ELEN90052 Advanced Signal Processing

Lab SP2-41: Online parameter estimation

Aim: To estimate parameters in models of audio signals online.

Lab: The lab consists of a software component using Matlab and a hardware component using Analog Devices' BF533 DSP.

Assessment: The lab is assessed based on a written lab report and on a demonstration of the algorithm implemented on the DSP board at the end of the lab class. The lab constitutes 15% of the overall assessment of the course. The marking scheme will be posted on LMS (the subject's web-page) and outlines the levels of expectation in various areas.

Student groups: The students should work in groups of two. It is preferable, but not required that you have the same lab partner for the hardware part as for the Matlab part. If you have different partners hand in separate reports (with your partner) for the Matlab part and the hardware part. (If you prefer you can also do the whole lab on you own.)

Submission of the reports: The reports for both parts should be submitted via the turnitin link on LMS by 9am on Monday of week 9.

Reports: The reports should be clearly written and explain in a logical way how the different tasks in the project have been carried out. Choices you make (e.g. model order, output and input variables, initial values, etc.) should be explained and justified. Results obtained using Matlab should be explained, i.e. it is not sufficient to copy the output of Matlab without further explanations of what the numbers or graphs mean. Figures should be included where it is appropriate. The Matlab and c code should be included in an appendix. The maximum length of the reports are 15 pages excluding the appendix. The c code should also be given to the lab demonstrator in electronic form.

Collaboration between and within groups: It is perfectly OK to discuss problems and possible solutions with other groups. However, each group has to carry out the lab independently, and e.g. copying of other groups' c code or Matlab code is not acceptable. Both group members should do an equal amount of work on both the lab itself and the writing of the report. It is not acceptable that one group member does the lab and the other writes up the results.

1 Matlab part

A music signal is played through a loudspeaker and recorded by a microphone. There are severe echo effects in the room. Your task is to develop a model for the signal picked up by the microphone and to estimate unknown parameters in the model. The algorithms you develop should be suitable for real time implementation.

1.1 Project tasks

Note: Students can introduce additional assumptions which do not violate the main specifications. You are not allowed to use inbuilt Matlab routines for parameter estimation.

a) In this part the echo paths remain constant. There are two echos present in the signal picked up by the microphone. The first echo is delayed by exactly 1 second, and the second echo is delayed by exactly 2.5 seconds. The echo amplitudes are constant, but unknown. You have access to both the signal played through the loudspeaker and the signal picked up by the microphone. The latter signal comes in two versions, a noise free one and a noisy one. All signals are uniformly sampled at $F_s = 8192$Hz.

Download the file `data1.zip` from LMS. The file `data1.mat` contains the signals `loudspeaker`, `mike1` and `noisymike1`, which are respectively the signal played through the loudspeaker, the signal picked up by the microphone in the case of no background noise, and the signal picked up by the microphone with background noise.

Implement RLS and LMS algorithms for estimating parameters in the model of the signal picked up by the microphone. Discuss and demonstrate the effect of design parameters and initial conditions. How does the presence of noise affect the results? Compare the results of the RLS and LMS based algorithms. Use the estimated models to reduce/remove the echo. Assess how well you have reduced the echo.

Note: You can play the signals through the audio device on your computer using the Matlab command `sound`, e.g. `sound(mike1)`, and you can also qualitatively test how successful you have been in reducing the echo that way.

Note: Although the final goal is echo cancellation, you will mainly be judged on your design and discussion of the parameter estimation algorithms.

b) This is a repeat of part a). The only difference being that the echo amplitudes are time varying. Noisy and noise free data `mike2` and `noisymike2` are stored on the file `data2.zip` on LMS.

c) Assume now that the delays in the two echo paths are "only" known up to ± 1 millisecond. I.e., it is between 0.999 and 1.001 seconds for the first echo, and between 2.499

and 2.501 seconds for the second echo. Design, implement and discuss algorithms in this case. Use the data from b).

d) (for bonus marks) Assume now that the delays are exactly known again, but that you do not have access to the signal from the loudspeaker. Design, implement and discuss algorithms for this case. Use the data from a).

2 DSP part

2.1 Prelab questions

You must show your workings on the prelab questions to the lab demonstrator before you are allowed to start the lab.

Read through the description of the lab and the lab tasks in sections **??**, **??** and **??**.

1. Write pseudo code for the implementation of the LMS and RLS algorithms for the problem in the lab. All matrix and vector operations must be written out as a number of scalar operations in pseudo c-code.

2. Try to minimise the number of floating point multiplications and divisions in the RLS algorithm. (The audio signals are sampled at 48 kHz so there is a limit to how may floating point multiplications and divisions you can carry out in each sampling interval.)

3. If the signal $u_2(t)$ in task d) in section **??** disappears after two seconds how would you change μ and λ in order to make it disappear after five seconds? (Only a qualitative answer is required, but explain your reasoning.) Express the guideline for selection of μ in the handout in terms of the signals $u_1(t)$ and $u_2(t)$, and explain how you will apply the guideline in the lab.

2.2 Lab part

Bring along the following to the lab

- A portable music player (e.g. smartphone with music files). Hereafter referred to as your audio device.

In this part of the lab you are going to estimate the gains of audio signals in real time using Analog Devices Blackfin 533 Digital Signal Processor which is a 16 bit fixed point processor (further details on `http://www.analog.com/processors/processors/blackfin/`). You will be using the CCES development environment (see separate note for a description of this program package) and the ADSP-BF533 EZ-Kit Lite board (hereafter called the board or the DSP board).

