Lecture 3: Rhythm

3B: Rhythm Formant Theory and Analysis

Dafydd Gibbon
Bielefeld University, Germany
2022-04-29

II Brazilian Congress of Prosody
Minicourse 9: 25, 27, 29 April 2022
(09:00-11:30 Brazilian Standard Time)

Lecture 3: Speech Melody

3B: Rhythm Formant Analysis

Dafydd Gibbon
Bielefeld University, Germany
2022-04-29

II Brazilian Congress of Prosody
Minicourse 9: 25, 27, 29 April 2022
(09:00-11:30 Brazilian Standard Time)

Rhythm Formant Theory and Analysis

Rhythm Formant Theory (RFT):

- A rhythm formant is a frequency zone of higher magnitude values in the normalised low frequency (LF) spectrum.
- Rhythm formants are detected both in the LF AM spectrum and also in the LF FM spectrum.

Rhythm Formant Analysis (RFA):

- The spectrum frequencies and their magnitudes are obtained by FFT and the magnitudes are normalised to the range 0,…,1.
- A minimum magnitude (e.g. about 0.2) is defined as a cutoff level, below which values are clipped to zero; only the higher values are retained.
- The clipped spectra of different recordings are compared using standard distance metrics and represented as distance maps, and hierarchically clustered using standard clustering criteria and represented as dendrograms.

Thanks to Laura, Dr. Liue Huangmei, for the term ‘formant’ in this context.

Code

The code is at

https://www.github.com/dafyddg/RFA

The main directory of this GitHub repository contains the following directories (**bold**) and files:

Articles	*articles on RFT and RFA*
IICBP2022-slides	*slides for Brazilian Phonetics minicourse 2022*
LittleHelpers	*small RFA demo scripts and data*
README.1st	*documentation*
RFA_multiple_signal_processing	*scripts for multiple file processing and cluster analysis*
RFA_multiple_signal_processing.zip	
RFA_single_signal_processing	*script and modules for single file analysis*
RFA_single_signal_processing.zip	

Aims of this talk

Overview of Rhythm Formants as low frequency modulations of speech

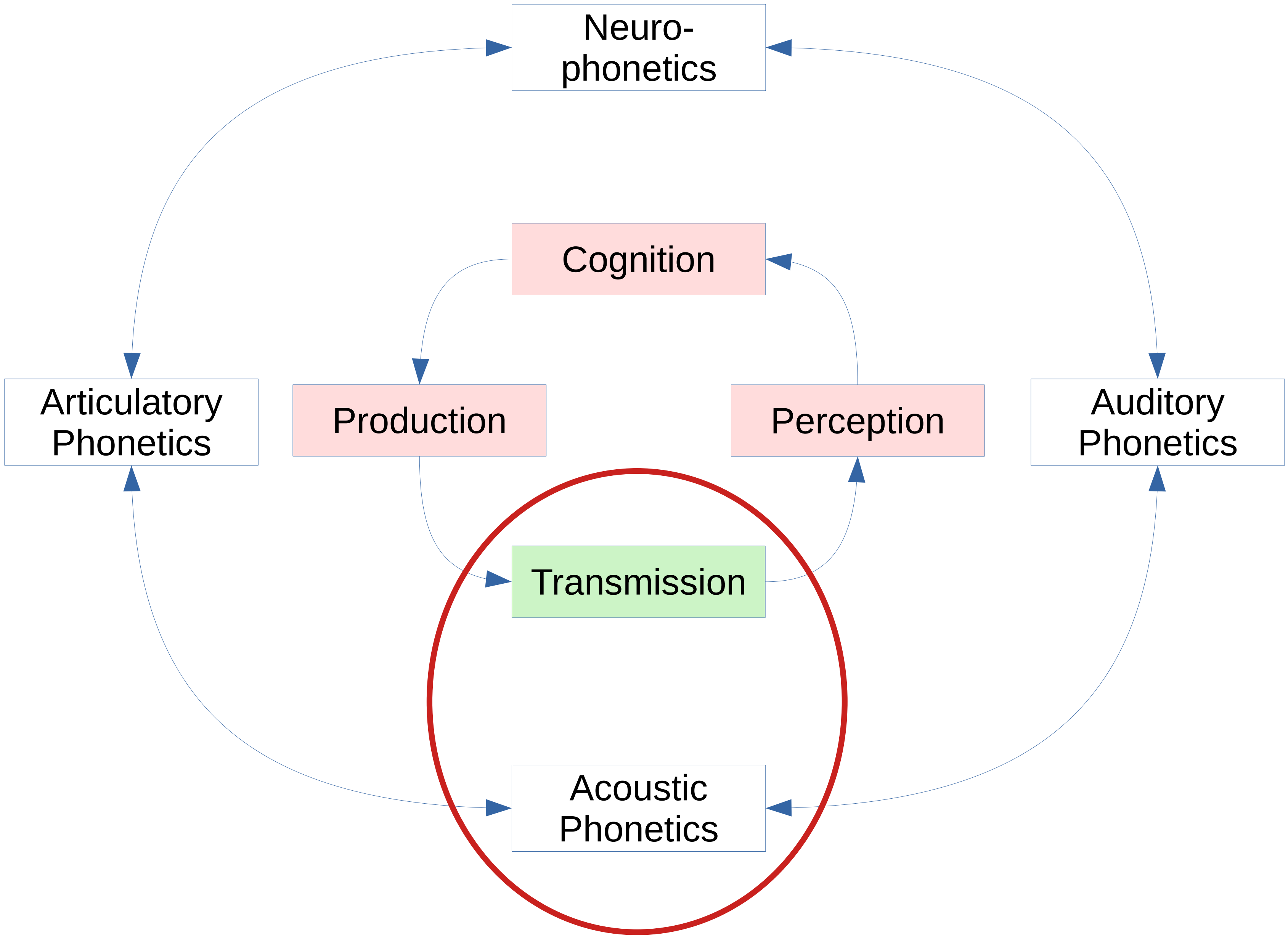
Demonstration of how my software (also Praat etc.) does

- AM and FM demodulation
- spectral analysis
- comparing spectra from different recordings of comparable data using distance tables, distance maps and distance based clustering

- Why?
  - If you’re a driver, it makes sense to know how a car works in practice.
  - If you’re a phonetician, it makes sense to know how ‘pitch’ extraction, spectral analysis, distance maps and clustering etc. work in practice.

Empirical Background: Phonetic Domain, Phase Cycle

Empirical Background: Phonetic Domains and Methods



Overview

- Production and perception phases of prosodic events are well known in phonetics:
  - source-filter theory: larynx as source, oral & nasal cavity as filter
  - cochlea transformation theory: extraction of signal frequencies
- Transmission theory is usually left to the audio engineers:
  - In this talk:
    - Modulation Theory:
      - Amplitude Modulation (AM)
      - Frequency Modulation (FM)
    - a 'do-it-yourself' approach to phonetic software
      - an alternative to using ready-made off-the-shelf applications
    - you can download demonstration examples in Python
    - BUT: <u>no programming experience is required</u>

http://wwwhomes.uni-bielefeld.de/gibbon/Lectures/SummerSchool2021-Gibbon/

Overview

- Production and perception phases of prosodic events are well known in phonetics:
  - source-filter theory: larynx as source, oral & nasal cavity as filter
  - cochlea transformation theory: extraction of signal frequencies
- Transmission theory is usually left to the audio engineers:
  - In this talk:
    - Modulation Theory:
      - Amplitude Modulation (AM)
      - Frequency Modulation (FM)
    - and a 'do-it-yourself' approach to phonetic software
      - an alternative to using ready-made off-the-shelf applications
    - You can download demonstration examples in Python
    - BUT: no programming experience is required

http://wwwhomes.uni-bielefeld.de/gibbon/Lectures/SummerSchool2021-Gibbon/

Overview

- Production and perception phases of prosodic events are well known in phonetics:
  - source-filter theory: larynx as source, oral & nasal cavity as filter
  - cochlea transformation theory: extraction of signal frequencies
- Transmission theory is usually left to the audio engineers:
  - In this talk:
    - Modulation Theory:
      - Amplitude Modulation (AM)
      - Frequency Modulation (FM)
    - a 'do-it-yourself' approach to phonetic software
      - an alternative to using ready-made off-the-shelf applications
    - you can download demonstration examples in Python
    - BUT: <u>no programming experience is required</u>

http://wwwhomes.uni-bielefeld.de/gibbon/Lectures/SummerSchool2021-Gibbon/

Rhythm Formants

Rhythm Formant Theory (RFT):

- A rhythm formant is a frequency zone of higher magnitude values in the normalised low frequency (LF) spectrum.
- Rhythm formants are detected in the LF AM spectrum and in the LF FM spectrum.

