STATISTICAL COMPUTATIONAL METHODS

Seminar Nr. 2

Computer Simulations of Discrete Random Variables; Discrete Methods

1. Function **rnd** in Statistics Toolbox; special functions **rand** and **randn**.

The **rnd** function in Statistics Toolbox, with syntax

```
distrnamernd(par, M, N)
```

generates an $M \times N$ matrix of random numbers from the $distrname$ distribution with parameters par. Examples:

```
binornd(20,0.1,2,3)
unifrnd(1,10,5,4)
exprnd(5,100,1)
exprnd(5,100) = exprnd(5,100,100) !!
```

There are the special commands $rand$ for the Standard Uniform $U(0,1)$ distribution and $randn$ for the Standard Normal $N(0,1)$ distribution. So, they both only require the size M, N as input. If we only want *one* number, there is no need to specify the dimension.

```
>> rand
ans =
    0.8147
```

2. Using a Standard Uniform $U(0,1)$ random number generator, write Matlab codes that simulate the following common discrete probability distributions:

a. Bernoulli Distribution $Bern(p)$, with parameter $p \in (0,1)$:

$$X \begin{pmatrix} 0 & 1 \\ 1-p & p \end{pmatrix}$$

Solution:

First, we generate just *one* variable:

```
% Simulate Bernoulli distr. Bern(p).
clear all
p = input('p (in (0,1)) = '); % the parameter of the Bern distr.
U = rand; % random number from U(0,1)
X = (U < p); % X takes value 1, if U < p and 0, if U >= p
```

Then we comment this first part and put it into a loop, to generate a large sample.

```
% Generate a sample of N such variables
N = input('nr. of simulations = '); % try different numbers, 10,
                                    % 100, 1e4, 1e5
for i = 1 : N
    U = rand;
    X(i) = (U < p); % generate one at a time
end
% OR
% U = rand(1,N);
% X = (U < p); % generate all at once
```

Now, to see that indeed X has the desired pdf, we look at the relative frequency of the value 0 and that of the value 1 (those should approximate $1-p$ and p, respectively). Try different runs for different numbers of simulations.

```
% Compare it to the Bern(p)= Bino(1,p) distribution.
UX = unique(X) % the values of X listed ONLY ONCE, no repetitions
nX = hist(X,length(UX)); % the frequency of each value in UX
                         % (how many times each occurs)
relfreq = nX/N % the relative freq. nX/N approximates the prob.
```

You will get something like this

```
>> simulate_bernoulli
p (in (0,1)) = 0.6
nr. of simulations = 1e4
UX =
     0     1


relfreq =
    0.3993    0.6007
```

b. Binomial Distribution $B(n,p)$, with parameters $n \in \mathbb{N}, p \in (0,1)$.
Recall that a Binomial $B(n,p)$ variable is the sum of n independent Bernoulli $Bern(p)$ variables. We do just that, first we generate one such variable, then a sample of size N.

```
% Simulate Binomial distr. Bino(n,p)
clear all
n = input('n (in N) = ');
p = input('p (in (0,1)) = '); % the parameters of the Bino distr.
% Generate one variable
% U = rand(n,1);
% X = sum(U < p);

% Generate a sample of such variables
N = input('nr. of simulations = ');
for i = 1 : N
    U = rand(n,1);
    X(i) = sum(U < p); % generate one variable at a time
end
% OR
% U = rand(n,N);
% X = sum(U < p); % sum is computed on each column
%                 % generate all at once
```

To see how good the simulation is, we could either compare the graphs (of the true Binomial distribution and the simulated one) or compare various probabilities.

```
% Application/Comparison
fprintf('simulated probab. P(X = 2) = %1.5f\n', mean(X == 2))
fprintf('true probab. P(X = 2) = %1.5f\n', binopdf(2, n, p))
fprintf('error = %e\n\n', abs(binopdf(2, n, p) - mean(X == 2)))

fprintf('simulated probab. P(X <= 2) = %1.5f\n', mean(X <= 2))
fprintf('true probab. P(X <= 2) = %1.5f\n', binocdf(2, n, p))
fprintf('error = %e\n\n', abs(binocdf(2, n, p) - mean(X <= 2)))

fprintf('simulated probab. P(X < 2) = %1.5f\n', mean(X < 2))
```

```
fprintf('true probab. P(X < 2) = %1.5f\n', binocdf(1, n, p))
fprintf('error = %e\n\n', abs(binocdf(1, n, p) - mean(X < 2)))

fprintf('simulated mean E(X) = %5.5f\n', mean(X))
fprintf('true mean E(X) = %5.5f\n', n*p)
fprintf('error = %e\n\n', abs(n*p - mean(X)))
```

