FZERO∼: FUNDAMENTAL ESTIMATIORN FOR MAX 6

Michael Zbyszyński
Cycling '74
San Francisco, CA, USA
mzed@cycling74.com

David Zicarelli
Cycling '74
San Francisco, CA, USA
zicarelli@cycling74.com

Regina Collecchia
Center for Computer Research in Music and Acoustics, Stanford University
Stanford, CA, USA
colleccr@ccrma.stanford.edu

ABSTRACT

fzero∼ is a monophonic, wavelet-based, real-time fundamental estimation object released as standard part of the Max 6 distribution. It was designed to provide usable results in a large variety of cases with a minimum of parameter modification by the user. It implements a Fast Lifting Wavelet Transform (FLWT) using the Haar Wavelet. The object provides an efficient estimation of the fundamental frequency in a large variety of real-world situations.

1. INTRODUCTION

The fundamental frequency (f_0) of a musical sound is one of its defining parameters and is key to the perception of pitch and melody.[1] The usefulness of quickly and reliably estimating f_0 has been demonstrated by numerous musical works and research papers, but fundamental estimation is still a difficult engineering problem and real-time implementation is non-trivial for musicians.

MSP was introduced almost 15 years ago, adding audio capabilities to the popular, real-time programming environment Max.[2] Since then, many excellent f_0 and pitch tracking objects have been developed (see §1.2). However, identifying the best object and configuration to use in a specific musical context is still challenging for many users. Musicians often use the fiddle∼ object, a pitch follower that can decompose a signal into sinusoids, which works well many cases. However, when fiddle∼ does not generate accurate results it can be difficult to understand why and adjust accordingly. Although there are multiple adjustable parameters in the object, finding an optimal trade-off between efficiency and accuracy can be elusive, and it is very hard to develop an intuition for the parameters and behaviour of FFT-based pitch analysis. Consequently, we felt there was an open niche in the Max environment for a simple, general-purpose fundamental estimation tool to supplement the existing objects and act as a starting point for musicians.

1.1. Design principles for the object

Discussions when designing this object were primarily concerned with the user experience rather than specific mathematics. Although we chose a pitch tracking algorithm that was not represented in previous Max pitch detectors, scientific novelty was not the goal.

The motivation was to create an object whose default behavior was acceptable in a large range of scenarios, and that didn't require extensive manipulation to yield usable results. Specifically, we wanted an object that provided

- default f_0 estimation results for a variety of source sounds, even noisy ones;
- intuitive control for musicians with presets for common musical instrument sources, and the ability to balance accuracy with latency in a straightforward way;
- basic onset detection;
- efficient, real-time performance. [3]

It wasn't clear if one pitch tracking algorithm could fulfill all of these goals in a wide variety of cases, or if we would need to develop a hybrid approach with different algorithms for different instruments or conditions. Typical pitch tracking patches will often include filters to remove noise outside of the desired frequency range, and other processing to compensate for dynamic variation. We considered building these features into the object, perhaps applying them dynamically. Our first step was to survey the existing field of fundamental estimation algorithms for Max and identify their strengths and shortcomings.

[1] The term fundamental frequency is used in this paper to refer to the empirical value of the lowest partial in a harmonic waveform, while pitch is used to describe the perceptual phenomenon. Often, these terms are interchangeable.

[2] http://cycling74.com/products/max/

[3] Efficiency was not specifically defined at this point, but was understood as using a relatively small (¡10%) amount of the CPU on a typical laptop.

1.2. Existing pitch detectors

Miller Puckette has written a series of pitch trackers using FFT feature detection that have been ported to Max. The oldest of these is `pt~`, written for Max/FTS on the IRCAM Signal Processing Workstation [13], as were the later objects `pitch~` and `jack~`. This object has been superseded by `fiddle~` [12], and more recently `sigmund~` .[4]. These objects have been widely adopted by Max users, and are effective in a wide range of circumstances. These objects also output information about partials other than the fundamental.

As the name "fiddle" suggests, this object is excellent for tracking violin signals. But frequency resolution and general performance suffer below about 200 Hz, the frequency of the lowest string on a violin. This is problem common to FFT-based approaches: poor low frequency resolution, a natural consequence of insufficient samples in a given window. At a sampling rate of 44.1KHz, a 2048-sample FFT window has a frequency resolution of approximately 11 Hz. While this is less than a semitone at 200 Hz, it is more than a whole step at 82 Hz: approximately the bottom string of a guitar in standard tuning. Dobrian [6] notes that `fiddle~` has difficulty in cases where the music is not "conceptually organised as discrete notes each having a single stable fundamental pitch."

