A PARAMETRIC EQUALIZER WITH INTERACTIVE POLES AND ZEROS CONTROL FOR DIGITAL SIGNAL PROCESSING EDUCATION

Andrea Casati and Giorgio Presti and Marco Tiraboschi

Laboratorio di Informatica Musicale (LIM)
Department of Computer Science, University of Milan
Milan, IT
marco.tiraboschi@unimi.it
giorgio.presti@unimi.it

ABSTRACT

This article presents *ZePolA*, a digital audio equalizer designed as an educational resource for understanding digital filter design. Unlike conventional equalization plug-ins, which define the frequency response first and then derive the filter coefficients, this software adopts an inverse approach: users directly manipulate the placement of poles and zeros on the complex plane, with the corresponding frequency response visualized in real time. This methodology provides an intuitive link between theoretical filter concepts and their practical application. The plug-in features three main panels: a filter parameter panel, a frequency response panel, and a filter design panel. It allows users to configure a cascade of first- or second-order filter elements, each parameterized by the location of its poles or zeros. The GUI supports interaction through drag-and-drop gestures, enabling immediate visual and auditory feedback. This hands-on approach is intended to enhance learning by bridging the gap between theoretical knowledge and practical application. To assess the educational value and usability of the plug-in, a preliminary evaluation was conducted with focus groups of students and lecturers. Future developments will include support for additional filter types and increased architectural flexibility. Moreover, a systematic validation study involving students and educators is proposed to quantitatively evaluate the plug-in's impact on learning outcomes. This work contributes to the field of digital signal processing education by offering an innovative tool that merges the hands-on approach of music production with a deeper theoretical understanding of digital filters, fostering an interactive and engaging educational experience.

1. INTRODUCTION

Digital Signal Processing (DSP) is a fundamental discipline not only in many subfields of Sound and Music Computing, but also more broadly in Computer Science and Communication Engineering, as demonstrated by the presence of dedicated panels in ERC calls [1]. In particular, the Z-transform and its relationship to Linear Time-Invariant (LTI) filters is one of the most important topics in DSP courses.

As with most mathematical subjects, a formal treatment is essential for teaching students the concepts behind LTI filters. However, intuition and visualization often prove to be powerful allies, both for approaching the topic initially and for gaining a deeper understanding. To this end, DSP lecturers often use the "rubber sheet" metaphor: placing a zero on the complex (Gaussian) plane is like pinning down the rubber sheet, while placing a pole is like lifting a point of the sheet toward infinity. The entire sheet deforms according to these principles, and the height of the sheet along the unit circle determines the magnitude of the frequency response.

Although this is a powerful metaphor, it has some limitations. First, it conveys nothing about the phase of the frequency response. Moreover, it is not quantitative, as it does not indicate how much the rubber sheet is stretched. Additionally, the representation of classical electronic filters is seldom addressed. Finally, many students struggle to imagine how filters actually sound based solely on their poles and zeros.

A simple visualization of a filter's poles and zeros can be easily implemented, for example, by writing a Python or MATLAB script, or even by creating a Max/MSP patch (an activity that could be pedagogically valuable in itself). However, allowing for direct graphical manipulation of poles and zeros on the Gaussian plane introduces implementation complexity that often exceeds the scope of introductory DSP courses.

Nonetheless, we believe that this mode of interaction, combined with multisensory (visual and auditory) feedback, can be extremely beneficial in helping students achieve the kind of intuitive understanding we aim for in our teaching.

For this reason, we propose an open-source, cross-platform, ready-to-use stand-alone application and audio plug-in that enables students and lecturers to manipulate the position of poles and zeros on the complex plane, with real-time visualization and audio processing of input signals. This tool is intended to foster a better understanding of the underlying DSP concepts and to provide an intuitive foundation that supports more natural reasoning on the complex plane, following the learning-by-doing and explorative learning paradigms [2, 3]. Previous studies have shown that - given that students already have enough knowledge to understand what is happening [4] - such tools, when properly designed [5], can be very effective in improving learning activities. [6]. Although this software could, in principle, be used for music production or as a filter designer, this are not intended purposes.

In Sec. 2, we provide an overview of related tools. In Sec. 3, we describe the implementation of the application. In Sec. 4, we report a preliminary evaluation of the proposed tool. Sec. 5 concludes the document.