Rhythm Formant Analysis (RFA):

- The spectrum magnitudes are obtained by FFT and normalised to the magnitude range 0,…,1.
- A minimum magnitude (e.g. about 0.2) is defined as a cutoff level, below which values are clipped to zero.
- The clipped spectra of different recordings are compared using standard distance metrics and represented as distance maps, and hierarchically clustered using standard clustering criteria and represented as dendrograms.

Modulation Theory

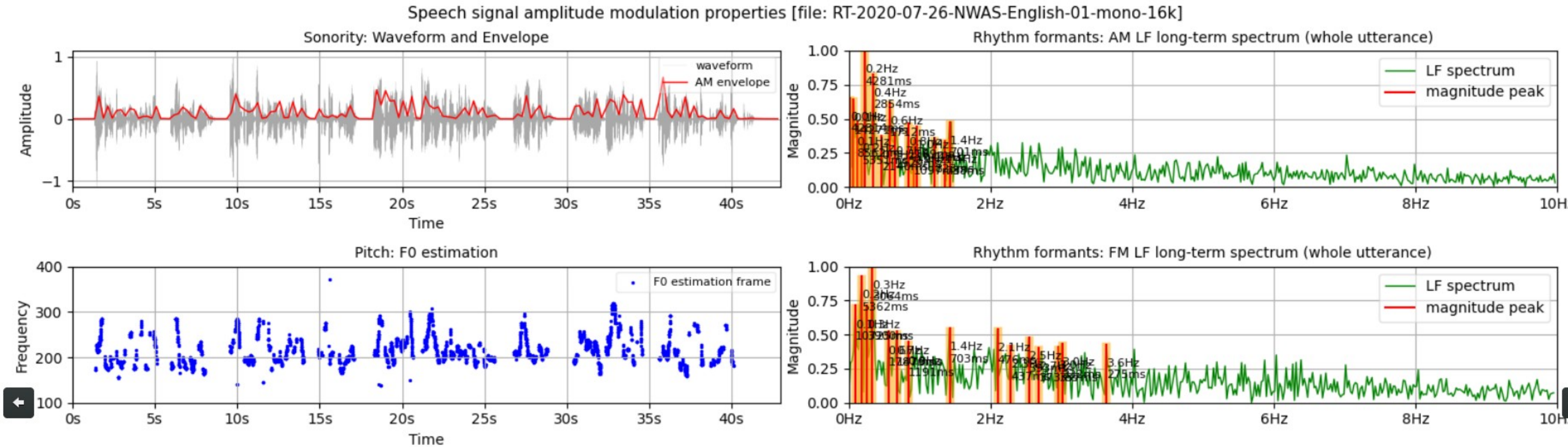
Demodulation and analysis procedures

- Time domain procedures:
  - Envelope extraction
  - Fundamental frequency estimation (‘pitch’ extraction)
- Frequency domain procedures:
  - Spectral analysis
  - Spectrogram analysis
  - there are also frequency domain procedures for F0 estimation
- Comparison using distance metrics
  - distance calculation with different distance metrics
  - hierarchical clustering with distance and different clustering criteria
- Output:
  - Graphical display
  - Numerical files and figure files

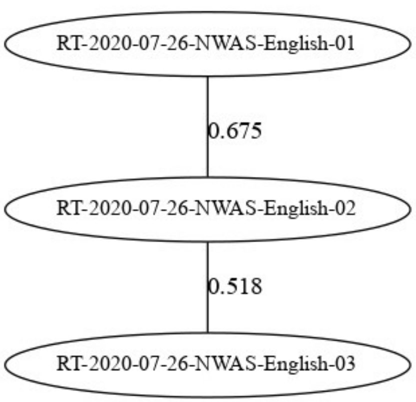
Demodulation and analysis: output examples

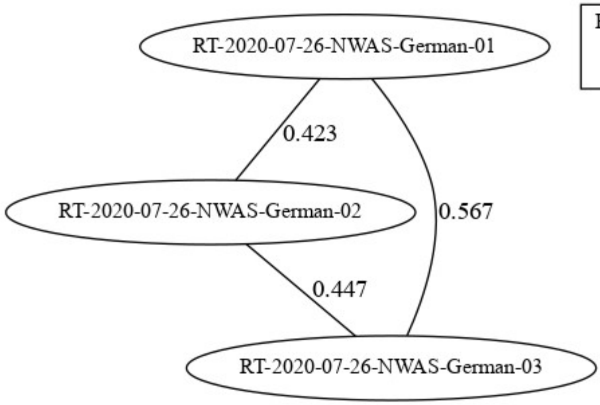
Example outputs

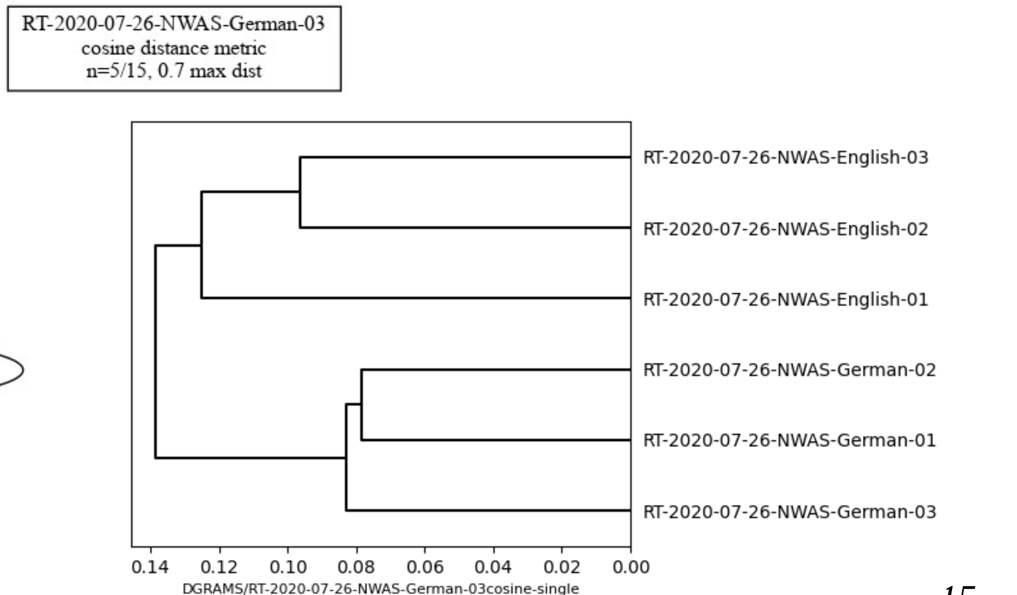
Story "The North Wind and the Sun", read by an adult female German-English bilingual



Similarity of readings: *The North Wind and the Sun*, bilingual in English and German

