EFFICIENTLY-VARIABLE NON-OVERSAMPLED ALGORITHMS IN VIRTUAL-ANALOG MUSIC SYNTHESIS

—

A ROOT-LOCUS PERSPECTIVE

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Timothy Scott Stilson
June 2006

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Julius O. Smith III) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Stephen P. Boyd)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Chris Chafe, Music)

Approved for the University Committee on Graduate Studies.

Abstract

This thesis summarizes several contributions to the areas of musical signal processing and virtual-analog musical sound synthesis. In virtual analog music synthesis, the waveform oscillators and the audio filters are usually required to be smoothly modulated, often to audio rates. As such, efficient synthesis requires the design of efficiently variable algorithms for these synthesis blocks.

The design of efficiently variable filters requires the understanding of how they vary with their parameters. The most popular virtual-analog filters are of a class (pole-variable filters) which vary primarily via the motion of their poles. The Root Locus Method is proposed as a framework for understanding the variation of such filters, and several filters are analyzed as a demonstration, including popular analog filter types (the State-Variable Filter and the four-pole lowpass filter designed by Robert Moog). Root locus analysis also suggests directions for designing filters, or may be used directly in designs. This informs a series of experiments in creating digital filters in the style of Moog's lowpass filter, culminating in a few new designs.

The design of efficiently variable waveform generators is approached with the goal of reducing aliasing artifacts in the generated waves. Popular waveforms in virtual analog synthesis are rich in harmonics, and their analog counterparts have extremely wide bandwidths, such that care must be taken when generating digital versions. A methodology is proposed whereby a useful subset of the desired waveforms can be derived from a Bandlimited Impulse Train (BLIT) via linear operations, which will preserve any bandlimiting which may exist in the BLIT. Further, methods for implementing BLITs are explored and contrasted, including a method introduced by the author.

Finally, methods for rendering visualizations of root loci are reviewed; as such visualizations are necessary in the use of the method as an analysis tool. A recent advance in implicit-function rendering was found to be of particular use, and its application to locus rendering allowed the creation of programs for interactive exploration of root loci and their behaviors, which assist in the goal of building user intuition into variable-filter behavior.

Preface

I am an intuitionist. I have a strong belief in the power of intuitive understanding of systems and of the usefulness therefore of methods for building intuition. I am also a visual thinker. As such, I have a belief in the visualization of systems to help explain their behavior and build intuition about it. I am also, not surprisingly, an enthusiast of mathematical and computer-generated art, to the point that it was a toss-up as to whether I would pursue computer graphics or audio signal processing in graduate school (I pursued computer architecture). Upon having to decide for post-masters work, I chose musical signal processing over graphics, mainly on the fact that it was a smaller world, and hence would probably have more opportunities for doing something new. Still, I continued (and continue) to pursue advanced hobby-level exploration of graphics, visualization of mathematical systems and the aesthetics of complexity.[1] As such, even my research in audio was subtly guided by the ability to produce "neat pictures" as by-products of the research.

Therefore, with the realization that certain filters in the new field of virtual analog music synthesis could be interpreted in terms of root-locus, whose visual aesthetics I had always admired anyway, I had a direction I could follow which would satisfy both of my interests.[2] Visualization of filter behavior (and of root-locus behavior in general) can be directly used for building intuition into the complex behaviors of these systems. At the same time, adding experimentation into the loop can often result in fortuitous leaps of understanding. This couples with a nearly unspoken basic philosophy of much work done at CCRMA: exact models of a system need not be the final answer if a cheaper and effectively indistinguishable approximation can be found, which leads to a constant contemplation of the design tradeoffs in search of elegant yet efficient models of audio systems. As such, these shaped the basic philosophy of this thesis:

- Build intuition on the problem. Visualize the system, visualize its behaviors. Hence the root locus approach.

[1] See `http://ccrma.stanford.edu/~stilti/images/chaotic_attractors/nav.html` for some examples.

[2] Hence an almost equal interest to be found in this thesis on the graphical issues of the root locus as with the signal-processing issues of the filters and oscillators.

- Understand first, then imitate, or even extend. This is applied in this thesis by treating certain analog systems as "platonic types", rather then specific circuits to be modeled. By understanding the basic concept of the system, we may later tackle deeper implementations, though such deeper work may end up being done outside the confines of this thesis.

- Inexpensive implementations are of primary importance. Hacks are good if they work. Don't model something if it isn't going to be perceived.

- If there is time, make nice pictures while you're at it.

This thesis is organized as follows:

Chapter 1 first reviews the history and concepts of electronic and digital music synthesis, in particular virtual analog, then reviews digital audio filtering and discusses definitions of certain filter properties. Next, overviews of variable-digital filter design and musical filter design are given.

Chapter 2 explores the use of Root Locus as a method for analyzing the variation of pole-variable filters with their control coefficients. Special attention is paid to the primary filters of Virtual Analog: the state-variable filter and the four-pole lowpass in the style of Moog's voltage-controlled-filter.

Chapter 3 tackles design problems in virtual-analog filters, keeping a root-locus perspective in the analysis of the design issues and in suggesting design directions. The state-variable filter is revisited, and one or two extensions and variations explored, along with theories as to alternate ways that the form could be derived. Discrete-time implementations of the Moog-style filter are then explored, first via the standard discretizations, then via further exploration purely in the discrete-time domain, culminating in two particularly useful design concepts.

Chapter 4 discusses the implementation of waveform generators for virtual analog, with emphasis on reducing or eliminating aliasing, while keeping efficient modulatability. Existing methods are explored, and the concept of deriving desired waveforms from a single easily-calculated bandlimited source (a "bandlimited impulse train" or BLIT) is discussed. Then, methods for generating BLITs are reviewed, and a particularly modulatable method based on sample-rate-conversion techniques is described, together with related variations.

Appendix A starts with a review of Root Locus, its literature and rules, then expands to explore the concept of higher-order root locus, which is encountered extensively in variable-filter analysis, and some of its more basic properties are discussed. Later, methods for drawing root loci are classified and described, including a particularly elegant and efficient method based on an implicit-function rendering technique by Gabriel Taubin [260], which has proved invaluable in this research.

Appendix B explores how root locus can be used to provide insight into linear mode coupling, and particularly to the coupling which produces the two-stage decay of piano tones. Several of the analysis results of Gabriel Weinreich are replicated from a root-locus perspective, and the analysis is particularly applied to note how coupling varies from harmonic to harmonic of the vibrating

strings.

Appendix C discusses some exploratory work into applying simple control systems in music synthesis to achieve various effects.

Appendix D describes a research result involving local modulation of cosine-type tuning parameters via terms of the series approximation.

Appendix E presents some musings on the relation between the direct-form filter forms and allpass-filter implementations, including discussion of allpass filters in the DF-I style and some of their numerical advantages.

Appendix F presents a method for fitting graphical and parametric EQ's to filter curves.

Acknowledgements

I would like to thank my advisor, Professor Julius Smith, for providing the initial suggestions which lead to the core of this research, during an evening party at the 1995 ICMC in Banff, and for not giving up on the chance that I might actually finish this thesis after going off to start a company only two years into my thesis work (a black-hole-like situation from which few Ph.D. students can escape once pulled in).

I would like to thank the National Science Foundation, which supported me during the first three years of my Masters and initial Ph.D. work.

I would like to thank CCRMA, for providing the atmosphere of artistic and technical interest without which this type of research would be much more difficult, and for supporting me for a few years. Additionally, I would like to thank the people at CCRMA, especially Chris Chafe, John Chowning and Max Matthews, and the other students (and former students) who were at CCRMA during the early days of my research (when I spent most of my time there), such as Perry Cook, Scott Levine, Dave Berners, Gary Scavone, Stephen Bilbao, and in particular, Bill Putnam, with whom I had many fun discussions on many diverse topics, and who disappeared into his own black hole (called Universal Audio), soon after I did. I would also like to acknowledge Jonathan Abel, another former CCRMA student who, though he was at his own startups during my early time at CCRMA, became an encouraging force while I was finishing the thesis. And finally I would like to acknowledge Aaron Masters, who finishes his EE Ph.D. thesis at CCRMA the same quarter as I do, and thus has been a useful sounding board and co-commisserator about the travails of finishing up a thesis.

I would like to thank the other members of the group that started out as the Sondius project within CCRMA, and spun out to form Staccato Systems, which has supported me the rest of the years this thesis has been cooking: Julius, David Jaffe, Scott VanDuyne, Pat Scandalis, Nick Porcaro, and Joe Koepnick.

I would like to thank Analog Devices, for acquiring Staccato Systems and providing a level of stability through the rocky years after the "dot-com crash," and allowing me to continue working in audio DSP.

I would like to thank those who I have worked with at Staccato Systems and ADI, and who

helped make it a more enjoyable experience, in particular Sean Costello, Wayne Jackson and Denis Labrecque, whose interest and excitement about sound synthesis often exceeded my own.

Finally I would like to acknowledge my family for continually asking how the writing was coming (for over ten years), and in particular my father Walter E. Stilson and grandfather Walter L. Stilson, neither of whom, unfortunately, survived to see it actually completed.

Contents

List of Tables

List of Figures

Chapter 1

Musical Sound Synthesis and Variable Filters

1.1 Virtual-Analog Music Synthesis

1.1.1 History

Much of this thesis deals with design problems which arise in the computer implementation of certain electronic musical sound synthesis techniques. The history of electronic music is a very interesting story which is, unfortunately, not the topic of this thesis. Joel Chadabe gives a good history[1] in his book *Electric Sound* [36], and we will summarize a few points to explain the desire to re-implement such techniques in the present day.[2]

Early electric musical instruments were not well known to the public in general, except for use here and there for special effects, usually to create "futuristic" sounds for science fiction shows in radio, television, and movies, or to provide a sense of futurism in ads and commercials. Notable examples are the work of the BBC Radiophonic Workshop, most notably in their work on the long-running "Dr Who" television program, and the work of Raymond Scott [43], who produced many of the now-cliché "futuristic" sounds for ads and worlds-fair exhibits through the 1950s and 1960s. There was also some use of modern technology in academic music, and some of the most well-known early synthesizers, such as the RCA Mark II synthesizer [194] at the Columbia-Princeton Electronic Music Center, were used in such endeavors.

However, electronic music synthesis did not have a large impact on music in general until the album *Switched-On Bach*, realized using the then-revolutionary capabilities of voltage control in the

[1] though strongly weighted to the academic-music side of the history

[2] This section is influenced by many discussions between the author and various people, by numerous television documentaries on the history of recent popular music, and by similar books, such as [204], [36], [55], [232], [206].

Moog synthesizers [172][173], which demonstrated that electronic techniques could be used for more than sound effects or (to the public) unlistenable experimental academic music. Although the ensuing flood of copycat albums did not contain many works up to the same level, the use of electronic sound synthesizer did finally become mainstream. It was not in classical music that it would really take hold, though: it was in the field of popular music where synthesis would become most widely used.

Synthesizers were already being experimented with by the likes of The Beatles before *Switched-On Bach*, and synthesis was already becoming a common sound-effect in psychedelia, but it was in Progressive Rock and Funk that synthesizers became central instruments. The Funk genre in particular first noted the usefulness of a synthesizer as a bass instrument and a rhythm instrument, as opposed to a mostly treble/melody/special-effect instrument, as it was used in the likes of Yes, ELP, Pink Floyd and other Prog-Rock bands. It was essentially through Funk that synthesizers made their way into the dance-music scene which grew in the 1970s.

Even though synthesizer use came to almost completely dominate music in the 1980s, we still mainly owe the current interest in analog synthesizer sounds to dance music, which continued to evolve through the 1980s into the 1990s into genres known as "House", "Techno", and "Jungle", with an ever-growing list of subgenera and specializations [206]. While the mainstream synthesizer market had moved to basically fully-digital systems by the late 80s, as FM and sample-based synthesis took over the market, experimenters in dance music still made use of the basic modulatability of analog synthesizers to produce highly rhythmic yet highly craftable and evolvable sounds. They took a set of older analog synthesizers, including some intended more as drum machines than melodic instruments, and made them the core of their own particular genres of music.

Thus, by the early/mid 1990s, as house/techno music was becoming more mainstream, the market for "vintage" analog synthesizers was actually stronger than ever, due to a mixture of their popularity in that thriving music scene, and a growing nostalgia for the sounds of the pre-digital synthesizers. At about the same time, computing power was coming to the point where real-time digital synthesizer algorithms could progress beyond the ultra-simple FM and wave-sampling paradigms which had dominated up till then. Synthesizer designers had for a while been realizing that more modulatability and interactivity were needed (especially in comparison to sample-based synths of the time). It became obvious that a popular and useful direction for digital synthesis was in the implementation of the capabilities of analog synthesis, particularly as used in the still-evolving dance genres. Such capabilities included:

- Highly flexible modulation capabilities — in particular, flexibility in all aspects of modulation: sourcing, sinking, and routing

- Configurable, modular algorithms (as opposed to static algorithms) — in essence, the ability to have the equivalent of a full modular synthesizer, which the user can re-patch as desired

- Implementations of highly-modulatable, yet still not grossly expensive algorithm primitives, such as oscillators and filters, with a minimum of identifiable "digital" artifacts such as aliasing

The first two capabilities are rather straightforward to implement at a low level, though there are significant user-interface issues that must be dealt with to make such systems both powerful and usable. It was the third capability that provided the most unsolved problems at the time. Though computing power was increasing continuously, there was still the desire to run as much processing as possible at any particular time, so the more efficient an algorithm could be made, the better.

Hence research such as this thesis. The research presented in this thesis was begun around 1995, when the ability to implement such algorithms on general-purpose computers in real-time was finally becoming possible, and when the current renaissance in analog-style synthesis was just beginning. the author published two papers on the topic in 1996, but then took about ten years off to work in industry, and other academic work has continued from that point. More importantly, in the years since this work started, there has been an explosion of commercial synthesizers, both software and hardware, implementing some sort of virtual analog, though the academic side stayed relatively quiet (this is changing, as more academics interested analog-synthesis simulation are starting to appear). As such, we must assume that much of the more obvious (and maybe less-obvious) extensions and variations to this work have been done in the commercial sector by now, though most of that work is trade secret, so one can only guess. the author's own opinion is that, due to the popularity of nonlinear aspects of virtual analog, the techniques of oversampling and distortion modeling have been particularly strongly explored (issues that this thesis will not get deeply into). However, since such work has been done as trade secrets, each manufacturer and developer has had to re-invent their own techniques for such things (and technical capabilities might vary widely across the industry), and so progress may not be as far as one might expect for a ten-year period. Hence, recent papers on circuit-based modeling of the Moog-style filter [110] are still probably not that far behind (or possibly still ahead of) the work being done commercially.

1.1.2 Review of Synthesis Techniques

The basic problem to be solved in sound synthesis can be considered to be one of how to create interesting sounds which can be viably and effectively controlled for performance. The various categories of synthesis algorithms have tended to approach that problem from various directions:[3]

Additive Based on the fact that all sound can be decomposed into a superposition of primitive waves (usually sinusoids), additive synthesis works by determining the set of sinusoids to generate at any moment, including their frequency, amplitude and phase, and generating them. Early methods did this by brute force (i.e., individual oscillators or specialized

[3]This is not intended to be an exhaustive list, but rather an attempt to touch on the major concepts. Interested readers are pointed to overviews such as [223] and [221]

oscillator-bank hardware) and were hence extremely limited in the number of sinusoids they could generate. Such implementations existed in the analog domain (indeed, the Telharmonium, whose concept was patented in 1897 [36], and considered by many to be the first electric music synthesizer, was effectively additive, as are the Hammond tone-wheel organs), but most explicitly additive synthesizers have been digital. Later methods ([37][224]) moved to using the inverse Fast Fourier Transform to generate many sinusoids efficiently. At this point, the difficulty for this method became one of managing the vast amount of data required to exactly specify the sound an its evolution through time.[4] If one does not want to be too particular with terminology, one can also consider any synthesis technique which builds up a complex sound spectrum by combining a number of simpler sounds which contain various subsets of the final spectrum to be somewhat additive.

Subtractive In contrast to additive synthesis, which builds up a sound from component parts, subtractive synthesis starts with very complex signals (spectrally) and uses filters to shape the sound to the desired spectral envelope. Due the to controllability of the filters and the useful effects of simple oscillator combinations, compelling animated sounds could be created rather simply (again, as opposed to having to specifically construct such behaviors in orthodox additive synthesis). Furthermore, this is the primary technique used in *Switched-On Bach* and most early commercial synthesizers up through the early 1980s, so it effectively became the "classic" synthesizer sound.

Wavetable/Sampling The most obvious digital music method is simply to record a sound and play it back in various ways. That is the heart of sample-based synthesis. As such, it can in some ways be considered a digital implementation of the "Musique Concréte" genre of modern music (a movement in early/mid 20th-century modern music which used tape recording and editing techniques to produce various effects), but it has been so popular that the connection is only noted by academics, on occasion, and usually followed by the discussion of whether Granular synthesis is a more appropriate successor. Within commercial music synthesizers, sample-based synthesis was originally hampered by the high cost of memory, such that it was either limited to very expensive systems (such as the Fairlight CMI, New-England Digital Synclavier, Kurzweil K520, or other studio-level instruments), or limited to very short samples. Early on, this second use was usually named "wavetable" synthesis, and the concept was that a single cycle of a waveform would be kept in memory and played back in a loop at various rates to get various pitches. As memory became more affordable, the concepts of wavetable synthesis and sample-based synthesis merged, the loop of the wavetable extending to cover more cycles (up to hundreds or thousands), and the full sound extending to the full recorded sample, including the recorded attack and decay of the sound. By the

[4] Furthermore, time/frequency information can be analyzed from recordings and used in a sampling-like manner, blurring the distinction between sampling and additive synthesis.

mid 1990s it was becoming a problem that sample-based synthesis only "sampled" discrete points in performance/sound space, and thus the basic method was limited in its variability ("expressiveness"). Most work since then in this field has been in adding expressiveness and modulatability, often by merging the technique with other methods, like subtractive synthesis. The ultimate in sample-based synthesis, however, may be the concept of "transition sampling", as used in the Synful synthesizer [156],[5] whereby not only are the held (pseudo-static) portions of instrument notes sampled, but all possible transitions between notes (within reason)[6], together with databasing and intelligence to select and splice together the correct static sections and transitions in real-time.[7] In general, since the late 1980s, sample-based synthesis has been the dominant method in commercial music synthesis, despite its limitations.

FM/Waveshaping Due to the high cost of memory in early digital systems, the most obvious digital technique, sample-based synthesis, was not the first viable real-time digital synthesis technique. Instead, *frequency modulation*, a very simple algorithm [42] which requires at most the memory to hold a sine table, became the first commercially successful digital synthesis method (via the Yamaha DX7 and later synthesizers). The high-level concept of FM is that waves of controllable complexity can be generated by controlling the modulation index in an FM algorithm. Unlike amplitude modulation, frequency modulation (and *phase-modulation*, which is simply FM with a differentiated input[8]) could generate extremely complex waves using a very small number of interconnected oscillators (as few as one, using feedback FM), and the spectral complexity could be easily controlled by simple scaling operations. As such, the technique is conceptually somewhere between additive and subtractive synthesis (in terms of spectral control): it need not be as brute-force as additive, but control of the spectral envelope is handled in a more round-about manner. Most appealing early on was the ease of creating inharmonic (bell-like) tones. Waveshaping, whereby a wave (often sinusoidal) of controllable amplitude is passed through some sort of controllable nonlinearity, is related mainly due to the fact that the shape of the nonlinearity can be considered to be one cycle of an oscillator shape, and the input wave phase-modulates that oscillator. However, once memory became inexpensive, wavetable and sampling synthesis became the norm, relegating FM to at most a subset of sounds for which it worked particularly well. This relegation had been essentially complete by the time of the resurgence in interest analog-synthesis techniques, so even though FM is inherently highly modulatable, it wasn't really considered in favor of directly recreating subtractive synthesis.

Granular Whereas additive synthesis builds up a complex spectrum by adding together a set of

[5] `www.synful.com`, note that even though the techniques are that of sampling, this synthesizer actually uses an IFFT additive core for the resynthesis of spectrally-analysed samples.

[6] A traditional failing point of sample-based synthesizers is unrealistic transitions

[7] Concatenative speech synthesizers have also evolved in this direction, for example: [264].

[8] or FM is PM with an integrated input

waves with simple spectra, granular synthesis [222] builds up a complex signal in time, adding up a set of waves that are simple in both frequency and time. Unlike most of the other techniques, granular synthesis is normally performed stochastically (though certain pitch-synchronous granular techniques exist, particularly in voice processing [225][133]). One view of granular synthesis presents it as a wavelet version of additive synthesis: creating a sound by superposition of "grains" which occupy certain small areas in time/frequency space. Other methods of granular synthesis, rather than starting from theoretical wavelets, use short-time snippets of some source wave(s) as their grains. As such, these methods are as much sound processors as sound synthesizers. Often, by controlling the way the grains are played and modified, various sound transformations such as pitch shifting, time-scale modification, and/or formant-modification can be accomplished. On the other hand, the grains can be played back in a way that completely obliterates the original signal, and some composers like to experiment with the boundary between transformation and obfuscation in their compositions. For the last decade or two, granular techniques have been among the most popular techniques in the academic computer-music scene.

Modal This type of synthesis ([3], [34]) describes a class of sounds and musical instruments as one or more resonators which are excited by some signal to produce the resulting sound. Originally, most modal synthesis modelled instruments with slowly-decaying resonators, such as bells, bars (marimba, vibraphone), various percussion instruments, and plucked or struck strings. Later, modal synthesis could be considered to cover the modeling of systems with less-resonant resonators, where the properties of the excitation become more important. At an extreme, one might classify source-filter vocal models as sort of modal synthesis (i.e. feeding a resonant system with somesort of excitation signal), though as with the modal synthesis of strings, the boundary between modal synthesis and physical modeling becomes blurred.[9] At some level, modal synthesis can be implemented both in analog and digital systems, though most explicitly modal methods have been digital, as with most synthesis methods in recent years.

Physical Modeling This (mostly digital) philosophy attempts to duplicate the behavior of a (usually) acoustic instrument by implementing a model of the physics of the sound production. Also sometimes called "Waveguide Synthesis" [235][236], as some of the basic concepts and efficiencies come in the modeling of the acoustic waveguides which exist in many acoustic instruments. As physical reality tends to be an massively parallel processing system, physical modeling synthesis involves some amount of simplification, and most of the research

[9] Julius Smith has noted that any diagonalized state-space model could be considered a modal-synthesis model, as the diagonalization makes the modes of the system explicit [239]. Using this as a definition, non-diagonalized source/filter models would end up classified as Subtractive Synthesis.

(and differences in approaches and philosophies) in this area centers around the simplification techniques and how far to simplify, or not to. At one extreme of simplification is a technique dubbed "PhISM" by its first author [47][48] (standing for "Physically-Informed Sound Modeling"), whereby the physics inspires mainly the implementation of the high-level behavior of the sound, and less the low-level implementation. Note that some large-scale modal-synthesis systems have also been used to implement very fine-grained physical models. Also note that particularly deep digital models of analog synthesizers end up using similar philosophies and techniques, and can effectively be considered physical models.

Ad-Hoc This describes the vague technique of using whatever method is most appropriate for any particular instrument or sub-algorithm within an instrument. It connotes the capability of the user to specify, to some level of granularity, the algorithm to be used for an instrument. Users of software synthesis systems such as CSound, Max/MSP, PD ("Pure Data"), CLM ("Common Lisp Music"), SuperCollider and STK (Synthesis ToolKit), to name just a few, are free to implement whatever methods they prefer, and are not constrained to any single method within any particular "orchestra" or "instrument" algorithm, and hence can be thought of as working in this realm. This can be considered the logical evolution of the concept of the modular synthesizer, with a virtual set of modules far beyond what was possible with early analog synthesizers.

1.1.3 What is "Virtual Analog"?

Virtual Analog is the attempt to implement digitally the algorithms employed in analog synthesizers (most typically subtractive synthesis) and hence emulate their sounds. The first commercial synthesizer of this type is generally considered to be the Clavia Nord Lead, which was introduced in 1995. Note that the term "Virtual Analog" was originally coined by Yamaha as the name of the technique used in their AN1x synthesizer, introduced in 1997. Yamaha had been using the term "Virtual Acoustics" for their implementations of physical modeling synthesis, so "Virtual Analog" was a straightforward choice. Other manufactures used their own terms when they introduced their analog-emulating synths, such as "Analog Modeling", "Analog Emulation", etc.

Interestingly, some enthusiasts are starting to use these multiple terms to differentiate between different implementation strategies within the field: "Virtual Analog" being applied to higher-level subtractive synthesis methods, and "Analog Modeling" being applied to low-level emulation of actual analog circuits [282]. The need for differentiating terms is necessary, of course, but it is sometimes strange to see terms which originally had effectively the same meaning attracting distinct meanings.[10]

[10]Or vice-versa, as in the case of wavetable synthesis and sample-based synthesis, which originally described distinct techniques but now are used interchangeably.

Unfortunately, the boundary of Virtual Analog is very vague. A synth which simply implements digital subtractive synthesis, with no regard for aliasing, and maybe just using basic digital filters (biquads, or maybe state-variable filters), but does implement good generic modulation capabilities, is in some ways emulating the modulation capabilities of analog synthesizers but not attempting to model the oscillators and filters very closely. It is even less clear if such a synthesizer uses a basic wavetable synthesizer as its core waveform generator (as in the Yamaha CS1x, which was effectively a sample-based General-Midi/XG synth, with expanded modulation capabilities). Since analog synthesizers had a variety of "features" which digital synthesizers may or may not implement, does a Virtual Analog system have to emulate them all, or just one, or "most"?

For the purposes of this thesis, Virtual Analog means the implementation of:

- Oscillators generating Impulse-train, Sawtooth, Triangle, Square, and Rect(angular) waves, with as little aliasing as is viable, and with modulation of amplitude, frequency, (and possibly other parameters, such as duty-cycle/pulse-width) as inexpensively as possible. Modulation is expected to reach up to audio rates. Phase sync is also a desirable feature.
- Resonant Lowpass (and possibly highpass and bandpass) filters, preferably in the same forms as in "classic" analog synths (i.e.,as in the Moog four-pole loop, or state-variable, as in the Yamaha CS-80) with full-rate modulatable corner/center frequency and Q, though it may be acceptable to restrict Q modulation in some usage cases if it is too expensive (as Q was not modulatable anyway in some synths, such as the Minimoog).

Other operations which occurred in basic subtractive-synthesis analog synthesizers, such as envelope generation, arpeggiation/sequencing and amplitude scaling, are considered sufficiently trivial as to not need special work to model "accurately". Various other special effects, which existed in some analog synthesizers, such as sample and hold, ring modulation, etc. are expected to be nontrivial, but not expected to be universally required and can be the subject of later research.

1.2 Basic Filter Concepts

The filters that we will be dealing with can be considered dynamical systems in either continuous time ("CT") or discrete time ("DT"). The waveform generators and filters used in most analog synthesizers operated in continuous time, whereas the algorithms we wish to implement will operate in discrete time. Signals in systems in these domains are often analyzed in transformed domains, "frequency domains" corresponding to each time domain.

Continuous-time filters are usually analyzed in the Laplace domain, whose main variable is a generalization of frequency (s), related to the time domain by the Laplace Transform [26]:

$$X(s) = \mathcal{L}[x(t)] = \int_{-\infty}^{\infty} x(t)e^{-st}dt \tag{1.1}$$

The Fourier Transform of at CT signal is the Laplace Transform evaluated only on the imaginary axis of the s plane (i.e.,on $\mathrm{Re}(s) = 0$). The Laplace Transform can be interpreted as projecting the time-domain signal onto a family of sinusoids, each described by a value of s: e^{sT}, as such, if $x(t)$ matches well with e^{sT} for a particular value of s, then $|X(s)|$ will be large at that value of s. If we look at the real and imaginary parts of s separately: $s = \sigma + j\omega$, then e^{st} corresponds to a sinusoid of whose phase rotates at a frequency of ω radians/sec (if t is in seconds), and whose amplitude decays or increases exponentially, depending on the sign of σ, as $e^{\sigma t}$. Therefore, if $x(t)$ contains an exponentially decaying sinusoid, then $X(s)$ will probably have a peak at a location off the imaginary axis, according to the decay rate. As noted, the Fourier Transform, being the projection onto sinusoids of constant amplitude and infinite extent, is a slice of the Laplace Transform on the line $\mathrm{Re}(s) = 0$, which makes sense since there is no decay or expansion when $\sigma = 0$.

The filters we will be dealing with can also be considered (at least as an approximation) as linear, and much of the time, we will think of them as being (at least locally) time-invariant. As such, their output $y(t)$ can be viewed as the convolution of their input $x(t)$ by their impulse response $h(t)$:

$$y(t) = (x * h)(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau \tag{1.2}$$

and by the properties of the Laplace Transform, the operation in the s domain is a multiplication of their transforms [135]:

$$Y(s) = X(s)H(s) \tag{1.3}$$

The filter's convolution kernel $h(t)$ is called its impulse response because if $x(t) = \delta(t)$, the Dirac delta function, then the convolution becomes

$$y(t) = \int_{-\infty}^{\infty} \delta(\tau)h(t-\tau)d\tau = h(t) \tag{1.4}$$

The impulse response can thus be thought of a good description of how the filter "rings out" when pinged with an input that is impulsive in nature.[11]

Integration and differentiation can be expressed in the s domain as equivalent to filtering by special filters:

$$\mathcal{L}\left[\frac{dx}{dt}\right] \Rightarrow sX(s) \tag{1.5}$$

$$\mathcal{L}\left[\int x(t)dt\right] \Rightarrow X(s)/s \tag{1.6}$$

As such, linear ordinary differential equations, which are actually continuous-time filters, can

[11] Of course the true output will be the convolution of the impulse response and the shape of the ping input, but for intuition purposes, one often things of such pings as being "impulse like".

easily be described in terms of s:

$$y + d_1\frac{dy}{dt} + d_2\frac{d^2y}{dt^2} + \cdots d_N\frac{d^Ny}{dt^N} = n_0x + n_1\frac{dx}{dt} + n_2\frac{d^2x}{dt^2} + \cdots + a_M\frac{d^Mx}{dt^M} \tag{1.7}$$

transforms to

$$\begin{aligned} Y(s)[1 + d_1s + d_2s^2 + \cdots + d_Ns^N] &= X(s)[n_0 + n_1s + n_2s^2 + \cdots + n_Ms^M] \\ \frac{Y(s)}{X(s)} &= \frac{n_0 + n_1s + n_2s^2 + \cdots + n_Ms^M}{1 + d_1s + d_2s^2 + \cdots + d_Ns^N} \\ \frac{Y(s)}{X(s)} &= \frac{N(s)}{D(s)} \end{aligned}$$

Many of the properties of such filters can be related to properties of the $N(s)$ and $D(s)$ polynomials. For example, roots of N are called "zeros" of the filter, and correspond to frequencies which are completely attenuated by the filter. Roots of D are called "poles" of the filter, and correspond to frequencies which are boosted (infinitely) by the filter. They are also thought of as the "resonances" of the filter.

Of interest in this thesis is the fact that the impulse response of a filter can often be understood almost completely in terms of the locations of its poles: poles with larger imaginary part ring at higher frequency. Poles with small negative real parts correspond to ringing modes which decay slowly. Larger negative real parts correspond to modes with faster decay. Poles on the imaginary axis correspond to modes with no decay. Poles with positive real parts correspond to modes which increase exponentially with time. In causal situations (i.e.,ones where there is time running in a forward direction, as opposed to situations where there may not be time at all (like spatial filters)), such poles would cause the filter to be unstable (typically an undesirable situation, though there are situations where temporary instability may be of use, and some nonlinear systems have saturations which will override any small-signal instabilities and keep the filter from completely "blowing up" by clipping the states of the filter[12]).

In discrete time, there is a similar relation between a discrete time signal $x(n)$ and its transform[13], denoted by $X(z)$. The transform is thus called the "Z Transform":

$$X(z) = \mathcal{Z}[x(n)] = \sum_{k=-\infty}^{\infty} x(k)z^{-k} \tag{1.8}$$

Convolution is similarly

$$y(n) = (x * h)(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) \tag{1.9}$$

[12]For example, it is not uncommon to place a filter into a slightly unstable state and use it as a (nearly) sinusoidal oscillator, with the saturation nonlinearity and the instability interacting to give a relatively constant-amplitude signal.

[13]or at least the most popular transform domain for discrete time systems, the z transform

and in the z domain

$$Y(z) = X(z)H(z). \tag{1.10}$$

Again, a (linear and time-invariant) discrete-time filter can be described by its impulse response $h(n)$.

As in the continuous-time case, where s had a particular meaning (differentiation), in discrete time, z is interpreted as a time shift of one sample, forward in time. Since that is typically a non-causal action, we tend to see z more "in real life" as its inverse z^{-1}, which corresponds to a delay of one sample: $X(z) + z^{-1}X(z)$ corresponds to $x(n) + x(n-1)$. Higher powers of z^{-1} correspond to more samples of delay: $z^{-10}X(z)$ corresponds to $x(n-10)$.

Therefore, discrete-time filters, which are usually interpreted as "difference equations," can be transformed into the z domain much like how we transformed a differential equation previously:

$$y(n) + d_1 y(n-1) + d_2 y(n-2) + \cdots d_N y(n-N) = n_0 x(n) + n_1 x(n-1) + \cdots + a_M x(n-M) \tag{1.11}$$

transforms to:

$$\begin{aligned}
Y(z)[1 + d_1 z^{-1} + d_2 z^{-2} + \cdots + d_N z^{-N}] &= X(z)[n_0 + n_1 z^{-1} + n_2 z^{-2} + \cdots + n_M z^{-M}] \\
\frac{Y(z)}{X(z)} &= \frac{n_0 + n_1 z^{-1} + n_2 z^{-2} + \cdots + n_M z^{-M}}{1 + d_1 z^{-1} + d_2 z^{-2} + \cdots + d_N z^{-N}} \\
\frac{Y(z)}{X(z)} &= \frac{z^{N-M}[n_0 z^M + n_1 z^{M-1} + \cdots + n_M]}{z^N + d_1 z^{N-1} + \cdots + d_N} \\
\frac{Y(z)}{X(z)} &= \frac{N(z)}{D(z)}
\end{aligned}$$

As in the continuous-time case, much of the filter's properties can be gleaned from the polynomials $N(z)$ and $D(z)$, in much the same way as before. However, the interpretation of locations in the z plane is a bit different than locations in the s plane. In continuous time, a point in the s plane corresponded to a signal (or impulse response) of e^{sT}, but in discrete time, a point in the z plane corresponds to a signal (or impulse response, or...) of z^n. As such, the values of z that correspond to decaying signals (or to stable poles in a causal situation) are those for which $|z| < 1$, and the values that correspond to increasing signals (or to unstable poles in a causal situation) are those for which $|z| > 1$. The values of z that correspond to signals with no change in amplitude are those right on $|z| = 1$. That is a circle of radius 1, centered on the origin (z=0). This is typically referred to as the "unit circle". Further, signals with high frequency correspond to locations in the z plane whose angles from the positive real axis are larger. Subsequently, $z = 1$ corresponds to zero frequency ("DC"), as $s = 0$ did in continuous-time, and as z moves along the unit circle away from $z = 1$ (either up or down), it corresponds to higher and higher frequencies, until it reaches $z = -1$, at which point it represents the frequency $f_s/2$ (i.e.,half the "sampling rate"). Due to a phenomenon called "aliasing", if you continue to rotate z past $z = -1$, it corresponds to smaller frequencies again

(the "negative frequencies", or the other path that could have been taken from $z = 0$). The signal is actually exactly what it would be if you sampled a signal whose frequency was higher than $f_s/2$, but the resulting sampled signal would appear as though it were the lower frequency. In fact, as is mentioned in most books on discrete-time signals, the unit circle represents *all* frequencies, they just keep wrapping around the circle on top of each other, so that any particular point represent not only some frequency less than $f_s/2$, but also that frequency plus/minus all multiples of f_s. Most tend to consider only the lowest range of frequencies ($-f_s/2 < f < f_s/2$), except when sampling a continuous-time signal, in which case some care must be taken about the frequencies higher than $f_s/2$ (usually one tries to filter than out with an antialiasing filter).