2. STATE OF THE ART

Many similar projects have been proposed in the literature. For example, the *ZtransformApplet* [7] is a tool that allows users to

edit filter coefficients and listen to the results by filtering a white noise signal, while displaying the difference equation, the transfer function, the pole-zero plot, and an adaptive block diagram. Unfortunately, this solution is limited by its low filter order and the fact that poles and zeros cannot be manipulated directly.

Tobias Fleischer's *Filter Explorer* [8] offers similar features: a standalone application and VST2 plug-in that allows users to enter parametric formulas for each coefficient and provides real-time visualization of the pole-zero plot, magnitude and phase response, and audio filtering. Although it is a valuable tool for designing filters and thus a commendable educational resource in that sense, like *ZtransformApplet*, it does not support direct interaction with the Z-plane. Moreover, it is an unmaintained project destined for obsolescence, as the VST2 standard is being phased out in favor of version 3.x, and Tobias Fleischer is no longer actively developing the tool.

Other similar tools are available online as web applications, such as Nigel Redmon's *Filter Frequency Response Grapher* [9]. However, these typically focus on filter design and do not support custom audio processing or direct manipulation of poles and zeros on the Z-plane.

It is also worth mentioning MATLAB's *FVTool* (recently replaced by *Filter Analyzer* [10]), which provides a comprehensive overview of a filter's behavior given its coefficients. Nevertheless, it does not offer direct manipulation of poles and zeros and cannot process audio signals out of the box.

With respect to direct manipulation of poles and zeros, one of the earliest VST plug-ins was Rob Conde's *Zero Pole Explorer* [11]. However, aside from being discontinued and limited to the VST2 format, it supports only second-order filters. A more modern Python application that addresses the filter order limitation is Saied Salem's *digital-filter-designer* [12], although it lacks real-time audio processing and is primarily focused on filter design.

Finally, there are online versions of similar concepts, such as Nigel Redmon's *Pole-Zero Placement* [13]. These tools typically support only a limited number of poles and zeros and do not provide real-time audio processing capabilities.

3. METHOD

We implemented the software using the JUCE C++ framework [14], although the DSP and filter design components are largely independent from it. For elliptic integrals, we used the Cephes Math Library [15].

3.1. DSP

We implemented the overall filter as a cascade of second-order *filter elements*, either 2-zeros or 2-poles. We chose second-order elements to ensure the filter output is always a real-valued signal. This is achieved by constraining the two zeros (or poles) to be complex conjugates of each other. The frequency response of a 2-zeros filter, with zeros at χ and $\overline{\chi}$, is

$$H_\chi^{(z)}(z) = (1 - \chi z^{-1})\,(1 - \overline{\chi} z^{-1}) = \tag{1}$$

$$= 1 - 2\Re\{\chi\}\; z^{-1} + |\chi|^2\; z^{-2} \tag{2}$$

and the frequency response of a 2-poles filter, with poles at χ and $\overline{\chi}$, is the reciprocal

$$H_\chi^{(p)}(z) = \frac{1}{H_\chi^{(z)}(z)} = \frac{1}{1 - 2\Re\{\chi\}\; z^{-1} + |\chi|^2\; z^{-2}} \tag{3}$$

We can see that the two coefficients are real-valued and identical, regardless of whether the element is a 2-zeros or 2-poles filter.

$$c_1 = -2\Re\{\chi\} \in \mathbb{R} \tag{4}$$

$$c_2 = |\chi|^2 \in \mathbb{R}^+ \tag{5}$$

The difference equation for a filter element is

$$y[n] = x[n] + c_1\, x[n-1] + c_2\, x[n-2] \tag{6}$$

for 2-zeros elements, and

$$y[n] = x[n] - c_1\, y[n-1] - c_2\, y[n-2] \tag{7}$$

for 2-poles elements. Each filter element therefore requires two memory cells for the coefficients and two for the previous signal values.

Filter elements can optionally be configured as 1-pole or 1-zero filters. In that case, to ensure the output remains real-valued, the zero or pole is forced to lie on the real axis. The frequency response becomes

$$H_\chi^{(1z)}(z) = 1 - \chi z^{-1} \tag{8}$$

$$H_\chi^{(1p)}(z) = \frac{1}{H_\chi^{(1z)}(z)} = \frac{1}{1 - \chi z^{-1}} \tag{9}$$

with corresponding coefficients

$$c_1 = -\chi \in \mathbb{R} \Leftrightarrow \chi \in \mathbb{R} \tag{10}$$

$$c_2 = 0 \tag{11}$$

Filter elements are connected in series in a *filter cascade*, with the output of one element feeding into the next. Each element can also be deactivated. To prevent numerical instability, each element has an individual input gain parameter.