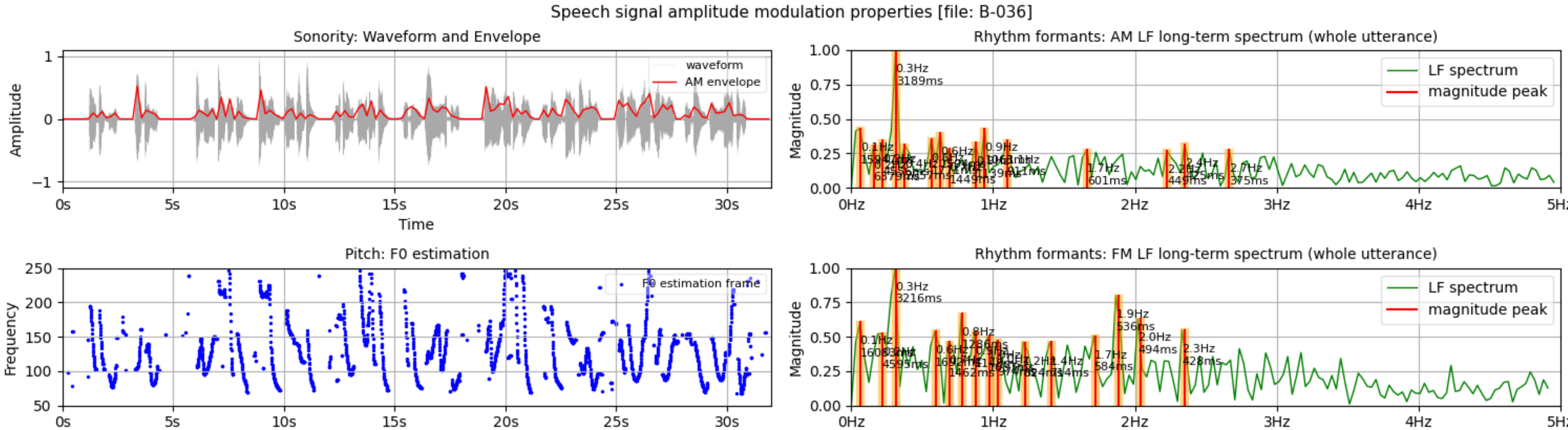






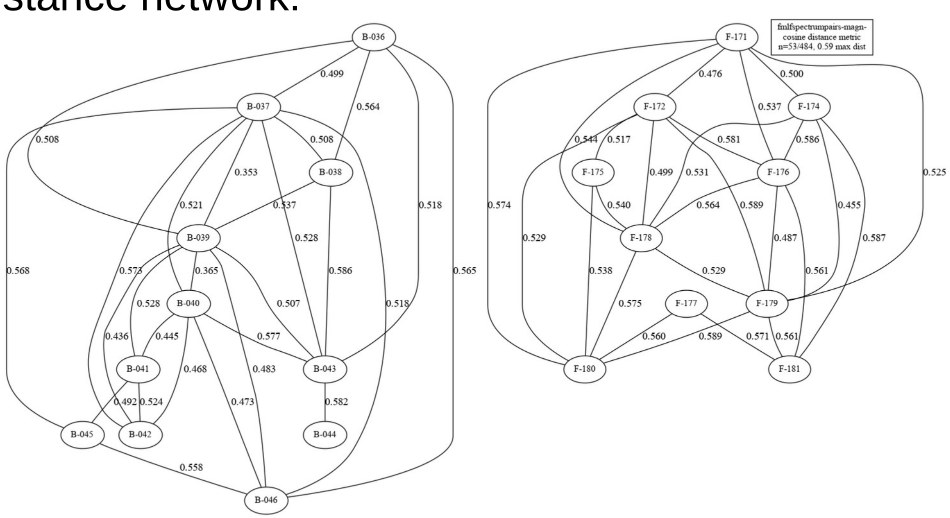
Example outputs

Poem recitation: B-036 塞上曲 [王昌龄]-mono-16k

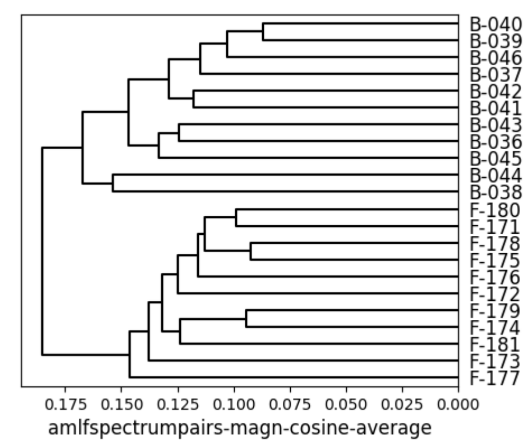


Comparing two styles of Tang dynasty poetry

Distance network:

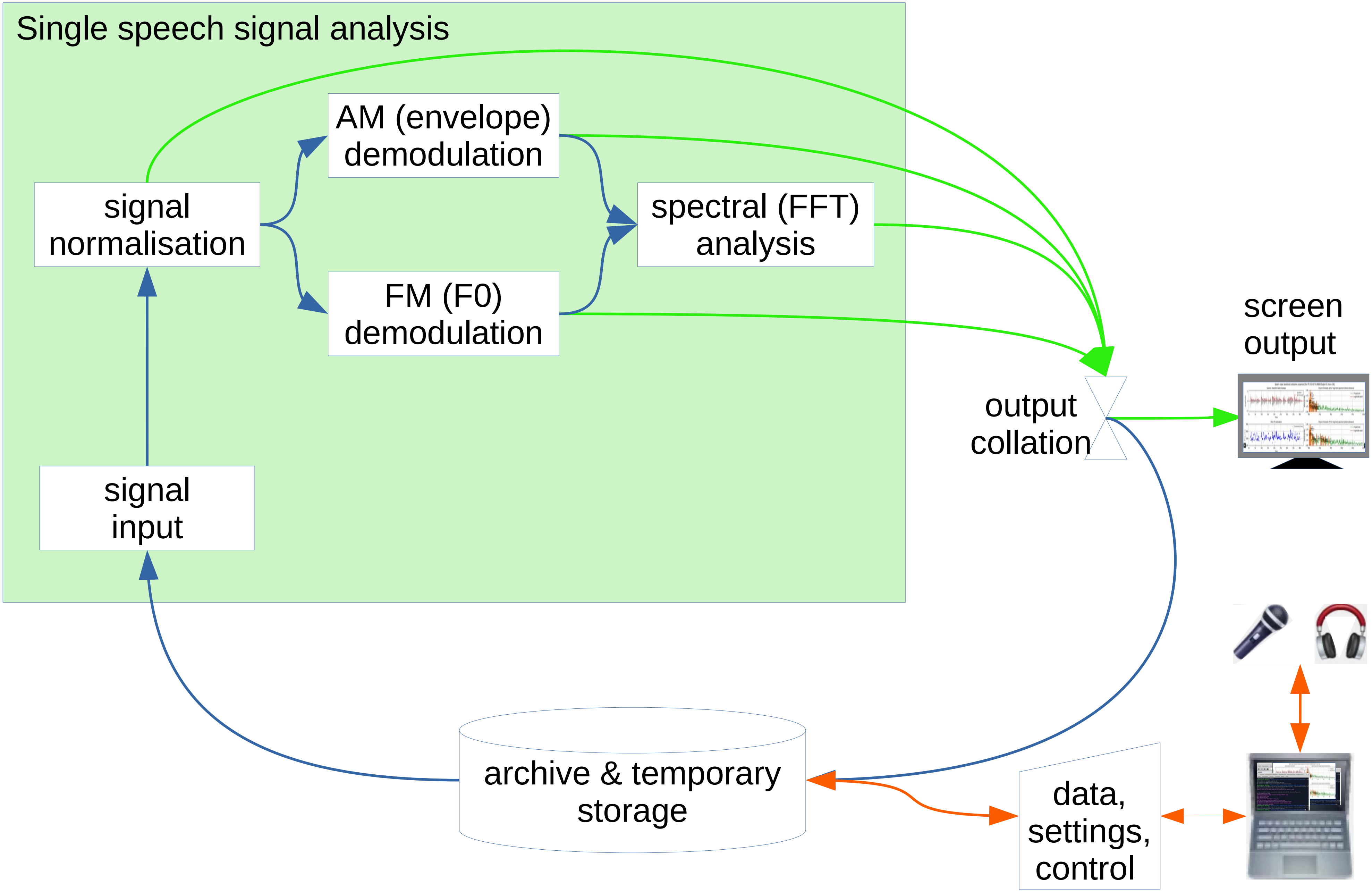


Hierarchical clustering::

