Ecole Normale Supérieure de Cachan
Master recherche 2ème année
Mathématiques Vision Apprentissage

"Stochastic methods for image and signal analysis" Feuille de Td no.4

Exercise 1: Simulating continuous random variables with Matlab

Box-Muller Algorithm

1. Let $\mathcal{U}$ and $\mathcal{V}$ be two random variables independent and uniformly distributed on $[0,1]$. Prove that the random variables $\mathcal{X}$ and $\mathcal{Y}$ defined by

$$\mathcal{X} = \sqrt{-2\ln(\mathcal{U})}\cos(2\pi\mathcal{V}) \text{ and } \mathcal{Y} = \sqrt{-2\ln(\mathcal{U})}\sin(2\pi\mathcal{V})$$

   are independent and follow the Gaussian distribution with mean 0 and variance 1 (denoted by $\mathcal{N}(0,1)$ and also called the normal distribution). This result is closely connected with the well-known method for evaluating $\int_{-\infty}^{+\infty} e^{-x^2/2}dx$ by squaring it and passing to polar coordinates. Can you see why?

2. Use this result (called Box-Muller algorithm) and the Matlab function `rand` to sample standard normal random variables: take N samples and plot on the same figure their histogram and the density function of the standard normal distribution.

Simulation of Gaussian vectors

1. Prove that if $\mathcal{X} = (\mathcal{X}_1, ..., \mathcal{X}_k)$ is a gaussian vector with mean 0 and covariance matrix equals to the Identity, and if $A \in \mathcal{M}_k(\mathbb{R})$ is such that $AA^t = C$, then $\mathcal{Y} = A\mathcal{X} + \mu$ is a gaussian vector with mean μ and covariance C. How can this be used in Matlab for sampling gaussian vectors?

2. Toy example: get N samples of a gaussian vector with mean μ and covariance C, where `C=[5 3 2 ; 3 4 1 ; 2 1 2]` and `mu=[-2 1 4]`. (Hint: use the Matlab function `chol`). Check the obtained results thanks to the Matlab functions `mean` and `cov`.

The inversion method

1. Prove the following result:
   The cumulative distribution function (cdf) of a random variable $\mathcal{X}$ is defined by $F(x) = \mathbb{P}(\mathcal{X} \leq x)$. If $\mathcal{U}$ is a random variable uniformly distributed on $[0,1]$, then $\mathcal{X} = F^{-1}(\mathcal{U})$ is a random variable with cdf F.

2. Use this result (called inversion method) to sample from an exponential distribution of parameter λ. We recall that its density function is $\lambda \exp(-\lambda x)$ for $x \geq 0$ and 0 otherwise. Take N samples and compare on the same figure the histogram of the obtained values and the density function.

3. Do the same thing for a Cauchy distribution (the density function is $1/\pi(1+x^2)$).

Checking the law of large numbers

1. Get N independent samples $X_1,\ldots,X_N$ of a distribution of your choice (Bernoulli, normal, exponential, etc.). Then plot on the same figure the function $n \mapsto (X_1 + \ldots + X_n)/n$ and the value of $\mathbb{E}(\mathcal{X})$. What do one see?

2. Do the same thing for a Cauchy distribution, and check experimentally that the law of large numbers fails in this case.

Exercise 2: Synthesizing music

The easiest way to synthesize random samples from stationary Gaussian processes is by randomly choosing Fourier coefficients from a Gaussian distribution with mean zero and whatever variance you want and then taking the Fourier transform. The Fourier coefficients are independent and one can use Box-Muller or, in MatLab, the built-in routine `x = randn(N,1)` and multiplying by the desired standard deviation.

1. First synthesize 'colored noise' with a *power law power spectrum*. In other words, take the real and imaginary part of the Fourier coefficients $\hat{s}_k, 0 \leq k < N$ to be random normal variables with mean 0 standard deviation $1/\min(k, N-k)^\lambda$, but with $\hat{s}_0 = 0, \hat{s}_{N-k} = \bar{\hat{s}}_k$. Alternately, one can ignore the condition $\hat{s}_{N-k} = \bar{\hat{s}}_k$ and take the real part after the Fourier transform. Note that low frequencies come from the Fourier coefficients with index either near zero or near N. You can take the size of the vector N to be 1024. Do this for $\lambda = 0, 0.5, 1, 2$ and plot the results. What do you see?

2. Next synthesize random single notes of music, by taking the Gaussian distribution to be $\frac{1}{Z}e^{-Q(\vec{s})/2}$ where $Q(\vec{s}) = \Sigma_{k=0}^{N-1}(s(k+p)-s(k))^2 + cs(k)^2$. Here $s(k)$ is assumed to 'wrap around' above N. Recall that we worked out the corresponding power in the lecture notes. You may try $N = 8192, p = 32$ and various quite small c's to look for samples that resemble music. Listen to what you have produced via MatLab's command `sound` whose default is 8192 Hz., making your note have frequency 256 Hz.

