A. Signal analysis exercises (KB, JCD, AS)

Part I: Using Matlab and Cool Edit / Adobe Audition

The following exercises are most likely much more than what you will have time for. This is supposed to be so. Some of the examples may also be useful in your future career.
Time does not permit us to give an thorough introduction to MatLab here. However, most of you probably have encountered MatLab already or will do so during your study. Note that MatLab has an extensive, built-in help function, where all commands can be looked up. Try for instance

```
help fft
```

Spectra and spectrograms in MatLab
MatLab commands are typed in the Command window (MatLab is case sensitive – procedures and functions are lower-case). All variables are viewed as matrices, so in principle a scalar is a 1x1 matrix.

The following MatLab command loads the samples contained in a *.wav file into a column array, s (a number of samples x 1 matrix).

```
[s sr]=wavread('your filename');
```

The sample rate, here named "sr" is an optional output parameter. Use ";" at the end of an expression to suppress the display of the result in the command window (important with long wave files).

The command

```
plot([0:length(s)-1]/sr,s)

creates a plot of the signal.
```

The following command plots the absolute values in the discrete Fourier transform of the signal against frequency (i.e. the amplitude spectrum of the signal):

```
plot([0:length(s)-1]/length(s)*sr,abs(fft(s)))
```

If you only want to display frequencies up to sr/2 (the Nyquist frequency) you may now type:

```
axis([0 sr/2 0 inf])
```

Read the vector parameter passed to the function "axis" this way: [min(x) max(x) min(y) max(y)]. The "inf" parameter sets the y-axis to extend up to the maximal value in the data.

Sometimes it is practical to store the Fourier data in an array of its own, for instance if you wish to do further manipulations of it. In that case we could skip the axis adjustment by writing:

```
L=length(s);
four_s=abs(fft(s));
plot([0:L/2-1]/L*sr,four_s(1:L/2))
```

or, using dB units:

```
plot([0:L/2-1]/L*sr,20*log10(fours(1:L/2)))
```

Note that indices start with 1, not 0. Note also that [] are used for giving matrix values, while () are used to reference them. Quick example:
`a=[0 1 2 5];` defines a horizontal 4 element array. Entering `a(0)` produces an error, while `a(1)` returns 0. `a(1:2)` returns `0 1`.

To produce a spectrogram of s, the following command can be used:

```
specgram(s,512,sr,16,12)
```

This creates a nice color coded display. The parameters are read as follows:

specgram(signal, twice the number of spectral lines, sampling rate, length of signal chunk, overlap (in samples)). The choices in the example above may not be appropriate.

The overlap must be smaller than the chunk length and the chunk length must be smaller than or equal to twice the number of spectral lines.

Scripts in MatLab
A MatLab script is a text file containing a series of commands that can be run again and again without retyping anything. This is very practical and you have a strong desire to start making these right away. Choose File → New → M-file. From the now open editor window choose File → save as and a name of your liking, eg. "cosxample"

Define an array containing a cosine signal, sc1, by typing the following in the edit window:

```
LC=128;  % length of array.
           %any text appearing after an "%" is ignored by
           %MatLab and is used to make comments. Like this one.
           %Don't type these comments!
f=4;       %the number of cycles in the signal
sc1=cos(f*2*pi*[0:LC-1]/LC);
fourscl=abs(fft(sc1));
plot([0:LC/2-1],fourscl(1:LC/2))
```

When you are done, choose Debug → Run.
If it doesn't work you will get an error message in the command window. Warnings can be ignored. When the script works, change the value of f to 4.5 and run it again. Try to explain the observations.

Try introducing the following changes to the plot line:

```
subplot(2,1,1)
plot([0:LC-1],sc1)
subplot(2,1,2)
plot([0:LC/2-1],fourscl(1:LC/2))
```

You can make a new graph window (leaving the old one(s) alive) by typing

```
Figure
```

Now, save your work under a new file name and modify it to produce a spectrogram.
One example could be:

```
specgram(sc1,512,1,128,120)
```

Try changing the spectrogram parameters. What do you learn about the effects of each of the parameters (hint: make LARGE changes, you can't ruin anything important).

Signal displays in Cool Edit
Cool Edit or Adobe Audition offers two views, time series and spectrogram. Open the file "Tonep22.wav". Look at the spectrogram display first. Measure the duration of the tone pips. Compare the results to what you find in the time series display. Explain the results.[j1]

Filtering in Cool Edit
Zoom to full view in the spectrogram display mode. Select Effects → Filters → FFT filters. This brings up a graphical filtering tool. Create a "brick filter" that removes all frequency components below 2000 Hz and above 3000 Hz. Observe the effect on the spectrogram. Switch to time series display. Now measure the durations again. Explain the findings.

