Visualizing DSP Concepts on the Web Using the R-Language `shiny` Package

by

Peter Squires
B.Sc., Memorial University of Newfoundland, 1988

Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering

© Peter Squires, 2017
University of Victoria

All rights reserved. This document may not be reproduced in whole or in part, by photocopying or other means, without the permission of the author.

Visualizing DSP Concepts on the Web Using the R-Language `shiny` Package

by

Peter Squires
B.Sc., Memorial University of Newfoundland, 1988

Supervisory Committee

Dr. T. Aaron Gulliver, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Wu-Sheng Lu, Departmental Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. T. Aaron Gulliver, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Wu-Sheng Lu, Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

This document describes a web application, `PeZdemoR`, that was designed using RStudio and the R computer language along with various R-packages (especially `shiny`, `signal`, and `pracma`), to visually demonstrate how pole and zero placement in the z-plane relates to the response characteristics of a digital filter. The resulting frequency response, impulse response (group delay), transfer function and difference equation are displayed. A section on audio filtering applications is included, providing spectrograms, spectral plots, time-series plots, and sound playback. Additionally, over 100 built-in examples of common filter designs and FFT windows have been developed.

Contents

List of Tables

List of Figures

ACKNOWLEDGMENTS

I would like to thank

- **my parents**, of course, without whom I would not be who I am today.
- **my loving wife, and our two wonderful children**, for supporting me in the tougher times.
- **my supervisor, Dr. T. Aaron Gulliver**, for mentoring, encouragement, and patience.
- **Dr. Wu-Sheng Lu**, for his sage advice and guidance.
- the **Government of Canada**, for partial funding under its 2014 work force adjustment (WFA) programme.
- the **online R-community** (e.g. CRAN Comprehensive R Archive Network, RStudio.com, StackOverflow.com, R-bloggers.com), for many helpful clues and snippets.

Pleasure is the state of being
brought about by what you
learn.
Learning is the process of
entering into the experience of this
kind of pleasure.
No pleasure, no learning.
No learning, no pleasure.
– Wang Ken (1483-1541), *Song of Joy*

DEDICATION

To all my family, near and far, and to our great nation – from sea to shining sea, the true north strong and free.

Chapter 1

Introduction

This report describes an interactive web-based educational tool, `PeZdemoR`, that can be used to support the teaching of basic Digital Signal Processing (DSP) concepts. These concepts include the relationship between the placements of the poles and zeros within the z-plane, and the resulting changes to the system response in both the time and frequency domains. This tool is primarily based upon the `shiny` package of the R-language, some supporting packages (e.g. `signal`, `pracma`), and a vast array of online community resources and documentation. The `shiny` package generates the necessary HTML, CSS and JavaScript code for any desired layout and behaviour for the user-interface inside the web-browser client. The `shiny` framework also manages the server-side interaction with the R-language engine.

The purpose of this project is to design an intuitive and flexible tool for learning basic DSP concepts, which students and instructors can use to experiment with different techniques and configurations, without getting overly involved in the mathematics. The application has been developed in base-R [25], with the extensive support of three R-packages `shiny` [8] (web-app development framework), `signal` [21] (signal processing), and `pracma` [5] (practical math functions, MATLAB-like).

Since the application depends upon web-browsers, the client-side is platform independent, and will work in Windows, Mac and Unix-like environments. No-cost options for server-side hosting include local (e.g. with a running RStudio application), SaaS (Software-as-a-Service) at http://shinyapps.io, or a user-implementable Linux server.

The target audience includes Teaching Assistants, students learning basic DSP, and

working engineers within industry when they are in a teaching mode, e.g. briefing executives or novices.

Similar to the Sturm and Gibson SSUM suite in *Signals and Systems using MATLAB* [30], the desire is to provide "first for practical effective demonstration, second to provide an interactive experience, and third to serve as a repository of algorithms and code". The `PeZdemoR` application serves as a proof-of-concept of what can be achieved with R and `shiny`, when coded by novice developers to quickly create prototypes of online teaching tools on the web. This is important because, after understanding the basics of digital sampling and Fourier transformations, filtering is essential to any further progress in DSP studies.

1.1 Contributions

To solve the problem of visualizing pole–zero effects on filter behaviour, this work includes these important contributions:

1. the `shiny` package is used to develop a web-based prototype application;
2. RMarkdown and RStudio are used to generate a variety of sophisticated documents, presentations, and web-apps. The use of RMarkdown serves as a middle ground between the authoring of content and the web-browser presentation of the HTML5, CSS, JavaScript, LaTeX codes;
3. DSP concepts are made easy to understand by exploring them interactively using a hands-on visual demo.

1.2 Organization of this Report

This section provides a roadmap of the remainder of this work. For each of the chapters below, there is a short summary of what the main focus is.

Chapter 2 describes in detail the open problem which is being solved, together with its context, its effects and overall motivation for research, and the challenges and limitations of the current solutions for educating novice DSP students.

Chapter 3 gives the design process, the proposed solution, design principles of the web application, overall system architecture, target users, use-case scenarios, and the programming frameworks used.

Chapter 4 describes the demonstration tool in detail, including the user-interface, interaction, ways to enter the initial data points, types of output visualization plots and equation boxes, and the settings page.

Chapter 5 provides a re-statement of the purpose and results, and some recommendations for future work. It also suggests ways to develop the concept of web applications for engineering education, and its applications.

This document was produced within RStudio [27], using the `knitr` engine ([35]) and the `pandoc` utility [17] to convert from the `RMarkdown` ([2]) format into a pdfLaTeX document.

The source-code is available on GitHub, https://github.com/popeye00/inSPiRe. Collaborations are welcomed. The web-app is currently hosted (Feb/2017) at https://inspire.shinyapps.io/PeZdemoR/, with a limited 25 hours per month on a no-cost account. Other hosting solutions also exist, either online at http://shinyapps.io, self-served by using Linux server, or preferably locally-served on a single machine or Local Area Network (LAN) by a running `RStudio` application.

Chapter 2

Visualization of Filter-Design

The first motivation for this work was to produce neater class notes and schoolwork [28] for submission to markers, beyond just physically binding distinct printouts into the same hardcopy. The desire was to combine all of the components, integrated together into one softcopy file All of the questions, answers, code, output, images, graphs, weblinks, LaTeX-formatted equations and text would all be contained within just one printable document. It was originally found that Word (MS Office Suite, with its built-in equation editor), and MATLAB (with its capabilities of Publish, Notebook, Report-Generator, and `.mlx` Live-Scripts) does have some of this, but with limited interoperability, or with proprietary outputs. `RMarkdown` has more versatile dynamic-document capability, especially within the `RStudio` application.

The second motivation was that, when studying placements of poles and zeros, and its resulting effects within the time and frequency domains, there exist few visual aids that connect together with knowledge already gained from other engineering courses. However, one exception to this is the `pezdemo` MATLAB GUI application from Georgia Tech (see Figure 2.1 for screenshot) discussed in *DSP-First* [19].

The third motivation was the discovery of the `shiny` R package for creating online web-apps, and the `signal` and `pracma` packages, which contain familiar MATLAB-like DSP functions. Any existing knowledge of MATLAB DSP coding can be exploited by adding just a few tweaks to port it to the R-language. Incorporating `Markdown` and `shiny` allows for proper publishing of the work online. As necessary, the output could be converted either into an HTML form for online (softcopy) viewing, or into various hardcopy formats (e.g. slides, PDF, Word documents, LaTeX) for printing.

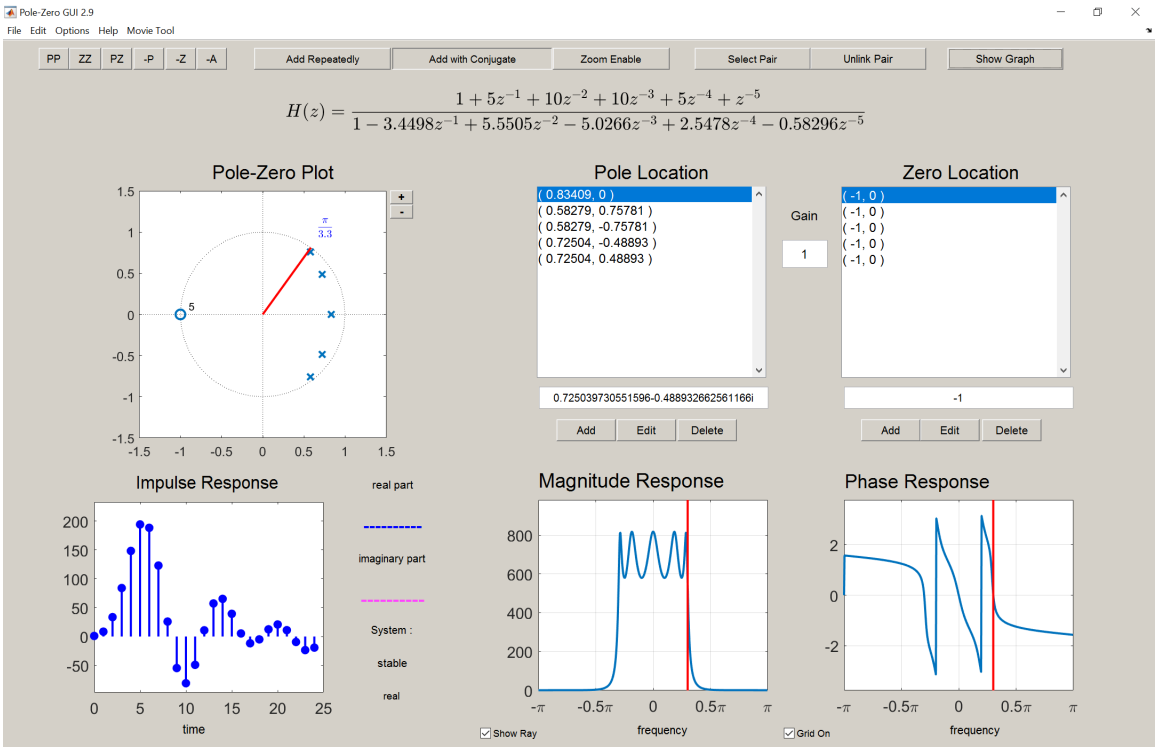

Figure 2.1: DSP-First pezdemo (a MATLAB GUI code), showing the 5th-order Chebyshev Low-Pass Filter (LPF), with the cursor at cutoff 0.3 (http://dspfirst.gatech.edu/matlab/).

2.1 Leveraging Different Student Learning Styles

At different times, engineering students may use one or more different types of learning styles (e.g. see Index of Learning Styles ILS [11], VARK [12], Kolb [15]). Having available a wider variety of teaching styles, and some integrated tools (such as the `PeZdemoR` web-app) that support a multi-modal approach will help increase student success rates.

For example, the Felder–Silverman Index of Learning Styles (ILS) model classifies students as fitting into one of the following learning-style dimensions [11]:

- sensing vs. intuitive learners,
- visual vs. verbal,
- inductive vs. deductive,
- active vs. reflective,
- sequential vs. global.

Felder also discusses some specific strategies for teaching to a range of learning styles, based upon the ILS model [11]. Similarly, as a specific case-in-point to this project, Kapadia [14] discusses *Teaching Filter Design to all Learning Styles*, concluding with the following (see Table 2.1):

> "The learning styles of many engineering students and the teaching styles of many Engineering professors do not always match. Many engineering students are **visual, inductive, and sequential** learners. Many Engineering professors and Engineering programs, on the other hand, are **auditory, deductive, and intuitive**."

Table 2.1: Mismatch of learning styles to teaching styles

Engineering Student Learning Styles	Engineering Teaching Styles
visual (pictures, diagrams, flow-charts)	auditory, verbal (spoken, written)
inductive (from specific, to the general)	deductive (from general, to the specific)
sequential (linear, orderly, small steps)	*[global (holistic, systems, leaps)]*
[sensing (facts, procedures, numbers)]	intuitive (concepts, theories, algebra)

This integrated `PeZdemoR` web-app covers many of the different learning styles, e.g. Visual, Audio, Read/Write, and Kinesthetic (VARK). In the VARK model, a kinesthetic (tactile, bodily, physical, real-world, direct involvement, learn by doing) teaching experience is defined as one in which [12]:

> "... all or any of these perceptual modes are used to connect the student to reality, either through experience, example, practice, or simulation."

