in the sequel. After starting Maple we load the following packages

```
> restart:with(linalg):with(stats):with(plots):
```

The Maple function `rand()` generates a random number with 12 digits. For example,

```
> rand();
```

427419669081

to get a random number in $[0,1]$ we define the function `Rand()`

```
> Rand:=proc()
> evalf((rand()*10^(-12)),5);
> end:
```

For example,

```
> Rand();
```

The function `Rands(n)` generates a vector with n RN's

```
> Rands:=proc(n)
> local x,k;
> x:=vector(n,[]);
> for k from 1 to n
> do x[k]:=Rand();od;
> evalm(x);end:
```

For example, try the following three commands

```
> Rands(20);
> convert(%,list);
> sort(%);
```

Why was it necessary to convert to a `list` before sorting?
The function `Cum(x)` computes the vector of partial sums of elements of x

```
> Cum:=proc(x)
> local k,n,v;
> n:=vectdim(x);v:=array(1..n);v[1]:=x[1];
> for k from 2 to n
> do v[k]:=v[k-1]+x[k] od;
> evalm(v);
> end:
```

For example,

```
> Cum([1,2,3,4,5,6,7,8,9,10]);
```

[1, 3, 6, 10, 15, 21, 28, 36, 45, 55]

The function `Lookup(x,a)` finds the last occurrence of `x[k]` $\leq$ `a`

```
> Lookup:=proc(x,a)
> local n,k,ka;
> n:=vectdim(x);ka:=0;
> for k from 1 to n do
> if a>=x[k] then ka:=k fi;od;
> RETURN(ka);end:
```

Examples:

```
> Lookup([1,2,3,4,5,6,5,8],5);
```

7

```
> Lookup([1,2,3,4,5,6,7,8],4.99);
```

4

The function `Statistics(x)` prints some statistics of the elements of a vector `x`,

```
> Statistics:=proc(x)
> local n,k,s,ss,w,fl;
> n:=vectdim(x);
> s:=sum('x[k]','k'=1..n)/n;
> ss:=sum('x[k]^2','k'=1..n)/n;
> printf("%A %G\n",'Average',evalf(s));
> printf("%A %G\n",'Std_dev',evalf(sqrt(ss-s^2)));
> w:=convert(x,list):w:=sort(w);fl:=floor(n/2);
> printf("%A %G\n",'Median',w[fl]);printf("%a %g\n",'Maximum',w[n]);
> printf("%A %G\n",'Minimum',w[1]);
> end:
```

For example, the statistics of a random sample of 100 RN's,

```
> Statistics(Rands(100));
```

```
Average .493814
```

```
Std_dev .299507
Median .52543
Maximum .99299
Minimum .005863
```

The sample mean is close to the expected value of 0.5, and the sample standard deviation is close to the theoretical value:

```
> sqrt(int((x-0.5)^2,x=0..1));
```

.2886751346

The function `Repetitions(v)` gives the number of repetitions in a vector `v`

```
> Repetitions:=proc(v)
> local n,k,r,w;
> n:=vectdim(v);r:=0;
> w:=convert(v,list);w:=sort(w);
> for k from 1 to n-1 do
> if w[k+1]=w[k] then r:=r+1;fi;
> eval(r);od;
> end:
```

Examples:

```
> Repetitions([1,-3,2,3,-3,4,5,4]);
```

2

```
> Repetitions([dog,cat,dog,cat,dog]);
```

3

3. Simulation with Maple

The Maple functions in this section generate random samples from discrete or continuous RV's, by solving (5).

3.1. **Discrete RV's.** For simplicity we assume discrete RV's to have only integer values. The functions `Simulation(x,p)` (`Simulations(x,p,n)`) generate one observation (resp. a vector of n observations) of a discrete RV with values `x` and corresponding probabilities `p` (the vectors `x` and `p` must have the same dimension)

```
> Simulation:=proc(x,p)
> local k,cp;
> cp:=Cum(p);
> k:=Lookup(cp,Rand());
> RETURN(x[k+1]);
> end:
```

```
> Simulations:=proc(x,p,n)
> local k,v;
> v:=vector(n,[]);
> for k from 1 to n
> do v[k]:=Simulation(x,p);od;
> evalm(v);end:
```