Let N be the number of active elements in the cascade. The difference equation for the i-th active filter element is

$$y_i[n] = x_i[n] + c_{i,1}\, x_i[n-1] + c_{i,2}\, x_i[n-2] \tag{12}$$

for 2-zeros elements, and

$$y_i[n] = x_i[n] - c_{i,1}\, y_i[n-1] - c_{i,2}\, y_i[n-2] \tag{13}$$

for 2-poles elements, where y_0 is the input signal, y_N is the output signal, and $x_i = k_i\, y_{i-1}$. The overall frequency response is the product of all individual frequency responses and their respective gain parameters k_i.

3.2. GUI

The GUI (Fig. 1) is organized into five panels:

- The parameter panel allows interaction with the filter element parameters.
- The plots panel visualizes the frequency response of the filter cascade (Discrete-Time Fourier Transform).
- The filter design panel is used to configure filter elements to design Butterworth, Chebyshev, and Elliptical filters.
- The master panel includes a master output gain slider and a bypass toggle button.
- The top menu offers various utility functions.

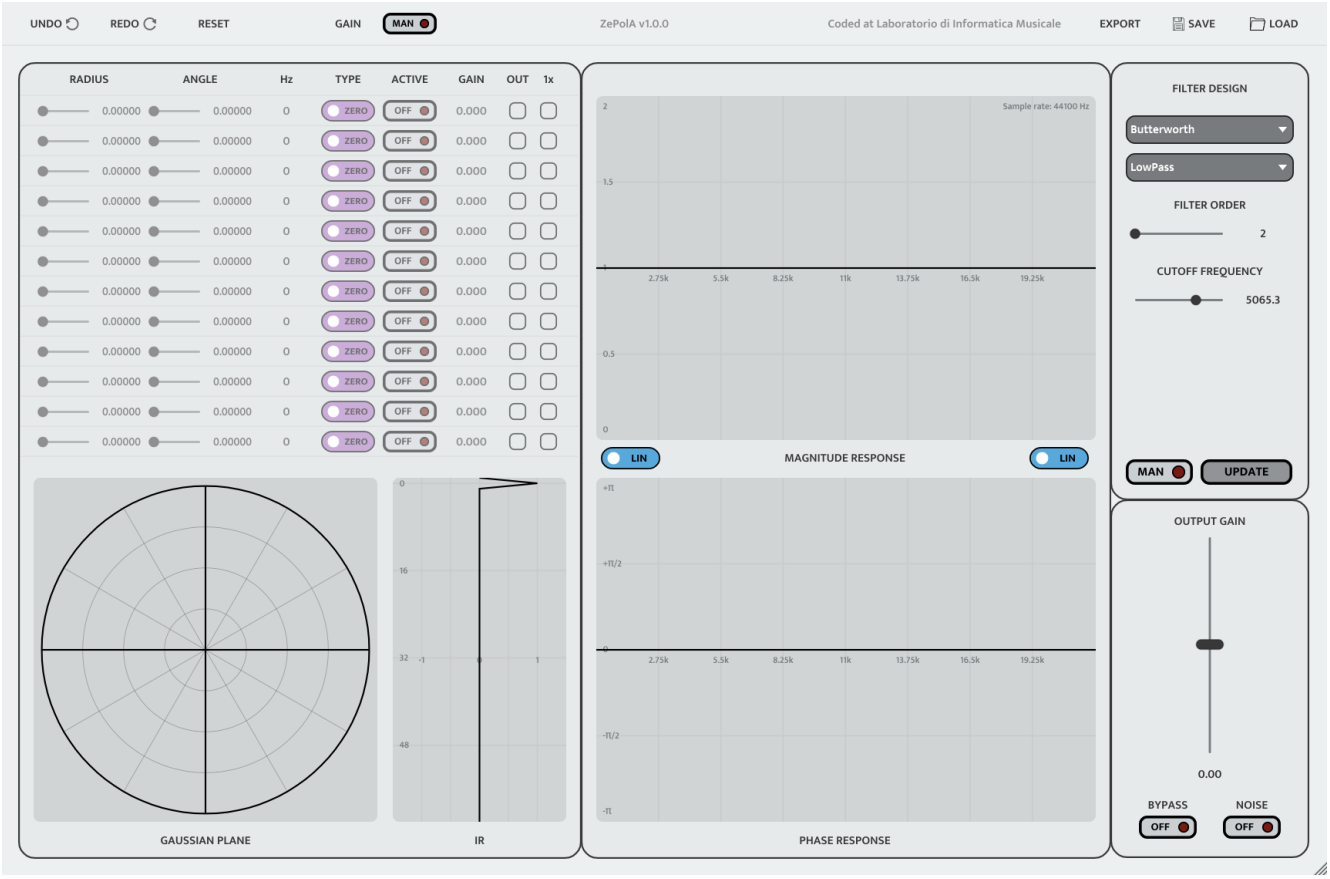

Figure 1: *The plugin GUI in its default state. All elements are off, the frequency response is constant and nothing is showing in the Gaussian plane. The left panel is the parameter panel, with all individual parameters, the Gaussian plane, and the shortcuts. The middle panel is the plot panel, with the DTFT visualization. The right panels are the filter design panel (top), and the master panel (bottom).*

3.2.1. Parameters

The parameter panel is the main focus of the GUI. In this panel, users can view and modify all parameters of all filter elements. At the top of the panel, there is a matrix of GUI components for parameter interaction. Each row corresponds to a filter element, and each column represents a parameter. Hovering over the column labels with the mouse reveals tooltips with additional information. Although the code supports any number of filter elements, we limited the maximum to 10 to balance flexibility and usability.