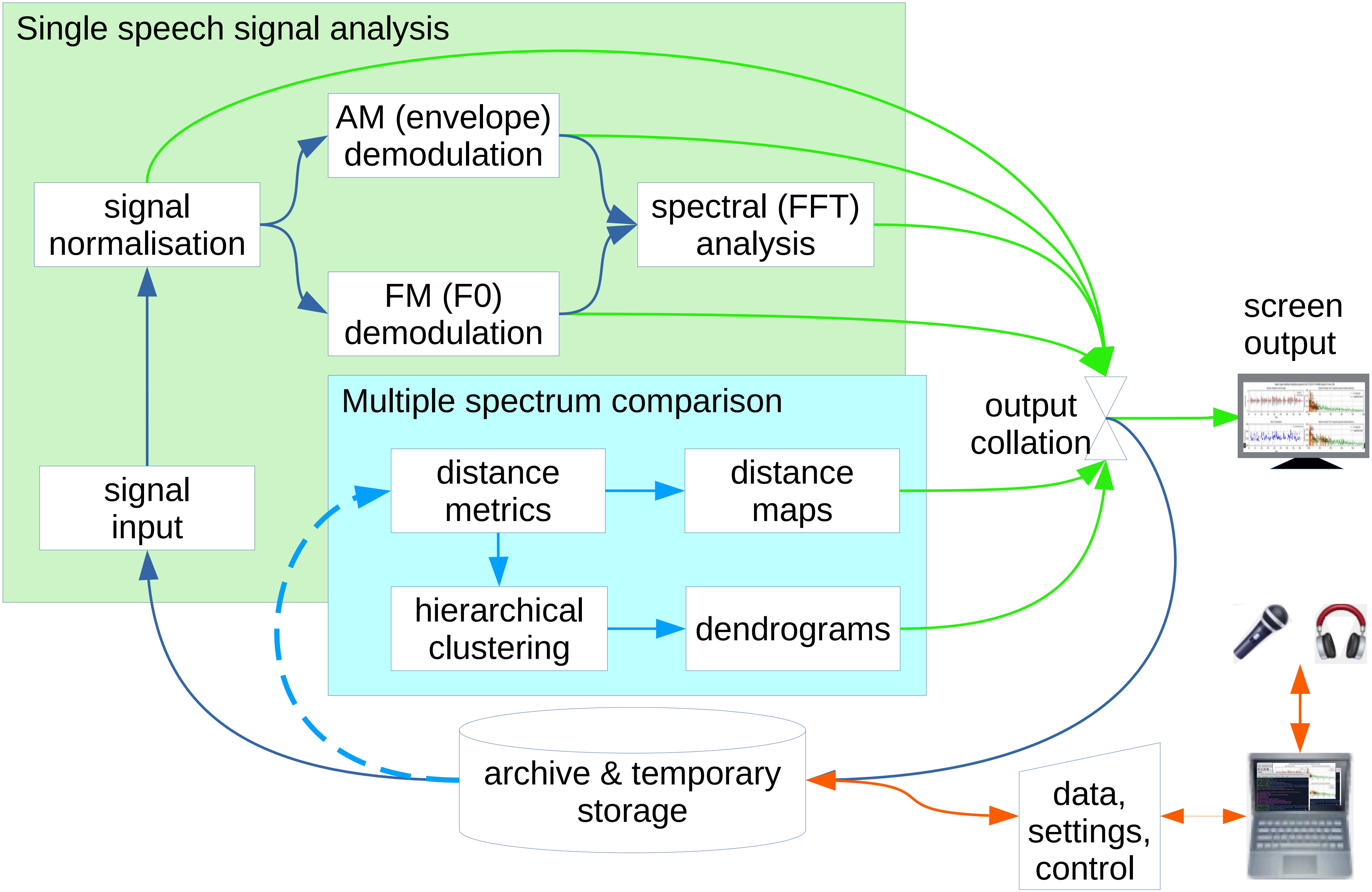


Demodulation and analysis: software design

Rhythm Formant Analysis Software Design: Data Flow



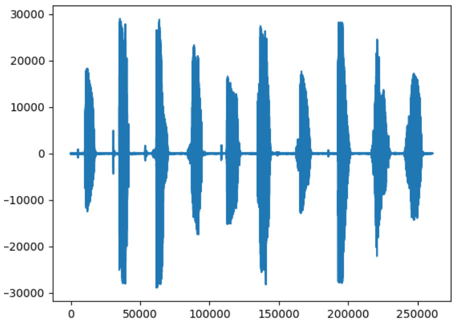
Rhythm Formant Analysis Software Design: Data Flow

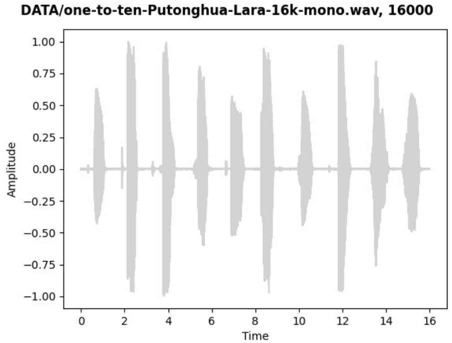


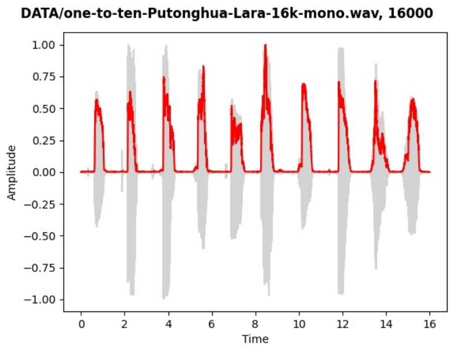
Demonstration:

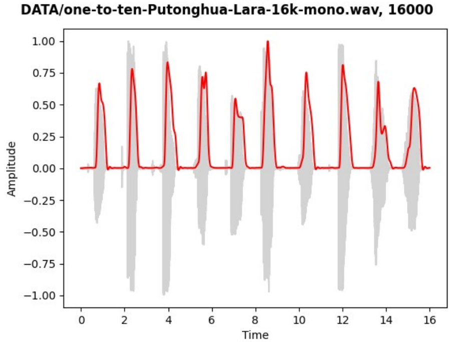
Demodulation, spectral analysis: processing single files

Demonstration applications: outputs

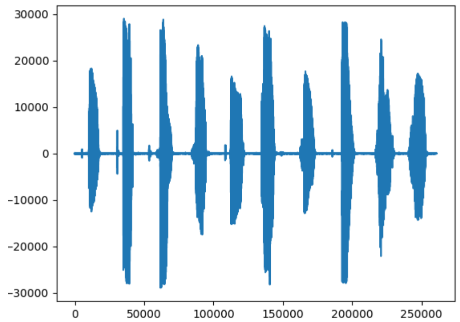




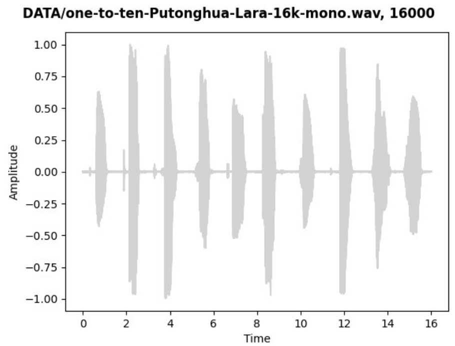


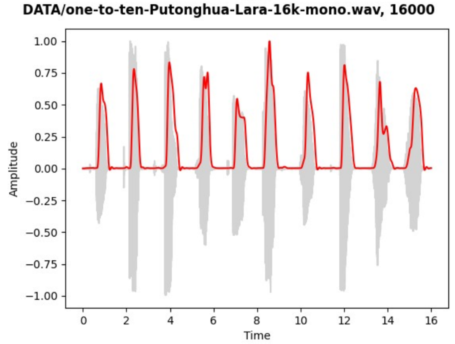


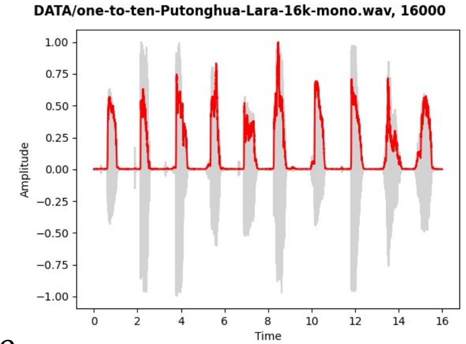
Demonstration apps - time domain outputs

