Modern theory of detection and estimation
Lab 3: Linear filtering

Miguel Lázaro Gredilla

Introduction

In this lab session you will perform Bayesian and non-Bayesian filtering, first on toy data to grasp the basic concepts, and then on a real echo-cancellation scenario.

1. Working with toy data

We will consider first a toy data scenario. You will be generating observed data yourself and then trying to estimate the filter you used.

Use this code at the beginning of your script:

```
var_s = 1;
var_n = 0.2;
N = 100;

s = [0.54 1.83 -2.26 0.86 0.32]';
randn('seed',0);rand('seed',0);
```

This sets the variance of the filter weights σ_s^2, the variance of the noise σ_ε^2, the length of the input and output signals, the filter $\mathbf{s}$ itself, and resets the random number generator, so that hopefully, despite using random numbers in this section, everyone gets the same results.

1.a) Start by generating an input signal $u[n], \quad 0 \leq n < N$ by creating a random vector of length N containing independent realizations from the pdf $\mathcal{N}(0, 1)$. Filter it with $\mathbf{s}$ and add white Gaussian noise of variance `var_n` to produce signal `x`, which will also be of length N.

In order to create matrix $\mathbf{U}$ you might want to check MatLAB's function `toeplitz`.

You can check that the output of the filter (before adding the Gaussian noise) is the same as the one obtained with MatLAB's function `filter`. Though you could use `filter` at this point and avoid building matrix $\mathbf{U}$ altogether, you will be needing it later, so it's recommended that you only use the `filter` command as a correctness check.

1.b) Compute the predictive probability density of the filter given the observed output $p(\mathbf{s}|\mathbf{x})$. You should be getting something like this:

```
mean_s =

    0.5204
    1.7575
   -2.3716
    0.8857
    0.3142


cov_s =

    0.0022    0.0002    0.0001    0.0002   -0.0001
    0.0002    0.0023    0.0002    0.0002    0.0002
    0.0001    0.0002    0.0023    0.0002    0.0002
    0.0002    0.0002    0.0002    0.0023    0.0002
   -0.0001    0.0002    0.0002    0.0002    0.0023
```

Provide an interpretation for this result.

1.c) Let's turn now to prediction. If the next input sample, $u[100]$, was equal to zero, what would the predictive probability density of the corresponding output $x[100]$ (after traversing the filter and i.i.d. Gaussian noise is added) be? Compare your result with the correct one:

```
mu_star =
    2.3322

v_star =
    0.2036
```

2. Echo cancellation scenario

In this section we consider a scenario of voice chat over a noisy channel with echo.

You will find two appropriately named variables in the attached data file `data.mat`. These are the voice registers, sampled at 22.05 KHz and containing 100 Ksamples each. The provided vectors include data for $0 \le n \le N-1$, with $N = 100000$.

Variable `voiceout` contains the voice of the local speaker as recorded on the local machine, whereas variable `voicein` contains the voice of the remote speaker as received on the local machine plus a distorted version of the local speaker's voice after traversing the channel to the remote machine and back (out-going and incoming channels are not necessarily identical).

More formally:

$$\texttt{voicein}[n] = \texttt{voiceremote}[n] + \sum_{k=-\infty}^{\infty} \texttt{voiceout}[n-k] s_n[k] \tag{1}$$

where $s_n[k]$ is a causal, time-varying (hence the subscript n) FIR filter that models the full round trip of the local voice. Signal `voiceremote` includes any distortion to the original remote speaker's voice that is introduced by the incoming channel, as well as any additional noise — we will not attempt to revert those, but only remove distortions related to `voiceout`.

MatLAB provides functions that will allow you to play these files (as well as any reconstruction of `voiceremote` that you may have). You can use this to informally test the quality of your reconstruction.

The round-trip channel model $s_n[k]$ is known to have very few non-zero values. More specifically, $s_n[k]$ is non-zero only for the following values of k:

```
nonzerolags = [3122 5953 9999 14999 18999 29295 39385];
```

From a programmatic point of view, be reminded that, in mathematical notation, $s_n[k]$ is a zero-based vector, whereas MatLAB uses one-based vectors.

2.a) For this part, disregard the time variation of the round-trip channel, and model it as a constant round-trip channel $s[k]$.