Now, on the topic of converting between a continuous-time signal/system and a discrete-time one, the ideal relationship is based on the fact that z represents a delay of -1 samples. In the s domain, that delay is the particular allpass filter e^{-sT}, where T is the sampling period. As such, the ideal relation between the two domains is $z = e^{sT}$. Using Euler's formula ($e^{j\theta} = \cos(\theta) + j\sin(\theta)$), one can easily see how the s plane and z plane relate to each other:

- The imaginary axis in the s plane maps to unit circle (wrapped around an infinite number of times).
- The real axis in the s plane maps to the positive real axis in the z plane.
- The left half s plane (left of the imaginary axis) maps inside the unit circle.
- The right half s plane maps outside the unit circle.
- $s = 0$ maps to $z = 1$.

We will look later on into existing methods for performing conversions from continuous-time to discrete time.

1.2.1 Some Basic Filter Definitions

FIR/IIR

Discrete-time filters with no feedback are denoted "FIR," which stands for "Finite Impulse Response," because an implementable form of such a filter can only have a finite number of taps, and hence its impulse response will last for a finite amount of time. Such filters only have a numerator polynomial in their transfer functions:

$$H_{FIR}(z) = n_0 + n_1 z^{-1} + \cdots + n_N z^{-N} \tag{1.12}$$

Such a filter is also generally considered to implement only zeros (frequencies where the transfer function goes to zero).

A filter with feedback tends to have a response consisting of some combination of exponentially-decaying parts. Since exponential decays theoretically never get all the way to zero, such a response will cover an infinite amount of time, hence the term "Infinite Impulse Response" filter, or more commonly "IIR." Feedback tends to cause the transfer function to have a denominator polynomial as well as a numerator polynomial:

$$H_{IIR}(z) = \frac{n_0 + n_1 z^{-1} + \cdots + n_M z^{-M}}{1 + d_1 z^{-1} + \cdots + d_N z^{-N}} \tag{1.13}$$

Such a filter is generally considered to implement both zeros and poles (frequencies where the denominator of the transfer function goes to zero, and thus where the transfer function has a singularity).

Coefficient Naming Issues

Unfortunately, different schools tend to use different standards for the naming of numerator and denominator polynomials. Some use the following form:

$$H(z) = \frac{a_0 + a_1 z^{-1} + \cdots + a_M z^{-M}}{1 + b_1 z^{-1} + \cdots + b_N z^{-N}} \tag{1.14}$$

And hence there is the assumption that to refer to "b_2" is to refer to a denominator coefficient. On the other hand, others (including Matlab) use the exact opposite:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}} \tag{1.15}$$

And hence "b_2" would be assumed to be a numerator coefficient. The problem becomes most difficult in describing coefficient design equations, as it quickly becomes confusing which coefficients are being described unless the equations are accompanied with either a transfer function or documentation denoting the usage of the coefficients.

the author, in his work at Analog Devices, Inc., which involves distributing a standard set of DSP modules and design code to users from all backgrounds, has decided to use neither standard, and instead name coefficients "n_0, n_3, d_2," etc., such that "N" means "numerator" and "D" means denominator. It is hoped that the use of such a system will reduce some confusion. the author has attempted to use that system where possible in this thesis. However, some text or figures from before this decision may slip by from time to time.

Similarly, there can be confusion regarding the sign of feedback coefficients. The difference equation

$$y[n] = x[n] + 0.3y[n-1] + 0.6y[n-2] \tag{1.16}$$

Which implements positive feedbacks, has the following transfer function:

$$Y/X = \frac{1}{1 - 0.6z^{-1} - 0.6z^{-2}} \tag{1.17}$$

which has negative denominator coefficients. Conversely, a transfer function such as

$$Y/X = \frac{1}{1 + .5z^{-1} + 0.5z^{-2}} \tag{1.18}$$

corresponds to the following difference equation:

$$y[n] = x[n] - 0.5y[n-1] + 0.5y[n-2] \tag{1.19}$$

Some people consider the difference equation to be the "standard representation", and hence prefer the first pair of equations (i.e.,prefer to think of the "correct sign" of the coefficients being that in the difference equation), whereas others (including Matlab) think of the transfer function as the standard representation, and so consider the transfer function coefficients to be the ones with the correct sign.

This issue is a bit more insidious than the first one. the author has no theories as to what can be done about it. In this thesis, the author hopes to not introduce too much sign confusion, hoping to explicitly show transfer functions and/or difference equations wherever coefficients or their equations are shown.

Some Nice Basic Polynomial Properties

These are all well known polynomial properties, but they are good to remember in DSP and in Root-Locus work.

- If a polynomial has real coefficients, all complex roots will come in complex-conjugate pairs. One must remember, though, that if a pair comes together onto the real axis, they can move to distinct real values and no longer be a conjugate pair.
- In a 2^{nd}-order polynomial with real roots, if the roots $r_{1,2}$ are complex, then:
  - Their radius can be found by: $r^2 = r_1 r_2$.
  - Their real parts can be found by: $(r_1 + r_2)/2$.
  - One of their imaginary parts can be found by: $(r_1 - r_2)/2$.

  These facts can be useful if the poles are still in symbolic form.
- If one flips the sign of all the odd-order coefficients, all the roots are negated. This can be shown by taking a polynomial (in s) and substituting $-s$ for s. Thus if the coefficient are

real, this has the effect of reflecting the roots about the imaginary axis. This can be used to determine the required numerator coefficients for a continuous-time allpass filter from the denominator coefficients.

- If one reverses the order of coefficients in a polynomial, the roots are inverted. This can be shown by taking a polynomial (say in z) and substituting z^{-1} for z. If the polynomial has real coefficients, this has the effect of inverting the radius of all the roots. This can be used to create the numerator for a discrete-time allpass filter from the denominator coefficients.

Classic Twopole Filter Parameters

This thesis will use definitions for pole properties based on analog twopole filters, as defined in [88, p. 140]. For a pole at some location s:

- $\omega_n = |s|$ is the *undamped natural frequency*
- $\omega_d = \mathrm{Im}(s)$ is the *damped natural frequency*
- $\sigma_0 = \mathrm{Re}(s)$ is the *damping rate*
- $\zeta = \sigma/\omega_n$ is the *damping ratio*
- $Q = 1/2\zeta = (2\zeta)^{-1}$ is the *quality factor*
- The angle of the pole from the imaginary axis relates to damping ratio and Q as: $\phi = \tan^{-1}(\sigma/\omega_d) = \sin^{-1}\sigma/\omega_n = \sin^{-1}(\zeta) = \sin^{-1}(1/2Q)$.
- Thus $Q = 1/(2\sin(\phi))$.

NOTE that ϕ above is the angle from the *imaginary axis*, not the real axis. Hence the use of sine rather than cosine.

In the s plane, contours of constant-value for various properties have various shapes:

Constant ω_n : Circles centered on $s = 0$ of radius ω_n.

Constant ω_0 : Horizontal lines, distance from real axis = $|\omega_0|$.

Constant σ : Vertical lines, distance from imaginary axis = $|\sigma|$.

Constant Q (constant ζ) : Rays emanating from the origin and heading into the left half plane at some angle $\phi = \sin^{-1}(1/2Q)$ from the imaginary axis. Poles in the right half plane have negative Q and ζ.

For digital filters, we may transform these properties to the z plane by $z = e^{sT}$. See Figure 1.1:

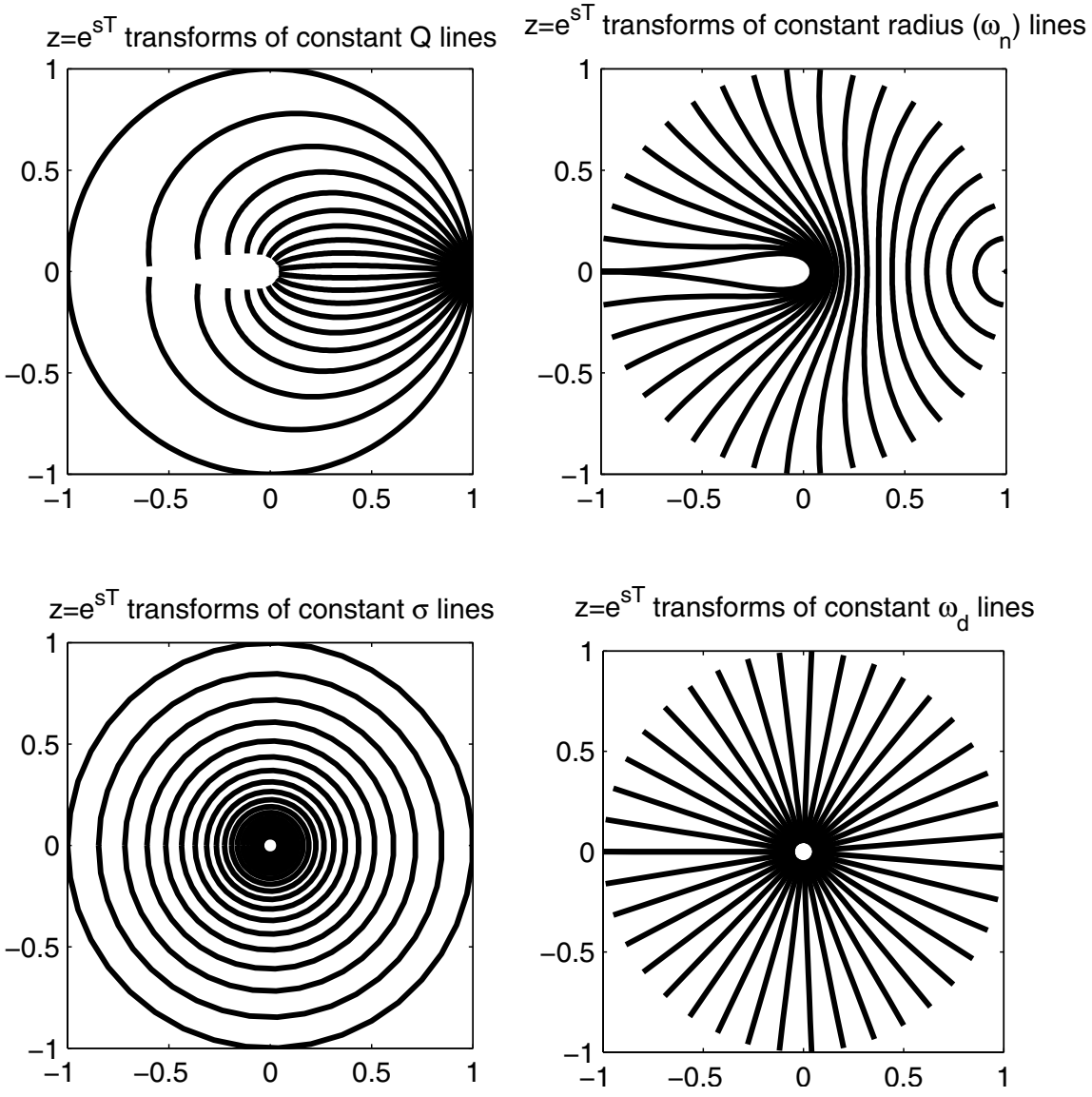


Figure 1.1: $z = e^{sT}$ transforms of various constant-parameter contours into the z plane from the s plane.

Constant ω_n : Small ω_n: circles centered on $z = 1$, larger ω_n, the circles distort and wrap around $z = 0$ until ω_n corresponds to an angle of π in the z plane.

Constant ω_0 : Straight radial lines passing through $z = 0$, angle a function of ω_0 and T.

Constant σ : Concentric circles centered on $z = 0$, radius a function of σ and T.

Constant Q (constant ζ) : Logarithmic spirals in the z plane, all exiting from $z = 1$ at angles that are functions of Q and T.

Discussion of Q, f_c, and bandwidth It is almost a tradition, in publications on parameterized filter design where Q and/or bandwidth are parameters, to note the lack of agreement in the filter-design community on a single definition of bandwidth and Q. Much of this disagreement stems from differing opinions on the concept of "true" bandwidth and Q: is it a property of the frequency response or is it a property of the dominant poles? Or in other words, is it a measurement of the width of a feature in the frequency response or is it a measurement of the decay rate of a resonance? At high Q (a narrow peak), things are typically quite straightforward to understand, and all definitions tend to agree, but as Q gets low (particularly near and below 1.0 or 2.0), things become confused.

This discussion will not attempt to solve this problem, but will give an overview of the issue, and state the definition that will be used in the rest of the thesis.

First, we should note that, historically, Q was originally defined to describe rather high-frequency, high-Q resonators, and as such was originally only applied to strongly resonant 2^{nd}-order systems (as a measure of the "purity", or "quality" of the resonator — hence the name "Q"). As such, the definition had some basic assumptions:

- There was a peak in the response.
- There was only one peak in the response.
- The peak had a recognizable maximum at a recognizable frequency (i.e, it was not flat-topped).
- There were no zeros to speak of.
- The impulse response decayed slowly enough for the concept of "cycles" to be obvious in the reading of the response.

As such, Q could be defined a number of ways (which we will soon get to), and for these resonators, the definitions were all equivalent (or to within some good approximations).

Since then, however, things become more difficult, as we try to apply the definition to systems with other properties, such as the following:

- Multiple resonances and peaks.
- Flat-top bands
- Significant spectral effects of zeros, or even filters with no poles at all, only zeros (i.e.,FIR filters).
- Response shapes with no peak at all (non-resonant lowpass or highpass filters, for example).
- Features close enough to DC or $f_s/2$ to have their spectral response shapes affected by leakage from the negative-frequency copies of the dynamics.

For now, these will be noted as "difficulties."

Review of Q definitions:

"Freq over BW" : ("Q_{bw}") The most often quoted definition: the center frequency of the peak divided by the bandwidth of the peak. Usually using the 3dB-points for defining bandwidth. At various times, researchers have used other bandwidth definitions where necessary, usually when a peak does not have a shape that allows 3dB points to be picked out (like a lowpass response with a weak peak) [57]. This definition breaks down when there is not a peak that can be measured, or if there is no single peak or center frequency. It can be applied to flat-top peaks, though the fall-off rates of the peak's sides might cause ringing in the impulse response which lasts longer than this Q definition would imply.

"Cycles to Decay" : ("Q_{decay}") This shows up in many many early definitions of Q, for example [180, p. 25]: "the number of cycles required for the amplitude [of oscillation] to reduce to $(1/e^{\pi})$.", and later in the same page it is written to be $\pi\nu_0/k$, where the oscillation amplitude is A_0e^{-kt}, and ν_0 is the undamped natural frequency (apparently in Hz). This translates to $\omega_0/(-2\sigma)$ or $1/2\tan(\phi)$ (where ϕ is the acute angle of the pole to the imaginary axis). In the z plane, using $z = e^{sT}$ to translate the pole location back to the s plane, this definition becomes $-(1/2)\angle(z)/\ln|z|$. This definition breaks down when the decay is sufficiently quick to not show cycles anymore (though the math can extend into this range just fine).

"Successive Cycle Decay" : ("Q_{succ}") $\pi/\ln(r)$, where r is the ratio of the amplitudes of successive cycles. This is effectively another way of measuring Q_{decay}, so we won't consider it further. Like the previous definition, this one breaks down conceptually when the decay is sufficiently fast as to hide any visible cycles.

"Pole Angle" : ("Q_{pole}") This definition $1/2\sin(\phi) = -\omega_n/2\sigma = -|s|/2\mathrm{Re}(s)$ [88] is based on the angle of the pole from the imaginary axis in the s plane, and hence is directly related to the damping ratio ζ as well. This definition differs from the Q_{decay} definition mainly in what the

limiting value is as the pole approaches the real axis: $Q_{decay} \to 0$, and $Q_{pole} \to 0.5$. The z-plane version of this definition is: $-(1/2)|\ln(z)|/\ln|z|$, or equivalently $-1/2\sin(\tan^{-1}(-\ln|z|/\angle z))$. This definition mainly breaks down when a dominant pole cannot be decided upon, or if there are no poles.

"SVF" : ("Q_{SVF}") This is an "expectation" definition: "What a state-variable filter sounds like when its q coefficient is set to $1/Q_{SVF}$." Since the Chamberlin state-variable filters are so popular, and since people often used them in ranges where the q coefficient gets as high as 2.0 (which due to their design equations is called "$Q = 1/2$", but is well below the existence of any sort of peak), a user could, through extensive experience listening to the filter, be of the belief that that particular filtering is what a Q of $1/2$ sounds like.

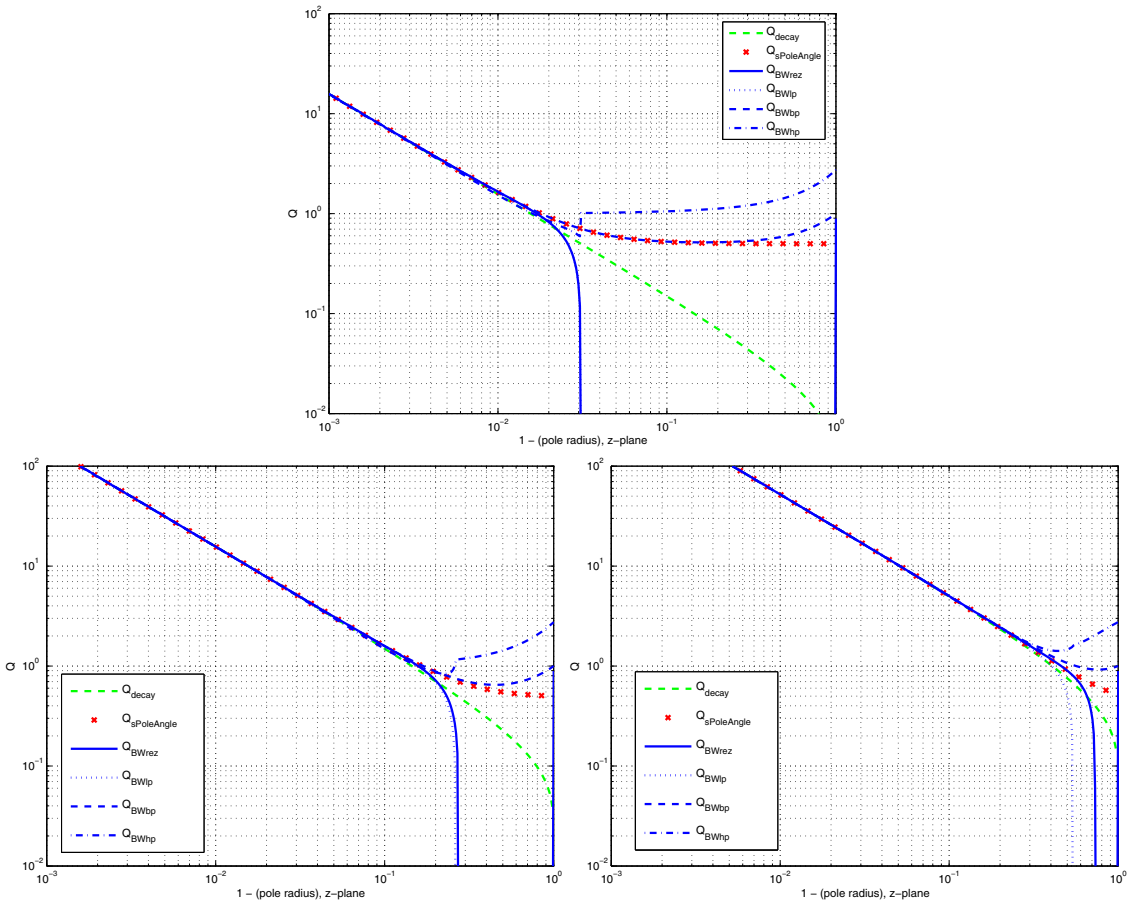


Figure 1.2: Comparison of Q definitions. Biquad with various zero arrangements (LP: both at $z = -1$, BP: one at $z = 1$, other at $z = -1$, HP: both at $z = 1$, Rez: no zeros), plotted against pole radii along lines of different angle in z plane. Top: $\theta = \pi/100$, Left: $\theta = \pi/10$, Right: $\theta = \pi/3$.

Let us compare some definitions experimentally . In Figure 1.2, we set up our test system as a two-pole two-zero filter with four different outputs: A lowpass one (two zeros on $z = -1$), a Highpass one (two zeros on $z = 1$), a Bandpass one (one zero each on $z = 1$ and $z = -1$), and a pure resonator (no finite zeros). We compare the measured Q definitions, which of course are different for each output, against the decay and pole-angle definitions. The filters are measured for various pole radii on a line of a particular angle. The most obvious effect we see is that all the definitions pretty-much agree for Q larger than 1 or 2 or so, but then diverge wildly "below" that. Much of the differences, particularly between the Q_{BW} cases, can be explained by difficulties in defining band edges once the corner peak has gone away. In these measurements, we followed the example of [57] and used "halfway power" definitions for the band edges (i.e.,the frequency where the response power is halfway between the peak value and the value at DC or $f_s/2$). If the value at DC or $f_s/2$ is zero, then this automatically becomes the $-3dB$ definition. Now, among the spectrum-based definitions, the output that is most like a "peak" is of course the bandpass output, and we see that over much of the range, the pole-angle definition is not too far away from it.

Next, let us compare using a Chamberlin state-variable filter (Figure 1.3). In this case, we only have the three outputs. The measurements are plotted against the state-variable filter's qq coefficient, which according to the Q_{SVF} definition, is equal to $1/Q$ (so we implicitly compare against that definition by whether a curve is a straight line or not). Note that since the filter is used up to qq = 2, we measured out to that limit. Again, if we compare the bandpass-measured Q and the pole-angle definition, we see that they are pretty close. In fact, when ff is less than about 0.3, these two definitions become effectively equal (and a straight line), for all values of qq! (this has been verified only experimentally, though). In essence, this tells that, at least at low frequencies in a digital state-variable-filter, Q_{pole}, Q_{bw}, and Q_{SVF} become equivalent. We also see that for larger ff, those two definitions are still the least deviated from the Q_{SVF} definition.

Therefore, as expected, all the definitions are equivalent for 2nd-order systems at high Q, but they diverge when the various definitions break down.

For example, a "cycles to decay" definition breaks down past critical damping, because there are no longer any features in the response that can be remotely described as "cycles". And as previously noted, beyond critical damping, there is no peak in 2^{nd}-order highpass and lowpass frequency responses, so peak-width based definitions break down.

Questions for Thought

Here we go back to the "difficulties" mentioned earlier (multiple peaks, flat tops, effects of zeros), and ask: can a definition originally made only for single-peak 2^{nd}-order resonators be successfully applied to other systems? For example, if an FIR filter implements an exponentially-decaying sinusoid, for its impulse response, can it be considered to have a Q? Now what about if it implements an exponentially-decaying triangular wave? or sawtooth? What if it implements a linear decay?

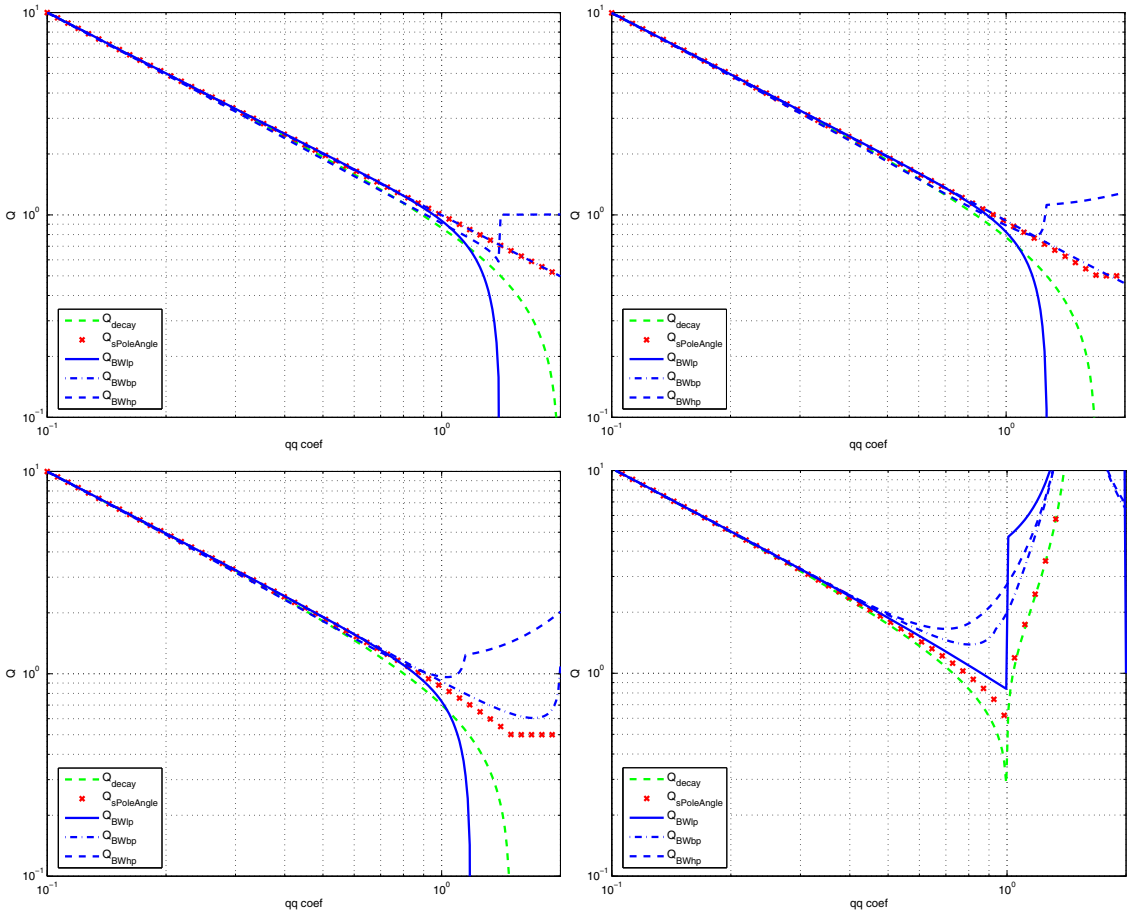


Figure 1.3: Comparison of Q definitions. Chamberlin SVF using various outputs for transfer-function-based measurements, plotted against SVF qq coefficient. Top Left: ff = 0.01, Top Right: ff = 0.3, Bottom Left: ff = 0.5, Bottom Right: ff = 1.0.

The author will be so bold as to opine that extending any definition of Q beyond high-Q 2nd-order resonators is rather arbitrary. It makes sense that Q_{BW} is the most common definition, as it can be applied to the widest variety of situations with some sort of success (like FIR filters). However, in situations where there truly is a dominant set of poles, there is no reason not to use one of the other definitions.

For this thesis, we will use the pole-angle definition. This choice is partly due to the fact that within the realm of symbolic low-order IIR filter design, the transfer-function-amplitude-based definitions quickly become very ungainly, and most design methods which use them eventually must resort to approximations due to the complexity of the formulas ([57] is one example). On the other hand, pole locations can often be, by comparison, relatively simple to deal with. Finally, since this thesis has at its heart the use of root-locus, and the assumption that the filters we will be working with can be analyzed according to their pole locations, a pole-based definition is the best fit. Still, there are some assumptions that are being made that should be noted:

- It is assumed that any other poles (as in the 4th and 5th-order Moog-style filters) stay sufficiently damped that the assumption of outer-pole dominance is satisfied. Or if they do not stay sufficiently damped, they are at least ringing at a sufficiently distant frequency so as to not adversely affect the shape of the primary peak.
- It is assumed that zeros do not play a significant part in the medium-to-high Q response of the system. If they exist, they are mainly used for gross shaping of the response (e.g. highpass, lowpass, bandpass).
- It is assumed that the primary region of operation is $Q > 1$, or if $Q < 1$ is used, the filter's response isn't expected to match any other definition of Q.

1.3 Review of Filter Discretization Methods

Although there is an ideal relationship between s and z ($z = e^{sT}$), it cannot be used to directly translate a rational function of finite degree in the s plane into one of finite degree in the z plane[14], and thus cannot be used to directly discretize a typical continuous-time filter in an order-preserving manner. Instead a number of techniques have been developed to translate rational functions in s into rational functions in z of similar degree, or to otherwise translate a continuous-time design to discrete time.

1.3.1 Common methods derived from numerical integration

Three well-known techniques derive from fixed-step numerical integration algorithms[87] [196]:

[14] i.e.,by variable substitution

Backward Difference Transform

This transform derives from the numerical integration concept whereby the area under a curve $x(t)$ is approximated by set of rectangles of width T. The "backward" part comes from the idea that each rectangle touches the curve at time nT and extends backward in time from that point. Hence, the running accumulation $y(nT)$ is:

$$\begin{aligned}
y(nT) &= y((n-1)T) + x(nT)T \\
&\overset{DT}{\Rightarrow} \\
y(n) &= y(n-1) + x(n)T \\
Y(1-z^{-1}) &= XT \\
\frac{Y}{X} &= \frac{T}{1-z^{-1}} = \frac{zT}{z-1} \\
&\text{Thus,} \\
\frac{1}{s} &\leftarrow \frac{Tz}{z-1} \\
s &\leftarrow \frac{z-1}{Tz} = \frac{1-z^{-1}}{T}
\end{aligned}$$

If we look at how the s plane maps to the z plane (Figure 1.4), we see that the left-half plane maps to a circle (which is expected since this is a conformal map), of radius $1/2$, which touches the unit circle at $z = 1$ and at $z = 0$ on the other side of the circle. As such, one would expect high-frequency behavior to be significantly damped in relation to the continuous time. The backward-difference integrator (top of Figure 1.4) is very straightforward, as one might expect. Something to note is that this integrator has a delay-free path from its input to its output. We should also note that, in many cases, the backward-difference is the first discretization tried, due to its simplicity. Since the region near $s = 0$ does map to a region near $z = 1$ (as will happen in all these cases), low-frequency dynamics do map rather well, so for sufficiently over-sampled sampling rates, this transform can be just fine.

In audio, this translates to: "If the important behavior of the filter is at very low frequencies relative to the sample rate, then the backward-difference may be fine." However, for a filter that may need to be swept up to significant percentages of the human hearing range, then backward-difference may not be viable when used at sampling rates which just barely contain the audible range, such as 44.1 kHz and 48 kHz, or do not at all, such as 22kHz and 24kHz.

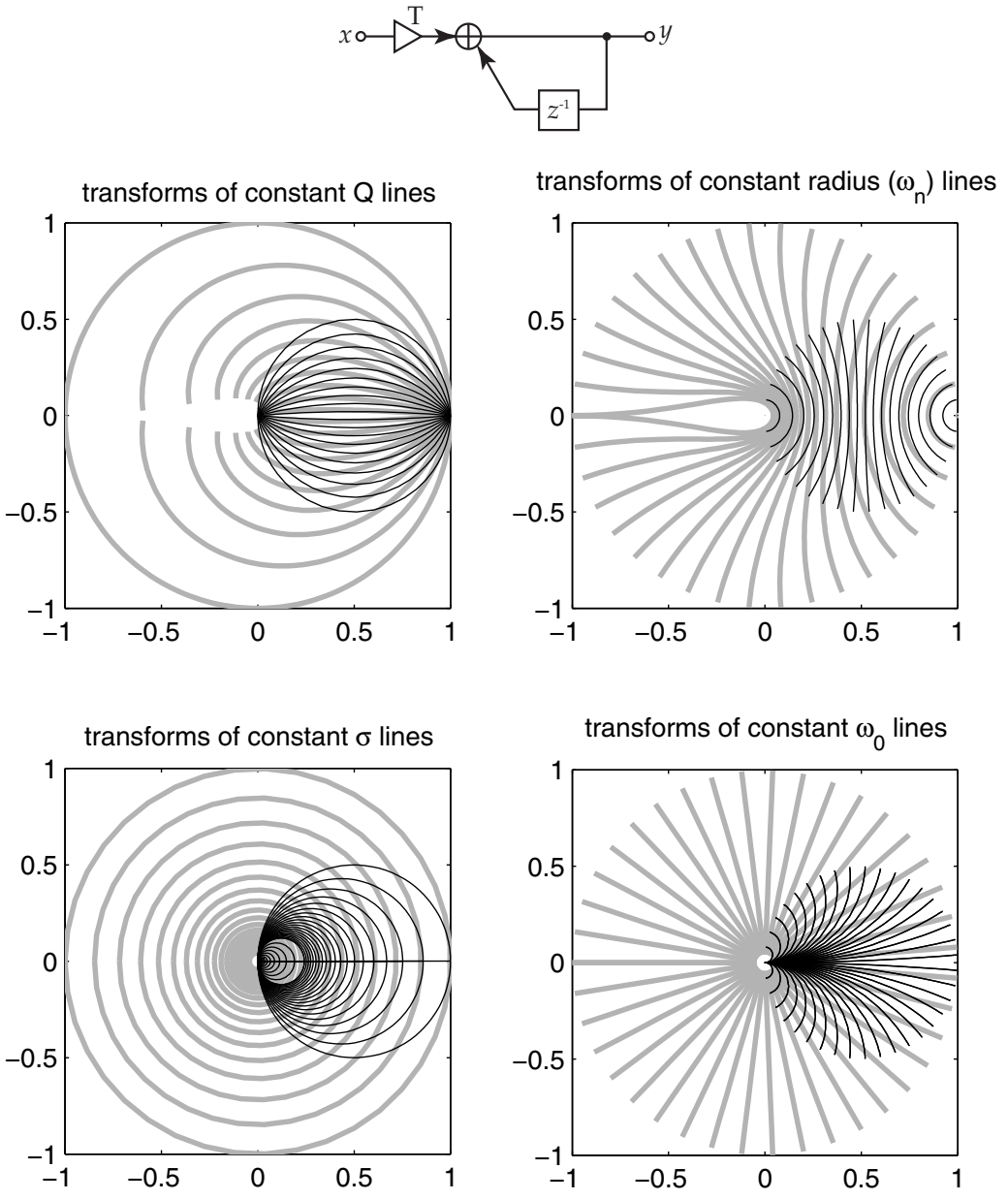


Figure 1.4: Top: Backward-Difference integrator. Bottom: Dark: Backward-Difference transforms of various constant-parameter contours into the z plane from the s plane. Light: the "ideal" $z = e^{sT}$ transforms of the contours.

