Sounds of Prosody

Rhythm, Melody and Quite Big Data

2019-07-24, 14:00-16:00 Beijing, 08:00-10:00 Berlin

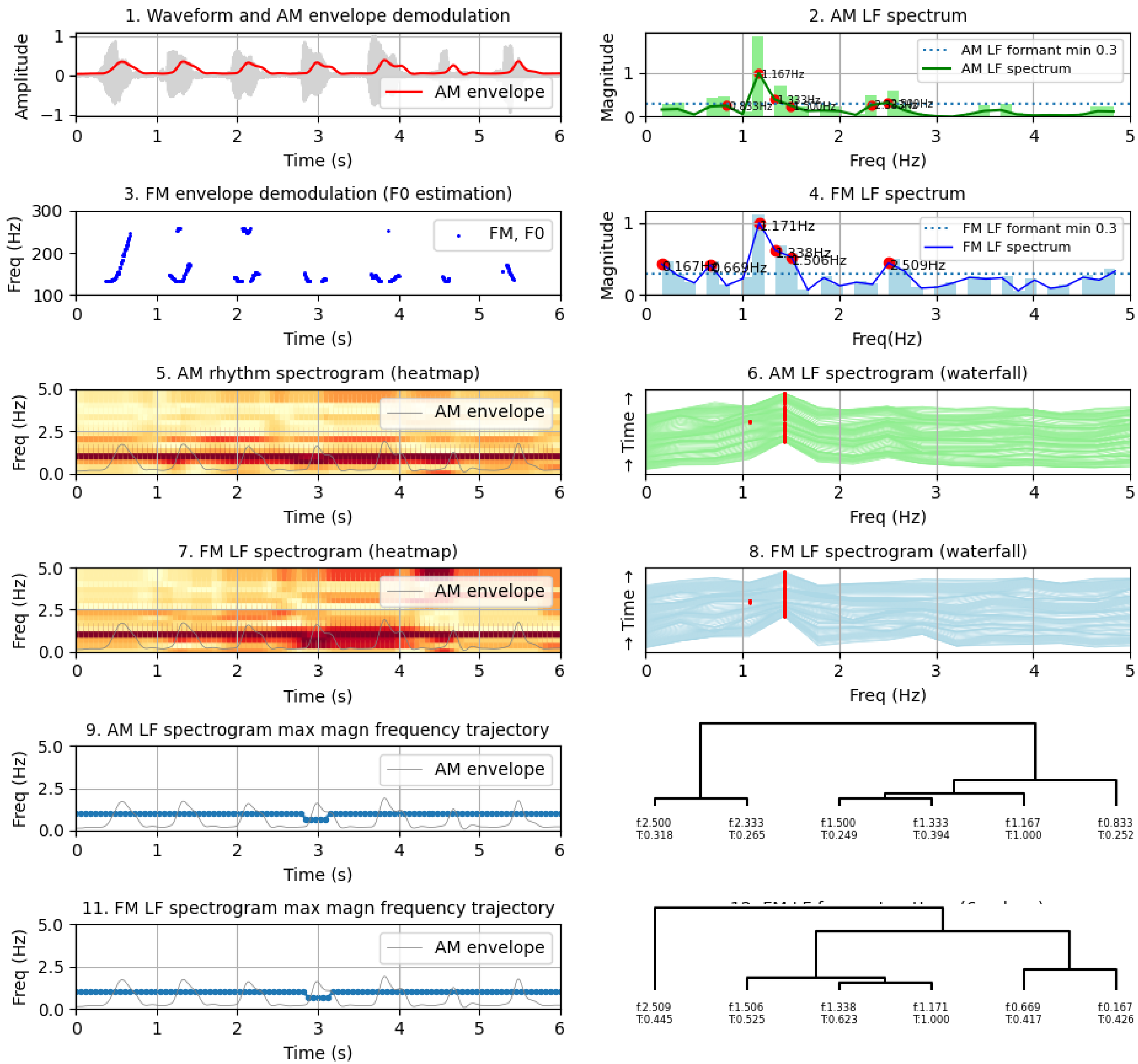
Dafydd Gibbon

Bielefeld University

Chinese Summer School:
Contemporary Phonetics and Phonology

The Main Topics

https://github.com/dafyddg/RFA



Questions from Lecture 2

Q: Does the prosodic lexicon also contain meanings?

A: Yes, the information is in features; the meaning of a pitch accent is a function from a coextensive locutionary segment to the deictic *origo* (*I*, *here*, *now*), comparable with other deictic forms such as “this”, “here”.

Q: What is the Type 3 grammar for call contours?

A: ... B → pa[h, chroma] C, C → *downstep* B (pitch accent loop; see Pierrehumbert discussion later)

Q: Are your grammars for intonation and tone like those of Pierrehumbert?

A: Yes, they use the same formal theory, as a finite state automaton (dates back to the 1940s, McCulloch-Pitts) or as a Type 3 grammar (dates back to the 1950s, Chomsky).

Q: What does ‘exponential complexity’ mean?

A: For a sentence of length n, where G is a property of the Grammar:

linear time: Gn
cubic time: Gn^3
exponential time: G^n

Q: What is the difference between rhythm and prosody?

A: Prosody is the music of speech, consisting of rhythms and melodies.

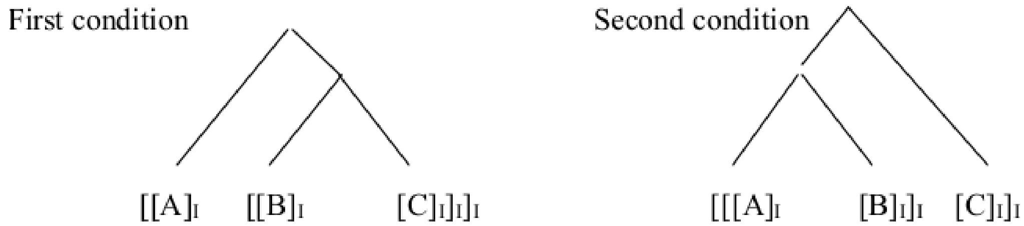
Q: Why is a right-branching structure not centre-embedding, even if the nodes have the same label, like in the example of Féry?

A: See the following slides.

Question from Lecture 2

Q: Why is a right-branching structure not centre-embedding, even if the nodes have the same label, like in the example of Féry?

“I assume a recursive structure: All sentences are i-phrases, the grouping of two sentences is also an i-phrase, and the whole utterance is an i-phrase as well.”



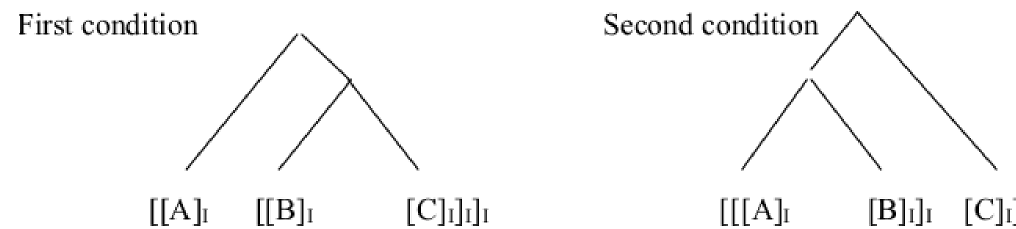
A: The labelling decision is internal to the theory, which is not very clear. Incidentally, the structures violate the Strict Level Hypothesis (but this does not affect the argument):

1. Moving left-to-right, in each case the following item is not embedded, but added on to the previous item, with right-branching it is top-down, with left-branching it is bottom up.
2. Left-branching and right-branching grammars describe the same terminal strings as the corresponding finite state automaton,and require only <u>finite memory</u> and <u>linear processing time</u>, unlike centre-embedding grammars.

Question from Lecture 2

Q: Why is a right-branching structure not centre-embedding, even if the nodes have the same label, like in the example of Féry?

"I assume a recursive structure: All sentences are i-phrases, the grouping of two sentences is also an i-phrase, and the whole utterance is an i-phrase as well."



A: The labelling decision is internal to the theory, which is not very clear. Incidentally, the structures violate the Strict Level Hypothesis (but this does not affect the argument):

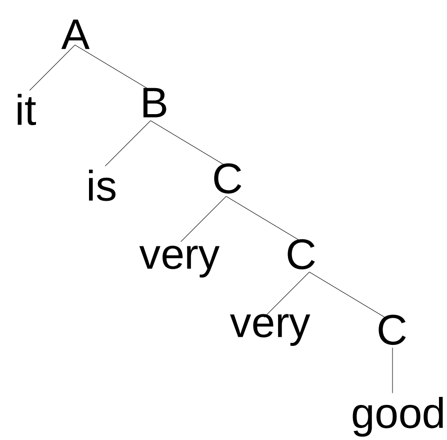
1. Moving left-to-right, in each cas … to the previous item, with right-branching it is top-down, with le…
2. Left-branching and right-branch… s the corresponding finite state automaton,and require only fin… e-embedding grammars.

There are processing differences, not differences in the language described:

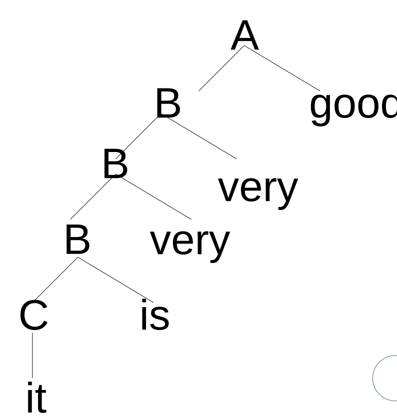
- Right branching: left-to-right is simplest.
- Left branching: right-to-left is simplest.

Right-branching Type 3:
A → it B
B → is C
C → very C
C → good

Left-branching Type 3:
A → B good
B → B very
B → C is
C → it

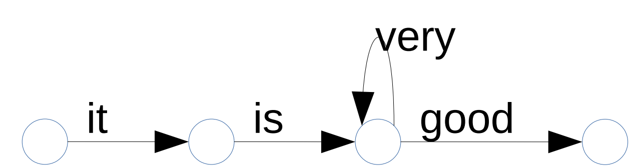


right-branching



left-branching

Right-recursive grammars are equivalent to iterative finite state automata.

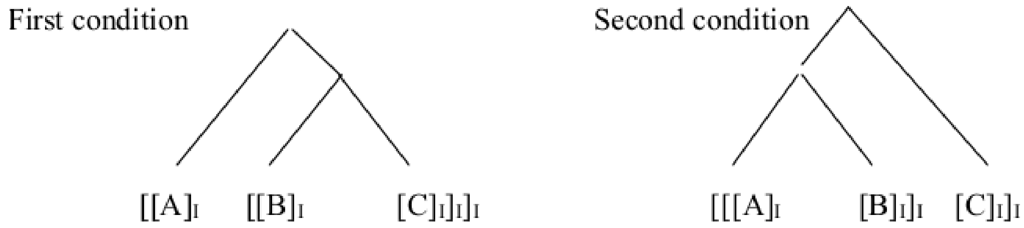


FSN ≡ FSA

Question from Lecture 2

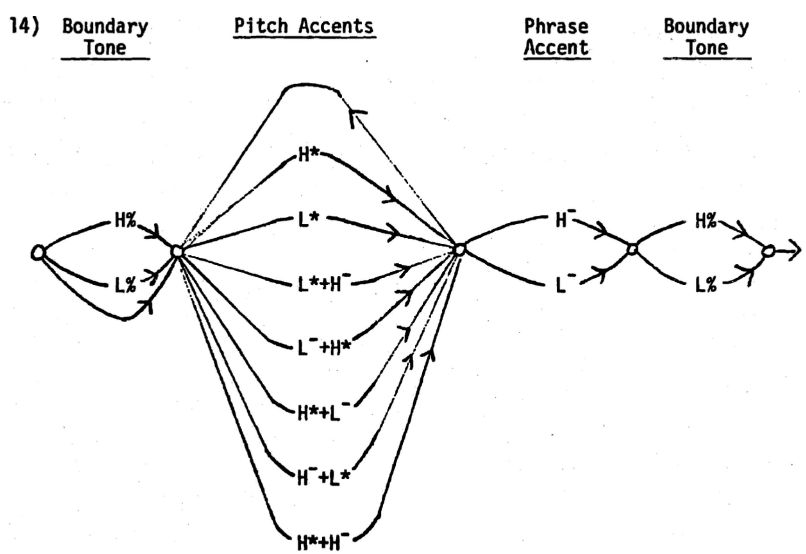
Q: Why is a right-branching structure not centre-embedding, even if the nodes have the same label, like in the example of Féry?

"I assume a recursive structure: All sentences are i-phrases, the grouping of two sentences is also an i-phrase, and the whole utterance is an i-phrase as well."



A: The labelling decision is internal to the theory, which is not very clear. Incidentally, the structures violate the Strict Level Hypothesis (but this does not affect the argument):

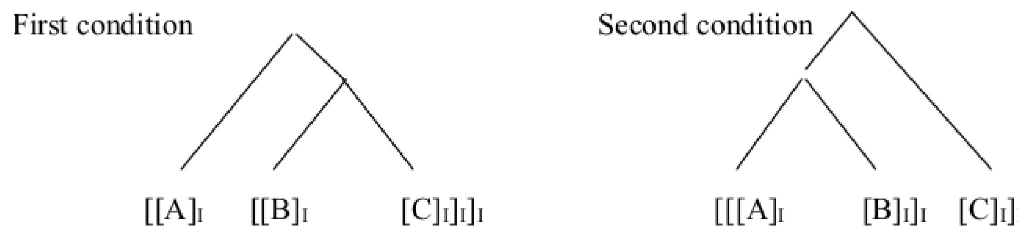
1. Moving left-to-right, in each case the following item is not embedded, but added on to the previous item, with right-branching it is top-down, with left-branching it is bottom up.
2. Left-branching and right-branching grammars describe the same terminal strings as the corresponding finite state automaton,and require only <u>finite memory</u> and <u>linear processing time</u>, unlike centre-embedding grammars.



Question from Lecture 2

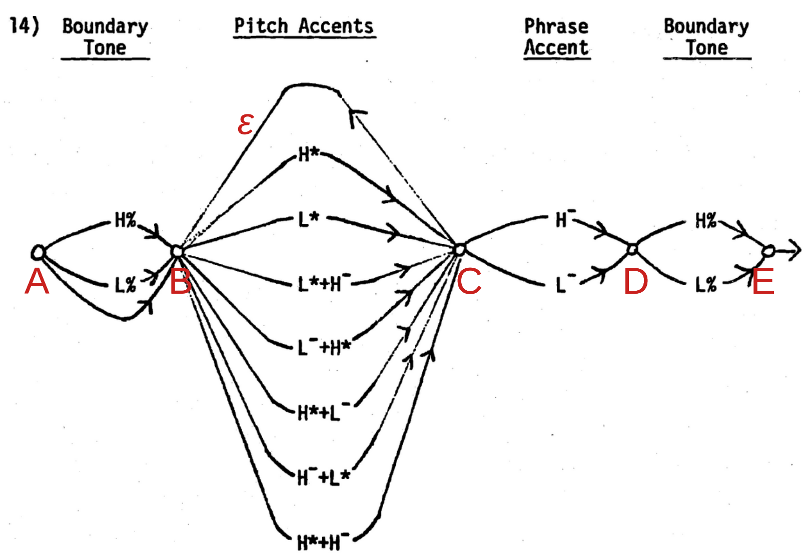
Q: Why is a right-branching structure not centre-embedding, even if the nodes have the same label, like in the example of Féry?

“I assume a recursive structure: All sentences are i-phrases, the grouping of two sentences is also an i-phrase, and the whole utterance is an i-phrase as well.”



A: The labelling decision is internal to the theory, which is not very clear. Incidentally, the structures violate the Strict Level Hypothesis (but this does not affect the argument):

1. Moving left-to-right, in each case the following item is not embedded, but added on to the previous item, with right-branching it is top-down, with left-branching it is bottom up.
2. Left-branching and right-branching grammars describe the same terminal strings as the corresponding finite state automaton,and require only <u>finite memory</u> and <u>linear processing time</u>, unlike centre-embedding grammars.



Equivalent right-branching Type 3 grammar:

A → ({ H%, L% }) B
A → ε B

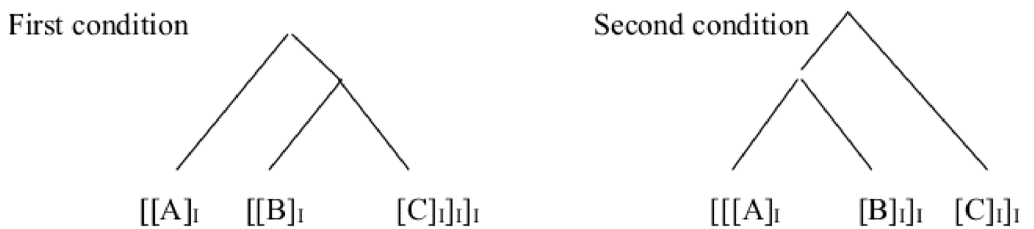
B → { H*, L*, L*H-, L-+H*, H*+L-, H-+L*, H*+H* } C
C → { H-, L- } D
C → ε B

D → { H%, L% }

Question from Lecture 2

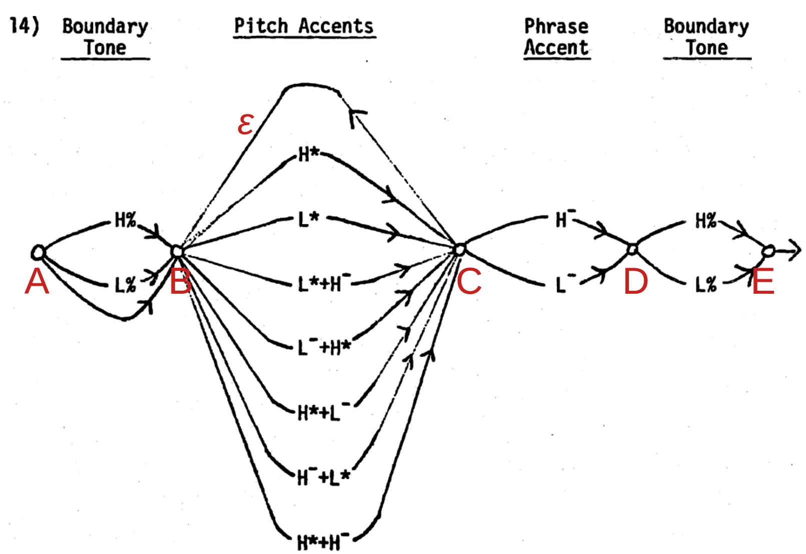
Q: Why is a right-branching structure not centre-embedding, even if the nodes have the same label, like in the example of Féry?

"I assume a recursive structure: All sentences are i-phrases, the grouping of two sentences is also an i-phrase, and the whole utterance is an i-phrase as well."



A: The labelling decision is internal to the theory, which is not very clear. Incidentally, the structures violate the Strict Level Hypothesis (but this does not affect the argument):

1. Moving left-to-right, in each case the following item is not embedded, but added on to the previous item, with right-branching it is top-down, with left-branching it is bottom up.
2. Left-branching and right-branching grammars describe the same terminal strings as the corresponding finite state automaton,and require only <u>finite memory</u> and <u>linear processing time</u>, unlike centre-embedding grammars.



Alternatively as a generate-and-test search system:

Grammar: infinite set of sequences from finite lexicon:

GEN = ($\{H\%_1, L\%_1\} \cup \{H^*, L^*, L^*H\text{-}, L\text{-}{+}H^*, H^*{+}L\text{-}, H\text{-}{+}L^*, H^*{+}H\text{-}\} \cup \{H\text{-}, L\text{-}\} \cup \{H\%_2, L\%_2\} \cup \{\varepsilon\}$)*

Finite constraint lexicon to evaluate for accepted subset:

CON = { $o(\langle A,B\rangle) \in \{H\%_1, L\%_1, \varepsilon\}$

$o(\langle B,C\rangle) \in \{H^*, L^*, L^*H\text{-}, L\text{-}{+}H^*, H^*{+}L\text{-}, H\text{-}{+}L^*, H^*{+}H\text{-}, H\text{-}, L\text{-}\}$

$o(\langle C,B\rangle) \in \{\varepsilon\}$

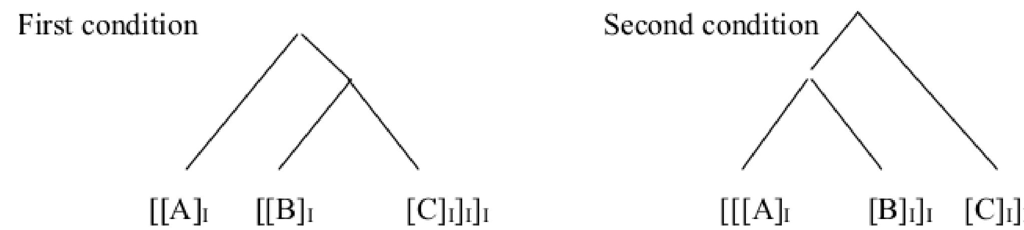
$o(\langle C,D\rangle) \in \{H\text{-}, L\text{-}\}$

$o(\langle D,E\rangle) \in \{H\%_2, L\%_2\}$ }

Question from Lecture 2

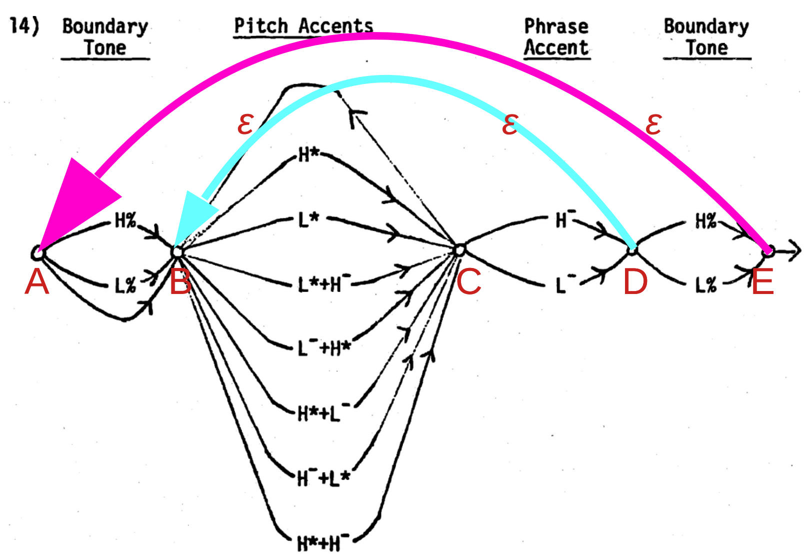
Q: Why is a right-branching structure not centre-embedding, even if the nodes have the same label, like in the example of Féry?