Here ia a run of this code:

```
>> simulate_binomial
n (in N) = 10
p (in (0,1)) = 0.7
nr. of simulations = 1e4
simulated probab. P(X = 2) = 0.00130
true probab. P(X = 2) = 0.00145
error = 1.467005e-04

simulated probab. P(X <= 2) = 0.00140
true probab. P(X <= 2) = 0.00159
error = 1.903864e-04

simulated probab. P(X < 2) = 0.00010
true probab. P(X < 2) = 0.00014
error = 4.368590e-05

simulated mean E(X) = 6.98750
true mean E(X) = 7.00000
error = 1.250000e-02
```

c. Geometric Distribution $Geo(p)$, with parameter $p \in (0, 1)$.
A Geometric $Geo(p)$ variable represents the number of failures that occurred before the first success.

```
% Simulate Geometric distr. Geo(p) by a discrete method.
clear all
p = input('p (in (0,1)) = '); % the parameter of the Geo distr.
% We count that number of failures until the first success
```

```
% X = 0; % initial number of failures
% while rand >= p % "rand < p" represents success, so
%                 % "rand >= p" represents failure
%   X = X + 1;  % add the number of failures
% end           % stop at the first success

% Generate a sample of such variables
N = input('nr. of simulations = '); % at least 10000
X = zeros(1, N);
for i = 1 : N
    while rand >= p
        X(i) = X(i) + 1;
    end
end
```

At the end, again, we compare the simulated results with the exact values. DO NOT forget to change "bino" to "geo" and the parameters (only one parameter, p). Also, the mean value $E(X) = \dfrac{q}{p}$ is different.

```
% Application/Comparison
fprintf('simulated probab. P(X = 2) = %1.5f\n', mean(X == 2))
fprintf('true probab. P(X = 2) = %1.5f\n', geopdf(2, p))
fprintf('error = %e\n\n', abs(geopdf(2, p) - mean(X == 2)))

fprintf('simulated probab. P(X <= 2) = %1.5f\n', mean(X <= 2))
fprintf('true probab. P(X <= 2) = %1.5f\n', geocdf(2, p))
fprintf('error = %e\n\n', abs(geocdf(2, p) - mean(X <= 2)))

fprintf('simulated probab. P(X < 2) = %1.5f\n', mean(X < 2))
fprintf('true probab. P(X < 2) = %1.5f\n', geocdf(1, p))
fprintf('error = %e\n\n', abs(geocdf(1, p) - mean(X < 2)))

fprintf('simulated mean E(X) = %5.5f\n', mean(X))
fprintf('true mean E(X) = %5.5f\n', (1 - p)/p)
fprintf('error = %e\n\n', abs((1 - p)/p - mean(X)))
```

Remark. To generate a Shifted Geometric $Y \in SGeo(p)$ variable, we simply add 1 to the variable above, $Y = X + 1$.

d. Negative Binomial Distribution $NB(n, p)$ with parameters $n \in \mathbb{N}, p \in (0, 1)$.
A Negative Binomial $NB(n, p)$ variable is the sum of n independent Geometric $Geo(p)$ variables.

```
% Simulate Pascal (Neg. Bin.) distr. by a discrete method.
clear all
n = input('n (in N) = ');
p = input('p (in (0,1)) = '); % the parameters of the NBin distr.
% for j = 1 : n
%    Y(j) = 0; % initial number of failures
%    while rand >= p
%        Y(j) = Y(j) + 1;  % add the number of failures
%    end
% end
% X = sum(Y);

% Generate a sample of such variables
N = input('nr. of simulations = '); % at least 10000
for i = 1 : N
    for j = 1 : n
        Y(j) = 0;
        while rand >= p
            Y(j) = Y(j) + 1;
        end
    end
    X(i) = sum(Y);
end
```

At the end, compare the simulated results with the exact values. DO NOT forget to change "geo" to "nbin", the parameters (two, n and p) and the expected value $E(X) = \dfrac{nq}{p}$.

e. Poisson Distribution $\mathcal{P}(\lambda)$ with parameter $\lambda > 0$.

Here, we use Algorithm 2.6 (Lecture 3) for generating arbitrary discrete distributions. In fact, since the pdf of a Poisson variable is

$$X \begin{pmatrix} k \\ \frac{\lambda^k}{k!} e^{-\lambda} \end{pmatrix}_{k=0,1,\ldots},$$

we use the algorithm in Example 2.7 (Lecture 3).

Recall that we use the cdf

$$F(i) = \sum_{k \le i} p_k$$

and here $p_k = \dfrac{\lambda^k}{k!} e^{-\lambda}$.

```
% Simulate Poisson distr. P(lambda), by a discrete method.
clear all
lambda = input('lambda ( > 0) = '); % the parameter
L = lambda;
% i = 0; % initial value
% F = exp(-L); % initial value of the cdf F(0)
% while (rand >= F) % check that U is in A_i,
                    % i.e. F(i-1) <= U < F(i);
%                   % the while loop ends when U < F(i)
%     F = F + exp(-L) * L^i/factorial(i)); % new value of F
%     i = i + 1; % count the values in A_i
% end
% X = i; % the Poisson variable

% Generate a sample of such variables
N = input('nr. of simulations = '); % at least 10000
for j = 1 : N
    clear F
    i = 0; % initial value
    F(j) = exp(-L); % initial value of the cdf F(0)
    while (rand >= F(j)) % check that U is in A_i,
                      % i.e. F(i-1) <= U < F(i);
                          % the while loop ends when U < F(i)
```

```
        F(j) = F(j) + exp(-L) * L^i/factorial(i); % new value of F
        i = i + 1; % count the values in A_i
    end
    X(j) = i; % the Poisson variable
end
```

And then compare the simulated results with the true values. DO NOT forget to change "nbin" to "poiss", the parameter (only one, λ) and the expected value $E(X) = \lambda$.