Forward Difference Transform

The forward-difference derives from the slightly different numerical-integration rule whereby the approximation rectangles touch the curve at their lower extent in time, and extend forward in time. As such, the running accumulation is slightly different:

$$\begin{aligned}
y(nT) &= y((n-1)T) + x((n-1)T)T \\
&\overset{DT}{\Rightarrow} \\
y(n) &= y(n-1) + x(n-1)T \\
Y(1-z^{-1}) &= XTz^{-1} \\
\frac{Y}{X} &= \frac{Tz^{-1}}{1-z^{-1}} = \frac{T}{z-1} \\
&\text{Thus,} \\
\frac{1}{s} &\leftarrow \frac{T}{z-1} = z^{-1}\frac{T}{1-z^{-1}} \\
s &\leftarrow \frac{z-1}{T} = z\frac{1-z^{-1}}{T}
\end{aligned}$$

Looking at how this transform maps the s plane into the z plane (Figure 1.5), we can see why it use is generally discouraged: the left-half plane simply maps to a shifted version of itself. Therefore, a stable system in continuous time is not guaranteed to transform into a stable system in discrete time, as with the backward difference. Of course, with a sufficiently high sample rate and a sufficiently-damped system, one might be able to make use of such a transform.

It is of interest that the transformed integrator has its delay in the feed-forward path, such that there is no delay-free path through the integrator. This can explain stability issues: a loop containing such an integrator would have more phase in the loop and hence less phase margin. On the other hand, the direct transformation of a loop using this transform is less likely to have problems with delay-free loops (which we will see in later chapters can be an issue).

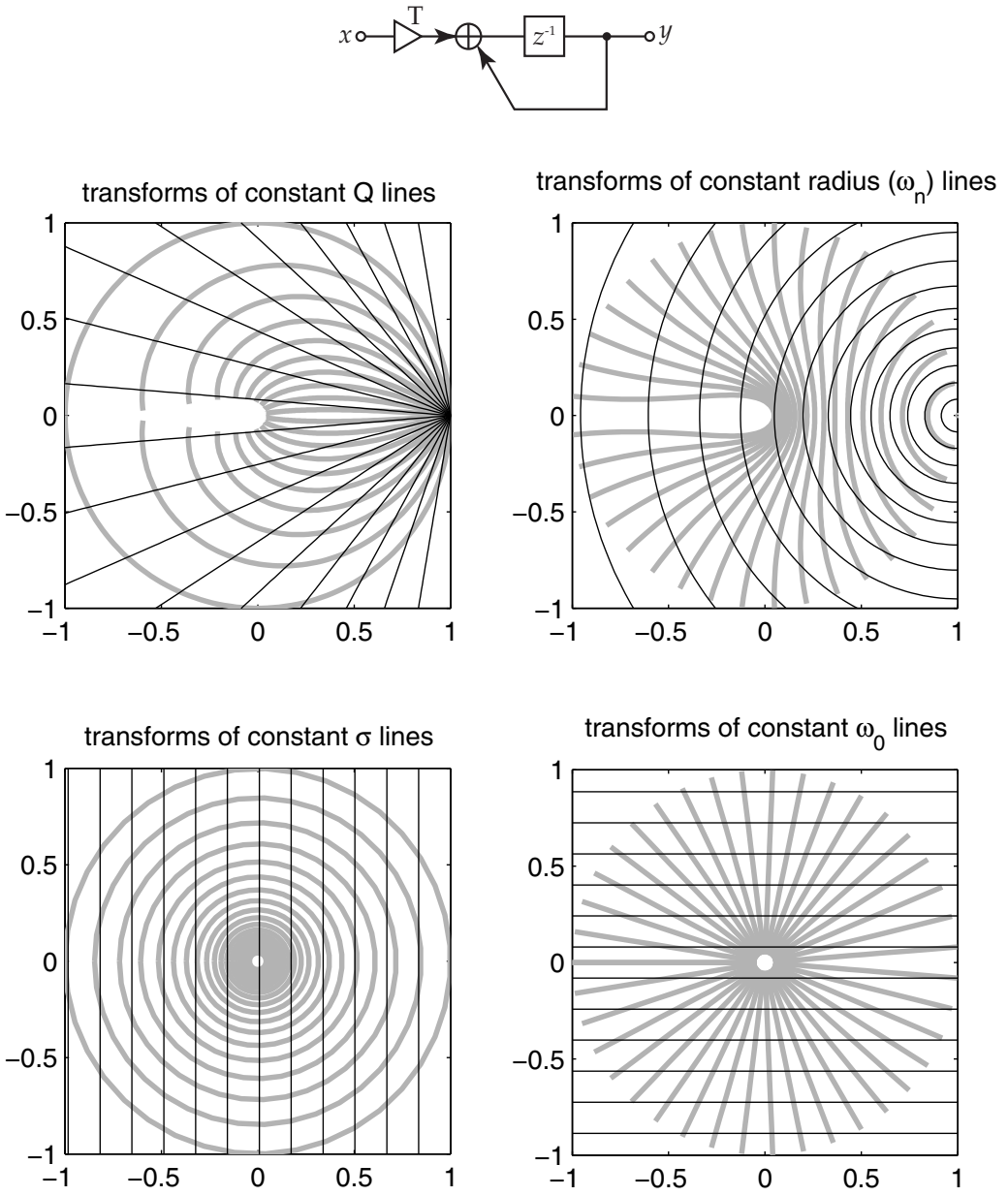


Figure 1.5: Top: Forward-Difference integrator. Bottom: Dark: Forward-Difference transforms of various constant-parameter contours into the z plane from the s plane. Light: the "ideal" $z = e^{sT}$ transforms of the contours.

Bilinear Transform

The bilinear transform, also known for some time in the Control-Systems community as "Tustin's Method", can be derived from the Trapezoidal Rule of numerical integration. The trapezoidal rule represents either a compromise between the backward- and forward-difference methods, or a slightly higher-order approximation than either method, depending on one's point of view. It approximates the curve with a set of trapezoids, which touch the curve at both ends of their extent in time. As such, the running accumulation becomes:

$$\begin{aligned}
y(nT) &= y((n-1)T) + T\left[x((n-1)T) + \frac{1}{2}(x(nT) - x((n-1)T))\right] \\
&= y((n-1)T) + \frac{T}{2}\left[(x(nT) + x((n-1)T))\right] \\
&\overset{DT}{\Rightarrow} \\
y(n) &= y(n-1) + (T/2)[x(n) + x(n-1)] \\
Y(1-z^{-1}) &= X(T/2)(1+z^{-1}) \\
\frac{Y}{X} &= \frac{T(1+z^{-1})}{2(1-z^{-1})} = \frac{T}{2}\frac{z+1}{z-1}
\end{aligned}$$

Thus,

$$\begin{aligned}
\frac{1}{s} &\leftarrow \frac{T}{2}\frac{z+1}{z-1} \\
s &\leftarrow \frac{2}{T}\frac{z-1}{z+1} = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}
\end{aligned}$$

Looking at how the s plane maps into the z plane via the bilinear transform (Figure 1.6), we see that of these first methods, the bilinear transform is the only one which maps the *whole* stable region of the s plane into the *whole* stable region of the z plane. Like all three methods, it is a conformal map, so all contours (which are all either circles or straight lines) map to circles (or straight lines) in the z plane. It is of aesthetic interest to note the many different ways that sets of circles can be constructed. The bilinearly-transformed integrator (Top of Figure 1.6) is a one-pole-one-zero filter, with a zero on $z = -1$. Note that there is a delay-free path through this integrator, which will become important in later chapters of this thesis.

There are many well-known (and well-discussed) properties of the bilinear transform, such as how it warps frequencies and thus maps the whole frequency range into a single trip around the unit circle, but we will not look into them deeply here (the interested reader is referred to books such as [87], [196], [238], and [170]).

One interesting thing to note in Figure 1.6 is the similarities and differences between the bilinear transforms of the various contours. Of course, the region near $s = 0$ maps rather well onto the region near $z = 1$, much better than the other two transforms, but notice what happens to the high frequencies in the s plane. In the $z = e^{sT}$ transform, the high frequencies pull in towards $z = 0$,

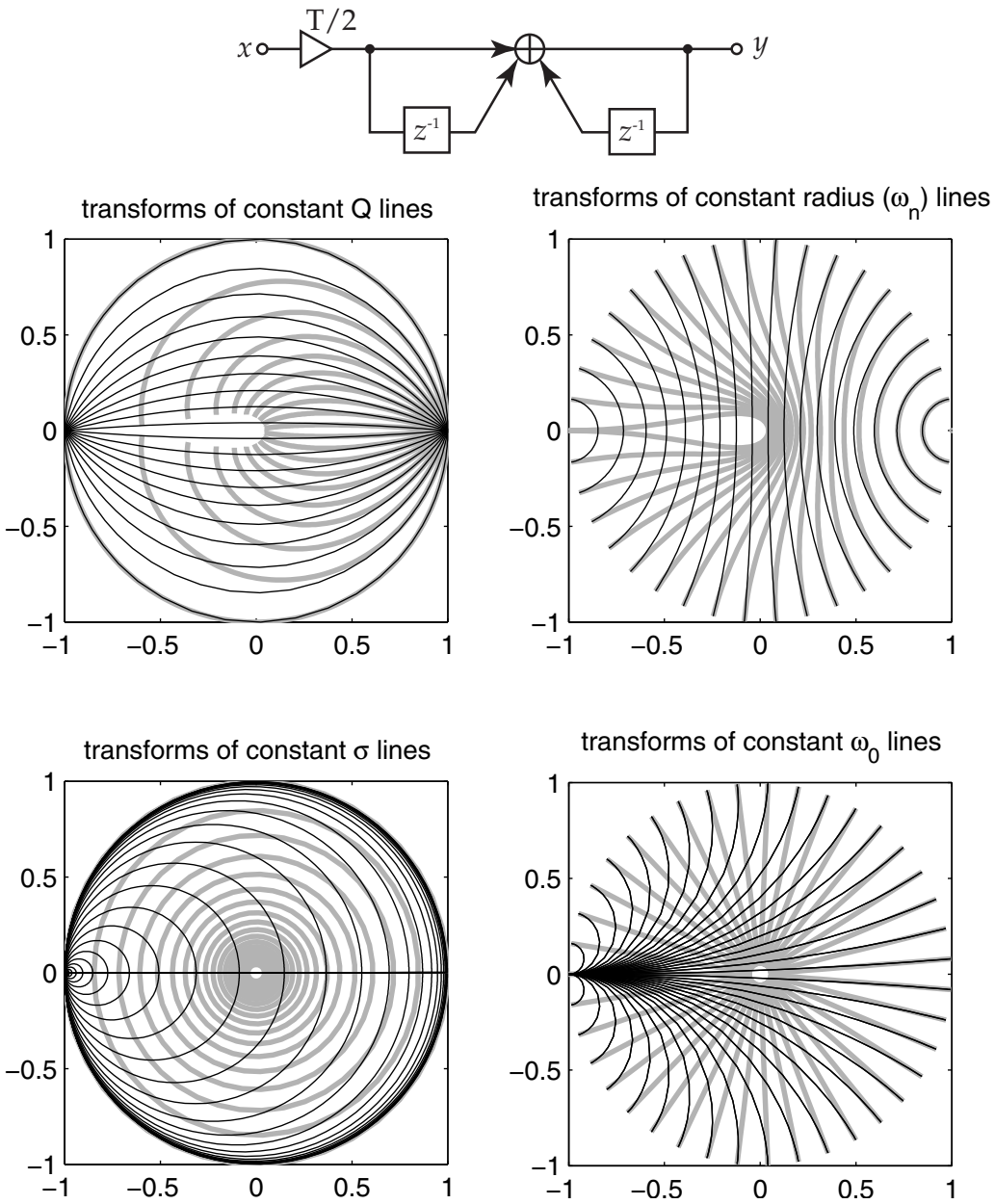


Figure 1.6: Top: Bilinear-Transform integrator (DF1). Bottom: Dark: Bilinear transforms of various constant-parameter contours into the z plane from the s plane. Light: the "ideal" $z = e^{sT}$ transforms of the contours.

whereas in the bilinear transform, they pull towards $z = -1$. In fact, with a bit of imagination, one can think of the bilinear-transformed shapes as having been formed by taking the $z = e^{sT}$ shapes and "pulling $z = 0$ over to $z = -1$", or vice versa. We will run across artifacts of this "pulling towards $z = -1$" in later chapters, especially in noting how various filters' Qs tend to drift too high at the top end of the f_c range for many filters. This is mostly due to the fact that the constant-Q trajectories for the $z = e^{sT}$ transform "pull inwards" near the negative real axis, and with almost all the design techniques, the poles do not exhibit that pulling-in behavior.

Psenicka has written a series of papers ([210] [209] [144]) noting how certain variable transformations (such as lowpass-to-highpass, lowpass-to-bandpass, and the bilinear transform) can be represented as matrix operations on arrays consisting of the numerator and denominator coefficients, where the matrices are constructed using the pascal triangle (i.e.,binomial coefficients). Konopacki ([144]) also worked on an extension of this idea.

1.3.2 Common methods not derived from numerical integration

Two other standard methods are in common use, at least in the signal-processing and control-systems communities: Pole-zero Mapping, and the Impulse-Invariant Transform.

Pole-Zero Mapping

This method attempts to use the $z = e^{sT}$ transform by simply creating a rational discrete-time system whose poles and zeros are at the transformed locations (as opposed to actually performing a transform on s). The method is basically a couple rules of thumb [87]:

- Take all s-plane poles and make z-plane poles at locations defined by $z = e^{sT}$.
- Take all finite s-plane zeros and make z-plane zeros at locations defined by $z = e^{sT}$.
- If there are N zeros at infinity in the s plane, make $N - 1$ z-plane zeros at $z = -1$.
- If there were any zeros at infinity in the s plane, make one z-plane zero at infinity. This tends to give the final result no delay-free paths.
- Match the gain at some desired frequency, typically DC.

There is no need for a figure showing the pole mappings, since they are by definition mapped the same as the $z = e^{sT}$ transform. However, whereas a theoretical $z = e^{sT}$ mapping would map the behavior of a whole system, pole-zero mapping only maps fixed poles, such that a system such as a root-locus-based filter, while having correctly-mapped open-loop poles and zeros, is not guaranteed to have intermediate (closed-loop) poles which follow the $z = e^{sT}$ mapping. In other words, if $\mathbf{S}_0$ is the the solution set of $D(s) + kN(s) = 0$ (i.e. the root locus in the s plane), then if $D'(z)$ and $N'(z)$ are the transformed polynomials, then

$$\mathbf{Z}_0 \neq e^{\mathbf{S}_0 T} \tag{1.20}$$

where $\mathbf{Z}_0$ is the solution set of $D'(z) + kN'(z) = 0$, the root locus of the transformed system. This can be seen most easily for systems which have asymptotes in the s plane: The $z = e^{sT}$ transform of an asymptote will be a curve that approaches a logarithmic spiral.[15] On the other hand, the transformed system will have only one asymptote (along the real axis), as all the rest of the infinite open-loop zeros have been mapped to $z = -1$, hence the "equivalent tracks" to the asymptotes will end up at $z = -1$, not on the logarithmic spirals. An example of this is shown in Figure 1.7.

Note that all the transforms which perform a rational $s \leftarrow f(z)$ mapping (backward difference, forward difference, bilinear) have the property that their loci do map, such that, in the terminology

[15] Either heading down to $z = 0$, out to $z \to \infty$, or approaching a circle centered on $z = 0$

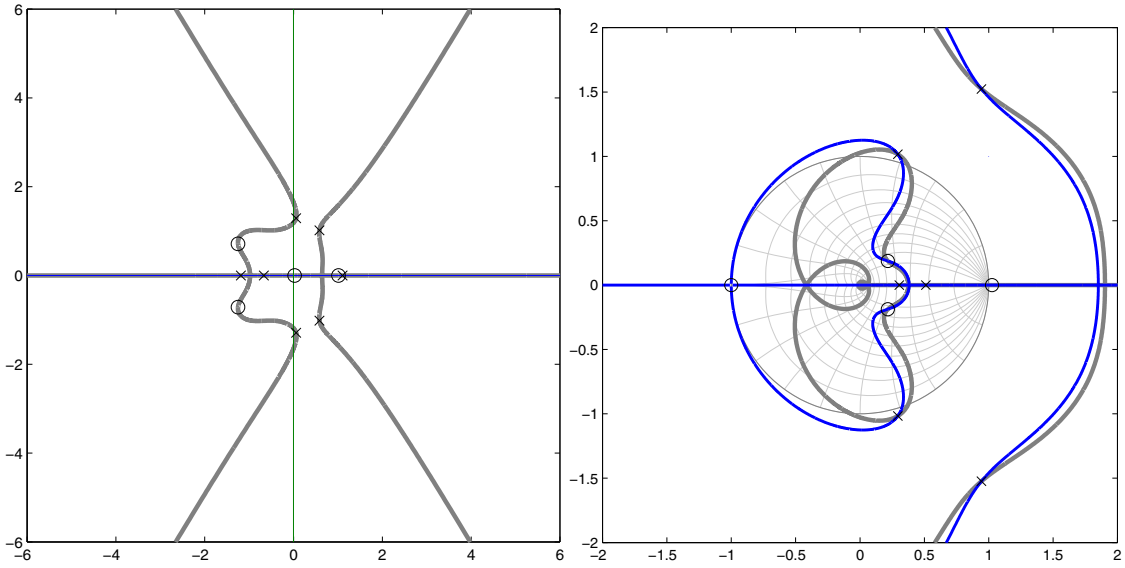


Figure 1.7: Root Locus of a pole/zero-mapped system is not the same as the $z = e^{sT}$ map of the original system's root locus. Left: Locus of a continuous-time system (asymptotes on multiples of $\pi/3$). Right: thick lines: $z = e^{sT}$ map of the original locus; thin lines: actual root locus of the mapped system. ($T = 1$ in this example.)

of the last example, $\mathbf{Z}_0 = f(\mathbf{S}_0)$. In other words, for those transforms, the root locus of a transformed system is the transform of the root locus of the original system.

An interesting thing to note about the pole/zero mapping transform is that if a multi-integrator system is transformed (i.e.,with all infinite zeros), then the result will have a mixture of zeros at $z = -1$ and at $z \to \infty$. We will see such mixed-zero patterns quite a few times later in this thesis, though not necessarily due to pole-zero mapped designs.

Impulse-Invariant Transform

This method[196] is not explicitly about mapping poles and zeros (though we will see it can be expressed in those terms). Instead its goal is to match the impulse response of the system at the sampling instants. In particular, the goal is to sample the continuous-time impulse response and create a discrete-time system with that impulse response. Now, since most continuous-time systems have poles (and hence have responses which theoretically extend infinitely in time), we will ignore the concept of simply windowing the response and assigning it to a large FIR filter; instead we break the impulse response up into its basic parts (i.e.,perform a partial-fraction expansion) and create equivalents in discrete time (here we use the terminology of the derivation in [196]):

Say the partial-fraction expansion of the continuous-time system $H_c(s)$ is:

$$H_c(s) = \sum_{k=1}^{N} \frac{A_k}{s - s_k} \tag{1.21}$$

The causal impulse response is thus:

$$h_c(t) = u(t) \left(\sum_{k=1}^{N} A_k e^{s_k t} \right) \tag{1.22}$$

where $u(t)$ is the unit step function. When sampled at times nT, this becomes:

$$h[n] = T h_c(nT) = u[n] \left(\sum_{k=1}^{N} T A_k e^{s_k T n} \right) \tag{1.23}$$

Which corresponds to

$$H(z) = \sum_{k=1}^{N} \frac{T A_k}{1 - e^{s_k T} z^{-1}} \tag{1.24}$$

The above assumes non-repeated poles, but can be extended to repeated poles straightforwardly.

Note that poles map essentially the same way as in pole-zero mapping, but zeros and gains do not. It is up to the specifics of the partial-fraction expansion to determine where the zeros will end up.

Recently, Leland B. Jackson [126] and (separately) Wolfgang Mecklenbrauker [166] realized that the traditional definition of the transform (above) has a problem if $h_+(0) \neq 0$: the traditional definition assumed that $Step_+(0) = 1$, but instead it should be taken to be $1/2$. As such, in cases where there is a discontinuity in $h(t)$ at zero, then the traditional formulation is offset by a term on the order of $T A_k \delta[n]$). Being scaled by T, which is normally very small, is the assumed reason for why this hasn't been noticed before (and because a large percentage of design responses are not discontinuous at $t = 0$). Thus, the equation for $H(z)$ above should have another term:

$$H(z) = \sum_{k=1}^{N} \frac{T A_k}{1 - e^{s_k T} z^{-1}} - \frac{T}{2} \sum_{k=1}^{N} A_k \tag{1.25}$$

Jackson notes that the new term isn't as strange as it might seem, since due to the Laplace initial-value theorem to $H_c(s)$,

$$h_c(0^+) = \sum_{k=1}^{N} A_k \tag{1.26}$$

So it will be zero if h_c is continuous at zero.

1.3.3 Other Methods

Delta Operator

This method, which involves translating to a domain other than the z domain (but which can be related to it), has garnered quite a bit of interest recently. The primary references are the paper by by Middleton and Goodwin in 1986 [167], two books they wrote which use this concept as the central topic ([82],[168]), and a nice overview for the Proceedings of the IEEE [95].

The main concept is that in continuous time, the primary dynamical device is the integrator (or its inverse, the differentiator), but in discrete-time it is the unit delay. This is considered inelegant, and instead, a new discrete-time domain is proposed which also has as its primary dynamical device the integrator (in discrete time). The delta operator is the discrete-time approximation of the derivative:

$$\delta x[n] = \frac{x[n+1] - x[n]}{T} \tag{1.27}$$

and thus, as differential equations are solved with integrators, δ-operator equations are solved with the δ^{-1} operator

$$\delta^{-1} x = \sum_{m=0}^{n} T x[m] \tag{1.28}$$

i.e.,a discrete-time integration. A drawback of the z-domain concept is that as the sampling rate gets very high compared to the dynamics of the system, all of the important behavior gets concentrated very close to $z = 1$ in the z plane. As such, as the sample rate gets large, system coefficients end up approaching binomial coefficients, regardless of the actual dynamics being implemented. Further, as polynomial coefficient sensitivity grows as the inverse of the distance between roots, the system gets more and more sensitive to coefficient errors (quantization).

With the delta transform, the stability region in the discrete-time transform domain (the "γ" plane) is a circle whose right edge touches $\gamma = 0$, is centered on $\gamma = -1/T$ (where T is the sample period), and has a radius of $1/T$. As such, the stability region's size and location is dependent on the sample rate: it grows as the sample rate grows, such that in the limit of $T \to 0$ (i.e. $f_s \to \infty$), it becomes the right-half plane, and hence matches up with the s plane. Thus, delta-operator system coefficients, instead of approaching binomial coefficients at large sample rates, approach the continuous-time coefficients. As a consequence, coefficient sensitivity and quantization issues are significantly reduced for high sample rates. In fact, some of the papers demonstrate very nice dynamical behaviors from systems with rather small word sizes. Thus, this technique has been suggested for very-high-sample-rate work ([94] [286]), even being applied to DSP directly at full sample rate in 1-bit sigma-delta systems, where the sample rate is several orders of magnitude higher than the dynamics of typical audio processing.

We should note that as the commercial audio community continues to increase the sample rates

in common use (192kHz is now common), the arguments for the shortcomings of z-domain processing and the virtues of δ-operator design become more and more applicable.

As noted, the basic operation in delta-operator systems is the integrator. We will see later that the Chamberlin and Dutilleux state-variable filters are built the same way: with integrators as their basic operation. Even Moog-style filters appear similar, effectively using first-order (one-pole) systems as their building blocks (which is not terribly far from using integrators). This thesis does not, unfortunately, explore the relationship between delta-operator design and these filters, because the author encountered it too late to shape the core research, but it does appear that much of the nice numerical properties of the state-variable filters can be related to properties for delta-operator systems.[16]

Wave-Digital Filters

This discretization technique, introduced by Fettweis in a series of papers ([78] [79] [80] [81]), approaches the discretization of passive systems in a manner reminiscent of physical modeling (though Fettweis' papers predated the discussion of physical modeling in music synthesis). The basic concept (liberally interpreted) is that a passive circuit can be modeled as voltage and/or current waves travelling (and scattering) along infinitessimal waveguides between basic dynamic elements (inductor, capacitor, etc) connected into the circuit via multi-port junctions. By applying the bilinear transform for the discretization of the elements (and choosing a particular value for the bilinea transform scaling constant), the method is able to avoid delay-free loops in circuit implementations. Using appropriately normalized junction implementations, such networks can not only implement models of passive networks, but can stay passive under time variation of network parameters (as long as such changes were physically appropriate). Stefan Bilbao has recently related wave-digital filter design to other space/time discretization methodologies (basic physical modeling, finite-element analysis, etc) ([16] [17]).

For the purposes of this thesis, wave-digital concepts do not necessarily have an obvious application, as the filters and oscillators under examination are not passive. Thus research for this thesis did not look too deeply into that subject. However, the technique appears particularly useful as a physically-informed discretization of passive circuits (or lumped mass/spring physics models), especially time-varying systems.

Other Notes

Tseng [270] designed a Simpson-rule integrator which is internally upsampled by a factor of two, so includes a fractional-delay filter to preform the upsampling. As a general rule, oversampling was avoided in this thesis, so this technique is mainly an interesting curiosity.

[16] It also bodes well for the numerical properties of Moog-style systems, though such properties are not explored in this thesis.

We should note, however, that while exploring the properties of the bilinear transform, it was noted that higher-order relatives do exist, with interesting properties. For example, the following transformation:

$$z \leftarrow \left(\frac{s+1}{s-1}\right)^N \tag{1.29}$$

If we plot this for N=1,2,3,4, on top of the $z = e^{sT}$ curves, using the constant-Q curves as reference (Figure 1.8), We see that for the "first wrap around", the curves approach the ideal $z = e^{sT}$ curves

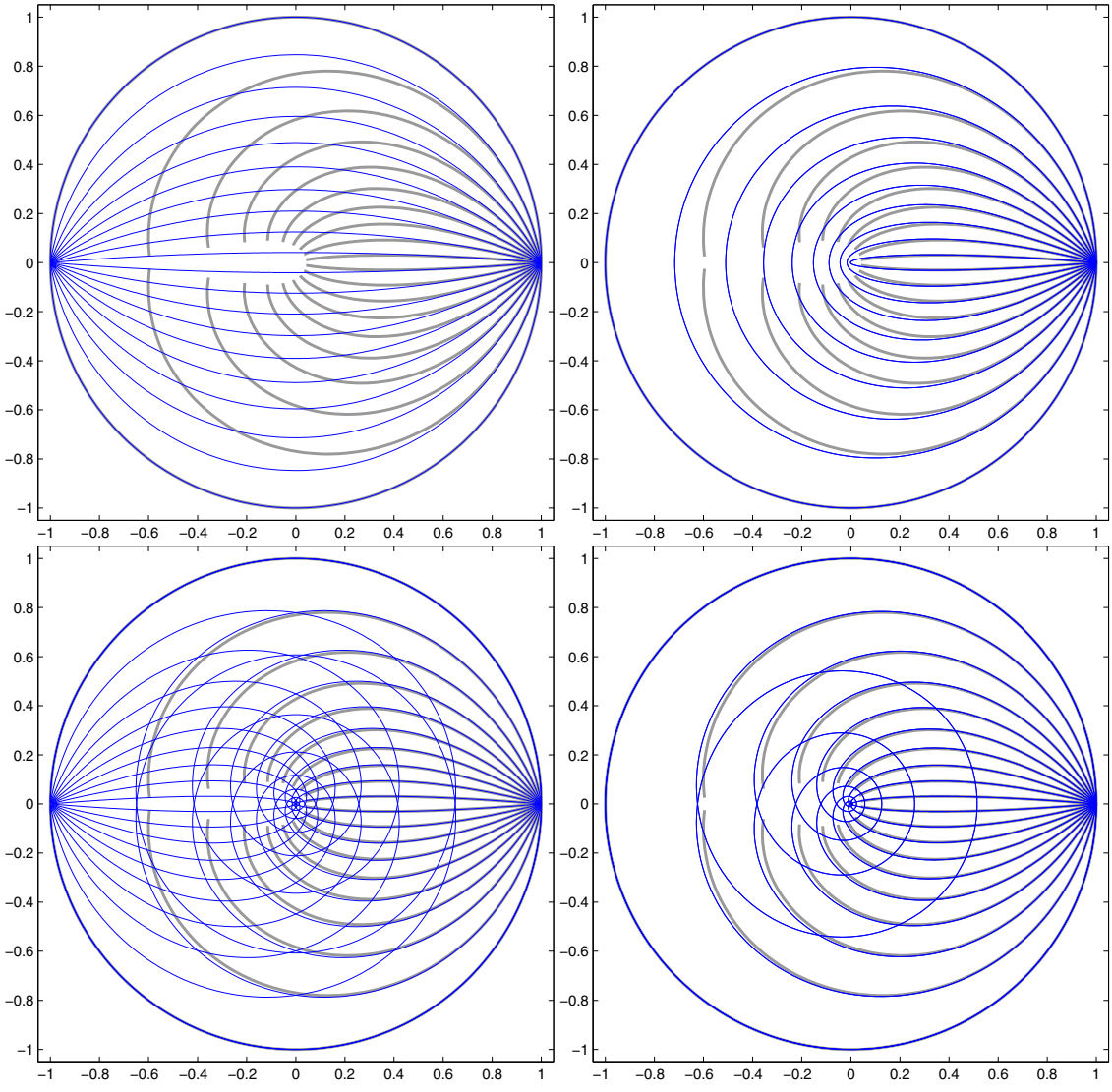


Figure 1.8: "Higher-Order Bilinear" transformations. Left to Right, Top to Bottom: N=1, 2, 3, 4. Note how the "first wrap" curves get closer and closer to the $z = e^{sT}$ curves.

progressively closer.

This leads us to note that this is a good approximation of e^x. In comparison to the well known limit:

$$\lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n \to e^x \tag{1.30}$$

The following is also a limit:

$$\lim_{n\to\infty}\left(\frac{1+\frac{x}{2n}}{1-\frac{x}{2n}}\right)^n \to e^x \tag{1.31}$$

Numerical experiments have shown (for example, Figure 1.9), that it converges much faster (with the caveat that the rational approximation does have poles, which means the approximation will only be particularly good for x smaller than $2n$, but within that region, it is much better than the other limit). On reflection, one can make comparisons between these two approximations and between the forward-difference and bilinear transforms.

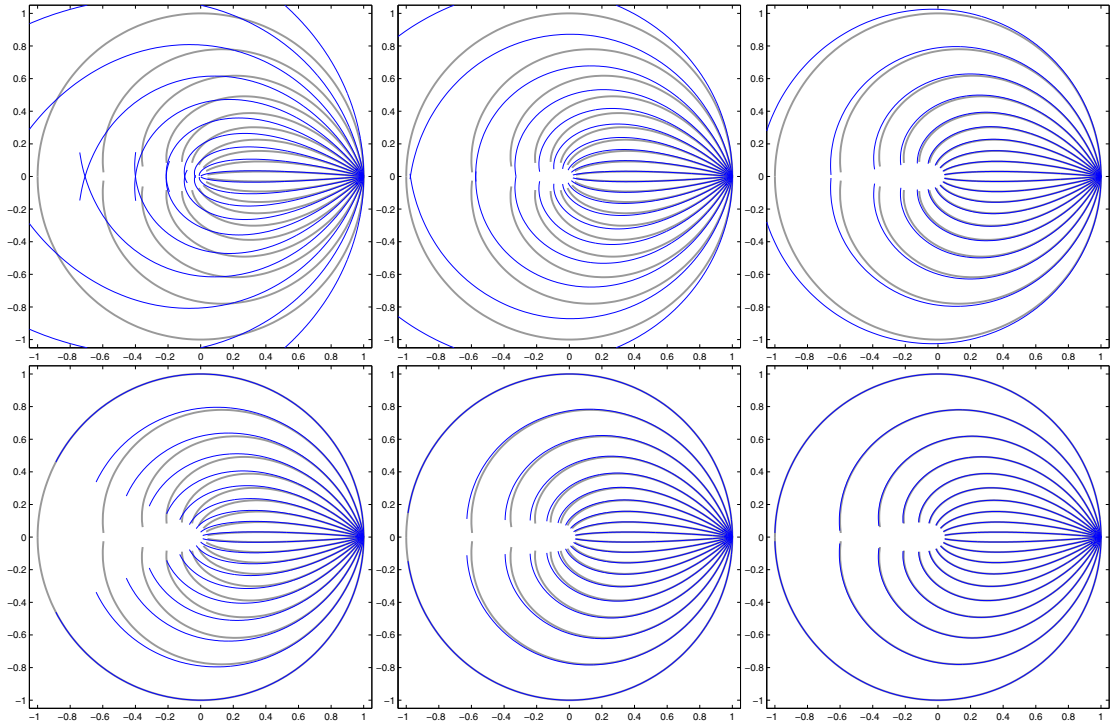


Figure 1.9: Convergence of two approximations of e^x: Top row: $(1 - x/n)^n$ (n=4, 10, 50). Bottom Row: $((1 - x/2n)/(1 + x/2n))^n$ (n=2, 4, 10).

Of course, these limits are mainly of theoretical use (as there are much better ways to approximate the exponential function), but it does give one food for thought when contemplating the discretizations.

Is this rational approximation directly useful in filter discretization? Not directly. The reason for this is that we would typically implement such a transformation by replacing s with some function of z. The above approximation would thus need to be inverted:

$$s \leftarrow \frac{\sqrt[N]{z}+1}{\sqrt[N]{z}-1} \tag{1.32}$$

Which leaves us with rational functions in *roots* of z (rather than in z directly), which are not directly implementable.

There is an intuitive interpretation of a filter in the roots of z, however: oversampling. A filter which consists of a rational function in $\sqrt[N]{z}$ might be thought of as internally oversampled by a factor of N. In essence: an oversampled filter can better approximate the behavior of $z = e^{sT}$ in the bottom fraction of its frequency range (which is where all the discretizations work well).

1.4 Review of Existing Variable Filter Methods

Filters in Virtual Analog can be seen as a subset of the larger set of Variable Filters. There is quite a wide field of research in that area of filter design. As expected, most variable-filter design tries to tackle the problem of designing a filter that can be usefully yet efficiently (and probably accurately) varied while running (in real-time, usually). As such, the most obvious situation of simply re-designing the filter on the fly is normally dismissed off-hand. On the other hand, some areas of variable-filter design (including this thesis) approach that problem by attempting to create a filter form for which "re-design" is trivially inexpensive.