2.2 Signals and Systems Concept Inventory

The Signals and Systems Concept Inventory (SSCI) [33] is a 25-question multiple-choice exam designed to assess the level of student understanding of core concepts taught in undergraduate linear signals and systems courses. The SSCI is reviewed against other concept inventories in [23]. In the Discrete-Time (DT) SSCI, four of the 25 questions relate directly to an understanding of pole–zero plots, and their resulting filter responses within the time or frequency domains. In a later analysis of the relative

difficulty of these questions, when compared to the other questions [6], the following was noted:

> "All three of the questions with difficulty above 1.5 ask about pole–zero diagrams. ... The most difficult question is number 18, which asks students to identify which pole–zero plots could have a real impulse response. Question 18 was previously identified as the most resistant to instruction."

2.3 Visualization

Visualization is the user-selective transformation or graphical representation of abstract data into a form that allows for interactive exploration, analysis, and understanding. In an educational context, visualization can be useful when teaching about a conceptual topic that is difficult to otherwise see. If this visualization can be modified in real-time, then it can provide simultaneous awareness of parallel patterns and structural relations within the same data. Within data-sets that are complicated, visualization can enable users both to better confirm the expected, and to discover any unexpected features. This is in contrast to the concept of presentation graphics, which is more concerned with the efficient one-way communication of final results, rather than exploring the data.

Displaying visualizations onto a large-screen format, or onto physically-distributed network displays, will allow multiple people to collaborate simultaneously on the same design in order to better explore the information together. In this way, the visualization environment becomes a virtual laboratory. This takes advantage of group skills for pattern recognition and intuition. Also, the sense of immediate gratification of achieving design goals on-screen can lend a competitive sense of gamification.

Having the data and structure available within one single interactive tool will avoid the need to repetitively alter, store, collate, and display every intermediate result. By changing parameters with the help of an integrated tool, the changes can be seen in near-realtime (using reactive dataflow programming). This live process with its tight feedback-loop will lead to the efficient use of design time.

2.4 Digital Signal Processing Domains

Digital Signal Processing (DSP) is the use of digital computers to execute signal-processing operations on digital signals. The signals are a sequence of numbers that represent samples of a continuous variable. The source of the continuous variable can include representations of audio (e.g. voice), sonar, radar, seismic vibrations, sensor-array signals, images, video, telecommunication signals, control signals, or any time-series [29], such as for medical, logistic or economic measurements.

Digital signals are studied primarily within the time domain (or spatial domain), and the frequency domain. Signals in either the time domain (or spatial domain) can be digitized directly from an electronic transducer. The Discrete Fourier-Transform (DFT), often implemented by an FFT (Fast Fourier-Transform) algorithm, can be used to convert these sample values into the corresponding frequency domain spectrum. This spectrum has a magnitude component and a phase component for each frequency. The spectrum shows which frequencies are more significant, and which are less significant.

In contrast, McClellan et al [18] consider the following three domains:

> "... the n-domain or time domain (... sequences, impulse responses, and difference equations), the $\widehat{\omega}$-domain or frequency domain (... frequency responses, and spectrum representations), and the z-domain (... z-transforms, operators, and poles and zeros)."

2.5 Digital Filters

A digital filter is a system that performs linear transformations on a sampled, discrete-time signal in order to reduce or enhance any desired aspects. A filter can be described by a block diagram, a difference equation, a collection of pole–zero values, an impulse response (or unit-step response), or a computer algorithm.

There are many ways of classifying digital filters, such as linear or non-linear, time-invariant or time-varying, causal or not-causal, infinite impulse-response (IIR) or finite impulse-response (FIR), stable or non-stable.

Chapter 3

Using a Web Application to Visualize Filter Design

A web application (web-app) is a client–server software application in which the client-side user-interface executes inside an internet web-browser. The server-side executes the R-language commands that generate the plots and other outputs that are to be displayed on the client-side.

From the server-side machine, the client-side document is initially downloaded into the (thin) client-side machine when first visiting the webpage. This document is written in a standard Document-Object Model (DOM) format, which is composed of HTML5, CSS3, and JavaScript. During the session, the web-browser interprets and displays the DOM document, changing as necessary according to the user input and any server-side updates.

3.1 R and Packages

R is a free and open source tool, primarily aimed at statisticians and data analysts, providing powerful statistical graphics for tasks such as: chemistry, physics, finance, clinical trials, medical imaging, psychometrics, machine learning, statistical methods, and engineering.

The popularity of R (e.g. versus MATLAB) is increasing, with approximately 2000 daily-downloads, averaged during recent weeks (Feb. 2017). There are currently

just over 10000 user-contributed R packages available on the CRAN Comprehensive R Archive Network package-repository (growing at dozens per week), with the top three most popular packages, `Rcpp`, `ggplot2`, `digest`, each averaging approximately () daily downloads. Many trend watchers, including IEEE Spectrum magazine [10], the TIOBE index [31], and RedMonk [22], now all rate the R-language higher in popularity ahead of MATLAB.

Some great cheatsheets for Base R, Advanced R, RStudio integrated development environment (IDE), `shiny`, and also R Markdown, are available at the RStudio website. Over 30 CRAN task-views allow one to browse the many packages by specialized topic. The task-views aimed at time-series, numerical mathematics, optimization, and spatial data are especially useful for engineering work.

For developing DSP demos (e.g. with processing using FFTs, DCTs, wavelets, sound, images, symbolic math), in addition to the packages used within this `PeZdemoR` demo, other R-packages that would be useful include `fftwtools`, `fftw`, `wavethresh`, `wmtsa`, `waveslim`, `wavelets`, `waved`, `dtt`, `seewave`, `monitoR`, `phontools`, `warbleR`, `ripa`, `imager`, `EBImage`, `magick`, `Ryacas`, and `rSymPy`.

3.1.1 The `shiny` Package

The **`shiny`** package is: [3] "a (free) contributed package to R that makes it incredibly easy to deliver interactive data summaries and queries to end users, through any web-browser. The `shiny` package comes with a variety of widgets for quickly building user-interfaces." The default styling of any `shiny`-generated application is based upon the default Bootstrap theme, but `shiny` is extensible, and it is easy to integrate a `shiny` application with any web content using HTML, CSS, and JavaScript libraries like jQuery.js (and many others).

3.2 Overview of the Proposed Web-app

The proposed approach uses a web-based application to graphically demonstrate the relationships between different visual aspects of pole–zero placements (z-domain), and the corresponding responses in the domains of time n (which represent multiples of

the sampling period, T_s), and frequency ω. Figure 3.1 shows the developed web-app prototype, called `PeZdemoR`, for an example 5th-order Chebyshev low-pass filter.

The web-app development framework includes the responsive `Bootstrap` front-end framework [24]. This includes: a 12-column wide grid reference system, some pre-made template layouts, colour themes, navigation bars, buttons, fonts, icons, and more, and it supports responsive web-design. Responsive web-design means the layout and elements of a webpage will adjust dynamically, while taking into account the characteristics of the device being used, e.g. desktop, tablet, or mobile phone. If the width is not sufficient for a landscape presentation, which is typical for mobile devices (less than 768 pixels wide in portrait orientation), then the graphic components are re-stacked vertically. `Bootstrap` comes with several JavaScript components in the form of `jQuery` plugins.

Using the open-sourced package `shiny` to generate and manage the client–server application for an engineering demo is new with regards to its ease of learning. It is easy in the sense of both learning how to develop this new web-app tool, and in learning targeted DSP concepts while using this tool. Typically, engineering demos are in the form of a dedicated standalone (e.g. MATLAB) desktop application. Being open source, this web-app is easily extensible to connect with more elaborate (free) JavaScript libraries like jQuery, d3, and Plotly (interactive graphics). The basic elements (e.g. buttons, sliders, radio controls, etc), available within the R `shiny` package provide an interaction paradigm that is familiar to most web users.

Use-case scenarios include instructor-led group demos, self-learning students (introduction to filters, exam preparation, drafting of assignments and labs), students using some other app to design the filter data that is imported into `PeZdemoR` (to prepare images for other documents), working engineers preparing a quick report before an executive briefing, or as an instrument for recording of minutes during a group design review or other team meeting.

There are over 100 examples of (editable) common filters, design algorithms, and FFT data-windows. This will be helpful both for novice DSP students and new users of the `PeZdemoR` tool. Different example configurations can be simultaneously opened and explored in different windows.

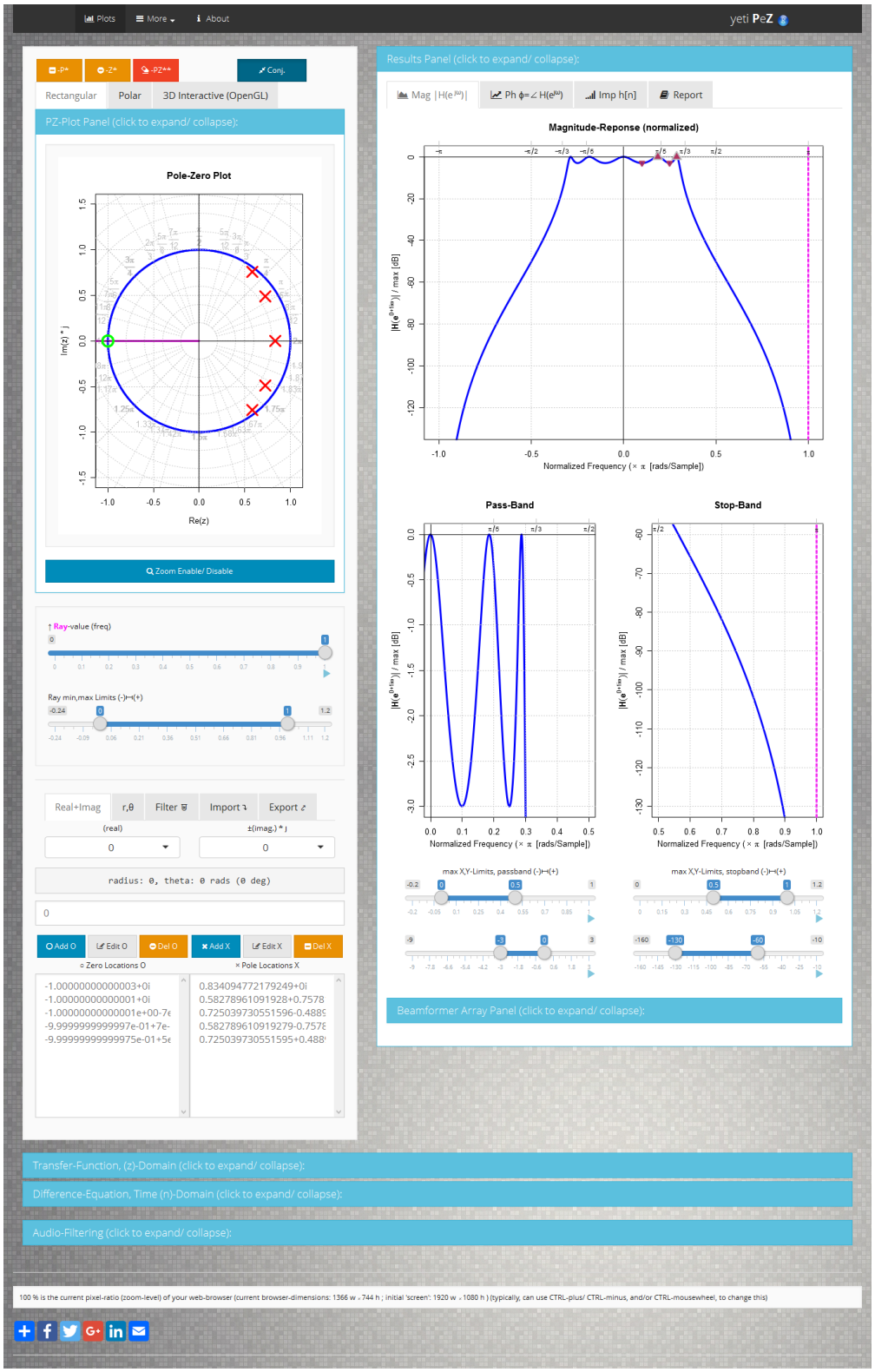

Figure 3.1: General visual layout of the PeZdemoR web application, showing an example 5th-order Chebyshev Low-Pass Filter.

Chapter 4

Prototype Development of a DSP Web-App in R Using Shiny

The design of the prototype web-app, called `PeZdemoR`, began with limited experience of general programming in R and MATLAB, moving through the online `shiny` tutorials, and then on to the excellent book by Beely, *Web Application Development with R Using Shiny* [3]. The barrier to entry for transition from user to tool developer is surprisingly less intimidating than it is for either a full-scale MATLAB GUI, or for an HTML, CSS, JavaScript web-project.

