COM3502 Speech Processing — Assignment 2

Jack Cheng Ding Han

150159519, aca15jch, jcdhan1@sheffield.ac.uk

Department of Computer Science, University of Sheffield

December 2017

Contents

1 Principles of Estimation

1.1 Formants

Because each of the formant frequencies $F_n, n \in \mathbb{N}$ are within a particular range, filters can be used to extract them. The input audio is passed through a high-pass filter with a cutoff frequency equal to the formant's lower bound and this is then passed through a low-pass filter with a cuttoff frequency equal to the formant's upper bound. This is then passed through two bandpass filters; one with a centre frequency equal to the lower bound and one with a centre frequency equal to the upper bound. The root-mean-square amplitude of these bandpass filters' outputs are divided (RMS amplitude of the lower bound is the numerator) and then passed through a series of arithmetic operations to retrieve the appropriate formant frequency.

For the first and second formants, I reused the bounds from my previous assignment; $300\,\text{Hz} \leq F_1 \leq 700\,\text{Hz}$ and $800\,\text{Hz} \leq F_2 \leq 2400\,\text{Hz}$, respectively. According to Nakata and Ichikawa [1], the third formant is in the range $2200\,\text{Hz} \leq F_3 \leq 3500\,\text{Hz}$.

To estimate the amplitudes of a given formant, the root mean square amplitude of the output of the aforementioned low-pass filter is retrieved and passed through the formula:

$$\frac{F_{nrms} - 60}{50}$$

1.2 Voicing

The approach to detecting whether a sound is voiced or unvoiced is the AMDF (average magnitude distance function) voicing detector[1] which uses an ADMF to decide between a zero-crossing approach (the only approach me and my partner used in our previous assignment) and the energy approach [2].

1.3 Fundamental Frequency

The penultimate stage in the estimation of the fundamental frequency F_0 involves the conversion of a MIDI pitch into a frequency. This is done using the following formula:

$$F_0 = 2^{(d-69)/12} \cdot 440\,\text{Hz}$$

Prior to to this stage, however, the MIDI pitch must be extracted from the input audio. The object utilised by my program, `fiddle~`, uses the Discrete Fourier Transform to compute the spectrum at an integer number of evenly spaced discrete frequencies. From this, it uses a technique called maximum-likelihood pitch detection where the best guess at the fundamental frequency is that which has the highest output for the function $\mathfrak{L}(f)$, this function is defined as [3]:

$$\mathfrak{L}(f) = \sum_{i=0}^{k} a_i \cdot t_i \cdot n_i$$

Where:

- k is the total number of peaks in the spectrum returned by the DFT.
- a_i is a factor depending on the amplitude of i^{th} peak.
- t_i depends on how closely the i^{th} peak is tuned to amplitude of the argument f.
- i depends on whether the peak is closest to a low or a high multiple of f.

[1] Implementation available at https://github.com/pd-l2ork/pd/tree/master/externals/ekext

2 Implementation

2.1 The Graphical User Interface

The main GUI utilises a graph to make sure one can use the program without distractions. The user only sees the aesthetic, descriptive and interactive parts.