We will look at the various general methods that have been described over the years, categorizing them as either FIR-only, IIR-only, or applicable to both FIR and IIR filter design. Afterward, we will look specifically as research into variable musical filters, in particular the two filter classes which this thesis is about: the state-variable filter and the Moog-style four-pole filter.

1.4.1 FIR

As the number of coefficients is typically large in FIR filter design, it is not immediately obvious to the general user that FIR variable-filter designs might exist.

Jarske et al. described in [130] a family of linear-phase FIR bandpass filters whose coefficients are easily calculated as a window multiplied by a sinusoid of variable frequency. The concept of this design being that complex filter-design algorithms need not be run for parameter changes, rather a very straightforward function could be used to recalculate all the coefficients. Still, all the coefficients do have to recalculated in this technique.

Yoshida and others [298] propose a method that I am calling "Multi-Dimensional Slice", whereby a multi-dimensional FIR filter is designed, whose higher-dimensional frequency response is a shape

(like a diamond or a tilted shape) whose slices along certain dimensions are various desired states of the filter variation of a 1-D filter. Variation (coef recalculation) occurs by moving the slice and recomputing the 1-D FIR coefficients.

Weighted Least-Squares [257] is an FIR variable-filter design method whereby FIR filters for a gridding of parameter space are designed, and then polynomials in the parameters are fitted to each coefficient. Again, all coefficients must be recalculated for variation (but we will see methods in a later section which can use FIR filters which do not involve any sort of coefficient recalculations). This method has gathered interest recently for use as a variable fractional-delay filtering method ([159] and [61], for example).

Note that several techniques which are commomly used with FIR designs (such as SVD/Vector-Array Decomposition) actually apply to IIR filters as well and are thus discussed later in Section 1.4.3.

1.4.2 IIR

Some IIR filters have few enough coefficients that brute-force (or table-assisted) coefficient recalculation can be viable in realtime. Robert Bristow-Johnson [28] and Sophocles Orfanidis [198] give good overviews of 2nd-order EQ-section design, in particular. They reference several well-known papers on EQ filter design, such as Dana Massie's exploration of normalized-ladder EQ design [162]. Methods after Regalia and Mitra [217], [170], which use internal allpass filters with variable phase response and variable combinations of the allpasses and the dry signals to perform inexpensive variation ([104], [301]). An early wave-digital EQ-design technique was presented in 1976 by M. Swamy and K. Thyagarajan [256]. Chris Hanna did further exploration of real-time EQ design and variation in 1994 [103]

Since it is known that direct-form filters do not necessarily interpolate with "nice" intermediate filters shapes, Ding and Rossum at EMU [64] described an intermediate filter representation which could be transformed to/from implementation coefficients, which linearly interpolated with nice intermediate shapes.

Similarly, lattice filters have useful interpolation properties, and Jean Laroche gave a good overview of that in [151].

Digital state-variable filters and digital versions of the Moog voltage-controlled lowpass are in this category, but will be discussed separately later in this chapter. As used in this thesis, they belong in the category of variable filters that are constructed to have coefficient design equations which are as cheaply-implementable as possible, and thus belong to the "brute-force re-design" class of variable filters.

1.4.3 FIR or IIR

Spectrum Warping

One of the earliest digital variable-filter techniques was Spectrum-Warping, first described by Schussler and Winkelkemper in 1970 [229], based on work by Constantinides a few years earlier [44],[45]. The basic method replaces the unit delay with a first-order allpass filter, which has the effect of warping the effective spectrum for the filter in which the delays were replaced, and hence allowing the frequency locations of the filter's features to be smoothly varied via the allpass coefficient. The method has amassed quite a bit of literature, and it is to this day in extremely common use.

The basic technique will not preserve linear phase in an FIR filter, so Oppenheim proposed a variant [197] which would preserve linear phase. Crochiere and Rabiner analyzed this variant in [52], and later, Dutta, Roy and Ahuja proposed a different linear-phase-preserving variant [70].

It was obvious early on that replacing a delay with an allpass in a loop would result in a delay-free loop, and hence an unimplementable system. In non-variable design, that is not a major problem, as the filter coefficient can just be recomputed to absorb the loop. However, in variable-filter design, such recomputation is to be avoided if at all possible. In 1979, Johnson proposed a simplified (yet still involved) coefficient recomputation [132], and later Steiglitz came up with a simpler recomputation [245].

James Moorer gave a nice overview of the usage of spectrum warping in [176], and later Matti Karjalainen gave an updated overview [137].

Murakoshi and others took spectrum warping and combined it with some of the math of [171] to fill in some more usage cases [185], and later applied it to complex-coefficient variable-filter design [186].

Spectrum warping also has application outside of variable filter design. For example, the math is very similar to the classical methods of taking a prototype filter and performing s (or z) substitutions to create filters of desired width, or highpass and bandpass filters [46]. Julius Smith in [234] uses warping to simply certain FIR and IIR filter design problems. Leland Jackson in [127] similarly uses the technique to simply an FIR filter design, while at the same time using specific warping coefficients that can be implemented very cheaply (i.e., with bit shifts and adds rather than full multiplication).

Multi-dimensional ("MD") Coefficient Fitting

This is a general class of design methods, of which the Weighted-Least Squares (WLS) design method mentioned earlier ([257]) is a member. The basic concept is:

1. Do numerical filter designs for a (usually dense) grid of the parameter space.

2. For each coefficient, fit a multivariable function (often polynomial) to the gridded coefficient data. The fitting model is assumed to be inexpensive to compute at runtime, so that changes in parameters require little computation to update the coefficients.

Typical applications of this concept are shown in [299] and [60]. The concept is more general than the WLS method in that the design method is not restricted to weighted-least squares fitting, and it can be applied to IIR filters as well. This technique can be applied to any type of base filter implementation and any type of coefficient fitting functions, though the choice of filter type and fitting model would obviously effect the ease of achieving good fits at efficient function orders, and thus effect quality (aside from any quality issues the base filter type may have separately). Hence, one should expect that some filter forms may work better with this technique than others.

Stephen Boyd has recently suggested [24] that rational-function models should be able to be fit very tightly (using convex optimization techniques) for relatively low-order fitting functions, and hence achieve very inexpensive parameter variation.

When inexpensive fits can actually be achieved for inexpensive filters (i.e. IIR filters), then methods of this type should be able to create quite useful functions.

Vector-Array Decomposition

This method works by sampling the (N+1)-dimensional desired frequency response (the frequency dimension + N control-parameter dimensions) and performing an array decomposition, such as Singular-Value Decomposition (SVD), on that data. This results in a set of 1-dimensional vectors, whose outer-products sum to the desired response. The 1-D vectors along the frequency dimension are simply filters. The other vectors are then approximated by polynomials, and the resulting variable filter implemented as a sum of the component filters, weighted by the various polynomials on the control parameters.

Note that this differs from the previous technique (MD coefficients) in that the polynomials are fitted to the results of an array decomposition, and represent weights in the summation of a set of filters, whereas in the previous trechnique, the polynomials represented filter coefficients directly. Also, this method, as it currently stands, uses only 1-D polynomials, whereas the prevous technique uses multivariable fitting functions.

This technique was originally implemented mainly using FIR filters, but quickly extended to general IIR filters. Deng has a series of papers on the method, starting in 1994 [63], and most recently in 2005 has presented a complex-valued version [62].

As a slight variation, Pun and others [211] describe an IIR system using a common denominator for the decomposed FIR filters, and also discusses a piecewise-polynomial extension. These authors also note the similarity between this structure and the Farrow structure (next section).

Fractional-Delay and SRC

A large area of research in the last 10-20 years has been the problem of variable fractional-delay filter implementation. Such filters have application in a rather wide range of fields. For example, time-varying delay effects such as chorus, flange, pitch-shift and even reverb use variable fractional delays to smoothly vary their delay lengths. Similarly, waveguide-based music synthesis techniques use them for precisely tuning and varying waveguide lengths. Sample-rate-conversion can be algebraically rearranged into a time-varying fractional-delay filter. Such a filter can thus also be considered as table interpolation, and as such has application in sample-based music synthesizers. All of these use some form or variation on the concept of fractional-delay filtering, and most of them require that the delay fraction be varied on a per-sample basis, such that cost is a definite issue.

Overviews of the general problem and various design approaches have been given by Timo Laakso et al. [149], and by Jon Dattorro [58].

The most obvious methods are various FIR interpolations, linear being the most common, but higher-order polynomials are not uncommon (most particularly various Lagrange interpolations) [149]. David Jaffe and Julius Smith compared linear interpolation and the use of a first-order allpass filter to get a variable fractional delay in [128], and discuss how the allpass interpolation can be used in systems where a delay must sweep smoothly over a multi-sample range.

Later, Vesa Välimäki discussed how to hide transients from more complicated discontinuous IIR/allpass coefficient changes [278], based on an optimization of [300], and later followed up in [274]. The basic concepts being the online initialization of a side filter with the new coefficients and the same input signal, then swapping states and coefficients (or switching over to the side filter) when its startup transient has decayed sufficiently.

Recently, Rauhala and Välimäki have applied variable-delay-filter concepts to the design and implementation of dispersion filters for piano-string modeling [214].

In 2001 Makundi, Laakso, and Välimäki took a technique from JP Thiran [267] for designing maximally-flat group-delay filters and created a variable version of it [161].

Much sample-rate-converter work doesn't apply back to the general field of variable-fractional-delays, as the interpolation/delay is often intertwined too deeply into the conversion algorithm (such as with most polyphase methods).[17] However, some techniques do apply to fractional-delay filtering:

Smith and Gosset described in 1984 an FIR arbitrary-fractional-sample delay where the coefficients were read out of a windowed-sinc lookup-table using a particular algorithm to determine which locations to lookup into the table [242]. Later, Paul Beckmann and the author created a variant [13] which, by constraining the rate conversion ratio to be within a narrow range of an ideal ratio (i.e.,limiting the set of fractional delays to a certain subset), allowed an optimization to the

[17] Though Ikehara in 1990 presented a variable-filter method [124] based on the polyphase decimation filter form.

table lookup, yet still allowed the converter to be used in an asynchronous fashion.

A sample-rate conversion structure that has gathered a bit of a following is the Farrow Structure [77], whereby the concept of interpolating between a set of fixed fractional-delay filters is transposed into a single interpolation polynomial on a signal, but where the polynomial coefficients are filters. Early work in this area lacked a good interpretation in the frequency domain and hence was limited in SNR (due to difficulty in understanding how to design the filters optimally for SNR), but recently Vesma and Saramaki have worked out much of that issue [284], [283] and higher-quality designs should be forthcoming. In 2003, Johansson and Lowenborg presented a variation [131] on the Farrow structure which could allow different filter orders between subfilters. It was noted in [211] that VAD-style variable filters can look very similar to Farrow structures, and that there is probably a connection.

1.5 Musical Filters in Subtractive Synthesis

Musical filtering, and subtractive synthesis in particular, originally started by using filters as available from other areas (radio, etc.), however, the advent of voltage control gave the field its own special set of filters. Whereas typical filter design had centered around fixed filters of the classical types (Butterworth, Chebychev, Elliptic) in lowpass, highpass and bandpass configurations, or around the parametric-EQ designs (peak/notch/shelf with tweakable gain, center, and Q), subtractive synthesis had two particularly useful designs which were of particular fame: the state-variable filter (which, it must be admitted, did not originate within the field of electronic music synthesis, but it was well-used in it), and the four-pole resonant lowpass originally popularized by the Moog implementation (which I will usually call the "Moog-style filter"). Both had modulatable center/corner frequency, which was quite useful in generating interesting effects, and either modulatable or easily-controllable resonance on the corner (or center) frequency. For a time, Moog was able to protect his patent [204] of the special ladder structure [173] which was the hallmark of the filter, but later variants became common (such as diode-based ladders in synths such as the Roland TB303).

In his AES paper on musical filters, Dattorro [57] has a discussion of the requirements of musical filters, as opposed to requirements typically seen elsewhere in engineering.

1.5.1 History: Continuous-time Analog Filters

State-Variable Filters

The analog state-variable filter was introduced by Kerwin, Huelsman, and Newcomb in 1967 [140], and a few later authors refer to the design as the "KHN" filter. Further discussion of the parametric-EQ uses of the filter are given by Frey [89] and Bonello [21].

The term "state-variable filter" can be attributed to the fact that the filter is directly in the form one would use to solve a differential equation using an analog computer: a series of integrators with each integrator's output fed back and mixed with the input through various calculations.

We owe much of the current interest in the State-Variable filter to its inclusion in Hal Chamberlin's classic *Musical Applications of Microprocessors*[37], which also gives us one of the first (and the most influential) discrete-time implementations of the filter.

On a side note, Chidlaw described a fourth-order extension of the state-variable filter concept for musical use in [41]. It has a possible drawback of having two peaks at high Q, though it is described as a central feature (the split distance is directly controllable). It is a simple and elegant design, though, and might have its uses.

Moog-Style Filters

The classic four-pole transistor ladder circuit was first published (after getting the patent submitted) as an AES talk by Robert Moog in 1965 [173]. The previous year, he had written an AES journal article on the use of voltage control in synthesis [172], but had not described this filter in that article.

We next find published work on the filter in a series of articles in Bernie Hutchin's Electronotes publication. In a series of three (non-contiguous) articles ([118], [113], and [114]), Hutchins describes the filter and works out explanations for its behavior. These are the first articles to analyze the Moog filter in a root-locus-like way, though root-locus itself is not discussed. Later, in four more papers, Hutchins and Bjorkman explore an extension to the Moog-style filter to any number of poles, which Hutchins called "Polygon Filters," based on the pattern of the poles in the s plane ([117], [19], [115] and [116]).

1.5.2 Digital Musical Filters

State-Variable Filter (SVF)

As in the history of analog music, digital music (mainly academic music) first started with the classical filters, when filters were used at all. However, due to the influence of Chamberlin's book, the state-variable filter, particularly in his form, became quite pervasive. Chamberlin presents the analog state-variable filter, but quickly moves to the digital form, showing a block diagram, giving computer code for implementing it, and describing how to calculate its control parameters. As mentioned earlier, one of the key benefits of this filter is that the equations for calculating the coefficients from the parameters are very simple: the Q coefficient is simply $1/Q$, and the frequency coefficient is $2\sin(\pi f_c/f_s)$. Since sine is effectively a straight line near zero, one could ignore the sine if exact tuning wasn't necessary, or use a simple polynomial to get pretty close at most musical frequencies.

Later, Bernie Hutchins discussed state-variable-filter digitization in Electronotes [121], using both backward-difference and bilinear methods. However, his block diagrams contain delay-free loops which are not discussed in the text, so it is unclear whether his actual implementations included unexpected loop delays or if he might have used the derived transfer functions in other implementable filter topologies.

Dutilleux, in his thesis on efficiently-variable synthesis ([69] and an AES preprint of the same subject, in English: [68]), re-derived the state-variable filter, and explored more of its capabilities and behaviors. His derivation gave a different-looking filter than Chamberlin's, but we will see later that it was actually very close, and it uses the same design equations. Interestingly, he attempted a bilinear transform of the filter but decided it wasn't worth figuring out the delay-free-loop problem. He did venture a small modification to the standard filter to give a similar result to doing the bilinear transform (placing a one-zero filter in line with the state-variable filter). He also had some good discussion of amplitude scaling issues (which this thesis will not look into deeply, so the reader is recommended to that paper for details).

The next major discussion of the state-variable filter was in Jon Dattorro's 1997 AES paper on musical effects and filters [57]. In that paper, he attempted to re-derive Chamberlin's form purely from frequency-domain specifications (successfully, though with a few series approximations here and there, as happens with most digital filter designs). He noted that by that time, the state-variable filter was in wide use in the industry and was well-known for good noise and quantization behaviors (which he then analyzed in great detail).

It is of interest that there are several 2^{nd}-order filter structures which are quite similar to the Chamberlin structure, for example, Yan and Mitra [294], the Gold and Rader Coupled Form [93], and a structure described by Agarwal and Burrus [4]. Similarly, we will note a few more in a later chapter of this thesis. We will be concentrating mainly on the SVF topology, but it would be very interesting to compare these to the SVF in a musical light.

Moog-style filters

In 1984, Bernie Hutchins discussed digitization of this filter type in Electronotes [121][120], going as far as to use the bilinear transform. However, he apparently forgot to note that such a transformed filter contains a delay-free loop (as will be discussed in a later chapter).[18]

By the mid 1990s, simple digitizations of the Moog-style filter had become common as sound-effect filters in various digital synths. Perry Cook has mentioned implementing a delayed backward-difference version of the filter [49] for MediaVision in the early 1990s, and similar anecdotes were heard from employees of other synthesizer manufacturers.

[18] Interestingly, Hutchins was more interested in the backward-difference discretizaion, considering the bilinear result too similar to the analog filter. Apparently, at the time, he was searching for "new" filter behaviors rather than faithful models of the analog filter.

In 1995 the author and Julius Smith analyzed the filter, and noted how directly the form resembled that of the feedback systems of classical control-systems design, and that root-locus would be a particularly applicable method for gaining intuition on the behaviors of the filter [249]. After trying the obvious discretizations, a compromise between the bilinear-transformed and backward-difference-transformed filter was found which could under some situations be used without the need for a table (or large polynomial) to separate the frequency and Q controls. Soon thereafter, Harvey Thornburg performed a series of experiments into the behaviors of various locations and numbers of nonlinearities in the filter loop [268]. The author went on an effective hiatus soon thereafter until recently, and during that time, quite a few virtual analog synthesizers came to market, demonstrating the market appeal of the concept and pushing the envelope in quality and efficiency:

Popular or influential Virtual Analog synthesizers (not a complete list, rather a sampling of notable synths, with assistance from `www.vintagesynth.com`[19])

- 1995: Clavia Nord Lead, the first well-known synth to specifically emulate a particular analog synthesizer (the MiniMoog).
- 1995: Korg Prophecy
- 1997: Yamaha AN1x
- 1997: Roland JP-8000
- 1997: Access Virus (and later versions)
- 1998: Novation SuperNova
- 1998: ReBirth (Software)
- 1998: Clavia Nord Modular
- 1998: BitHeadz Retro AS-1 (Software)
- 1999: Native Instruments Reaktor (Software), and a slew of later emulations of specific synths.
- 1999: Waldorf Q
- 2000: Reason (Software)
- 2001: Korg MS-2000
- 2003: Arturia CS-80V and Moog Modular V (Software)
- 2004: Alesis Ion and Micron
- 2004: CreamWare MiniMax

[19] And Sean Costello and Denis Labrecque

In 2004, Antti Huovilainen published a paper [110] on a fully nonlinear filter model derived purely from the circuit equations, a concept which continually expanding computing power had finally made viable for real-time implementation. That paper was followed by more papers expanding on the concepts and working back towards more efficient architectures, also bringing in recent work in bandlimited waveform synthesis ([111] [277]).

This thesis will expand on the ideas in [249], introducing one or two further tweaks on inexpensive Moog-style filter design, looking much more deeply into the concept of using root-locus to understand pole-variable filters, and adding the state-variable filter into the filters discussed and analyzed. Additionally, the work in another paper from the author [248], on bandlimited waveform generation, will be discussed. Finally, as part of looking into root locus, the author experimented with several algorithms for rendering root locus diagrams, and that work will be discussed in the first appendix. Other, less-related research topics are discussed in later appendices.

Caveats

Scope of filter designs. The quest for inexpensive variable filters can be thought of as following one of two different directions:

- Design a good filter and work on making the tuning transformations as inexpensive as possible. It is noted by Boyd [24] that extremely inexpensive yet tight rational-function fits can be made to coefficient values across multidimensional control spaces (such as (Q, f_c)) using straightforward convex optimization techniques. As such, extremely inexpensive filters may be built using these methods.

- Work on deriving a filter which implements the desired tuning inherently, such that there need not be any extra tuning transforms (or that they be as trivial as possible).

This thesis takes the latter course in the exploration of the design of Moog-style filters. The former course is expected to be equally valid and should be expected to produce useful designs as well.

Time variation and nonlinearity. The filter analysis methods employed in this thesis (root locus, plotting Q, etc.) are only valid for time-invariant systems. Thus, filter designs based on these analyses are only expected to satisfy their design goals during periods of relatively static usage (or "slow" variation). It is expected that these will be the most common usage modes. Still, the design goal of inexpensive parameter variation is still valid. First, it allows variation of moderate speed to be done smoothly (without zipper artifacts or other artifacts associated with updating coefficients sparsely in time). Second, it does not prohibit the use of the filter outside of the design region (i.e. with fast time variation) — the filter is still usable (inexpensively). The user must be warned, however, that in such situations, such filters are no longer expected to act "as designed", since the

design used only time-invariant concepts.[20]

In other words, filter design requirements referring to fast time-variation are purely there for efficiency reasons — the designs are otherwise based on purely time-invariant assumptions.

Further, the analysis methods are only valid for linear systems. As such, the linear behavior of the filters is of primary interest in the design. However, since it is known that nonlinear implementations are highly desirable among users, there will be a default constraint on the filter designs: do not preclude such implementations. This is practically interpretated as: do not stray too far from the original filter topology. In the Moog-styule filters, this means we will stay with the topology of four onepole filters in a loops with feedback. There is no attempt to back up this constraint by theory, it is simply assumed that by keeping this topology, if one adds state saturation into the loop (at one or more places within the loop), that the saturation will behave similarly to the analog saturation. It is also assumed that other filter toplogies would deviate more dramatically in the presense of saturation. Therefore, although the use of nonlinearity is not analyzed as part of the filter designs, it is assumed that is will be part of most actual implementations and thus constrains the space of filter topologies that are considered.[21]

1.6 Filter topics that will not be covered deeply in this thesis

Within the fields of musical-filter design, variable-filter design, and time-varying filter analysis, there are several common topics which will not be covered very deeply in this thesis, as other features of filter performance were the primary focus of the research.

Coefficient and state quantization issues, sensitivity and noise This was an area of great discussion in the late 1960s and early 1970s, and still generates useful research. Leland B. Jackson and others analyzed direct-form filters [125], [183], [157], and later, papers such as [291], looked into error-feedback as a method for reducing noise in such filters. Gold and Rader [93] presented the often-referenced Coupled-Form 2^{nd}-order filter, well-known for good noise and coefficient-sensitivity properties. Later, Yan and Mitra presented two variations on the form [294]. Gray and Markel discussed lattice and ladder filters [97] [98] and their properties, and others continued the analysis [183]. Later, Dana Massie looked into the properties of the normalized ladder [162]. Dattorro researched the noise properties of the Chamberlin-form state-variable filter [57], noting that it had long been known for its good behaviors in this area. Lesser-known structures have been analyzed at various times. For example, Agarwal and Burrus described a 2^{nd}-order form quite similar to the state-variable filter which has good properties [4], which was further analyzed [184]

[20] Still, usage outside the region where the time-invariant assumption holds is likely to be aesthetically useful, even if "incorrect".

[21] It is left to later research to prove the vailidity or invalidity of these assumptions. The intuitive basis for these assumptions is the idea that the small-signal behavior of the system is the linear system we design and analyze.

by David Munson and Bede Liu (who has written extensively on quantization issues in all areas of signal processing). More recently, Jean Ritzerfeld has presented two slightly more complicated 2^{nd}-order forms with low noise properties [219].

Stability of Time-varying filters This is also an area of significant research. Jean Laroche has a nice introduction to the problem in [151] which looks at the behavior of 2nd-order filters in DFII, Coupled Form, Lattice, and Normalized Ladder forms (also noting how all those forms but DFII give rather good interpolated filter settings). The normalized ladder has in particular been known for nice time-variation stability [98], and recently Stefan Bilbao has been doing very nice research into filter structures which maintain passivity and stability through arbitrary modulation ([18] for example). For the purposes of this thesis, possible instabilities due to modulation are simply considered as normal behaviors of the filters.[22] As mentioned previously, the design methods employed in this thesis do not attempt to match the filter's behavior with that of an analog filter during fast time variation, only during static or slowly varying situations.

Other artifacts from Time Variation Clicks and "zipper noise" due to slow and/or discontinuous changes in filter parameters are another artifact of certain types of time variation, and there has been research into methods for minimizing such effects. Jean Laroche looked into the problem in his paper on time-varying stability [151], and Mourjopoulos [181] and Hanna [103] have separately worked on reducing such artifacts. Since one of the design assumptions of filter variation for this thesis is per-sample modulation, zipper noise is not expected, and any discontinuity would have to be in the modulating signal itself, so that the filter does not have any responsibility to reduce this type of artifact (except to be able to accept parameter changes at the sample rate inexpensively).

1.7 End Notes

Finally, it must be noted that, due to the popularity of electronic music synthesis, and particularly due to the Techno/House revolution, there is an extremely large do-it-yourself and synth-software community on the internet through which much more background information can be found than could be put into this thesis, interspersed among a plethora of opinions, theories, and critiques of any aspect that you can think of about music synthesis, virtual analog, and the world in general. Particular resources of note are (and this is nowhere near a complete list, and in no particular order):

Vintage Synth Explorer `http://www.vintagesynth.com/`

Harmony Central `http://www.harmony-central.com/`

[22] Since the use of saturation nonlinearity in the filter loop is such a popular feature of most Moog-style filter implementations, such nonlinearities would likely moderate the effects of time-variation-induced instability to some extent.

The Synth-DIY mailing list http://www.euronet.nl/~rja/Emusic/Synth-diy/

Don Tillman's Page http://www.till.com/

MusicDSP.org http://www.musicdsp.org/

MusicDSP mailing list http://shoko.calarts.edu/~glmrboy/musicdsp/music-dsp.html

comp.dsp and its webpage http://www.dspguru.com/comp.dsp/tricks/tricks.htm

Music Machines http://machines.hyperreal.org/

Analog Heaven mailing list http://machines.hyperreal.org/Analogue-Heaven/

Synth Museum http://www.synthmuseum.com/

SynthZone http://www.synthzone.com

SynthZone Analog page http://www.synthzone.com/analogue.htm

Synthfool http://www.synthfool.com/

Chapter 2

Root-Locus Interpretation of Pole-Variable Filters

2.1 Introduction

Upon realizing that the Moog filter was in the exact topology in which classical control-systems are taught, and in which root-locus analysis is taught, it was a small jump to realize that root locus is very well suited to describing the behavior of filter variation, particularly of the class of variable filters whose primary behaviors are determined by their pole locations, such as the state-variable filter and the Moog-style filter. As such, we set about to look at the variation of a number of filters (including those) in terms of their pole motions, primarily to build intuition as to why they vary in the ways that they do.

As noted in the introductory chapter, these analyses are only valid for time-invariant systems (or are only close for slowly time-varying systems). As such, these analyses only apply to usage modes where the variation is slow or locally constant. This is expected to cover a large range of normal usage. These analyses do not apply when the filters are modulated at fast rates. This thesis does not explore analyses for these situations.

2.1.1 Introduction to Root Locus

A more in-depth introduction to Root Locus will be given in Appendix A. In short, a root locus is the the locus of all roots of a polynomial (usually) as one or more parameters change. More specifically, it usually is used to analyze the motion of the poles of a feedback system as the feedback gain varies. Take a linear system $G(s)$ (or $G(z)$),[1] which can be expressed as a rational function

[1] Root locus works exactly the same in s and z, as polynomials are polynomials regardless of the variable. The difference between s and z is the interpretation we place on pole and zero locations in each plane.

$G(s) = N(s)/D(s)$, where D and N are simple polynomials in s. Then, we put negative feedback around the loop with a gain of k in the feedback, and get get the total system

$$\frac{G(s)}{1 + kG(s)} = \frac{N(s)}{D(s) + kN(s)} \tag{2.1}$$

The poles of the system as k varies are therefore the roots of $D(s) + kN(s) = 0$ as k varies.

Put in other words, the root locus traces the locations in the s plane where the phase of $G(s)$ is $\pi \pm n(2\pi)$.

We will not list all the root locus properties here (again, see Appendix A), but one or two will get us started:

When $k = 0$, the poles of the system are the roots of $D(s)$ (i.e., the poles of the open-loop system). As k moves away from zero, the poles move (in a direction depending on the sign of k) away from their staring point and through various curves until, as $k \to \infty$, the poles approach the roots of $N(s)$ (i.e., the zeros of the open-loop system). Therefore, as one might expect, the roots of D and of N both affect the paths of the pole curves.

Now, if we solve for k, we get:

$$k = -\frac{D(s)}{N(s)} \tag{2.2}$$

And since k is real (we will assume that we are dealing with real-valued systems with real-valued coefficients), this implies that:

$$\mathrm{Im}\left(\frac{D(s)}{N(s)}\right) = 0 \tag{2.3}$$

This implicit equation defines a set of curves in the s plane, which are the root locus (for all real values of k). If one were to choose a point on these curves, one could solve for the k at that point by plugging that value of s into Equation 2.2.

One of the uses of root locus is that, with practice, one can develop intuition into how the curves are shaped relative to the open-loop pole and zero locations. Hence, by developing that intuition and applying it to the analysis of variable filter pole movement, one may gain intuition into the basic behaviors of the filter variation.

2.1.2 But why? Isn't Root Locus ancient history?

Well, yes, in the field of Control Systems, many newer and better controller design techniques have been developed over the years,[2] and Root Locus is no longer particularly exciting there. However, Root Locus analyzes systems in ways that weren't necessarily usable to their fullest in controls design: it shows the whole evolution of a part of the system behavior through the range of a particular parameter. Control systems design, on the other hand, was (at least historically) about the

[2]The interested reader is referred to any modern control-system textbook, such as [66][168][87].

search for a single setting of the controller to get the best response of the complete system. Thus most of the data in a root locus was not directly applicable in designing a controller. Furthermore, pole locations were found to not necessarily be the optimal descriptions of the system from which to analyze a controlled system. Therefore, other design methods (optimal design, etc.) eventually outclassed the root locus as a way of designing controllers.

However, there are fields where the knowledge of how a system changes with its parameters (as opposed to trying to find a single 'best' parameter set) is particularly useful. The field of variable audio filter design is such a field. Variable filters are intended to be used across a range of their parameters, and preferably to be "retuned" cheaply, so as to be varied in real-time (even modulated up to audio rates), so the complete locus can be of use, rather than just a region near where the locus crosses the stability boundary. Additionally, there is a subclass of variable filters whose behavior can be almost completely described by pole locations (which was not necessarily true in control systems). Root Locus directly shows the way such a filter varies with changes in one of its coefficients. Such visualization of the filter behavior can be a powerful tool to assist a designer in understanding their filters.

2.1.3 Pole-Variable Filters

The class of filters we will be concentrating on are those for which root-locus analysis is most applicable, a class that I call *Pole-Variable Filters*. In essence, variable filters whose behavior and variation are primarily describable in terms of the locations (and motions) of their poles. Obviously, variable all-pole filters fit this exactly, but we can also look at filters whose zeros have only a gross effect on the behavior of the filter (such as defining a lowpass versus a highpass or bandpass response), and which may not vary at all with the filter variation. Many basic 2^{nd}-order filters fit this description, including the Chamberlin State-variable filter and the Moog-style filter which are our primary interest.

2.2 Looking at Some Basic Filters

We will first look at how a few basic filters can be interpreted in terms of root loci in their coefficients. Then we will digress into some discussions on the concept of a "Root Locus Filter", before jumping into looking at the Digital State-Variable Filter and the Moog-style VCF in terms of root loci. As the next chapter deals with these two filters in much more depth, we will not delve too deeply into their analysis in this chapter. Instead, we will just show how their basic behaviors can be visualized and interpreted in terms of their root loci.

2.2.1 "Circle Filters"

When we think about variable filters from a root-locus standpoint, one of the first root-locus behaviors we might think of is the fact that the circle tends to show up frequently in loci. The circle is also an important shape in discrete-time filtering — the region of stability is a circle, and circles around the origin in the z plane represent contours of equal pole decay-rate (and, roughly, bandwidth, depending on definitions). See Figure 1.1 on p. 16 for review of these contours.

As such, we can define a straightforward variable filter as one whose locus in one control is a circle about the origin, and whose other control is the radius of the circle. We will call this filter, not surprisingly, the "circle filter", though it does resemble some existing designs (more on that later).

We can set up such a filter by placing an open-loop zero on $z = 0$, which will set up the center of the root-locus zero. The other zero can be placed somewhere outside the unit circle. It turns out a good location is the inverse of the first zero location (i.e., ∞).

To choose the open-loop poles, we note that if we have two poles on the real axis, on the same side of the zero, then the locus will split off the axis between them, and circle around the zero. Thus, to control radius, we can simply control the placement of the poles such that the split-off point is the desired radius r. Two common methods for doing this are:

- Place both poles on $z = r$. We will call this filter a Type 1 circle filter.
- Place one pole on $z = 1$, and the other pole on $z = r^2$. We will call this filter a Type 2 circle filter.

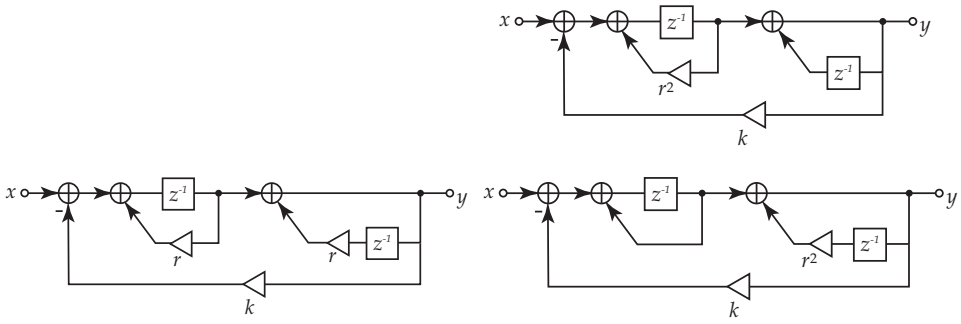


Figure 2.1: "Circle Filters". Filters set up directly to have a useful root locus (in this case a constant-bandwidth variable filter). Left: Type 1, Right: Type 2, two possible implementations.

Therefore, we need to create a loop containing two poles on the desired locations, and one zero on $z = 0$, with a loop gain. Such filters are shown in Figure 2.1. We make use of the fact that a backward-difference onepole filter has its zero at z=0, and a forward-difference onepole has its

zero at ∞ (see Section 1.3). As we have designed, the root-locus equation for these filters are:

$$\text{Type 1:} \qquad (z-r)^2 + kz = 0 \qquad \rightarrow (z^2 - 2rz + r^2) + kz = 0 \tag{2.4}$$

$$\text{Type 2:} \quad (z-1)(z-r^2) + kz = 0 \quad \rightarrow (z^2 - (r^2+1)z + r^2) + kz = 0 \tag{2.5}$$

And the loci in k are as shown in Figure 2.2. For Type 1, at $k = 0$, the poles are on the real axis, and as k increases, the poles move around a circle centered on $z = 0$ at the radius r, their angle controlled by k, until they meet at $z = -r$ when $k = 4r$, then split, one heading in towards $z = 0$ and the other heading towards $z \rightarrow -\infty$.