4.1 User Interface (`ui.R`) Design

Any client–server application is composed of two parts, the front-end visual layout and behaviour (look-and-feel) components, and the back-end server components. This two-part structure is also reflected in the software code structure. There are two main software components, `ui.R` (user-interface) and `server.r`. For editing convenience, these two files are often combined into just one file called `app.R`. Figure 4.1 shows the conceptual architecture between the three main components of the user-interface definition (`ui.R`), the server code (`server.R`), and the web-browser. Whenever a data value changes within any of the elements located inside the web-browser, a message is sent to the server. Each widget component within the server monitors for any changes to any of its data inputs, and will request recalculation only as necessary, and only if

it is being actively displayed. Any recalculated plots or text will be sent back to the client for updating inside the web-browser.

The user-interface includes `shiny` widgets (functions), and provides the overall visual layout. This is where the user receives and provides any information, and also where the user controls the structure and source or destination of the information. The server portion does all of the background computations and image rendering (see next section).

The advantage of this web application is the simultaneous display of the same system in different forms. Visual comparisons can be readily made between the pole–zero placement in the z-plane, the magnitude (and phase) response in the frequency domain, and the impulse response in the discrete time domain. Comparisons can also be made between the phase response and group delay, time-series and auto-correlation, geometric relationships and algebraic expressions, original signal and filtered signal, or frequency domain and spatial response (e.g. due to an antenna or speaker array).

The visual layout of `PeZdemoR` (see Figure 3.1) has three main areas: the left-hand sidebar (typically used for user inputs), the right-hand main panel (typically used for outputs), and the pair of lower equation panels. These three panel areas are collapsible, and each area makes extensive use of tab panels in order to limit the viewer focus, e.g. during an instructor-led walkthrough or quiz, or during a self-guided exploration. Tab panels within these main collapse panels control what is being viewed at any particular time. These panels and accordion-like collapses can be used for selective hiding of information.

Below the pair of equation boxes is the audio filtering section of the web application, providing for listening of playback (e.g. echos, treble-boost or cut, bass-boost or cut, equalizer bands, notch filter), and spectral visualizations. This uses the package `audio` [32] for Windows-based playback, and either `tuneR` [16] or `R.matlab` [4] for reading and writing audio files. Inputs can be .wav, .mp3, or .mat-file (MATLAB) audio signals (e.g. white-noise, pink-noise), or other recordings can be used (e.g. recorded from open-source audio applications like Audacity). Some built-in examples include Dual Tone Multi Frequency (DTMF) tones, birdsongs, a set of sinusoids, standard Donoho and Johnstone test signals (e.g. Heavisine, bumps, blocks, doppler, ramp, which are taken from the `rwt` package [26]), and general time-series examples from [29].

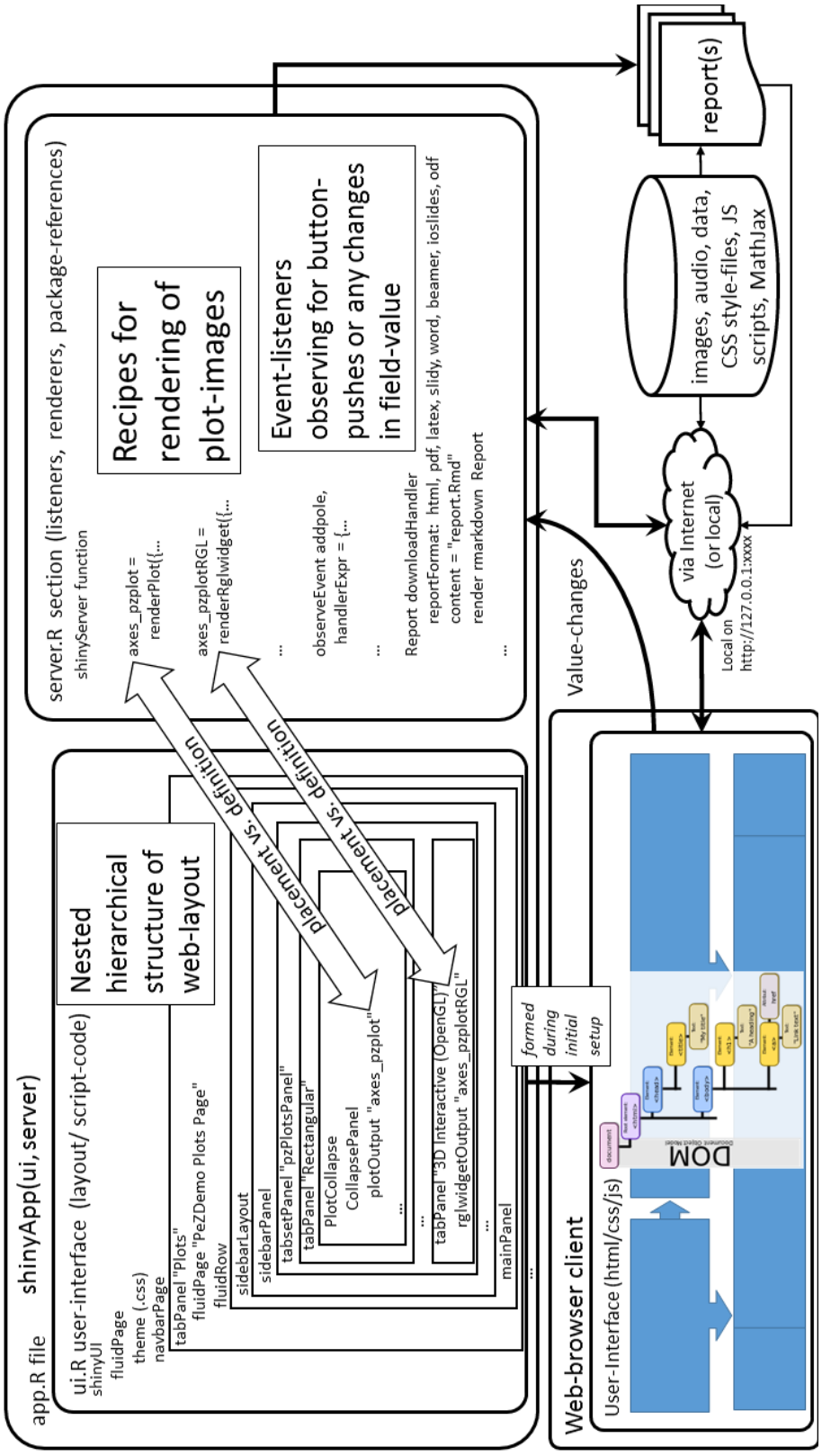

Figure 4.1: This system architecture diagram shows the conceptual separation between the three main software components including the user-interface definition (ui.R), the server code (server.R), and the web-browser.

4.2 `Plots` Tab Panel

This is the main tab of the overall navigation bar. Note that, throughout the remainder of this document, a 5th-order Chebyshev (type I) Low-Pass Filter (LPF) example will be used to illustrate plot capabilities. Also, a magenta-coloured ray-pointer (at 0.3 frequency cutoff) will appear on any plots within the app that has a frequency-axis (or, for the spatial beamformer plot, the ray-pointer represents the angle off-axis).

4.2.1 (Left) Side Panel

4.2.1.1 Pole–Zero Plots (z-plane)

The system transfer function is expressed as a ratio of two polynomials. The numerator has M roots (corresponding to the zeros of the transfer function $H(z)$), and the denominator has N roots (corresponding to the poles of $H(z)$). Figure 4.2 shows the location within the complex z-plane of the poles and zeros of the transfer function for the filter. For a root-locus plot in control theory applications, an s-plane background grid is also available, showing contours of constant ζ (damping factor) and constant ω_n (resonant frequency).

4.2.1.2 Three Dimensional (3D) Pole–Zero Plot

Figure 4.3 shows a three-dimensional (3D) transfer function plot with 4 degrees of freedom (pitch, yaw, roll, zoom-in/out) manipulation ability, which uses the package `rgl`[1]. It also produces a pop-under window for independent full-screen viewing. Similar to [7], on the use of three-dimensional (3D) plots of the transfer function:

> "The magnitude of the filter transfer function is a graph in a three-dimensional coordinate system. It defines a 'mountainous area', where poles correspond to 'peaks' and zeros to 'valleys'; in this area, we 'go on a hike along the unit-circle' – the closer that one is to a pole or zero, the more pronounced is the maximum or minimum of the magnitude response."

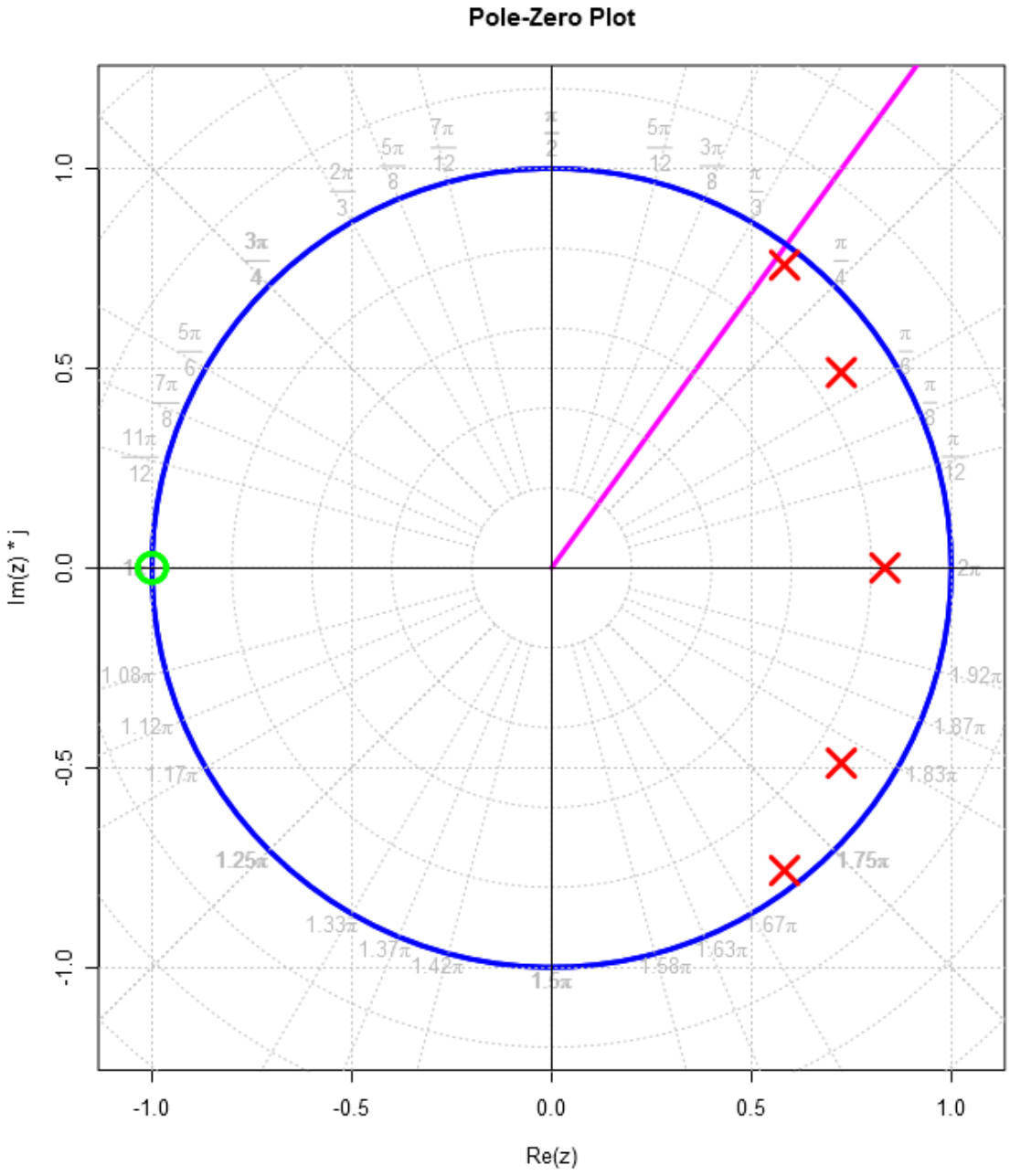

Figure 4.2: A pole–zero plot for the 5th-order Chebyshev Low-Pass Filter (LPF) with the cursor at cutoff 0.3.

