DTU Elektro, Hearing Systems Group
Building 352, DTU, 2800 Lyngby
Phone: 45 25 39 30, Fax: 45 88 05 77   www.hea.elektro.dtu.dk

Course 31606: Signals and Linear Systems in Discrete Time

Cloudy...

Bastian Epp: 45 25 39 53 (office), e-mail: bepp@elektro.dtu.dk

Impulse responses, transfer functions, the power of poles and zeros and block-processing

1 Simple filters

It was quite a step to go from the Fourier transform to the (more general) z-transform. But in the end we realized that this description is helpful to estimate (and in fact calculate) the transfer function. In this part we will have a look at some impulse responses and transfer functions of filters to get some experience with the z-transform. It is very helpful to have some analytical skills in this domain, they will come handy once you are asked to design a digital filter from scratch.

1.1 Impulse responses, transfer functions

Let's go back to the running sum filter of order N with the impulse response:

$$h_{RS}(n) = \begin{cases} 1 & \text{if} \quad 0 \leq n \leq N \\ 0 & \text{if} \quad n < 0 \vee n > N \end{cases}$$

This impulse response is very suitable for the unilateral z-transform since it describes a causal sequence.

- Find the poles and zeros of the filters with order 3 and 5 by calculating the roots of the corresponding polynomial with the `root` function

  ```
  useful commands:  roots
  ```

- Plot the poles, the zeros and the frequency response of the filters using `zplane` and `freqz`

  ```
  useful commands:  zplane, freqz
  ```

- What happens to the frequency response when considering a moving average rather than a running sum? Recall, that the impulse response of the moving average $h_{MA}(n)$ is the impulse response of the running sum normalized by the length of the impulse response $N_0 = N + 1$:

$$h_{MA} = \frac{1}{N_0} \cdot h_{RS}$$

- Some frequencies are heavily suppressed (attenuated) in these filters. Is there a simple way to enhance (that means to amplify) these frequencies rather than to attenuate them?

  By switching the poles and zeros

- Design a filter that blocks the normalized frequency of 0.1 by placing 4 zeros in the z-plane. Make sure the filter returns real-values outputs when fed with real-valued inputs by placing complex-conjugate pairs of poles and zeros.

- Add to the previous filter 4 poles lying on a circle with a radius $R < 1$ (you chose R) at angles identical to the angles of the zeros. How does the frequency response change when you increase R and when R approaches 1?

1.2 The power of poles and zeros

Now that we have some routine with zeros and poles, let's do something fun:

- Place poles and zeros such that you generate a low-pass filter (i.e., passing the low frequencies)
- Place poles and zeros such that you generate a high-pass filter (i.e., passing the high frequencies)
- Design a filter having five poles and zeros that passes all the frequencies (i.e., an all-pass filter)
- Generate a sinusoid at some sampling frequency and pass it through your filters. Plot the original signal and the resulting signal in the time domain. What can you observe?
- Pick some sound file of your choice, process it by the filter and listen to it using `sound` or `soundsc` (watch the volume!)

  ```
  useful commands:  filter, soundsc
  ```

- How do the suppressed frequencies depend on the chosen sampling frequency of the input signal?

How does the processing of the sound change when you use more/less poles/zeros? Can you already estimate the transfer function by looking at the pole-zero plot? Can you say anything about the expected gain and phase shift of single responses?

2 Block-processing

There were times (believe it or not) where memory was incredibly expensive, such signals having a "usual" size could not be processed at once. We discussed in lecture 4 how we can chop a signal into pieces and process it "bite by bite"[1] getting a signal that is identical to as if we were just processing it as a whole. Here you will have some hands-on experience with such a technique to get more familiar with chopping a signal into pieces and processing it block-wise. This technique is very helpful in online processing of incoming signals (i.e., music equipment...) and of course in case you have limited processing resources.

2.1 Bite by bite

Write a function that is suitable to "read" a signal from one device (i.e., a microphone) and to output it to another device (i.e., a loudspeaker). The signal streams will be simulated by block-wise reading from a long vector holding the signal. Use a block size of `blocksize = 1024` in the following task:

- Allocate memory for the output buffer by generating a vector of zeros of the correct size

  ```
  useful commands:  zeros
  ```

- Create another vector holding the boundary conditions at the "edges" of the signal chunks (why is that required?)

[1]...ok...silly joke...

- Write a loop (like the one below) that does the block-processing:
  1. Copy the current block into the input buffer (i.e., a variable called `input_buffer`)
  2. Use the filter command including the boundary conditions
     ```
     >> [output_buffer, boundary] = filter(b,a,input_buffer, boundary)
     ```
  3. Copy the output `output_buffer` into the output array
  4. There might be a leftover at the end of the signal - treat this one separately and ensure the output array is correctly filled
- Test the script by processing a longer sound file (either one of the previous exercises or your favourite piece of music) using the filter coefficients of a running sum filter
- Double-check your algorithm by filtering the whole signal in one piece and compare it to the block-processed signal. If the implementation is correct, there should be literally no difference (as we found analytically in the lecture)

3 Making the room IIR

In a previous hands-on you checked what happens when we model a room with sound reflections at the walls using a simple delay. This was done by specifying the non-recursive filter coefficients by:

```
b = [1; zeros(delay_samples,1); alpha];
```

This gave us a simple echo delayed by a specified value. This is pretty artificial since in a closed room there will be sound bouncing back and forth between the walls, resulting in multiple echoes. A more realistic situation is that the sound is bouncing back and forth between the walls, being attenuated by the same amount each time hitting a wall. This is a perfect situation for the application of a recursive filter.

Linking physical problems to concepts in signal processing is the key to success - DSP is more than abstract math!

3.1 Bounce bounce ...

Implement a room situation including reflections using a recursive filter. The sound travels to the wall (i.e., is delayed), reflects after being attenuated, bounces off the next wall...

Create the following filters and show the impulse response and the transfer function (gain and phase on a suitable axis) - can you find the filter coefficients in the plot? In what domain?? frequency

- A FIR filter based on the difference equation using a delay of 30 ms and an attenuation coefficient of $\alpha = 0.6$:
$$y(n) = x(n) + \alpha x(n - delay)$$
- An IIR filter based on the difference equation:
$$y(n) + \alpha y(n - delay) = x(n)$$ look at page 106!!!!

Use an impulse and some speech signal (e.g., mini-me.wav) to excite the filters and see/hear if you can observe a difference.

- Vary the delay to values of 10 ms and 200 ms and describe what you observe in the impulse response, the transfer function and in the processed signal.

Note: This is an implementation of a filter applied to a specific digital signal, so please provide the correct axes (correct frequencies, not normalized ones).