For Type 2, the poles are separated on the real axis at $k = 0$, and they move towards each other and meet at $z = r$ when $k = (1-\sqrt{r})^2$, then head around the circle, meeting again when $k = (1+\sqrt{r})^2$, then splitting toward $z = 0$ and $z \rightarrow -\infty$.

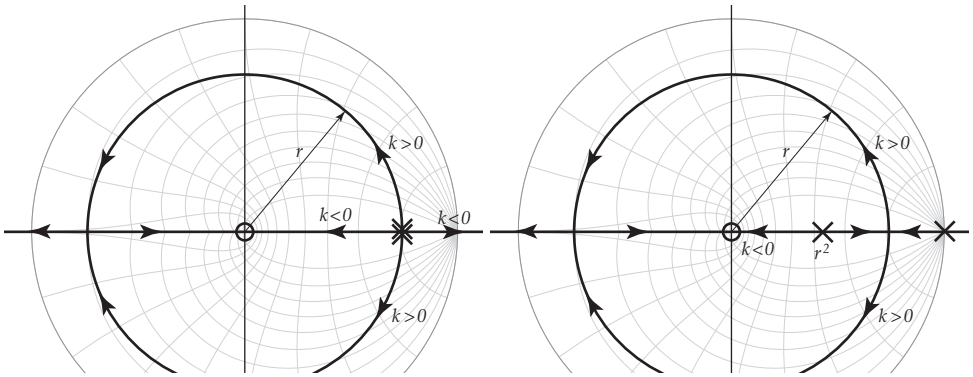


Figure 2.2: Circle Filter loci. Left: Type 1, Right: Type 2.

Loci in r for these filters are shown in Figure 2.3. Note that the loci are 2^{nd}-order in r. Since the parameters are not completely independent, the loci in r do no appear as radial lines. For the Type 1 filter, the open-loop poles are $z = 0$ and $z = -k$. The open-loop zeros (the roots of the k^2 term) are both at infinity, and the roots of the k term are at $z = 0$ and $z \rightarrow \infty$. Interestingly, the off-real-axis shape for this locus is a $\sqrt{x}$ shape.[3]

However, these filters still have rather straightforward controls: the radius is directly controlled by the coefficients of the onepole filters (either as-is or squared), and k controls the frequency, though not linearly. By analogy to the well-known 2^{nd}-order denominator expression $z^2 - 2r\cos(\theta)z + r^2$, we get:

$$\text{Type 1:} \qquad k = 2r(1-\cos(\theta)), \qquad \theta \in [0,\pi] \Rightarrow k \in [0,4r] \tag{2.6}$$

$$\text{Type 2:} \qquad k = r^2 - 2r\cos(\theta) + 1, \qquad \theta \in [0,\pi] \Rightarrow k \in [(r-1)^2, (r+1)^2] \tag{2.7}$$

[3] In fact, we can show that for $r > k/4$, the shape is $y = \sqrt{k}\sqrt{x + k/4}$.

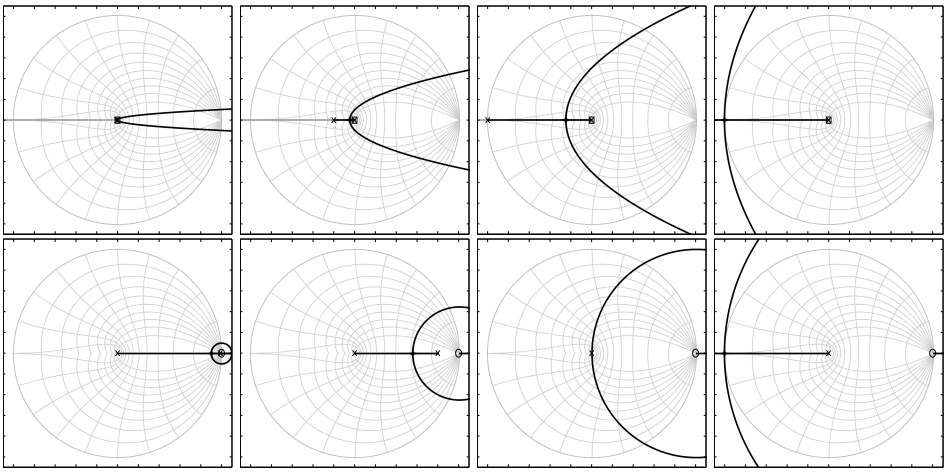


Figure 2.3: Circle Filter loci in r. Top: Type 1, Bottom: Type 2. Left to right: $k = 0.01$, $k = 0.2$, $k = 1$, $k = 4$.

$$k = (r-1)^2 + 2r(1 - cos(\theta))$$

As such, we see that the Type 1 filter can control those parameters nearly independently, requiring just a scaling by r in the calculation of k to separate the controls, which is not very expensive. The Type 2 filter needs a little more calculation to separate k from r, but not that much more.

If one compares the filter block diagrams to those found in Dattorro 1988 [56], we see that the Type 1 form is quite close to Rader-Gold form [93] (if we were to split k up into two factors of $\sqrt{k}$ and rotate one of them back through to between the two first-order filters). Also, the upper Type-2 implementation is quite similar to Agarwal-Burrus form [4] (particularly if we were to replace r^2 with $1-p_1$). In both cases, however, the Circle-filter forms pick off the feedback from the 2^{nd} onepole *before* the delay, whereas the Rader-Gold and Agarwal-Burrus forms pick off after the delay.

We call this type of filter (one where the frequency control sweeps constant-radius contours) a “Constant-Bandwidth" filter, as sweeping the frequency with a constant r gives a filter whose bandwidth stays constant. By creating appropriate zeros (via combinations of outputs from various locations within the structure), one can create filters of various shapes (highpass, lowpass, boost, cut, shelf, etc.).

Now, many users prefer, instead of constant-bandwidth filters, constant-Q filters (i.e., filters whose Q can be kept constant through frequency sweeps). As noted, we will look more at this filter in a later chapter, and work on turning it into a Constant-Q filter.

2.2.2 Direct-Form All-Pole Filter

First-Order Filter

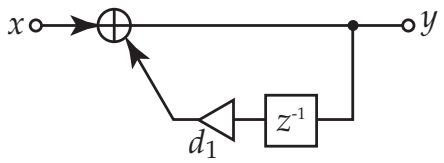


Figure 2.4: A discrete-time one-pole filter

In Figure 2.4, we draw a first-order discrete-time one-pole filter with transfer function

$$\frac{1}{1 + d_1 z^{-1}}. \tag{2.8}$$

We note that this filter is basically in root-locus form already, with d_1 as the loop gain variable. The Root-locus equation is:

$$z + d_1 = 0 \tag{2.9}$$

Comparing with the root-locus form template $D + kN = 0$, we can say that the "open-loop zeros" are at $z \to \infty$, and the "open-loop pole" is at $z = 0$. This is a simple root locus, starting at $z = 0$ when $d_1 = 0$, and then moving along the real axis, either to the left for $d_1 > 0$, or to the right for

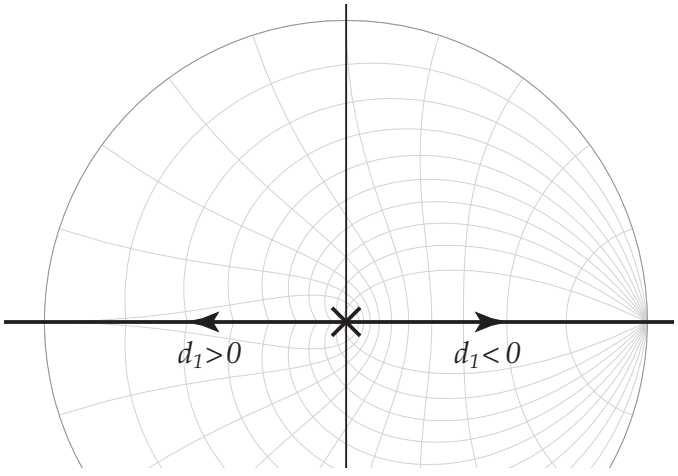


Figure 2.5: One-pole filter. Locus in $d_1 \geq 0$. The locus stays on the real axis. The locus in $d_1 < 0$ heads towards $z \to +\infty$. The locus in $d_1 > 0$ heads towards $z \to -\infty$.

$d_1 < 0$ (see Figure 2.5). Further, its location is $-d_1$ (so it crosses the unit circle at $|d_1| = 1$).

Second-Order Filter, Locus in d_2

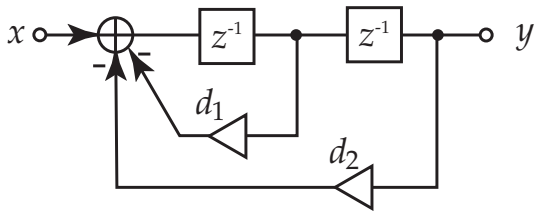


Figure 2.6: A discrete-time two-pole filter.

We can think of any order of direct-form allpole filter as a hierarchy of loops. Each order picks off on the output of the last delay and adds a new loop back to the input (see Figure 2.6). The transfer function is

$$\frac{1}{1 + d_1 z^{-1} + d_2 z^{-2}} \tag{2.10}$$

and the root-locus function using d_2 as the gain variable is

$$z(z + d_1) + d_2 = 0. \tag{2.11}$$

So the open-loop zeros are again at $z \to \infty$, and the open-loop poles are $z = 0$ and $z = -d_1$. For $d_2 \geq 0$, this locus starts at these locations at $d_2 = 0$, and the poles move towards each other until meeting at $z = d_1/2$, when $d_2 = \frac{|d_1|^2}{4}$. Beyond that value, the poles split off the real axis and head vertically, parallel to the imaginary axis (one going up, one going down), with their real parts equal to $-d_1/2$ (Figure 2.7). These will hit the unit circle when $d_2 = 1$, which can be verified by the traditional denominator expression $z^2 - 2r\cos(\theta)z + r^2$, hence $d_2 = r^2$, and thus when $r = 1$, $d_2 = 1$. For $d_2 \leq 0$, the poles stay on the real axis and head away from each other towards $z \to \infty$ in both directions (crossing the unit circle at $d_2 = -1 \pm d_1$).

It is often noted that direct-form root locations have bad coefficient sensitivity [195]. This can be corroborated from the root-locus standpoint by remembering that the k sensitivity gets large as the poles get near each other, and even goes to infinity when the poles touch. Thus the poles are expected to get very sensitive to d_2 around $d_2 = |d_1|^2/4$. As such, the poles "jump" up off the real axis, and if d_2 is quantized, the distance between adjacent pole locations will get very large, as seen in a plot of quantized-coefficient pole locations (see Figure 2.8, as shown in Oppenheim and Shafer [196], and others). Note that the sensitivity is only in d_2; the poles are not exceptionally sensitive to d_1.

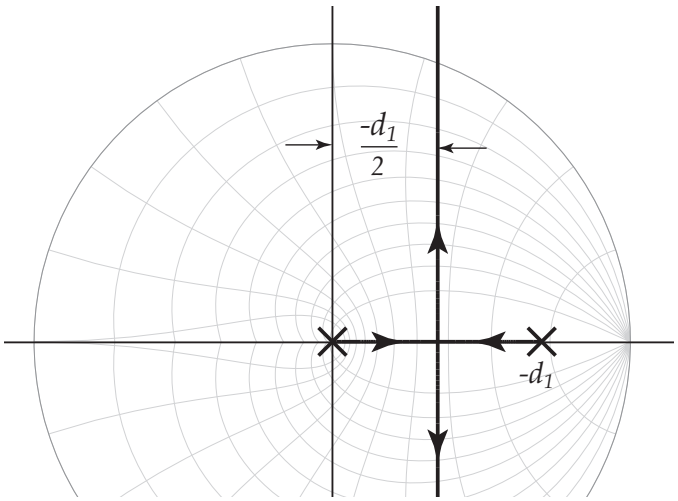


Figure 2.7: Direct-Form two-pole filter. Locus in $d_2 \geq 0$. The locus in $d_2 < 0$ stays on the real axis and heads away from the poles towards $z \to \pm\infty$.

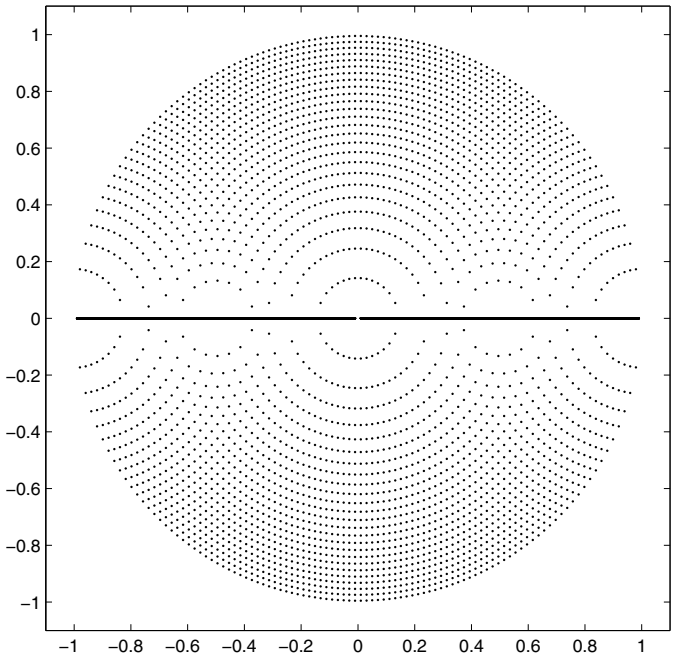


Figure 2.8: Possible stable pole locations of a direct-form twopole, d_1 and d_2 quantized to multiples of 0.02.

Second-Order Filter, Locus in d_1

If we instead look at the 2nd-order filter's root-locus equation, reading d_1 as the locus gain variable, we get:

$$(z^2 + d_2) + d_1 z = 0 \tag{2.12}$$

So, the open loop zeros are $z = 0$ and $z \to \infty$, and the open-loop poles are $z = \pm\sqrt{-d_2}$. The root locus for this is one of two cases:

- If $d_2 > 0$, the open-loop poles are on the imaginary axis, at $z = \pm j\sqrt{d_2}$. The complete root locus is first a circle, centered on the open-loop zero at $z = 0$, with a radius of $\sqrt{d_2}$. The locus starts at $z = \pm j\sqrt{d_2}$, for $d_1 = 0$, and moves down the circle until hitting the real axis at $d_1 = 2\sqrt{d_2}$. For larger d_1, the locus splits with one pole heading in towards $z = 0$ and the other heading out to $z \to \infty$.

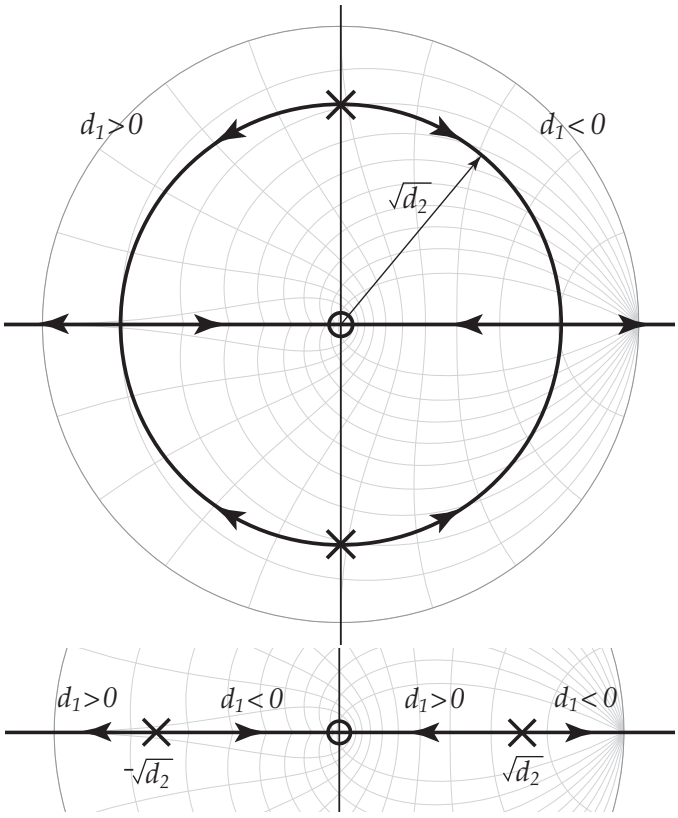


Figure 2.9: Direct-Form two-pole filter: Root Locus in d_1. Top: $d_2 > 0$, Bottom: $d_2 < 0$.

The locus is split into two halves: the $d_1 > 0$ half is in the left half plane, and the $d_1 < 0$ half is in the right half plane. The locus is actually symmetric about the imaginary axis (as well

as the real axis), which can be shown by noting that $(z^2 + d_2) + d_1(-z) = (z^2 + d^2)(-d_1)z$ (i.e., negating d_1 is equivalent to negating z, and since the locus is symmetric about the real axis, negating z has the effect of flipping the plane across the imaginary axis).

- If $d_2 < 0$, the open-loop poles are on the real axis, at $z = \pm\sqrt{-d_2}$. The locus for $d_1 > 0$ are the regions $z < -\sqrt{d_2}$ and $0 < z < \sqrt{d_2}$, and for $d_1 < 0$ it is the regions $-\sqrt{d_2} < z < 0$ and $z > \sqrt{d_2}$. In other words, one of the poles heads towards the zero at $z = 0$, and the other towards $z \to \infty$.

Higher-Order Direct-Form Filters

Past 2nd-order, things get more complicated. There are significantly more possibilities (more variables to interpret loci in, and more combinations of possible locus shapes, etc.). As such, the loci are more difficult to describe *in general terms*, though it is still straightforward to use root locus to analyze specific situations, such as looking at perturbations in coefficients from a particular design.

For example, we can look quickly at a 3rd-order allpole. The root-locus equations in the three coefficients are:

$$z(z^2 + d_1 z + d_2) + d_3 = 0 \tag{2.13}$$

$$(z^3 + d_1 z^2 + d_3) + d_2 z = 0 \tag{2.14}$$

$$(z^3 + d_2 z + d_3) + d_1 z^2 = 0 \tag{2.15}$$

So, for the locus in d_3, the open-loop poles are the poles of the 2nd-order part of the filter, plus a new pole at $z = 0$, and there are three zeros at $z \to \infty$. Thus the locus in $d_3 > 0$ starts at the pole locations of the internal 2nd-order filter and $z = 0$, and heads out to asymptotes at $\theta = \pm\pi/3, \pi$. The locus in $d_3 < 0$ starts at the same locations, in the opposite directions, and ends up on asymptotes at $\theta = 0, \pm 2\pi/3$.

The locus in d_2 brings one of the open-loop zeros in to $z = 0$, and the locus in d_1 brings in another onto $z = 0$. Thus the locus in $d_2 >$ will have asymptotes at $\theta = \pm\pi/2$, and the locus in $d_2 < 0$ will have asymptotes on $\theta = 0, \pi$. The loci in d_1 will have asymptotes at either $\theta = 0$ or $\theta = \pi$. The open-loop poles are not simple to describe, however.[4]

Figure 2.10 shows some representative loci for a 3rd-order allpole. These loci are "variation loci", because they show the effect of varying one parameter in a particular filter design, and keeping the others at their values for the design. The filter in this case is $z^3/(z^3 - 0.5z^2 - 0.5z + 0.5)$. As such, the three root-locus equations for the figure are actually:

$$z(z^2 - 0.5z - 0.5) + d_3 = 0 \tag{2.16}$$

[4] . . . without bringing in the equations for the roots of a third-order filter, which get a bit cumbersome for unspecified coefficient values.

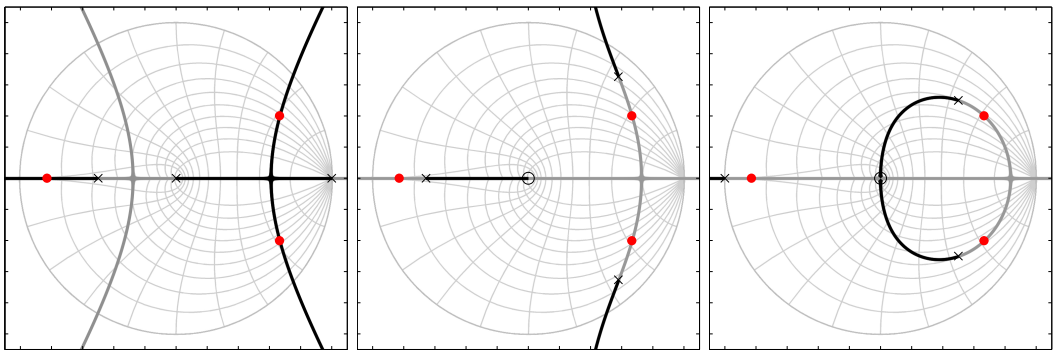


Figure 2.10: Direct-Form three-pole filter: Variation Root Loci in d_3 (left), d_2 (center), and d_1 (right). Black: coef>0, Gray: coef<0. Dots: poles for base configuration of $d_1 = -0.5, d_2 = -0.5, d_3 = 0.5$.

$$(z^3 - 0.5z^2 + 0.5) + d_2 z = 0 \tag{2.17}$$

$$(z^3 - 0.5z + 0.5) + d_1 z^2 = 0 \tag{2.18}$$

In general, we can say the following about an n^{th}-order direct-form allpole: The locus in the highest-order coefficient has open-loop poles which are simply the poles of the filter of order $n-1$, plus one at $z = 0$, and all the zeros are at $z \to \infty$. Describing loci in other coefficients is more complicated, as the open-loop poles don't have as simple an explanation, just roots of some polynomials. The zeros are interesting, though: if the filter is of order N, then the locus in the i^{th} coefficient has $N - i$ zeros at $z = 0$, and there are always N open loop poles, so there are i asymptotes.

2.3 Some Philosophy

Musical filters as those upon which we will be concentrating tend to have two major realtime controls:

- A control to affect the frequency of interest (the center of a bandpass, peak, cut or notch, or the corner of a shelf or lowpass/highpass response, or generally the frequency of resonance in high-Q situations).
- A control to affect the "strength" of the filtering effect, such as a rolloff rate, or a bandwidth, or Q.

Other controls, such as ones to select a filter type or filter shape, are considered for the purposes of this thesis to be "non-realtime", and hence do not need to be dealt with as cheaply as the primary controls, or at the same speed (i.e.,they are not expected to be modulated, especially not at rates approaching audio frequencies).

In particular, we will be considering the two controls to be resonance frequency ("f_c") and resonance Q ("Q") (or bandwidth, depending on the situation). By defining our frequency as the "resonance" frequency, we are assuming that most of the time, there will be only a single resonance peak, and that the Q of the system will usually be such that there is some sort of peak at the resonance. As such, we will be primarily looking at bandpass shapes (not flattop), or lowpass and highpass shapes with resonance at the cutoff frequency. Parametric Equalizer (EQ) sections, while possibly fitting this framework, are not considered, mainly because this research came from the desire to model classic analog-synthesizer VCFs, and parametric EQs were generally considered a separate class of filters from VCFs. For bandpass filters, f_c is assumed to be the center of the band (or near the center).[5] For low-pass and high-pass filters, f_c is assumed to be associated with the cutoff of the passband.

When analyzing applicable filters from a root-locus perspective, we naturally look to find relationships between the frequency/Q controls and the root-locus parameters.

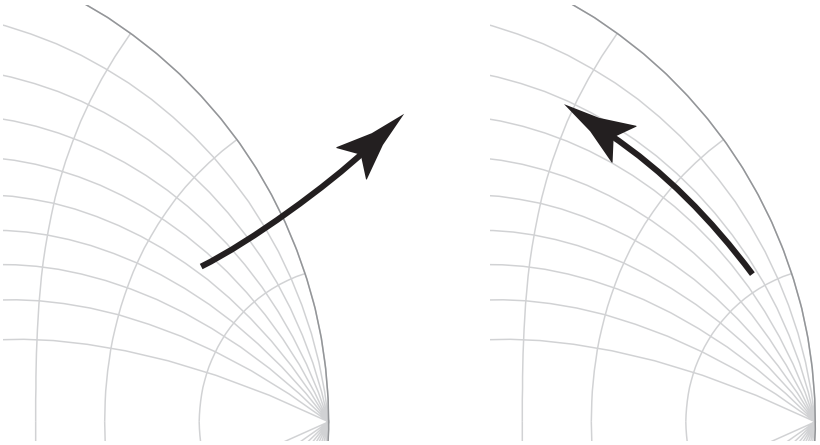


Figure 2.11: Basic root locus trajectories for two classes of discrete-time root-locus filters. Left: the locus "goes out" and hence controls Q or bandwidth; Right: the locus "goes around" and hence controls f_c.

In well-suited filters, the root-locus gain might map directly (or very closely) to one of the desired control parameters. In terms of the shape of the root locus, this would give us two classifications for basic root-locus filter types:

Locus Goes "Out" In these filters, the locus points "out" of the unit circle, and hence the locus gain is effectively a Q or bandwidth/damping control.

[5] This implies that "flat-top" bandpass filters, such as those formed by cascading high-pass and low-pass filters aren't being considered in this discussion. Such filters, if formed using resonant high- and low-pass filters, might have two resonances, one at each end of the passband.

Locus Goes "Around" In these filters, the locus goes approximately parallel to the unit circle or to constant-Q lines, and hence the locus gain is effectively an f_c control.

Ideally, there will be two controls, which correspond directly to each of the above types.

Earlier in this chapter, we have already seen filters for which frequency and bandwidth are controlled by simple parameters (the Circle Filter and even the 2nd-order direct-form filter, though the frequency mapping is a bit cumbersome in that one). This can be attributed to the fact that the constant-bandwidth contours are circles, and the circle is a common root-locus shape.

Constant-Q contours (Section 1.2.1), on the other hand, are shapes that are not seen in first-order root loci. Most importantly, the constant-Q contours leave the real axis from $z = 1$ at a variety of angles (in fact, the angle is directly related to Q). However, it is a property of first-order loci (see Section A.1.2 on p. 254) to always leave the axis perpendicularly (if a pair), or at angles evenly spaced around the range $[0, 2\pi]$, which has two implications:

- The set of departure angles (for a real-coefficient filter, which we are assuming) is limited to integer divisions of 2π. Thus, if we desire to achieve breakaway angles far from $\pi/2$, the required number of simultaneous poles gets large (i.e., one would need to implement a high-order filter to get poles moving away from $z = 1$ at low angles, but at the same time, there would therefore be other poles moving away from $z = 1$ at higher-Q angles).
- For more than two poles leaving the real axis at a single point, if any are moving into the left half plane (assuming the plane is divided at that point), then there must also be one or more poles moving into the right half plane. Therefore, if the point is $z = 1$, then in order to have poles moving into the circle on constant-Q lines, there must also be poles simultaneously moving out of the unit circle. In other words, it requires an unstable filter.

Therefore, we hypothesize that in order to achieve constant-Q trajectories (at least near $z = 1$, if not through the whole frequency range), that higher-order root-loci will be required. Higher-order loci are discussed in Appendix A, but in general, a higher-order locus, as we will be using the term, is a locus where the root-locus equation has higher than first-order terms in the gain variable. For example,

$$D(z) + kN_1(z) + k^2N_2(z) + ... = 0$$

is a 2nd-order root locus equation. We will look into this in relation to the State-Variable filter in the next chapter.

But first, some more general discussion.

We note that there are two ways that the (corner/center) frequency parameter can map onto the frequency coefficient:

poles on DC $\Leftrightarrow k = 0$ We will see this in the frequency-coefficient locus of the state-variable filter.

Therefore, the root-locus in the frequency coefficient has its open-loop poles on the real axis

(or even at $z = 1$), so that $k = 0$ corresponds to DC. The Type-1 circle filter also shows this behavior.

poles on DC $\Leftrightarrow k \neq 0$ This often happens if k is directly a pole coefficient, as in the one-pole filter $1/(z + k)$, as we will see in discretizations of Moog-style filters later, which exhibit this property ($DC \Leftrightarrow k = -1$).

The first case is most straightforward to understand in terms of root loci, since an "endpoint" of the locus (i.e., $k = 0$) also corresponds to an endpoint of the controlled parameter (i.e., $f_c = 0$). Therefore, the open-loop poles and zeros correspond to the endpoint setup of the filter, and thus can usually have a straightforward interpretation in terms of the operation of the filter.

The second case is more tricky because the frequency endpoint is now not a locus-parameter endpoint, and so the open-loop roots tend to have a less obvious meaning (they can seem to be in random locations), and as such, the intended goal of using root-locus to build up intuition on the filter behavior is partially thwarted.

For example, if we look back earlier in this chapter we can see an example of this difference: the Type 1 circle filter (Section 2.2.1 on p. 53) is set up so that its frequency endpoint is right on $k = 0$ (Figure 2.2 on p. 54), and the doubled open-loop pole location is rather straightforward to understand as a radius control, as well as understanding the frequency trajectory as "emanating" from those poles and moving around the circle. However, the d_1 locus in the direct-form twopole filter (Figure 2.7 on p. 58) has its open-loop poles halfway through the frequency range, such that DC corresponds to some non-obvious value of d_1 ($-2r$ in this case, or $-2\sqrt{d_2}$).

However, it is sometimes possible to offset k (to put $k = 0$ on the parameter boundary). Consider a 1st-order root locus equation:

$$D(z) + kN(z) = 0 \tag{2.19}$$

This implies a set of curves in the z plane, containing the root locations for all real values of k. Therefore, we can offset k, as $k = \hat{k} + a$, and have a locus in $\hat{k}$ which is the same shape as the locus in k, since as k varies between $-\infty$ and $+\infty$, $\hat{k}$ will also vary over the same range, just offset. In fact, for any smooth transformation $f : \mathbf{R} \to \mathbf{R}$ whose domain and range both cover all of $\mathbf{R}$, then transforming $k = f(\hat{k})$ in the locus equation will still represent a locus of the same shape (though if f is anything but an affine transformation, the resulting locus will no longer be a first-order locus).[6]

Back to the transform $k = \hat{k} + a$. Substituting it into Equation 2.19, we get:

$$(D(z) + aN(z)) + \hat{k}N(z) = 0 \tag{2.20}$$

[6]This does imply that one may be able to transform some seemingly higher-order loci into first-order loci by change of variables if appropriate functions can be found.

In other words, offsetting k simply moves the "open-loop" poles to some other location along the locus, and the total locus shape stays unchanged. Interestingly, the open-loop zeros stay fixed.

As such, we can note that given a 1st-order locus, any other locus with the same open-loop zeros and with poles somewhere along the given locus, will have the same shape, and will just have a shifted k mapping compared to the given locus. If we try moving the open-loop zeros along the locus, or try moving both the open-loop poles and zeros, we end up with the following properties:

$$(D(z) + aN(z)) + kN(z) = 0 \quad \Leftrightarrow \quad D(z) + (k + a)N(z) = 0 \tag{2.21}$$

$$D(z) + k(D(z) + aN(z)) = 0 \quad \Leftrightarrow \quad D(z) + \frac{ak}{1 + k}N(z) = 0 \tag{2.22}$$

$$(D(z) + aN(z)) + k(D(z) + bN(z)) = 0 \quad \Leftrightarrow \quad D(z) + \frac{a + kb}{1 + k}N(z) = 0 \tag{2.23}$$

The outcome of this is that if we have a filter, such as a Moog-style filter, where a frequency endpoint lands on a non-zero value of the frequency coefficient, we a can do a change of variables to offset the coefficient and get a locus in the offset coefficient which puts the frequency endpoint on $\hat{k} = 0$. The new open-loop roots may therefore have more intuitive interpretations. We will use this fact when analyzing Moog-style filter digitizations later.

This concept applies to higher-order loci as well (in fact, in the next chapter, we use it with 4th-order loci). Let's see what happens when we offset a 2nd-order locus:

$$\begin{aligned} D + kN_1 + k^2N_2 = 0 \quad &\overset{k=\hat{k}+a}{\rightarrow} \\ (D + aN_1 + a^2N_2) + \hat{k}(N_1 + 2aN_2) + \hat{k}^2N_2 = 0 \quad &\end{aligned} \tag{2.24}$$

The "open-loop poles" move along the locus (as in the 1st-order case), and the higher-order open-loop zeros stay put (again, as in the 1st-order case), and the low-order zeros move along a locus in N_1 and N_2.

At the N^{th} order, we get the generic case:

$$\begin{aligned} D + kN_1 + k^2N_2 + \ldots + k^NN_N = 0 \quad &\overset{k=\hat{k}+a}{\rightarrow} \\ (D + aN_1 + \ldots + a^NN_N) + \ldots + \hat{k}^NN_N = 0 \quad &\end{aligned} \tag{2.25}$$

The end terms acts as previous described: the "open-loop poles" move along the locus to their $k = a$ locations, and the highest-order "open-loop" zeros stay put. The intermediate orders move according to standard polynomial properties:[7]

$$\begin{aligned} c_0 + c_1x + c_2x^2 + \ldots + c_Nx^N \quad &\overset{x=y+a}{\rightarrow} \\ (c_0 + ac_1 + ... + a^Nc_N) \quad &+ \end{aligned}$$

[7] Note: this is not unlike the technique in [210].

$$\begin{aligned}(c_1 + 2ac_2 + ... + a^{N-1}Nc_N)y \quad &+ \\ (c_2 + 3ac_3 + 6a^2c_4 + ...)y^2 \quad &+ \\ \cdots \\ (c_{N-1} + Nac_N)y^{N-1} \quad &+ \\ c_N y^N \end{aligned} \tag{2.26}$$

The coefficient transform can be expressed in matrix form as:

$$\begin{bmatrix} \hat{c}_0 \\ \hat{c}_1 \\ \vdots \\ \hat{c}_N \end{bmatrix} = \mathbf{A} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_N \end{bmatrix} \tag{2.27}$$

where A is a square $N \times N$ matrix with elements:

$$A_{ij} = a^{j-i}\binom{j}{i} \tag{2.28}$$

$$A = \begin{bmatrix} 1 & a & a^2 & a^3 & a^4 & \\ 0 & a & 2a^2 & 3a^3 & 4a^4 & \cdots \\ 0 & 0 & a^2 & 3a^3 & 6a^4 & \\ 0 & 0 & 0 & a^3 & 4a^4 & \\ & & & \vdots & & \ddots \end{bmatrix} \tag{2.29}$$

Interestingly, this transformation applied to the coefficient a first-order feedback system (i.e., a one-pole filter), turns the onepole into an integrator-based filter rather than a delay-based one (see Figure 2.12).

In a similar vein, Agarwal and Burrus use a simple offsetting change of variables to derive a new filter type from the direct-form two-pole filter (in order to get much better performance in the presence of quantized coefficients and states). In their case, they offset z, along the lines of:

$$\begin{aligned} \hat{z} &= z + 1 \\ z &= \hat{z} - 1 \\ z^{-1} &= \frac{1}{\hat{z} - 1} \end{aligned}$$

such that a delay gets turned into an integrator. We ote that the form they derive is not unlike the state-variable filter form and the circle-filter forms.

