Departament de Disseny i Programació de Sistemes Electrònics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Pseudorandom Noise Generator

Based on a linear feedback shift register with Octave

Jordi Bonet-Dalmau

September 19, 2014

In many areas where producing an unpredictable result is desirable random sequences are used. Among these areas there is computer simulation: e.g. when simulating a system to know its performance in the presence of noise. But it is not easy to achieve the goal of true randomness. Otherwise, it is easy to generate sequencies whose properties approximate the properties of random sequences. These sequences are called pseudorandom.

In this lab session we will build a pseudorandom sequence generator based on a linear feedback shift register (LFSR), an structure that you have seen in *Introducció als Sistemes Digitals*. This generator could be implemented in many ways. We have chosen a computer in which a program written in a high level language is executed. We can use any language you know like Python, C or Octave, although we will focus on the later as we have done in *Senyals i Sistemes*. The pseudorandom sequence will be send to the sound card of the computer in order to hear how this generated noise sounds.

Finally, you must realize that a simple binary noise generator can be implemented using a single digital output of devices you know how to control, like a microcontroller or an FPGA, using an small part of the resources they have.

1 Using the sound card from Octave

Before starting to build the pseudorandom noise generator, we will verify that the communications between Octave and the sound card is good. What follows is a *résumé* of the information given in the two first lab sessions of *Senyals i Sistems*: *Pràctica 1* and *Pràctica 2*.

The best way to acces the sound card is executing a command line program of your operating system called sox. This program can be called as play to play a sound and as rec to record a sound. You can verify that the packet sox is installed on your system by simply executing on a terminal man sox. If not installed, execute on the same terminal:

```
$ sudo aptitude install sox
```

Task 1. If you have succeeded in the previous step, try to record a sound using the function *rec_so.m*.

```
function x=rec_so(N, Fm)
    if nargin==1, Fm=48e3, elseif (nargin !=2), print_usage (), end
    file=[tmpnam(), '.wav'];
    tf=N/Fm;
    input ('Please␣hit␣ENTER␣and␣speak␣afterwards!\n', 1);
    %cmd = sprintf ('rec -c1 -r%d %s trim 0 %d',Fm, file, tf), system (cmd);
    system(['rec -c1 -r ',num2str(Fm),' ',file,' trim 0 ',num2str(tf)])
    x=wavread(file);
    system(['rm ',file]);
end
```

You can save this function in a folder where you will open octave or you can add the path using addpath from octave.

```
> addpath("/home/user/where_function_is")
```

If you are using a lab computer, connect the microphone to the rear input, not the front input. Choose the sampling frequency F_m and the duration of the recorded sound t_f, and determine the number of samples N.

```
> Fm=xx;tf:yy;N=f(tf,Fm);x=rec_so(N,Fm)
```

You can plot the vector of samples x to verify the amplitude of the sound you have recorded. Be carefull not to saturate the microphone (maximum abolute value is 1).

Task 2. Now play the recorded sound using using the function *play_so.m*.

```
function play_so(x,Fm)
    if nargin==1, Fm=48e3, elseif (nargin !=2), print_usage (), end
    file=[tmpnam(),'.wav'];
    wavwrite(x,Fm,file);
    system(['play ',file]);
    system(['rm ',file]);
end
```

If you are using a lab computer, connect your earphones to the rear input, not the front input.

```
>play_so(x,Fm)
```

What is the length of the played sound if you use a sampling frequency F_m diferent that the one you have used to record?

2 Noise with normally distributed values

First we will use the **randn**.m function from Octave. This function gives normally distributed pseudo random values with zero mean and variance one.

Task 3. Play with **randn**.m to know what a normally distribution is.

```
> N=1e5
> x1=randn(N,1);max(x1),min(x1),mean(x1),var(x1),
> figure(1),plot(x1)
> n=[-5:.1:5];h=hist(x1,n);figure(2),plot(n,h)
```

Send the samples to the sound car at different sampling frequencies and verify that it is *like* noise.

```
>play_so(x1,Fm)
```

You can repeat with **rand**.m which gives uniformly distributed values on the interval (0,1).

3 Noise with binary values

Now we will try to make noise from binary values. As it has been said before, this kind of noise is easily generated using a single digital output. To achieve randomness we will convert the previous positive random values to one binary value v_0 and the negative to the other binary value v_1. To use all the dynamic range of the sound card these values could be $v_0 = -1$ and $v_1 = 1$.

Task 4. From the previous computed normally distributed pseudo random values x_1 compute a new set of binary pseudo random values x_2. Instead of using a loop to verify the sign of each element of x_1 use logical indexing. Here is an example of logical indexing.

```
> x1=randn(1,5)
x1 =   -1.277148   -1.068257    0.236539    0.076126   -0.722842
> x2=x1;
> x2>=0
ans =    0    0    1    1    0
> x2<0
ans =    1    1    0    0    1
> x2(x2>=0)=1
x2 =   -1.27715   -1.06826    1.00000    1.00000   -0.72284
> x2(x2<0)=-1
x2 =  -1   -1    1    1   -1
```

Send the samples to the sound car and compare the results when using x_1 and x_2

```
>play_so(x1,Fm),pause(1),play_so(x2,Fm)
```

4 Noise with binary values using LFSR

Now we will make noise avoiding the function randn.m of Octave. The idea is to use an algorithm that can be easily implemented. We will use a linear feedback shift register (LFSR), an structure that was introduced at the beginning of this degree.