“I assume a recursive structure: All sentences are i-phrases, the grouping of two sentences is also an i-phrase, and the whole utterance is an i-phrase as well.”



A: The labelling decision is internal to the theory, which is not very clear. Incidentally, the structures violate the Strict Level Hypothesis (but this does not affect the argument):

1. Moving left-to-right, in each case the following item is not embedded, but added on to the previous item, with right-branching it is top-down, with left-branching it is bottom up.
2. Left-branching and right-branching grammars describe the same terminal strings as the corresponding finite state automaton,and require only <u>finite memory</u> and <u>linear processing time</u>, unlike centre-embedding grammars.



Equivalent right-branching Type 3 grammar:

A → ({ H%, L% }) B
A → ε B

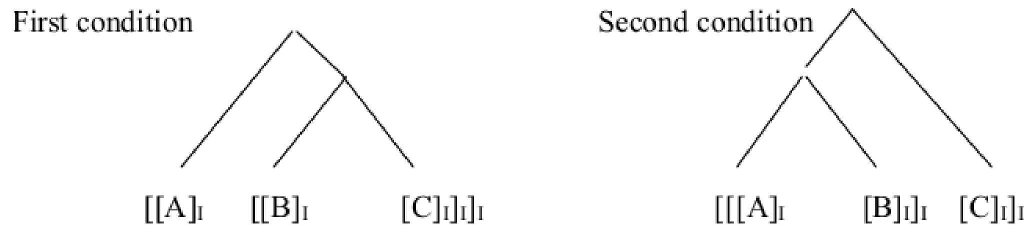
B → { H*, L*, L*H-, L-+H*, H*+L-, H-+L*, H*+H* } C
C → { H-, L- } D
C → ε B

D → { H%, L% }
D → ε B
E → ε A

Question from Lecture 2

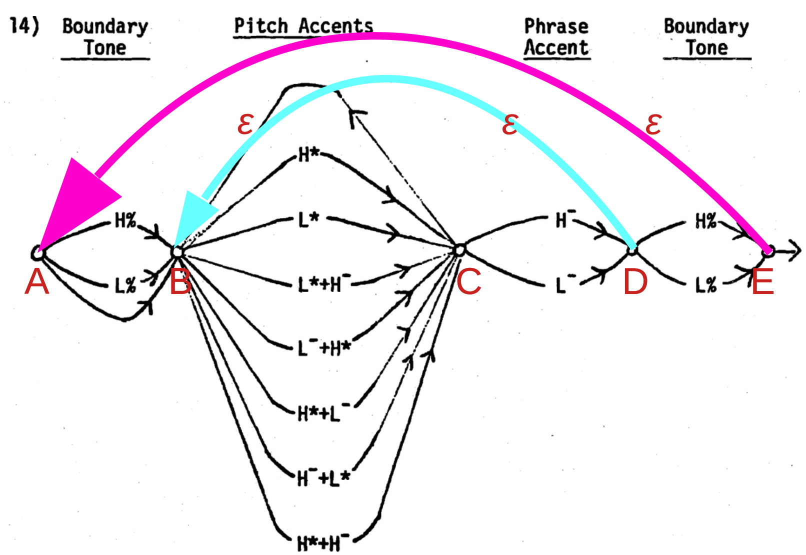
Q: Why is a right-branching structure not centre-embedding, even if the nodes have the same label, like in the example of Féry?

“I assume a recursive structure: All sentences are i-phrases, the grouping of two sentences is also an i-phrase, and the whole utterance is an i-phrase as well.”



A: The labelling decision is internal to the theory, which is not very clear. Incidentally, the structures violate the Strict Level Hypothesis (but this does not affect the argument):

1. Moving left-to-right, in each case the following item is not embedded, but added on to the previous item, with right-branching it is top-down, with left-branching it is bottom up.
2. Left-branching and right-branching grammars describe the same terminal strings as the corresponding finite state automaton,and require only <u>finite memory</u> and <u>linear processing time</u>, unlike centre-embedding grammars.



Equivalent right-branching Type 3 grammar:

A → ({ H%, L% }) B
A → ε B

B → { H*, L*, L*H-, L-+H*, H*+L-, H-+L*, H*+H* } C
C → { H-, L- } D
C → ε B

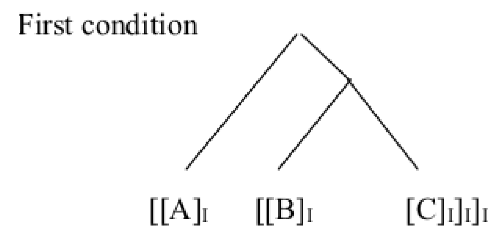
D → { H%, L% }
D → ε B
E → ε A

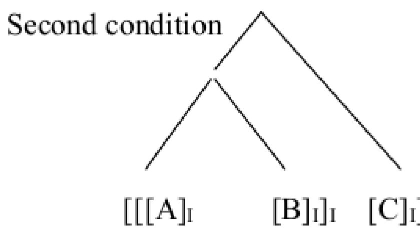
Global contour:
downtrend: declination, downstep, level, inclination, upstep
Three iterations:
downtrend loop
reset loop
restart loop

Question from Lecture 2

Q: Why is a right-branching structure not centre-embedding, even if the nodes have the same label, like in the example of Féry?

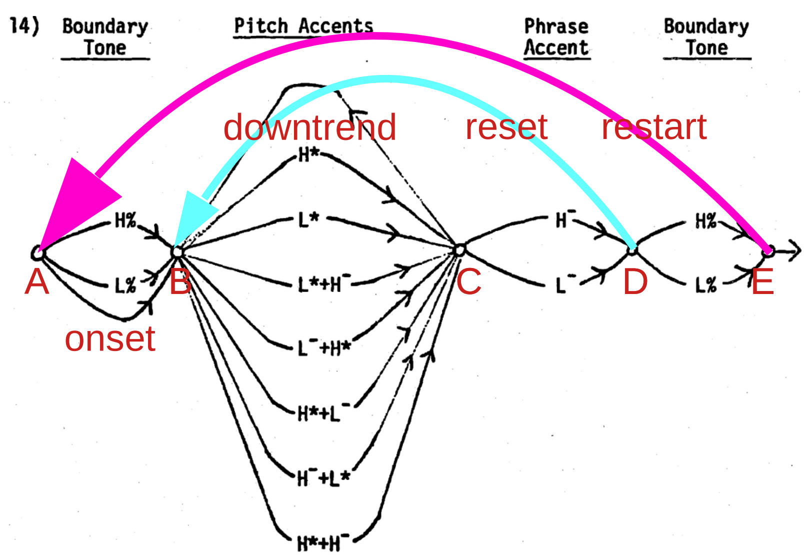
“I assume a recursive structure: All sentences are i-phrases, the grouping of two sentences is also an i-phrase, and the whole utterance is an i-phrase as well.”





A: The labelling decision is internal to the theory, which is not very clear. Incidentally, the structures violate the Strict Level Hypothesis (but this does not affect the argument):

1. Moving left-to-right, in each case the following item is not embedded, but added on to the previous item, with right-branching it is top-down, with left-branching it is bottom up.
2. Left-branching and right-branching grammars describe the same terminal strings as the corresponding finite state automaton,and require only <u>finite memory</u> and <u>linear processing time</u>, unlike centre-embedding grammars.



Equivalent right-branching Type 3 grammar:

A → ({ H%, L% }) B
A → onset B

B → { H*, L*, L*H-, L-+H*, H*+L-, H-+L*, H*+H* } C
C → { H-, L- } D
C → downtrend B

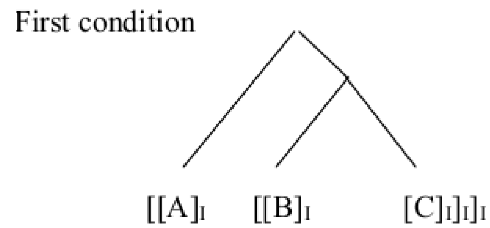
D → { H%, L% }
D → reset B
E → restart A

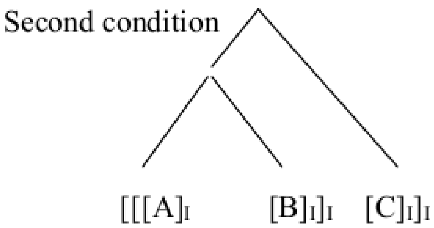
Global contour:
downtrend: declination, downstep, level, inclination, upstep
Three iterations:
downtrend looop
reset loop
restart loop

Question from Lecture 2

Q: Why is a right-branching structure not centre-embedding, even if the nodes have the same label, like in the example of Féry?

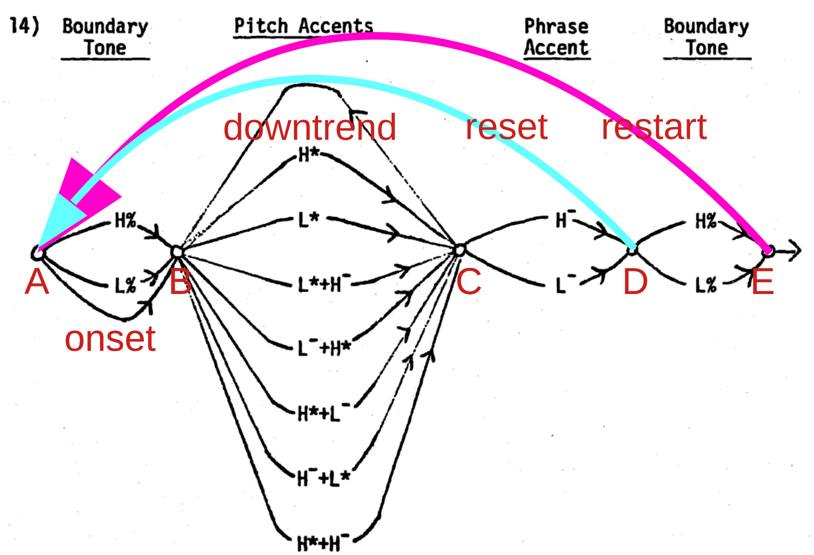
“I assume a recursive structure: All sentences are i-phrases, the grouping of two sentences is also an i-phrase, and the whole utterance is an i-phrase as well.”

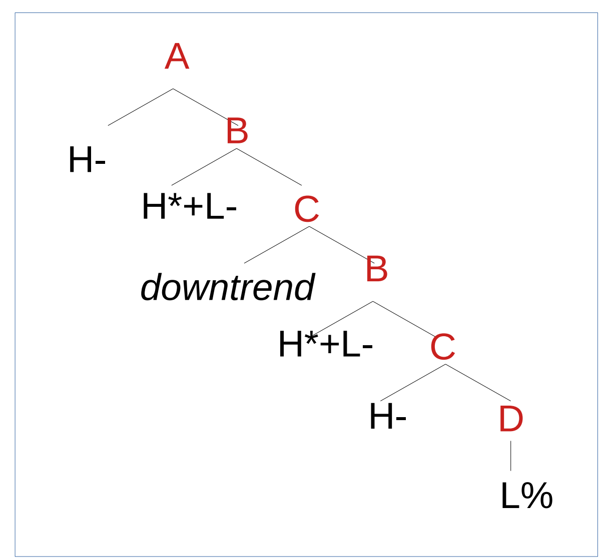




A: The labelling decision is internal to the theory, which is not very clear. Incidentally, the structures violate the Strict Level Hypothesis (but this does not affect the argument):

1. Moving left-to-right, in each case the following item is not embedded, but added on to the previous item, with right-branching it is top-down, with left-branching it is bottom up.
2. Left-branching and right-branching grammars describe the same terminal strings as the corresponding finite state automaton,and require only <u>finite memory</u> and <u>linear processing time</u>, unlike centre-embedding grammars.

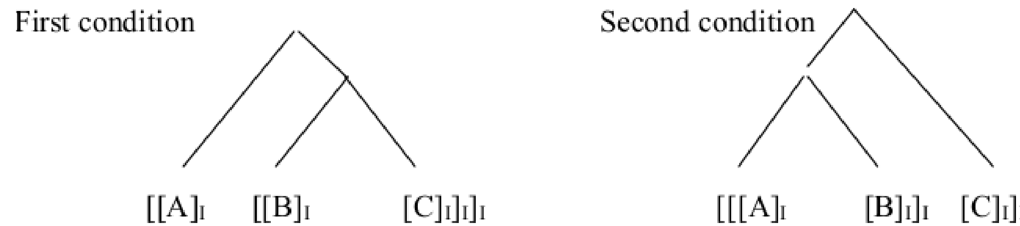




Question from Lecture 2

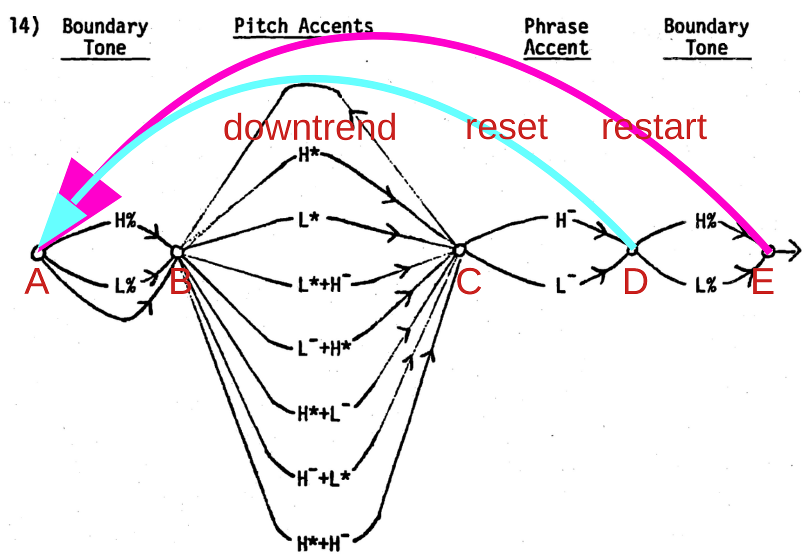
Q: Why is a right-branching structure not centre-embedding, even if the nodes have the same label, like in the example of Féry?

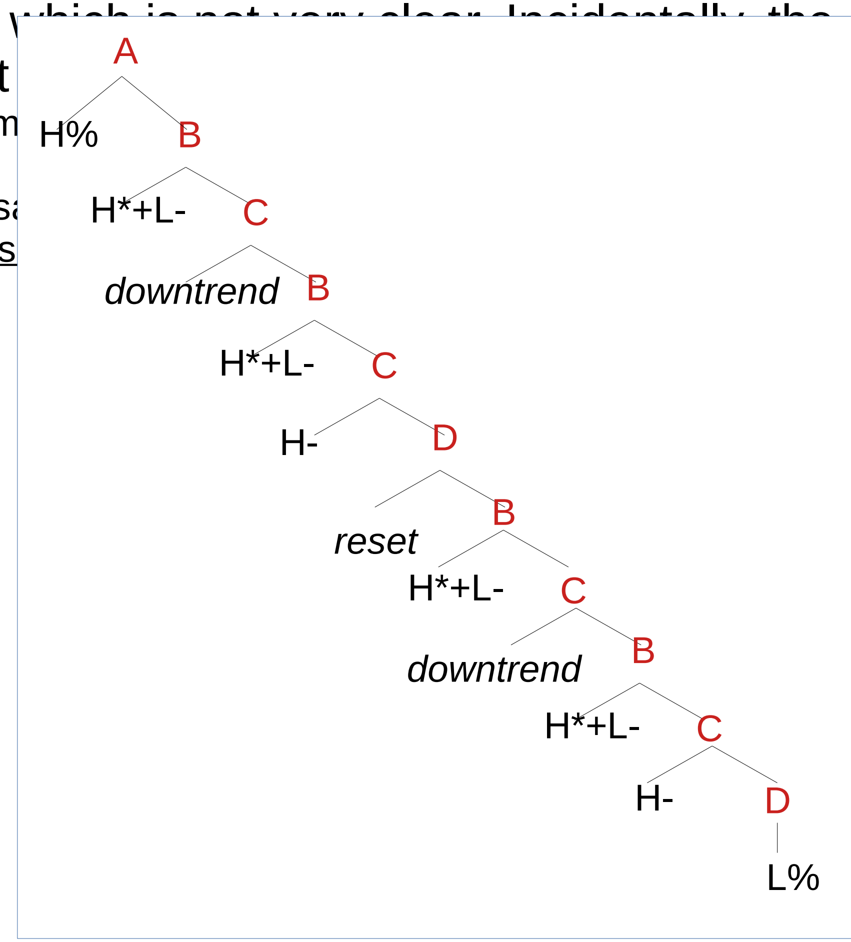
“I assume a recursive structure: All sentences are i-phrases, the grouping of two sentences is also an i-phrase, and the whole utterance is an i-phrase as well.”



A: The labelling decision is internal to the theory, …
structures violate the Strict Level Hypothesis (but …

1. Moving left-to-right, in each case the following item is not em… …ht-branching it is top-down, with left-branching it is bottom up.
2. Left-branching and right-branching grammars describe the s… …te automaton,and require only <u>finite memory</u> and <u>linear proces</u>…





Sounds of Prosody

RFT: Rhythm Formant Theory
RFA: Rhythm Formant Analysis

How to do it:

1. Algorithms
2. Case studies

Code, articles: https://github.com/dafyddg/RFA

Timing of Speech

Music and speech depend on the temporal constraints given by the human body:

- Body rhythm timing:
  - approximately one main movement per second:
    - foot stamping, running, walking
    - hand clapping, head nodding
    - chewing, sucking
    - hand-shaking, intimate interaction
    - syllable and word sequences

Different speech rhythms:

- rhythms of syllable constituents (C, V)
- Rhythms of syllable types (strong, weak; stressed, unstressed)
- Rhythms of words or feet, phrases, sentences
- Rhythms of discourse episodes

Timing of Speech

Music and speech depend on the temporal constraints given by the human body:

- Body rhythm timing:
  - approximately one main movement per second:
    - foot stamping, running, walking
    - hand clapping, head nodding
    - chewing, sucking
    - hand-shaking, intimate interaction
    - syllable and word sequences

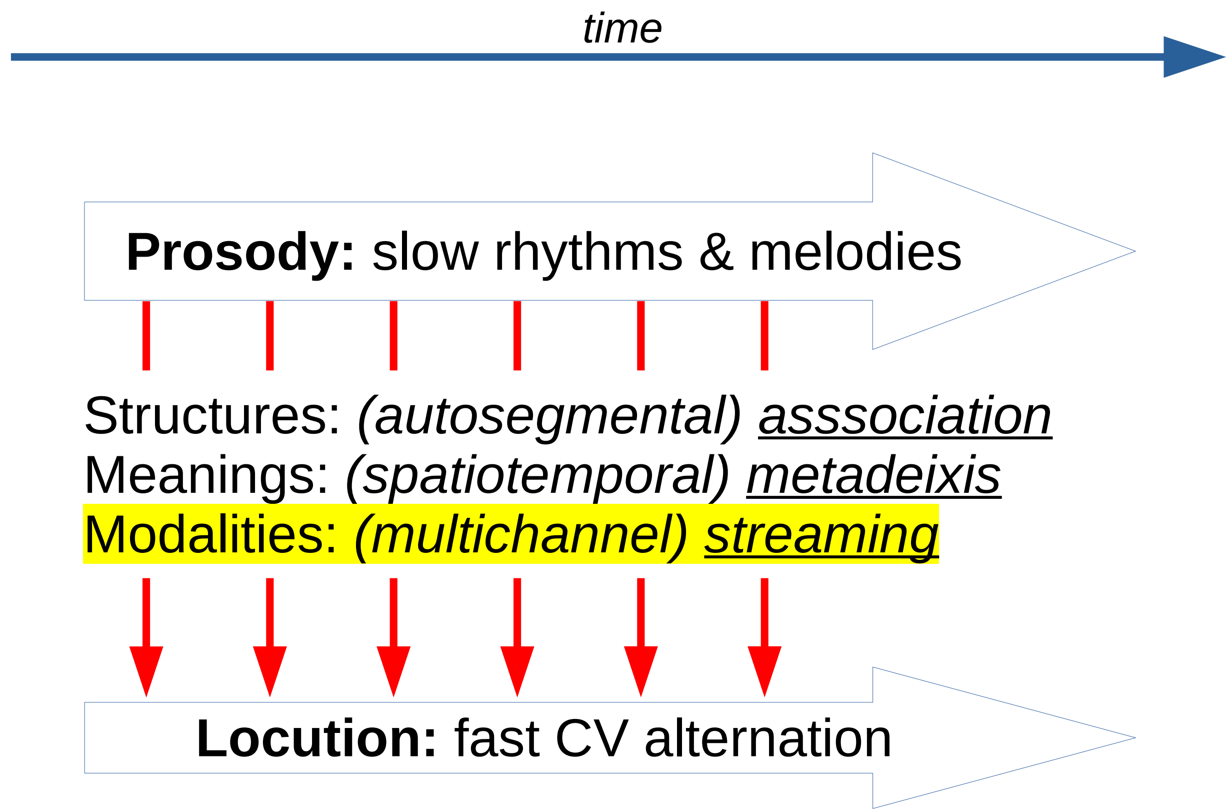
Different speech rhythms:

- rhythms of syllable constituents (C, V)
- Rhythms of syllable types (strong, weak; stressed, unstressed)
- Rhythms of words or feet, phrases, sentences
- Rhythms of discourse episodes

The association of the 'Rhythm hierarchy' with the 'Prosodic Hierarchy' is flexible and depends on

- semantic constraints (e.g. contrast)
- pragmatic constraints (e.g. emphasis)

Metalocutionary Theory of Prosodic Function



Time Types:

cloud time (intuitive everyday ‘real’ time)

clock time (Newtonian time, universal quantitative time)

rubber time (Aristotelian time: Event Phonology, tree structures)

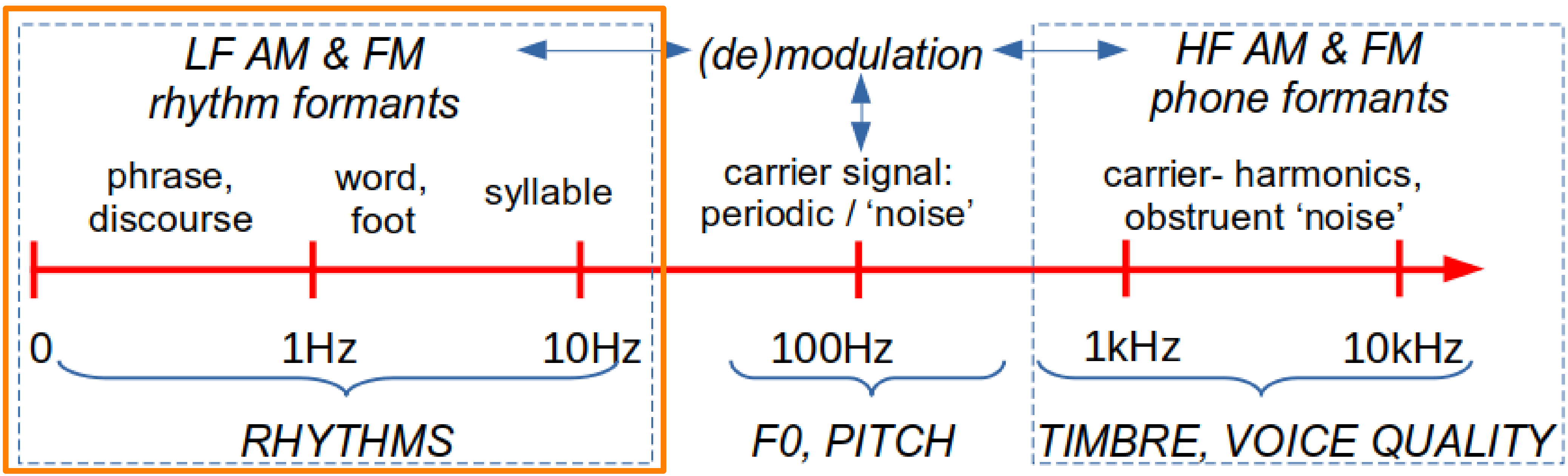
categorial time (abstract time points: duration contrast; context)

The Modulation Code: Time and the Frequency Scale

Low frequencies:
rhythm

Mid frequencies:
rhythm

High frequencies:
consonants and vowels



Low Frequency
AM and FM modulations

High Frequency
AM and FM modulations

Theory and practice of Rhythm Analysis: RFT and RFA

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; the higher values are then shown as red dots in the RFA spectrum.
- The spectra of different recordings are
  - compared using standard distance metrics
    - and represented as distance maps,
    - also (1) hierarchically clustered using (2) standard clustering criteria and represented as dendrograms.