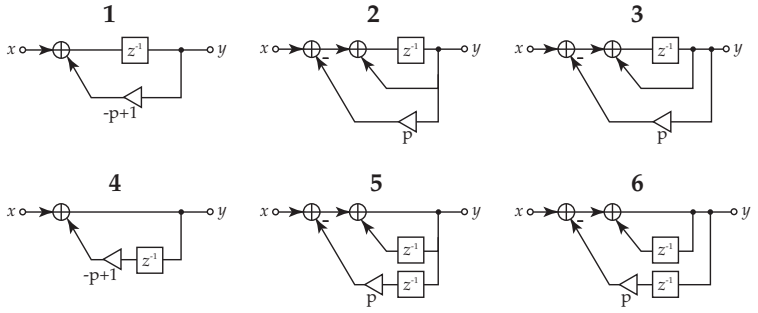


Figure 2.12: Offsetting a one-pole filter's coefficient by one can give a filter which is based on an integrator rather than a delay, much like in δ-operator, and in the state-variable filter. This is most obvious with a forward-difference integrator (top row).

In the case of the forward-difference onepole, these two transforms are effectively equivalent:

$$\frac{1}{z+\hat{p}} \overset{\hat{p}\leftarrow p+1}{=} \frac{1}{z+(p+1)} = \frac{1}{(z+1)+p}$$
$$\frac{1}{\hat{z}+p} \overset{\hat{z}\leftarrow z+1}{=} \frac{1}{(z+1)+p}$$

The transform on z, as done by Agarwal and Burrus, is expected to have a more general applicability as a filter manipulation tool, though.

2.4 State-Variable Filter

2.4.1 Review

Three of the more well-known discussions on the digital state-variable filter are Chamberlin's book *Musical Applications of Microprocessors* [37], Dutilleux's Ph.D. thesis [69], and Dattorro's AES paper on effects and filters [57]. Chamberlin introduces the filter, and discusses uses and interesting variants on the filter, though no derivation of the exact form is discussed.[8] In Dutilleux's thesis, he derives the state-variable filter again, though he draws it slightly differently from Chamberlin's filter (he also explores one or two variants, including attempting a bilinear transformed version) Dattorro creates a derivation of the filter based purely on a spectral specification of the lowpass filter response and a few necessary approximations, though he does the derivation directly on the transfer function, borrowing Chamberlin's topology for the implementation once he is able to derive a transfer function in the same form as that of the Chamberlin form.

[8] Dattorro states that Chamberlin's derivation was via impulse-invariant transformation of the continuous-time filter.

Comparing Chamberlin's and Dutilleux's forms

The filter form which Dutilleux derives looks on the surface a bit different from Chamberlin's form (Figure 2.13). However, they are actually very similar, as we will show. But first, we should point

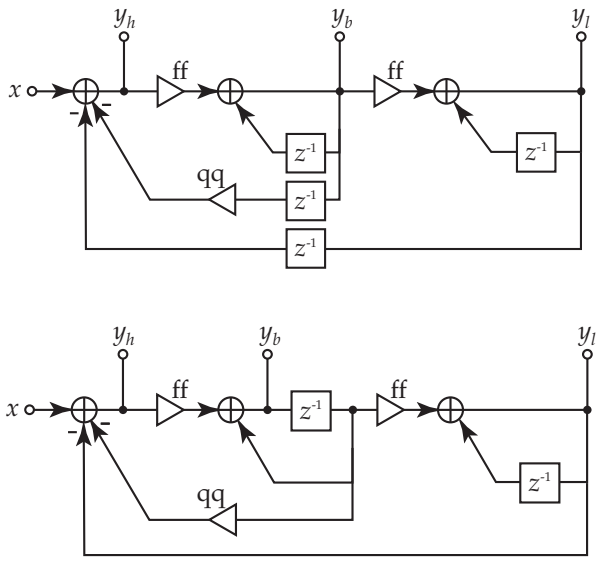


Figure 2.13: State-Variable Filter Forms: Top: Dutilleux Form (as typically presented), Bottom: Chamberlin Form

out that the filter drawn in [57], which is identified as the Chamberlin form, has the bandpass output picked off after the delay of the first integrator, whereas Chamberlin's (and Dutilleux's) picks it off before the delay. In this discussion, I will be using the form as drawn by Chamberlin when referring to the "Chamberlin Form".

Now, Dutilleux tends to draw his form non-minimally (i.e., with more unit delays than the actual filter order). This is partly why it looks different from Chamberlin's form. Looking at Figure 2.13, we can see that each pair of delays comes after a signal split, and as such each pair contains exactly the same data. Therefore, we can make the form minimal by simply pulling the delays back through the splits. This is shown in Figure 2.14. This form is now very similar to Chamberlin's form. Comparing the two, we note that the difference appears to be in the integrator forms. Chamberlin has forward-difference integrator as the first integrator, and backward-difference integrator for the second. Dutilleux, on the other hand, has a backward-difference as the first integrator, and the second integrator is a backward difference with respect to the lowpass output, but forward-difference with respect to the feedback. As such, both forms end up with one of each integrator type in the feedback loop, just in opposite order. However, the forms are even more closely related

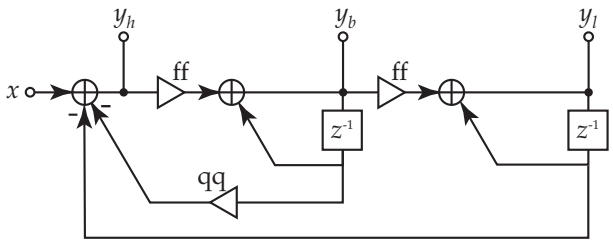


Figure 2.14: Minimal Dutilleux Form

than that. With a bit of block-diagram manipulation, we can actually transform Chamberlin's form into a form nearly identical to Dutilleux's form. This is shown in Figure 2.15.[9] We must admit, however, that the first two steps in the transformation may not be transparent with respect to a fixed-point number system implementation. In particular, splitting up the MAC for the second integrator. As such, the two forms may have slightly different fixed-point behaviors, but note that the after the second step, the MAC is back to its original form, so the difference is expected to be quite small.[10]

Therefore, the Chamberlin form and Dutilleux form differ (mathematically) in just a unit delay on the lowpass filter output. Thus, taken separately, their outputs all have the same filter shape (magnitude).

The transfer functions are:

$$Y_h = X\frac{(z-1)^2}{z^2 + (\mathrm{ff}^2 + \mathrm{ff\,qq} - 2)z + (1 - \mathrm{ff\,qq})} \tag{2.30}$$

$$Y_b = X\frac{\mathrm{ff}z(z-1)}{z^2 + (\mathrm{ff}^2 + \mathrm{ff\,qq} - 2)z + (1 - \mathrm{ff\,qq})} \tag{2.31}$$

$$Y_{lCham} = X\frac{\mathrm{ff}^2 z}{z^2 + (\mathrm{ff}^2 + \mathrm{ff\,qq} - 2)z + (1 - \mathrm{ff\,qq})} \tag{2.32}$$

$$Y_{lDut} = X\frac{\mathrm{ff}^2 \boxed{z^2}}{z^2 + (\mathrm{ff}^2 + \mathrm{ff\,qq} - 2)z + (1 - \mathrm{ff\,qq})} \tag{2.33}$$

The places where the Dutilleux form differs from the Chamberlin form have been outlined.

[9] This does not imply that these two forms are completely equivalent, since a couple of the block-diagram manipulation steps imply differences which might be significant in fixed-point or in the presence of saturation. Instead, this is merely intended to show how similar they are, and that with an ideal number system, they would be effectively equivalent.

[10] One might argue that the first two steps simply demonstrate that one can interchange the order of backward-difference integrator cascaded with a forward-difference integrator (as long as there aren't outputs picking off from certain places within the cascade).

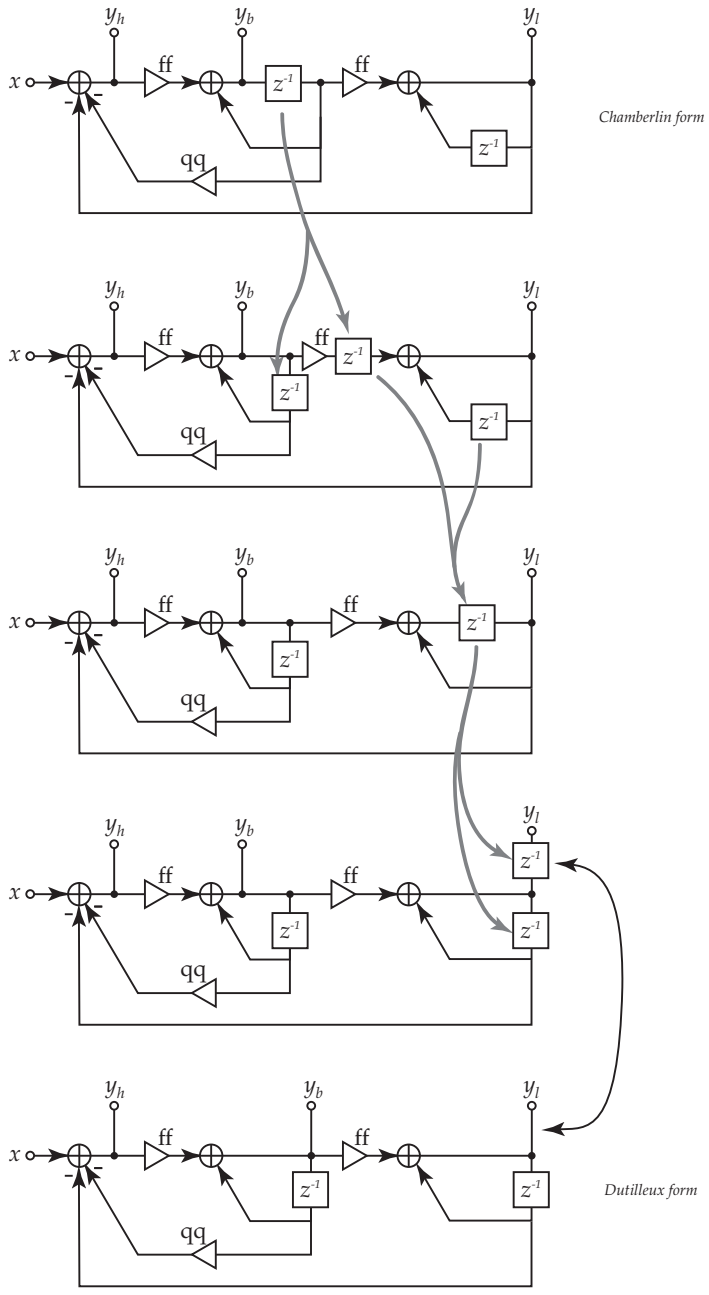


Figure 2.15: converting from Chamberlin to Dutilleux Form. Note that the final difference is just a delay in the lowpass output.

The difference equations are:

Chamberlin:

$$y_l(n) = y_l(n-1) + \mathrm{ff}\, y_b(n-1) \tag{2.34}$$

$$y_h(n) = x(n) - y_l(n) - \mathrm{qq}\, y_b(n-1) \tag{2.35}$$

$$y_b(n) = \mathrm{ff}\, y_h(n) + y_b(n-1) \tag{2.36}$$

Dutilleaux:

$$y_l(n) = y_l(n-1) + \mathrm{ff}\, y_b(\boxed{n}) \tag{2.37}$$

$$y_h(n) = x(n) - y_l(\boxed{n-1}) - \mathrm{qq}\, y_b(n-1) \tag{2.38}$$

$$y_b(n) = \mathrm{ff}\, y_h(n) + y_b(n-1) \tag{2.39}$$

Finally, for completeness, here are the state-space forms.

Chamberlin:

$$\begin{bmatrix} s_1(n+1) \\ s_2(n+1) \end{bmatrix} = \begin{bmatrix} 1 - \mathrm{ffqq} - \mathrm{ff}^2 & -\mathrm{ff} \\ \mathrm{ff} & 1 \end{bmatrix} \begin{bmatrix} s_1(n) \\ s_2(n) \end{bmatrix} + \begin{bmatrix} \mathrm{ff} \\ 0 \end{bmatrix} x(n) \tag{2.40}$$

$$\begin{bmatrix} y_l(n) \\ y_b(n) \\ y_h(n) \end{bmatrix} = \begin{bmatrix} \mathrm{ff} & 1 \\ 1 - \mathrm{ff\,qq} - \mathrm{ff}^2 & -\mathrm{ff} \\ -\mathrm{qq} - \mathrm{ff} & -1 \end{bmatrix} \begin{bmatrix} s_1(n) \\ s_2(n) \end{bmatrix} + \begin{bmatrix} 0 \\ \mathrm{ff} \\ 1 \end{bmatrix} x(n) \tag{2.41}$$

Dutilleux:

$$\begin{bmatrix} s_1(n+1) \\ s_2(n+1) \end{bmatrix} = \begin{bmatrix} 1 - \mathrm{ffqq} & -\mathrm{ff} \\ \mathrm{ff} - \mathrm{ff}^2\mathrm{qq} & 1 - \mathrm{ff}^2 \end{bmatrix} \begin{bmatrix} s_1(n) \\ s_2(n) \end{bmatrix} + \begin{bmatrix} \mathrm{ff} \\ \mathrm{ff}^2 \end{bmatrix} x(n) \tag{2.42}$$

$$\begin{bmatrix} y_l(n) \\ y_b(n) \\ y_h(n) \end{bmatrix} = \begin{bmatrix} \mathrm{ff} - \mathrm{ff}^2\mathrm{qq} & 1 - \mathrm{ff}^2 \\ 1 - \mathrm{ff\,qq} & -\mathrm{ff} \\ -\mathrm{qq} & -1 \end{bmatrix} \begin{bmatrix} s_1(n) \\ s_2(n) \end{bmatrix} + \begin{bmatrix} \mathrm{ff}^2 \\ \mathrm{ff} \\ 1 \end{bmatrix} x(n) \tag{2.43}$$

Both forms use the following design formulas:

$$\mathrm{ff} = 2\sin(\theta/2) = 2\sin(\pi f_c / f_s) \tag{2.44}$$

$$\mathrm{qq} = 1/Q \tag{2.45}$$

Though, as noted in the discussion of Q (Section 1.2.1), the qq formula becomes approximate as f_c gets large (into the left-half plane) and/or when Q gets very small (less than 2 or so).

It is common to combine outputs of the filter to get other responses. For example, the highpass and lowpass outputs of the Chamberlin form can be summed to get a notch response. However,

since the Dutilleux form has a negative unit delay on the lowpass output compared to the Chamberlin filter, the combination to get a notch from it is $y_h + z^{-1}y_l$.

Similarly, the formula to turn a Chamberlin filter into an allpass is

$$y_{ap} = y_l + y_h - \mathrm{qq}\, y_b \quad \text{or} \quad y_{ap} = 1 - \mathrm{qq}\, y_b(1 + z^{-1}), \tag{2.46}$$

whereas for the Dutilleux form, it is

$$y_{ap} = y_l z^{-1} + y_h - \mathrm{qq}\, y_b \quad \text{or} \quad y_{ap} = 1 - \mathrm{qq}\, y_b(1 + z^{-1}). \tag{2.47}$$

Note that it is the same in the alternative form since y_l isn't used in that form.

2.4.2 Comparing with Agarwal-Burrus form

In [56], a filter form is shown and called "Agarwal-Burrus Form". It corresponds to the form referred to as "Realization 2" in the paper by Agarwal and Burrus [4]. A block diagram of this form is shown in Figure 2.16 (with outputs not shown, for simplicity). This is quite similar to Chamberlin form, except mainly that the outer loop gain is put in the feedback rather than in the feedforward path. In fact, with some block diagram algebra (Figure 2.17), we can translate to a form almost identical to Chamberlin form[11] (the only remaining difference being where in the second integrator the feedback is picked off.

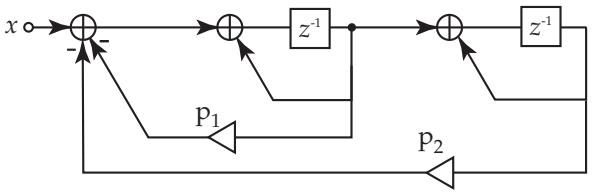


Figure 2.16: Agarwal-Burrus Form

Interestingly, this form is derived by taking a direct-form two-pole filter and "translating the origin of the z plane to $z = 1$". In other words, doing the change of variables

$$w = z - 1. \tag{2.48}$$

[11] Again, by showing that one form can be translated to another, we are not claming them to be equivalent (since such modifications do make a difference numerically), rather, we are noting their similarity by the small number of steps to translate between them.

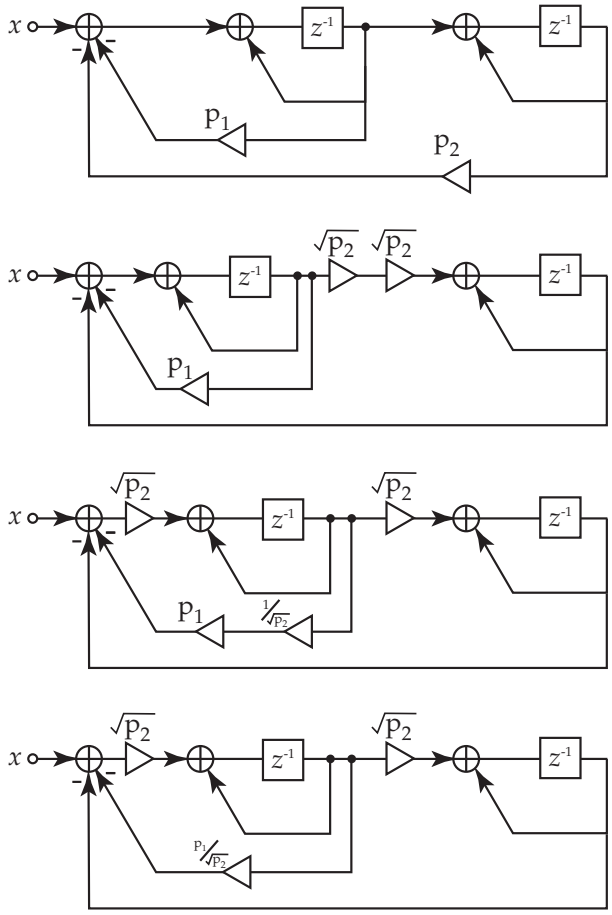


Figure 2.17: Relating Agarwal-Burrus Form to Chamberlin From (nearly)

The resulting w plane has its origin on $z = 1$, the idea being that filter coefficients for poles extremely near the origin are easier to represent (or implement) than poles extremely near 1, as would come about due to high sampling frequencies relative to the pole dynamics. (This is in a similar vein to the arguments for delta-operator discretization [168], and we will see in a moment that the result is quite similar).

A twopole thus transforms as

$$\begin{aligned} \frac{z^{-2}}{1 + a_1 z^{-1} + z_2 z^{-2}} &\Rightarrow \frac{1}{w^2 + (2 + a_1)w + (1 + a_1 + a_2)} \\ &= \frac{w^{-2}}{1 + (2 + a_1)w^{-1} + (1 + a_1 + a_2)w^{-2}}. \end{aligned} \tag{2.49}$$

Thus,

$$\begin{aligned} p_1 &= 2 + a1 = 2(1 - r\cos(\theta)) \\ p_2 &= 1 + a1 + a2 = 1 - 2r\cos(\theta) + r^2. \end{aligned} \tag{2.50}$$

Similarly, for the version transformed to look like the Chamberlin form, we can note, through the use of some approximations, that:

$$\sqrt{p_2} = \sqrt{1 - 2r\cos(\theta) + r^2} \overset{r\to 1}{\approx} \sqrt{2 - 2\cos(\theta)} = 2\sin(\theta/2) \tag{2.51}$$

Which is the same tuning equation as the Chamberlin filter. Furthermore, we can relate $p_1/\sqrt{p_2}$ to $1/Q$, the Chamberlin-filter's Q formula, as follows:

$$\frac{p_1}{\sqrt{p_2}} = \frac{2(1 - r\cos(\theta))}{\sqrt{1 - 2r\cos(\theta) + r^2}} \tag{2.52}$$

Now, if we look at the limit of very low frequencies (i.e. $\theta \to 0$), then Q can be defined as it is in the s plane: as $Q = 1/(2\sin(\phi))$, where ϕ is the angle of the pole from vertical.

A quick geometrical construction (Figure 2.18) allows us to define $\sin(\phi)$ as:

$$\begin{aligned} \sin\phi &= \frac{1 - r\cos(\theta)}{\sqrt{r^2\sin^2(\theta) + (1 - r\cos(\theta))^2}} \\ &= \frac{1 - r\cos(\theta)}{\sqrt{r^2(\sin^2(\theta) + \cos^2(\theta)) + 1 - 2r\cos(\theta)}} \\ &= \frac{1 - r\cos(\theta)}{\sqrt{1 - 2r\cos(\theta) + r^2}} \end{aligned}$$

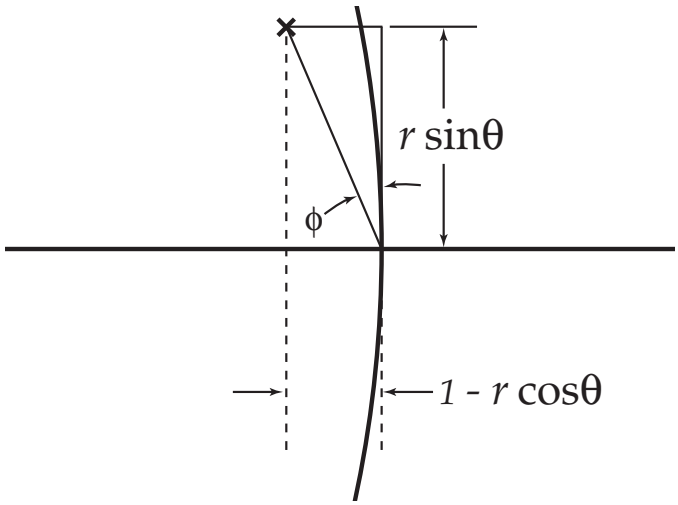


Figure 2.18: Diagram for Q coefficient construction in modified Agarwal-Burrus form.

$$= \frac{p_1}{2\sqrt{p_2}}$$

Thus:

$$\frac{p_1}{\sqrt{p_2}} = \frac{1}{Q} \tag{2.53}$$

i.e., the same as the Chamberlin qq coefficient. Note that this relation is only based on a low-frequency approximation (so that the pole angle can be related to an s-plane pole angle), it does not otherwise approximate anything in the above equations.

Thus, the design equations for Agarwal-Burrus form, in analogy to state-variable-filter's design equations,[12] are:

$$p_2 = (2\sin(\theta/2))^2 = 2(1-\cos(\theta)) \tag{2.54}$$

$$p_1 = \frac{\sqrt{p_2}}{Q} \tag{2.55}$$

$$\tag{2.56}$$

An interesting part of this comparison is the fact that p_2 was split into two factors of $\sqrt{p_2}$. We will look in a later section into why doing this kind of split can sometimes make a filter design easier to tune.

[12] i.e., the tuning equation is derived assuming a pole radius of 1.0

2.4.3 Other Discussion

Dutilleux [68] has a good discussion of output normalization in the state-variable filter. We will not be going into that in this analysis, but we recommend it to the reader.

For the rest of this document, we will be using the Chamberlin form when we discuss the digital state-variable filter.

It was mentioned in passing above that these state-variable filter implementations consist of a forward-difference integrator for one of the integrators, and a backward difference for the other. This definitely points to a discretization by some other means than an $s = f(z)$ substitution, as that tends to give all integrators/onepoles the same digital form. Interestingly, if we look back at how we derived the Circle Filter at the start of this chapter, we ended up with this same situation, because we were specifically placing one open-loop zero on $z = 0$ and the other at $z \to \infty$ in order to get our desired root-locus shape.

2.4.4 Root-Locus Interpretation of State-Variable Filter

Now we will look at the state-variable filter in terms of root-loci of its control parameters.

Continuous-Time SVF

The continuous-time state-variable filter, as described by Chamberlin in [37]), is shown in Figure 2.19.

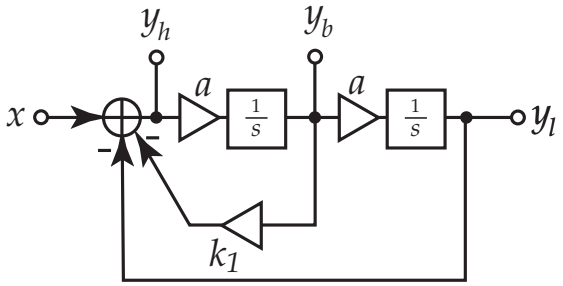


Figure 2.19: Continuous-time state-variable filter

The lowpass transfer function for the filter is:

$$\frac{a^2}{s^2 + ak_1 s + a^2} \tag{2.57}$$

So that the root-locus equations, arranged in terms of k_1 and a are:

$$(s + a^2) + (as)k_1 = 0 \quad (2.58)$$

$$(s^2) + (k_1 s)a + a^2 = 0 \quad (2.59)$$

Therefore, the locus in a will be 2nd-order.

For the locus in k, the open-loop poles are at $\pm ja$, and there is one open-loop zero at $s = 0$ (putting the other at $s \to \infty$). The Locus for this (Figure 2.20) is very much the same as the d_1 locus for a 2nd-order direct-form filter (Figure 2.9), which should not be a surprise at all, since the continuous-time state-variable filter is exactly in direct form, and ak_1 is the same coefficient as d_1. As such, the full locus is a circle of radius a, plus the real axis. For $k_1 > 0$, the poles move in the left-half plane down a circle of radius a towards the real axis. Remember that a circle centered on $s = 0$ is a contour of equal natural frequency in the s-plane, and that the angle of a pole from the imaginary axis is directly related to the pole's Q, so that this is a constant-frequency Q sweep. Now, since the k_1 coefficient is scaled by a, the value of k_1 at which the poles hit the real axis is also scaled. In the discrete-time twopole filter, the poles hit the real axis at $d_1 = \sqrt{2d_2}$, but for this filter, the poles hit the real axis at $k_1 = 2$ (i.e., independent of a). Hence the Q control is trivially independent of the f_c control in the continuous-time state-variable filter. For completeness, the rest of the locus is such: if $k_1 < 0$, the locus is the same shape, but reflected about the imaginary axis into the right-half plane, and so the resulting poles are unstable. If $a < 0$, we get all of the poles on the real axis, and there is always one pole in the right-half plane, so that filter is again unstable. Hence the filter is only stable if both $k_1 > 0$ and $a > 0$.[13]

The 2nd-order locus in a is shown in Figure 2.21. The roots start at $s = 0$ when $a = 0$, and head away towards $s \to \infty$. The angle at which they depart is controlled by k_1:

$$\phi = \sin^{-1}\left(\frac{k_1}{2}\right) \quad (2.60)$$

Now, since the pole-location definition of Q is $Q = 2\sin(\phi)$ (Section 1.2.1), then $k = 1/Q$ exactly, and the locus in a verifies that this filter is a constant-Q filter. This $Q \to k$ mapping is interesting to contemplate, given that the discrete-time qq control uses the same design equation, though it is only accurate for f_c well below $f_s/4$ (and since most users don't go past $f_s/6$ anyway, this design formula is fine, just not "exact").

As discussed earlier, this kind of locus is not possible with first-order loci. We will discuss this behavior of 2nd-order loci in the next chapter.

[13] Theoretically, the right-half-plane pole might end up on $s = 0$ and hence be marginally stable if $k_1 \to -\infty$, but that is not a viable scenario.

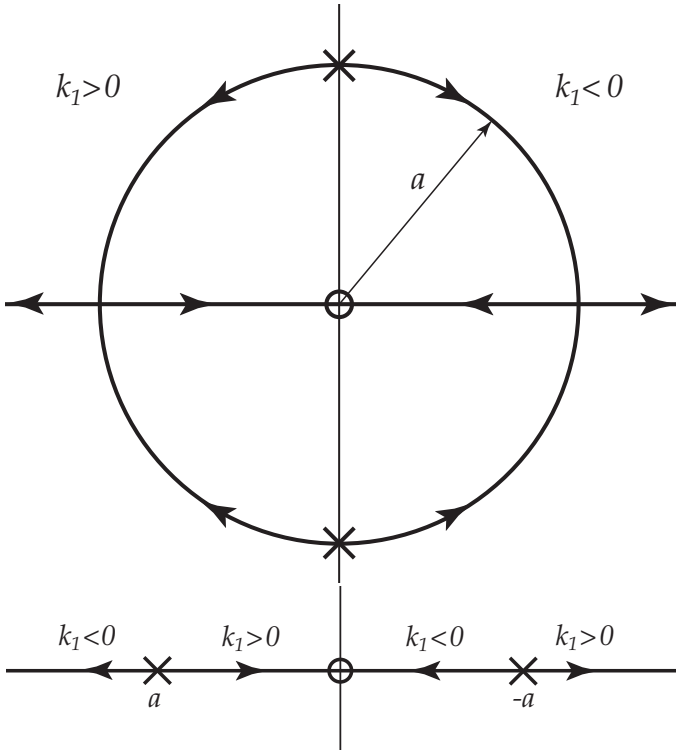


Figure 2.20: Continuous-Time State-Variable Filter: Root Locus in k_1. Top: $a > 0$, Bottom: $a < 0$.

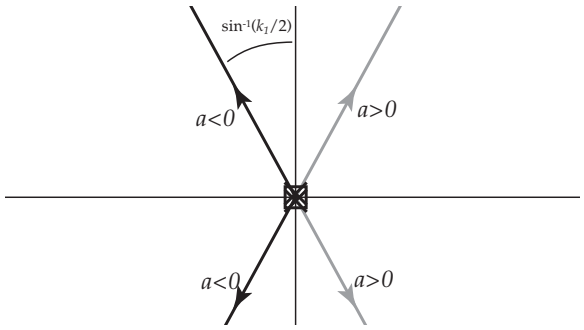


Figure 2.21: Continuous-Time State-Variable Filter: 2^{nd}-order root locus in a.

Digression: Discrete-Time "X" Filter

If we contemplate for a minute the 2nd-order locus in a in the continuous-time state-variable filter, we realize that the 'X' shape is also a useful shape in the z-plane: If we were to place the center of the X on $z = 0$, then we would have a filter with one control on the pole radius ($a = r$) and another control for pole angle ($k_1 = 2\sin(\theta + \pi/2) = -2\cos(\theta)$). Such a filter is shown in Figure 2.22. It has

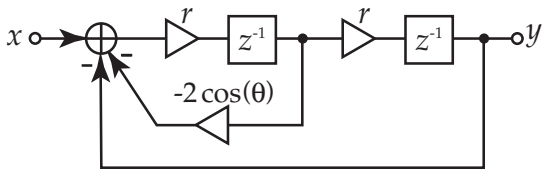


Figure 2.22: Discrete-Time "X" Filter: another basic Root-Locus filter.

the same form and root-locus as the continuous-time State-Variable Filter, but placed in discrete-time. This form is expected to have been derived already, though the author did not find a reference.

One can compare it to the discrete-time 2nd-order allpole filter (Figure 2.6), and consider that d_2 has been split into two gains and rotated around back through the delays. Rotating back through d_1 also caused d_1 to become separated (as d_1 normally is $-2r\cos(\theta)$, and this version has the r removed the from that expression).

The transfer function of the filter is:

$$\frac{a^2}{z^2 + ak_1z + a^2}$$

Note that this filter as drawn is badly scaled. It needs added zeros and/or coefficient-dependent output scaling in order to achieve useful gains across f_c and *bandwidth*. This issue is not uncommon, however: an output picked off between the sum and the first scale has the same transfer function (and hence scaling problem) as the direct-form twopole filter, though combinations of outputs picked off after the delays will not be the same as with the twopole due to the scalings inline with the delays (i.e. to make a biquad out of this form, the numerator coefficients would need to take the scalings by r into account).

Discrete-Time SVF

Now we look at the root-loci of the discrete-time state-variable filter (Chamberlin form, [37] [57]). The root-locus equation, in the form normally written in the denominator of the transfer function

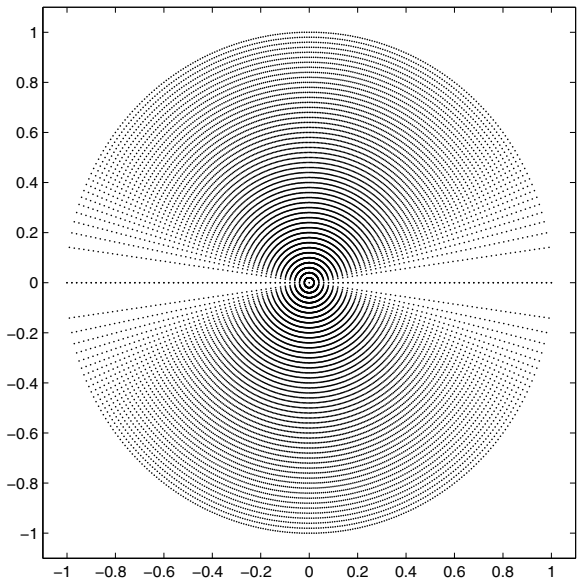


Figure 2.23: Possible stable pole locations of an 'X' filter, both coefficients quantized to multiples of 0.02.

is:

$$z^2 + (\mathrm{ff}^2 + \mathrm{ff}\,\mathrm{qq} - 2)z + (1 - \mathrm{ff}\,\mathrm{qq}) = 0 \tag{2.61}$$

Factoring this into root-locus forms in qq and ff gives:

$$\text{in qq:} \quad (z^2 + (\mathrm{ff}^2 - 2)z + 1) + \mathrm{qq}\,(\mathrm{ff}z - \mathrm{ff}) \quad = 0 \tag{2.62}$$

$$\text{in ff:} \quad (z-1)^2 + \mathrm{ff}\,\mathrm{qq}\,(z-1) + \mathrm{ff}^2(z-0) \quad = 0 \tag{2.63}$$

We note that the locus is 1st-order in the qq coefficient and 2nd-order in the ff coefficient.