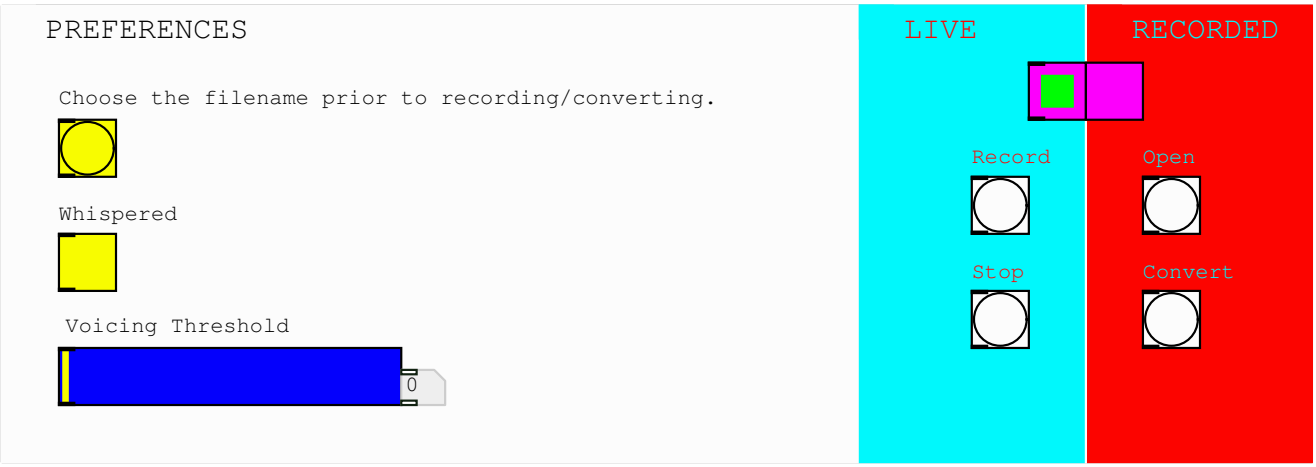

Figure 1: The Graphical User Interface — `main.pd`

Aesthetic parts:

- A white `canvas` for preferences.
- A cyan `canvas` for the live controls.
- A red `canvas` for the recorded controls.

Descriptive parts:

- A `number` to tell the user the current voicing threshold they selected using the `Hslider`.

Interactive parts:

- A `Hradio` allowing the user to switch between the microphone and an existing file as the source of the input audio.
- Preferences
  - A `Bang` that when clicked, opens a dialog allowing the user to choose path and name of the output file.
  - A `Toggle` that when checked turns the input audio into whispered speech
- Live Controls
  - A `Bang` that when clicked, starts the process of converting live audio into a list of 12-tuples.
  - A `Bang` that when clicked, stops the process of converting the input audio into a list of 12-tuples and saves the current list to a `.pfs` file. If the user is using an existing audio file as the source of the input audio, this also flashes when the audio file has stopped playing.
- Recorded Controls
  - A `Bang` that when clicked, opens a dialog which allows the user to choose an existing audio file to convert into a `.pfs` file.
  - A `Bang` that when clicked, starts the conversion process into a `.pfs` file as well as play the file through the user's speakers. The saving of the list of 12-tuples into a `.pfs` file ends automatically when the input audio file stops playing.

The contents of the subpatch in the graph unites the outlets of three objects together using a `pack` object.