The *radius* slider controls the magnitude of the zero/pole position, i.e. its distance from the origin in the Gaussian Plane. Its value is constrained between 0 and 1 to keep positions inside the unit circle. However, for zeros, an additional *out* parameter allows them to be placed outside the unit circle. For poles, we impose an internal maximum radius of 0.99999 to prevent them from reaching or exceeding the unit circle, since poles on or outside the unit circle would make the causal filter unstable.

The *angle* slider sets the phase of the zero/pole position, normalized from 0 (DC component) to 1 (Nyquist frequency). The configured position always lies in the upper half of the complex plane; the conjugate with negative imaginary part is handled automatically (as illustrated in Fig. 2 and Fig. 3). Alternatively, users can enter or drag a frequency value in hertz using the corresponding editable label.

The *type* toggle button selects whether the filter element is a 2-zero or a 2-pole.

The *active* toggle button turns the element on or off. Inactive rows are shown in a lighter color for visual distinction.

The *gain* label, which is both editable and draggable, specifies the input gain for the filter element in decibels. This is useful for compensating any unwanted gain added by the element.

The *out* checkbox applies only to zeros. When activated, it inverts the effective radius, placing the zero outside the unit circle. If clicked while the element is a pole, it will convert the pole into a zero. Switching the type back to pole will reset this checkbox. This feature is useful for creating all-pass filters.

The *1x* checkbox converts the filter element into a first-order filter (1-zero or 1-pole), forcing its position to lie on the real axis.

Gaussian Plane

All parameters, except for the last two checkboxes (*out* and *1x*), can be modified by interacting with the Gaussian plane.

Right-clicking on an empty spot in the Gaussian plane will activate the closest inactive filter element, and a circle (for zeros) or a cross (for poles) will appear at the corresponding position based on the element's parameters. Right-clicking on a circle or cross will deactivate the corresponding filter element. A semi-transparent