The root-locus in qq has its open-loop poles at the roots of $z^2 + (\mathrm{ff}^2 - 2)z + 1 = 0$, which, it turns out, are on the unit circle, at $\theta = (1/2)\sin^{-1}(2\mathrm{ff})$, and it open-loop zeros are at $z = 1$ and $z \to \infty$. The complete locus is a circle centered on $z = 1$ of radius ff, plus the real axis. For $\mathrm{qq} > 0$, the locus starts at the roots on the unit circle and moves inward along the circle until they meet at $z = 1 - \mathrm{ff}$ when $\mathrm{qq} = 2 - \mathrm{ff}$. Then one zero heads towards $z = 1$ and the other heads to the left towards $z \to -\infty$, crossing the unit circle when $\mathrm{qq} = \frac{4-\mathrm{ff}^2}{2\mathrm{ff}}$. The locus for $\mathrm{qq} < 0$ follows the part of the circle that lies outside the unit circle (hence unstable), meeting up on the real axis at $z = 1 + \mathrm{ff}$, then splitting to have one pole head back to $z = 1$ and the other head out towards $z \to +\infty$. Example loci are shown in Figure 2.24. This shape is reminiscent of the continuous-time state-variable-filter's locus in k_1 (see the previous section), mapped into the z-plane with a pole/zero-matching transform. If one remembers back to the introduction (Section 1.2.1), constant-radius lines in the s plane (i.e.,

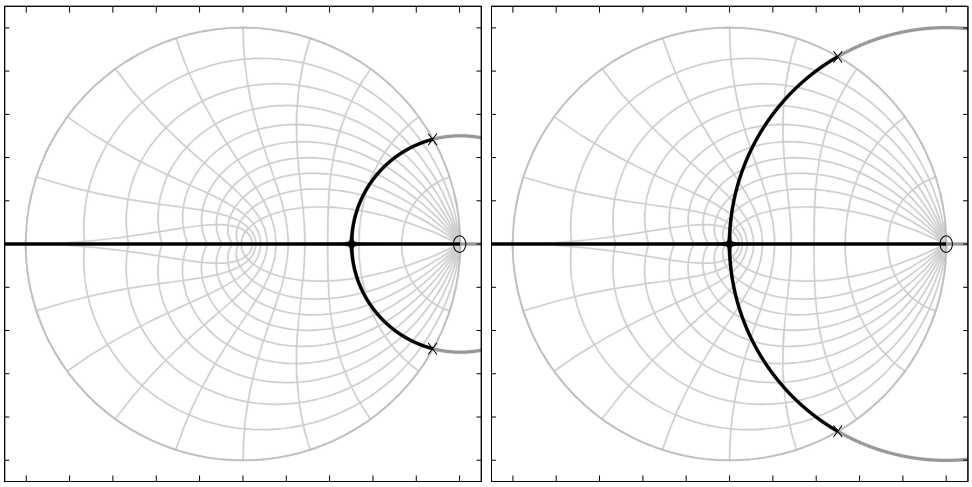


Figure 2.24: Discrete-Time State-Variable Filter: Root Loci in qq. Left: ff = 1/2, Right: ff = 1. Gray: qq < 0

constant natural-frequency) map via the $z = e^{sT}$ transform onto such circles near $z = 1$, though the circles get warped as we get further from there (at higher frequencies and higher damping), which reinforces that this filter is only approximating the ability of the continuous-time filter to achieve constant-frequency Q-sweeps.

The 2nd-order locus in ff has the following open-loop coefficient polynomials:

$$\begin{aligned} D(z) &= (z-1)^2 && \text{roots at } (1,1) \\ N_1(z) &= \text{qq}(z-1) && \text{roots at } (1,\infty) \\ N_2(z) &= z && \text{roots at } (0,\infty) \end{aligned}$$

Thus, its open-loop poles are coincident on $z = 1$, as well as one root of N_1. The open-loop zeros (the roots of N_2) are at $z = 0$ and $z \to \infty$ (as is the other root of N_1). The locus in |ff| << 1 is an 'X' shape, much like that of the continuous-time filter, though centered on $z = 1$. As such, for ff > 0, the poles start at $z = 1$ and move into the unit circle on angles controlled by the qq coefficient (near $\pm\pi/2$ for qq $\to 0$, and near π for qq $\to 2$). They trace a "teardrop" shape before coming back together on the real axis at $z =$ qq $- 1$ when ff $= 2 -$ qq. Then they split, and one heads towards $z \to -\infty$ and the other heads towards $z = 0$ (though, interestingly, sometimes overshooting $z = 0$ before reversing back towards it, which can be seen most clearly in the qq = 1 locus in Figure 2.25.) Some example loci are shown in Figure 2.25. The loci for ff < 0 start out on the right-hand arms of the 'X' shape, then they also start wrapping around to the left, though outside the unit circle, meeting back on the real axis at $-$qq $- 1$ when ff $= -2 -$ qq, then splitting towards $z = 0$ and $z \to \infty$.

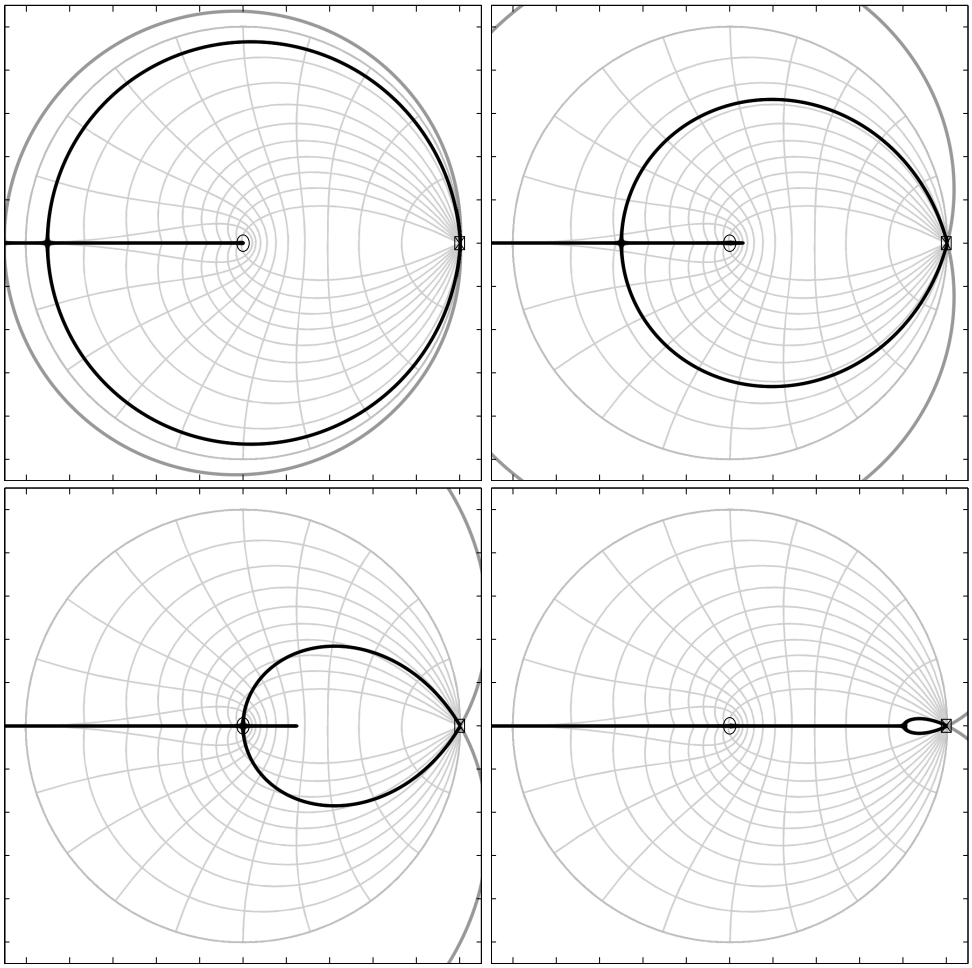


Figure 2.25: Discrete-Time State-Variable Filter: Root Loci in ff. Top Left: qq = 0.1, Top Right: qq = 0.5, Bottom Left: qq = 1, Bottom Right: qq = 1.8. Gray: ff < 0.

As qq passes 1.0, the "teardrop" shape shrinks sufficiently that it no longer makes it past $z = 0$, and as qq $\rightarrow$ 2, it shrinks to the point where it disappears at qq = 2.

We can see that the locus in ff > 0 approximates the constant-Q contours quite well at high Q (low qq) and low ff, deviating mostly in the left half plane, or when the teardrop shape didn't make it all the way around $z = 0$.

We will discuss the details and implications of this locus more in the next chapter.

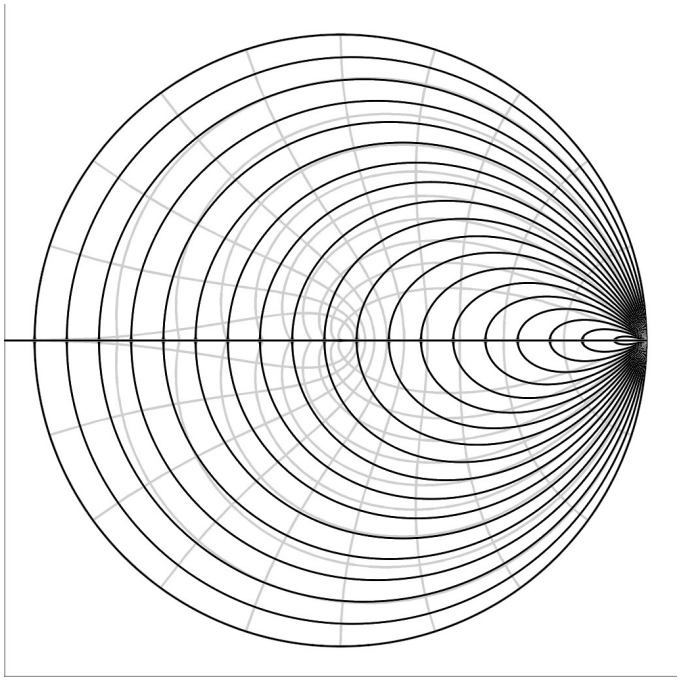


Figure 2.26: Family of State-variable filter root loci in ff, for various 0 < qq < 2.

2.5 Continuous-Time Moog-Style Filter

At his point, we will only look at the continuous-time version of the ideal Moog-style filter. Discrete-time versions are left for the next chapter.

2.5.1 Review

The earliest discussions of the behavior of the ideal continuous-time Moog-style filter that we are familiar with are Moog's AES talk [173] and Hutchins' first article on this filter type in Electronotes [118]. It is shown there that the transistor-ladder structure and the feedback loop implemented four one-pole filters, with essentially the same pole location on the real axis, and that the feedback

loop was a simple gain loop. The result being that the behavior was directly describable as a root locus (though it wasn't until [113] that Hutchins made a root-locus-like interpretation of the corner peaking). Further, Hutchins showed that the same core filter behavior could be implemented with a loop made of op-amp sections, showing that the core behavior could be replicated in a dataflow paradigm (which further reinforces the validity of a root-locus interpretation, as analysis could be purely done from block-diagram concepts rather than having to mix them with simultaneous-equations methods of standard circuit analysis).

Hutchins also theorizes in [113] that the particular topology (four onepoles in a row with feedback, as opposed to using Butterworth, etc.) was in part used due to its ability to be easily voltage controlled. This roughly translates to the digital era as a realization that such a topology has cheaper, simpler, and more independent controls than many other designs.

2.5.2 Root-Locus Interpretation

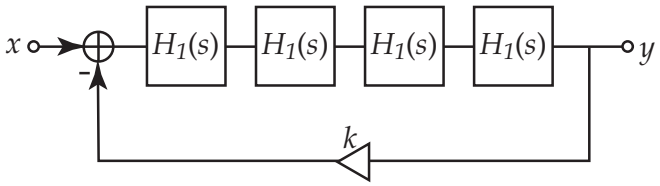


Figure 2.27: Ideal Moog-Style Filter

The ideal Moog-style filter is a cascade of four identical one-pole filters (scaled to have a DC gain of 1.0), with a gain loop around the cascade (with negative feedback), as in Figure 2.27. The system transfer function is thus:

$$\frac{{H_1}^4(s)}{1 + k{H_1}^4(s)} = \frac{\left(\frac{a}{s+a}\right)^4}{1 + k\left(\frac{a}{s+a}\right)^4} = \frac{a^4}{(s+a)^4 + ka^4} \tag{2.64}$$

The root-locus equation in k is:

$$(s+a)^4 + ka^4 = 0 \tag{2.65}$$

Thus, the open-loop poles are, as expected, co-located on $s = -a$, and all the open-loop zeros are at $s \to \infty$. The root locus in $k > 0$ is thus an 'X' shape, centered on $s = -a$, and heading out towards $s \to \infty$ along the angles $\pm\pi/4$ and $\pm 3\pi/4$ (see Figure 2.28). These angles are such that when the two right-hand tracks hit the imaginary axis, they will hit at $s = \pm ja$. Thus the high-Q tuning of the filter is directly controlled by the open-loop pole locations, with no extra mapping necessary (except possibly an exponential frequency mapping to allow octaves to be spaced equally in the

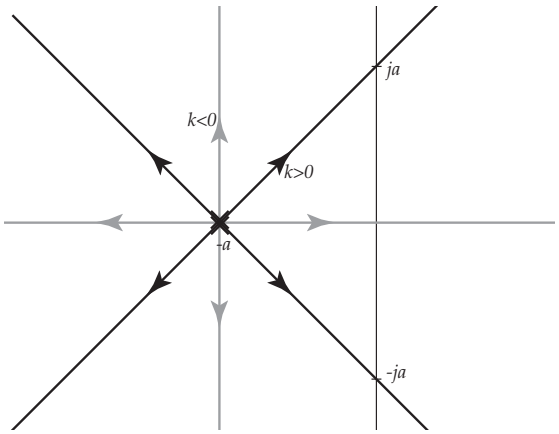


Figure 2.28: Continuous-Time Moog-Style Filter: Root Locus in k.

control domain). Further, the left-hand pair automatically moves to high damping, making the right-hand pair dominant (and hence being effectively the sole component in the determination of Q and f_c). Finally, the DC normalization of the open-loop filters causes the 180° point of the loop to always land at $|(a/(s+a))^4|_{s\to ja} = (1/\sqrt{2})^4 = 1/4$, so that the loop gain k when the tracks hit the imaginary axis is always 4, independent of a.

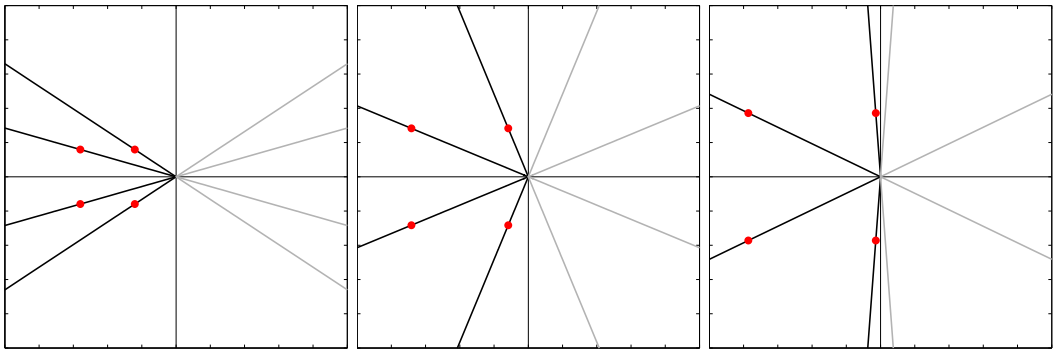


Figure 2.29: Continuous-Time Moog-Style Filter: Root Loci in a. Left: $k = 0.1$, Middle: $k = 1$, Right: $k = 3$. Gray: $a < 0$. Dots: roots for a particular value of a.

The scaling is also such that when $0 < k < 4$, the angle of the poles from the imaginary axis is independent of a, and only a function of k. This can be seen in the root-loci in a (Figure 2.29).

The locus in a is 4^{th}-order in a:

$$(s+a)^4 + ka^4 = s^4 + 4s^3a + 6s^2a^2 + 4sa^3 + a^4(1+k) = 0 \tag{2.66}$$

See Section A.1.3 on p. 264 for discussion of higher-order root-loci. A 4^{th}-order locus has three "middle numerators" (naming them in reference to the equation $D(s)+kN_1(s)+k^2N_2(s)+k^3N_3(s)+ ... + k^nN_n(s) = 0$ as "N_1" through "N_3", with D and N_4 being the "normal" open-loop pole zero polynomials). We get an interesting pattern in the roots of the polynomials:

$$\begin{aligned} D \text{ has roots at:} &\quad 0,0,0,0 \\ N_1 \text{ has roots at:} &\quad 0,0,0,\infty \\ N_2 \text{ has roots at:} &\quad 0,0,\infty,\infty \\ N_3 \text{ has roots at:} &\quad 0,\infty,\infty,\infty \\ N_4 \text{ has roots at:} &\quad \infty,\infty,\infty,\infty \end{aligned}$$

We saw a similar pattern in the State-Variable filters (both continuous-time and discrete-time), where D had double roots on DC, and N_1 also had a root on DC. The locus in a therefore starts at $s = 0$. For $a > 0$, four root tracks head into the left-half plane at angles controlled by k. These tracks continue as straight lines all the way out to $s \to \infty$. For $a < 0$, the roots follow the reflection of the $a > 0$ about the imaginary axis (or the negative... since we're dealing with real coefficients, the negative of a locus is the same as the reflection about the imaginary axis): the poles start at $s = 0$ and head towards $s \to \infty$ along the reflected angles.

As discussed in Section 1.2.1, the pole-location definition of Q in continuous-time is based on the angle of the pole from the imaginary axis. Since the pole tracks in the a-locus are rays originating at $s = 0$, all points along them have the same angle with the imaginary axis, and hence all have the same Q. Therefore, we have a visualization of the fact that this filter is a constant-Q filter.

2.5.3 "Polygon Filters"

Hutchins used the Moog-style filter as a template and described a family of filters he called "Polygon Filters" [117] [19] [115] [116] . These filters are of the same form, but with different numbers of onepole filters, ranging from two upwards. As such, their first-order locus in k is a "burst" of trajectories from $s = -a$, their angles equally distributed about the plane. k-loci for various numbers of onepole filters are shown in Figure 2.30. He named these filters "polygon" filters due to the fact that the roots for any particular value of k form the vertices of an N-sided regular polygon centered on $s = -a$.

In [117] [115], and [116], Hutchins explores the properties of these filters. We mention them here due to their direct relationship to the continuous-time Moog-style filter, which can be classified as an $N = 4$ polygon filter. All the polygon filters have independent f_c and Q controls. Further, the right-most poles will always be a pair (if the feedback is of the right sign, regardless of the number of poles) The difference between the filters being the overall filter order (and hence the

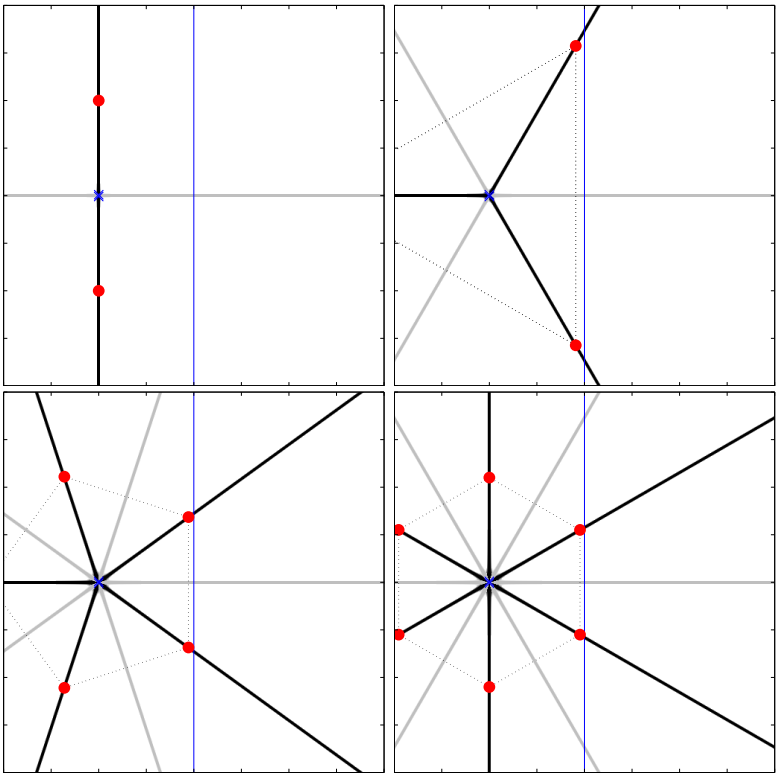


Figure 2.30: "Polygon Filters". Extensions of the Moog-Filter idea to other numbers of onepole filters. Shown: 2, 3, 5, and 6 filters in the loop. Gray: $k < 0$. The dots show the pole locations for a particular k, and demonstrate the polygons which give rise to the name for this class of filters.

final amplitude rolloff rate), and the amount of dominance of the "primary" poles (as when N gets large, the next pair of poles starts getting pretty close to the imaginary axis as well).

Finally, out of general interest, the gain at which the primary poles hit the imaginary axis is:

$$\left|1 + j \tan\left(\frac{\pi}{N}\right)\right|^N = \sec^N\left(\frac{\pi}{N}\right)$$

2.5.4 Other Analyses

There are a few other behaviors in Moog-style filters that are of interest.

Output Combinations It is possible to pick off at the input (or output) of each one-pole filter and combine these outputs to achieve various gross filter shapes (like highpass, bandpass, etc.). Hutchins talks about this for the op-amp version of the analog filter [113], and the results translate rather straightforwardly to the digital filter, as written about in [111] and [277]. It

is a fun mental exercise to compare this output-combination method to that used in the state-variable filter. It is an open problem, though, to see how the use of different open-loop zero locations in some of the designs of the next chapter may effect the choice of combination gains.

Non-coincident open-loop poles Hutchins also looked into this for an analog Moog-style filter in [114]. He notes that significantly higher loop gains are required to reach the imaginary axis when the poles are not coincident. However, there is quite a bit of exploration that can still be done here to understand this situation, answering questions like how the gain changes with the pole separations, etc. Hutchins also analyzed a particular pole arrangement $(s_0 - 1, s_1 = -(2^x), s_2 = -(2^{2x}), s_3 = -(2^{3x}))$,[14] and it would be useful to see how the gain characteristics might change with different patterns, or if it is mainly just an issue of gross separation.

[14]For this configuration, he calculated the gain for infinite Q to be: $3.8(105)^x$.

Chapter 3

Designing Filters Approximating Constant Q

3.1 Philosophy

The continuous-time versions of two filters we are concentrating on, the State-Variable Filter and the Moog-style resonant lowpass filter, have a property which is generally called "Constant Q." This means that as the filter's corner/center frequency is swept to various values, the Q of the filter's resonance stays constant. This behavior is generally accepted to be perceptually preferable, as the hearing system appears to judge resonance much as it does pitch, such that at a particular center frequency, a certain bandwidth sounds equivalent to double the bandwidth at double the center frequency. As such, we desire to design digital versions of these filters with the same property.

3.1.1 Problem Definition

Our Holy Grail: Find an inexpensive digital filter topology which is intrinsically constant Q, or even better, constant Q with independent (linear or log) f_c and Q controls. i.e., a filter which implements constant Q variation purely due to its architecture, with no required $f_c \rightarrow$ coef. or $Q \rightarrow$ coef. mappings.

If that is not achievable, find a topology whereby the filter and the mappings are as simple as possible.

A consequence of Constant Q is a property we will call "separability of Q and f_c," meaning that the frequency control can be moved without affecting the Q of the filter, and vice versa.[1] In designs where we attempt to control these features as cheaply as possible, separability can be effected. As

[1] We will also refer to such a filter a having "independent" controls. The two terms are effectively interchangeable from the standpoint of this thesis.

such, we must be vigilant to keep separability in mind, either as an inherent capability of the filter, or as an additional "separation" calculation which is as cheap as possible.

For the purposes of this thesis, we will not look at digital filters in general, rather we will concentrate mainly on filters similar in topology to the digital state-variable filter of Chamberlin [37] and the Moog four-pole Voltage-controlled Filter [173] (in particular, four stages in the feed-forward path). As such, there are certainly good designs in other directions which we will not consider. For example, we should note that Massie showed [162] that the 2nd-order ladder implementing an EQ section also has separate freq and Q controls, not unlike the Chamberlin filter.

3.1.2 Requirements, Assumptions, Restrictions

This is an overview of the design philosophy that was used for this research.

Wide View

As Cheap as Possible This comes from the realities of implementation and current industry philosophies. A consumer electronics company views Moore's Law in terms of being able to pack more processing into a 50-cent chip than before (or better, a 5-cent chip), not in terms of a top-of-the-line general-purpose processor being able to do more than the previous generation. As such, algorithm implementations must always be as cheap as possible in such situations. The power available on a desktop computer is almost an unreachable luxury. Now, of course, much work is done on such over-powered systems, but even then, the more that can be done at once in realtime, the more choices the user has. As such, there is always a push for absolute optimization. Practical variations on this rule which are taken seriously are: "As cheap as we can get away with" and "Only model some part of the system if it is necessary, and then only if it is audible." Basically, these describe the concept that if a behavior of the system is not noticeable don't waste cycles modeling it. Of course, on a deeper level, these bring up issues with how one can prove something is audible or not, and eventually lead to psychoacoustics. Work along these lines would definitely be assisted by continued research into measuring (and predicting the audibility of differences in) model fidelity for use in deciding just how much not to model. However, this thesis does not attempt to solve that problem, and such decisions are made according to the author's opinions.

No Oversampling At its basis, this follows from the previous rule: oversampling immediately multiplies the cost of an algorithm, if not the whole system (depending on where the boundaries of the oversampling are put). But also, oversampling can be used to ignore some interesting problems. To put it bluntly, quite often "oversampling makes things too easy." Many of the issues that will be dealt with in this chapter are nearly non-issues with oversampling on the order of 3x, 4x or more. In the field of modeling continuous-time behavior, if you can

restrict the region of interest to some region near $z = 1$, then even the simplest discretizations work just fine. As such, we conjecture that some of the deeper questions about what it means to discretize a system can simply be ignored when oversampling. Further, we note that a design that works well non-oversampled would work all the better oversampled.[2] Finally, quite a few systems simply must be implemented without oversampling, for any number of reasons, so there is definitely interesting and useful work that can be done in the non-oversampled world. Therefore, for this thesis, the choice of f_s is not to be made by the algorithm design, but rather by standard system considerations (standards, cost, fidelity, etc.). As such, we can assume that f_s might get as low as 22050 Hz or 24000 Hz in an inexpensive synthesis system, but we will assume as a default that f_s is either 44100 Hz or 48000 Hz.

Linear This will be discussed in more detail later on in the chapter, but much like with oversampling, there are situations where linear design is sufficient. First, implementing nonlinearity in such filters almost always requires oversampling, though many systems are made with nonlinearity and no oversampling and just live with whatever aliasing might occur. Second, virtual analog filters are of a type that generally have a "small-signal" region that is effectively linear. As such, there must be a good linear design within a good nonlinear design. As such, this thesis only looks at the linear part of the design, on the assumption that the reader can add their own nonlinear implementation as desired.[3] Finally, as mentioned in the discussion of oversampling, there are still interesting problems to be tackled in the purely linear domain.

Understanding over Modeling A major thrust of this research was not to come up with perfect models of particular analog filters, but rather to understand the design of constant-Q digital filters which exhibit inexpensive parameter mappings.[4] It is believed that once one understands the issues, modeling can be more effectively accomplished later.

More specifics

There are several more specific points about the design directions we will use:

- When there are more than two poles (i.e., in the Moog-style filters), we assume that one pair will be dominant in normal operation, and hence the non-dominant poles can effectively be ignored in analysis and discussion of tuning and Q. The inner poles are considered mainly to contribute to the rolloff slope of the filter. Note that some of the less well-behaved discretizations of the analog Moog-style filter do not maintain the dominance relation of the poles from

[2] Of course, one can make a good argument, based on the "only model what is audible" argument, that if for some reason oversampling is otherwise required, one should take advantage of it and ruthlessly ignore behaviors outside the audible range.

[3] However, as mentioned in the introductory chapter, this thesis restricts the space of filter topologies to those largely similar to the analog topologies. This is done on the assumption that those topologies will be most likely to work as desired when saturation nonlinearities are added, as the such implementations are expected to be the most common.

[4] In particular, to understand how Constant-Q behavior can arise in a simple digital filter topology.

the s plane (the forward difference being the most obvious, wherein at the top half of the one-pole tuning range, the "left-hand" poles actually reach the unit circle first). As such, these discretizations are usually rejected because of this assumption.

- As discussed in Chapter 1, we use the pole-angle definition of Q. Poles in the z plane are translated to the s plane via $z = e^{sT}$ and the pole-angle definition is calculated from that location.

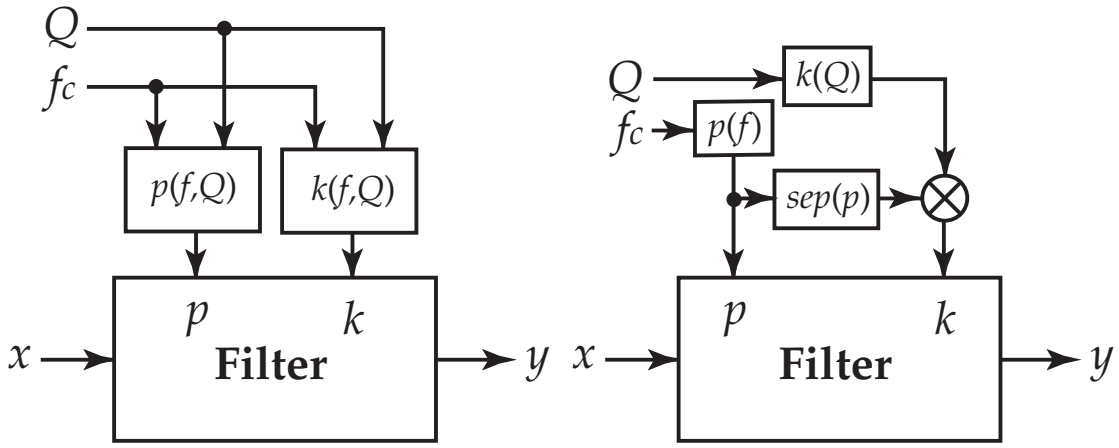


Figure 3.1: Using tuning tables for 2-parameter filters. Left: 2-D tables for each coefficient, which can theoretically tune perfectly. Right: three 1-D tables, which are assumed to be sufficient.

- As a benchmark for defining "cheap" control mappings, we attempt to be less expensive than three 1-D lookup tables (one each for the two major controls, plus a separation table to reduce dependence between the controls). It is assumed that the use of 2-D lookup tables ("LUT"s), as in Figure 3.1, can tune perfectly (assuming two major controls, usually f_c and Q), so the cost of one 2-D lookup table per coefficient will be defined as "too expensive". We theorize that the three 1-D LUTs just mentioned are sufficient for normal operation (we do some experiments to verify this in Section 3.3.2). As such, we define that the use of three LUTs, while acceptable, is still considered expensive, and we desire to design a system that can be controlled with less mapping cost than that. The three LUTs are assumed usually to be:
  - $f_c \rightarrow p$ at the unit circle (i.e., at infinite Q), where p is the tuning coefficient.
  - $Q \rightarrow k$, where k is the resonance coefficient. This LUT may not be necessary in all designs, depending on how accurately Q needs to be specified.

  - $p \rightarrow k$ at infinite Q. This table will be called the "separation table" (or "separation function" or "separation curve" if it is not a table), and is used to try to make the control of f_c and Q separate in situations where the filter is not inherently separable. It is primarily the viability of using this table that is under scrutiny in the experiments of Section 3.3.2.

  Note that the Chamberlin state-variable filter trivially meets this goal, as its controls are already effectively separate (except at very low Q, and most users don't seem to mind), so it does not need a separation table, and thus can use 2 LUTs at worst. Thus this issue is mostly for the Moog-style designs. Still, in the discussion of extending the state-variable filter usage range beyond $f_s/6$, we use the same guidelines (i.e., attempt to not add more cost than about that of a single LUT).

- For the purposes of this thesis, a 2^{nd}- or 3^{rd}-order polynomial is taken to be about the limit for a polynomial to be "cheaper than a 1-D LUT". Of course, the exact comparison is highly dependent on the architecture (and/or processor) in which a filter is being implemented, together with the specifics of the surrounding code, and the capability of the optimizing compiler/programmer. Since those are not the same for all cases, we use these definitions as vague guidelines.

- **On Modulatability** One of the main reasons for desiring cheap design calculations is that within the Virtual Analog paradigm, it is not unheard-of to modulate algorithm parameters with audio signals (f_c in particular, but Q modulation is sometimes implemented as well). Thus, the algorithm should expect to have different values of its controls every sample. Of course, in situations where one can prove that modulation and control changes will happen much less often, one could get away with more expensive calculations when the parameters change (as long as the system has sufficient buffering to absorb such bursts of calculation). However, this thesis takes as an assumption that control mapping calculations have to be done every sample.

  - **Precise vs. Imprecise Modulatability** If modulation of a particular parameter is mainly a "special effect", it may be possible to split the parameter into a cheap combination of a more expensive exact calculation (which would be done only when changing the gross settings) and a very inexpensive offset calculation which would happen every sample. For example, a parameter mapping such as $y = f(x)$ could be implemented as

    $$y = f(x_{gross}) + x_{mod} f'(x_{gross})$$

    This would not be useful if the modulation had to be precise, but may be fine for special-effect modulation.

– **Local vs. Global Modulatability** In the realm of Precise Modulatability, there is a further classification: local vs. global. With global modulatability, the modulation is assumed to be able to visit any value within the whole range of the parameter at any instant. Thus the precision of the modulation must be exact. However, if the modulation is known to only cover a range that is much smaller than the full range of the controller, then split mappings such as the one mentioned previously may be able to stay within acceptable precision limits. Appendix D discusses such a scheme.

This thesis assumes precise global modulatability as a design requirement. As noted in the introductory chapter, this emphasis on modulation is motivated mainly as an optimization issue, to allow smooth parameter variations and not preclude audio-rate modulation. There is no design requirement for (nor expectation that) the modulated filler will match continuous-time filter behaviors during fast modulation. During fast modulation, all of the analyses used in the design no longer apply. It is known that time variation causes different topologies which are equivalent in the absence of time variation to behave quite differently. As such, this design philosophy stressed modulatability only to allow it to be implemented efficiently, not to attempt to match any particular behavior in the presence of fast modulation.

A further design philosophy in this thesis is to make the digital topology approximate the analog topology. In other words, we do not discretize the filter and implement the discretization in some direct-form topology. Rather, we attempt to match the topology. Usually, this involves discretizing subsets of the filter as opposed to digitizing the filter as an indivisible whole. For example, in the Moog-style filter, this means discretizing the component onepole filters. Similarly, Chamberlin and Dutilleux apparently used this philosophy with their discretizations of the state-variable filter.

Dataflow Analog Filters The fact that we can actually discretize these two filters in this way, still keeping the basic topology and having the filters still work reasonably, implies that their topologies are somehow advantageous for digitization. Why might this be so? A likely explanation may be that both of these filters are essentially what we might call "dataflow" filters.

In general analog circuits, there are loading effects at junctions between components or subcircuits, such that the actual behavior of a circuit must be determined by deriving a set of equations represented by all of the various component/subcircuite interconnections and solving for the actual behavior. As such, if we were to represent a general circuit as a block diagram of connected subcircuits, we would not be able to simply assume that each block takes its input voltage and outputs a particular voltage based on some pre-defined behavior of the block, independent of other blocks. Instead, the properties of the blocks on either side of a block would also affect its operation (as, in turn, would all the blocks they were connected to). Again, the blocks would simply define

constraints on the final system behavior, which can only be finally determined by solving the simultaneous set of equations that all the constraints define. Thus, one could not look at the inputs and outputs of blocks in such a block diagram and interpret them as representing the direction of information flow within the circuit.