Examples: A fair coin

```
> Simulation([H,T],[0.5,0.5]);
```

H

```
> Simulations([H,T],[0.5,0.5],20);
```

$[T, H, H, T, T, T, T, T, T, T, H, H, T, H, T, H, T, T, T, T]$

and a fair die

```
> Simulation([1,2,3,4,5,6],[1/6,1/6,1/6,1/6,1/6,1/6]);
```

1

Try

```
> v:=Simulations([0,1],[0.5,0.5],100);
> Statistics(v);
> v:=Simulations([1,2,3,4,5,6],[1/6,1/6,1/6,1/6,1/6,1/6],100):
> Statistics(v);
```

The above functions `Simulation(x,p)` and `Simulations(x,p,n)` work for RV's with finitely many values (the vector `x` is finitely dimensional). For discrete RV's with infinitely many values, the functions `DiscreteSimulation(f,x,L,U)` (`DiscreteSimulations(f,x,L,U,n)`) generate one observation (resp. a vector of n observations) from a RV X with values x and corresponding probabilities $f(x)$. `L` is the smallest possible value of X, and the values of X are truncated at `U`, a sufficiently large integer, possibly, `U`$= \infty$.

```
> DiscreteSimulation:=proc(f,x,L,U)
> local t,F,m;
> randomize(seed);t:=Rand();F:=0;
> for m from L to U do F:=evalf(F+subs(x=m,f));
> if F>t then RETURN(m);BREAK;fi;
> od;
> end:

> DiscreteSimulations:=proc(f,x,L,U,n)
> local v,k;
> v:=vector(n,[]);
> for k from 1 to n do
> v[k]:=DiscreteSimulation(f,x,L,U);
> od:
> evalm(v);convert(v,list);
> end:
```

Example: the Poisson distribution

$$f(x) = \frac{\lambda^x\, e^{-\lambda}}{x!}\ ,\ x = 0, 1, \cdots \tag{7}$$

has expected value λ and variance λ. We simulate 20 observations with $\lambda = 2.3$,

```
> DiscreteSimulations((2.3^x)*exp(-2.3)/x!,x,0,infinity,20);
```

[2, 3, 2, 3, 3, 2, 1, 2, 5, 2, 3, 3, 3, 2, 3, 5, 2, 1, 3, 2]

Try

```
> DiscreteSimulations((2.3^x)*exp(-2.3)/x!,x,0,infinity,100):
> Statistics(%);
```

Example: for $0 < p < 1$, the geometric distribution

$$f(x) = (1-p)^{x-1}\, p\ ,\ x = 1, 2, \cdots \tag{8}$$

has expected value $1/p$. We simulate 20 observations for $p = 0.5$,

```
> DiscreteSimulations((0.5)^(x-1)*0.5,x,1,100,20);
```

[1, 2, 4, 1, 1, 2, 2, 3, 2, 3, 1, 4, 1, 1, 1, 2, 1, 6, 3, 1]

Try

```
> DiscreteSimulations((0.5)^(x-1)*0.5,x,1,100,100);
> Statistics(%);
```

3.2. **Continuous RV's.** The functions `ContSimulation(f,x,L)` (`ContSimulations(f,x,L,n)`) generates one observation (resp. a vector of n observations) of a random variable with probability density function f. The user must specify the leftmost point L of the support of X, possibly, $L = -\infty$.

```
> ContSimulation:=proc(f,x,L)
> local t,F;
> F:=int(f,x=L..t);
> randomize(seed);solve(F=Rand(),t);
> end:
```

```
> ContSimulations:=proc(f,x,L,n)
> local k,v;
> v:=vector(n,[]);
> for k from 1 to n
> do v[k]:=ContSimulation(f,x,L);od:
> evalm(v);convert(v,list);
> end:
```

Example: the uniform distribution on $[0, 1]$,

```
> f:=1:ContSimulation(f,x,0);
```

.1508600000

```
> f:=1:ContSimulations(f,x,0,5);
```

[.8026400000, .5785700000, .1714600000, .004109700000, .6926100000]

Example: the standard normal distribution,

$$f(x) = \frac{1}{\sqrt{2\,\pi}}\, e^{-x^2/2}\ , -\infty < x < \infty \tag{9}$$

```
> f:=exp(-x^2/2)/sqrt(2*Pi):ContSimulation(f,x,-infinity);
```

-1.193585880

```
> f:=exp(-x^2/2)/sqrt(2*Pi):L:=-infinity:ContSimulations(f,x,L,5);
```

$[-.8772700053, .3136063975, 1.483129529, -.4936972609, -1.212764164]$

Try

```
> f:=1:v:=ContSimulations(f,x,0,100):
> Statistics(%);
> f:=exp(-x^2/2)/sqrt(2*Pi):ContSimulations(f,x,-infinity,100):
> Statistics(%);
```

4. The birthday problem

We assume that birthdays are uniform throughout the year, so that the probability that a person chosen at random will have a birthday on any date is $1/365$.