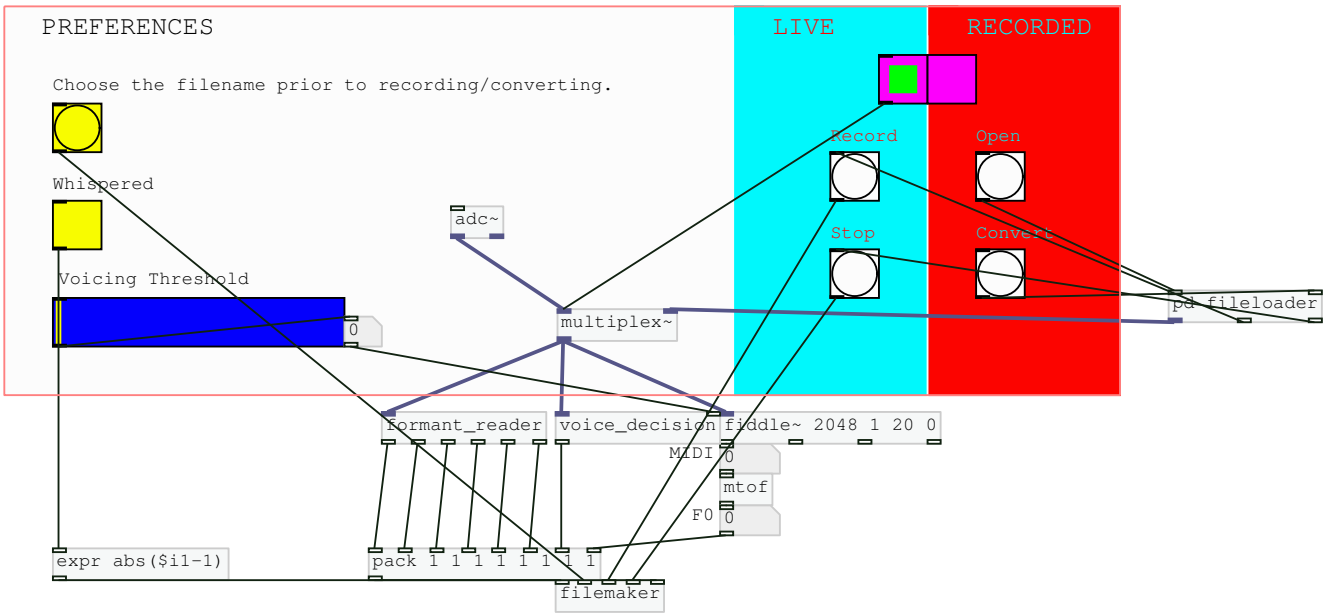

Figure 2: Uniting the outlets of different objects.

A subpatch lets the user choose a file to convert into the `.pfs` format. It plays the file through the speakers using the `dac~` patch and sends a bang when it stops playing which allows the conversion to `.pfs` to end automatically.

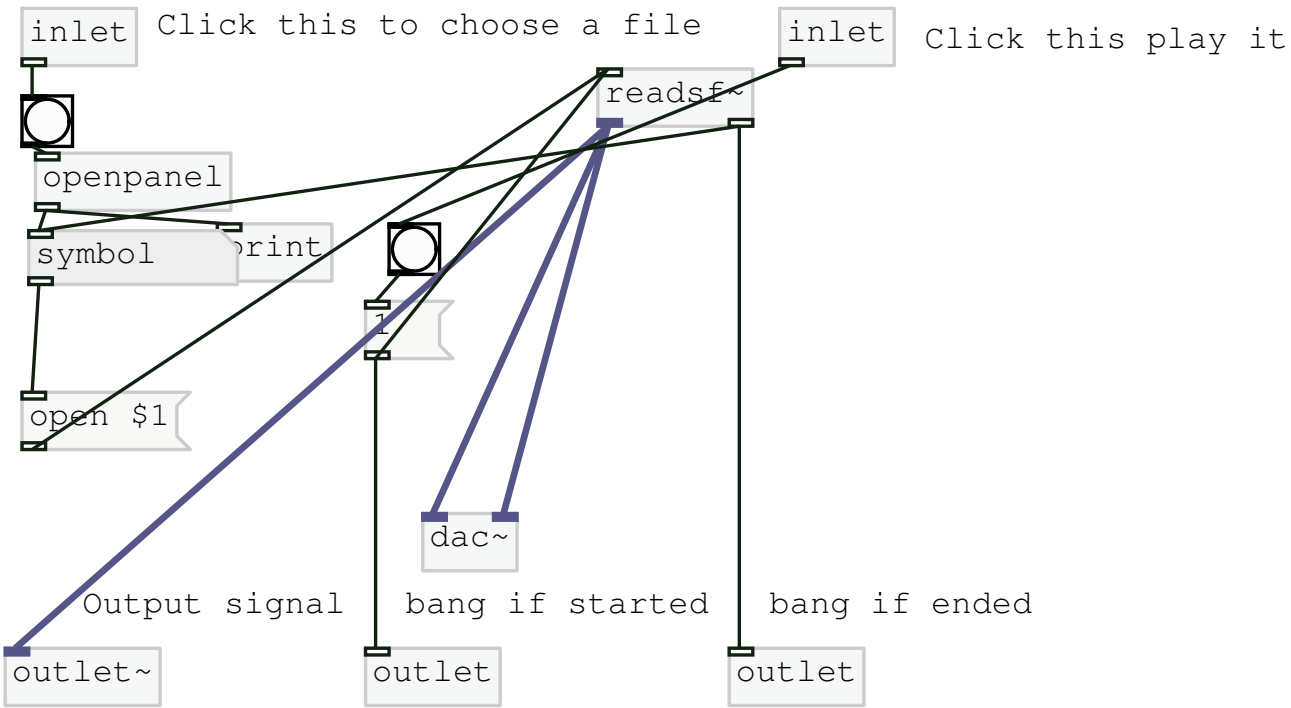

Figure 3: The subpatch in `main.pd` called `fileloader`

2.2 Parameter Retrieval

2.2.1 Formants and Amplitudes

The principles in section 1.1 are done using the abstraction `formant_freq_amp.pd`