Tristan Jehan has written a family of analysis objects that are based on `fiddle~` and suffer from the same characteristics listed above. The `analyzer~` object includes pitch tracking and other perceptual information such as loudness, brightness, and noisiness. Jehan's `pitch~` object outputs a subset of that data, including the pitch and amplitude of the fundamental and higher partials [8].

The `yin~` f_0 estimator uses autocorrelation and cancellation to estimate the frequency [2]. This approach has been shown by Obin [11] to be very robust for plucked or struck strings. An implementation is available through IRCAM's online forum.[5]

Arsia Cont's `transcribe~` object [3] tackles polyphonic pitch detection by using non-negative decomposition techniques. `iana~`, by Todor Todoroff, uses a frequency-domain approach derived from Terhardt and reports a large amount of spectral data about the incoming signal. It has been used effectively for real-time analysis and resynthesis [14]. Both of these are also available through the IRCAM forum.

Each of these objects have strengths and perform best under the circumstances for which they were designed. An expert user should desire access to many tools that solve a wide range of musical problems. But to the more common user of Max, the choices can be overwhelming and using them requires some finesse. Furthermore, inclusion of proprietary objects in a musical composition make that piece more difficult to share and preserve, which is a serious consideration for computer musicians.

In summary, the most commonly used objects (fiddle , yin) implement either FFT-based feature detection or autocorrelation. In addition to these known approaches, two other methods that had not yet been explored in Max held promise. These were wavelet transforms and multiple FFT methods (e.g. cepstrum, modulation spectrum, or "Fourier of Fourier transforms"). When this project began in 2009, Marchand's "FFT of FFT" approach [10] was explored. Although this still looks promising, it was determined to be too computationally expensive for our applications. Wavelet transforms had been relatively untested in Max, but seemed to offer the potential to meet all of our design principles while also introducing a new tool to the array of Max analysis objects.

2. OBJECT DESIGN

The design of the fzero max object can be described from two perspectives: the implementation of the underlying algorithm and the presentation of the algorithm's functions to the user. Although this paper goes into details of the algorithm, it was important to our design process that deep knowledge of the underlying algorithm is unnecessary for the end user.

2.1. Algorithm

The underlying mathematics for `fzero~` can be found in Larson and Maddox [9] and is echoed below in §2.1.1 and §2.1.2. This algorithm stood out because

- the wavelet transform has shown promising results elsewhere [7],
- it has good low-frequency resolution (limited by the need to fit at least two periods of the fundamental into a buffer),
- it shows resistance to noise (Larson reports good results up to a SNR of 20-25 dB),
- it can be effective on signals with relatively weak fundamentals,
- there was potential for computational efficiency and hence real-time implementation.

The following subsections list the components of the algorithm, step by step. The analysis is performed on a buffer of samples who's default size is 2048. The sample rate in Max is configurable by the user separately from this object.

2.1.1. FLWT

The first step in the fundamental estimation algorithm is a Fast Lifting Wavelet Transform, using a Haar wavelet. Using the jargon of wavelets, this transform splits the signal into **approximation** and **detail** components written $d(n)$ and $a(n)$. Respectively, these components are

[4] http://crca.ucsd.edu/ tapel/software.html

[5] http://imtr.ircam.fr/imtr/Max/MSP_externals

equivalent to a downsampling low-pass (approximation) and upsampling high-pass (detail) filter. The transform can be re-applied to the approximation, further isolating the fundamental frequency. The equations used are from Daubechies and Sweldens [5]:

$$d_0(n) = x(2n+1) \quad (1)$$
$$a_0(n) = x(2n) \quad (2)$$
$$d_1(n) = d_0(n) - a_0(n) \quad (3)$$
$$a_1(n) = a_0(n) + d_1(n) \quad (4)$$

where $x(n)$ is the original signal, $a_1(n)$ the first approximation, and $d_1(n)$ the first detail. Since we are not so concerned with the high-pass characteristics, the detail component $d_1(n)$ is ignored. The approximation component is equivalent to a first-order, downsampling low-pass filter, derived from the equations above:

