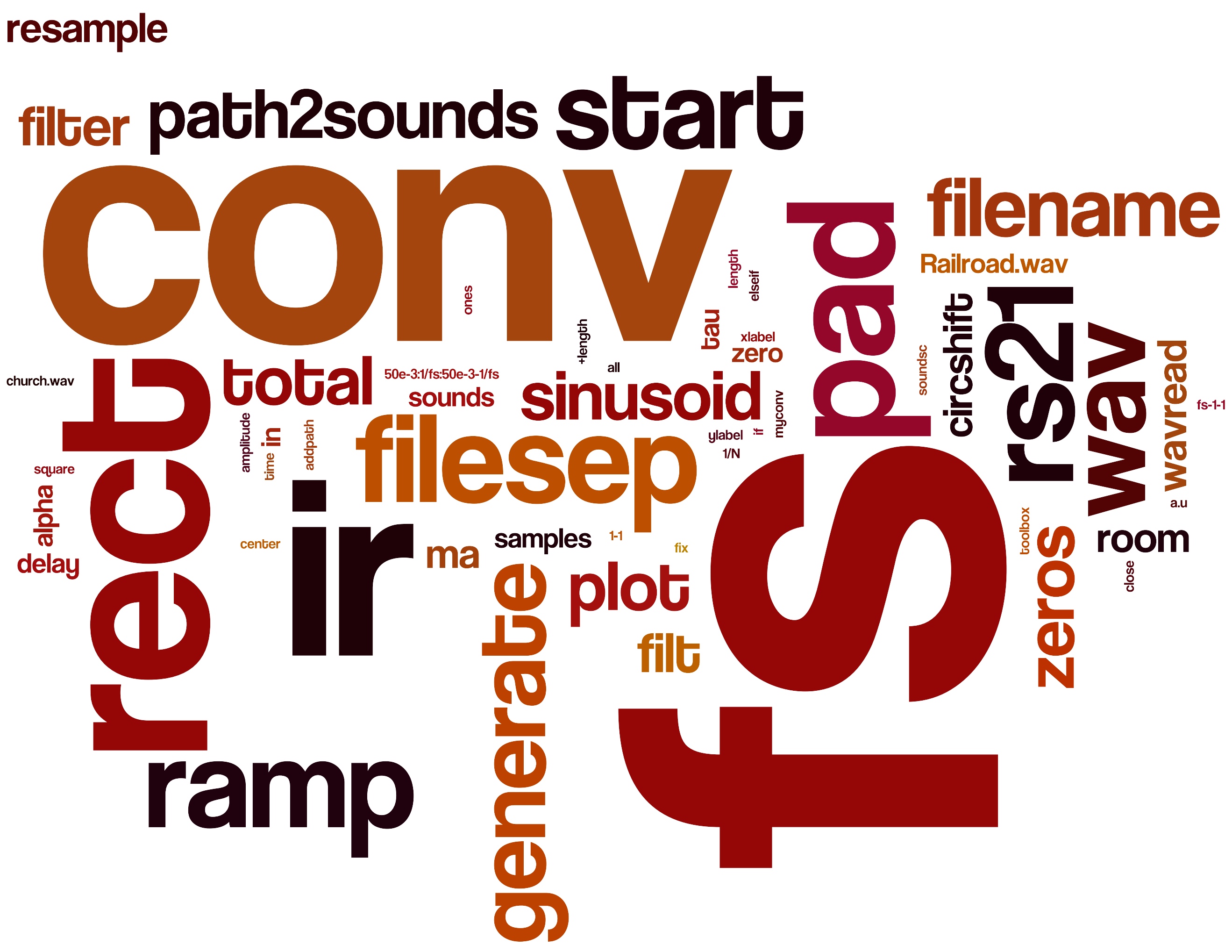
Cloudy...


resample
filter
path2sounds
start
conv
filename
Railroad.wav
pad
length
elseif
ones
+length
tau
xlabel
zero
all
church.wav
rect
total
50e-3:1/fs:50e-3-1/fs
sounds
sinusoid
soundsc
circshift
rs21
wav
wavread
fs-1-1
amplitude
time
in
addpath
filesep
ylabel
if
myconv
1/N
square
alpha
delay
center
ir
ma
samples
1-1
fix
fs
a.u
toolbox
zeros
room
close
plot
filt
generate
ramp

Convolution, filtering and simple linear systems

1 Convolution

Here we look at what the mathematical operation of a convolution does to simple signals. This concepts plays an important role, and having a feeling for what a convolution does, is a very helpful tool in order to understand more complex signal processing schemes. In the screencast we also discussed (if that could qualify as discussion) the moving average. We could see that application of a moving average (or running sum) leads to a smoothing of the signal. This seems to be somehow connected to the transfer function of the moving average. In this part we will also see what happens to the data when interpreting the running sum as an impulse response of a filter.