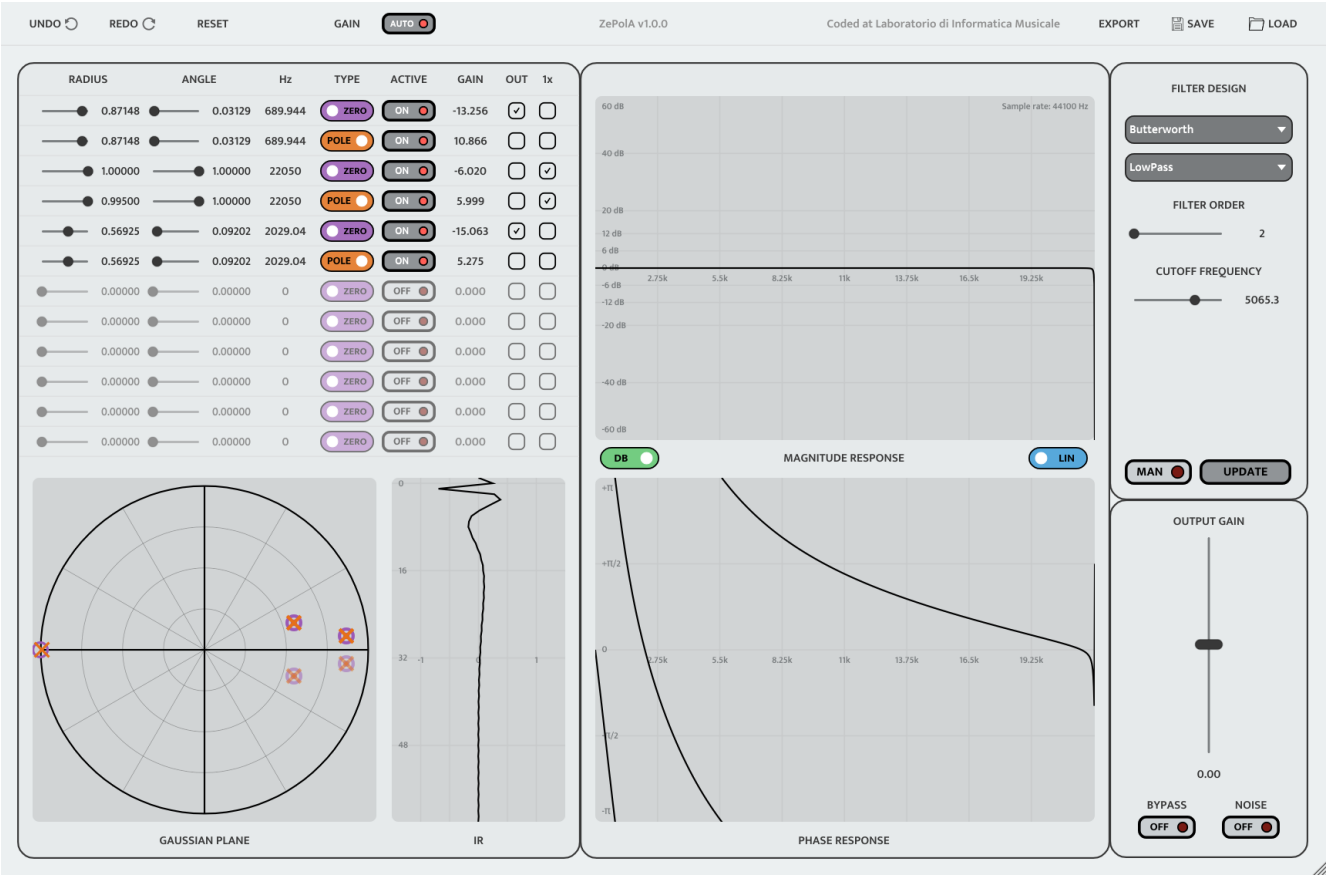

Figure 2: *The plugin GUI after setting some parameters. A cut at the Nyquist frequency is obtained by placing a 1-zero element (thinner circle) at* -1 *and a 1-pole element (thinner cross) at* -0.995*. Two matching pairs of 2-zero and 2-pole are also placed (superimposed crosses and zeros), with inverted zero magnitudes. This has no effect on the magnitude response, but only on the phase response.*

copy of the circle or cross will show the conjugate position. The circle or cross becomes thinner for first-order filter elements, and the circle will have a dot in its center if the zero's magnitude is inverted (i.e., if the *out* checkbox is checked).

Dragging a circle or cross will modify both the radius and angle parameters of the filter element. This interaction allows for exploration of the relationship between zero/pole placements and the resulting frequency response.

Double-clicking a circle or cross will toggle a 2-zero element to a 2-pole element and vice-versa. Finally, scrolling over a circle or cross will adjust the input gain for the corresponding filter element.

Shortcuts

The shortcuts panel was implemented in the prototype version of the software. It was later excluded because user experiments showed it was not useful. It used to provide overall controls for the parameters:

- the *All On* and *All Off* buttons toggled all filter elements on and off, respectively;
- the *Phases x2* and *Phases ÷2* buttons used to multiply and divide, respectively, the phases of all filter elements;
- the *Swap Ps/Zs* button used to swap zeros and poles.

3.2.2. Plots

The plots panel displays the DTFT (Discrete-Time Fourier Transform) of the digital filter. It takes into account all active elements, their input gain, and the master output gain. The top axis visualizes the magnitude response, while the bottom axis shows the phase response. The abscissa values are in hertz, and the top axis also displays a label indicating the current sample rate.

Two buttons allow for logarithmic scaling of the plots. The left button (*lin/dB*) toggles the magnitude values between linear amplitudes and decibels, while the right button (*lin/log*) switches the frequency scale between linear and logarithmic. In logarithmic frequency mode, the Nyquist frequency is displayed as the highest frequency, and 1 kHz is the central frequency.