3. Some notes which are better approximations to various instruments can be synthesized by choosing a better rule for the power in various frequencies. Try experimenting so only a few harmonics are present. A realistic note is also amplitude modulated: it may have an attack where the power rises and a slow decline before the next note. This can be achieved by multiplying the actual sound by a suitable function.

4. Finally, let's introduce multiple notes. Take a longer period, say 2^{15} samples, and now divide it into intervals by a handful of Poisson points and put notes as above in each interval. You can synthesize the actual model described in the lecture notes, but this will sound like some weird composer. But now you are in a position to improve the model in many ways, e.g. using measures, bars, scales and keys.

Exercise 3: Parsing an oboe cadenza

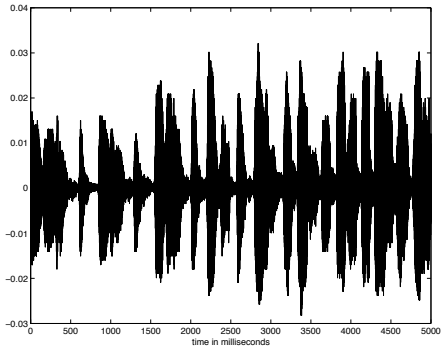

Figure 1: The 5 second interval of oboe music in this exercise.

Find the individual notes in the simple piece of music which started this chapter. The music is in the file `fivesec.au`, in a compact format readable by MatLab via `auread`. Alternately, it is also there as an ascii file of 40,000 numbers. The numbers are samples of the sound wave taken at 8,000 Hertz for a interval of 5 seconds. In the course of this interval, the oboe plays 22 notes (some are double, i.e. the same note with a short rest in between). The goal is to find the note boundaries and the frequencies of the notes themselves by implementing the piecewise Gaussian model discussed in above, where discrete time is divided into segments $\{0 = n_0 < n_1 < \cdots < n_K = 40000\}$ and the k^{th} note is played in the interval $[n_{k-1}, n_k]$. Let the approximate discrete *period* of the k^{th} note be the integer p_k and model the note in this interval by the Gaussian distribution:

$$p(s|_{[n_{k-1},n_k]}) = \frac{1}{Z} e^{-\sum_{n_{k-1}}^{n_k - p_k} (s(n+p_k) - s(n))^2}.$$

Here you may assume that any period will be in the range $5 \leq p \leq 40$, (which correspond to frequencies of $8000/p$ which is the three octave range $[200, 1600]$ hertz). In addition, you should put a simple exponential prior on the number K of notes by $\mathbb{P}(K = k) = \frac{1}{Z} e^{-ak}$. To find the most probable sequence of notes in this model, we suggest precalculating the table:

$$C(n, p) = \sum_{k=1}^{n} s(k)s(k + p).$$

Then we can approximate the exponent by:

$$\sum_{k=1}^{K} \sum_{n_{k-1}}^{n_k - p_k} (s(n + p_k) - s(n))^2 \approx 2 \sum_{n=1}^{40000} s(n)^2 - 2 \sum_{k=1}^{K} (C(n_k, p_k) - C(n_{k-1}, p_k)).$$

Your algorithm should proceed by dynamic programming, calculating by induction on the time step n the best segmentation of the music from the beginning through sample n. To do this, you must loop only over the immediately preceding note boundary (including 0) *and* possible p's in the last time segment. If you run out of memory, a good way to simplify the calculation is to restrict the time steps to multiples of some small number, e.g. 10. Find the

optimal segmentation of the whole sequence with various values of the parameter a in the prior. Too large values of a lead to segmentations with too few notes and small values of a may lead to too many notes. Remember there should be 22 notes. Check your results by running `specgram` or simply `plot` to see what is happening.

```
s=auread('fivesec.au');
figure; plot(s)
N1=length(s)
maxP = 40;
delT = 5;
N = N1/delT;
a = 0.001;
% table of autocorrelation:
cor = zeros(N,maxP);
for p=1:maxP,
    x = cumsum(s(1:N1-p) .* s(1+p:N1));
    cor(1:length(1:delT:N1-p),p) = x(1:delT:N1-p);
end
% dynamic programming
E = zeros(N,1);
for k=1:(N-1),
   X = a + cor(1:k,:) - ones(k,1)*cor(k,:) + E(1:k)*ones(1,maxP);
   E(k+1) = min(min(X(:,5:maxP)));
   [u,v] = find(X==E(k+1));
   T(k+1) = u(1);
   P(k+1) = v(1);
end

t0 = N-maxP;
times = t0;
periods = [];
while t0 > 1
     t1 = T(t0);
     times = [ t1 times];
     periods = [ P(t0) periods];
     t0 = t1;
end

periods
times*delT

plot(1:delT:40000,s(1:delT:40000))
hold on;
plot(times*delT,0,'r*')
title('oboe cadenza')
```