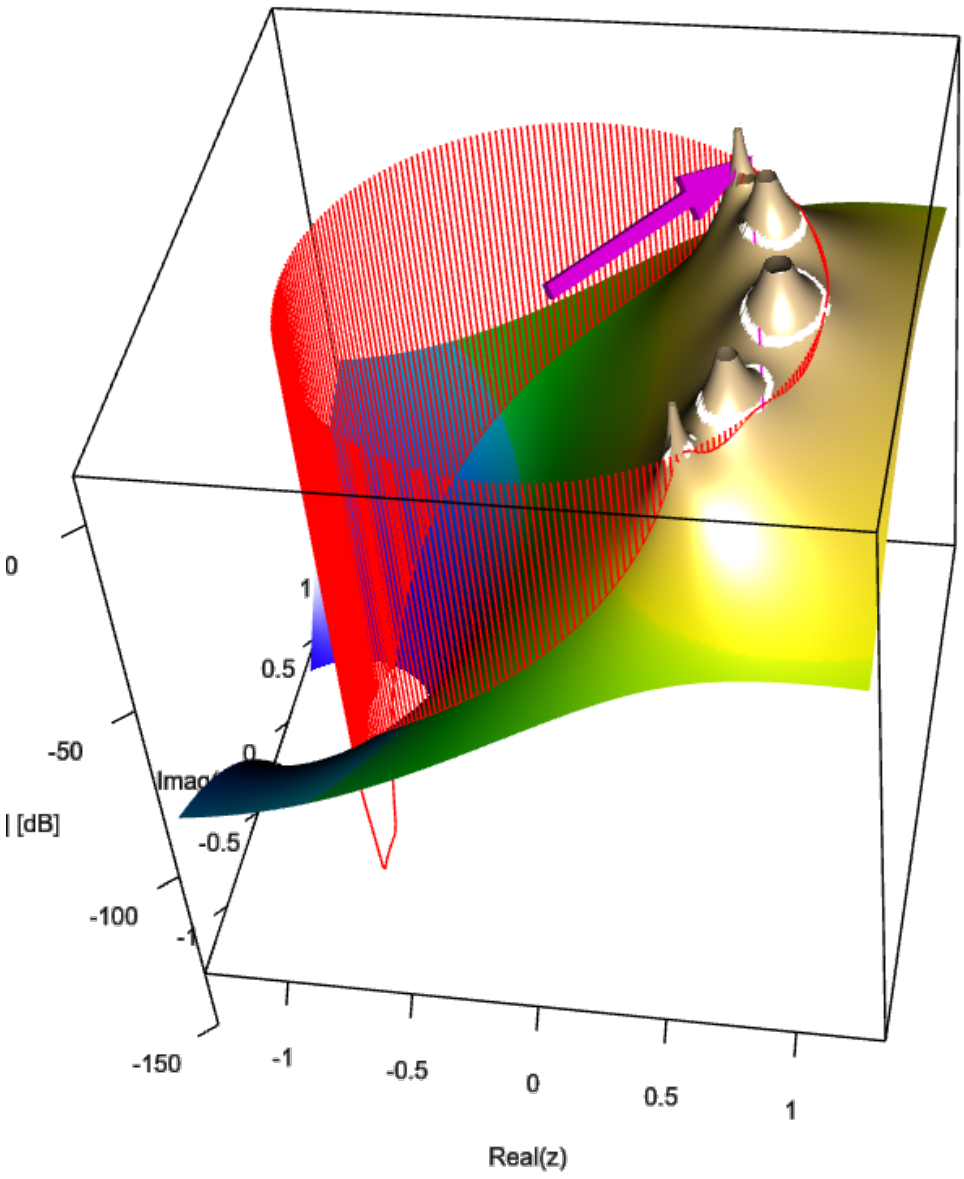

Figure 4.3: A 3D response plot for the 5th-order Chebyshev LPF with the cursor at cutoff 0.3. The online version can be manipulated using the mouse (or by touch-gestures on mobile devices).

4.2.1.3 Entry Panel for the Poles and Zeros

Figures 4.4 and 4.5 show two ways that the poles and zeros (or alternatively the two filter-coefficient vectors, `b` and `a`) can be entered into the `PeZdemoR` web-app. The first way is to enter the numerical values, one-by-one, directly into the combo pull-down with text entry boxes, using either rectangular or polar coordinates. The other way is to enter R-language commands that will generate the desired numerical values. Note that the command syntax of both R and the `signal` package is similar to that in the Signal Processing Toolbox of MATLAB.

4.2.1.3.1 Combo Pull-Down Fields – Rectangular or Polar Tabs

The primary method of entry for pole–zero values is the pull-down text entry boxes. The pole–zero locations can either be entered separately using real and imaginary values, or as radius (r) and angle (θ) values under the polar tab. One can also use complex exponential form or the Euler form using trigonometry functions, e.g. use either `exp(-2i*pi/3)` or `cos(pi/3)+1i*sin(pi/3)` in either of the two pull-down text entry boxes. When entering values or commands, one can use either `1i` or `1j` as the imaginary number value. When a value is selected within the pole–zero lists, it also appears in a text entry field, allowing for editing or duplicating of existing values.

4.2.1.3.2 Filter Command Entry Tab

Under the filter command tab, there is a pull-down text entry box of over 100 built-in filter examples (see Appendix). When a filter is selected, the corresponding filter commands appear within a text entry box below. Instead of entering the numerical values as above, this allows for editing the built-in filter model command, or for entering a series of R-language statements. The required filter values are generated by entering complete algorithms using R-language syntax (note, the code must fit onto one line, using semicolons as necessary, and finish with either a filter modelling command such as `Zpg(zero=, pole=, gain=)`, `Arma(b=, a=)` or `Ma(b=)`, or an R-language list construction command, `list(a=, b=)`).

The capability exists for importing text coordinates from other apps (e.g. from a MATLAB filter coefficients file, `.fcf` in ASCII) as parameters into a command (see also the next section, discussing file imports). For example, a MATLAB

`.fcf` file for the 5th-order Chebyshev LPF design contains the numerical values that could be copy-and-pasted as the following R-command inside the `PeZdemoR` command entry field, `Arma( b=0.0020201693976176572*c(1,5,10,10,5,1), a=c(1, -3.1623646477361964, 4.7607003645490282, -4.0527940829480622, 1.9343905258872045, -0.4152867390282089))`

Within any algorithm, as an option to entering numerical frequency values such as 0.3, one can instead use the ray-pointer (the magenta-coloured cursor) variable, such as in `theta=input$slider1`.

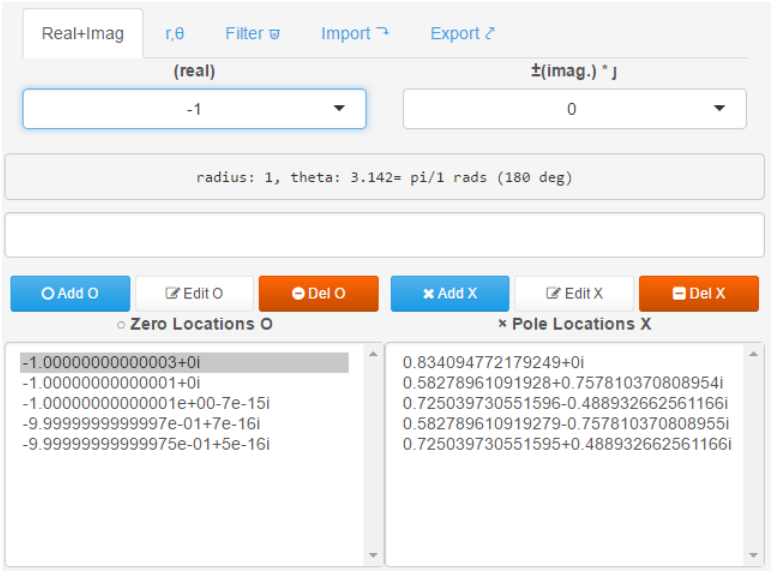

Figure 4.4: Entry panel for pole–zero numerical values.

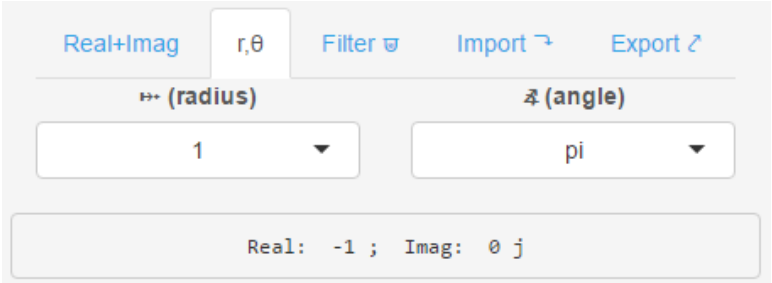

Figure 4.5: Entry panel for pole–zero values using polar coordinates.

4.2.1.4 File Import and Export Panel for Poles and Zeros

Figures 4.6 and 4.7 show the file import and export capability for the pole–zero values, including the formats CSV, binary (C-style binary), `.RData` (R-style, specific named-variables), MATLAB (`.mat`, specific named-variables), and YAML (a human-readable data-serialization language).

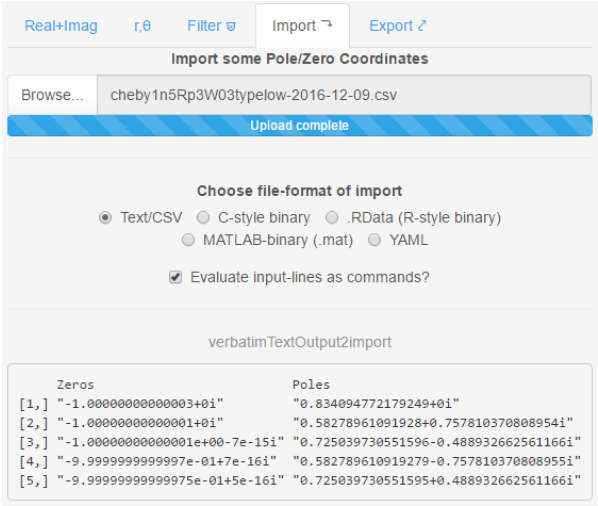

Figure 4.6: File import panel for pole–zero values. File formats available include .csv, .bin, .Rdata, .mat, and .yml.

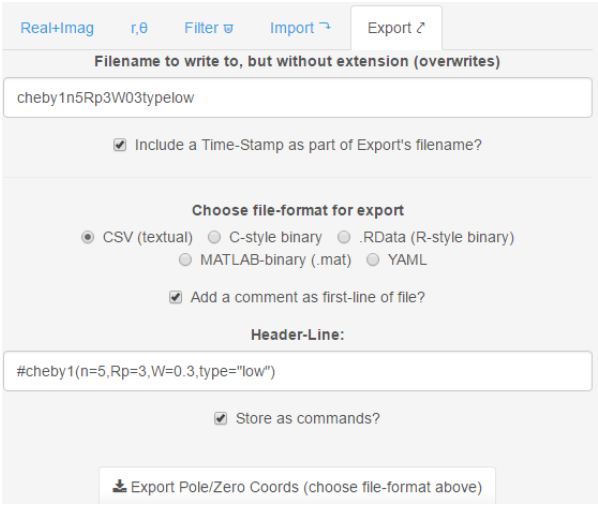

Figure 4.7: File export panel for pole–zero values. File formats available include .csv, .bin, .Rdata, .mat, and .yml.

4.2.2 Main (Right) Panel

4.2.2.1 Frequency Response (Magnitude) Plots

Figure 4.8 shows the frequency response (or gain function, G) of the system, which is the frequency-dependent absolute value of the ratio of the steady-state output amplitude to the input amplitude. Various features can be noted from viewing the frequency response plot: cutoff, roll-off, transition band, passband-ripple, and stopband-ripple.

The frequency response can be expressed in a number of different forms, each describing which frequency bands are passed (the passband), and which are rejected (the stopband), such as Low-Pass Filter (LPF), High-Pass Filter (HPF), Band-Pass Filter (BPF), Band-Stop Filter (BSF), notch filter, comb filter (it has multiple regularly-spaced narrow passbands or rejection-bands, giving the appearance of a comb-shape, with tines down), all-pass filter (all frequencies will pass unchanged in magnitude, but their phases are modified).

4.2.2.2 Phase Response and Group Delay Plots

Figure 4.9a shows the phase response, which is the relationship between the phase of a sinusoidal input, and the corresponding output signal. The input is used as a reference and is defined as having zero phase.

Figure 4.9b shows the group delay, which is calculated by differentiating (with respect to frequency) the phase response of the system. Thus, the group delay is a measure of the instantaneous slope of the phase response at any particular frequency.