Figure 1: Lab setup

2.3 Lab setup

The DSP board is connected to a PC through a USB port. The audio output of the PC is connected to the audio input 1 of the board and your audio device will be connected to audio input 2. The audio output of the board is connected to a pair of loudspeakers. Your demonstrator will assist you if you have trouble setting things up correctly. You will program the estimation algorithms in c using CCES. The CCES environment provides facilities for debugging, compiling, linking and downloading of the code from the PC to the board. On the DSP board the audio signal from the PC and your audio device are sampled at 48 kHz.

The sampled signals are denoted by $u_1(t)$ (PC) and $u_2(t)$ (audio device). The signal is then multiplied by a time varying gains $\theta_1(t)$ and $\theta_2(t)$, to produce the signal

$$y_1(t) = \theta_1(t)u_1(t) + \theta_2(t)u_2(t)$$

You are going to write a function which estimates $\theta(t) = \begin{bmatrix}\theta_1(t) & \theta_2(t)\end{bmatrix}^T$ using LMS and RLS. The function is called every sampling interval with the current values of u_1, u_2, y_1 and the estimate $\hat{\theta} = \begin{bmatrix}\hat{\theta}_1 & \hat{\theta}_2\end{bmatrix}^T$. There are separate signals for the left and right channel. The gains are the same for the two channels and you are only going to use the signals from the left channel for estimation of the gains.

The signal sent to the loudspeakers is

$$y(t) = (1 - |\theta_1(t) - \hat{\theta}_1(t)|)u_1(t) + \alpha(\theta_2(t) - \hat{\theta}_1(t))u_2(t) \quad (1)$$

i.e. as the estimate approaches the true value of θ you should hear the signal $u_1(t)$ from the audio output of the PC. The second signal is amplified (or attenuated) with a factor α whose default value is 1.0.

The skeleton for the gain estimation function is given in Appendix **??** together with some c-programming hints.

The variables `in1`, `in2` and `out` contain the signal values at the current sampling instant of u_1, u_2 and y_1 respectively. The array gain contains the values of θ_1 and θ_2.

2.4 Lab tasks

Warning: Never put on a headset or have the loudspeakers turned up at high volumes before you have verified that the code is working as expected. LMS can go unstable and produce very loud and unpleasant output.

a) Start up CCES on the PC. Open the project file SP2-41 which can be downloaded from LMS. The file `gainestimate.c` contains the skeleton function in Appendix **??**. Connect your audio device to audio input 2. Set some music playing on the PC's e.g. using Windows Media Player.

In order to test the lab setup, let the function gainestimate return the values $\hat{\theta} = \begin{bmatrix} 0.0 & 1.0 \end{bmatrix}^T$ regardless of what the variables passed to the function are. Build the project and run it on the board (see the note on the CCES environment). The output to the speakers should switch between the music played on the PC and on your audio device every 10 seconds. Depending on the recording levels you may also hear the other music signal faintly (or not so faintly) in the background.

b) Adjust the volumes of the audio devices such that the signals sent through the speakers appear approximately equally loud in each 10 seconds segments. If you are unsure, ask the lab demonstrator to check that your setup is right. **Note:** If you do change tracks you may have to readjust the volumes.

If for some reason you are unable to adjust the volumes, do the following: Open the file `Process_Data.c`. The last variable declared inside the function `Process_Data` is `amp`. This variable contains the amplification α in equation (**??**), and you can adjust the relative volume of the two signals by changing this variable.

c) Before you implement the estimation algorithms, check the magnitude of the signals, and select the step size in LMS accordingly.

d) Implement and test both LMS and RLS algorithms for estimation of $\theta(t)$. Do LMS first. Once you have got LMS up and running, comment out the code and write the RLS code within the same function. To help you debug your algorithms, the history of values for θ_1 and θ_2 are saved in the global variables `theta1_history` and `theta2_history` respectively. Try different step sizes and forgetting factors. What happens if $\lambda = 1$? What happens if you choose μ too large? Try to find values of λ and μ such that the second audio signal $u_2(t)$ disappears about 5 seconds after a change in gain.

Note: C automatically passes arrays to functions using simulated call by reference. The called functions can modify the element values in the caller's original array. Therefore when you modify the array `gain` in the function `gainestimate` you also modify the value `theta` in `Process_Data`.

A Programming hints and skeleton function

A.1 Skeleton function

```
#include <math.h>

static float insignal1[20], insignal2[20];

void gainestimate(float in1, float in2, float out, float gain[2])
{
    /* check values of in signal by using plot facility */
    /* printf("in1=%f in2=%f\n",in1,in2); */

    /* check 20 values of in signal by using the plot facility*/
    for (i=18; i>0; i--)
    {
        insignal1[i]=insignal1[i-1];
        insignal2[i]=insignal2[i-1];
    }
    insignal1[0]=in1;
    insignal2[0]=in2;
    /*program your lms or rls code here */
}
```

A.2 Programming hints

- Local variables within a function which are declared as `static` retain their values when the function is exited. The next time the function is called the local static variables contain the values they had when the function last exited. The following statement declares local variables `x` and `y` to be static and initialises them to 0.1 and 0.9 respectively.

  ```
  static float x=0.1, y=0.9;
  ```

- The following statement declares a two by two matrix of floats and initialises it to the identity matrix

  ```
  float a[2][2]={ {1.0, 0.0},{0.0, 1.0} };
  ```

  The following declaration makes it static

  ```
  static float a[2][2]={ {1.0, 0.0},{0.0, 1.0} };
  ```