Task 5. Use one of the polynomials that appear on the following link, *LFSR*, to compute a new set of binary pseudo random values x_3. Here are some of the steps you will need to do. First you have to initialize the shift register *sr* with a *seed*. I suggest using all ones. Next you have to start an iterative process (each iteration is equivalent to rising clock). In this iteration you have to compute the value *in*, computed from some values of the shift register, that is at the input of the first shift register. You have to store *in* at each iteration in order to obtain the pseudo random sequence. Finally, don't forget to convert the values of this sequence $\{0, 1\}$ to the dynamic range of the sound card $\{1, -1\}$

Here is an example of some lines of the code using a polynomial of degree $n = 10$ to compute N values.

```
sr=ones(n,1);
x3=zeros(N,1);
for i=1:N
   in=xor(sr(10),sr(7));
   ...
end
...
play_so(x3,Fm)
```

Compare the results of hearing x_1 and x_2 and x_3 for diferent degrees n. Do you like the result for $n = 10$? and for $n = 16$?

You can compare the results with some online white noise generator like *onlinetonegenerator* or *simplynoise*.

5 Spectrum of the noise

Now you can compare the spectrum of each one of the generated noise sequences x_1, x_2 and x_3 Using the function *f_TF.m*.

Task 6. Play each one of the noises at different F_m and observe its spectrum. Try to relate the spectrum at different F_m with how the noise sounds. The human ear sensitivity range is commonly given as 20 Hz to 20 kHz, though there is considerable variation between individuals, especially at high frequencies. In addition, earphones, and speakers in general, don't cover well frequencies above 10 kHz.

```
[X3,F]=f_TF(x3,Fm);
figure(3),plot(F,abs(X3))
```

6 Advanced activity

During the reproduction of the previous signals some problems related to the sound card may arise. Sound cards can work with only one or a few sampling frequencies. The sound card can deal with other sampling frequencies but at the cost of processing using sample rate-conversion algotithms. Depending on the sound card possibilities, recording with some non standard sampling frequencies can give wrong samples. If you experience problems try to use always an standard frequency like 44 100 Hz.

Another issue is the digital to analog converter of the sound card. It will be interesting to compare the analog signal at the output of different sound cards when a periodic pattern (easy to see on an oscilloscope) is send to them.

Task 7. First generate an $N-$length sequence x_3 with a periodic pattern made of one -1 and two 1. We assume that the sampling frequency of these samples is F_{m3}. Then generate another sequence x_4 from x_3 which is made repeating each value of x_3 n_r times (avoid the use of a loop of length N). As a consequence, the length of x_4 is n_r times that of x_3. We assume that the sampling frequency of these samples is $F_{m4} = F_{m3} \times n_r$. An example is showed in *Figura* 1.

Connect and appropiate cable to the output of the sound card and the input of the oscilloscope (jack on the computer side and pins on the oscilloscope side). Finally play these sequences and visualize them on the oscilloscope.

```
>play_so(x3,Fm3),pause(1),play_so(x4,Fm4)
```

Try different patterns and sampling frequencies.

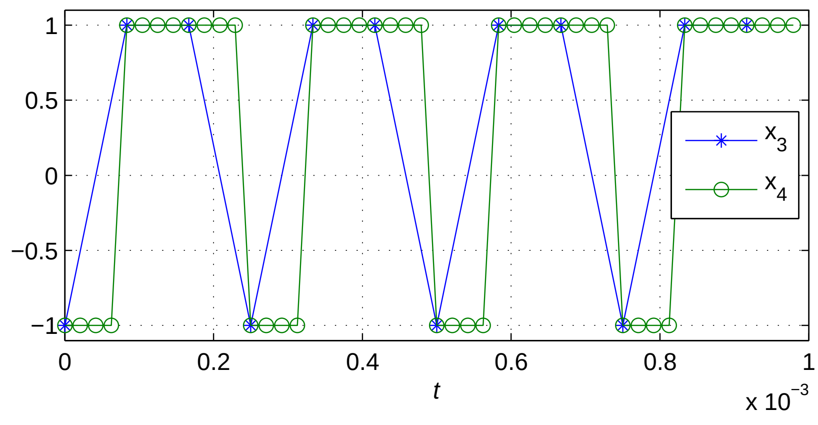

Figure 1: Samples of x_3 and x_4 during 1 ms. Pattern [-1 1 1], $F_{m3} = 12\,\text{kHz}$, $n_r = 4$