Given a random sample of n people, we denote by $p(n)$ the probability that at least two have a common birthday. Clearly $p(n)$ increases with n, from $p(1) = 0$ to $p(366) = 1$. Find the smallest n such that $p(n) \geq 1/2$. See also [2, p. 29].

The answer, $n = 23$, comes as a surprise to students who typically guess a much larger number.

The function `ProbCommonBirthday(n,N)` computes the probability of at least one common birthday in a group of n, if a year has N days. This is computed using the probability of the complementary event, i.e. no common birthday

```
> ProbCommonBirthday:=proc(n,N)
> local k;
> evalf(1-product((N-k+1)/N,k=1..n));
> end:
```

As claimed, for $n = 23$ the probability goes above $1/2$,

```
> ProbCommonBirthday(23,365);
```

.5072972343

In a planet with a longer year of 10,000 days, the answer to the birthday problem is $n = 119$,

```
> v:=vector(150,[ ]):for k from 1 to 150 do v[k]:=ProbCommonBirthday(k,10000);od:
> 1+Lookup(v,0.5);
```

119

We now use simulation to illustrate the above results. The function `Birthdays(n,N)` generates n random dates from a year with N days (on earth $N = 365$)

```
> Birthdays:=proc(n,N)
> local v,k;
> v:=vector(n,[]);
> for k from 1 to n do
> v[k]:=floor(Rand()*N);od;
> evalm(v);end:
```

For example,

```
> Birthdays(10,365);
```

[343, 264, 240, 120, 199, 252, 291, 94, 298, 225]

```
> convert(%,list):sort(%);
```

[94, 120, 199, 225, 240, 252, 264, 291, 298, 343]

Now simulate 23 random birthdays,

```
> Birthdays(23,365);
```

[356, 167, 306, 357, 270, 141, 264, 257, 289, 176, 318, 253, 172, 22, 255, 216, 28, 162, 53, 214, 97, 35, 4]

```
> Repetitions(%);
```

0

i.e. no common birthday in this sample. With many replications of this simulation, we get 0 repetitions about 1/2 of the time.

5. The St Petersburg paradox

Two players, **A** and **B**. A fair coin is tossed until `H` first appears. If this happens on trial number n then **B** pays **A** the amount 2^{n-1}, and the game stops, see [2, p. 199]. What is the *value* of the game for **A**? In other words, if **B** were to sell tickets for this game, how much should **A** be willing to pay for a ticket? A reasonable answer is the *expected value* of **A**'s gain.

The probability of first `H` in trial n is $(1/2)^n$, therefore the expected value of **A**'s income is infinite,

```
> evalf(sum('2^(k-1)*(1/2)^k','k'=1..infinity));
```

∞

hence the paradox (if you ask how much people would be willing to pay, the typical answers are below 20).

When simulating the St Petersburg game we must truncate it, i.e. put a limit n on the number of trials (if `H` has not appeared before trial n, then **B** pays **A** the amount 2^{n-1} and the game ends). The function `Petersburg(n)` simulates **A**'s income in such a truncated game.

```
> Petersburg:=proc(n)
> local v,k,x,p,temp;
> x:=[H,T]:p:=[1/2,1/2]:v:=array(1..n):temp:=0;
> for k from 1 to n do
> v[k]:=Simulation(x,p);
> if v[k]=H then break;fi;od;
> eval(2^(k-1));
> end:
```

Example: the St Petersburg game with at most 20 trials,

```
> Petersburg(20);
```

4

Now we simulate 1000 games, each limited to at most 20 trials, and compute statistics

```
> v:=array(1..1000):for k from 1 to 1000 do v[k]:=Petersburg(20);od:
> Statistics(v);

Average 13.723
Std_dev 120.329308
Median 2
Maximum 2048
Minimum 1
```

showing that the upper bound $n = 20$ was not exceeded in this simulation (the longest game lasted 12 trials, giving a return of $2^{11} = 2048$).

Note: In one of the many attempts to explain this paradox, Daniel Bernoulli (1738) suggested that *wealth* w is actually worth $C\log(w)$ (*logarithmic utility*), for some $C > 0$. Accordingly, the value of the game for **A** is his *expected utility*

```
> evalf(sum('C*log(2^(k-1))*(1/2)^k','k'=1..infinity));
```

$$.6931471806\,C$$

6. The three doors puzzle

We owe this gem to Marilyn vos Savant, [5].