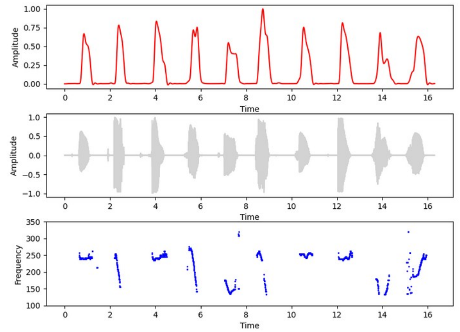


TIME
DOMAIN

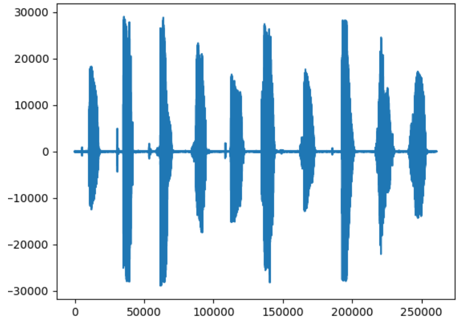


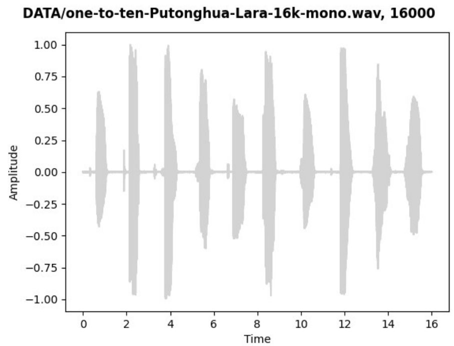


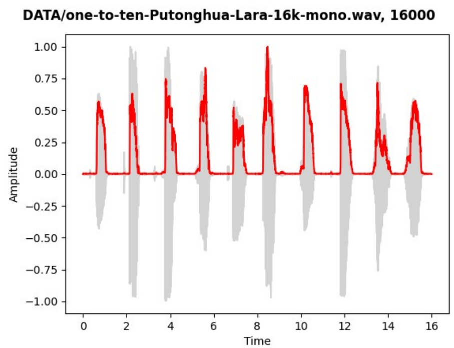


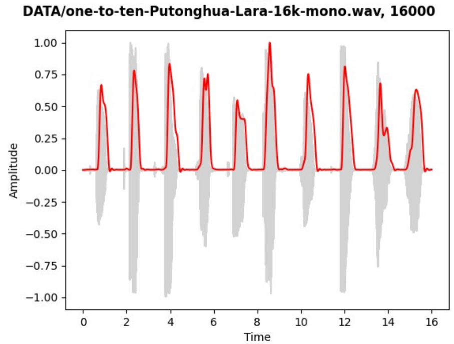


Demonstration apps – time and frequency domain outputs

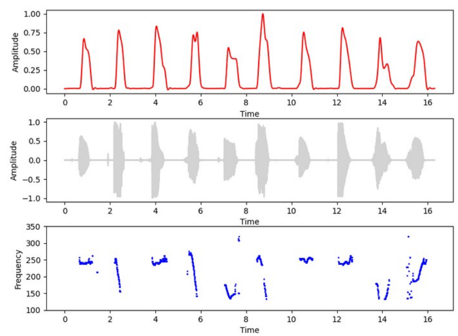




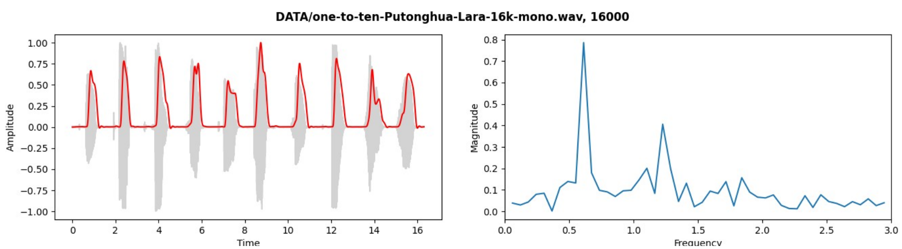


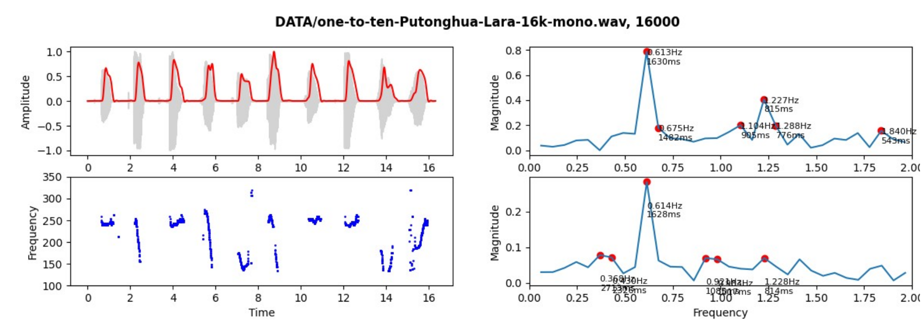


TIME DOMAIN



TIME DOMAIN	FREQUENCY DOMAIN
(waveform)	(spectrum)
Amplitude as a function of time	*Magnitude as a function of frequency*





Software description: time domain analysis

Time domain analysis: waveform display

```
# A_wavefor

import sys
import matpl
import scip

wavfilename
fs, signal =

plt.plot(si
plt.show()
```

Description

The programming language (in this case Python3) is provided with a large collection of algorithm implementations for processing various kinds of data for different purposes, stored in specialised ‘libraries’.

In this case, system function is imported, which allows the filename to be input from the command line, a science library function is imported which permits input of an audio file, and a graphics library is imported to produce figures.

A mono WAV file is read, and the speech signal and the sampling frequency are extracted from the file.

The signal is plotted as a graph and displayed.

Time domain analysis: waveform display

```
# A_waveform_display.py Waveform. D. Gibbon 2021-07-06

import sys                                # import specialised modules
import matplotlib.pyplot as plt
import scipy.io.wavfile as wave

wavfilename = sys.argv[1]                 # get input filename from command line
fs, signal = wave.read(wavfilename)       # read sampling frequency and signal

plt.plot(signal)                          # plot waveform
plt.show()                                # display figure
```

Time domain analysis: formatted waveform display

```
# B_waveforn

import sys
import numpy
import matpl
import scipy

wavfilename                                                  mmand line
fs, signal =                                                 d signal
signallength                                                 tes
signalsecond                                                 conds
signal = sig                                                 ... 1

#----------                                                  -----------

plt.suptitle                                                 le

xaxis = np.l                                                 in seconds
plt.plot(xax                                                 in grey
plt.xlabel("                                                 ls
plt.ylabel("

plt.tight_la
plt.show()
```

Description

In this application, in principle exactly the same thing happens, except that the figure is formatted more informatively.

For the calculations which are involved, a library of numerical functions is imported.

After reading the file, the amplitude of the signal is normalised between -1 and 1 for the *y*-axis of the graph, and the overall time in seconds is calculated for the *x*-axis from the sampling frequency and the length of the signal.

The normalised signal is plotted as a graph and displayed with the appropriate *x*-axis and *y*-axis information.

Time domain analysis: formatted waveform display

```
# B_waveform_display.py Formatted waveform display. D. Gibbon. 2021-07-06

import sys                                                  # import specialised modules
import numpy as np
import matplotlib.pyplot as plt
import scipy.io.wavfile as wave

wavfilename = sys.argv[1]                                   # get input filename from command line
fs, signal = wave.read(wavfilename)                         # read sampling frequency and signal
signallength = len(signal)                                  # define signal length in bytes
signalseconds = int(signallength / fs)                      # define signal length in seconds
signal = signal / max(abs(signal))                          # normalise signal -1 ... 0 ... 1

#------------------------------------------------------------------------------------------

plt.suptitle("%s, %d"%(wavfilename, fs), fontweight="bold")     # display a title

xaxis = np.linspace(0, signalseconds, signallength)             # define x axis in seconds
plt.plot(xaxis, signal, color="lightgrey")                      # plot waveform in grey
plt.xlabel("Time")                                              # add axis labels
plt.ylabel("Amplitude")

plt.tight_layout(pad=3)
plt.show()                                                      # display figure
```

Time domain analysis: waveform and envelope

```
# C_waveform_envelope_display.py Waveform & AM envelope medfilt. D. Gibbon 2021-07-06

import sys
import numpy
import matpl
import scipy
from scipy.s

wavfilename                                                                          l line
fs, signal =                                                                         gnal
signallength
signalsecond                                                                         s
signal = sig                                                                         .

envelope = m                                                                         e envelope
envelope = e

#----------                                                                          -----------

plt.suptitle                                                                         e

xaxis = np.l                                                                         in seconds
plt.plot(xax                                                                         in grey
plt.plot(xax                                                                         in red
plt.xlabel("                                                                         s
plt.ylabel("

plt.tight_la
plt.show()                                                          # display figure
```

Description

In this application, everything which happened in the previous applications also happens, but in addition, the *amplitude modulation of the signal* is demodulated.

This is done by taking the *absolute signal*, that is, only positive values of the signal (or conversion of negative values of the signal into positive values), and low-pass filtering (smoothing) the result.

Low-pass filtering (smoothing) is done here with a ***moving median filter***, which moves through the signal calculating the median values of intervals in the signal. The method is rather slow, and somewhat difficult to characterise. But it works...

Time domain analysis: waveform and envelope

```
# C_waveform_envelope_display.py Waveform & AM envelope medfilt. D. Gibbon 2021-07-06

import sys                                          # import specialised modules
import numpy as np
import matplotlib.pyplot as plt
import scipy.io.wavfile as wave
from scipy.signal import medfilt

wavfilename = sys.argv[1]                           # get input filename from command line
fs, signal = wave.read(wavfilename)                 # read sampling frequency and signal
signallength = len(signal)                          # define signal length in bytes
signalseconds = int(signallength / fs)              # define signal length in seconds
signal = signal / max(abs(signal))                  # normalise signal -1 ... 0 ... 1

envelope = medfilt(abs(signal), 301)                # extract low frequency amplitude envelope
envelope = envelope / max(envelope)                 # normalise envelope to 0 ... 1

#-----------------------------------------------------------------------------------------

plt.suptitle("%s, %d"%(wavfilename, fs), fontweight="bold")     # display a title

xaxis = np.linspace(0, signalseconds, signallength)             # define x axis in seconds
plt.plot(xaxis, signal, color="lightgrey")                      # plot waveform in grey
plt.plot(xaxis, envelope, color="red"                           # plot envelope in red
plt.xlabel("Time")                                              # add axis labels
plt.ylabel("Amplitude")

plt.tight_layout(pad=3)
plt.show()                                                      # display figure
```

Time domain analysis: waveform and envelope

```
# D_waveform_envelope_display.py Wwaveform, AM envelope Butterworth. D. Gibbon 2021-07-06

import sys
import numpy
import matpl
import scipy
from scipy.s

wavfilename                                                              mmand line
fs, signal =                                                             d signal
signallength                                                             tes
signalsecond                                                             conds
signal = sig                                                             .. 1

b, a = butte
envelope = l                                                             envelope
envelope = e

#-----------                                                             -----------

plt.suptitle                                                             e

xaxis = np.l                                                             in seconds

plt.plot(xax                                                             in grey
plt.plot(xax                                                             in red
plt.xlabel("                                                             s
plt.ylabel("

plt.tight_layout(pad=3)
plt.show()                                                     # display figure
```

Description

Again, in this application, everything which happened in the previous applications.