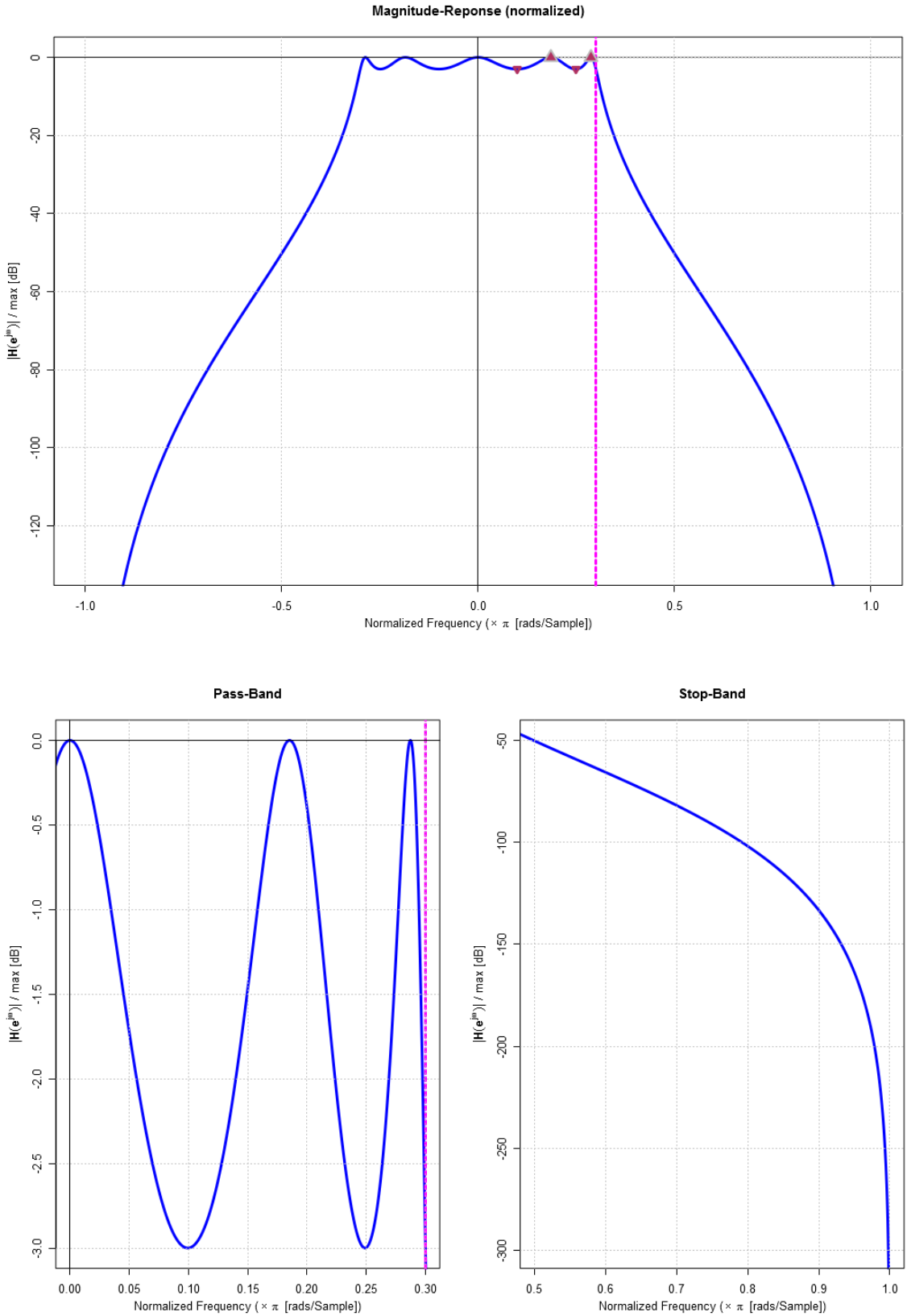

Figure 4.8: Magnitude response for the 5th-order Chebyshev LPF (normalized dB), showing the passband and stopband, with the cursor at cutoff 0.3.

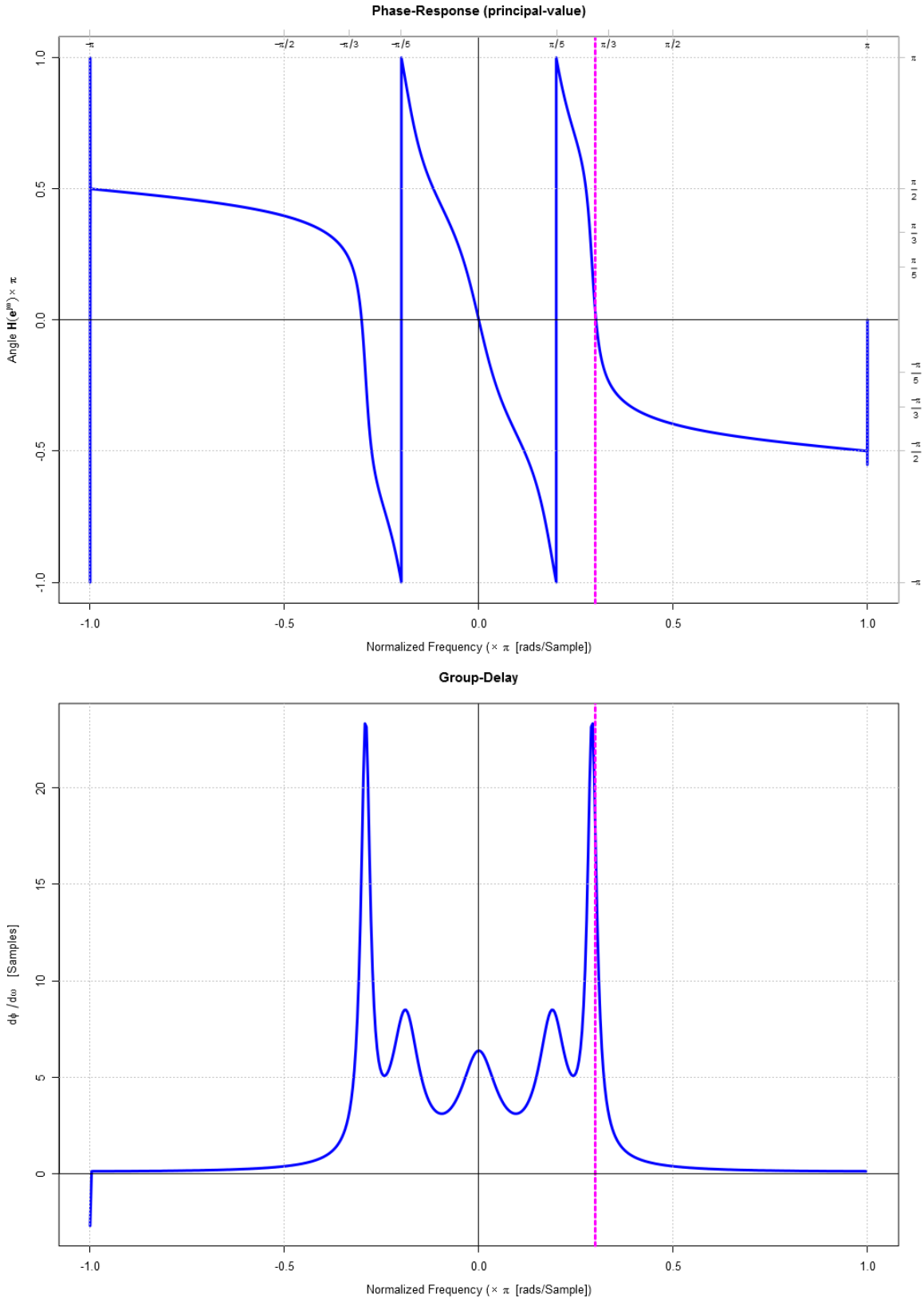

Figure 4.9: Phase response and group delay for the 5th-order Chebyshev LPF (normalized), with the cursor at cutoff 0.3.

4.2.2.3 Impulse Response and Unit-Step Response

Figure 4.10a shows the impulse response, $h_{[k]}$ $k = 0, 1, 2, \ldots$, which is the response to the impulse signal, $x = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix}$. Plotting this impulse response shows how the filter will respond to a sudden, short disturbance, and uniquely characterizes the filter behaviour. The integral equivalent to the impulse response is the unit-step response, as shown in Figure 4.10b).

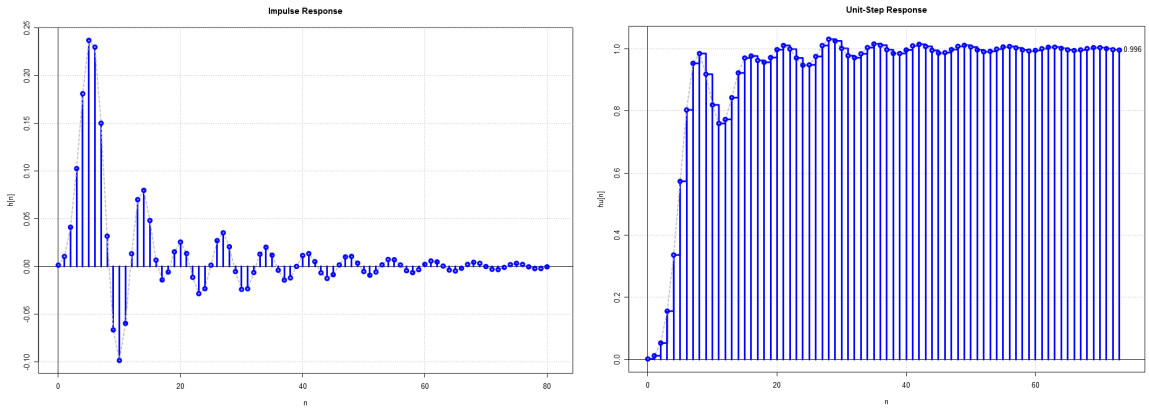

Figure 4.10: Impulse response and unit-step response for the 5th-order Chebyshev LPF (normalized), with the cursor at cutoff 0.3.

4.2.2.4 Other Plots – Spatial Filtering, Autocorrelation

4.2.2.4.1 Spatial Filtering (Array Beamformer)

The spatial filtering (array beamformer) plot shows the spatial-selectivity (directional resolution) and multipath interference avoidance (or sidelobe reduction) of an array of radiating or receiving elements, either while either transmitting or receiving.

4.2.2.4.2 Autocorrelation Function (ACF/PACF)

For the estimation of a Moving Average (MA) model, the Auto-Correlation Function (ACF) plot can be used to determine the appropriate number of lagged error terms that need to be included. Additionally, the Partial-ACF (PACF) can help determine the appropriate lags, p, in either an auto-regressive (recursive) AR(p) model, or in an extended ARIMA(p, d, q) model (e.g. as described in [29]).

4.2.2.5 Generation of Dynamic-Reports

Figure 4.11 shows the capability for complete RMarkdown-formatted [34] auto-report generation, with tight integration of code, figures, tables, equations (e.g. LaTeX/ MathJax), and prose, including inline R-command evaluation. This allows for traditional static-type reports in various formats (e.g. HTML, PDF, PDF-slides, Beamer, Word Doc, HTML-slides, Open-Document Format), dynamic (`shiny` based HTML) reports [34], with run-time editing within the replicated report output. The format of the report template uses the powerful `RMarkdown` language. As in most LaTeX-editors (but not so for word-processors like MS Word), the ability to selectively comment-out paragraphs will assist in document development and maintenance. There are many online articles and books available on the `RMarkdown` language (e.g. see [28]).

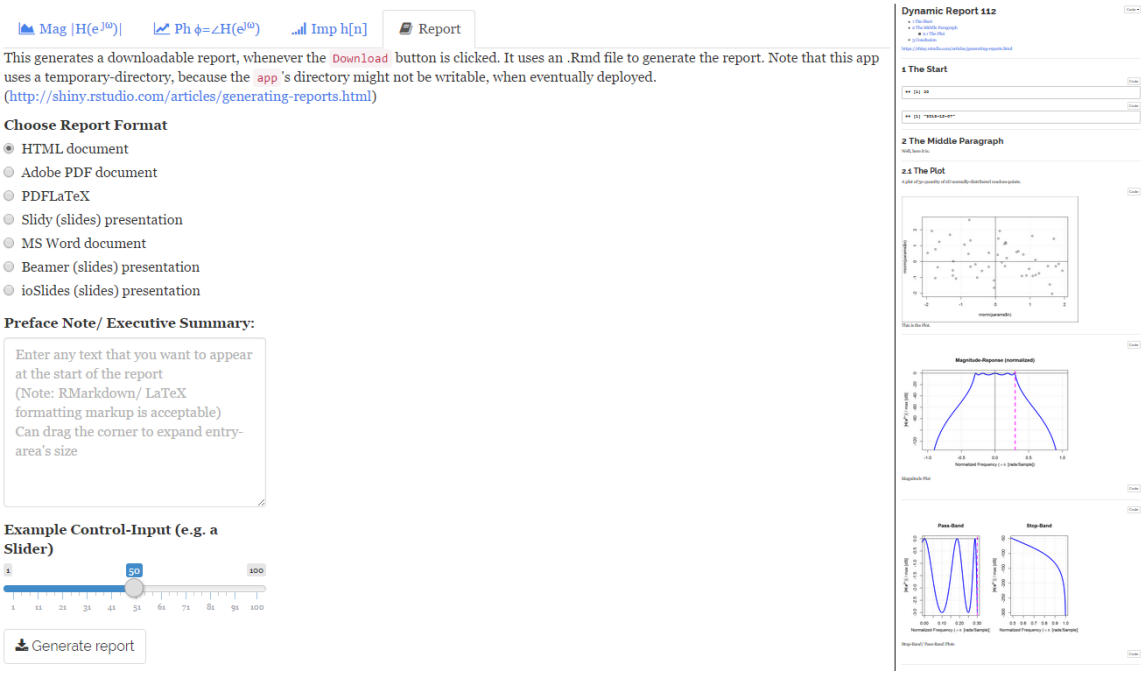

Figure 4.11: Report generator panel for creating dynamic-reports (e.g. in HTML format) from an RMarkdown template. This sample template within the web-app includes automatic section-numbering, a table of contents, code folding, and a full-precision listing of filter values. A portion of a sample HTML output for the 5th-order Chebyshev LPF design is also shown.