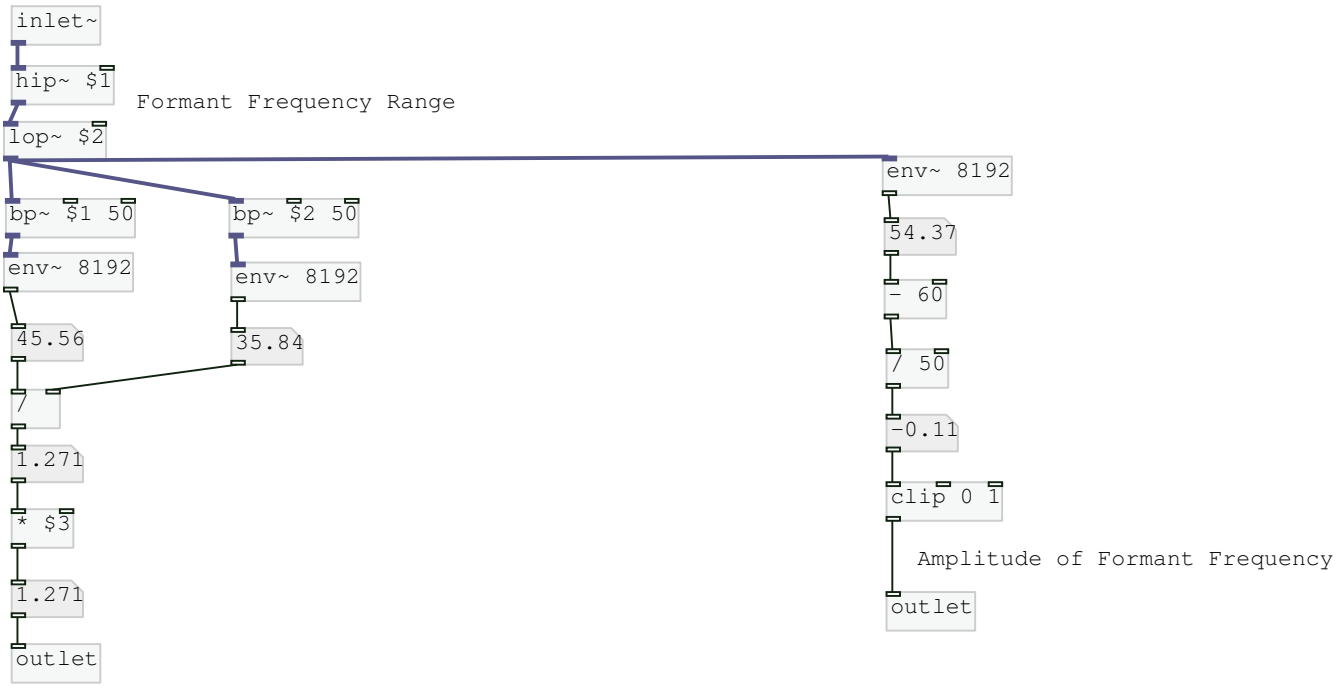

Figure 4: `formant_freq_amp.pd`

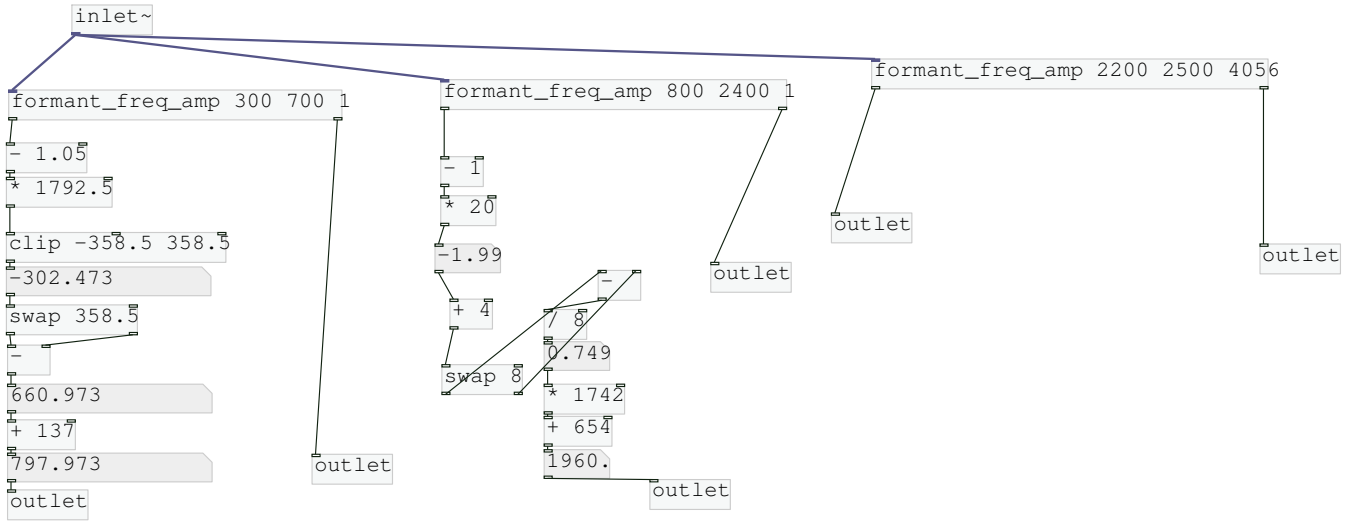

Figure 5: `formant_reader.pd` uses three abstractions to find F_1, A_1, F_2, A_2, F_3 and A_3.

2.2.2 Voicing

`voice_decision.pd`, determines the value of V; 0 for an unvoiced sound and 1 for a voiced sound.