Low-pass filtering is done here with a ***Butterworth filter***, which lowers the amplitude of frequencies above a specified cutoff frequency. This is advisable since the idea is to capture only the very low frequencies in the spectrum which make up the rhythms of speech.This filter is much more efficient than the moving median filter.

Time domain analysis: waveform and envelope

```
# D_waveform_envelope_display.py Wwaveform, AM envelope Butterworth. D. Gibbon 2021-07-06

import sys                                              # import specialised modules
import numpy as np
import matplotlib.pyplot as plt
import scipy.io.wavfile as wave
from scipy.signal import medfilt, butter, lfilter

wavfilename = sys.argv[1]                               # get input filename from command line
fs, signal = wave.read(wavfilename)                     # read sampling frequency and signal
signallength = len(signal)                              # define signal length in bytes
signalseconds = int(signallength / fs)                  # define signal length in seconds
signal = signal / max(abs(signal))                      # normalise signal -1 ... 0 ... 1

b, a = butter(5, 5 / (0.5 * fs), btype="low")           # define Butterworth filter
envelope = lfilter(b, a, abs(signal))                   # apply filter to create lf envelope
envelope = envelope / max(envelope)                     # normalise envelope 0 ... 1

#---------------------------------------------------------------------------------------------

plt.suptitle("%s, %d"%(wavfilename, fs), fontweight="bold")           # display a title

xaxis = np.linspace(0, signalseconds, signallength)                   # define x axis in seconds

plt.plot(xaxis, signal, color="lightgrey")                            # plot waveform in grey
plt.plot(xaxis, envelope, color="red")                                # plot waveform in red
plt.xlabel("Time")                                                    # add axis labels
plt.ylabel("Amplitude")

plt.tight_layout(pad=3)
plt.show()                                                            # display figure
```

Frequency domain analysis: FFT and AM spectrum

```
#  E_waveform envelope spectrum display Addition of LF spectrum, D. Gibbon, 2021-07-06

import sys
import numpy as np
import matplotlib.pyplot as
import scipy.io.wavfile as w
from scipy.signal import med

wavfilename = sys.argv[1]
fs, signal = wave.read(wavfi
signallength = len(signal)
signalseconds = signallength
signal = signal / max(abs(si

b, a = butter(5, 5 / (0.5 *
envelope = lfilter(b, a, abs
envelope = envelope / max(en


specmags = n                                              ith FFT
specmags = s
specmaglen =
specfreqs =
spectrummax                                               spectrum
lfspecmaglen                                              m length
lfspecmags =                                              itudes
lfspecfreqs                                               uencies
#-----------                                              -----------
fig,((plt01,                                              e format

plt.suptitle("%s, %d"%(wavfi

xaxistime = np.linspace(0, s
plt01.plot(xaxistime, signal
plt01.plot(xaxistime, envelo
plt01.set_xlabel("Time")
plt01.set_ylabel("Amplitude"

plt02.plot(l
plt02.set_xl
plt02.set_yl
plt02.set_xlim(0,spectrummax)


plt.tight_layout(pad=3)
plt.show()                                # display figure
```

Description

In this app, a major step forward is taken: the amplitude envelope has been extracted and now it is time to analyse the rhythms. No additional library is needed for this.

The first step in analysing the speech rhythms is done by first applying a ***Fast Fourier Transform*** to the entire envelope in order to produce a spectral analysis.

This step means moving from the *time domain* of the signal, in which the amplitude of the signal is a function of the time in seconds, to the *frequency domain*, with the magnitude of each frequency in the signal displayed as a *spectrum*, magnitudes normalised from 0 to 1.

The frequencies in the spectrum can be seen to cluster in identifiable regions, which are interpreted as *rhythm formants*. The *rhythm formants* have very low frequencies below about 10 Hz, that is, 10 beats per second. The *phone formants*, which identify vowels and consonants, have much higher frequencies above about 300 Hz, ranging to several thousand Hz.

Frequency domain analysis: FFT and AM spectrum

```
#  E_waveform_envelope_spectrum_display Addition of LF spectrum. D. Gibbon, 2021-07-06

import sys                                           # import specialised modules
import numpy as np
import matplotlib.pyplot as plt
import scipy.io.wavfile as wave
from scipy.signal import medfilt, butter, lfilter

wavfilename = sys.argv[1]                            # get input filename from command line
fs, signal = wave.read(wavfilename)                  # read sampling frequency and signal
signallength = len(signal)                           # define signal length in bytes
signalseconds = signallength / fs                    # define signal length in seconds
signal = signal / max(abs(signal))                   # normalise signal -1 ... 0 ... 1

b, a = butter(5, 5 / (0.5 * fs), btype="low")        # define Butterworth filter
envelope = lfilter(b, a, abs(signal))                # apply filter to create lf envelope
envelope = envelope / max(envelope)                  # normalise envelope 0 ... 1

specmags = np.abs(np.fft.rfft(envelope))       # calculate spectrum magnitudes with FFT
specmags = specmags / np.max(specmags)         # normalise magnitudes to 0 .. 1
specmaglen = len(specmags)                     # get length of spectrum
specfreqs = np.linspace(0,fs/2,specmaglen)     # get frequencies in spectrum
spectrummax = 3                                # define maximum frequency in lf spectrum
lfspecmaglen = int(round(spectrummax * specmaglen / (fs / 2))) # get lf spectrum length
lfspecmags = specmags[1:lfspecmaglen]          # set low frequency spectrum magnitudes
lfspecfreqs = specfreqs[1:lfspecmaglen]        # set low frequency spectrum frequencies

#-----------------------------------------------------------------------------------------

fig,((plt01, plt02)) = plt.subplots(nrows=1, ncols=2, figsize=(14, 4))  # figure format

plt.suptitle("%s, %d"%(wavfilename, fs), fontweight="bold")         # display a title

xaxistime = np.linspace(0, signalseconds, signallength)             # define x axis in seconds
plt01.plot(xaxistime, signal, color="lightgrey")                    # plot waveform in grey
plt01.plot(xaxistime, envelope, color="red")
plt01.set_xlabel("Time")
plt01.set_ylabel("Amplitude")

plt02.plot(lfspecfreqs, lfspecmags)
plt02.set_xlabel("Frequency")
plt02.set_ylabel("Magnitude")
plt02.set_xlim(0,spectrummax)

plt.tight_layout(pad=3)
plt.show()                                                          # display figure
```

Frequency domain analysis: peaks in AM spectrum

```
#  F_waveform_envelope_spectrum_display Addition of LF spectrum dots. D. Gibbon, 2021-07-06
import sys
import numpy as np
import matplotlib.pyplot as
import scipy.io.wavfile as w
from scipy.signal import med

wavfilename = sys.argv[1]
fs, signal = wave.read(wavfi
signallength = len(signal)
signalseconds = signallength
signal = signal / max(abs(si

b, a = butter(5, 5 / (0.5 *
envelope = lfilter(b, a, abs
envelope = envelope / max(en


specmags = np.abs(np.
specmags = specmags /
specmaglen = len(spec
specfreqs = np.linspa
spectrummax = 3
lfspecmaglen = int(ro
lfspecmags = specmags
lfspecfreqs = specfre

topmagscount                                                              spectrum
topmags = so
toppos = [ l                                                              positions
topfreqs = [                                                              s

#----------                                                              -----------

fig,((plt01, plt02))

plt.suptitle("%s, %d"%(wavfi

xaxistime = np.linspace(0, s
plt01.plot(xaxistime, signal
plt01.plot(xaxistime, envelo
plt01.set_xlabel("Time")
plt01.set_ylabel("Amplitude"

plt02.plot(lfspecfreq
plt02.scatte                                                              ed dots
for f,m in z                                                              op values
    plt02.te                                                              d values
plt02.set_xlabel("Frequency")
plt02.set_ylabel("Magnitude")
plt02.set_xlim(0,spectrummax)

plt.tight_layout(pad=3)
plt.show()                                # display figure
```

Description

This app again takes a small step forward, and ***defines critical minimal values for frequency magnitudes in the spectrum*** which are relevant for ***Rhythm Formant Analysis***. These values are found by trial and error in the first stages of analysis, and later predicted on the basis of previous analyses.

The relevant frequency magnitudes are marked in the spectrum.