Thanks to Dr. Liu Huangmei, for suggesting the term ‘formant’ in this context.

Rhythm Formant Analysis: implementation

RFA implementation (GitHub repository):

The applications included in the set are intended for experiments based on the low frequency long-term AM and FM spectrum:

The set of demonstration applications can be freely adapted and modified to suit your own needs.

RFA directory:

- Articles
- IICBP2022-slides
- LittleHelpers
- README.1st
- README.md
- RFA_multiple_signal_processing
- RFA_single_signal_processing

Code, articles: https://github.com/dafyddg/RFA

Rhythm Formant Analysis: implementation

RFA implementation (GitHub repository):

The applications included in the set are intended for experiments based on the low frequency long-term AM and FM spectrum:

The set of demonstration applications can be freely adapted and modified to suit your own needs.

```
RFA directory:
    Articles
    IICBP2022-slides
    LittleHelpers
    README.1st
    README.md
    RFA_multiple_signal_processing
    RFA_single_signal_processing
```

```
RFA_single_signal_processing:
    DATA
        English_male_MLK01.wav
        English_male_one-to-seven.wav
        English_male_one-to-thirty_16k.wav
        Female_English_German:
            RT_E1.wav
            RT_E2.wav
            RT_E3.wav
            RT_G1.wav
            RT_G2.wav
            RT_G3.wav
        Putonghua_female_one-to-seven.wav
        sine-200x5x6.wav
    FIGURES
    module_dendrogram.py
    module_F0.py
    module_spectrogram.py
    rfa_single_conf.py
    rfa_single.py
```

Code, articles: https://github.com/dafyddg/RFA

Rhythm Formant Analysis: implementation

RFA implementation (GitHub repository):

The applications included in the set are intended for experiments based on the low frequency long-term AM and FM spectrum:

The MIT licence is used, so demonstration applications can be freely adapted and modified to suit your needs (with acknowledgments).

RFA
- Articles
- IICBP2022-slides
- LittleHelpers
- README.1st
- README.md
- RFA_multiple_signal_processing
- RFA_single_signal_processing

RFA_single_signal_processing
- DATA
  - English_male_MLK01.wav
  - English_male_one-to-seven.wav
  - English_male_one-to-thirty_16k.wav
  - Female_English_German:
    - RT_E1.wav
    - RT_E2.wav
    - RT_E3.wav
    - RT_G1.wav
    - RT_G2.wav
    - RT_G3.wav
  - Putonghua_female_one-to-seven.wav
  - sine-200x5x6.wav
- FIGURES
- module_dendrogram.py
- module_F0.py
- module_spectrogram.py
- rfa_single_conf.py
- rfa_single.py

RFA_multiple_signal_processing
- CSV
- DATA
  - Female_English_German
    - RT_E1.wav
    - RT_E2.wav
    - RT_E3.wav
    - RT_G1.wav
    - RT_G2.wav
    - RT_G3.wav
- DENDRO
- GRAPHVIZ
- module_F0.py
- module_spectrogram.py
- numdistnetdendro_conf.py
- numdistnetdendro.py
- rfa_mult_conf.py
- rfa_mult.py

Code, articles: https://github.com/dafyddg/RFA

Empirical Background: Phonetic Domain, Phase Cycle

Aims of this part of the talk

Overview of Rhythm Formants as low frequency modulations of speech

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

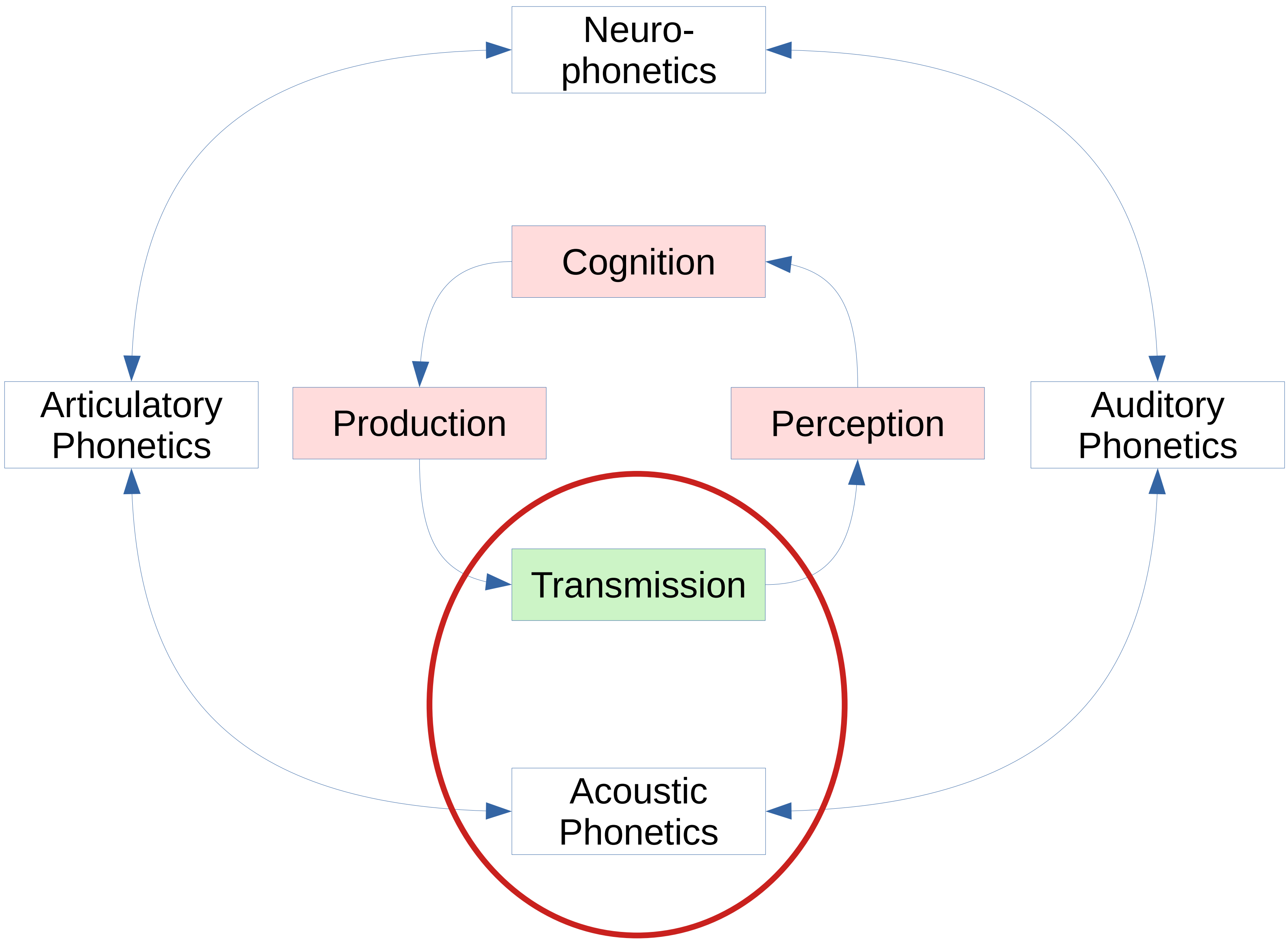
- AM and FM demodulation
- spectral analysis
- comparing spectra from different recordings of comparable data using distance tables, distance maps and distance based clustering

- 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.

SPOILER

It's easier than you think!

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:
- So let's do something in this talk to correct this:
  - Modulation Theory:
    - Amplitude Modulation (AM)
    - Frequency Modulation (FM)
  - a 'do-it-yourself' approach to phonetic software
    - an alternative, for some purposes, to using ready-made off-the-shelf applications
  - you can download demonstration examples in Python
    BUT: no programming experience is required

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.
- The spectra of different recordings are compared using
  - standard distance metrics, then
    - represented as distance maps, and
    - hierarchically clustered using standard clustering criteria, and represented as dendrograms.

Modulation Theory

Formal background: Modulation Theory

carrier signal modulated with information signal

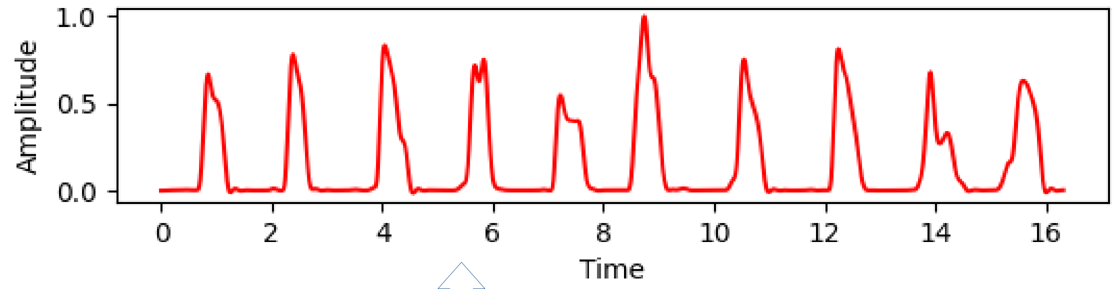
1) carrier signal with *frequency modulation* signal (**FM**)
   tone, pitch accent, intonation → larynx

2) carrier signal with *amplitude modulation* signal (**AM**)
   consonants, vowels, syllables → oral & nasal cavities

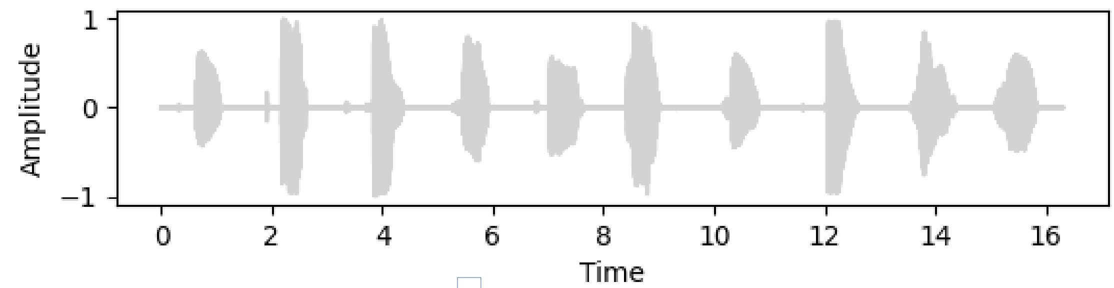
3) **speech:** carrier signal with AM and FM simultaneously

AM and FM Demodulation

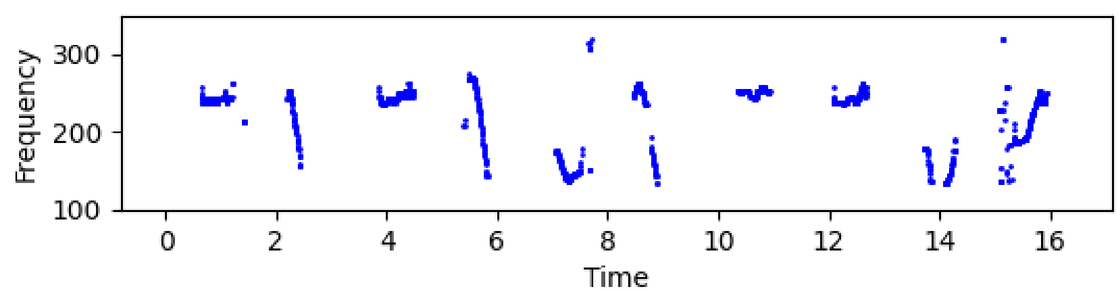
AM envelope demodulation:
- phonetics:
  amplitude curve, syllable,
  stress-accent
- phonology:
  sonority curve, syllables, stress



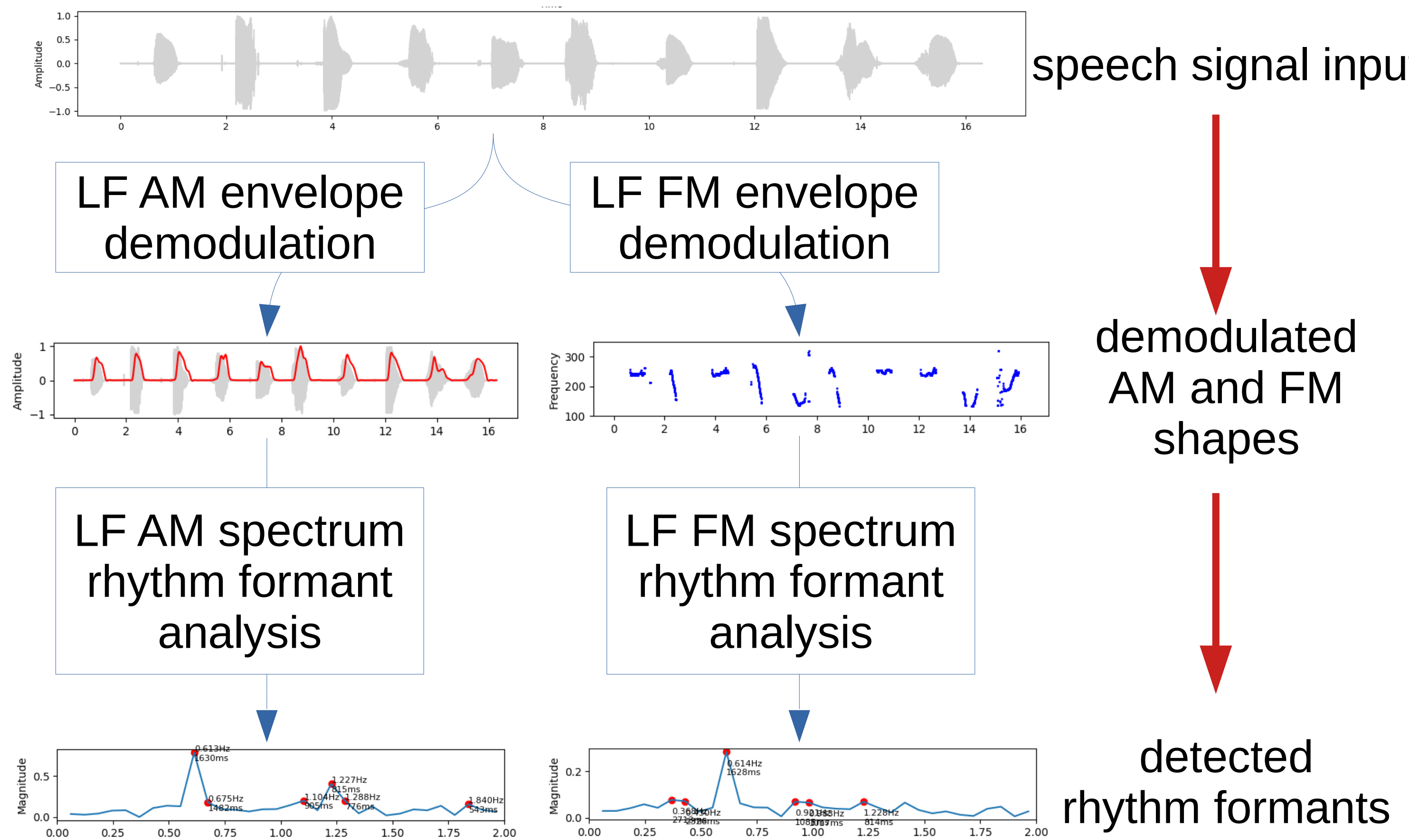
Modulated carrier signal



FM envelope demodulation:
- phonetics:
  F0, pitch track
- phonology:
  tones, pitch accents, intonation



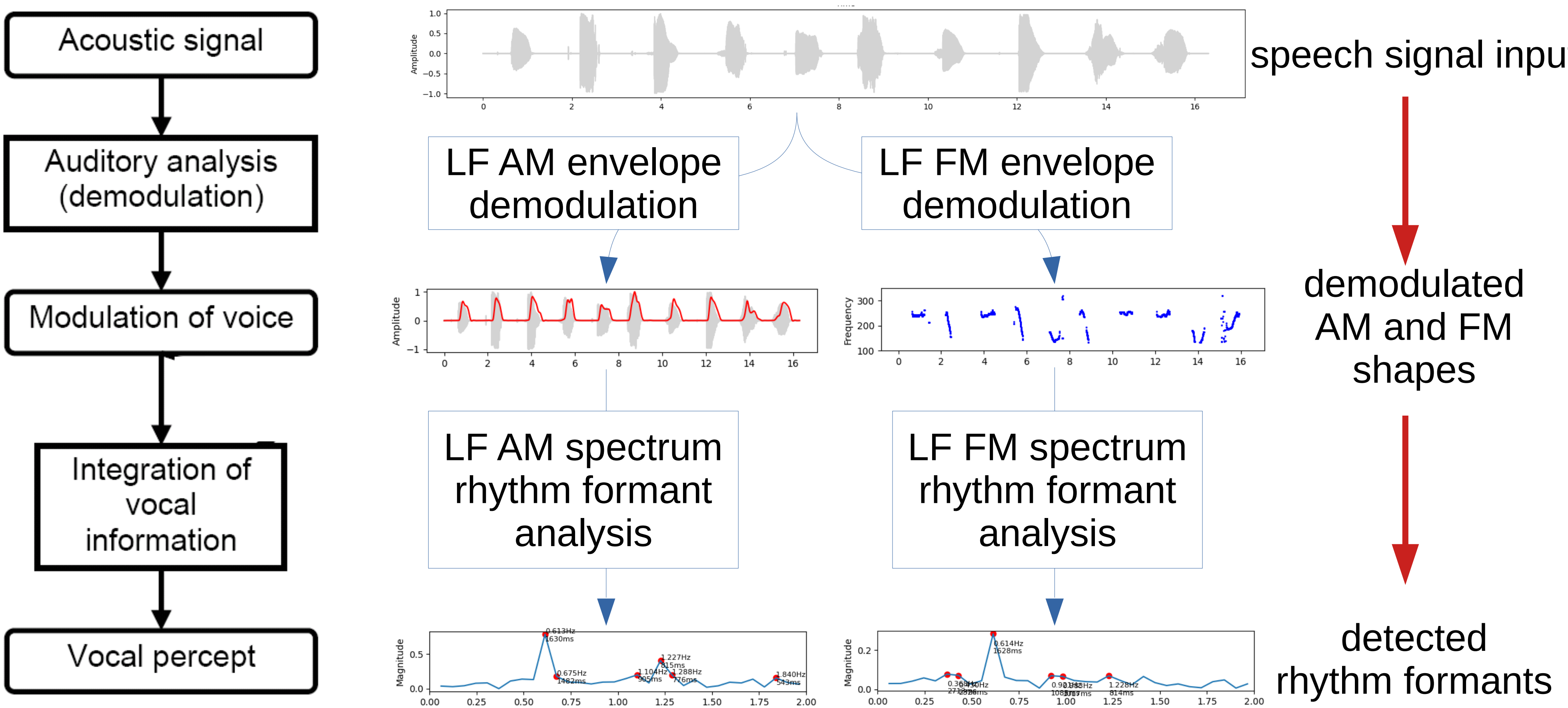
AM and FM demodulation and detection of rhythm



Hartmut Traunmüller (1994) "Conventional, biological, and environmental factors in speech communication: A modulation theory" Phonetica 51: 170-183. doi (Also in PERILUS XVIII: 92-102.)