A third plot is next to the Gaussian plane, where the shortcuts panel used to be. It shows the (truncated) impulse response of the filter in the time domain. Because of the aspect ratio of the available space, we decided to place the discrete-time (in samples) on the vertical axis and the amplitude values on the horizontal axis.

3.2.3. Filter Design

The filter design panel allows the user to define a filter using higher-level parameters, instead of directly manipulating pole and zero

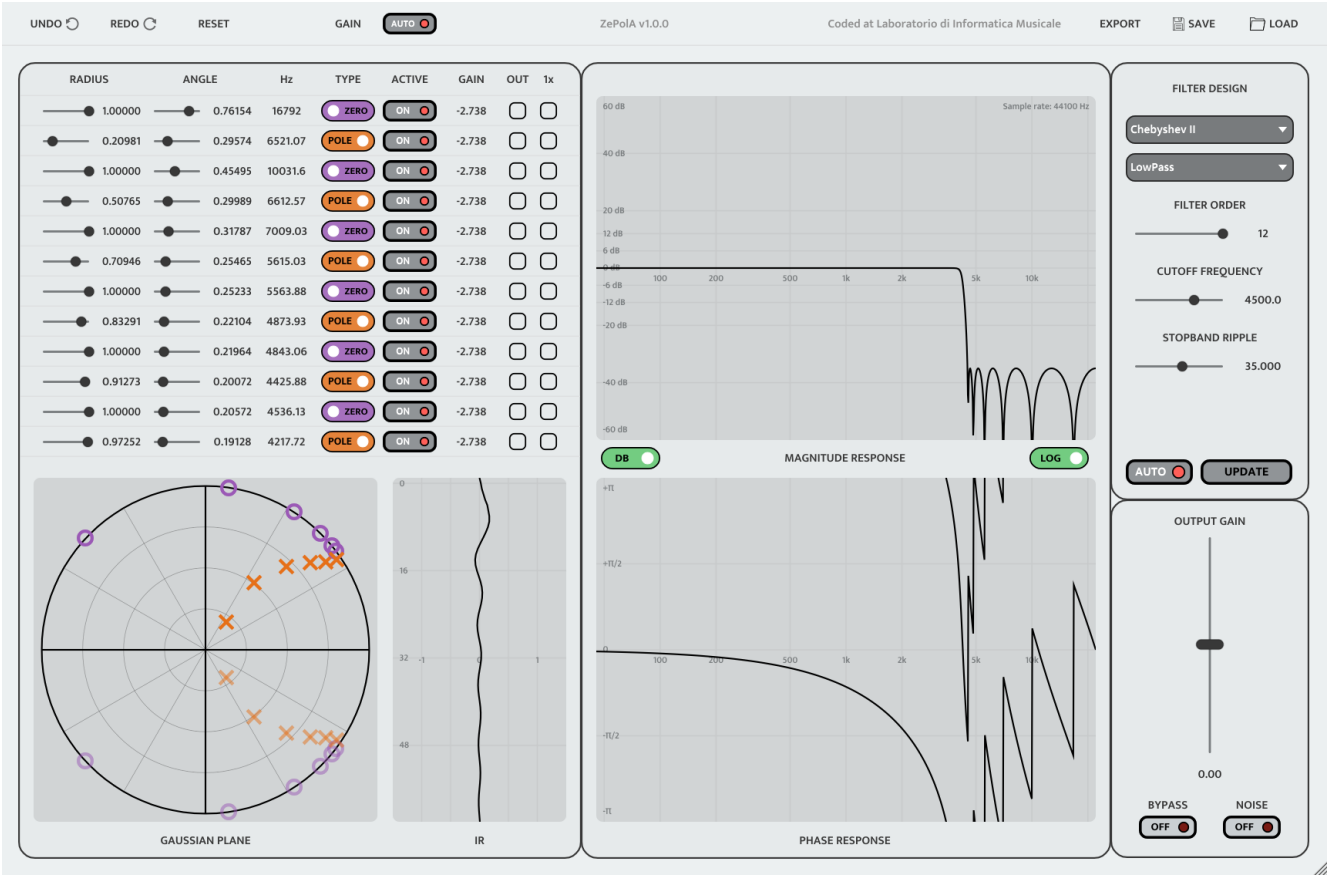

Figure 3: *The plugin GUI with parameters set though the filter designer panel to obtain a Chebyshev Type-II lowpass filter. We used the maximum filter order (12), a cutoff frequency of* 4.5 *kHz and* 35 *dB of stopband ripple.*

positions. Four filter types are currently supported: *Butterworth*, *Chebyshev Type-I*, *Chebyshev Type-II* (Fig. 3), and *Elliptic*.