Frequency domain analysis: peaks in AM spectrum

```
#  F_waveform_envelope_spectrum_display Addition of LF spectrum dots. D. Gibbon, 2021-07-06

import sys                                        # import specialised modules
import numpy as np
import matplotlib.pyplot as plt
import scipy.io.wavfile as wave
from scipy.signal import medfilt, butter, lfilter

wavfilename = sys.argv[1]                         # get input filename from command line
fs, signal = wave.read(wavfilename)               # read sampling frequency and signal
signallength = len(signal)                        # define signal length in bytes
signalseconds = signallength / fs                 # define signal length in seconds
signal = signal / max(abs(signal))                # normalise signal -1 ... 0 ... 1

b, a = butter(5, 5 / (0.5 * fs), btype="low")     # define Butterworth filter
envelope = lfilter(b, a, abs(signal))             # apply filter to create lf envelope
envelope = envelope / max(envelope)               # normalise envelope 0 ... 1


specmags = np.abs(np.fft.rfft(envelope))           # calculate spectrum magnitudes with FFT
specmags = specmags / np.max(specmags)             # normalise magnitudes to 0 .. 1
specmaglen = len(specmags)                                # get length of spectrum
specfreqs = np.linspace(0,fs/2,specmaglen)         # get frequencies in spectrum
spectrummax = 3                                                   # define maximum frequency in lf spectrum
lfspecmaglen = int(round(spectrummax * specmaglen / (fs / 2))) # get lf spectrum length
lfspecmags = specmags[1:lfspecmaglen]                     # set low frequency spectrum magnitudes
lfspecfreqs = specfreqs[1:lfspecmaglen]            # set low frequency spectrum frequencies

topmagscount = 6                                     # define max frequency of lf spectrum
topmags = sorted(lfspecmags)[-topmagscount:]         # get top magnitudes
toppos = [ list(lfspecmags).index(m) for m in topmags ]    # get top magnitude positions
topfreqs = [ lfspecfreqs[p] for p in toppos ]              # get top frequencies

#----------------------------------------------------------------------------------------

fig,((plt01, plt02)) = plt.subplots(nrows=1, ncols=2, figsize=(14, 4))    # figure format

plt.suptitle("%s, %d"%(wavfilename, fs), fontweight="bold")        # display a title

xaxistime = np.linspace(0, signalseconds, signallength)            # define x axis in seconds
plt01.plot(xaxistime, signal, color="lightgrey")                   # plot waveform in grey
plt01.plot(xaxistime, envelope, color="red")
plt01.set_xlabel("Time")
plt01.set_ylabel("Amplitude")

plt02.plot(lfspecfreqs, lfspecmags)
plt02.scatter(topfreqs, topmags, color="red")                      # Scatter plot red dots
for f,m in zip(topfreqs, topmags):                                 # loop through top values
    plt02.text(f, m-0.1, "%.3fHz\n%dms"%(f,1000/f), fontsize=8)# print formatted values
plt02.set_xlabel("Frequency")
plt02.set_ylabel("Magnitude")
plt02.set_xlim(0,spectrummax)

plt.tight_layout(pad=3)
plt.show()                                                         # display figure
```

Frequency Domain Analysis: File output

```
# G_waveform

import sys
import numpy as np
import matplotlib.pyplot
import scipy.io.wavfile
from scipy.signal import

wavfilename = sys.argv[1
fs, signal = wave.read(w
signallength = len(signa
signalseconds = signalle
signal = signal / max(ab

b, a = butter(5, 5 / (0.
envelope = lfilter(b, a,
envelope = envelope / ma

specmags = np.abs(np.fft
specmags = specmags / np
specmaglen = len(specmag
specfreqs = np.linspace(
spectrummax = 3
lfspecmaglen = int(round
lfspecmags = specmags[1:
lfspecfreqs = specfreqs[

topmagscount = 6
topmags = sorted(lfspecm
toppos = [ list(lfspecma
topfreqs = [ lfspecfreqs

#----------------------

fig,((plt01, plt02)) = p

plt.suptitle("%s, %d"%(w

xaxistime = np.linspace(
plt01.plot(xaxistime, si
plt01.plot(xaxistime, en
plt01.set_xlabel("Time")
plt01.set_ylabel("Amplit

plt02.plot(lfspecfreqs,
plt02.scatter(topfreqs,
for f,m in zip(topfreqs,
        plt02.text(f, m-
plt02.set_xlabel("Freque
plt02.set_ylabel("Magnit
plt02.set_xlim(0,spectru
```

Description

The small step forward taken by this app is simply to output the values of the spectrum to a file, formated as a table in CSV format, as well as saving the figure in PNG format.

This format can be imported by other applications, such as spreadsheet programs like Excel or LibreOffice Calc.

The figure display is not affected.

```
ename):
 w')
(text)

ename):
 a')
(text)
```

```
'.join(
pecfreqs ]

n(
pecmags ]

filename)
ilename)

ename)
```

```
plt.tight_layout(pad=3)
plt.savefig(wavfilename[:-3]+".png")
plt.show()                                  # display figure
```

Frequency Domain Analysis: File output

```
# G_waveform_spectrum_file_outputs.py D. Gibbon, 2021-07-

import sys                                              # import specialised modules
import numpy as np
import matplotlib.pyplot as plt
import scipy.io.wavfile as wave
from scipy.signal import medfilt, butter, lfilter

wavfilename = sys.argv[1]                               # get input filename from command line
fs, signal = wave.read(wavfilename)                     # read sampling frequency and signal
signallength = len(signal)                              # define signal length in bytes
signalseconds = signallength / fs                       # define signal length in seconds
signal = signal / max(abs(signal))                      # normalise signal -1 ... 0 ... 1

b, a = butter(5, 5 / (0.5 * fs), btype="low")           # define Butterworth filter
envelope = lfilter(b, a, abs(signal))                   # apply filter to create lf envelope
envelope = envelope / max(envelope)                     # normalise envelope 0 ... 1

specmags = np.abs(np.fft.rfft(envelope))                # calculate spectrum magnitudes with FFT
specmags = specmags / np.max(specmags)                  # normalise magnitudes to 0 .. 1
specmaglen = len(specmags)                                      # get length of spectrum
specfreqs = np.linspace(0,fs/2,specmaglen)              # get frequencies in spectrum
spectrummax = 3                                         # define maximum frequency in lf spectrum
lfspecmaglen = int(round(spectrummax * specmaglen / (fs / 2))) # get lf spectrum length
lfspecmags = specmags[1:lfspecmaglen]                   # set low frequency spectrum magnitudes
lfspecfreqs = specfreqs[1:lfspecmaglen]                 # set low frequency spectrum frequencies

topmagscount = 6                                        # define max frequency of lf spectrum
topmags = sorted(lfspecmags)[-topmagscount:]            # get top magnitudes
toppos = [ list(lfspecmags).index(m) for m in topmags ] # get top magnitude positions
topfreqs = [ lfspecfreqs[p] for p in toppos ]                   # get top frequencies

#-----------------------------------------------------------------------------------------

fig,((plt01, plt02)) = plt.subplots(nrows=1, ncols=2, figsize=(14, 4))  # figure format

plt.suptitle("%s, %d"%(wavfilename, fs), fontweight="bold")             # display a title

xaxistime = np.linspace(0, signalseconds, signallength)                 # define x axis in seconds
plt01.plot(xaxistime, signal, color="lightgrey")                        # plot waveform in grey
plt01.plot(xaxistime, envelope, color="red")
plt01.set_xlabel("Time")
plt01.set_ylabel("Amplitude")

plt02.plot(lfspecfreqs, lfspecmags)
plt02.scatter(topfreqs, topmags, color="red")                           # Scatter plot red dots
for f,m in zip(topfreqs, topmags):                                      # loop through top values
        plt02.text(f, m-0.1, "%.3fHz\n%dms"%(f,1000/f), fontsize=8)# print formatted values
plt02.set_xlabel("Frequency")
plt02.set_ylabel("Magnitude")
plt02.set_xlim(0,spectrummax)

plt.tight_layout(pad=3)
plt.savefig(wavfilename[:-3]+".png")
plt.show()                                                              # display figure
```

```
import os

def outputtextlines(text, filename):
        handle = open(filename,'w')
        linelist = handle.write(text)
        handle.close()
        return

def appendtextlines(text, filename):
        handle = open(filename,'a')
        linelist = handle.write(text)
        handle.close()
        return
```

```
csvfreqs = "lffreqs\t"+"\t".join(
        [ "%.3f"%x for x in lfspecfreqs ]
        )+"\n"
csvmags = "lfmags\t"+"\t".join(
        [ "%.3f"%x for x in lfspecmags ]
        )+"\n"

outputtextlines(csvfreqs, csvfilename)
appendtextlines(csvmags, csvfilename)

os.system("soffice %s"%csvfilename)
```