Hartmut Traunmüller (2007) "Demodulation, mirror neurons and audiovisual perception nullify the motor theory" Contr. to Fonetik 2007, TMH-QPSR 50: 17-20. Detpt. of Speech, Music and Hearing, Royal Inst. of Technology, Stockholm.

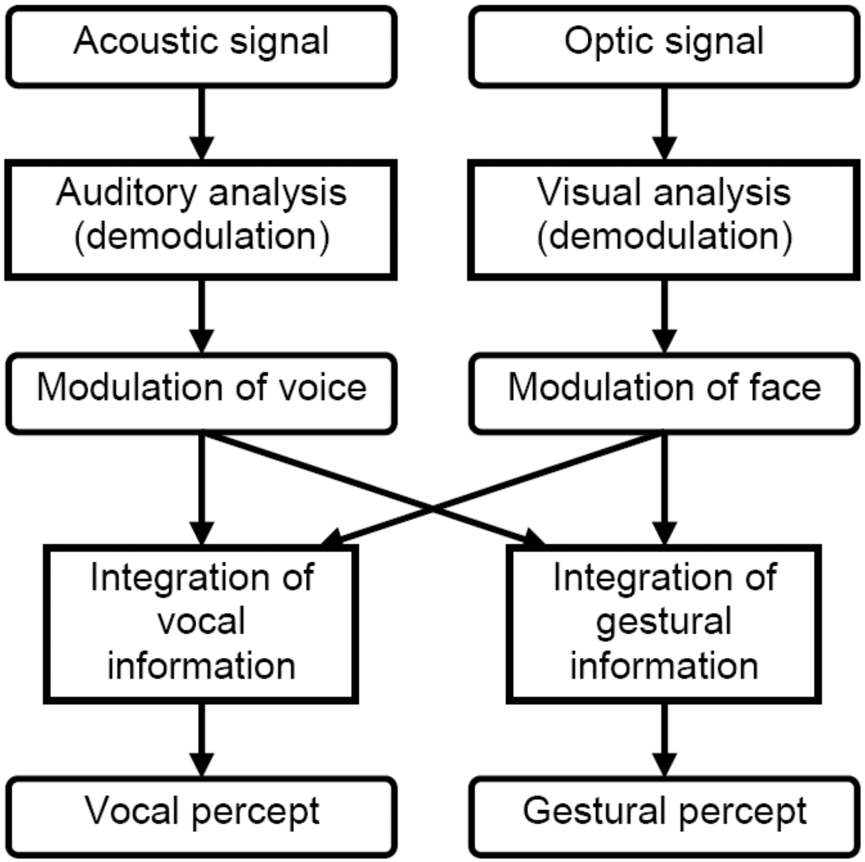
Comparison with Traunmüller's demodulation model



Hartmut Traunmüller (1994) "Conventional, biological, and environmental factors in speech communication: A modulation theory" Phonetica 51: 170-183. doi (Also in PERILUS XVIII: 92-102.)

Hartmut Traunmüller (2007) "Demodulation, mirror neurons and audiovisual perception nullify the motor theory" Contr. to Fonetik 2007, TMH-QPSR 50: 17-20. Detpt. of Speech, Music and Hearing, Royal Inst. of Technology, Stockholm.

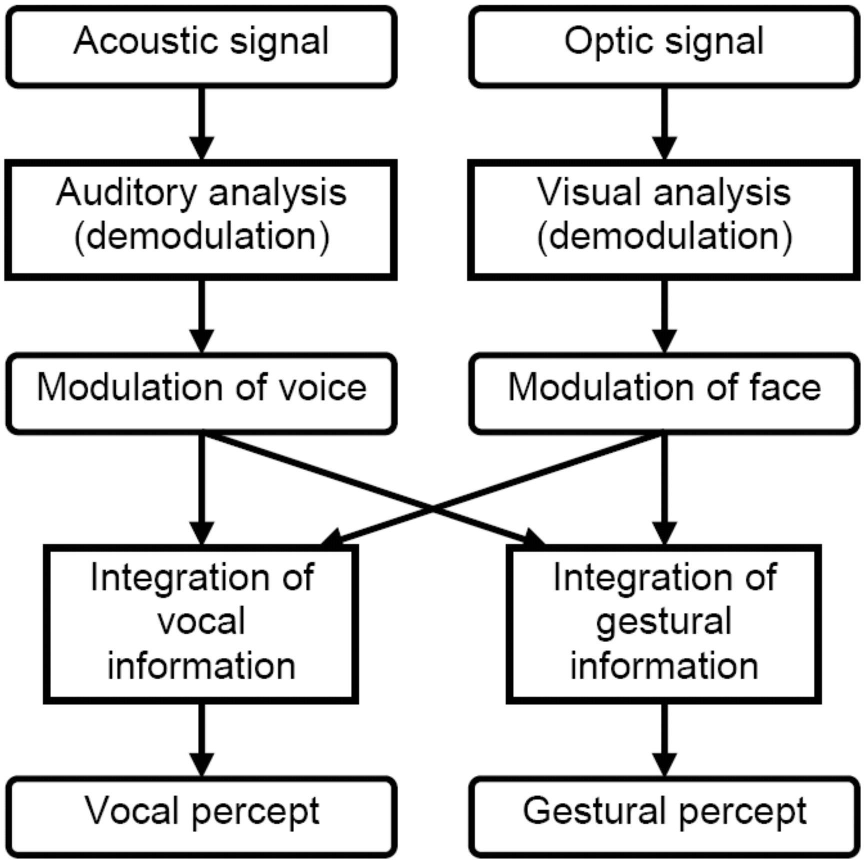
Traunmüller: audiovisual perception (2007)

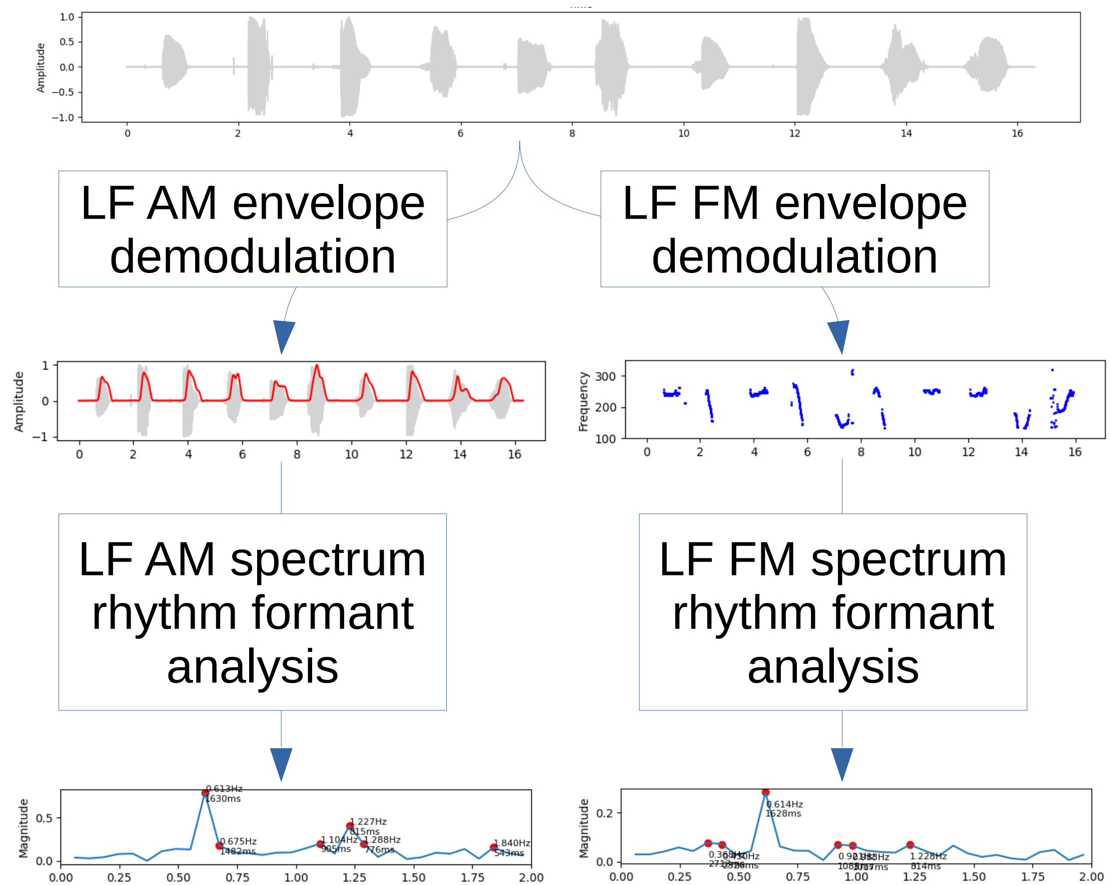


Hartmut Traunmüller (1994) "Conventional, biological, and environmental factors in speech communication: A modulation theory" Phonetica 51: 170-183. doi (Also in PERILUS XVIII: 92-102.)

Hartmut Traunmüller (2007) "Demodulation, mirror neurons and audiovisual perception nullify the motor theory" Contr. to Fonetik 2007, TMH-QPSR 50: 17-20. Detpt. of Speech, Music and Hearing, Royal Inst. of Technology, Stockholm.

Traunmüller: audiovisual perception (2007)



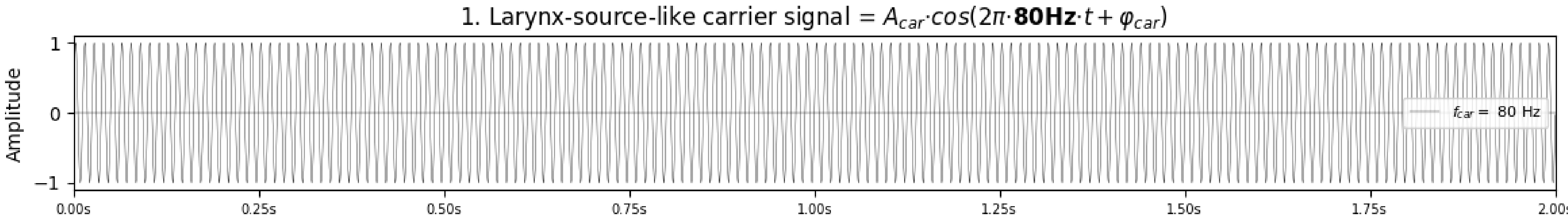


Hartmut Traunmüller (1994) "Conventional, biological, and environmental factors in speech communication: A modulation theory" Phonetica 51: 170-183. doi (Also in PERILUS XVIII: 92-102.)

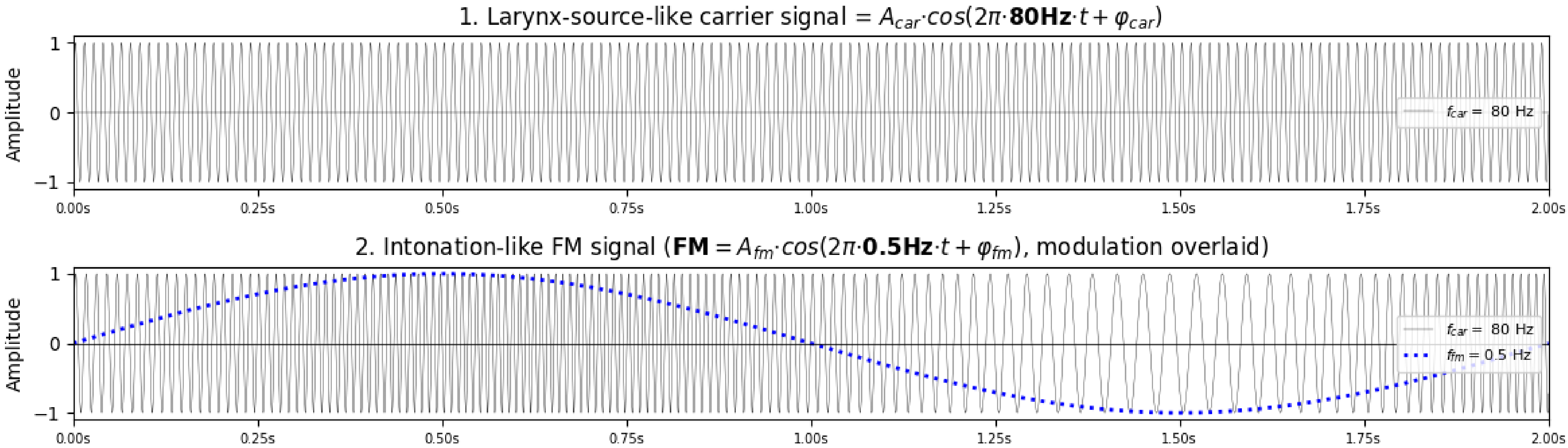
Hartmut Traunmüller (2007) "Demodulation, mirror neurons and audiovisual perception nullify the motor theory" Contr. to Fonetik 2007, TMH-QPSR 50: 17-20. Detpt. of Speech, Music and Hearing, Royal Inst. of Technology, Stockholm.

AM and FM modulation step by step

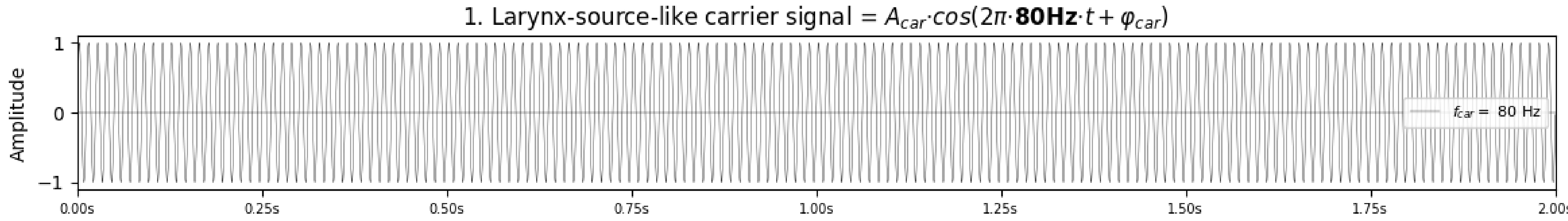
Modulation: carrier signal

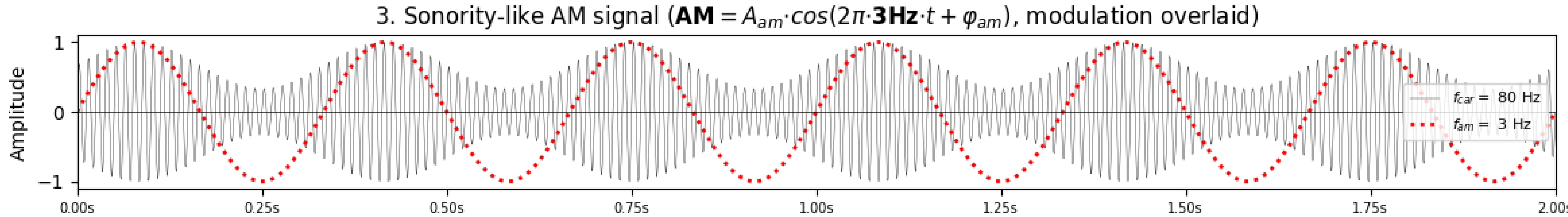


Modulation: FM signal with low frequency information

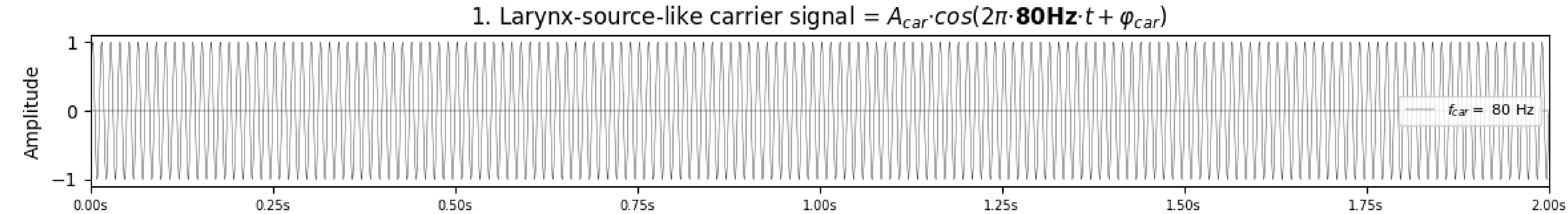


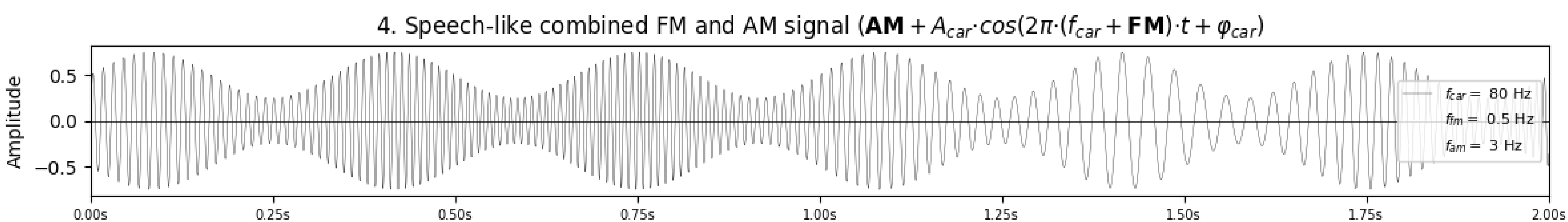
Modulation: AM signal with low frequency information





Modulation Theory





Demodulation and analysis procedures in RFA

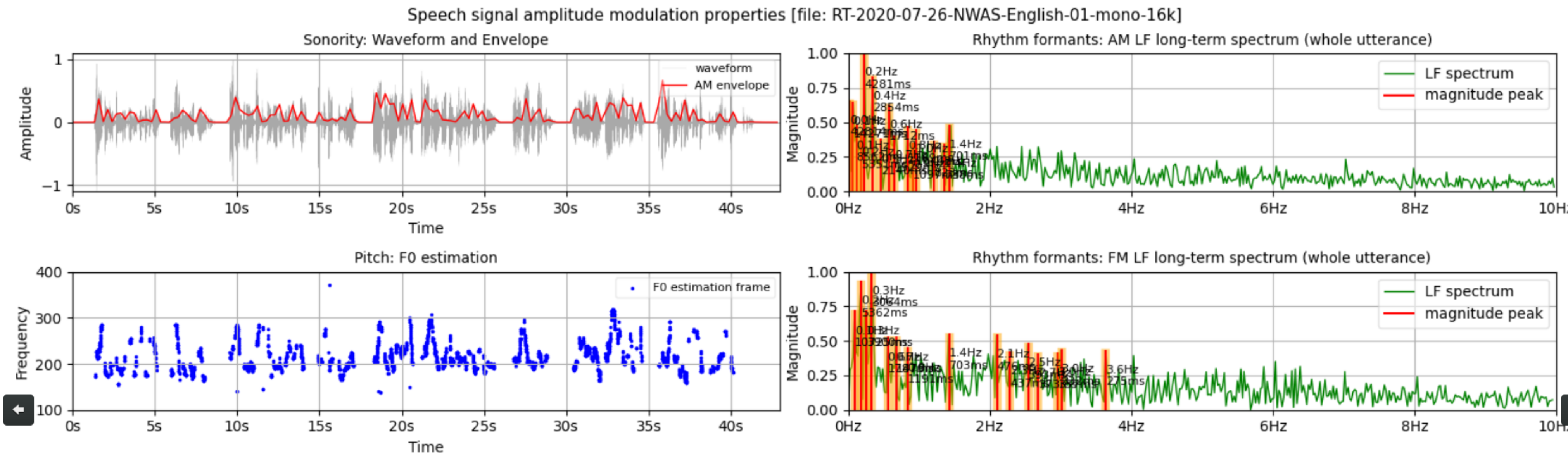
Demodulation and analysis procedures in RFA

- Time domain processing:
  - Envelope extraction
  - Fundamental frequency estimation (‘pitch’ extraction)
- Time domain to frequency domain transformation:
  - Spectral analysis
  - Spectrogram analysis
  - F0 estimation:
    - time domain procedures: zero-crossing count, autocorrelation (AC), average magnitude difference (AMDF)
    - frequency domain procedures: spectrum transformation and analysis
- 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

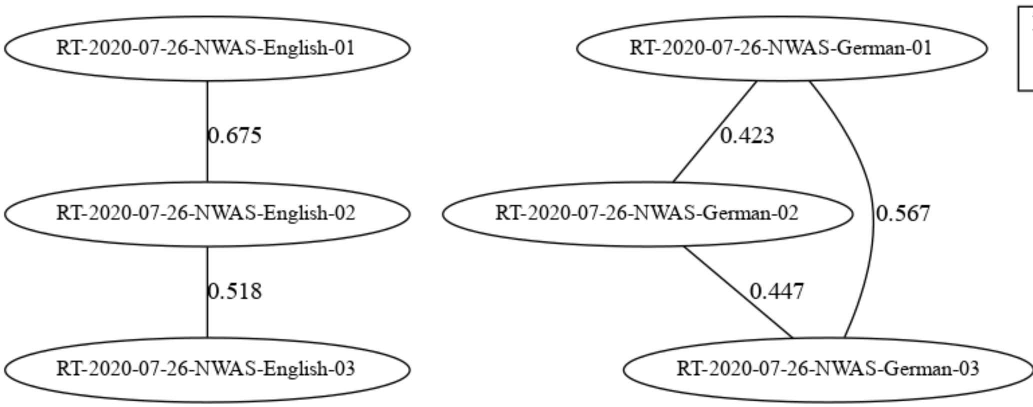
Demodulation and analysis: output examples