There are three closed doors. Behind one door (unknown to you) is a prize. The two other doors offer nothing. You want to pick the door with the prize. After making your choice, but before your door is opened, one of the other doors is opened to reveal that it is empty. At this point you are given the option of keeping the door you have selected, which is then opened, or switching to the other unopened door, which is then opened.

Should you stay with your original choice, or should you switch? Intuition tells us that there is no reason to switch (after all, the door that was opened did not contain the prize), however analysis shows that

- if you keep the original selection, your probability of winning is 1/3, and
- if you switch doors, the probability of winning is 2/3,

i.e. the winning chances are doubled by switching! See [4] for detailed analysis.

The function `ThreeDoors(n)` simulates n plays of this game, under the policy of non-switching, and computes the frequency of wins, which for n sufficiently large should be close to the theoretical value of 1/3.

```
> ThreeDoors:=proc(n)
> local k,p,q,d,w;w:=0; #w = number of wins in n trials
> for k from 1 to n do
> p:=Simulation([1,2,3],[1/3,1/3,1/3]);#door with prize
> q:=Simulation([1,2,3],[1/3,1/3,1/3]);#door chosen by cotestant
> d:=p+Simulation([1,2],[1/2,1/2]) mod 3;#empty door openned
> if p=q then w:=w+1 fi;od; #contestant stays with original choice
> printf("%A %G\n",'Frequency_of_wins',evalf(w/n));
> end:
```

For example,

```
> ThreeDoors(100);

Frequency_of_wins .34
```

To simulate n plays under the policy of switching, change the line

```
> if p=q then w:=w+1 fi;od; #contestant stays with original choice
```

to

```
> if p<>q then w:=w+1 fi;od; #contestant switches
```

For a generalization to more than 3 doors see the webpage [1].

7. The Buffon needle

A needle of length L is dropped into a plane marked by parallel lines, D apart. Assume that the ratio L/D is sufficiently small, say $L/D < \pi/2$. Then

$$\text{Prob}\{\text{the needle will cross any line}\} = \frac{2L}{\pi D}\,. \tag{10}$$

This result is often stated for $L = D = 1$, see [3, p. 61].

The function `Buffon(L,D,n)` simulates n needles, and records the frequency of line crossings, as well as the theoretical result (10).

```
> Buffon:=proc(L,D,n)
> local k,c,x;
> c:=0;
> for k from 1 to n do
> x:=evalf(D*Rand()+L*cos(2*Pi*Rand()));
> if (x>D or x<0) then c:=c+1 fi;
> od;
> lprint('Frequency_of_crossings');
> lprint('Simulated_value',evalf(c/n));
> lprint('Theoretical_value',evalf(2*L/(Pi*D)));
> end:
```

Example: 100 needles with $D = L = 1$,

```
> Buffon(1,1,100);

Frequency_of_crossings
Simulated_value   .5600000000
Theoretical_value   .6366197722
```

We next illustrate the agreement between the theoretical result (10) and the simulated frequencies of line crossings. Since the result depends on the ratio L/D we can fix $D = 1$, and vary only L. The function `Buffons(n)` displays the theoretical probabilities (straight line) and the simulated frequencies of `Buffon(L,1,n)` for 12 needle lengths `L=0.1(0.1)1.2`, see Figure 1.

```
> Buffons:=proc(n)
> local m,L,k,c,x,V,W;
> V:=array(1..24);W:=array(1..24);
> for m from 1 to 24 do
> L:=evalf(m/20);
> W[m]:=[L,evalf(2*L/Pi)];
> c:=0;
> for k from 1 to n do
> x:=evalf(Rand()+L*cos(2*Pi*Rand()));
> if (x>1 or x<0) then c:=c+1 fi;
> od;
> V[m]:=[L,evalf(c/n)];
> od;
> V:=pointplot(V,color=black,connect=true):
> W:=pointplot(W,color=red,connect=true,thickness=3):
> display(V,W);
> end:

> Buffons(100);
```

8. The newsboy problem

This problem concerns the optimal stock a company should carry if demand is uncertain. The usual story (see [6, p. 802]) is about a newsboy who must decide on the number of newspapers, x, to order every day. Daily demand is random, taking on values `d` with (respective) probabilities `p`, for example,

- `d = [6, 7, 8, 9, 10, 11, 12, 13, 14]`,
- `p = [0.05, 0.1, 0.15, 0.2, 0.25, 0.15, 0.05, 0.03, 0.02]`.