1.1 Toolbox - Simple convolution

Let us start with looking at what a simple convolution does to signals. Generate the three signals below [1] with a suitable sampling frequency and a total duration of $T_{tot} = 0.1\ s$. If the signal is shorter than the total duration, add zeros to match T_{tot} (this is called zero-padding - check the total number of samples you get in the end). Write separate functions for each signal (e.g., `generate_square(), generate_ramp(), zero_pad()`):

- A normalized square-wave signal (sometimes called 'boxcar-function') with $T_0 = 10$ ms

$$\mathrm{rect}_{T_0}(t) = \begin{cases} \frac{1}{T_0} & \text{if} \quad 0 \leq t \leq T_0 \\ 0 & \text{if} \quad t > T_0 \end{cases} \tag{1.1}$$

  `MATLAB: ones(N,M), zeros(N,M)`

- A linear ramp of duration $T_0 = 10$ ms

$$\mathrm{lin}_{T_0}(t) = \begin{cases} t & \text{if} \quad 0 \leq t \leq T_0 \\ 0 & \text{if} \quad t > T_0 \end{cases} \tag{1.2}$$

Think about which sampling rate to choose - it would be good to end up with integer numbers of samples...

Use the signals generated above to perform the following signal operations:

- $\mathrm{rect}_{T_0}(\mathrm{t}) * \mathrm{rect}_{T_0}(\mathrm{t})$
- $\mathrm{lin}_{T_0}(\mathrm{t}) * \mathrm{rect}_{T_0}(\mathrm{t})$
- $\mathrm{lin}_{T_0}(\mathrm{t}) * \mathrm{lin}_{T_0}(\mathrm{t})$

Plot the resulting signals and explain what you see. How long are the resulting signals?

`useful commands:  conv, length`

[1] Recall the previous hands-on: When generating a digital signal, you normally start with generating the time vector representing the points in time where the signal is being sampled. Then you assign the desired values at these samples - done!

Optional: Convolution vs. correlation

Mathematically very close to a convolution is the operation of a 'cross correlation' (Matlab command `xcorr(signal_1, signal_2)`). This operation is often used to identify similarities between two signals. Try `xcorr` with the same combinations as above. Can you figure out what the difference is between `xcorr` and `conv`?

1.2 Filtering

This section is about getting some feeling for what happens when a signal is being filtered. We'll start with a MATLAB example:

- Generate an impulse response of an order 21 moving average with a sampling frequency of 10 kHz and a time vector that goes from $t = [0 : 100]$ms
- Generate three sinusoids with frequencies of 500 Hz, 2200 Hz and 4050 Hz with a duration of 1 s
- Plot the unfiltered signals and the impulse response on top of each other with different colours (select a proper part of the abscissa to be able to see all signals)

Now we are good to do some filtering:

- Filter the signals with the impulse response (using a convolution) and plot the original and the filtered signal on top of each other
- What happens at the beginning of the filtered signal?
- How do the amplitudes of the different filtered signal vary?
- Try to find the frequency with the highest attenutation (i.e. lowest amplitude after filtering)

Now - this was the "hands-on" way of doing it. Matlab actually provides a function that is slightly more efficient for filtering, namely the `filter(B,A,signal)` function. In order to use the filter function, you only need the "non-zero" samples of the moving average. Filter the sinusoids using only the 22 samples that are non-zero (i.e., `h=1/22*[ones(22,1)]`). Can you see differences between the convolution operation and the filter operation?

Now let's do it by hand!

- Calculate analytically the result of the convolution (i.e. filtering) of a sinusoid with frequency $f = 10$ Hz, sampled with a sampling rate of $f_s = 50$ Hz through a running sum filter of order 5. Both signals are causal, i.e., are zero for $k < 0$. As a reminder, the convolution sum is given by:

$$y[k] = \sum_{u=-\infty}^{\infty} f[u] \cdot h[k-u] \tag{1.3}$$

- Double-check your result with MATLAB (re-use the code above)
- Show graphically why the result makes sense
- Calculate all frequencies that are blocked through the running sum / moving average filters used above

2 Simple linear systems

Moving on to linear systems. Linear systems might serve as a nice framework to work with (and to understand) certain physical phenomena. Since linear system theory is well developed, this application is helpful in many fields. Here we look at an application from the acoustical world: reverberation. Reverberation of a room describes the effect that a sound field emitted by a sound source is being reflected at the walls of a room. It is essentially that same what happens when you hear multiple copies of a sound for example during construction of the Copenhagen City Ring. It might be useful to write a function that takes the sound file as an input and saves a new .wav file somewhere where you can find it.

2.1 Room reverberation

A simple model for room reverberation is the sum of a signal and an attenuated, delayed version of the same signal:

$$s_{rev}(t) = s(t) + \alpha s(t-\tau) \quad , \alpha > 0 \tag{2.1}$$

Where α denotes the attenuation of the reflected signal and τ the time delay. To be physically feasible, it makes sense to restrict the values of α to [0,1[. Generate a room impulse response with one reflection:

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

- Use the different signals (e.g., the ones of the hands-on of lecture 1) and filter them with the room impulse response using different values for the delay τ and the attenuation α
- Look at the time signals and listen to them
- Use the file spoken_sentence.wav. Apply different delays and amplitudes and see how well you can understand what is being said

  ```
  useful commands:  resample
  ```

Obviously, the single reflection is a very simplified model of a room. Generate a new impulse response and add multiple reflections. How do the resulting sounds change?

2.2 Putting sounds into different places

As the name suggests, a room impulse response is normally the response of a room due to excitation with an impulse. Since these are rather simple to record, one can model these rooms as systems with a certain impulse response. Use the following impulse responses and apply them to some signals:

- Grundtvigs_church.wav
- church.wav
- large_hall.wav
- room.wav

Convolution (or filtering) signal with these impulse responses will simulate the presence of these signals in these rooms. If you have a mic, you can easily record your own impulse responses. It doesn't need to be a room, it could be a table, a trash bin, a bottle...