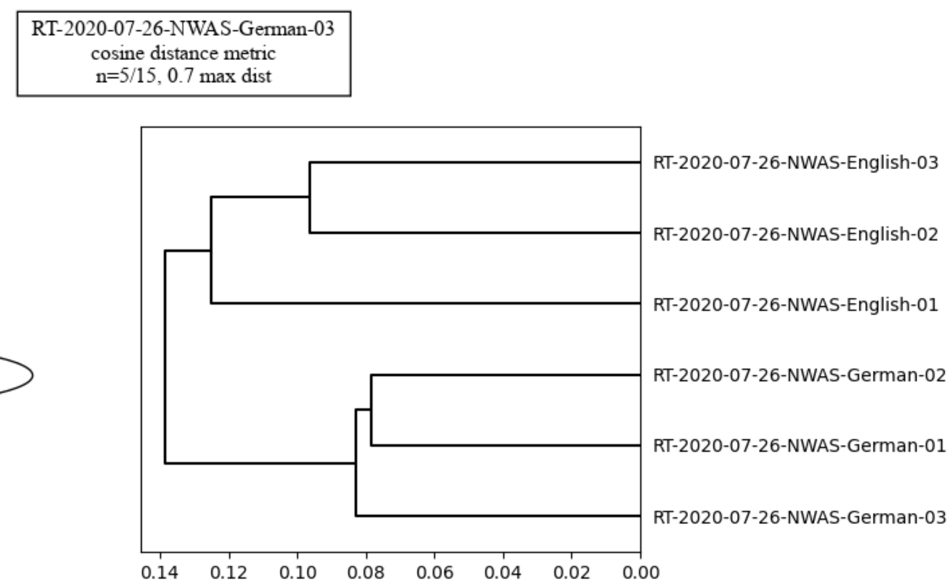
Example outputs

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



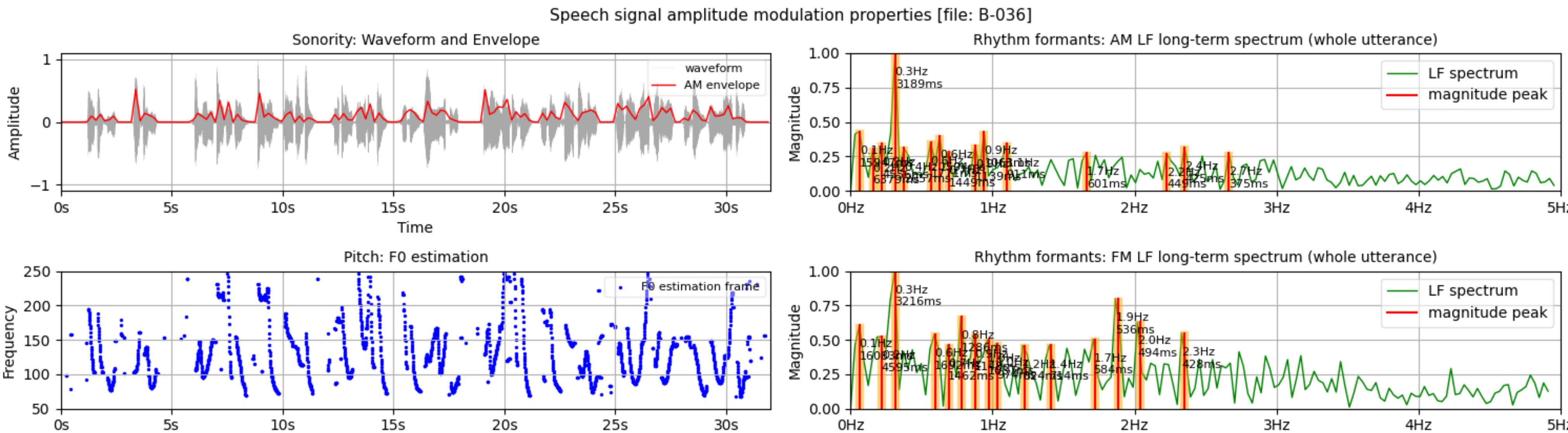
Similarity of readings: *The North Wind and the Sun*, bilingual in English and German





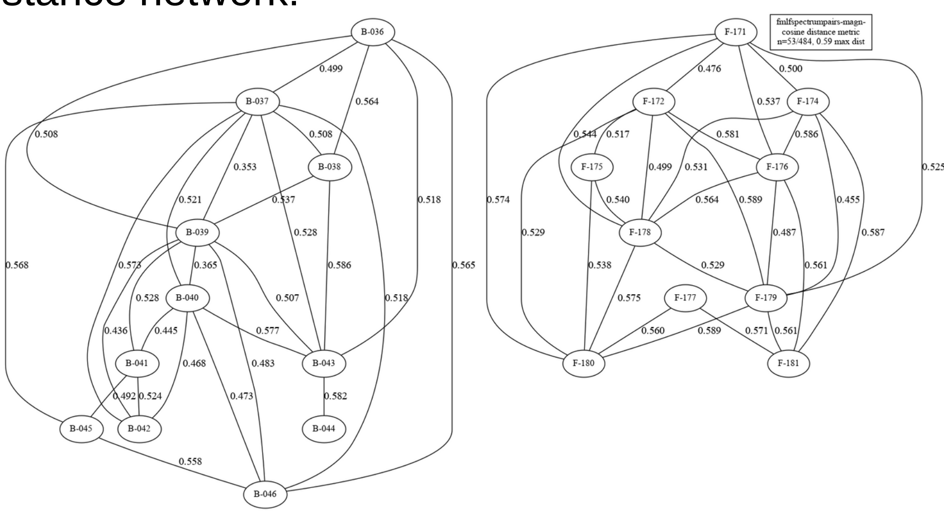
Example outputs

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

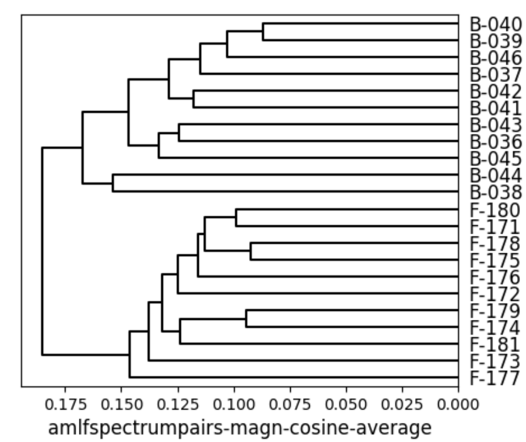


Comparing two styles of Tang dynasty poetry

Distance network:

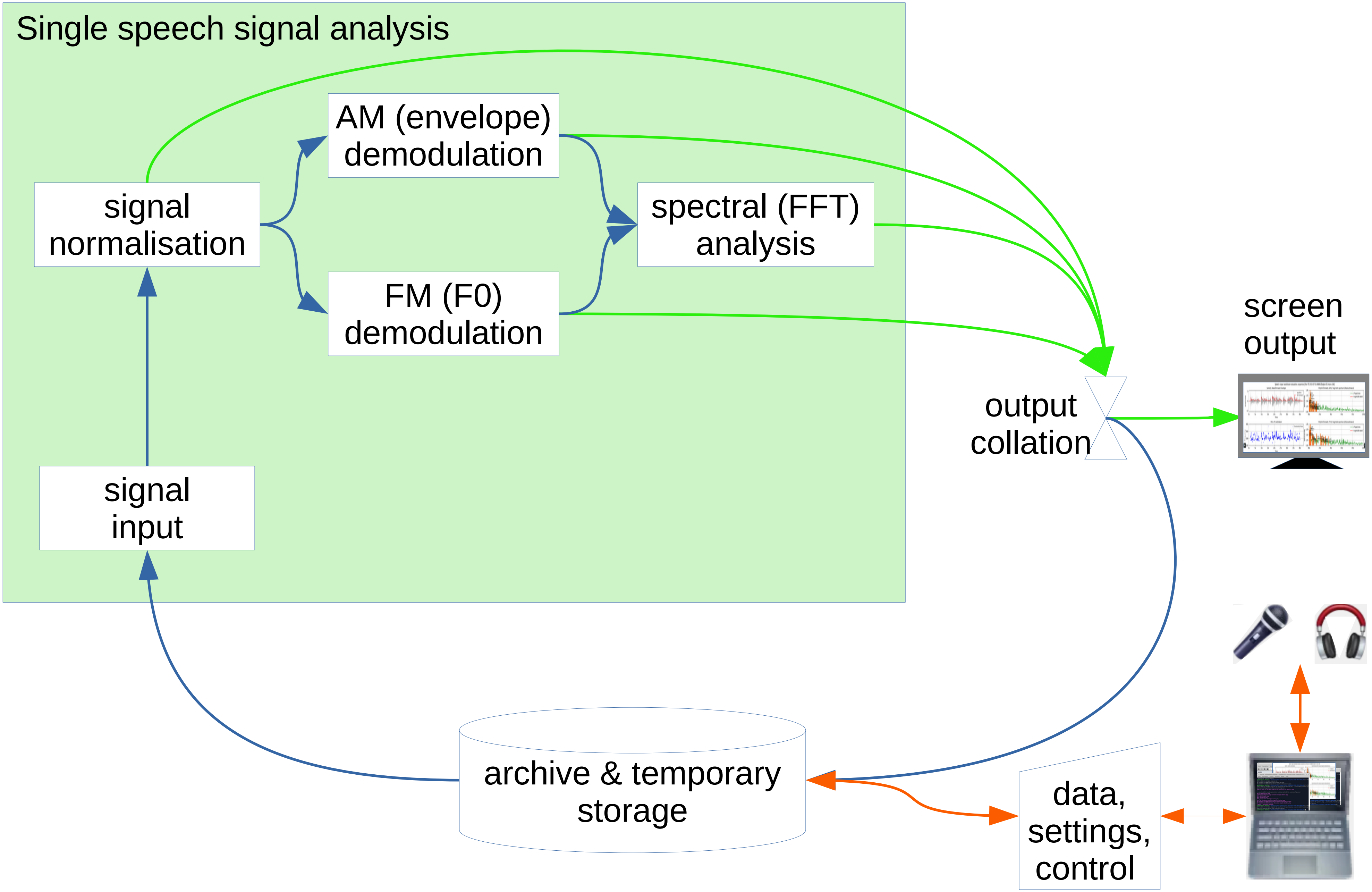


Hierarchical clustering::

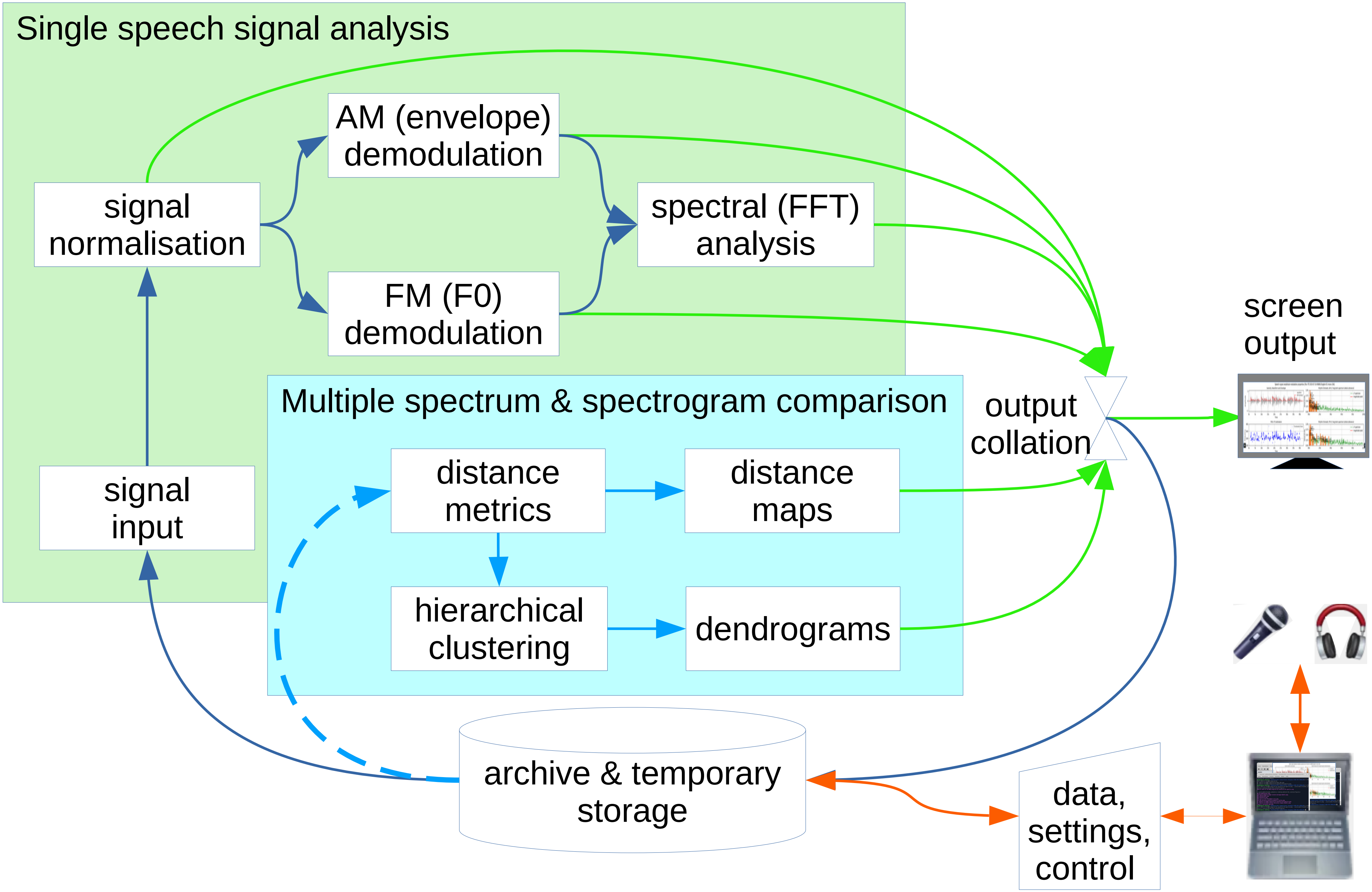


Demodulation and analysis: software design

Rhythm Formant Analysis Software Design: Data Flow



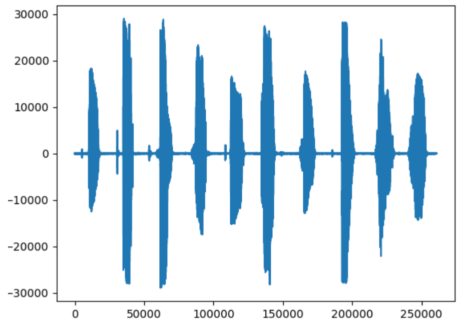
Rhythm Formant Analysis Software Design: Data Flow



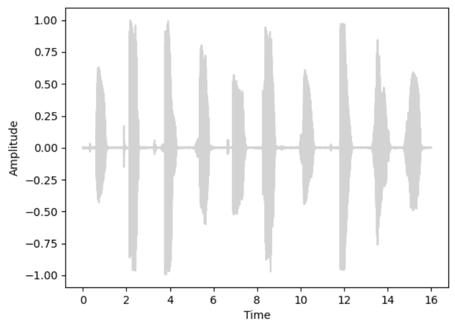
Demonstration:

Demodulation, spectral analysis: processing single files

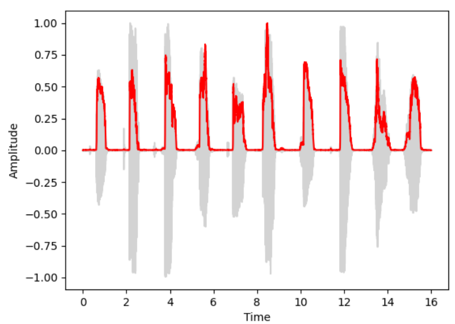
Demonstration applications: outputs



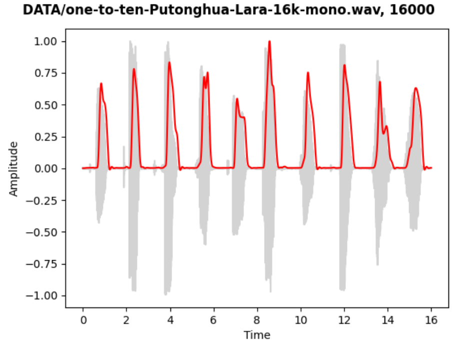
DATA/one-to-ten-Putonghua-Lara-16k-mono.wav, 16000



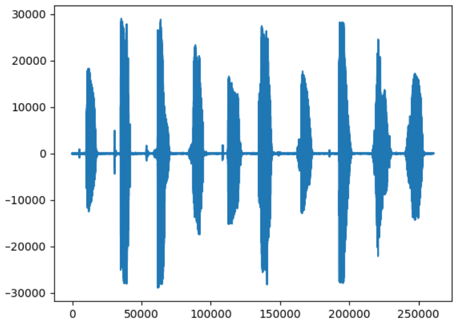
DATA/one-to-ten-Putonghua-Lara-16k-mono.wav, 16000



DATA/one-to-ten-Putonghua-Lara-16k-mono.wav, 16000

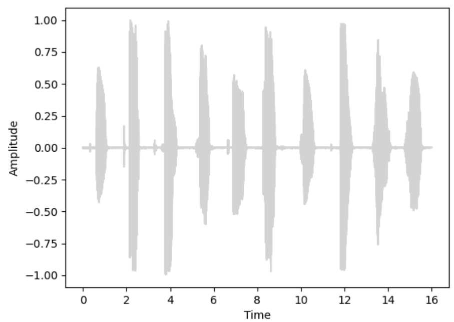


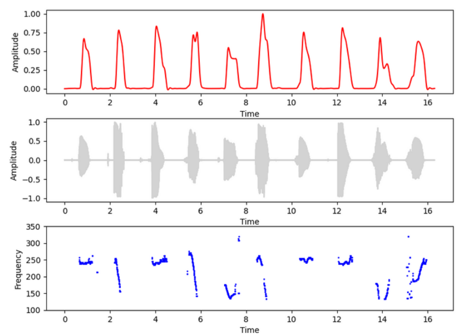
Demonstration apps - time domain outputs



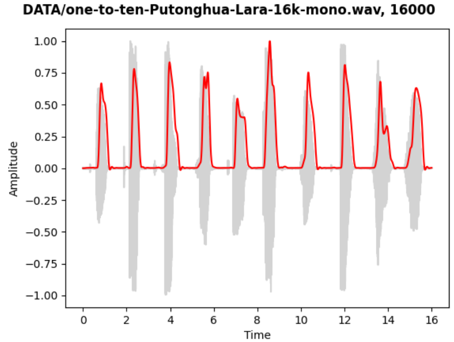
TIME
DOMAIN

DATA/one-to-ten-Putonghua-Lara-16k-mono.wav, 16000

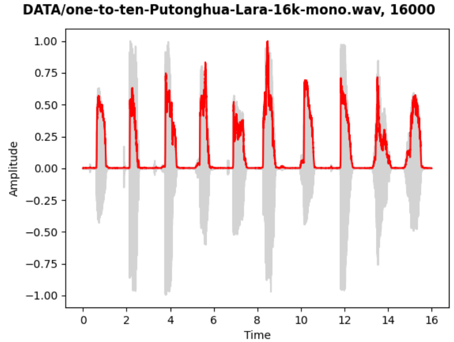




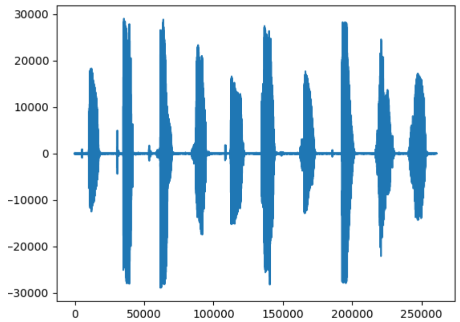
DATA/one-to-ten-Putonghua-Lara-16k-mono.wav, 16000



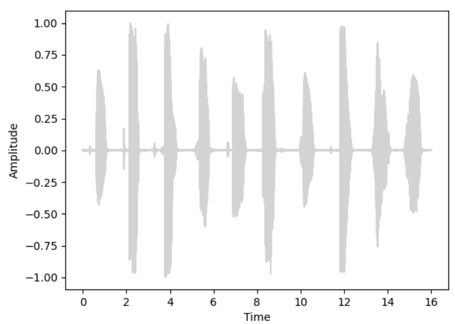
DATA/one-to-ten-Putonghua-Lara-16k-mono.wav, 16000



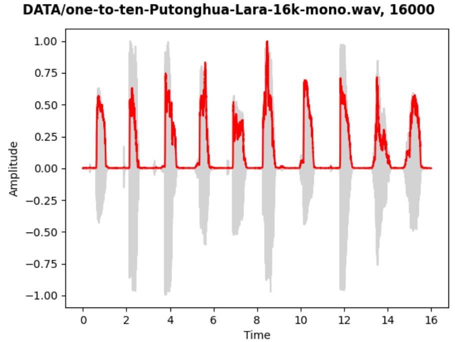
Demonstration apps – time and frequency domain outputs



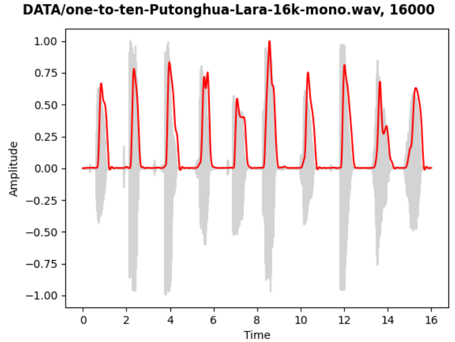
DATA/one-to-ten-Putonghua-Lara-16k-mono.wav, 16000



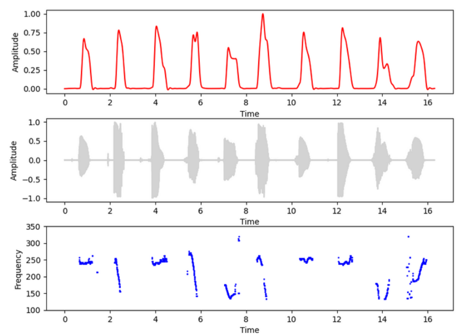
DATA/one-to-ten-Putonghua-Lara-16k-mono.wav, 16000