Noise analysis in Cool Edit and using MatLab
Load the file 'pinknoise-44k.wav' into an array in MatLab. Use a script, so you can quickly make changes. Create an amplitude spectrum of this noise signal. Can you justify from looking at this plot that this is "pink" noise, i.e. noise with a power density that is inversely related to frequency? Maybe you can, but try using one or both of these methods instead: 1) Make averages of the amplitude at several frequencies and display the result, or 2) divide the signal into chunks and display the average amplitude of the Fourier transform of each chunk. An example of a solution to both methods:

```
[n sr]=wavread('C:\My Documents\Vorträge\ss06\pinknoise-44k.wav');
sr=sr/1000; %to get it in kHz

subplot(3,1,1) %three panels in one figure, this is the first
four_n=abs(fft(n));  %good old Fourier
plot([0:length(n)/2-1]/length(n)*sr,four_n(1:length(n)/2))
axis([0 sr/2 0 inf])

M=256; %length of chunks
for j=1:floor(length(n)/M) %loops look like this in MatLab
    noise_matrix([1:M],j)=abs(fft(n(j*M-M+1:j*M))); %The chunk solution
    mean_amp(j)=mean(four_n(j*M-M+1:j*M));   %The average freq. bin method
end;                  %loops always end with an end

subplot(3,1,2) %next panel
plot([0:M-1]/M*sr,mean(noise_matrix,2))
    %here "mean" makes a horizontal (dimension=2) average
axis([0 sr/2 0 inf])

subplot(3,1,3) %note that j still has the last value from the loop
plot([0:j/2-1]/j*sr,mean_amp(1:j/2))
axis([0 sr/2 0 inf])
```

Isn't it easier to justify now? Try manipulating the length, M. Does it affect the similarity between the methods?

Zero padding in MatLab

In MatLab, make a script that opens the wave file "risso_clicks_hoj_trunc.wav". E.g:

```
[rd sr]=wavread('C:\My Documents\ss06\risso_clicks_hoj_trunc.wav');
          %this will look differently on your computer
```

Display it in the time domain and in the frequency domain. You can modify your previously made scripts and save under a new name. To see what happens to the amplitude spectrum when you increase the length of a signal, type the following (possibly leaving out the comments. You could also use other variable names, if you don't like these). The script changes the length of the signal by adding zeros at the end (zero-padding).

```
clf   %clears content of the active figure window
four_rd=abs(fft(rd)); %create amplitude spectrum
OL=length(rd); %Create variable of original length of sound file
plot([0:OL/2-1]/OL*sr,four_rd(1:OL/2)) %make plot
hold on    % now old traces are no
           % longer erased before drawing new ones

NL=1024; % new length of signal
rdpad(NL)=0; % sets the length of the array to NL, inserting zeros;
four_rdpad=abs(fft(rdpad)); % spectrum of the longer signal
plot([0:NL-1]/NL*sr,four_adpad(1:NL/2),'r')
           %the 'r' parameter at the end makes the trace red
```

In the spectrograms of both Matlab and Cool Edit this is the way the number of spectral lines is increased beyond half the length of the Fourier transform.

Interpolation

The lesson from the last exercise is that zero-padding a signal before Fourier transforming it creates an interpolation between the points in the original spectrum. Because the Fourier transform is almost its own inverse transform, we may guess that putting zeros at the end of the complex spectrum results in an interpolation of the time domain signal. The thing that stands in our way is the frequency components above the Nyquist frequency. If we remove these, we get a time domain signal that is complex, just as one-sided time domain signals results in complex spectra. In Matlab we can get rid of the imaginary part of the complex signal by simply writing `s=real(ifft(S))`. So the following is a recipe for interpolation of the signal "rd" from the example above:

```
clear %clear all variables
[rd sr]=wavread('C:\My Documents\ss06\risso_clicks_hoj_trunc.wav');
four_rd=fft(rd); %calculate complex spectrum
four_rd(length(rd)/2+1:1024)=0; %ditch negative frequencies
          %and make 1024 samples long
ip_rd=2*1024/length(rd)*real(ifft(four_rd));
plot([1:1024]/(sr*1024/length(rd)),ip_rd)

In this example there is some necessary scaling going on. Also, when removing
the negative frequency components, you reduce the overall amplitude of the
signal by a factor of to. To get it back the real part of the corresponding time
domain signal is amplified by the same factor.
```

It is also possible to reduce the sample rate in a similar way. If you have the time you may try this.

Part II. Use of Sound Analysis Programs – Cool Edit / BatSound

The following is intended for the signal analysis program "BatSound", but similar operations are relevant for all signal analysis programs, i.e. Cool Edit/Adobe Audition.

Signals and suggestions for signal analysis practical.
Version 3.0. Date: 6-8-2003
Authors: Simon Boel-Pedersen, **AS, JCD**.

1. Introduction.

The following signals have been chosen to facilitate a deeper understanding of the signal processing of the spectrograph program BatSound. Some of the signals are computed mathematically and the - known - properties are listed below. With these signals it may be possible to compare the analysis results with that of the signal analysis theory. Other signals are real life signals where one must cope with the uncertainty that is part of the process of analyzing - unknown - signals.
We recommend that you use the dual, i.e. time signal and spectrogram display in BatSound. For later presentation it is recommended to copy the time and spectrographic displays and paste them in PowerPoint.

2. Description of signals.

All signals are present as mono WAV-files with a standardised header.