For all filter types, the user can specify whether the filter should be *low-pass* or *high-pass*, the filter order (ranging from 2 to 10), and the cut-off frequency.

Chebyshev Type-I and *Elliptic* filters also include a pass-band ripple parameter (in decibels), while *Chebyshev Type-II* and *Elliptic* filters feature a stop-band ripple parameter. Only the parameters that are relevant to the selected filter type are visible.

The *Update* button applies the specified parameters to the filter elements, calculating the corresponding low-level parameters. We followed a similar approach to the filter design functions in `scipy.signal` [16]. When the *Auto* button is enabled, the filter parameters are automatically applied whenever a change is made.

3.2.4. Top Menu

The top menu houses functionalities that don't neatly fit into any other panel.

The *Undo* and *Redo* buttons allow you to revert or restore changes to the filter parameters, respectively. The *Reset* button restores all parameters to their default values.

The *Load* and *Save* buttons allow you to read or write the software parameters to an XML file. The *Export* button, on the other hand, saves the filter element coefficients to a CSV file: the i-th row contains the MA coefficients ($b_0 = 1, b_1, b_2$), the AR coefficients ($a_0 = 1, a_1, a_2$), and the gain coefficient of the i-th active filter element.

The *Auto/Man* button next to the *Gain* label toggles the auto-gain feature on or off. When enabled, the auto-gain feature automatically adjusts the gain coefficient of a filter element whenever any of its parameters change. The gain is computed using a heuristic, gradient-based optimization method that approximates the overall filter DTFT magnitude peak. To prevent excessive over-compensation, this method is stopped early when poles are close to the unit circle, as they tend to produce very high magnitude peaks. This auto-gain feature is compatible with the filter design.

3.2.5. Environment Variables

To further customize the experience with this software, we support several environment variables

`ZEPOLA_N_FILTER_ELEMENTS` selects the number of filter elements in the software. Although increasing by much this variable from the default (12) can result in a messy parameter panel and a slow auto-gain feature, it can be still useful for visualizing filters built with the filter designer.

`ZEPOLA_IR_PLOT_LENGTH` determines the length of the IR plot, in samples. Increasing by much this variable from the default (64) can result in performance loss.

`ZEPOLA_POLE_MAGNITUDE_CEIL` is the maximum radius for a pole element. Setting this to 1 makes it possible to place poles directly on the unit circle, resulting in unstable filters. Default is $1 - 10^{-5}$.

`ZEPOLA_INVERSE_MAGNITUDE_FLOOR` is the minimum radius for elements with inverted magnitudes. Setting this closer to 0 makes it possible to place poles or zeros closer to infinity, possibly causing instability or division errors. Default is 10^{-6}.

`ZEPOLA_FILTER_ELEMENT_GAIN_FLOOR_DB` is the minimum gain for an element, in decibel. Default is -128 dB. Decreasing this value by much could cause floating-point underflow errors.

`ZEPOLA_ALLOW_INVERTED_POLES` can be set to `true` to allow inverted magnitude for pole elements. This results in extremely unstable filters, but we received feedback by some people who are curious to visualize these kinds of scenarios.

4. RESULTS

To evaluate the effectiveness of the proposed tool, two focus groups were organized to gather feedback and suggestions. The first group consisted of three undergraduate students from the sound and music computing programme, while the second group consisted of three signal processing lecturers. The students had all already taken the digital signal processing exam.

The focus groups were placed in front of a single computer with the application running. The first phase involved allowing the participants to explore the tool freely without any prior training on its features. Afterward, some information was provided, and finally, an unstructured interview was conducted.

4.1. Students

Overall, the students were enthusiastic about the tool. Initially, they encountered some difficulty controlling the gain of the response, but once they understood the auto-gain feature, they expressed that they would leave it enabled by default. Apart from a few minor GUI suggestions (e.g., adding the the label *Hz* on top of the pole/zero frequency label and making the toggle styles consistent across the interface), they enthusiastically confirmed that the tool would have been highly beneficial for their learning. They mentioned that it went "beyond the formulae" and would have been especially useful if it had been introduced during or immediately after the mathematical introduction to digital filters, but always alongside the display of the mathematics involved.