DATA/one-to-ten-Putonghua-Lara-16k-mono.wav, 16000

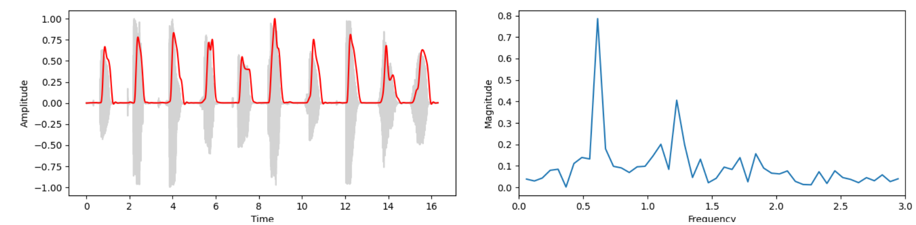


TIME DOMAIN

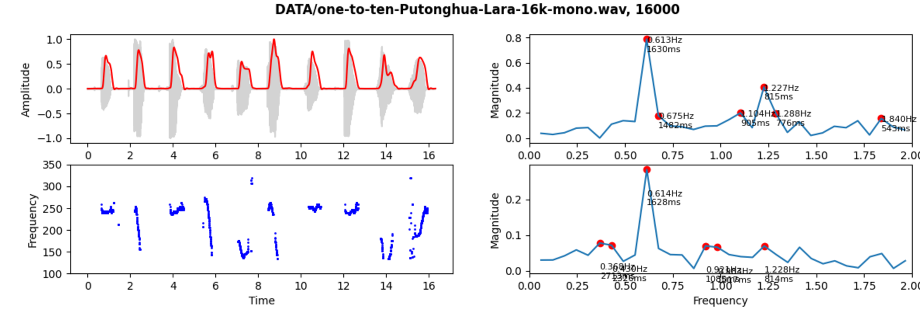


TIME DOMAIN	FREQUENCY DOMAIN
(waveform)	(spectrum)
Amplitude as a function of time	*Magnitude as a function of frequency*

DATA/one-to-ten-Putonghua-Lara-16k-mono.wav, 16000



DATA/one-to-ten-Putonghua-Lara-16k-mono.wav, 16000



Software description: time domain analysis

Time domain analysis: waveform display

```
# A_wavefor
import sys
import matpl
import scipy
wavfilename
fs, signal =
plt.plot(sig
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_waveform

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()                                              e
```

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                                                                   ces
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 ... 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                                    with 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

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

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

ename):
 a')
(text)

'.join(
pecfreqs ]

n(
pecmags ]

filename)
ilename)

ename)
```

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.

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

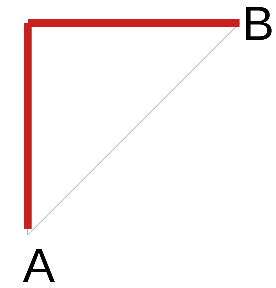
An English example:
The North Wind and the Sun

A German example:
Nordwind und Sonne

Distance metrics

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

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

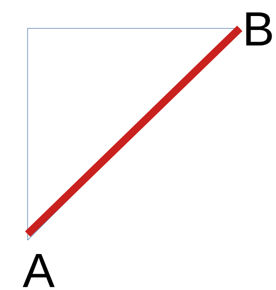


Canberra Distance
(Normalised Manhattan Distance)

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

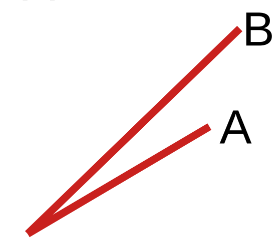
Euclidean Distance
direct distance
'as the crow flies'

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



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

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

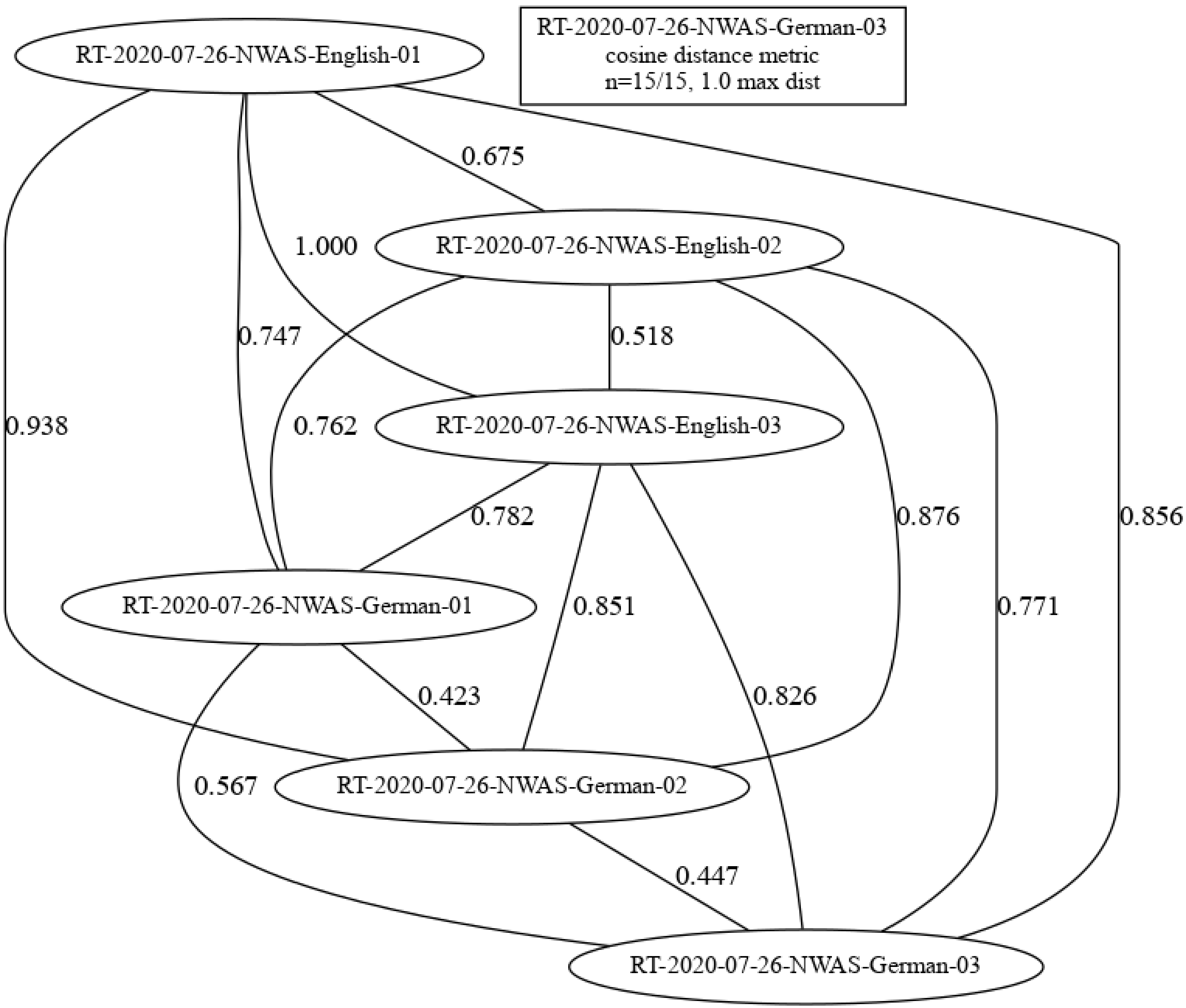


Spectrum Comparison: Distance Table

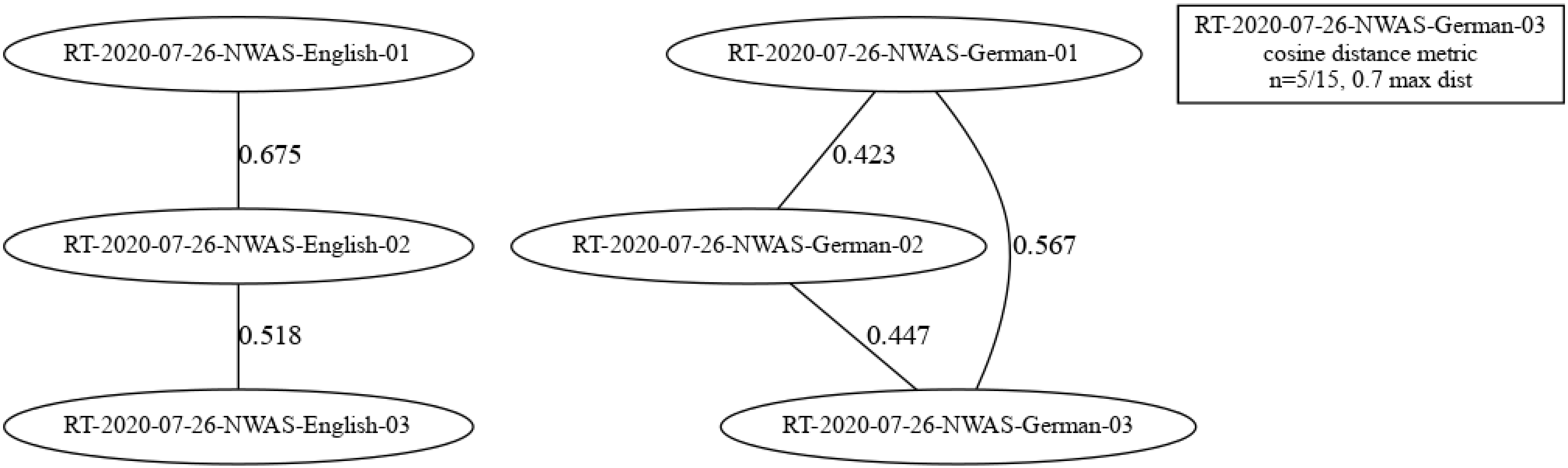
	Eng 01	Eng 02	Eng 03	Ger 01	Ger 02	Ger 03
		0.67477731	1.	0.74745837	0.93762055	0.85622088
			0.5184008	0.76221046	0.87568858	0.7706713
				0.78197106	0.85094568	0.82617612
					0.42298678	0.56668163
						0.44727788

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

Spectrum Comparison – Distance Networks, Part One

```
#  H_waveform_envelope_spectrum_distancenetwork.py. D. Gibbon, 2021-07-06
import sys, re, glob                                                    # import specialised modules
import numpy as np
import matplotlib.pyplot as plt
import scipy.io.wavfile as wave
from scipy.signal import butter, lfilter, medfilt, hilbert
import scipy.spatial.distance as dist
from graphviz import Graph

spectrummax = 3
distancelimit = 0.7
distancemetrics = [ 'canberra', 'chebyshev', 'cityblock',
                     'correlation', 'cosine', 'euclidean' ]

wavfiledirectory = sys.argv[1]
wavfilelist = sorted(glob.glob(wavfiledirectory+"*.wav"))
datasetname = sys.argv[2]

namelist = []
rawvaluelist = []
for wavfilename in wavfilelist:                   # Make spectra for all files
    wavfilebase = re.sub(".*/", "", wavfilename)
    wavfilebase = re.sub("-mono-16k.wav","",wavfilebase)

    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, 10 / (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))
    specmaglen = len(specmags)
    lfspecmaglen = int(round(spectrummax * specmaglen / (fs / 2)))
    lfspecmags = specmags[1:lfspecmaglen]
    lfspecmags = lfspecmags / max(lfspecmags)

    namelist += [ wavfilebase ]
    rawvaluelist += [ lfspecmags ]
```

Spectrum Comparison – Distance Networks, Part Two

```
Previous code:
     read all files and calculate spectrum for each file.
     calculate file namelist and rawvaluelist of spectra

Operations:
     use interpolation to ensure that lengths of spectra are equal
     calculate distances (differences) between spectra with distance metrics

newsize = np.max( [ len(val) for val in rawvaluelist ] )   # Make equal data lengths

valuelist = []
for val in rawvaluelist:
     size = len(val)
     xloc = np.arange(size)
     new_xloc = np.linspace(0, size, newsize)
     new_data = np.interp(new_xloc, xloc, val)                  # Interpolation
     valuelist += [ new_data ]
valuelist = np.array(valuelist)

for distancemetric in distancemetrics:

     distances = dist.pdist(valuelist, metric=distancemetric)
     dist_square = dist.squareform(distances)                                   # format as 2D table
     dist_list = dist_square.reshape(dist_square.shape[0] * dist_square.shape[1])          # reformat
     dist_list = (dist_list - np.min(dist_list)) / (np.max(dist_list) - np.min(dist_list))    # normalise
     dist_square = dist_list.reshape(dist_square.shape)

Output:
     Distances between spectra in a two-dimensional table
```

Spectrum Comparison – Distance Networks, Part Three

```
Previous code:
     read all files and calculate spectrum for each file.
     calculate file namelist and rawvaluelist of spectra

Operations:
     Create and save distance network graph

     d = Graph('D', filename=graphvizfilename, engine='dot', format='png')
     d.attr('node', shape='ellipse', fontsize='12', size='6,6', rankdir='LR')
     allcount = 0
     count = 0
     for i in range(0, len(namelist)-1):
          for j in range(i+1, len(namelist)):
               firstname = namelist[i]
               secondname = namelist[j]
               distance = dist_square[i][j]
               allcount += 1
               if distance <= distancelimit:
                    count += 1
                    d.node(firstname)
                    d.node(secondname)
                    d.edge(firstname, secondname, label="%.3f"%distance)
               else:
                    print(firstname,distance,secondname,"too large.")
     d.node(wavfilebase+"\n"+distancemetric + ' distance metric\nn=%d/%d, %s max dist'%(count,allcount,distancelimit),
               shape='box')
     graphvizfilename="GRAPHVIZ/"+datasetname+"-graphviz  -"+distancemetric
     d.render(graphvizfilename, view=False)                              # switch screen view or only save file
     plt.close("all")

Output:
     Distance network graph
```

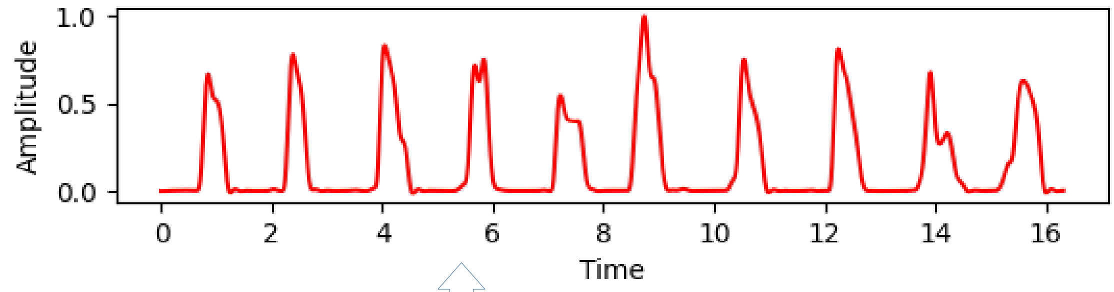
Spectrum Comparison – Hierarchical Clustering

```
Previous code:
      read all files and calculate spectrum for each file.
      calculate distances between spectra

Operations:
      Create and save hierarchical clustering dendrogram

import scipy.cluster.hierarchy as hy
figwidth = 6.5; figheight = 4
boxwidth = 0.6; boxheight = 0.83
halign = 0.02 ; valign = 0.14
orientation = "left"
dendrolevels = 20

for distancemetric in distancemetrics:

      distances = dist.pdist(valuelist, metric=distancemetric)
      clustermethods = methodlist_euclid if distancemetric == "euclidean" else methodlist_other

      for clustermethod in clustermethods:
                  print("Distance metric:", distancemetric, "Clustering method:", clustermethod)
                  fig = plt.figure(figsize=(figwidth, figheight))
                  ax1 = fig.add_axes([halign, valign, boxwidth, boxheight])
                  ax1.set_xlabel("%s%s-%s"%(figurefilebase,distancemetric,clustermethod), fontsize=8)

                  orientation = 'left'    # Change to 'right' or 'top' if leaf labels are cut off
                  Y1 = hy.linkage(distances, method=clustermethod)
                  hy.dendrogram(Y1,
                        p = dendrolevels, truncate_mode = "level",
                        orientation=orientation,
#                       cutoff = 0.3*np.max(Y1[:,2])
                        above_threshold_color='black', color_threshold=0,
                        count_sort="False", distance_sort=False, labels=namelist, leaf_font_size=10)
                  figurefilename = figurefilebase + "%s-%s-dendro.png"%(distancemetric,clustermethod)
                  plt.savefig(figurefilename)
                  plt.close(fig)          # Close each graph in loop after saving and displaying