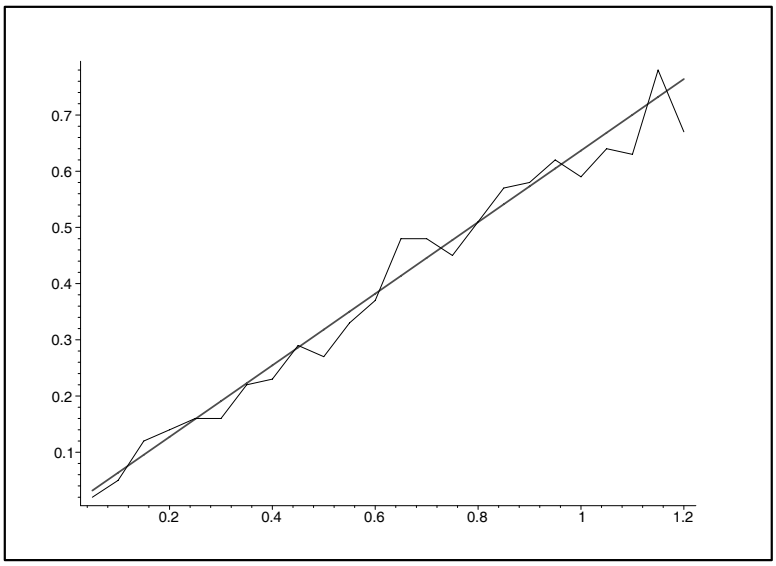

FIGURE 1. The x-axis is the ratio L/D, the y-axis is probability or frequency

A newspaper costs c\$, sells for r\$ and can be returned for s\$ if unsold. Typically $r > c > s$, where $r - c$ is the profit per sold newspaper, and $c - s$ is the loss if a newspaper in stock has not been sold. In some examples the salvage value s is negative, i.e. it is costly to dispose of unsold stock.

The function `DailyProfit(x,d,c,r,s)` computes the daily profit if the stock ordered is x, and the demand realized is d,

```
> DailyProfit:=proc(x,d,c,r,s)
> evalf(r*min(x,d)+s*max(x-d,0)-c*x);
> end:
```

Example:

```
> DailyProfit(9,6,.2,.4,.1);
```

.9

The function `DailyProfits(x,d,p,c,r,s,n)` simulates daily profits for a period of n days, for a stock `x`, demand = `d` (vector of values) with probabilities `p`

```
> DailyProfits:=proc(x,d,p,c,r,s,n)
> local k,temp,dapr;
> dapr:=array(1..n);
> for k from 1 to n do
> temp:=Simulation(d,p);
> dapr[k]:=DailyProfit(x,temp,c,r,s);
> evalm(dapr);od;
> end:
```

Example: for the following demand distribution

```
> d:=[6,7,8,9,10,11,12,13,14]:
> p:=[0.05,0.1,0.15,0.2,0.25,0.15,0.05,0.03, 0.02]:
```

$c = 0.2$, $r = 0.4$, $s = 0.1$ we simulate (but do not show) 100 daily profits under the policy $x = 10$,

```
> DailyProfits(10,d,p,0.2,0.4,0.1,100):
> Statistics(%);

Average 1.739
Std_dev .335826
Median 2.0
Maximum 2.0
```

```
Minimum .8
```

The average daily profit for a period of n days is similarly computed by the function

```
> AverageDailyProfit:=proc(x,d,p,c,r,s,n)
> local k,temp,cump;cump:=0;
> for k from 1 to n do
> temp:=DailyProfit(x,Simulation(d,p),c,r,s);
> cump:=cump+temp;od:
> evalf(cump/n);
> end:
```

Example: for the above demand distribution

```
> d:=[6,7,8,9,10,11,12,13,14]:
> p:=[0.05,0.1,0.15,0.2,0.25,0.15,0.05,0.03, 0.02]:
```

we simulate the average daily profits for 100 days under all possible policies x=6(1)14,

```
> for k from 6 to 14 do lprint('stock',k,'average_daily_profit',
> AverageDailyProfit(k,d,p,0.2,0.4,0.1,100));od:
```

```
stock   6   average_daily_profit   1.200000000
stock   7   average_daily_profit   1.391000000
stock   8   average_daily_profit   1.555000000
stock   9   average_daily_profit   1.626000000
stock   10   average_daily_profit   1.682000000
stock   11   average_daily_profit   1.747000000
stock   12   average_daily_profit   1.593000000
stock   13   average_daily_profit   1.421000000
stock   14   average_daily_profit   1.369000000
```

In this simulation the highest average daily profit occurs for the policy $x = 11$. The optimal policy can be shown to be $x = 10$.