Provide a probabilistic estimation of $s[k]$. You know that only a few values are non-zero, so focus on those elements only. You can use the following ground-truth values in this case:

```
var_s = 0.2;
var_n = 4e-5;
```

Pitfall: If you try to use the standard equations from the notes in this case, you will run into trouble twice. First, when you try to compute $\mathbf{U}$, you will, almost surely, run out of memory. Second, even if you used a computer from the future, applying the standard equations to compute the probabilistic prediction will result in placing non-zero probability on non-zero values for all coefficients of $s[k]$. And we know that most coefficients are exactly zero, so that belief is incorrect.

Hint: Follow the derivations in the notes, but use the fact that only a few values on $s[k]$ are non-zero to obtain a very similar, yet much faster equation to compute a solution that provides a joint posterior for the non-zero coefficients (and by definition predicts exactly zero for all the remaining coefficients).

You should be getting something like this:

```
mean_s =

    1.3189
    0.5548
    0.5585
    0.0007
    0.2646
    0.2187
    0.2594


cov_s =

   1.0e-06 *

    0.2043   -0.0087    0.0001   -0.0000   -0.0031    ...
        0.0002   -0.0000
   -0.0087    0.2042   -0.0001    0.0002    0.0006    ...
        0.0018   -0.0000
```

```
     0.0001   -0.0001    0.2040    0.0003    0.0005    ...
        0.0053    0.0000
    -0.0000    0.0002    0.0003    0.2039    0.0014   ...
       -0.0003    0.0024
    -0.0031    0.0006    0.0005    0.0014    0.2039    ...
       0.0001    0.0002
     0.0002    0.0018    0.0053   -0.0003    0.0001    ...
        0.2040   -0.0002
    -0.0000   -0.0000    0.0000    0.0024    0.0002   ...
       -0.0002    0.2039
```

2.b) Provide an MMSE estimation of `voiceremote`. Listen to the provided `voicein` and `voiceout` and then to your obtained estimation of `voiceremote`.

When listening to your estimation, you should be able to hear some echos of `voiceout` and someone else talking faintly that could not be heard when listening to `voicein` or `voiceout`.

2.c) Now make use the LMS algorithm. As mentioned in class, this algorithm enjoys adaptivity properties, so it should be able to *track* the channel, which is actually time-varying. Remember that $s_n[k]$ is non-zero only for the values of k mentioned before.

Also, note that the LMS algorithm provides an online estimation of `voiceremote`. I.e., after you process sample n from `voiceout` and `voicein`, you already get an estimation of sample n of `voiceremote`. This means that estimations can be provided on-the-fly, instead of in batch form as we did previously.

Hint: Using the vanilla LMS algorithm will result in unnecessary overhead. You can simplify it by taking advantage of having just a handful of non-zero coefficients. This is not required, but will turn out to be convenient.

If you listen to `voiceremote_LMS`, you should be able to hear a distorted version of the remote speaker without clear echos of `voiceout`.

2.d) You can check that this result is better than the previous one just by listening to it and proiding your subjective impresion. Can you think of a way to provide an objective assessment of the quality of this estimation as compared to your previous estimation?

3. Extension exercise

3.a) The evolution of every non-zero coefficient at lag k follows this equation

$$s_n[k] = c^{(k)} + \sum_{j=1}^{10} a_j^{(k)} \cos(2\pi nj/f_s + \phi_j^{(k)}).$$

with $f_s = 22050$ Hz. I.e., each coefficient is the composition of a constant term plus up to 10 sinusoids. We say "up to", because we can remove any or even all sinusoids for a given coefficient by setting the corresponding $a_j^{(k)}$'s to zero.

Estimate the values of $\{c^{(k)}, a_j^{(k)}, \phi_j^{(k)}\}$ for every k corresponding to a non-zero coefficient and $1 \leq j \leq 10$.

Hint: Cast the problem as a linear regression problem.

3.b) Use the previous estimation of $s_n[k]$ to estimate `voiceremote`. If you listen to it you should hear an almost perfect reconstruction: No echos, no sound distortion.

4. Reference implementation

As a reference, a complete implementation of the solution to this questions took less than 70 lines of MatLAB code (including comments and proper spacing) and takes less than 10 seconds to run on a mid-2012 macbook air. Part 3.a) is the most computationally intense and can take up to 1GB of memory.