Output: Hierarchical cluster graph (dendrogram)
```

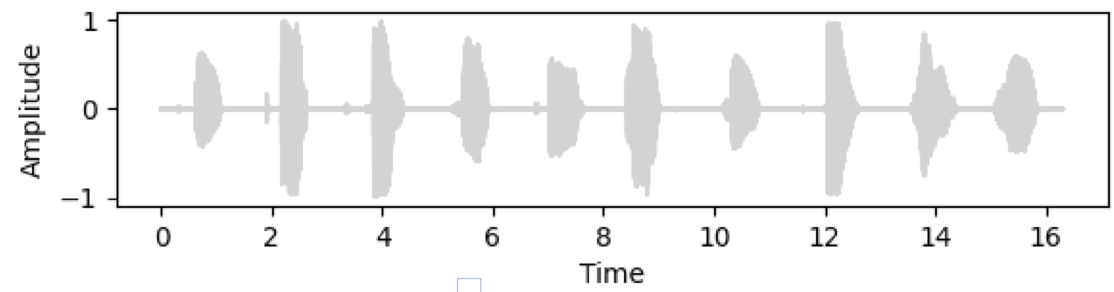
FM Demodulation

Low Frequency AM and FM Demodulation

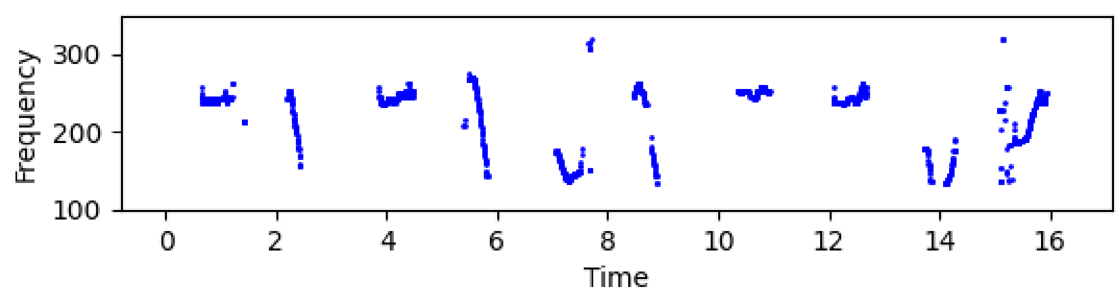
AM envelope demodulation:
- phonetics:
  amplitude curve, syllable,
  stress-accent
- phonology:
  sonority curve, syllables, stress



Modulated carrier signal



FM envelope demodulation:
- phonetics:
  F0, pitch track
- phonology:
  tones, pitch accents, intonation



FM Demodulation – F0 estimation (‘pitch’ extraction)

There are many algorithms for F0 estimation, for example:

Time domain algorithms:

autocorrelation (AC), average magnitude difference function (AMDF), average squared difference function (ASDF) ...

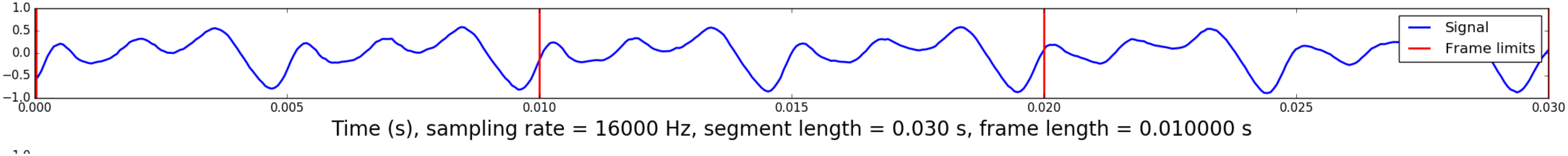
Frequency domain algorithms:

harmonic peak detection, spectral comb, ...

The AMDF algorithm:

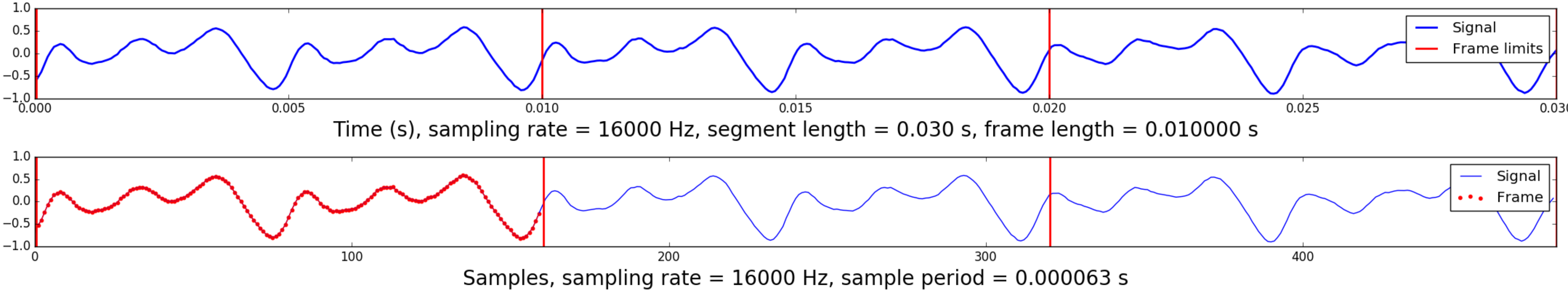
1. Divide the speech signal into equal time frames.
2. Make a copy of the first frame, noting the start position.
3. Move the copy through the first frame:
   - compare with the signal at each point
   - save the differences in a list
4. Find the first smallest difference in the list:
   - find its position in the signal
   - find the fundamental period (P0) by subtracting the start position from this position and divide by the sampling frequency.
   - then the fundamental frequency in this frame is: F0 = 1/P0
5. Move to the next frame and repeat until the last frame.

For all algorithms: divide the signal into equal time frames



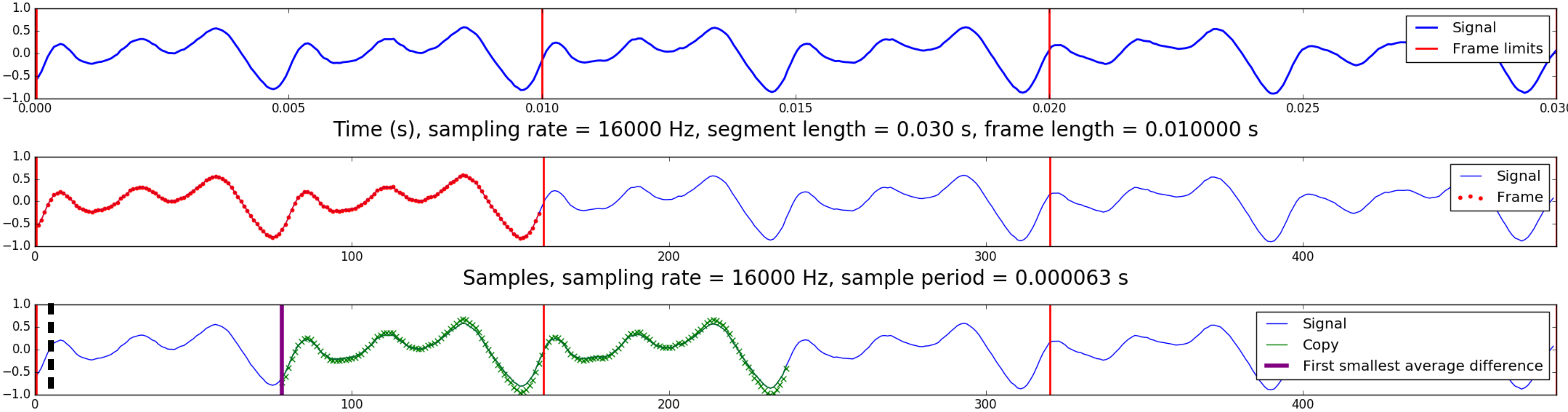
The duration of the time frame depends on the lowest frequency to be measured.

AMDF: make a copy of the first time frame



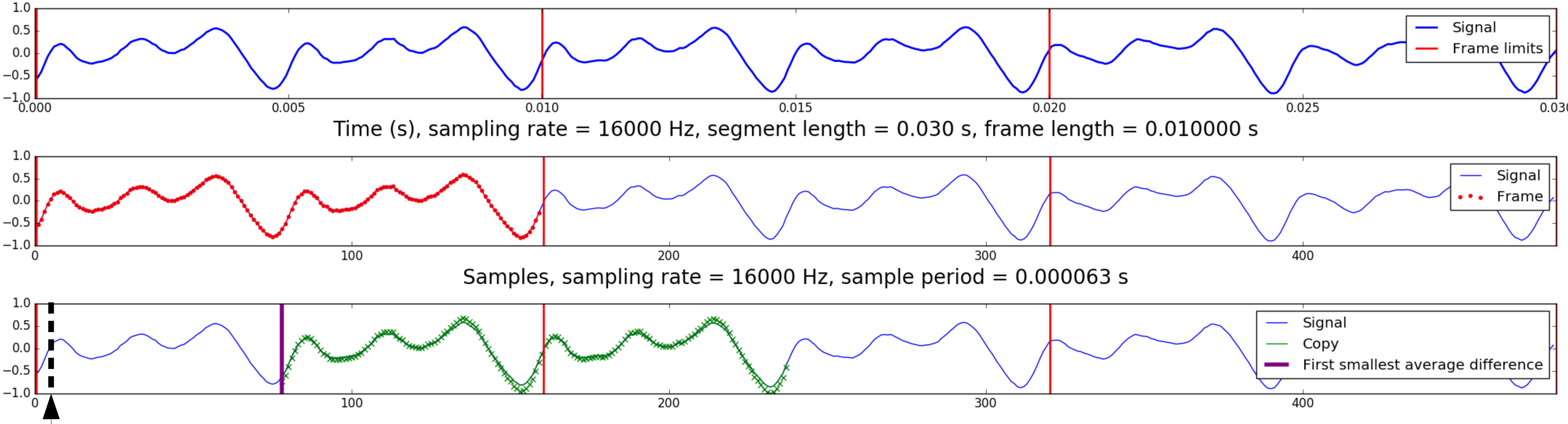
Note the start position of the time frame in the signal.

AMDF: move copy through first time frame



1. Compare the copy with the signal point by point at each position in the frame
2. Save each difference in a list, together with its current position in the frame
3. When finished with comparisons at all positions in the frame:
   search the list for the smallest difference with the copy and its position.

AMDF: move the copy through the first frame to the end



1. Compare the copy with the signal point by point at each position in the frame

2. Save each difference in a list, together with its current position in the frame

3. When finished with comparisons at all positions in the frame:
   search the list for the smallest difference with the copy and its position.

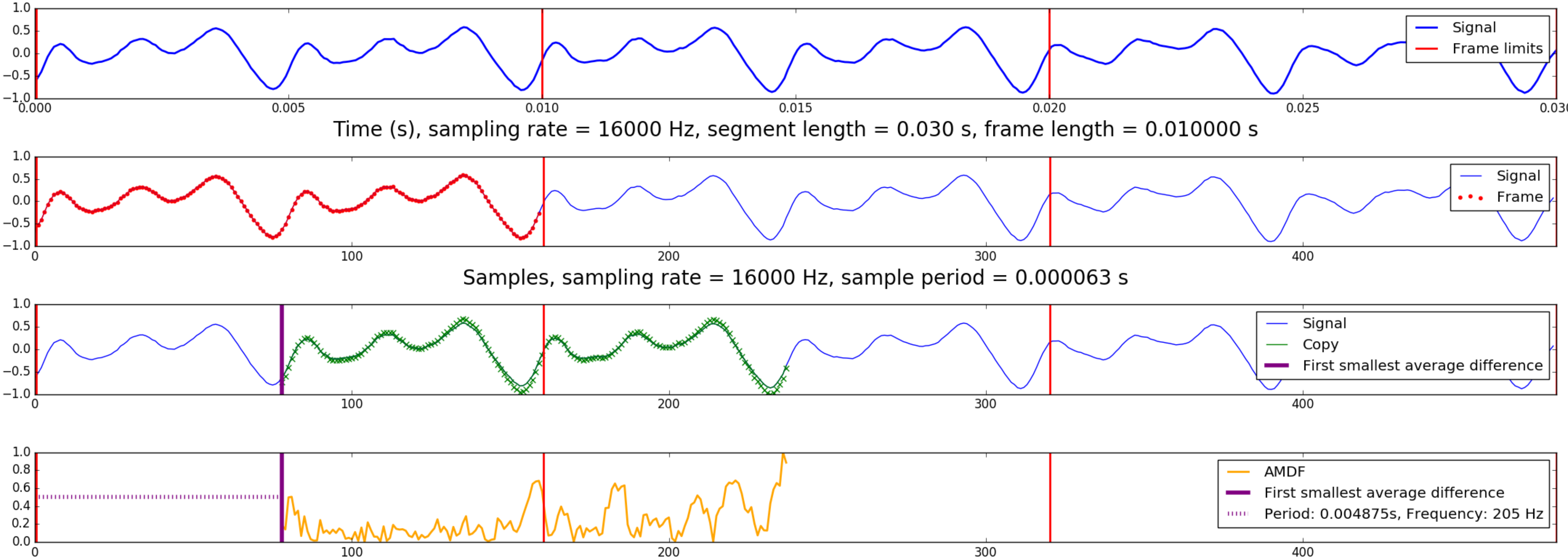
In practice, comparison of the copy with the signal starts with an offset slightly after the first position in the frame otherwise the smallest difference would always be zero! The position of the offset depends on the highest frequency to be measured.

Definition of AMDF

τ is the lag, which ranges from the beginning to the end of the frame

$$D(\tau)=\frac{1}{N-1-\tau}\sum_{n=0}^{N-1-\tau}\left|x(n)-x(n+\tau)\right|, 0\leq\tau\leq N-1$$

AMDF: calculate differences, minimal difference, T, F0



1. Note the position of the minimal difference between copy and signal
2. Calculate time period *T* of the frame as the difference between
   - the beginning of the frame and
   - the position of the minimal difference

     (in this case: 0.004875 s, i.e. 4.875 ms) divided by the sampling frequency *fs*
3. Calculate the frequency from the period: *F0* = 1 / *T*

   (in this case: 1 / 0.04875 = 205 Hz)

Move to the next frame and repeat the procedure for the remaining frames

FM demodulation, Part 1: waveform, AM envelope

```
# J_waveform_envelope_F0.py

import re, sys
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import scipy.io.wavfile as wave
from scipy.signal import butter, lfilter, medfilt
from module_fm_demodulation import *

wavfilename = sys.argv[1]                          # get input filename from command line
fs, signal = wave.read(wavfilename)                # read sampling frequency and signal
wavfilebase = re.sub("^.*/","",wavfilename)
wavfilebase = re.sub("-16k-mono","",wavfilebase[:-4])
figurefilename = "PNG/RFA_%s.png"%wavfilebase

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
```

FM demodulation, Part 2: F0 estimation

FM demodulation using the AMDF (Average Magnitude Difference Function) method.

The F0 estimation routines are longer and more complex than previous routines, so they are simply summarised here, for reasons of time, space and effort:

```
f0estimate(signal,fs)
    clipper(sig,thresh,type)                                # Clip low level noise
    butterworthfilter(signaldata, cutoff, order, fs, type)  # Low pass filter
    f0movingwindow(signal, fs, windowshape, framelength, frameskip, f0diffoffsetlength)
        f0amdf(signal, fs, windowshape, framestart, framelength, f0diffoffsetlength)
```

Postprocessing: moving median filter to remove ‘noisy’ outliers.

FM demodulation, Part 3: F0 parameters

```
A number of parameters are defined:

centrethresh = 0.0                    # Deals with silence and low volume noise
limitthresh = 0.9
fmbutterhigh = f0min * 2              # low pass filter
fmbutterhighorder = 5
fmbutterlow = f0max                   # high pass filter
fmbutterloworder = 2

f0min = 50                            # minimum expected F0
f0max = 450                           # maximum expected F0

# Voice range dependent AMDF parameters
f0framelengthfactor = 0.75            # relative to f0min, > 1
f0frameskipfactor = 0.5               # Default is 1, the frame length
f0diffoffsetlengthfactor = 0.1        # relative fo f0max
f0framedispersion = 0.1               # quasi-noise/voiceless detector - can this work?
f0peakoperation = "median"            # the implmementation of "average"
f0differenceoffset = 0.5

# Atomatic voice model calculation based on minimum and maximum frequency settings
f0frameduration =  1 / f0min
f0frameduration = f0framelengthfactor * f0frameduration
framerate = 2 / f0frameduration
framelength = int(f0frameduration * fs)
frameskip = int(framelength * f0frameskipfactor)
windowshape = tukey(framelength, f0tukeyfraction)

# AMDF offset
f0diffoffsetdur = 1 / f0max                                             # seconds
f0diffoffsetlength = int(f0diffoffsetlengthfactor * f0diffoffsetdur * fs)   # samples
```

FM demodulation, Part 4: F0 estimation

F0 preprocessing: filtering:

```
fsignal = clipper(signal,centrethresh,"centre")
fsignal = clipper(fsignal,limitthresh,"limit")
fsignal = butterworthfilter(fsignal, fmbutterlow, fmbutterloworder, fs, "low")
fsignal = butterworthfilter(fsignal, fmbutterhigh, fmbutterhighorder, fs, "high")
```

F0 estimation frame loop:

```
def f0estimate(signal,fs, framelength, frameskip, f0medfilter):
    f0array = np.array([
                    f0amdf(signal, fs, )
                    for framestart in range(0, len(signal)-3*framelength, frameskip)
                        ])
    f0array = medfilt(f0array, f0medfilter)
    return f0array
```

The function of moving median filters is to provide a low-pass smoothing result without being too influenced by outlier values.

This is a very common technique for smoothing F0 tracks (‘pitch’ tracks).

FM demodulation – F0 extraction, Part 5, AMDF

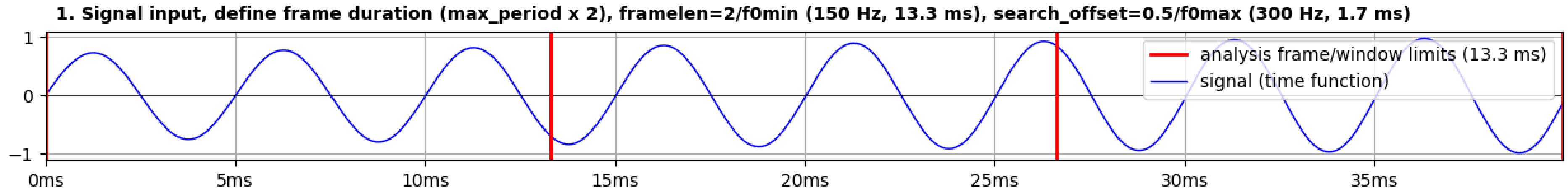
```
def f0amdf(signal, fs, windowshape, framestart, framelength, f0diffoffsetlength):

    framestop = framestart + framelength
    framecopy = signal[framestart:framestop]
    framecopydiff = np.diff(framecopy)
    framestd = np.std(framecopydiff)

    if framestd < f0framedispersion:      # anti-noise, quasi-voice-detector

        movingwindowrange = range(framestart+f0diffoffsetlength, framestop)

        meandiffs = [np.sum(
                    np.abs(framecopy - signal[movwinstart:movwinstart+framelength]))
                    for movwinstart in movingwindowrange  ]

        meandiffs = list(np.array(meandiffs)/np.max(meandiffs))
        smallestmeandiff = np.min(meandiffs)

        if smallestmeandiff < f0differenceoffset:
            smallestmeandiffpos = meandiffs.index(smallestmeandiff) + f0diffoffsetlength
            period = smallestmeandiffpos / fs
            frequency = 1 / period

        else:
            frequency = 0
    else:
        frequency = 0

    return frequency
```

FM demodulation – F0 extraction, Part 6, graphics

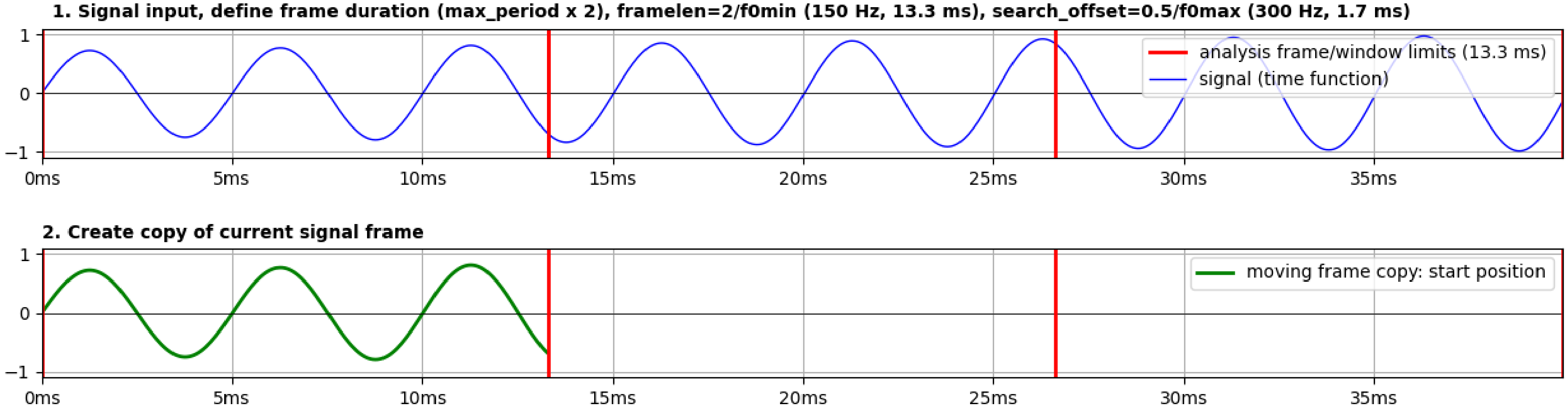
```
The graphics output is a small extension of existing graphics output routines.

fig, (plt01, plt02, plt03) = plt.subplots(nrows=3, ncols=1, figsize=(6, 6))
plt.suptitle = "%s [file: %s]"%("Speech signal demodulation", wavfilebase)

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

xaxistime = np.linspace(0, signalseconds, signallength)    # define x axis in seconds
plt02.plot(xaxistime, signal, color="lightgrey")        # plot waveform in grey
plt02.set_xlabel("Time")
plt02.set_ylabel("Amplitude")

xaxistime = np.linspace(0, signalseconds, f0arraylength)   # define x axis in seconds
plt03.scatter(xaxistime, f0array, s=1, color="blue")       # plot waveform in grey
plt03.set_ylim(f0min, f0max)
plt03.set_xlabel("Time")
plt03.set_ylabel("Frequency")

