Applied Signal Processing
Hand-In Problem 2 FIR Differentiator Design

December 2, 2019

This problem concerns estimating the speed of an object from noisy position data. The data file (`hip2.mat`) for the problem contains two variables: `true_position` representing the true position and `noisy_position` representing the measured position. The position is the distance, in meters, traveled by a truck, and the sampling interval Δt is 1 second. By definition, the speed (velocity) is the derivative of the position, so a very natural estimate of the velocity from sampled position data is:

$$\hat{v}(n) = (p(n) - p(n-1))/\Delta t\,, \tag{1}$$

where Δt is the sampling interval and $p(n)$ is the sampled position. This is called the Euler approximation of the derivative. From (1) it is clear that the derivative approximation $\hat{v}(n)$ can be seen as the output after FIR filtering with the filter

$$h_{\text{Euler}}(n) = \begin{cases} 1/\Delta t, & n = 0 \\ -1/\Delta t, & n = 1 \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Apply this Euler approximation to the true and measured signals respectively (use `conv` in MATLAB), and plot the result. When plotting the output from the filter disregard the last sample. (The last sample is extremely large compared to the others. We will come back to this later on.) Apparently, differentiation using sampled data is an extremely noise-sensitive task. To improve the estimator we can make use of the fact that the position signal is slowly varying and hence has a low-pass character, whereas the noise is fairly quickly changing and contain all frequencies. Your task is to design a FIR filter that performs approximate differentiation for low frequencies up to frequency 0.05 Hz, while blocking the frequency band above 0.1 Hz to reduce the influence of the noise. The continuous-time filter we wish to approximate has frequency response (why?)

$$H(\omega) = \begin{cases} j\omega & |\omega| \leq 2\pi \cdot 0.05 \text{ rad/s} \\ 0 & |\omega| > 2\pi \cdot 0.1 \text{ rad/s} \end{cases} \tag{3}$$

Thus, the passband has an amplitude response which is linear with frequency and a constant $+90^\circ$ phase shift. The latter is easily fixed by enforcing odd symmetry in the FIR coefficients. Note that the MATLAB function `firpm` can force the filter's phase to be $+90^\circ$ degreas automatically for you, by adding the string `'differentiator'` as the last argument to the function. For more information see the help text, i.e. in MATLAB type `help firpm`.

In MATLAB the convention is that a FIR filter is said to be order N if it has $N + 1$ filter coefficients. Numerical values of frequencies are in MATLAB given normalized relative to half the sampling frequency. Hence, normalized frequency of 1 corresponds half the sampling frequency.

Design task and questions Your task is to design a low-pass FIR differentiator filter which has an amplitude function which deviates less than 0.01 from the desired specification given in (3). The order of the filter should be such that it has 61 filter coefficients. It is advisable to make use of the command `firpm` (see Matlab's documentation for usage instructions). Remember that the width of the transition region strongly influences the ripple magnitude.

See Canvas for the skeleton codebase, and follow the instructions in the code. Remember to include the secret passphrase and the `student_id` parameter you've used at the start of your report.

Questions Once you've developed a filter the passes the self-test function, provide motivated answers for the following questions.

1. Include the code you've written and briefly explain the chosen input arguments to the `firpm` command.

2. Plot the resulting filter coefficients using the `stem` command and plot the magnitude of the frequency function in linear scale. Mark the magnitude plot with the specifications of the ideal filter.

3. The designed filter will introduce a delay. How long is the delay? Explain the reason for the delay.

4. Filter the `true_position` and `noisy_position` signals through the designed filter. Scale the output to obtain the unit km/h. Plot the signals in the same graph. Compensate the plots for the delay introduced by the filter. At the end of the filter output very large oscillations occur. Why? *Hint: test filtering with the signals* `y1 = [noisy_position, fliplr(noisy_position)]` and `y2 = [noisy_position, zeros(1, length(noisy_position)]`. *Why do/don't we get similar behavior?* Make sure your plot shows the useful part of the signal, e.g. using `axis([0 600 0 220])`.

5. Repeat the previous step using the trivial Euler filter defined in (2). Compare and comment on the results.

6. What is the maximum speed of the vehicle found using the observed signal and "true" signal?

Good Luck!