4.2.3 Transfer Function and Difference Equation Boxes

A digital filter is uniquely characterized by either a transfer function or a difference equation. The transfer function expresses the relationship between the system output and its input. When designing a filter, the objective is typically to develop a transfer function which meets the required specifications of order, cutoff, transition band, and ripple. The transfer function can be expressed in the z-domain as

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_{[0]} + b_{[1]} \cdot z^{-1} + b_{[2]} \cdot z^{-2} + \cdots + b_{[N]} \cdot z^{-N}}{1 + a_{[1]} \cdot z^{-1} + a_{[2]} \cdot z^{-2} + \cdots + a_{[M]} \cdot z^{-M}}$$

where the larger of N or M is called the order of the filter.

The web-app eases the tedious algebraic development by keeping track especially of the many plus and minus signs, and showing why some programming languages use syntax that is actually $-a_{[i]}$ versus $a_{[i]}$ (depending on which side of the equation it is on). It also shows why $a_{[0]} \equiv 1$ (identical), and $b_{[0]} = 1$ (normalized), and helps to clarify why written subscripts start at zero (0) versus some programming languages that, instead, start array subscripts at 1. For lower-order equations (order less than or equal to 2), the transfer function equation box will also show intermediate multiplications (i.e. $(z - z_{[1]}) \cdot (z - z_{[2]}) = z^2 - (z_{[1]} + z_{[2]}) \cdot z + (z_{[1]} \cdot z_{[2]})$, or similarly for $(z - p_{[1]}) \cdot (z - p_{[2]})$), along with the summing or cancellation of imaginary components.

Each term of the displayed transfer function or difference equation is built-up gradually, character-by-character, until the full LaTeX-string is completed, and then rendered in HTML using MathJax [9]. The use of MathJax allows for better interoperability by allowing for downloading of LaTeX or MathML code (e.g. MS Word compatible), for inclusion of the equations into other documents. In an online HTML environment, MathJax also provides a zoom feature, making it possible to view the formula at a larger size up to ±400% or more (e.g. for visually impaired users, or for large-screen viewing for groups).

4.2.3.1 Sample Transfer Function

Figure 4.12 shows the transfer function for the 5th-order Chebyshev LPF. The transfer function is the Laplace transform of the impulse response. Note that, within the

figure, underbraces and overbraces are used to better show the origins of the computed values, and indicates the existence of any equal terms or any conjugate matches.

4.2.3.2 Sample Difference Equation

For a digital filter, the transfer function can be converted into a difference equation using the z-transform. The transfer function is written as the ratio of two polynomials. In order to make the corresponding filter causal (using past values only, not future values), both the numerator and the denominator are divided by the order of the equation (highest power of z).

The coefficients of the denominator, $a_{[k]}$, are called feedback coefficients, and the coefficients of the numerator are called feedforward coefficients, $b_{[k]}$. The resultant linear difference equation is

$$y_{[n]} = -\sum_{k=1}^{M} a_{[k]} \cdot y[n-k] + \sum_{k=0}^{N} b_{[k]} \cdot x[n-k]$$

This equation shows how to compute the next output sample, $y_{[n]}$, in terms of the past outputs, $y[n-p]$, the present input, $x_{[n]}$, and the past inputs, $x[n-p]$.

A recurrence relation is an equation that recursively defines a sequence of values, which begin after some set of initial conditions. Each follow-on term of the sequence is then defined as a function of the preceding terms. The term difference equation here refers to a linear recurrence relation, with constant coefficients, as in Figure 4.13 for the 5th-order Chebyshev LPF.

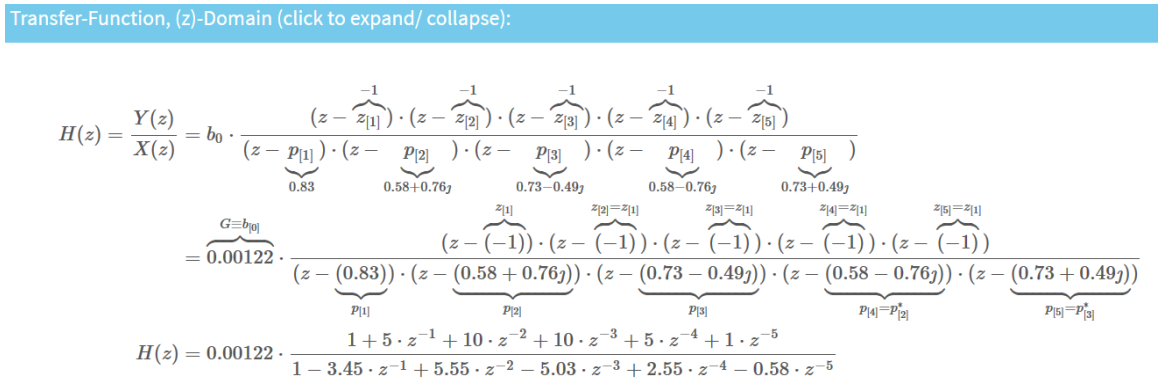

Figure 4.12: Transfer function panel for the 5th-order Chebyshev LPF, showing the algebraic development.

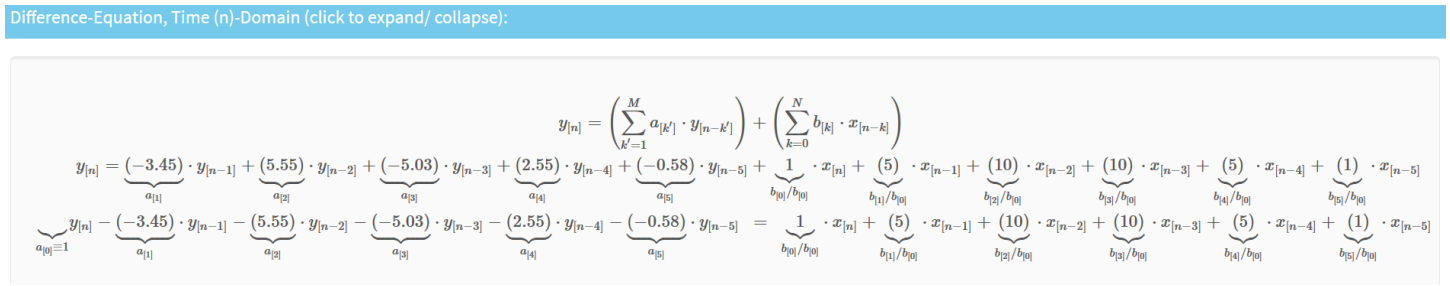

Figure 4.13: Difference equation panel for the 5th-order Chebyshev LPF, showing the algebraic development.

4.2.3.3 Filter Coefficients (b, a) View-Panel

Figure 4.14 shows a part of the difference equation collapse panel which includes a listing of the b and a coefficients, along with a gain-entry numeric field (which is the $b_{[0]}$ value). The remainder of the $b_{[i]}$ coefficients are divided by this value to normalize the first term in the numerator polynomial to unity.

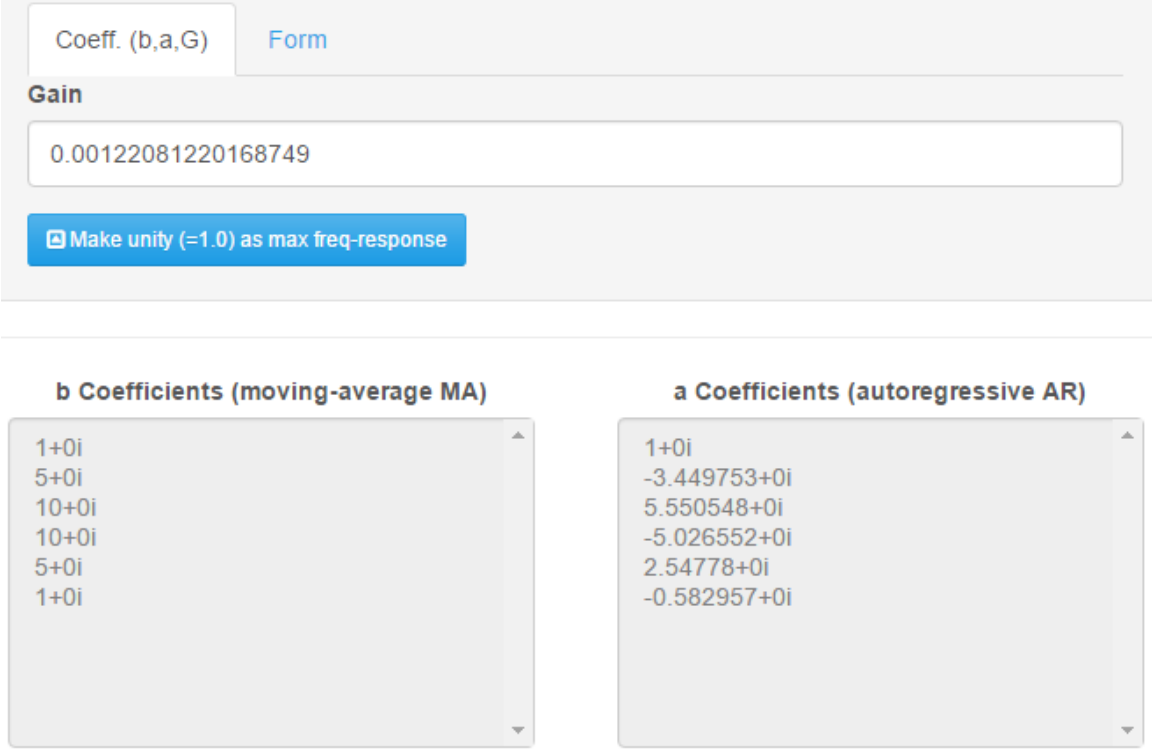

Figure 4.14: Filter coefficients viewing panel for the 5th-order Chebyshev LPF, showing the gain-adjustment field.

4.2.4 Audio Filtering Application

Figure 4.15 shows before (original) and after (filtered) versions of spectrograms, spectral plots and time-series plots. Figure 4.16 shows the panel for signal generation. Playback is possible on a Windows platform using the `wmm` Windows MultiMedia audio-driver from the `audio` [32] package.