$$a_1(n) = \frac{x(2n+1)+x(2n)}{2} \quad (5)$$

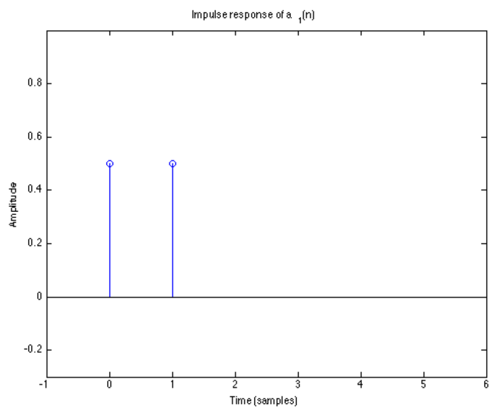

Figure 1. The impulse response of the Haar approximation component $a_1(n)$.

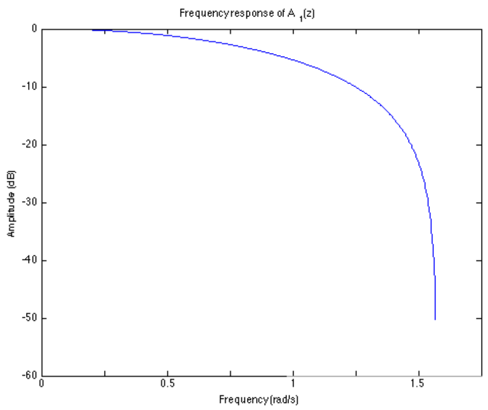

Figure 2. The frequency response of the Haar approximation component $A_1(z)$, a simple lowpass filter. Above, $F_s = 2\pi$.Therefore, the maximum frequency component is equivalent to $F_s/4$, half the Nyquist frequency due to downsampling.

Iterating this filter multiple times on the resulting signal allows the algorithm to focus on the fundamental frequency and to discard noise and higher partials. This imposes an upper frequency limit of detectable pitches at about 5% of the sample rate. We had success tracking pitches up to 2500 Hz at a 44.1KHz sample rate), which is approximately the highest note for a Western, concert flute. Although this eliminates higher pitches that are occasionally used in musical performance, the trade-off is increased resolution in the lower frequency ranges where fundamentals are more likely to occur, compared to FFT-based analyses.

2.1.2. Finding local maxima

After each level of transformation, the algorithm detects local minimum and maximum peaks in the signal. First a DC blocking filter is applied, then direction changes over a specified amplitude threshold are recorded. The peaks must be more than a specified number of samples δ apart, which is determined by the maximum frequency attribute F and the wavelet level i:

$$\delta = \max\left(\left\lfloor \frac{F_s}{2^i F} \right\rfloor, 1\right). \quad (6)$$

Once peaks have been found, the distance between a given peak and a number of subsequent peaks is determined. This helps correctly identify the fundamental in cases where a higher partial is creating peaks between those peaks with represent the fundamental. These peaks are filtered out in later analysis levels.

A potential problem for many analysis methods is when a note begins or changes in the middle of the buffer. By “buffer,” we mean window or frame as the terms are used in a windowing function such as the Short Time Fourier Transform. This has been somewhat mitigated by analyzing peaks starting from the most recent sample and moving toward the oldest, stopping when an adequate number of peaks have been detected.[6] Many cases that were initially difficult, such as samples that come from an attack transient, reverberation, or a note that has ended, are now effectively suppressed.

2.1.3. Determining mode distance

Once the peak-to-peak differences have been calculated, they are analyzed to determine a *mode distance*, i.e., . The number of similar distances (i.e. within a specified tolerance, δ) is counted, and the distance with the most near neighbors is taken to be the center mode. The mode from the previous analysis is used to distinguish between close cases, biasing the selection toward the previous mode. In the case of a tie involving one mode that is twice as long as a second, the longer mode is chosen.

To increase resolution at higher frequencies, the mean of the distances within δ of the period estimate at that level of the algorithm is taken. This has negligible effect at lower frequencies.

[6] Based on our testing, 16 maxima and 16 minima were more than enough to return the correct result, so the buffer size multiplied with 2^{-7} is a good rule of thumb.