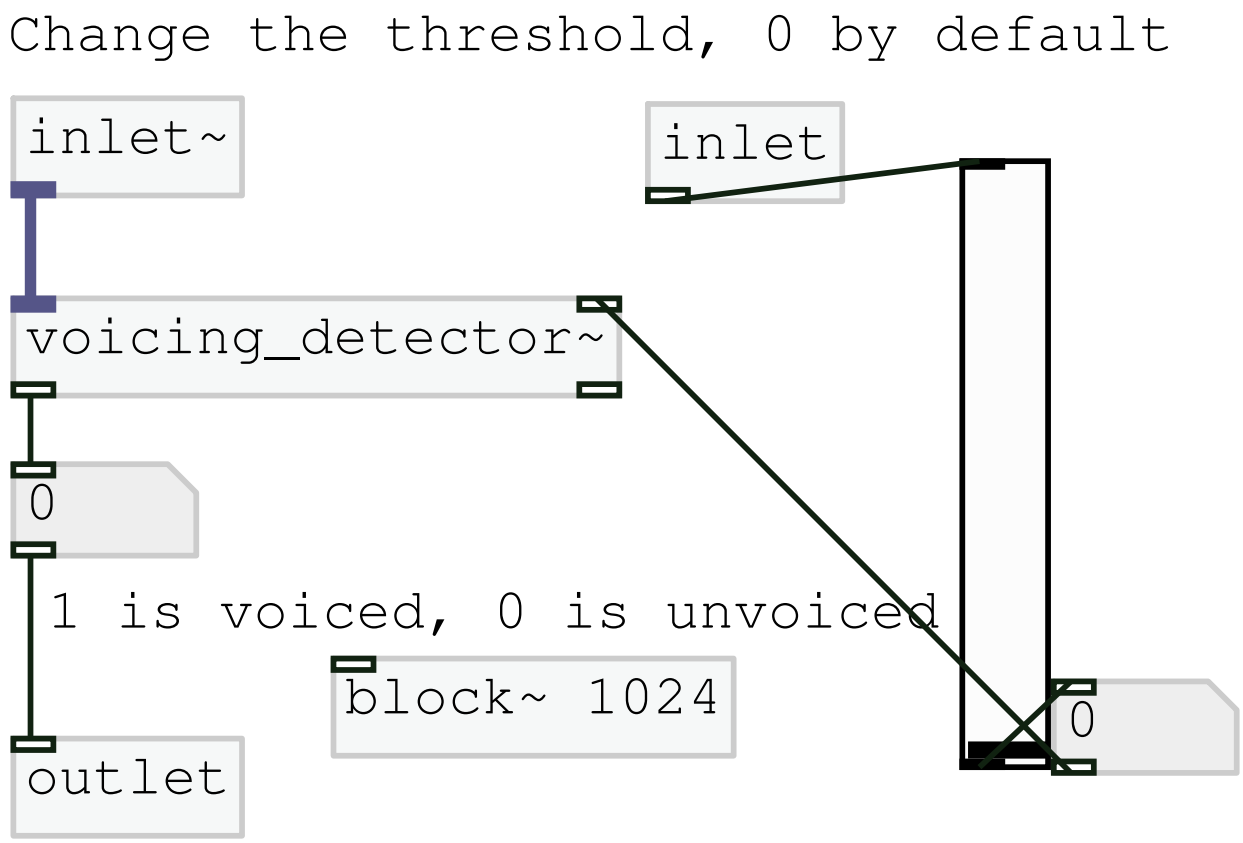

Figure 6: `voice_decision.pd`

2.2.3 Fundamental Frequency

A `fiddle~` object calculates the pitch and an `mtf` object calculates the fundamental frequency F_0 from the pitch.

2.3 Generating a File

The ;ost from the first inlet is unpacked with each of the components passed through functions. Depending on the type of quantity they represent, different kinds of functions are used to turn them into a valid parameter on a line in the final `.pfs` file that will be generated. Frequencies are passed through linear functions whereas the amplitudes and voicing decision are passed through functions that are logarithmic. The pseudo-whispering decision is done by sending the value as the coefficient to the voicing calculation. If it is unvoiced it is 0, if it is voiced it is 1 multiplied by the calculation which is just the calculation.