plt.tight_layout(pad=1, w_pad=0, h_pad=5)
plt.savefig(figurefilename)
plt.show()
```

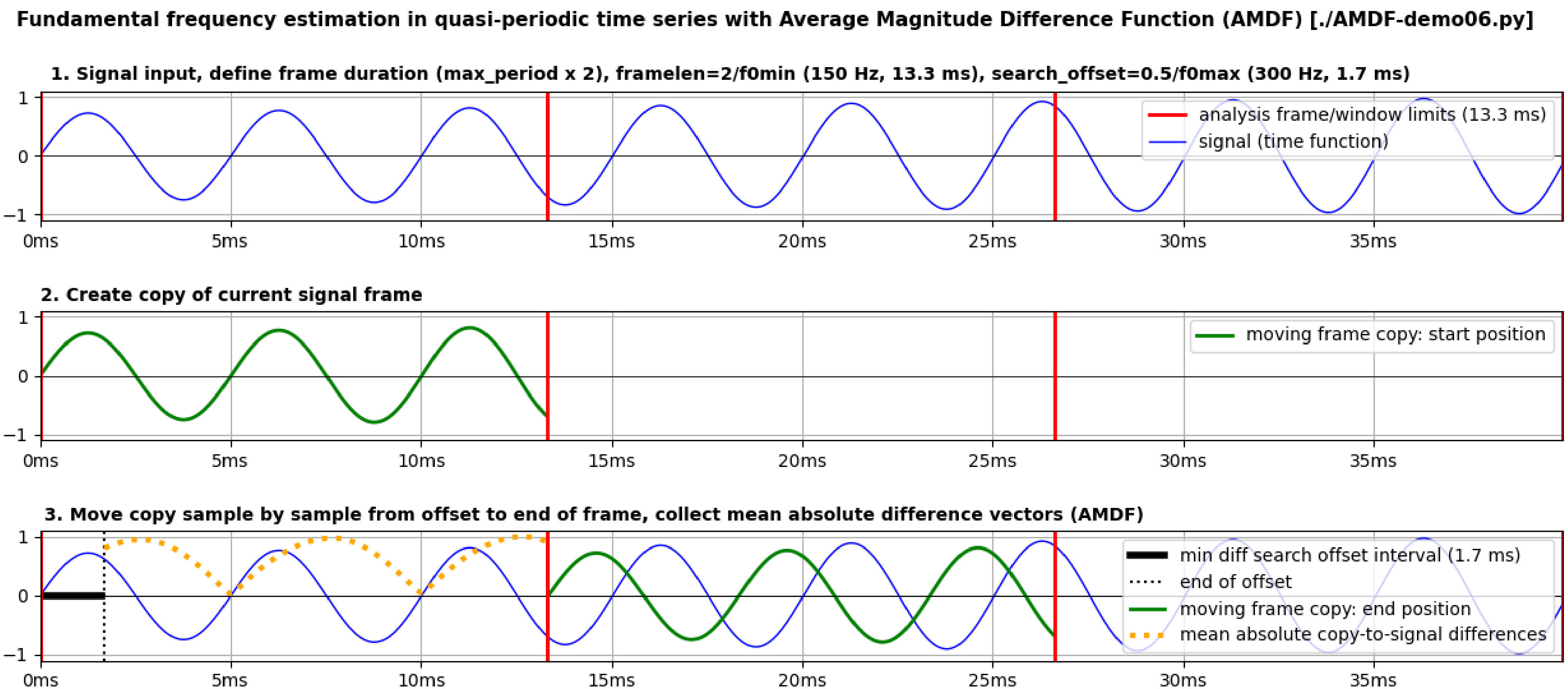
Revision of AMDF

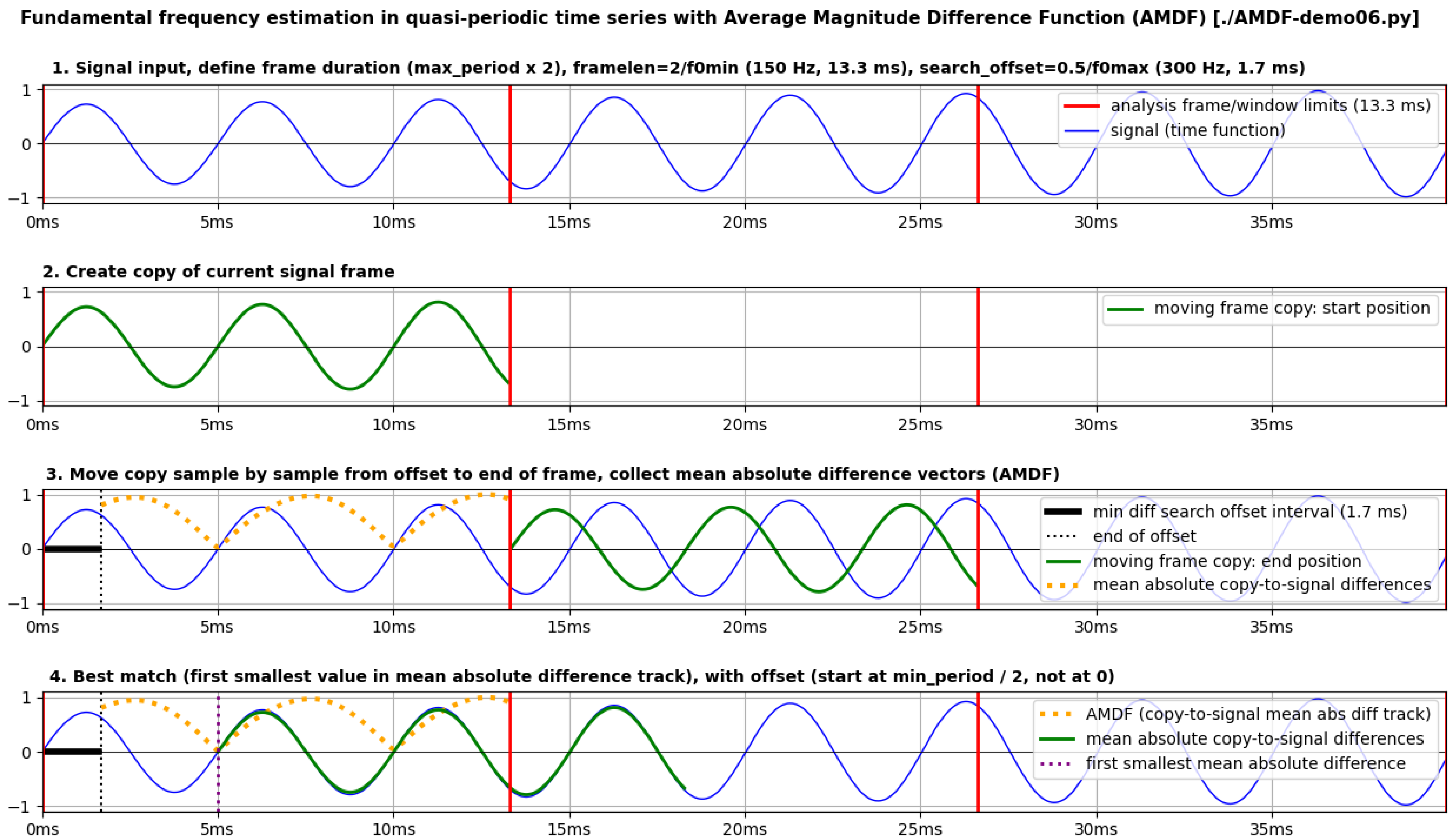
Fundamental frequency estimation in quasi-periodic time series with Average Magnitude Difference Function (AMDF) [./AMDF-demo06.py]

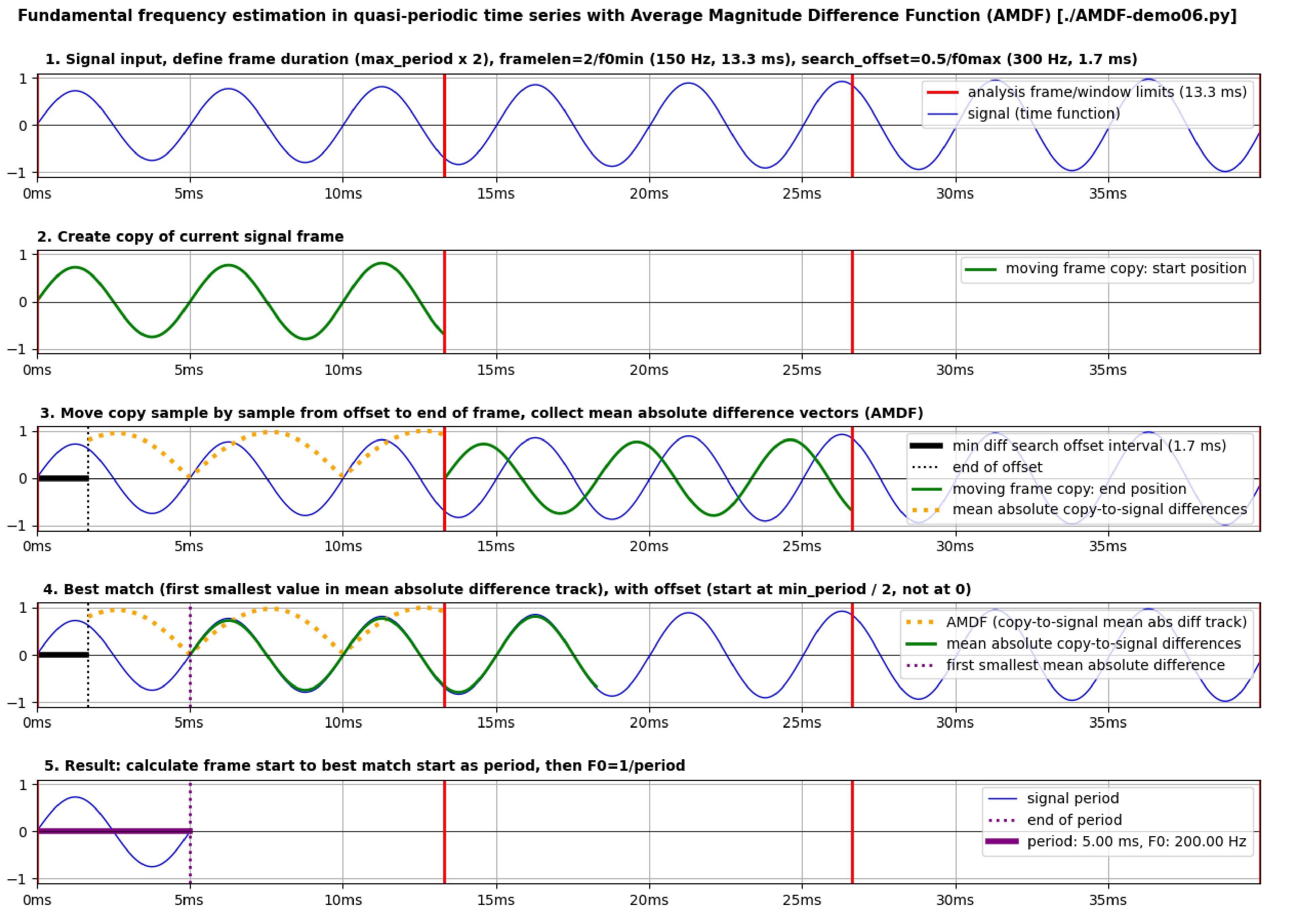


Fundamental frequency estimation in quasi-periodic time series with Average Magnitude Difference Function (AMDF) [./AMDF-demo06.py]




Fundamental frequency estimation in quasi-periodic time series with Average Magnitude Difference Function (AMDF) [./AMDF-demo06.py]
1. Signal input, define frame duration (max_period x 2), framelen=2/f0min (150 Hz, 13.3 ms), search_offset=0.5/f0max (300 Hz, 1.7 ms)
analysis frame/window limits (13.3 ms)
signal (time function)
2. Create copy of current signal frame
moving frame copy: start position
3. Move copy sample by sample from offset to end of frame, collect mean absolute difference vectors (AMDF)
min diff search offset interval (1.7 ms)
end of offset
moving frame copy: end position
mean absolute copy-to-signal differences
0ms
5ms
10ms
15ms
20ms
25ms
30ms
35ms


Fundamental frequency estimation in quasi-periodic time series with Average Magnitude Difference Function (AMDF) [./AMDF-demo06.py]
1. Signal input, define frame duration (max_period x 2), framelen=2/f0min (150 Hz, 13.3 ms), search_offset=0.5/f0max (300 Hz, 1.7 ms)
analysis frame/window limits (13.3 ms)
signal (time function)
2. Create copy of current signal frame
moving frame copy: start position
3. Move copy sample by sample from offset to end of frame, collect mean absolute difference vectors (AMDF)
min diff search offset interval (1.7 ms)
end of offset
moving frame copy: end position
mean absolute copy-to-signal differences
4. Best match (first smallest value in mean absolute difference track), with offset (start at min_period / 2, not at 0)
AMDF (copy-to-signal mean abs diff track)
mean absolute copy-to-signal differences
first smallest mean absolute difference
0ms
5ms
10ms
15ms
20ms
25ms
30ms
35ms


Fundamental frequency estimation in quasi-periodic time series with Average Magnitude Difference Function (AMDF) [./AMDF-demo06.py]
1. Signal input, define frame duration (max_period x 2), framelen=2/f0min (150 Hz, 13.3 ms), search_offset=0.5/f0max (300 Hz, 1.7 ms)
analysis frame/window limits (13.3 ms)
signal (time function)
2. Create copy of current signal frame
moving frame copy: start position
3. Move copy sample by sample from offset to end of frame, collect mean absolute difference vectors (AMDF)
min diff search offset interval (1.7 ms)
end of offset
moving frame copy: end position
mean absolute copy-to-signal differences
4. Best match (first smallest value in mean absolute difference track), with offset (start at min_period / 2, not at 0)
AMDF (copy-to-signal mean abs diff track)
mean absolute copy-to-signal differences
first smallest mean absolute difference
5. Result: calculate frame start to best match start as period, then F0=1/period
signal period
end of period
period: 5.00 ms, F0: 200.00 Hz
0ms
5ms
10ms
15ms
20ms
25ms
30ms
35ms

FM spectral analysis, Part 1, F0 estimation

```
# K_waveform_envelope_F0_spectrum.py. D. Gibbon, 2021-07-06

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

specmax = 2
magscount = 

if len(sys.a
    wavfilen                                                        mand line
else:
    wavfilen
wavfilebase 
wavfilebase 
figurefilena

fs, signal =                                                         signal
signallength                                                         es
signalsecond                                                         onds
signal = sig                                                         .. 1

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

f0array, framerate = f0estimate(signal, fs)
f0arraylength = len(f0array)
```

Description

In this demonstration application, a novel and unusual step is taken: the spectrum of the demodulated FM signal is calculated.

The procedures are entirely parallel, but with *f0array* instead of *envelope*, and *framerate* instead of *fs*.

For example, corresponding lines can be compared:

```
amspecmags = np.abs(np.fft.rfft(envelope))
fmspecmags = np.abs(np.fft.rfft(f0array))

amspecfreqs = np.linspace(0,fs/2,amspecmaglen)
fmspecfreqs = np.linspace(0,framerate/2,fmspecmaglen)
```

AM and FM spectral analysis, Part 2, spectral analysis

```
amspecmags = np.abs(np.fft.rfft(envelope))
amspecmags = amspecmags / np.max(amspecmags)
amspecmaglen = len(amspecmags)
amspecfreqs = np.linspace(0,fs/2,amspecmaglen)

amspectrummax = specmax
lfamspecmaglen = int(round(amspectrummax * amspecmaglen / (fs / 2)))
lfamspecmags = amspecmags[1:lfamspecmaglen]
lfamspecfreqs = amspecfreqs[1:lfamspecmaglen]

amtopmagscount = magscount                         # define max frequency of lf spectrum
amtopmags = sorted(lfamspecmags)[-amtopmagscount:]
amtoppos = [ list(lfamspecmags).index(m) for m in amtopmags ]
amtopfreqs = [ lfamspecfreqs[p] for p in amtoppos ]

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

fmspecmags = np.abs(np.fft.rfft(f0array))
fmspecmags = fmspecmags / np.max(fmspecmags)
fmspecmaglen = len(fmspecmags)
fmspecfreqs = np.linspace(0,framerate/2,fmspecmaglen)

fmspectrummax = specmax
lffmspecmaglen = int(round(fmspectrummax * fmspecmaglen / (framerate / 2)))
lffmspecmags = fmspecmags[1:lffmspecmaglen]
lffmspecfreqs = fmspecfreqs[1:lffmspecmaglen]

fmtopmagscount = magscount                         # define max frequency of lf spectrum
fmtopmags = sorted(lffmspecmags)[-fmtopmagscount:]
fmtoppos = [ list(lffmspecmags).index(m) for m in fmtopmags ]
fmtopfreqs = [ lffmspecfreqs[p] for p in fmtoppos ]
```

AM and FM spectral analysis, Part 3: graphics

```
fig,((plt01, plt02),(plt03, plt04)) = plt.subplots(nrows=2, ncols=2, figsize=(14, 4))# define figure
format
plt.suptitle("%s, %d"%(wavfilename, fs), fontweight="bold")# display a title

# Time domain
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")

xaxistime = np.linspace(0, signalseconds, f0arraylength)         # define x axis in seconds
plt03.scatter(xaxistime, f0array, s=1, color="blue")             # plot waveform in grey
plt03.set_ylim(f0min, f0max)
plt03.set_xlabel("Time")
plt03.set_ylabel("Frequency")

# Frequency domain
plt02.plot(lfamspecfreqs, lfamspecmags)
plt02.scatter(amtopfreqs, amtopmags, color="red")
for f,m in zip(amtopfreqs, amtopmags):
     plt02.text(f, m-0.1, "%.3fHz\n%dms"%(f,1000/f), fontsize=8)
plt02.set_xlabel("Frequency")
plt02.set_ylabel("Magnitude")
plt02.set_xlim(0,amspectrummax)

plt04.plot(lffmspecfreqs, lffmspecmags)
plt04.scatter(fmtopfreqs, fmtopmags, color="red")
for f,m in zip(fmtopfreqs, fmtopmags):
     plt04.text(f, m-0.1, "%.3fHz\n%dms"%(f,1000/f), fontsize=8)
plt04.set_xlabel("Frequency")
plt04.set_ylabel("Magnitude")
plt04.set_xlim(0,amspectrummax)

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

Finally ...

Science about trying to prove yourself to be wrong.

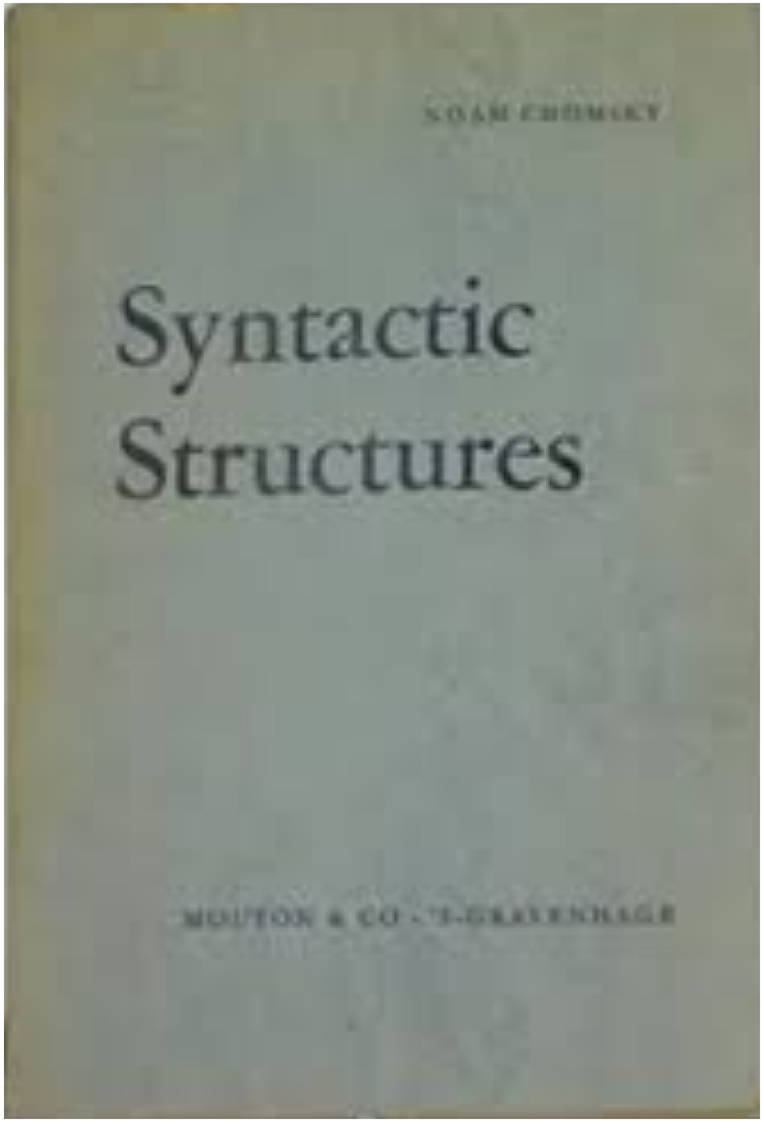
Then trying to do more with new data if you are right (and others agree that you are right using similar methods).

But improving your theory or method, or using different data if you are wrong.

Scientific Discovery: a clear example of Critical Rationalism

Chomsky, N. 1957. *Syntactic Structures*. The Hague: Mouton.

TABLE OF CONTENTS



Scientific Discovery: a clear example of Critical Rationalism

Chomsky, N. 1957. *Syntactic Structures*. The Hague: Mouton.

TABLE OF CONTENTS

1. Domain characterisation and delimitation
2. Finite State Grammars – falsified!
3. Phrase Structure Grammars – falsified!
4. Transformational Grammars – not falsified!

Scientific Discovery: a clear example of Critical Rationalism

Chomsky, N. 1957. *Syntactic Structures*. The Hague: Mouton.

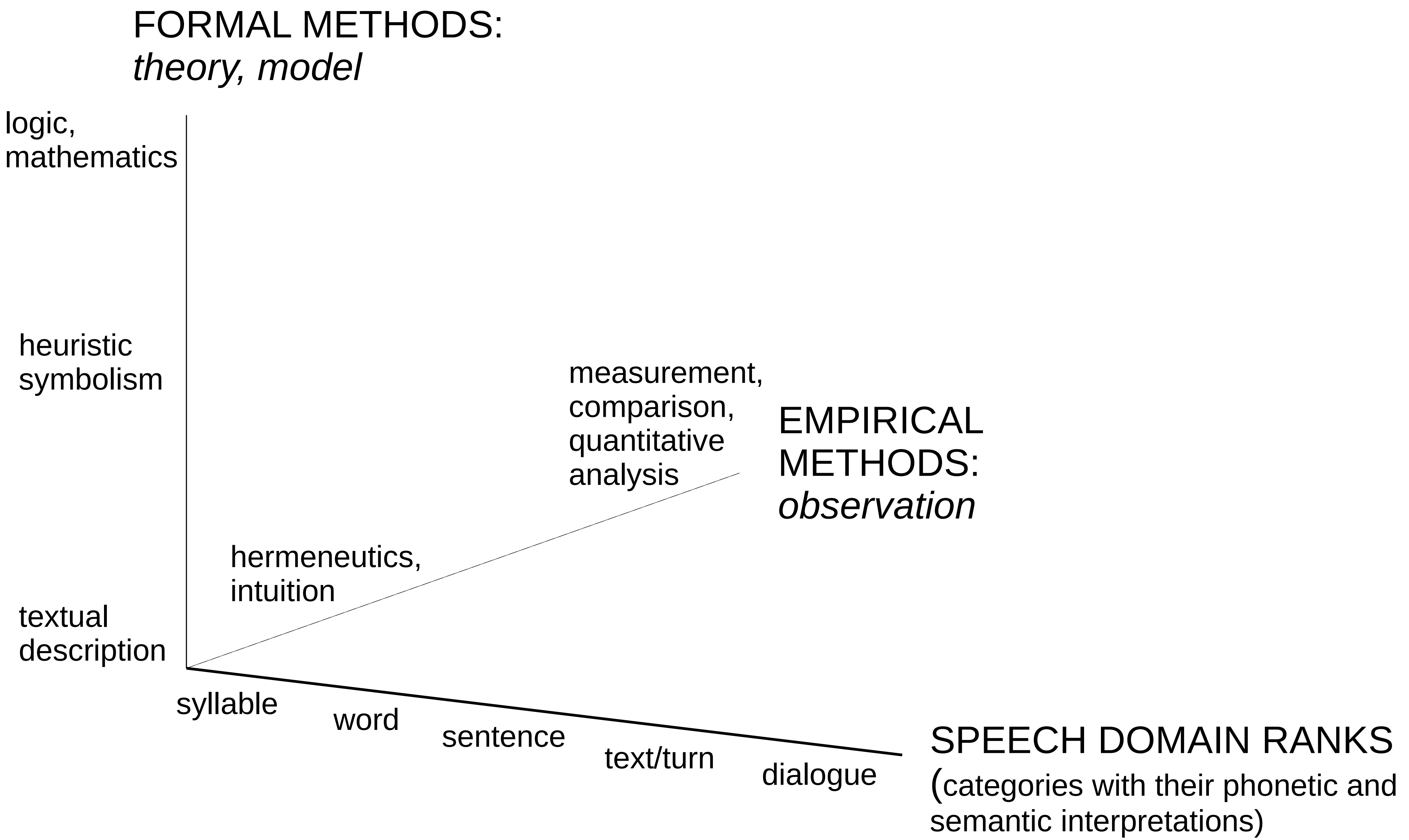
TABLE OF CONTENTS

1. Domain characterisation and delimitation
2. Finite State Grammars – falsified!
3. Phrase Structure Grammars – falsified!
4. Transformational Grammars – not falsified!

Chomsky's models have been shown to overgeneralise: complete but not sound.

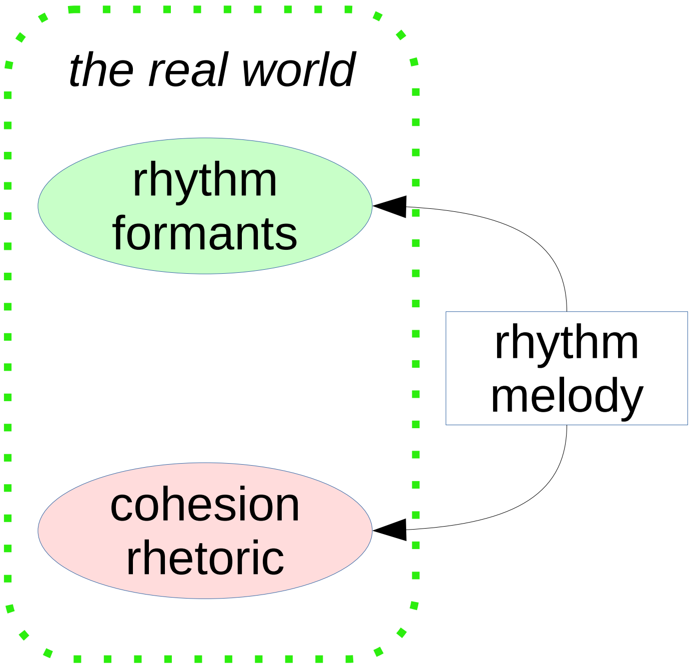
For example, phonology, prosody, morphology, as well as syntax in conversational speech (but not semantics), can be fully modelled with Finite State Grammars.

A Critical Rationalist approach to methodology



Summary

- Lecture 1:
  - Semiotics of prosody
  - Rhythm and melody
- Lecture 2:
  - Rhythm analysis method:
    - Rhythm Formant Theory
    - Rhythm Formant Analysis
- Lecture 3:
  - Modulation Theory
  - Rhythm Formant Analysis: “do it yourself”
  - Scientific methodology



谢谢

Many thanks for participating,

and good luck with your coding!

Thanks – looking forward to future contacts!