9. An investment problem

This problem is how to allocate wealth between a safe asset (e.g. a bank account) and a risky asset (e.g. a stock).

Your initial wealth is w0. Investments decisions are annual. Your strategy is to invest a fixed fraction f of your wealth at the beginning of each year in stocks (and deposit the rest in a one-year bank certificate). The bank's annual interest rate is i% (fixed). The investment in stocks is multiplied by a factor r during a year. The annual multiplier is random, assuming the values r with probabilities p. The wealth after n years is simulated by

```
> Wealth:=proc(f,i,r,p,w0,n)
> local k,v;
> v:=array(0..n);v[0]:=w0;
> for k from 1 to n do
> v[k]:=evalf(f*v[k-1]*Simulation(r,p)+(1-f)*(1+i/100)*v[k-1])
> od;
> RETURN(v[n]);
> end:
```

Example: a policy of allocating 50% of wealth to stocks is simulated for 3 years. The bank's annual interest rate is 8%. The annual multiplier of stocks assumes the values [0,1,2] with probabilities [0.3,0.2,0.5], i.e. each year the investment doubles with probability 0.5, stays unchanged with probability 0.2, and is lost with probability 0.3. The initial wealth is 1000$.

```
> r:=[0,1,2]:p:=[0.3,0.2,0.5]:Wealth(0.5,8,r,p,1000,3);
```

1280.664000

To approximate an optimal policy (the optimal fraction `f` to be invested in stocks), we simulate the final wealth after `n` years for many values of `f`. The function `Policies(i,r,p,w0,n)` does this for `n` years, and 11 values of `f`= 0(0.1)1. The extreme cases `f`= 0 and `f`= 1 correspond to wealth invested only in bank and only in stocks, respectively.

```
> Policies:=proc(i,r,p,w0,n)
> local f,m;lprint('Return_after',n,'years');
> for m from 0 to 10 do f:=evalf(m/10);
> printf("%A %6G %-6s %12G\n",'Fraction_invested',f,'Return', evalf(Wealth(f,i,r,p,w0,n),9));
> od;
> end:

> r:=[0,1,2]:p:=[0.3,0.2,0.5]:Policies(5,r,p,1000,10);

Return_after   10   years
Fraction_invested 0        Return  1628.89462
Fraction_invested .1      Return  1353.62163
Fraction_invested .2      Return   584.232572
Fraction_invested .3      Return 13940.2879
Fraction_invested .4      Return   698.844896
Fraction_invested .5      Return  3212.61001
Fraction_invested .6      Return 78385.4223
Fraction_invested .7      Return   167.335548
Fraction_invested .8      Return  4245.19520
Fraction_invested .9      Return   532.207023
Fraction_invested 1        Return     0
```

In this simulation the policy of investing 60% of wealth in stocks gave the highest return.

10. Queuing systems

10.1. **Arrivals.** We are concerned with certain time events, called *arrivals*. The *inter-arrival times* (times between arrivals) are assumed iid with density $f(t)$. It is often the case that inter-arrival times have the *exponential density*

$$f(t) = \lambda\, e^{-\lambda t}\ ,\ t \geq 0\ , \tag{11}$$

where λ is the expected number of arrivals per unit time, called the *arrival rate*. The number of arrivals X in any period of length T can then be shown to be Poisson distributed (7) with parameters λT, i.e.

$$\text{Prob}\{X = x\} = \frac{(\lambda T)^x\, e^{-\lambda T}}{x!}\ ,\ x = 0, 1, 2, \cdots$$

Such arrivals are therefore called *Poisson* or *exponential*, e.g. [6, p. 860].

The function `ArrivalTimes(f,t,L,T)` simulates the inter-arrival times with density function `f(t)` during the time interval $[0, T]$. The 3rd argument `L` is the leftmost point of the support of `f`.

```
> ArrivalTimes:=proc(f,t,L,T)
> local a,v,n;
> a:=0;v:=vector(1,[0]);t:='t';
> while a<T do
> a:=evalf(a+ContSimulation(f,t,L));
> if a<T then v:=concat(v,vector(1,[a]));fi;
> evalm(v);od;
> end:
```

Example: Poisson arrivals occur at an hourly rate of $\lambda = 5$. Simulate the arrivals in 4 hours.

```
> f:=5*exp(-5*t):ArrivalTimes(f,t,0,4);
```