Comparing multiple files

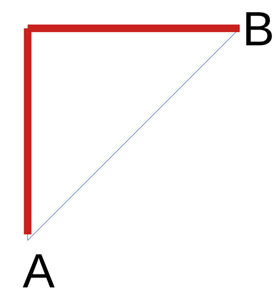
Comparison of English and German story readings

An English example: *The North Wind and the Sun*	A German example: *Nordwind und Sonne*

Distance metrics

Manhattan Distance
(Cityblock distance, Taxicab Distance)
'around the corner'

$$\sum_{i=1}^{n} |x_i - y_i|$$

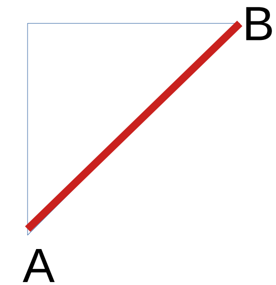


Canberra Distance
(Normalised Manhattan Distance)

$$\sum_{i=1}^{n} \frac{|x_i - y_i|}{|x_i| + |y_i|}$$

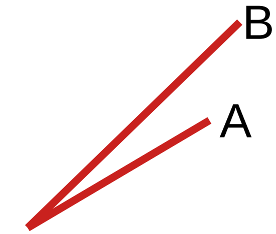
Euclidean Distance
direct distance
'as the crow flies'

$$\sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$



Cosine Distance
angle, direction, not magnitude
so not distance itself
'hiker's orientation'

$$\frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \sqrt{\sum_{i=1}^{n} y_i^2}}$$

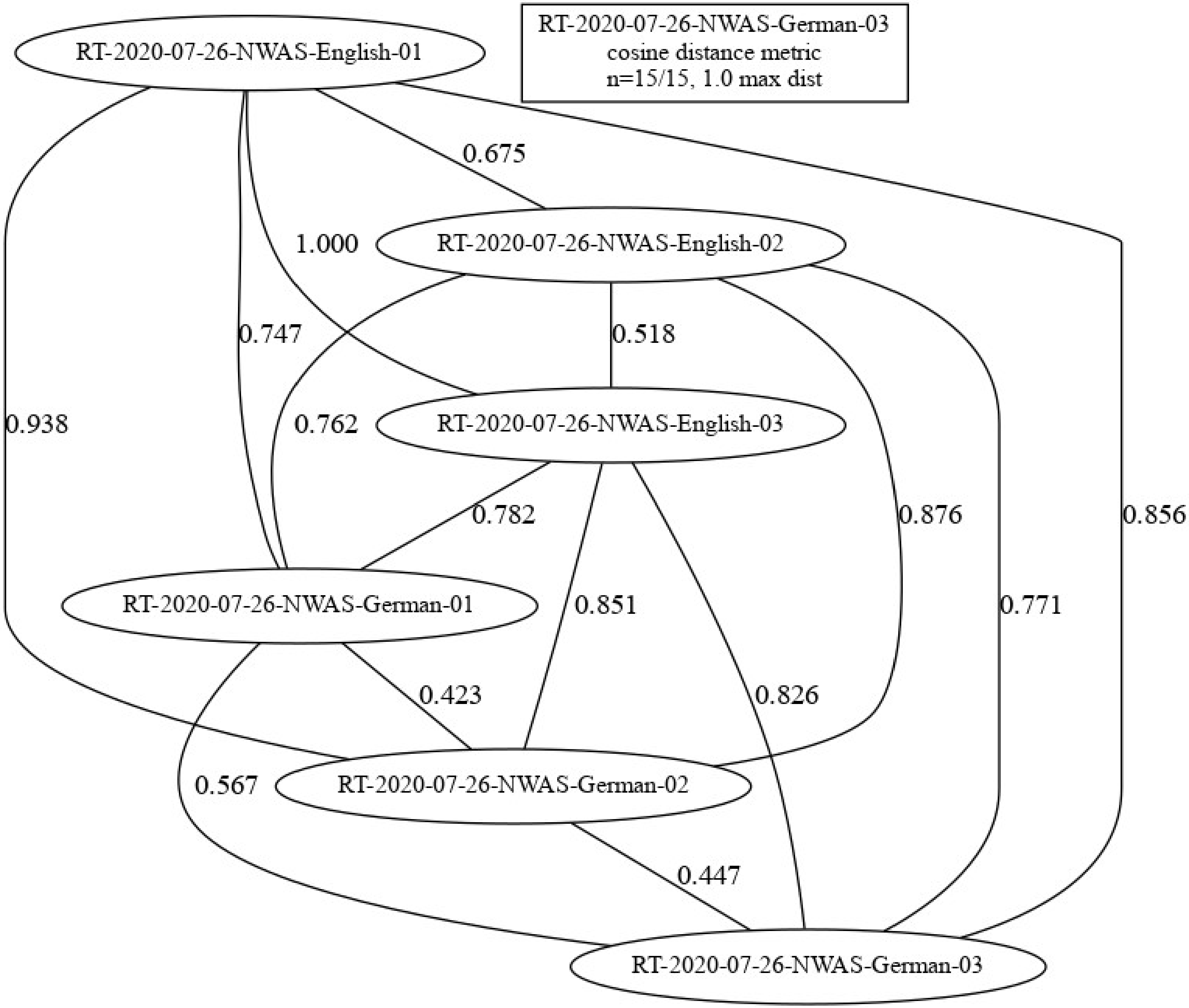


Spectrum Comparison: Distance Table

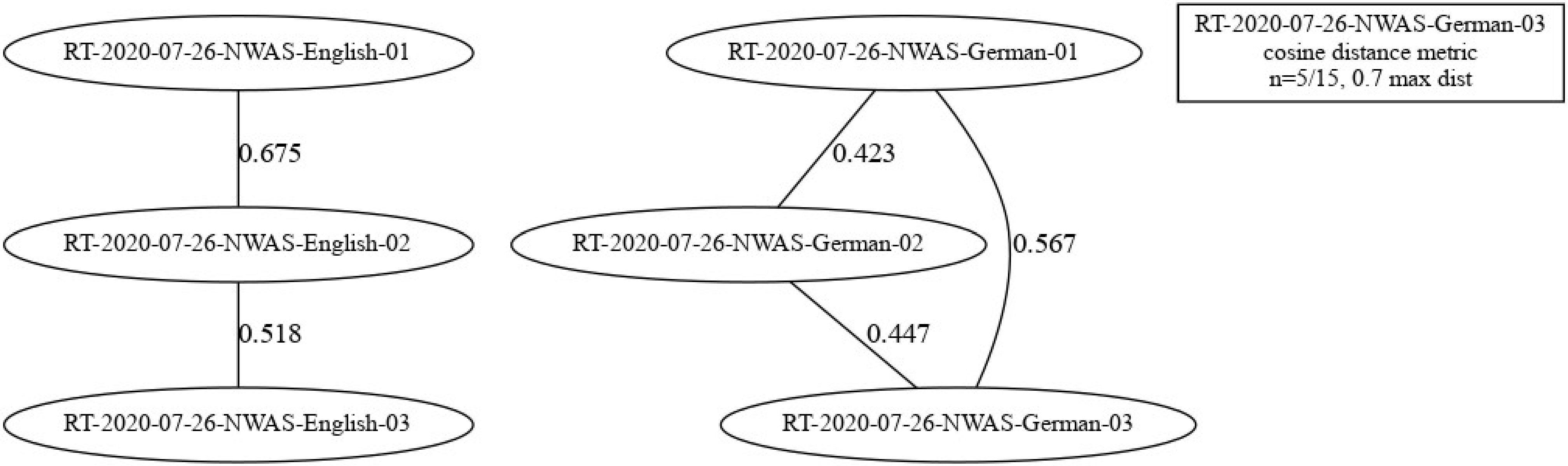
	Eng 01	Eng 02	Eng 03	Ger 01	Ger 02	Ger 03
		0.67477731	1.	0.74745837	0.93762055	0.85622088
			0.5184008	0.76221046	0.87568858	0.7706713
				0.78197106	0.85094568	0.82617612
					0.42298678	0.56668163
						0.44727788

Adult Female English-German bilingual reading
The North Wind and the Sun,
3 English, 3 German, in order of production.

Distance map



Distance map



An English example:
The North Wind and the Sun

A German example:
Nordwind und Sonne

Code

The code is at

https://www.github.com/dafyddg/RFA

The main directory of this GitHub repository contains the following directories (**bold**) and files:

Articles	*articles on RFT and RFA*
IICBP2022-slides	*slides for Brazilian Phonetics minicourse 2022*
LittleHelpers	*small RFA demo scripts and data*
README.1st	*documentation*
RFA_multiple_signal_processing	*scripts for multiple file processing and cluster analysis*
RFA_multiple_signal_processing.zip	
RFA_single_signal_processing	*script and modules for single file analysis*
RFA_single_signal_processing.zip	