2.1 TONEP22.WAV.

The signal consists of 10 tone pulses with frequency 2.756 kHz sampled at 22.05 kHz with the following data:
Duration: 92.9 msec (2048 samples)
23.2 msec (512 samples)
5.8 msec (128 samples)
2.9 msec (64 samples)
1.45 msec (32 samples)
92.9 msec (2048 samples)
23.2 msec (512 samples)
5.8 msec (128 samples)
2.9 msec (64 samples)
1.45 msec (32 samples).
Constant amplitude: First 5 pulses: 16376
Last 5 pulses: 2047 (-18 dB re the first group).
Interpulse pause: 46.4 msec (1024 samples).

Suggestions for analysis.

1. Adjust the Threshold and the Amplitude Contrast to make the display look reasonable and so that all 10 pulses are visible on the spectrographic display.
2. Compare the duration of the pulses with the "duration" in the corresponding spectrographic display.

3. Repeat 1. and 2. with different values of the FFT size and observe the differences between the spectrographic displays (e. g. duration, duration of the transient phenomena at the start and at the end of each pulse, the displayed amplitude in the middle of the pulse at the frequency corresponding to the frequency of the pulse). Try also different types of windows functions.

2.2 TONEP22N14A.WAV.

The signal consists of TONEP22.WAV with added noise within the frequency range 1 - 4 kHz. The signal to noise ratio (SNR) for the first 5 pulses is approx. 25 dB, for the last 5 signals approx. 7 dB.

Suggestions for analysis.

1. Adjust the Threshold and the Amplitude Contrast to make the display look reasonable and so that all 10 pulses are visible on the spectrographic display (if possible).
2. Compare this result with that of TONEP22.WAV.
3. Repeat 1. and 2. with different values of the transform size and observe the differences between the spectrographic displays.

2.3 SWEEP4X.WAV.

The signal consists of 4 linear frequency sweeps from 3 kHz to 7 kHz sampled at 20 kHz.
Duration: 102.4 msec (2048 samples)
51.2 msec (1024 samples)
25.6 msec (512 samples)
12.8 msec (256 samples)
Constant amplitude: 1800.
Interpulse pause: 102.4 msec (2048 samples).

Suggestions for analysis.

1. Adjust the Threshold and the Amplitude Contrast to make the display look reasonable and so that all 4 pulses are visible on the spectrographic display.
2. Compare the "width" of the different pulses and their amplitude on the spectrographic display.
3. Repeat 2. for different values of the transform size.
4. Make power spectra with different types of window functions. Set the cursor right before onset of the longest sweep and adjust FFT size to include the sweep.

2.4 SWEEP6DB.WAV.

The signal consists of 15 linear frequency sweeps from 2 kHz to 8 kHz sampled at 22.05 kHz. The duration of each sweep is 30 msec (661 samples).

Amplitudes: 32767
16384 (-6 dB re 32767)
8192 (-12 dB re 32767)
4096 (-18 dB re 32767)
2048 (-24 dB re 32767)
1024 (-30 dB re 32767)

512 (-36 dB re 32767)
256 (-42 dB re 32767)
128 (-48 dB re 32767)
64 (-54 dB re 32767)
32 (-60 dB re 32767)
16 (-66 dB re 32767)
8 (-72 dB re 32767)
4 (-78 dB re 32767)
32767

Interpulse pause: 46.4 msec (1024 samples).

Suggestions for analysis.

1. Adjust the Threshold and the Amplitude Contrast to make the display look reasonable and so that all 15 pulses are visible on the spectrographic display (if possible) and observe the frequency range occupied by the pulses.
2. Repeat 1. for different values of the transform size.

2.5 OS11.WAV.

The signal consists of 1 sound from a longer Blackbird utterance. Sampling frequency is 11.025 kHz and the signal has a duration of approx. 200 msec. It is the same signal that have been used for construction of one display line in the presentation of Spectrum Analysis with Filters.

Suggestions for analysis.

1. Change the transform size and adjust the Threshold and the Amplitude Contrast to make one line in the display look similar to the "ideal" display line shown during the presentation (if possible).
2. Repeat 1. for different values of the overlap.

2.6 WREN20.WAV.

The signal consists of a natural sequence of sounds from a Wren sampled at 20 kHz (recorded by Jo Holland). It contains a lot of different signal types ranging from almost constant frequency parts to very rapid frequency sweeps. Apply the knowledge obtained from the signals above in the analysis of this signal or of selected parts of it.

2.7 LHAS22.WAV.

The signal consists of an advertisement call of the South-East Asian frog *Leptobrachium hasseltii* sampled at 22.05 kHz. The recording is contaminated by noise of biological origin (insect sounds) and abiotic origin (wind etc.). Display the signal and try to remove 'unwanted' components by filtering.

2.8 SUMATRANRHINO.MM1.WAV(optional)

The signal is the 'eep'call from a Sumatran Rhinoceros. The recording was published recently (Acoust Res Let Online 4: 83-88, 2003). Display and analyze the signal and compare it to the published sonogram (fig 1).