[0 , .3066065162 , .3264989939 , .4009640632 , .5688624114 , 1.319013383 , 1.511668888 , 1.558669103 , 2.032307273 , 2.044055041 , 2.113683049 , 2.136709110 , 2.161735605 , 2.497621862 , 2.753310851 , 3.124250705 , 3.226365836 , 3.347323512 , 3.430775390]

Example: inter-arrival times are uniform on $[0, 1]$ and iid. Simulate the arrivals in 10 hours.

```
> f:=1:ArrivalTimes(f,t,0,10);
```

$$[0\,,\ .1348300000\,,\ .3175800000\,,\ 1.234480000\,,\ 2.202280000\,,\ 2.232799000\,,\ 3.086759000\,,\ 3.910949000\,,\ 4.017519000\,,\ 4.256649000\,,\ 4.630179000\,,\ 5.339369000\,,\ 5.868629000\,,\ 6.780309000\,,\ 7.621939000\,,\ 7.722309000\,,\ 8.257649000\,,\ 8.818969000\,,\ 9.357239000]$$

10.2. **The inspection paradox.** The function **NextBus(f, t, L, T)** simulates the waiting time of a person arriving at random in a bus station, if the bus inter-arrivals times are distributed with density `f` and `T` is the duration of simulation. The 3rd argument, `L`, is the leftmost point of the support of f.

```
> NextBus:=proc(f,t,L,T)
> local v,k,s;t:='t';
> v:=ArrivalTimes(f,t,L,T);v:=convert(v,vector);
> s:=evalf((T/2)*Rand());
> k:=Lookup(v,s);
> evalf(v[k+1]-s);
> end:
```

The time since the last bus, the *spent time*, is similarly simulated by `LastBus(f,t,L,T)`,

```
> LastBus:=proc(f,,t,L,T)
> local v,k,s;t:='t';
> v:=ArrivalTimes(f,t,L,T);v:=convert(v,vector);
> s:=evalf(T*Rand());
> k:=Lookup(v,s);
> evalf(s-v[k]);
> end:
```

Example: buses arrive exponentially at an hourly rate of $\lambda = 2$. In a period of 5 hours we collect 20 waiting times of in the vector `Next`, and 20 spent times in a vector `Last`

```
> f:=2*exp(-2*t):
> n:=20:Next:=array(1..n):for m from 1 to n do Next[m]:=NextBus(f,t,0,10);od:evalm(Next):
> n:=20:Last:=array(1..n):for m from 1 to n do Last[m]:=LastBus(f,t,0,10); od:evalm(Last):
```

A statistical analysis of these vectors

```
> Statistics(Next);

Average .403627
Std_dev .425477
Median .251768
Maximum 1.725911
Minimum .009675


> Statistics(Last);

Average .3404
Std_dev .284394
Median .25049
Maximum .980071
Minimum .027908
```

The expected inter-arrival time of buses is 0.5 hour. We expect

$$\text{average waiting time} + \text{average spent time} \approx 0.5 \text{ hour}\ ,$$

but instead we get $.403627 + .3404 = .744027$. We see here an instance of the well known *inspection paradox*, [3, p. 185].
Example: the inter-arrival times of buses are uniform on $[0, 1]$ and iid. Simulate 20 waiting times and spent times for $T = 5$ hours.

```
> f:=1:n:=20:
> Next:=array(1..n):for m from 1 to n do Next[m]:=NextBus(f,t,0,5);od:evalm(Next):
```

```
> Last:=array(1..n):for m from 1 to n do Last[m]:=LastBus(f,t,0,5); od:evalm(Last):
> Statistics(Next);

Average .359556
Std_dev .262997
Median .261565
Maximum .84241
Minimum .01939


> Statistics(Last);

Average .281846
Std_dev .175078
Median .260432
Maximum .59076
Minimum .022917
```

Again,

$$\text{average waiting time} + \text{average spent time} = .359556 + .281846 > 0.5 = \text{expected inter-arrival time}$$

Observe the paradox by repeating the above simulations over longer periods and for larger values of n.

10.3. **The M/M/1 system.** When customers arrive for service, they may have to wait for an available server. *Queuing theory* is the study of queuing systems, comprised of customers, servers and queues.

An M/M/1 system is a queuing system with:

- a single server,
- Poisson arrivals with arrival rate λ, and
- Poisson service times with service rate μ.

It is assumed that

$$\mu > \lambda \,,$$

otherwise the server will be unable to cope with the incoming customers (resulting in an infinite queue).