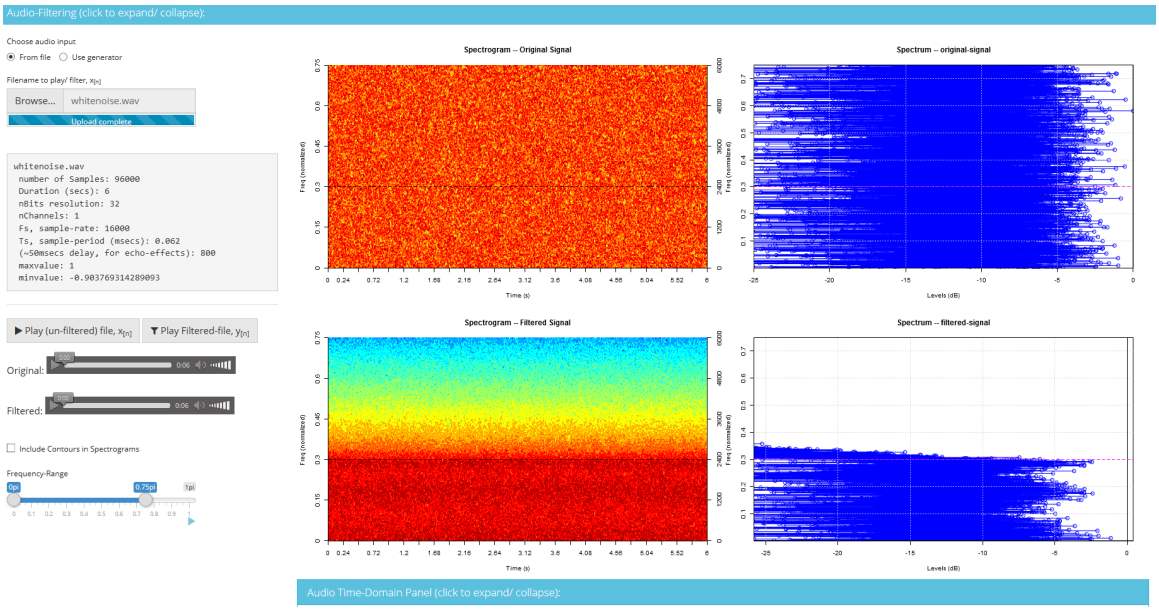

Figure 4.15: Audio filtering panel showing a white-noise signal being filtered by the 5th-order Chebyshev LPF, with the cursor at cutoff 0.3. PeZdemoR currently accepts .wav (e.g. made using Audacity http://www.audacityteam.org/, or some other), .mp3 music, and MATLAB .mat audio files (e.g. handel.mat, chirp.mat, furelise.mat, gong.mat, and others).

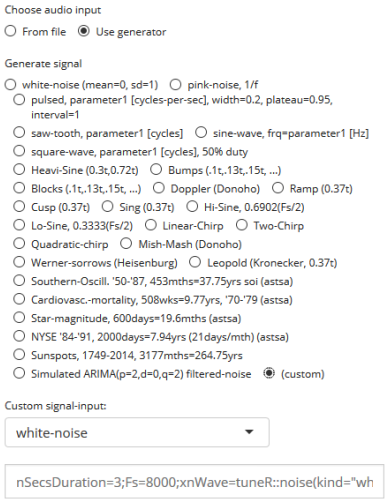

Figure 4.16: Audio generator control-panel.

4.3 Settings Page for Personalization (`Other` Tab)

Many settings can be customized to suit personal taste, both aesthetically and functionally, e.g. themes, colors, `semilogy` axes, and removing plot features. Using the settings page, these items can be changed temporarily (just for the current session) by selecting the appropriate control element, or changed within the source-code by altering the appropriate variables at the beginning of the `app.R` file.

The frequency axis (x-axis) upper limit can be changed to be related to the Nyquist sampling frequency (e.g. $0 \rightarrow 1$, $0 \rightarrow \pi$, $0 \rightarrow 0.5$, $0 \rightarrow \frac{f_s}{2}$, for one sided FFTs). There is an option for displaying two sided or one sided FFTs to cover the full rotation around the unit circle (to show negative frequencies). A logarithmic scale can be used for the frequency axis (x-axis), e.g. for Bode plots.

The amplitude plot of the magnitude response can be normalized to be unity, 1. As well, there is an option for plotting the amplitude of the magnitude response on a logarithmic scale (dB, or normalized to 0 dB full scale).

The default setting is to have a color theme randomly chosen at the start of every session. However, this can be fixed to a particular `Bootstrap` theme (or a custom CSS-file theme), by altering just a single line of the code.

Chapter 5

Conclusions

This report introduced the prototype web application `PeZdemoR` for visualization of filter responses when varying the placement of poles and zeros within the z-plane. The web-app allows for the design of small to medium-scale filters, and includes an audio portion demonstrating one-dimensional input processing by the designed filter.

The app was developed using `RStudio` with the R-language, along with the `shiny`, `signal`, and `pracma` packages. Both the written project report, and the associated slides for the oral examination, were composed within `RStudio` using the `RMarkdown` language.

The many ways of getting the pole–zero data (or filter coefficient data) into the web-app were detailed, including numerical input, R-coded algorithms, and pole–zero file importing (enhancing interoperability with other software applications). Over 100 built-in example filters from different engineering disciplines are provided with the web-app.

Output from the web-app is available in many forms, including image-downloading and file-exporting, and a sample dynamic report of the results can be generated in many forms from a customizable template. Both the resulting transfer function and the difference equation are computed and available onscreen for LaTeX-code copy-pasting.

The pedagogical application of the demonstration tool was discussed, including the general difficulties of teaching this DSP topic to undergraduate engineering students who use different learning styles. The visual appearance and functionality is customizable to suit personal taste.

5.1 Future Work

This proof-of-concept application could be expanded to be a part of new series called "interactive Signal-Processing in R for educators" (`inSPiRe`). For example, one could port the demos from the *DSP First* website, such as Complex-Spin, Spectrogram, Filter-Design, FourierSeries, PhasorRaces, SinDrill, ZDrill, CLTI, DLTI, Continuous–Discrete Sampling, Discrete-Convolution, Continuous-Convolution.

Like `fdatool` and the *DSP First* `pezdemo`, a capability could be developed for dragging, click-add, and click-delete for individual points on the pole–zero diagram. This may need integration with other JavaScript library components (e.g. jqPlot, jQuery).

The perils of using floating-point (finite-precision) representations (versus fixed-wordlength representations) could be investigated further, such as quantization effects, roundoff errors, or catastrophic loss-of-precision (loss-of-significance).

Multi-dimensionality could be added, allowing for filtering of 2D images or video.

More discrete (z-domain) to continuous (s-domain) linked-views could be developed, like in Hopkins `PZgui` Toolbox (MATLAB) for studying Single-Input Single-Output (SISO) transfer functions [13].

The app could include gamification for guessing filter responses, similar to the `zpgui` Magnitude-Response Learning Tool game [7].

The `R.matlab` package [4] and the MATLAB-server capability (included with typical MATLAB installations), could be used to integrate with existing MATLAB commands and Toolboxes, as suggested in [20]. In order to import existing MATLAB DSP code directly, a MATLAB to R translator capability could be added.

Appendix A

Built-In Examples

Table A.1: Some of the 100 built-in sample filters

Name	R-Code; Notes
FIR Window, order 40, LPF, 0.3, Hamming	`N=40; fir1( n=N, w=0.3, type="low", window=hamming(N+1) );` # `fir1` uses a least-squares approximation
FIR, arbitrary piecewise-linear (type II), order 100, BPF, 0.3 to 0.7, Hamming	`fir2( n=100, f=c(0,0.3,0.3,0.7,0.7,1), m=c(0,0,1,0.5,0,0), grid_n=512, ramp_n=5, window=hamming(101) );` # `fir2` uses freq-sampling
Arbitrary MA, FIR, given b	`Ma(b=c(1/3,2/3,1/3))`
Arbitrary ARMA, IIR, given b, a	`Arma( b=c(1/3,2/3,1/3), a=c(1,1-eps) );` # `eps`$= 1.110223 \times 10^{-16}$
Arbitrary ARMA, given zeros, poles, gain	`Zpg( zero=c(-1,-1), pole=c(-(1-eps)), gain=1/3)`
LPF, Bilinear z-Transform (function), conversion of analog, cutoff=0.4	`omegac=0.4; signal::bilinear( Sz=c(-fpmaxx/10), Sp=c(-tan(pi*omegac/2)), Sg=tan(pi*omegac/2)/(fpmaxx/10), T=2 )`
Transform band-edges s-plane LPF 0.3 to z-plane BPF 0.3-0.7	`omegac1=0.3; omegac2=0.7; signal::sftrans( Sz=c(-1e16), Sp=c(-tan(pi*omegac1/2)), Sg=tan(pi*omegac1/2)/(1e16), W=c(omegac1,omegac2), stop=FALSE)`
Chebyshev I, order 5, 3dB ripple, LPF, 0.3	`cheby1(n=5, Rp=3, W=0.3, type="low")`
Chebyshev I, order 5, 3dB ripple, HPF, 0.7	`cheby1(n=5, Rp=3, W=0.7, type="high")`
Chebyshev I, order 5, 3dB ripple, BPF, 0.3 to 0.7	`cheby1(n=5, Rp=3, W=c(0.3,0.7), type="pass")`
Chebyshev I, order 5, 3dB ripple, BSF, 0.3 to 0.7	`cheby1(n=5, Rp=3, W=c(0.3,0.7), type="stop")`
L-point Moving-Average, L=5, FIR	`L=5; FftFilter(rep(1/L, times=L), n=512)$b`
Freq-Sampling IIR algorithm, ord. 12, fc=0.7, delay=7	`N=12; D=7; fc=0.7; L=2*N; FF=matrix(data=0, nrow=N, ncol=1); for (k in seq(1, N, by=1)){ f=2*k/(L+1); if (f <= fc) { FF[k]=exp(-1i*D*f*pi) } }; Fb=Conj(FF); FF=c(1,FF,pracma::flipud(Fb)); h=Re(pracma::ifft(FF)); r1=t(h); r=c(r1[1],pracma::fliplr(as.matrix(t(r1[2:length(r1)])))); H=pracma::Toeplitz(h,r); H0=H[,1:(N+1)]; H1=H0[1:(N+1),]; h1=H0[seq((N+2),(L+1)),1]; H2=H0[seq((N+2),(L+1)),seq(2,(N+1))]; ah=-pracma::inv(H2) %*% h1; a=c(1,ah); b=H1 %*% a; list(a=a,b=b);` # W.-S.Lu
Butterworth, order 5, LPF, 0.3	`butter(n=5, W=0.3, type="low")`
Butterworth, order 5, HPF, 0.7	`butter(n=5, W=0.7, type="high")`
Butterworth, order 5, BPF, 0.3 to 0.7	`butter(n=5, W=c(0.3,0.7), type="pass")`
Butterworth, order 5, BSF, 0.3 to 0.7	`butter(n=5, W=c(0.3,0.7), type="stop")`
Chebyshev II, order 5, 3 dB ripple, LPF, 0.3	`cheby2(n=5, Rp=20, W=0.3, type="low")`
Chebyshev II, order 5, 3 dB ripple, HPF, 0.7	`cheby2(n=5, Rp=20, W=0.7, type="high")`
Chebyshev II, order 5, 3 dB ripple, BPF, 0.3 to 0.7	`cheby2(n=5, Rp=20, W=c(0.3,0.7), type="pass")`
Chebyshev II, order 5, 3 dB ripple, BSF, 0.3 to 0.7	`cheby2(n=5, Rp=20, W=c(0.3,0.7), type="stop")`