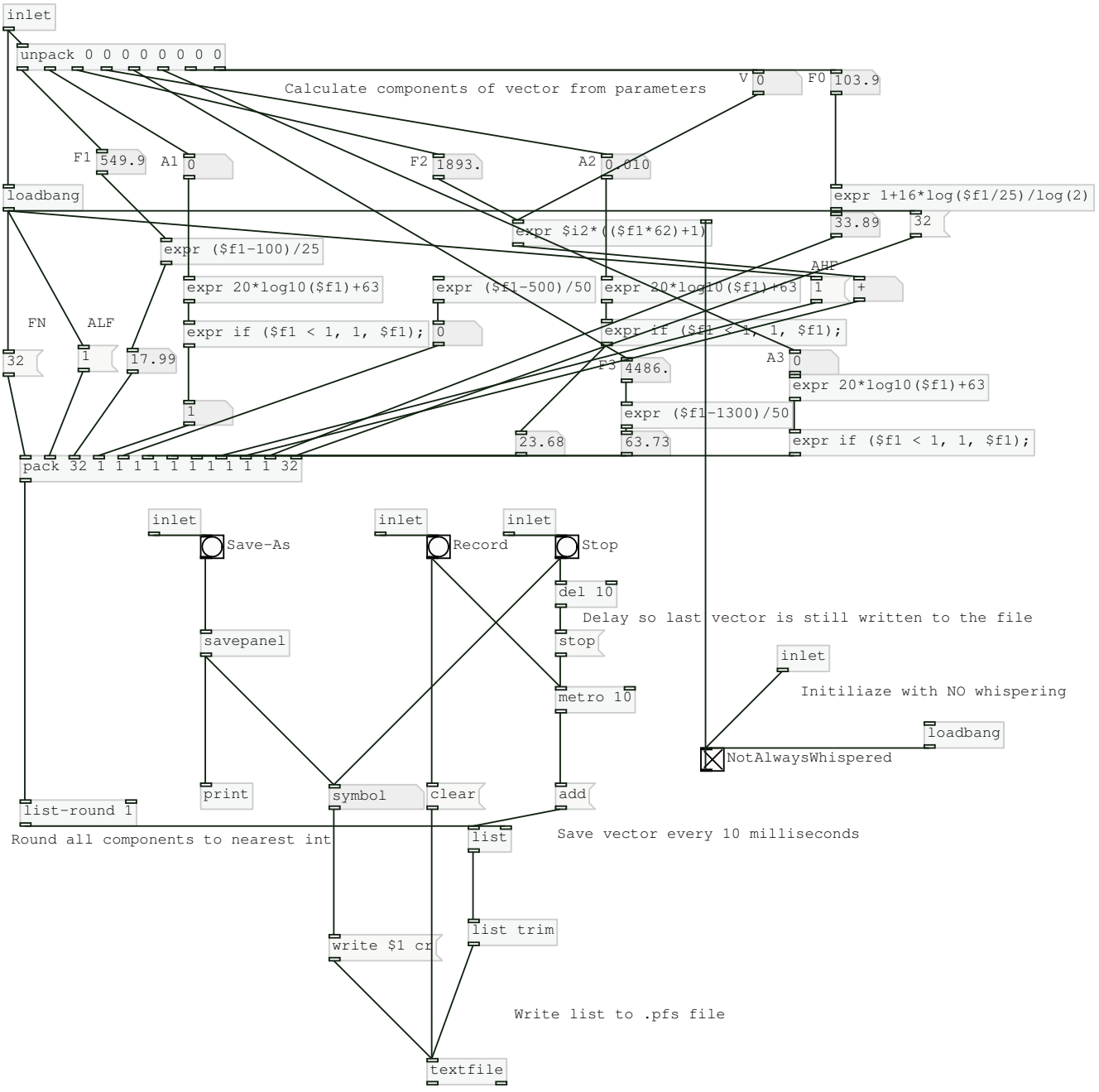

Figure 7: `filemaker.pd`

3 Evaluation

1. The quality of the output speech:
   (a) Recording normal speech is only just intelligible when there's more vowels than consonants. This is because my program treats voicing as a binary decision; either 0 or 1 (or in the files, either 1 or 63). Voicing is not enough to distinguish consonants; their method of articulation will affect how they sound; plosives [4], fricatives [5, Chapter 1], nasals [5, Chapter 2], trills etc [4].
   (b) Recording pseudo-whispered is unintelligible because of the problems with consonants mentioned above.
2. The effect of setting the frequency of F_1 to a constant value is ideally restricting the level of openness of vowels (see appendix 4.1.1).
3. The effect of setting the frequency of F_2 to a constant value is ideally restricting the level of backness of vowels (see appendix 4.1.2).
4. A way to formally evaluate performance:
   (a) Use the program to convert the `.wav` file to `.pfs`.
   (b) Convert the output file back into a `.wav`.
   (c) Repeat step 1 using the new `.wav` file until the output `.wav` is totally unintelligible. Counting how many runs it takes for this to occur.

4 Appendices

4.1 Constant Formants

4.1.1 Constant First Formant

`speech_min_f1.pfs` represents with vowels restricted to close vowels e.g. [i], [y], [ɨ], [ʉ̟], [ɯ] or [u]. `speech_max_f1.pfs` represents speech restricted to the open vowels e.g. [a], [œ], [ä], [ɞ], [ɑ] or [ɒ].