M/M/1 is the simplest queuing system, and many of its characteristics can be computed explicitly, see [6, p. 868]. For example,

$$\text{server efficiency} \;=\; \frac{\lambda}{\mu} \,, \text{ (the fraction of time the server is busy)} \,, \tag{12}$$

$$\text{expected waiting time} \;=\; \frac{\lambda}{\mu\,(\mu-\lambda)} \,. \tag{13}$$

The function `Queue(lambda,mu,T)` simulates an M/M/1 system over T hours, recording for each customer the arrival time, the waiting time (0 if the server is free, positive otherwise) and the time of service completion. The server efficiency and the expected waiting time are then computed and compared with the theoretical values (12)–(13). Finally, the maximum waiting time is returned.

```
> Queue:=proc(lambda,mu,T)
> local a,b,c,f,g,k,w,cumw,cumb,maxw,server;
> f:=lambda*exp(-lambda*t);g:=mu*exp(-mu*t):#arrival & service densities
> a:=0;c:=0;w:=0;cumw:=0;maxw:=0;cumb:=0;
> k:=1;
> printf("%-6s %6s %6s %12s %8s\n",'Customer',
> 'Arrival','Waiting','Service','Server');
> printf("%-6s %6s %9s %9s %12s\n",'No','Time','Time','End','Status');
> while a<T do
> a:=evalf(a+ContSimulation(f,t,0),5);#a=arrival time
> if c<a then w:=0;server:='free'
> else w:=evalf(c-a);server:='busy' fi;#w=waiting time
> cumw:=evalf(cumw+w,5);maxw:=max(maxw,w);
> c:=max(a,c);#c=time service can start,b=service duration
> b:=ContSimulation(g,t,0);cumb:=cumb+b;c:=evalf(c+b,5);
> printf("%3g %9G %9G %9G %6s\n",k,a,w,c,server);
> k:=k+1;od;
> lprint();
```

```
> lprint(Server_efficiency);
> printf("%a %g %a %g\n",'Theoretical',evalf(lambda/mu),
> 'simulated',evalf(cumb/T,5));
> lprint('Average_waiting_time');
> printf("%a %0g %a %g\n",'Theoretical',evalf(lambda/(mu*(mu-lambda))),
> 'simulated',evalf(cumw/k));
> lprint('Max_waiting_time',evalf(maxw));
> end:
```

Example: an M/M/1 system with $\lambda = 3$, $\mu = 5$, is simulated for 50 hours. Only the first 15 and last 7 customers are shown.

```
> Queue(3,5,50);
```

```
Customer Arrival Waiting      Service   Server
No       Time    Time         End       Status

1          .30106   0           .30684    free
2          .54296   0           .92687    free
3          .62022    .30665     .95558    busy
4          .79308    .16250    1.1179     busy
5         1.1295     0         2.0077     free
6         2.1128     0         2.1909     free
7         2.4385     0         2.6645     free
8         2.7792     0         2.7885     free
9         3.2359     0         3.5401     free
10         3.4641     .0760     3.6518     busy
11         3.5363     .1155     3.8148     busy
12         3.6676     .1472     4.0242     busy
13         4.1893    0          4.2228     free
14         4.9919    0          5.1096     free
15         5.2142    0          5.4417     free


150        48.253     .028      48.450      busy
151        48.779    0          48.889      free
152        49.024    0          49.751      free
153        49.526     .225      49.760      busy
154        49.607     .153      50.087      busy
155        49.925     .162      50.150      busy
156        50.431    0          50.462      free

Server_efficiency
Theoretical .6      simulated .59144

Average_waiting_time
Theoretical .3      simulated .246471

Max_waiting_time   1.552
```

References

[1] The three doors puzzle: http://www.intergalact.com/threedoor/threedoor.html
[2] W. Feller, *An Introduction to Probability Theory and its Applications*, Vol I, Wiley 1950
[3] W. Feller, *An Introduction to Probability Theory and its Applications*, Vol II, Wiley 1966
[4] L. Gilman, *The Car and the Goats*, Amer. Math. Monthly **99**(1992), 3–7
[5] M. vos Savant, *Ask Marilyn*, Parade, (a) September 9, 1990, (b) December 2, 1990, (c) February 17, 1991
[6] H. Wagner, *Principles of Operations Research* (2nd edition), Prentice-Hall, 1975

RUTCOR–Rutgers Center for Operations Research, Rutgers University, Piscataway, NJ 08854-8003, USA
E-mail address: bisrael@rutcor.rutgers.edu