2.1.4. *Checking against previous level*

The results from the first pass of the transform are re-analyzed following the same procedure, up to a maximum of five levels. In later levels, the algorithm checks at this point to see if the current mode distance (after adjusting for downsampling) is the same as that of the previous level. If so, that is taken that to be the period of the signal and it is converted into a frequency and reported. Otherwise, the algorithm moves to the next level and starts the process again. If the level limit is exceeded, it is assumed that the signal is unpitched and no result is reported. Also, the analysis stops if no peaks are found, which is often the case for high frequency inputs.

We initially considered that the maximum number of levels would be a user-configurable feature. However, we discovered that the algorithm did a good job of determining how many levels to run based on the input signal. Exposing this parameter would have added complexity for the user with little or no benefit.

2.1.5. *Onset detection*

The first version of `fzero~` did not include onset detection, but early testers convinced us that this is a necessary function. The object looks for changes in amplitude or estimated pitch, and reports an onset when either criterium is met. This method is effective and relatively unsophisticated.

The exact moment when a note starts is problematic for pitch detection because many instruments have frequency-rich attack transients that obscure the fundamental frequency. This particular algorithm, like many others, requires a few periods of f_0 to be buffered before it can be properly identified, so while knowing the exact onset time is very useful musically, this feature has the side effect of occurring at moments where the analysis does not perform optimally.

2.2. Max Object

As previously stated, the goal of `fzero~` is to present users with a simple interface that works "out of the box." In most cases, it should work well with the default settings. The following parameters have been exposed for advanced users and/or exceptional cases.

2.2.1. *Outputs*

The `fzero~` object has three outputs. From left to right, these are

- a floating-point value in Hertz when a fundamental within the specified parameters has been detected,[7]
- peak amplitude of the last analysis buffer,
- a bang[8] when a new onset has been detected or a list of the pitch and amplitude that triggered the onset report.

2.2.2. *Object attributes*

`fzero~` analyzes a circular buffer of samples, the size of which is set by the **size** attribute (in samples, the default = 2048). The buffer size affects both latency and efficiency. Larger buffers generally provide a more accurate analysis, especially for low fundamental frequencies, with a penalty in efficiency and latency. These penalties are minimized by the self-scaling aspects of the algorithm.

The **period** attribute[9] (in samples, the default = 256) determines how often the analysis is run. It is usually fewer samples than the buffer size, allowing the object to reduce the time between analysis results while retaining the accuracy of a large buffer.The FLWT is not re-run on the entire buffer; it is only performed on samples that have arrived since the previous analysis. However, the peak detection and subsequent steps must be run on the whole buffer for every period. A shorter analysis period has lower latency, but is less efficient. Also, analyzing more frequently decreases the length of time a previous estimate will influence the current results. The results are more prone to jitter with smaller periods, as in the piano's timbre with overtones that sustain while the fundamental disappears. Increasing the period to at least 512 samples yields a smoother result, which might be desirable in some cases.

The **threshold**, **freqmin**, and **freqmax** parameters limit the cases where `fzero~` will output a result. If the peak amplitude of the buffer does not exceed the specified threshold, the object will not do an analysis. This filters out meaningless estimates from a noisy signal. **Freqmax** sets the minimum distance between peaks (see equation (3)), limiting the maximum analyzed frequency and making the calculation more efficient. **Freqmin** filters out low frequency results, but has no effect on the calculations.

Onsetamp, **onsetpitch**, and **onsetperiod** affect the onset detector. The first two parameters set the amplitude threshold and the amount of pitch deviation (respectively) that would trigger an onset report. **Onsetpitch** is set in floating point MIDI values, allowing for a consistent perceptual distance over the whole frequency range. Optionally, the **onsetlist** attribute causes the object to report the pitch and amplitude that triggered the onset report.

2.2.3. *Extra information in the help file*

The quality of any fundamental estimation algorithm is highly dependent on the quality of the input signal, so helping users is as important as a sophisticated algorithm. The `fzero~` object is effective at suppressing noise and normalizing input amplitudes, but it still performs best

[7] We considered that this output should be in MIDI note values, but this conversion is trivial with Max's `ftom` object.

[8] A "bang" is a special Max message that generally initiates an event.

[9] The denotation of "period" here is equivalent to "hop size" in a windowing function.

with pitched, monophonic input. The final section of the help file provides practical tips on microphone placement and choices for better results.

We added a collection of common western instrument ranges into the help file to make it easier for users to set the correct range of analyzed frequencies. Originally, we thought that the object might accept instrument names as messages and use them internally to set multiple parameters. However as the object developed, fewer parameters needed adjusting. Setting the desired frequency range was deemed adequate and adding this information to the help file exposes it to be reused across the entire application.