Name	R-Code; Notes
Elliptical, ord. 5, ripple:3dB(40dB stop), LPF, 0.3	`ellip(n=5, Rp=3, Rs=40, W=0.3, type="low")`
Elliptical, ord. 5, ripple:3dB(40dB stop), HPF, 0.7	`ellip(n=5, Rp=3, Rs=40, W=0.7, type="high")`
Elliptical, ord. 5, ripple:3dB(40dB stop), BPF, 0.3-0.7	`ellip(n=5, Rp=3, Rs=40, W=c(0.3,0.7), type="pass")`
Elliptical, ord. 5, ripple:3dB(40dB stop), BSF, 0.3-0.7	`ellip(n=5, Rp=3, Rs=40, W=c(0.3,0.7), type="stop")`
Min. Order Butterworth, Chebyshev I, Elliptical: ripple:0.5dB(29dB stop), LPF, 0.3	`butter(n=buttord(Wp=0.285, Ws=0.345, Rp=0.5, Rs=29))`, `cheby1(n=cheb1ord(Wp=0.3, Ws=0.34, Rp=0.5, Rs=29))`, `ellip(n=ellipord(Wp=0.3, Ws=0.34, Rp=0.5, Rs=29))`
Remez (Parks-McClellan optimal FIR), ord. 15, LPF, 0.3	`remez(n=15, f=c(0,0.3,0.4,1), a=c(1,1,0,0), ftype="bandpass")`
Bartlett, Blackman, vonHann(ing), Hamming, ... (41-pt)	`bartlett(41)`, `blackman(41)`, `hanning(41)`, `hamming(41)`, ...
All-Pass, poles inside circle, zeros outside at conj.-reciprocal	`r1=1.3; th1=0.6; Zpg( zero=c(r1*exp(th1*pi*1i), r1*exp(-th1*pi*1i)), pole=c((1/r1)*exp(th1*pi*1i),(1/r1)*exp(-th1*pi*1i)), gain=1)`
Band-Limited Differentiator, 23pt Hamming, fs=512, fc=0.3	`fs=512; Ts=1/fs; N1=23; M=(N1-1)/2; n=0:(M-1); k=M-n; k2=k^2; fc=0.3*pi; h1=sin(k*fc); h2=(fc*k)*cos(k*fc); hd=(h1-h2)/(Ts*pi*k2); hd=c(hd,0,-pracma::fliplr(as.matrix(t(hd)))); win=signal::hamming(N1); win*hd`; # W.-S.Lu
LPF, Bilinear z-Transform, conversion of analog-prototype, cutoff=0.4	`omegac=0.4; omegacprime=tan(omegac*pi/2); Zpg( zero=c(-1), pole=c(-(omegacprime-1)/(omegacprime+1)), gain=omegacprime/(omegacprime+1) )`; # http://slideplayer.com/slide/4173818/
FIR Window, order 10, BPF, 0.3 to 0.7, Hamming	`N=10; fir1( n=N, w=c(0.3,0.7), type="pass", window=hamming(N+1) )`
FIR Window, order 10, BSF, 0.3 to 0.7, Hamming	`N=10; fir1( n=N, w=c(0.3,0.7), type="stop", window=hamming(N+1) )`
Zero located at infinity, given b	`Ma(b=c(1,-fpmaxx))`; # `fpmaxx`= 1.797693×10^{308}
Over-damped: real Poles	`Zpg( zero=c(0), pole=c(-0.92345,0.92345), gain=1)`
Under-damp (damped-sin, exponen. envelope): conj. Poles	`Zpg( zero=c(0), pole=c(-0.5+0.52345i,-0.5-0.52345i), gain=1)`
Un-damped (osc., natural-freq, resonate): imag. Poles	`Zpg( zero=c(0), pole=c(0.92345i,-0.92345i), gain=1)`
Critical-damped: multi/ repeated real Poles	`Zpg( zero=c(0), pole=c(-0.92345,-0.92345), gain=1)`
Sinusoid/ Osc./ Resonator, poles on imag-axis, near-circle	`Arma( b=c(1), a=c(1,0,1-eps) )`
Sinusoid/ Osc./ Resonator, conj.-poles, arbitrary angle	`theta=1/6; Zpg( zero=c(0), pole=0.99999999999999*c(exp(theta*pi*1i), exp(-theta*pi*1i)), gain=1)`
Three-term (Delay-Line) 'IIR-equiv.' Mov-Avg filter, N=5	`N=5; Arma( b=c(1/N,rep(0,times=N-1),-1/N), a=c(1,-(1-eps)) )`
Cascaded Integrator-Comb/Hogenauer (MA filt.), R=5 (b,a)	`R=5; M=1; Arma( b=c(1, rep(0,times=R*M-1),-1), a=c(1,-(1-eps)) )`
Cascaded Integrator-Comb CIC/ Hogenauer, R=8 (Zpg)	`R=8; Zpg( zero=c(1, -1, 1i, -1i, 1/sqrt(2)+1/sqrt(2)*1i, 1/sqrt(2)-1/sqrt(2)*1i, -1/sqrt(2)+1/sqrt(2)*1i, -1/sqrt(2)-1/sqrt(2)*1i), pole=c(1-eps), gain=1/R)`
Comb-Filter, 5 poles w/3 zeros	`Arma( b=c(1,0,0,0.5^3), a=c(1,0,0,0,0,0.9^5) )`; # https://ccrma.stanford.edu/~jos/fp/Impulse_Response.html

Name	R-Code; Notes
Integrator/ Accumulator 1/s, given b, a	`Arma( b=c(1,1), a=c(1,-(1-eps)) )`
Notch-Filter, Fractional-Sample Delay-line, D=2pi/omega0, (Pei Tseng '98 Fig 2)	`omega0=0.22*pi; D=2*pi/omega0; rho=0.99; Arma(b=c(1,rep(0,times=floor(D-1)),-1), a=c(1,rep(0,times=floor(D-1)),-(rho)^D) )`
Peaking-Filter, fc=0.22	`theta=0.22; rp=0.999; rz=0.997; Zpg( zero=c(rz*exp(theta*pi*1i),rz*exp(-theta*pi*1i)), pole=c(rp*exp(theta*pi*1i),rp*exp(-theta*pi*1i)), gain=1)`
Notch-Out Filter, fc=0.22	`theta=0.22; rp=0.997; rz=0.999; Zpg( zero=c(rz*exp(theta*pi*1i),rz*exp(-theta*pi*1i)), pole=c(rp*exp(theta*pi*1i),rp*exp(-theta*pi*1i)), gain=1)`
Least-Squares IIR algorithm (unstable), ord. 12, fc=0.7, delay=7, 120 Samples	`N=12; D=7; L=120; fc=0.7; L1=0.5*L; FF=matrix(data=0, nrow=L1, ncol=1); for (k in seq(1, L1)){f=2*k/(L+1); if (f <= fc) { FF[k]=exp(-1i*D*f*pi) } }; Fb=Conj(FF); FF=c(1,FF,pracma::flipud(Fb)); h=Re(pracma::ifft(FF)); r1=t(h); r=c(r1[1],pracma::fliplr(as.matrix(t(r1[2:length(r1)])))); H=pracma::Toeplitz(h,r); H0=H[,1:(N+1)]; H1=H0[1:(N+1),]; h1=H0[seq((N+2),(L+1)),1]; H2=H0[seq((N+2),(L+1)),seq(2,(N+1))]; ah=-pracma::inv(t(H2) %*% H2) %*% t(H2) %*% h1; a=c(1,ah); b=H1 %*% a; list(a=a,b=b); # W.-S.Lu`
All-Pass, pole at 0, zero at infinity	`Arma( b=c(1,-fpmaxx), a=c(1) )`
All-Pass, reversed-ordering of (real) filter-coefficients	`myb=c(1,2,3,4,5,6); Arma(b=myb,a=rev(myb))`
Minimum-Phase, with GrpDelay< 2 (OSB, 1989, Fig 5.30a)	`Zpg( zero=c(0.9*exp(0.6*pi*1i), 0.9*exp(-0.6*pi*1i), 0.8*exp(0.8*pi*1i), 0.8*exp(-0.8*pi*1i)), pole=c(0), gain=1); # OSB 1989, Fig. 5.30a`
Hilbert (absolute values), order 40, Hamming	`N=41; M=20; hz=matrix(data=0, nrow=1, ncol=M); zw=seq(1,(M-1),by=2); hz[seq(1,(M-1),by=2)]=2/(pi*zw); hd=c(-pracma::fliplr(as.matrix(hz)),0,hz); w=signal::hamming(N); hd*w; # W.-S.Lu`
Ideal Differentiator (noiseless inputs only), 23pt Hamming, fs=512, fc=0.3	`t=seq(0,2-1/512,by=1/512); fs=512; Ts=1/fs; N=23; M=(N-1)/2; n=1:M; h=cos(n*pi)/(Ts*n); h=c(-pracma::fliplr(as.matrix(t(h))),0,h); win=signal::hamming(N); win*h; # W.-S.Lu`
Savitzky-Golay smoothing, order 3 (cubic), length 5	`p=3; n=5; sgolay(p,n) %*% c(1, rep(0,times=n-1))`
Echo/ Slapback-effects (delay-line comb)	`N=450; Arma( b=c(1/N,rep(0,times=N-1), -1/N), a=c(1) )`
Notch-Out Filter, fc=ray-slider	`theta=input$slider1; rp=0.997; rz=0.999; Zpg( zero=c(rz*exp(theta*pi*1i),rz*exp(-theta*pi*1i)), pole=c(rp*exp(theta*pi*1i),rp*exp(-theta*pi*1i)), gain=1 )`

References

[1] D. Adler, D. Murdoch, et al. *'rgl': 3D Visualization Using OpenGL*, 2016. R package version 0.96.0.

[2] J.J. Allaire, J. Cheng, Y. Xie, J. McPherson, et al. *'rmarkdown': Dynamic Documents for R*, 2016. R package version 1.0.

[3] C. Beeley. *Web Application Development with R Using 'shiny' – Second Edition.* Packt Publishing, Limited, Birmingham, 2nd edition, 2016.

[4] H. Bengtsson. *'R.matlab': Read and Write .mat-type Files and Call MATLAB from within R*, 2016. R package version 3.6.0.

[5] H.W. Borchers. *'pracma': Practical Numerical Math Functions*, 2016. R package version 1.9.3.

[6] J.R. Buck, K.E. Wage, and M.A. Hjalmarson. Item response analysis of the continuous-time signals and systems concept inventory. In *IEEE Digital Signal Processing Workshop and IEEE Signal Processing Education Workshop*, pages 726–730, Jan. 2009.

[7] P. Casapicola, M. Polzl, and B.C. Geiger. A MATLAB-based magnitude response game for DSP education. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2214–2218, May 2014.

[8] W. Chang, J. Cheng, J.J. Allaire, Y. Xie, and J. McPherson. *'shiny': Web Application Framework for R*, 2016. R package version 0.13.2.

[9] MathJax Consortium. MathJax – a JavaScript display engine for mathematics, Oct. 2016. website: https://www.mathjax.org/.

[10] N. Diakopoulos and S. Cass. Interactive: The top programming languages 2016, July 2016. website: http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages.

[11] R. Felder. Matters of style. *ASEE Prism*, 6(4):18–23, 1996.

[12] N.D. Fleming and C. Mills. Not another inventory, rather a catalyst for reflection. 1992. website: http://vark-learn.com/introduction-to-vark/articles/.

[13] M. Hopkins. Relating continuous time and discrete time in the classroom. In *2008 Annual Conference and Exposition*, Pittsburgh, Pennsylvania, June 2008. ASEE Conferences.

[14] R.J. Kapadia. Teaching and learning styles in engineering education. pages T4B–1–T4B–4. IEEE, 2008.

[15] D.A. Kolb. *Experiential learning: experience as the source of learning and development*. Prentice-Hall, N.J., USA, 1984.

[16] U. Ligges, S. Krey, O. Mersmann, and S. Schnackenberg. *'tuneR': Analysis of music*, 2014. website: https://cran.r-project.org/package=tuneR.

[17] J. MacFarlane. 'pandoc': A universal document converter, Oct. 2016. website: http://pandoc.org.

[18] J.H. McClellan, R.W. Schafer, and M.A. Yoder. *Signal Processing First*. Pearson Education International, New Jersey, 2003.

[19] J.H. McClellen and M.A. Yoder. *DSP First: A Multimedia Approach*. 1997.

[20] M. Mejia. Three ways to use MATLAB from R, Aug. 2014. website: https://mandymejia.wordpress.com/2014/08/18/three-ways-to-use-matlab-from-r/.

[21] O. Mersmann, U. Ligges, and S. Krey. *'signal': Signal processing*, 2014.

[22] S. O'Grady. The RedMonk programming language rankings, June 2016. website: http://redmonk.com/sogrady/2016/07/20/language-rankings-6-16/.

[23] T. Ogunfunmi, G.L. Herman, and M. Rahman. On the use of concept inventories for circuits and systems courses. *IEEE Circuits and Systems Magazine*, 14(3):12–26, thirdquarter 2014.

[24] M. Otto, J. Thornton, et al. Bootstrap: An HTML, CSS, and JS framework for developing responsive, mobile-first projects on the web, July 2016. website: http://getbootstrap.com.

[25] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016.

[26] P. Roebuck and Rice University's DSP group. *'rwt': Rice Wavelet Toolbox wrapper*, 2014. R package version 1.0.0.

[27] RStudio Team. *RStudio: Integrated Development Environment for R*. RStudio, Inc., Boston, MA, 2015.

[28] C. Shalizi. Using RMarkdown for class reports, Aug. 2016. website: http://www.stat.cmu.edu/ cshalizi/rmarkdown/.

[29] R.H. Shumway and D.S. Stoffer. *Time series analysis and its applications: with R examples*. Springer, New York, 3rd edition, 2011.

[30] B.L. Sturm and J.D. Gibson. Signals and systems using MATLAB (SSUM): an integrated suite of applications for exploring and teaching media signal processing. pages F2E–21. IEEE, 2005.

[31] TIOBE Software. TIOBE index, Dec. 2016. website: http://www.tiobe.com/tiobe-index/.

[32] S. Urbanek. *'audio': Audio Interface for R*, 2013. R package version 0.1-5.

[33] K.E. Wage, J.R. Buck, C.H.G. Wright, and T.B. Welch. The signals and systems concept inventory. *IEEE Transactions on Education*, 48(3):448–461, Aug. 2005.

[34] Y. Xie. *Dynamic Documents with R and 'knitr'*. Chapman and Hall/CRC, Boca Raton, FL, 2nd edition, 2015. ISBN 978-1498716963.

[35] Y Xie. *'knitr': A General-Purpose Package for Dynamic Report Generation in R*, 2016. R package version 1.14.