4.1.2 Constant Second Formant

`speech_min_f2.pfs` represents speech with vowels restricted to back vowels e.g .[ɯ], [u], [ʏ] , [o], [ɣ̟],[o̟], [ʌ], [ɔ], [ɑ] or [ɒ]. `speech_max_f2.pfs` represents speech with vowels restricted to front vowels e.g. [i], [y], [e], [ø], [e̟], [ø̟], [ɛ], [œ], [æ], [a] or [œ].

4.1.3 Miscellaneous

I used the print option on the menu to export PostScript files, which unlike print-screening, yields a vector image as opposed to a raster image.

5 References

[1] K. Nakata and A. Ichikawa, *Speech synthesizer*, US Patent 3,532,821, Oct. 1970. [Online]. Available: `http://www.google.ch/patents/US3532821`.

[2] J. E. Markel and A. H. Gray, *Linear Prediction of Speech*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1982, ISBN: 0387075631.

[3] M. Puckette, T. Apel, and D. D. Zicarelli, "Real-time audio analysis tools for pd and msp," Oct. 1998.

[4] M. Ashby and J. Maidment, "Manner of articulation," in *Introducing Phonetic Science*, ser. Cambridge Introductions to Language and Linguistics. Cambridge University Press, 2005, pp. 51–68. DOI: `10.1017/CBO9780511808852.004`.

[5] J. N. Holmes and W. Holmes, *Speech synthesis and recognition*, English, 2nd ed. London ; New York : Taylor & Francis, 2001, Previous ed.: Wokingham : Van Nostrand Reinhold, 1988, ISBN: 0748408568 (hbk.)