3. CONCLUSIONS & FUTURE WORK

Max 6 and `fzero~` have only been broadly distributed for a few months; there is not yet a substantial volume of firsthand data. The array of musical instruments, microphones, and acoustic situations where it could be used is dauntingly large, and it remains for future research to quantify how this algorithm performs in real-world situations. Empirical data comparing many of the objects cataloged in this paper would be of great use to the community and future developers. Currently, the reports have been favorable.

In an informal survey of users, most reported CPU usage of 1-2% of the audio thread in their normal performance configuration. The worst machine was a 32-bit, MacBook Pro, 2GB Core Duo, where `fzero~` consumes 7-9% of CPU cycles with normal settings.

Currently, `fzero~` assumes that a fundamental detected in the previous frame is accurate and useful for determining the current estimate. The effect is usually beneficial, but can also serve to reinforce bad estimates. Development in this area might be fruitful.

4. ACKNOWLEDGEMENTS

This work was supported by Cycling '74. Joshua Kit Clayton, Timothy Place, Emmanuel Jourdan, and Rob Sussman contributed to the final code. Emmanuel Jourdan and Ben Bracken contributed important testing input. Adrian Freed at CNMAT started me off in the right direction on pitch detection algorithms. John MacCallum and Richard Dudas contributed important ideas along the way.

5. REFERENCES

[1] Camacho, A. *SWIPE: A sawtooth waveform inspired pitch estimator for speech and music*, Doctoral Dissertation, University of Florida, Florida, 2007.

[2] de Cheveigne, A. & Kawahara, H. "Yin, a fundamental frequency estimator for speech and music," *Journal of the Acoustical Society of America*, 111(4), 2002.

[3] Cont, A., Dubnov, S. & Wessel, D. "Realtime multiple-pitch and multiple instrument recognition for music signals using sparse non-negative constraints," *Proc. 10th Int. Conference on Digital Audio Effects*, Bordeaux, France, 2007.

[4] de la Cuadra, P., Master, A. & Sapp, C. "Efficient pitch detection techniques for interactive music," *Proc. 2001 International Computer Music Conference*, Havana, Cuba, 2001.

[5] Daubechies, I. & Sweldens W. 1998. "Factoring wavelet transforms into lifting steps," *Journal of Fourier Analysis and Applications*, Cambridge, MA, 1998.

[6] Dobrian, C. "Strategies for continuous pitch and amplitude tracking in realtime interactive improvisation software," *Proc. 2004 Sound and Music Computing Conference*, Paris, France, 2004.

[7] Fitch, J. & Shabana. W. "A wavelet-based pitch detector for musical signals," *Proc. 2nd COSTG6 Workshop on Digital Audio Effects*, 1999.

[8] Jehan, T. *Musical Signal Parameter Estimation*, MS Thesis in Electrical Engineering and Computer Sciences from IFSIC, University of Rennes 1, France. CNMAT, Berkeley, 1997.

[9] Larson, E. & Maddox, R. "Real-time time domain pitch tracking using wavelets," *Proc. U. of Illinois at Urbana Research Experience for Undergraduates Program*, 2005.

[10] Marchand, S. "An efficient pitch-tracking algorithm using a combination of Fourier transforms," *Proc. COST G-6 Conference on Digital Audio Effects*, Limerick, Ireland, 2001.

[11] Obin, N. *Evaluation des algorithmes d'estimation de la fréquence fondamentale dans le cadre de signaux musicaux monophoniques*. IRCAM/CNMAT, UC Berkeley, USA, 2005.

[12] Puckette, M., Apel, T., & Zicarelli, D. "Real-time audio analysis tools for PD and MSP," *Proc. International Computer Music Conference*. Ann Arbor, USA, 1998.

[13] Puckette, M., "FTS: A real-time monitor for multiprocessor music synthesis," *Computer Music Journal* 15(3), MIT Press, Cambridge, MA, USA, pp. 182-185, 1991.

[14] Todoroff, T., Daubresse, E. & Fineberg, F. "IANA: A real-time environment for analysis and extraction of frequency components of complex orchestral sounds and its application within a musical realization," *Proc. International Computer Music Conference*, Banff Centre for the Arts, Canada, 1995.