4.2. Lecturers

Initially, the lecturers were somewhat puzzled, perceiving it as an unusual filter design tool. However, they soon recognized its pedagogical value and suggested additional features such as the ability to place zeros outside the unit circle, the option to display the impulse response, and the ability to force zeros and poles to be real-valued. They mentioned that they would consider using the tool in their classrooms, alongside the rubber sheet metaphor, to help visualize filter design concepts.

In response, we implemented many of the suggested GUI improvements, including the ability to place zeros outside the unit circle (using a graphical metaphor to save GUI space), the ability to configure single (real) poles and zeros, and the impulse response display. We decided not to enable the auto-gain feature by default, as we suspect that students might overlook the gain property, and we wanted to ensure that they were consciously aware of this parameter.

5. CONCLUSIONS

We presented *ZePolA*, a tool designed to facilitate intuitive learning of the DSP concepts involved in digital filters and digital LTI systems in general. The tool was evaluated by focus groups consisting of both students and professors, receiving positive feedback from both groups.

The tool is available as an open-source project on GitHub[1], along with compiled binaries for Windows, macOS, and Linux.

In future releases, we plan to add several features, including the display of the system's impulse response, visualization of filter coefficients, additional filter design techniques (such as biquadratic filters), direct manipulation of the magnitude response, an overlaid spectrum analyzer on the magnitude response, and step-by-step linear prediction of the input spectrum to track how the poles move on the plane during each iteration.

6. ACKNOWLEDGMENTS

We acknowledge the contribution of the three students and the three lecturers of the Department of Computer Science, University of Milan, who took part in the focus groups. In particular we would like to thank Davide Rocchesso for suggesting the name of this tool.

7. AUTHOR CONTRIBUTIONS

Andrea Casati: Software. **Giorgio Presti**: Investigation, Resources, Writing, Supervision. **Marco Tiraboschi**: Conceptualization, Software, Formal analysis, Writing.

8. REFERENCES

[1] European Research Council, *Panel Structure 2024 Calls*, 2023, https://erc.europa.eu/sites/default/files/2023-03/ERC_panel_structure_2024_calls.pdf.

[2] Yuichiro Anzai and Herbert A Simon, "The theory of learning by doing.," *Psychological review*, vol. 86, no. 2, pp. 124, 1979.

[3] Ference Marton and Roger Säljö, "On qualitative differences in learning: I—outcome and process," *British journal of educational psychology*, vol. 46, no. 1, pp. 4–11, 1976.

[4] Shelley Yeo, Robert Loss, Marjan Zadnik, Allan Harrison, and David Treagust, "What do students really learn from interactive multimedia? a physics case study," *American Journal of Physics*, vol. 72, no. 10, pp. 1351–1358, 2004.

[5] Wendy K Adams, S Reid, R LeMaster, SB McKagan, KK Perkins, and CE Wieman, "A study of interface design for engagement and learning with educational simulations.," *Journal of Interactive Learning Research, submitted*, 2006.

[1] https://github.com/LIMUNIMI/ZePolA

[6] Carl E Wieman and Katherine K Perkins, "A powerful tool for teaching science," *Nature physics*, vol. 2, no. 5, pp. 290–292, 2006.

[7] Silvin Willemsen, "ZtransformApplet," Available at https://github.com/SilvinWillemsen/ZtransformApplet, accessed April 05, 2025.

[8] Tobias Fleischer, "Filter Explorer," Available at https://web.archive.org/web/20080618075424/http://www.tobybear.de/p_filterexp.html, accessed April 05, 2025.

[9] Nigel Redmon, "Filter frequency response grapher," Available at https://www.earlevel.com/main/2016/12/08/filter-frequency-response-grapher/, accessed April 05, 2025.

[10] Mathworks, "Filter Analyzer," Available at https://it.mathworks.com/help/signal/ref/filteranalyzer-app.html, accessed April 05, 2025.

[11] Rob Conde, "Pole Zero Explorer," Available at https://web.archive.org/web/20220109225320/http://jaha.smartelectronix.com/, accessed April 05, 2025.

[12] Saied Salem, "digital-filter-designer," Available at https://github.com/saied-salem/digital-filter-designer, accessed April 05, 2025.

[13] Nigel Redmon, "Pole-Zero placement v2," Available at https://www.earlevel.com/main/2013/10/28/pole-zero-placement-v2/, accessed April 05, 2025.

[14] "Juce," https://github.com/juce-framework/JUCE.

[15] Stephen L. Moshier, "Cephes math library," `http://www.netlib.org/cephes`, 1984, Release 2.8: June, 2000.

[16] Pauli Virtanen et al., "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.