On the other hand, in digital simulation, the most straightforward behavior to implement is for any particular block to have a specific causal relationship from its inputs to its outputs, completely independent of what it is connected to. In such systems, the output of a block purely is caused by its input values (and any internal states), and not othersise affected by what it is attched to. In such a situation, one can interpret a block diagram as representing a flow of information (data), from block intputs to block outputs. Such systems can de considered "dataflow" systems, since there is a well-defined flow of data through the system.

Now, it is possible to make analog blocks which strongly approximate such behavior: give them very low output impedances, and very high input impedances. Thus, in a connection of two such blocks ($A \rightarrow B$), block B will have negligible effect on the behavior of block A. Using blocks of this type, one can construct circuits which can be considered dataflow circuits, since each block will effectively implement a causal input/output relation.[5], denoting that each state is only effected by those states "behind" it in the data flow order. In such circuits, information (signal values) would flow forward through the blocks much as in a numerical simulation.

Circuit discretization can be thought of as consisting to two sub-problems:

- Deriving the system equations.
- Discretizing the system equations.

If a circuit is a dataflow type, then the first sub-problem can be simplified to that of deriving the equations for each block separately and then arriving at the system equations by simple combination of the sub-block equations (usually as simple as routing through their inputs and outputs). In essence, the dataflow circuit is already in a topology that can be used for numerical simulation. Further, in a simulation implemented using the same block diagram, one can physically interpret the numerical values between blocks as equivalent to the signal values (usually voltages) at the same locations (modulo any time-discretization artifacts). Thus, situations such as saturation in the analog system can be directly modeled in the digital system (at least at the boudaries between blocks), with aliasing being the only thing to worry about. In non-dataflow circuits, the effects of nonlinearities within the circuit (even memoryless ones) can be significantly more complicated to model and may enter into the digitized version in much more complicated ways. Thus, even though the filters described in this thesis do not attempt to model analog saturation, it is expected that the topologies can have saturation added into them in a very straightforward (and probably physically correct) manner.

[5]One can loosely say that in a state-space realization, that the main matrix would be banded upper-triangular (or block upper triangular), as long as the states are assigned in the natural order of the data flow

Now, things are a bit more complicated in block diagrams with loops. In a continuous-time system, a dataflow loop can contain no delay, and as such, the loop will implement the solution to an equation not contained in the blocks themselves. For example, let's put a feedback loop around a continuous-time block implementing $Y(s) = F(s)X(s)$, summing with an input $U(s)$:

$$\begin{aligned} X(s) &= U(s) + Y(s) \\ X(s) &= U(s) + F(s)X(s) \\ X(s)(1 - F(s)) &= U(s) \\ X(s) &= U(s)\frac{1}{1 - F(s)} \\ Y(s) &= U(s)\frac{F(s)}{1 - F(s)} \end{aligned}$$

As such, the transfer function $F/(1 + F)$ has been implemented, even though it is not directly implemented by any of the blocks. The loop has (nearly) instantaneously calculated the solution to some constraint. [6] On the surface, this is a similar problem as before, where we are back to the system behavior being defined via the solution to some equation rather than appearing as a direct implementation of the block equations. However, the above equation manipulation can be done in z as well as in s:

$$\begin{aligned} X(z) &= U(z) + Y(z) \\ X(z) &= U(z) + F(z)X(z) \\ X(z)(1 - F(z)) &= U(z) \\ X(z) &= U(z)\frac{1}{1 - F(z)} \\ Y(z) &= U(z)\frac{F(z)}{1 - F(z)} \end{aligned}$$

In other words, discrete-time loops can also implement such equation solutions, the only issue being time-discretizaion artifacts. These appear partly as a consequence of the fact that a loop cannot be implemented in discrete time without at least one sample of delay in the loop (unless one solves the above equation ahead of time and implements the solution, but then one usually does not implement it in a form like the original loop). Therefore, a discrete-time loop is limited in its ability to exactly model a continuous-time loop by the fact that it requires some delay in the loop (extra linear phase). One does note, however, that at low frequencies, the effect of a single-sample delay becomes very small (and at DC the effect goes away). This essentially restates the fact that

[6] Now, at very high frequencies (many MHz) and correspondingly very short time scales, this concept breaks down, as electromagnetic wave propagation times start becoming non-negligible within a circuit, and the circuit can be viewed as an interconnected set of waveguides (wires and circuit components), such that one can actually think in terms of information flowing between elements. However, at audio frequencies, such effects happen so quickly as to be effectively instantaneous.

most useful discretizations are most accurate near DC, and tend to become less acurate away from DC.

In summary, an analog circuit that is a dataflow type is expected to be discretizable using the same topology as the analog circuit (at the level ot the impedance-isolated sub-circuits), and as such the discretization can be done on each sub-circuit (i.e. each block-diagram block) rather than having to discretize the whole system as a single indivisible block. Further, the values of simulation signals between blocks should be physically comparable to the equivalaent circuit signals (usually voltages), and operations such as saturation should be applicable in a physically-correct manner.

The state-variable filter is usually implemented using op-amp-based integrators, and hence is a dataflow filter (since the op-amp stages have the required input/output impedance relationship). Similarly, It has been shown (by Hutchins among others [118][113]) that the Moog-style filter can be implemented with op-amp-based one-pole filters, and hence the required impedance relationships between blocks, such that it too can be considered a dataflow filter.

3.2 Revisiting the Digital State-Variable Filter

Chamberlin, Dutilleux and Dattorro have already done the basic research on designing this filter type, so we will not try to re-design it here.[7] Instead, we will look into trying to make the filter a bit more usable.

3.2.1 Extending the usable frequency range

Due to instability in certain parts of the coefficient range, the rule of thumb for using the filter is to limit the cutoff frequency to $f_s/6$.[8] We will explore how and when the filter goes unstable, and attempt to find ways to extend the frequencies which can be used.[9]

Let's look at the properties of the filter. Remember, the denominator is

$$z^2 + (\mathrm{ff}^2 + \mathrm{ff\ qq} - 2)z + (1 - \mathrm{ff\ qq}).$$

- When the pole are complex, the pole radius is $\sqrt{1 - \mathrm{ff\ qq}}$. Since both coefficients are positive in standard usage (we saw in the root loci that the filter is unstable in much of the negative ranges of each coefficient, and the design formulas only give positive coefficients for normal f_c and Q), this means that the filter will not go unstable by poles leaving the unit circle while complex. In other words, when the filter goes unstable, it is due to a pole crossing through $z = -1$ or $z = 1$. Finally, the loci tell us that it will be the left real pole crossing $z = -1$.

[7] However, a few variant filters will be explored as a consequence of noting various things in the course of root-locus analysis.

[8] Dattorro also argues that design approximations start breaking down past that frequency [57]

[9] the author would like to note the help of Sean Costello in testing some of these extensions.

- The left real pole hits $z = -1$ at $\text{ff} = \sqrt{\text{qq}^2 + 4} - \text{qq}$ (or equivalently when $\text{qq} = \frac{\text{ff}^2 - 4}{-2\text{ff}}$). Thus the filter is stable when $\text{ff} < \sqrt{\text{qq}^2 + 4} - \text{qq}$.
- The poles are co-located (i.e., just leaving the real axis or just meeting on the real axis), when $b^2 - 4ac = 0 \Rightarrow \text{ff}^2((\text{ff} + \text{qq})^2 - 4)) = 0 \Rightarrow \text{ff} = 0,\ 2 - \text{qq},$ and $-2 - \text{qq}$. Since we only use $\text{ff} \geq 0$, that leaves $\text{ff} = 0$ and $\text{ff} = 2 - \text{qq}$. The first case is when the poles are at DC, and the second case is when the poles rejoin the real axis.
- One of the poles is at $z = 0$ when $\text{ff} = 1/\text{qq}$. This may be of use for the $\text{qq} > 1$ cases, where the teardrop comes together to the right of $z = 0$, hence the pole that will eventually go unstable has to pass through $z = 0$, and we might want to stop it there.

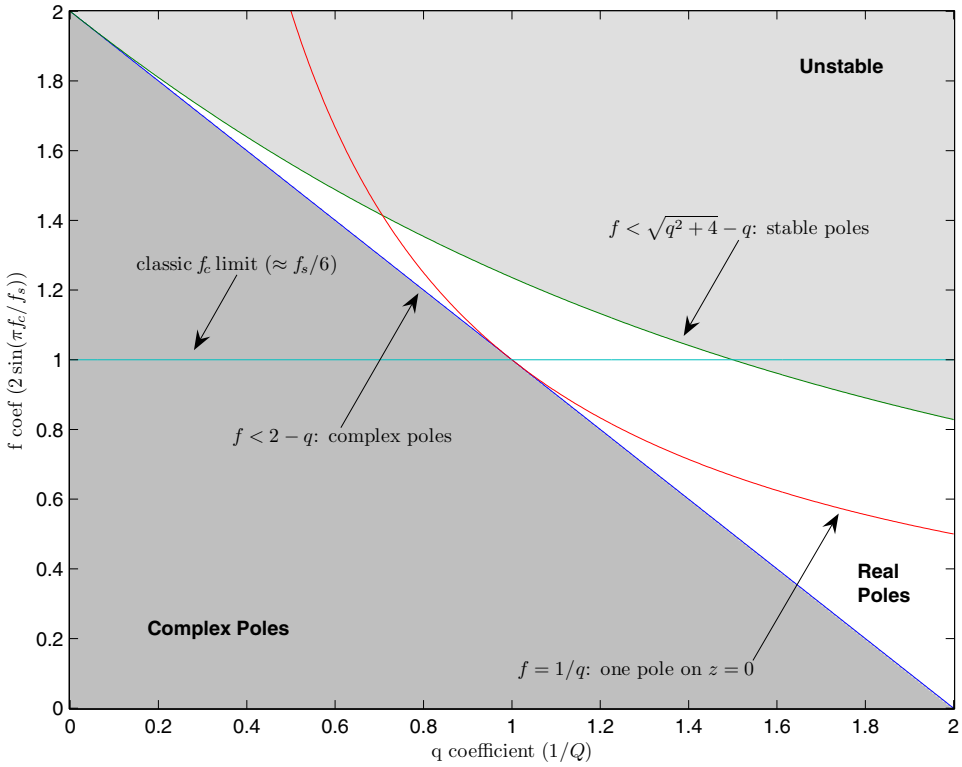


Figure 3.2: State-Variable Filter Contours

This gives us the following contours in the (ff, qq) plane, which are shown in Figure 3.2:

- $\text{ff} = 1$: Classic $f_c < f_s/6$ limit.
- $\text{ff} = \sqrt{\text{qq}^2 + 4} - \text{qq}$: Stability limit

- ff = 2 – qq: Poles rejoin real axis.
- ff = 1/qq: One pole on $z = 0$.

We can see in Figure 3.2 that the $f_s/6$ limit is not a perfect limit — if the user uses the filter in the range 1.5 < qq < 2, then the filter will go unstable before reaching $f_s/6$. Therefore, some writers place a lower upper limit on qq (i.e., a high lower limit on Q) of $\sqrt{2}$, 1.5, or even 1.0 ([57]). However, it is known from experience that some users prefer to use the filter all the way up to qq = 2.

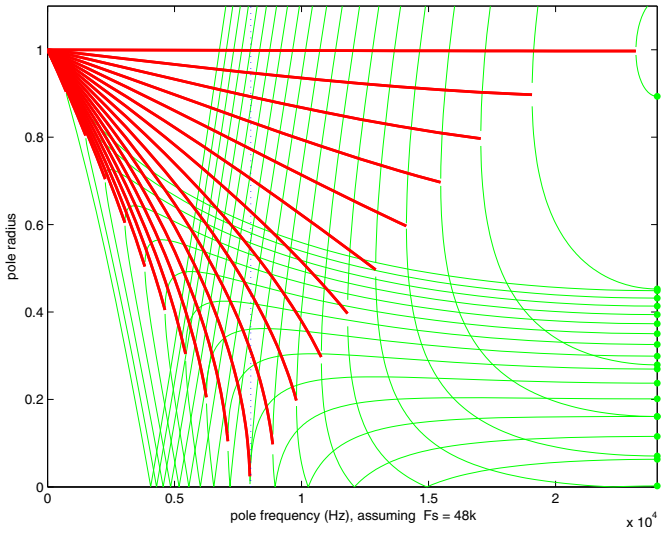


Figure 3.3: State-Variable Filter Pole Radii vs. tuning coefficient ff, various Q settings between 1/2 and very high Q. Thick lines: complex poles. Thin lines: real poles. The standard $f_s/6$ limit is denoted by the dotted vertical line.

We will be using a new plot to analyze these proposed frequency-extension methods. The plot shows pole radius versus frequency coefficient for a number of Q tracks in the state-variable filter. Figure 3.3 shows these tracks for an unmodified filter. Note that plots show when the tracks consist of complex poles vs. real poles by line thickness (thick representing complex poles), and that when a track goes to real poles, two thin tracks emerge from it, one for each pole (as the poles are no longer at the same distance from the origin). In this diagram, we see that the instability occurs as a pole track crosses upward through the radius=1.0 line. High Q's correspond to nearly horizontal thick lines, whereas lower Q's have more steeply descending thick lines. One can note that very low Q's transition to real poles rather quickly (i.e., at low values of ff), as can be corroborated by earlier analyses. A track "bounces off the bottom" when the pole crosses through $z = 0$, usually from $\mathrm{Re}(z) > 0$ to $\mathrm{Re}(z) < 0$. The minimum in the thick lines occurs at ff = 1, where the complex poles come together right on $z = 0$, and hence reach a radius of zero.

Clamp ff **to** 2 – qq**?**

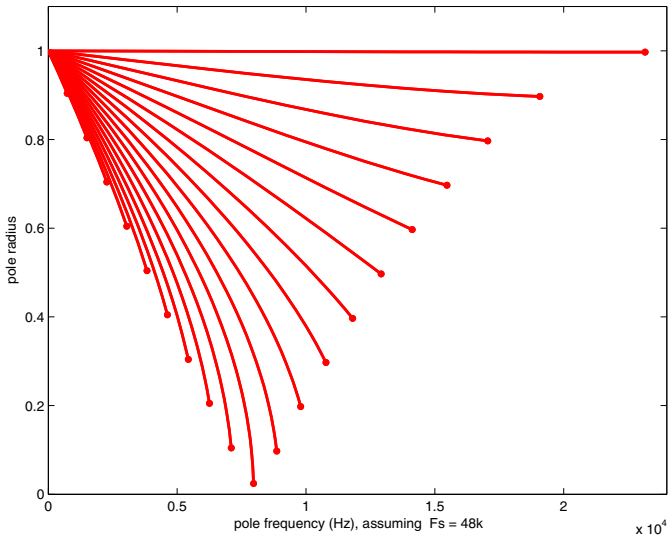


Figure 3.4: State-Variable Filter Pole Radii: ff clamped to 2 – qq (i.e., stop the poles when they hit the real axis). Note that once clamped, the pole stops moving. This is designated by the dot at the end of the trace. Unfortunately, this clamping doesn't allow any motion of ff at all when qq = 2

A possible strategy is to simply limit the range of ff by clamping it to some boundary, to keep the filter from moving into a region of "bad behavior".

The most obvious boundary to clamp at is where the poles transition to the real axis, as one might expect that complex poles are what define a resonant filter. This complex/real boundary occurs at ff = 2 – qq, and the poles are thus complex in the range $0 \leq \text{ff} \leq 2 - \text{qq}$. Therefore, we define the following clamping:

$$\text{ff}_{actual} = \min(\text{ff}, 2 - \text{qq}) \tag{3.1}$$

Unfortunately, this clamping has a major drawback in everyday use: when Q gets very low (below 1.0), the effective range of ff becomes severely limited, to the point where, at $Q = 1/2$, qq = 2, and ff is limited to the range $0 \leq \text{ff} \leq 0$. In other words, the filter cannot be tuned at all. In the lowpass mode in particular, the filter effectively blocks most frequencies, and the filter appears to be silent for any value of ff that the user may attempt. The filter appears to not do anything, or be somehow broken.

This behavior can be explained by the fact that the "teardrop" of complex pole locations contracts down to nothing when qq = 2, so there are no non-real pole locations.

Now, if one restricts the range of Q to $Q \geq 1$, then qq < 1 and 2 – qq works just fine as a limit for ff (see Figure 3.2), especially because qq = 1 is the largest qq for which the "teardrop" still encircles

$z = 0$. At $Q = 1$, ff can range up to ff = 1, the traditional $f_s/6$ limit. Note that in [57], Dattorro partially explains the ff ≤ 1 limit by defining the maximum useful value of qq as 1.

However, many users of the state-variable filter routinely use $Q = 1/2$ as their lower Q limit, rather than $Q = 1$ or $Q = \sqrt{2}/2$. It is because of this range of operation that limiting the poles to be complex becomes non-viable.

We must note, then, that it is common to use the filter in a range where both poles are real. This can present a philosophical problem, as it is natural to consider a "resonant" filter, such as the types being explored in this thesis, as being defined in terms of parameters that only have meaning if the poles are complex. For example, in [57], Dattorro explicitly assumes that the poles must be complex while deriving the ranges for ff and qq. Do the design formulas for ff and qq still have meaning when the poles become real? Unfortunately, this will have to be answered by later research. An open question is whether the filters that are implemented when the poles become real have any unexpected drawbacks. Do the filter shapes extrapolate well the trend from before the poles go real? Do they do so smoothly? We can know, simply from the fact that users appear to be comfortable with the way the filter acts in this region, that the behavior cannot be "too bad." In fact, as noted in the discussion of Q definitions in Chapter 1, there are probably many users for whom the real-pole "sound" of the state-variable filter defines how a filter *should* sound at very low Q.

In summary, due to the ranges of coefficients in accepted usage, restricting the poles to only move within the complex-pole region is not viable for the Chamberlin state-variable filter.

Clamp near unit circle?

Another obvious boundary is near the stability boundary:

$$\text{ff} < \sqrt{\text{qq}^2 + 4} - \text{qq} - \varepsilon \tag{3.2}$$

However, one can reason that this is probably not a useful clipping boundary, as it will tend to let the filter get too close to instability (unless one sets ε particularly high, in which case, it would start interfering with other filter behaviors). In fact, if clipped right on the boundary, the filter would end up merely quasi-stable during the whole time that the unclamped ff was beyond the boundary, which would not be a "nice" mode of operation.

Hybrid clamping

One may be tempted to clamp ff at 1/qq, which is where one pole crosses through $z = 0$. This would seem to be a good compromise between letting the poles go real, but not letting them get too close to the unit circle. However, this interpretation only works at very low Q, where the poles come together to the right of $z = 0$. In that situation, the pole that will go unstable does indeed have

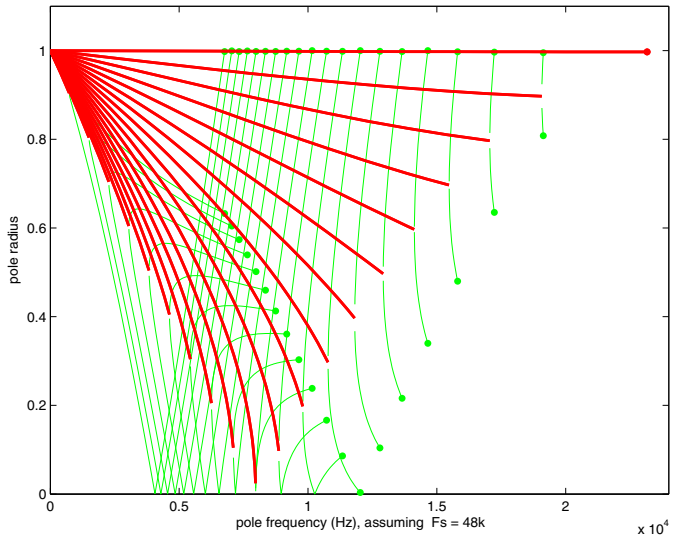


Figure 3.5: State-Variable Filter Pole Radii: ff clamped to $\sqrt{qq^2 + 4} - qq$ (the stability boundary)

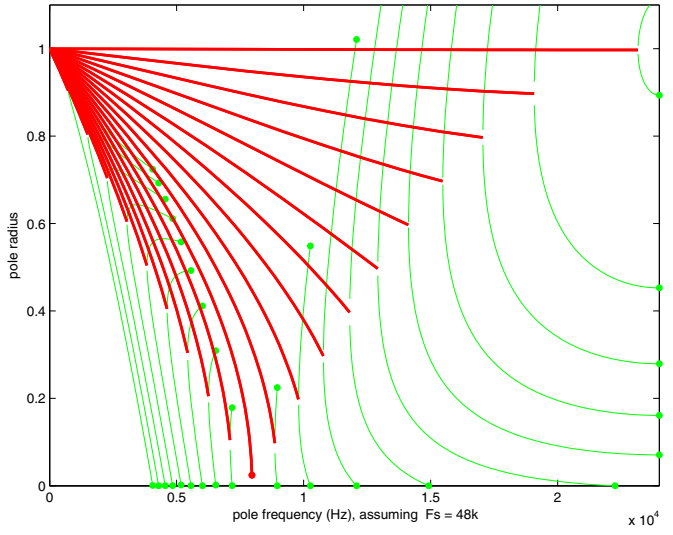


Figure 3.6: State-Variable Filter Pole Radii: ff clamped to $1/qq$ (when a pole hits $z = 0$). The problem is we don't know which pole has hit $z = 0$, and for high Q, it is the wrong pole, so we still get instability.

to cross through $z = 0$ to reach the unit circle, and such a clamping rule would be of use. However, for $Q > 1$, the poles come together to the left of $z = 0$, so that the one that would cross through $z = 0$ is the right-hand pole, not the left-hand pole, which is the one which goes unstable. Thus, the clamping rule would do nothing to keep the filter stable in this situation.

One might combine clamping rules into a hybrid rule, which clamps differently depending on what range of Q the filter is in. A likely hybrid would be the following:

$$\begin{aligned} Q > 1: \quad & \mathrm{ff}_{actual} = \min(\mathrm{ff}, 2 - \mathrm{qq}) \\ Q \leq 1: \quad & \mathrm{ff}_{actual} = \min(\mathrm{ff}, 1/\mathrm{qq}) \end{aligned}$$

For high Q, the complex/real pole boundary is a fine boundary (it only has problems for $Q < 1$), and for low Q, the poles come together to the right of $z = 0$ and hence it is a useful boundary as well. Furthermore, these two boundaries touch at $Q = 1$, thus allowing a smooth transition between the two ranges. A drawback of this clamping, however, is the divide, but if $\mathrm{qq} = 1/Q$ is being performed anyways then $1/\mathrm{qq}$ already exists (i.e., Q), so clipping ff to Q may be fine. If, however, Q has been lost by runtime, then a divide becomes necessary again. Note that the comparison can be done without a divide, since ff and qq are both positive, so that the comparison $\mathrm{ff} > 1/\mathrm{qq}$ is equivalent to the comparison $(\mathrm{ff}\ \mathrm{qq}) > 1$. However, if clamping must be done, then the value $1/\mathrm{qq}$ still must be computed.

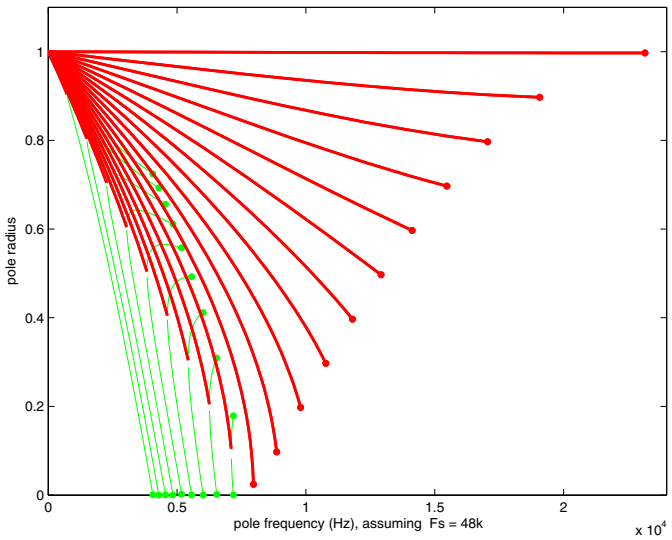


Figure 3.7: State-Variable Filter Pole Radii: Hybrid Clamping: ff clamped to $1/\mathrm{qq}$ when $\mathrm{qq} > 1$, and to $2 - \mathrm{qq}$ when $\mathrm{qq} < 1$.

Polynomial clamping

If one studies the shapes of the boundaries in Figure 3.2, one may wonder if there exists a function which lies in the real-poles region which may be inexpensive to compute, and hence act as a useful compromise clamping function. In fact, a 2nd-order polynomial in qq turns out to work pretty well. Figures 3.8 and 3.9 show the effects of the two clamping functions

$$\begin{aligned} \text{ff}_{actual} &= \min(\text{ff}, 0.2\text{qq}^2 - \text{qq} + 2) \\ \text{ff}_{actual} &= \min(\text{ff}, 0.15\text{qq}^2 - \text{qq} + 2) \end{aligned} \tag{3.3}$$

respectively. They also represent two points on a tradeoff continuum which the user or designer may explore: for the 0.2 version may be considered to get too close to unstable at low Q, or the 0.15 version may be considered to clamp "too early" and overly restrict the range of ff…

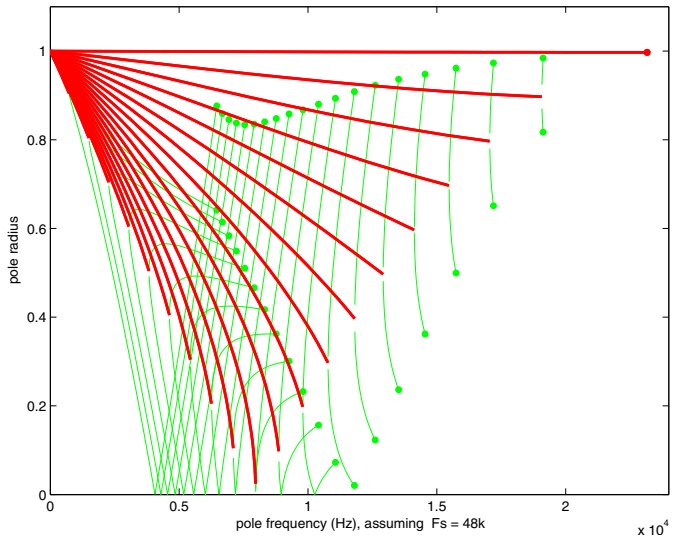


Figure 3.8: State-Variable Filter Pole Radii: ff clamped to $0.2\text{qq}^2 + 2 - \text{qq}$.

Frequency-extension wrapup

These various attempts at extending the useful range of the state-variable filter beyond $f_c < f_s/6$ primarily differ in where they clamp ff. Almost all of the methods clamp within the real-pole region (as it was seen that clamping at the complex/real boundary is too restrictive). Now, since we prefer to only define Q when the poles are complex (i.e., both poles have the same angle), any differences between these clamping methods take place outside the range of defined Q, and so it would not

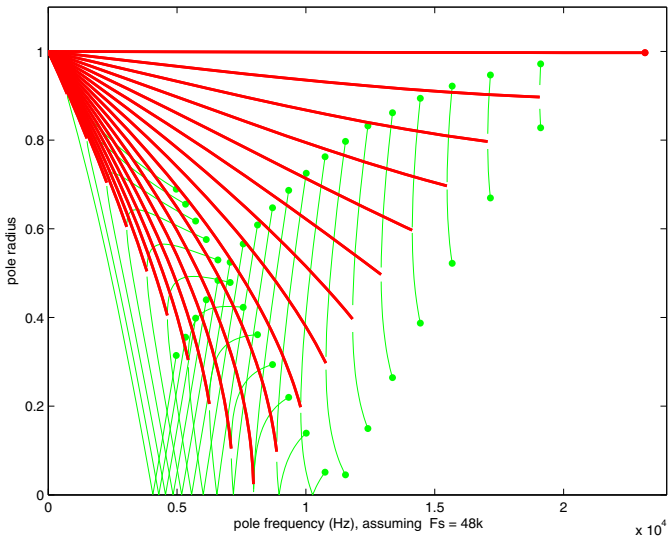


Figure 3.9: State-Variable Filter Pole Radii: ff clamped to $0.15qq^2 + 2 - qq$.

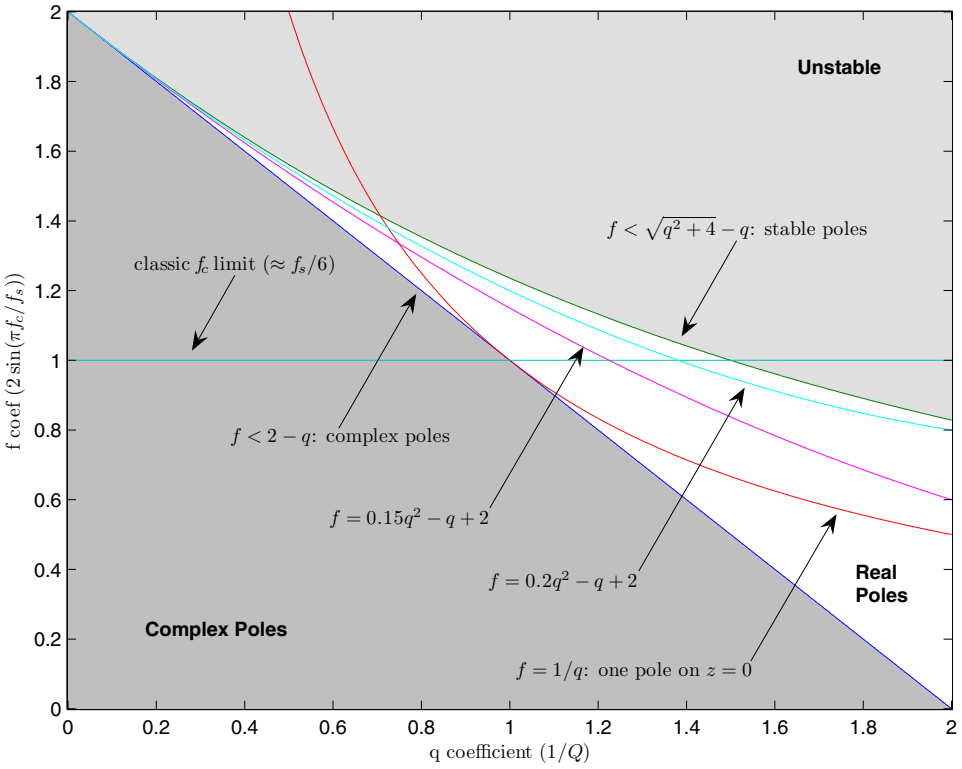


Figure 3.10: State-Variable Filter Contours, showing proposed ff-limiting polynomials

be of use to compare these methods using analyses such as a Q-vs-f_c plot, since they would all produce the same plot. All the "action" would take place outside the view of the plot.

Again, the fact that many users use the state-variable filter in regions where the poles are real, and still think of there being a Q in these regions, presents some philosophical quandaries for the filter designer. The designer cannot get away with just ignoring this region (as was attempted by clamping to 2 – qq, and was implied by Dattorro in [57]).

3.2.2 Deriving a SVF Variant From Locus Tracks

In Appendix A, Section A.2, we look at how one might derive the continuous-time state-variable-filter's locus equations directly from desired locus track shapes. Here, we will attempt to do the same thing in discrete time, and see if we come up with a filter similar to the discrete-time state-variable filter.

The ideal constant-Q tracks are

$$r = e^{-\alpha\theta}. \tag{3.4}$$

However, this definition is difficult to use, since we desire polynomials for the root locus, so we look at the series expansion:

$$r = 1 - \alpha\theta + \frac{1}{2}\alpha^2\theta^2 - \cdots \tag{3.5}$$

and noting that for much of the Q range (i.e. $Q >> 1$), α is rather small, use the approximation

$$r = 1 - \alpha\theta. \tag{3.6}$$

Where α is some function of Q (and thus only approximate constant Q at low frequencies or high Qs). This defines a spiral in the z plane which approximates the desired constant-Q track. The desired pole locations are thus

$$z_{0,1} = re^{\pm j\theta} = (1 - \alpha\theta)e^{\pm j\theta} \tag{3.7}$$

Plugging these into an idealized root-locus equation gives:

$$\begin{aligned} (z - z_0)(z - z_1) &= 0 \\ z^2 - zr(e^{j\theta} + e^{-j\theta}) + r^2 &= 0 \\ z^2 - 2z(1 - \alpha\theta)\cos(\theta) + (1 - \alpha\theta)^2 &= 0 \end{aligned} \tag{3.8}$$

At this point, we should recognize the standard formula for a complex-poles biquad denominator (i.e., $z^2 - 2r\cos(\theta)z + r^2$). We also realize that some sort of approximation will be necessary if we want to turn this into a polynomial with respect to θ (i.e., to have the filter coefficients be polynomials with respect to θ, as in the state-variable filter). Thus, we expand cosine into the first

few terms of its Taylor series approximation:

$$\cos(\theta) = 1 - \frac{\theta^2}{2} + \frac{\theta^4}{24} + \cdots \tag{3.9}$$

Substituting into the above, we get

$$(z-1)^2 + 2\alpha(z-1)\theta + (z+\alpha^2)\theta^2 - \alpha z\theta^3 - \frac{z\theta^4}{12} + \cdots = 0. \tag{3.10}$$

Truncating after the θ^2 term, and renaming θ to t (since it is no longer the actual pole angle), we get

$$(z-1)^2 + 2\alpha(z-1)t + (z+\alpha^2)t^2 = 0, \tag{3.11}$$

which has the familiar double-roots on $z = 1$ for $D(z)$, and the single root on $z = 1$ for $N_1(z)$. Interestingly, the finite root of $N_2(z)$ isn't right on zero, as it is in the Chamberlin form (see Section 2.4.4). As such, we already see a slight divergence from the Chamberlin form. If we collect in terms of powers of z, we get

$$z^2 + (t^2 + 2\alpha t - 2)z + (1 - \alpha t)^2, \tag{3.12}$$

which differs from the denominator of Chamberlin form in two places (if we replace qq with α and ff with t): first, in the z term, we have $2\alpha t$ rather than αt, and second, the final term is squared relative to Chamberlin form. As such, we should expect a different tuning formula, but in fact, we get the same, since we measure it at $r = 1$, hence $\alpha = 0$, in which case the z term appears equivalent to the z term of the Chamberlin form when qq = 0. Thus we get the same tuning formula:

$$t = \sqrt{2}\sqrt{1 - \cos(\theta)} = 2\sin(\theta/2) \tag{3.13}$$

If we look at the t-loci of this design (Figure 3.11), we see an interesting behavior at low Q: whereas the "teardrop" in the Chamberlin form contracts in towards $z = 1$ as qq $\to 2$, in this form the teardrop does not contract, and continues to wrap around $z = 0$ even at extremely low Q. This is a nice feature in comparison to how the Chamberlin filter's teardrop contracts down to nothing at low Q.

The stability and complex-poles contours for this design are:

$$|r_{1,2}| \leq 1 \quad \Rightarrow \quad t \leq \frac{2}{1+\alpha} \tag{3.14}$$

$$\text{Complex poles} \quad \Rightarrow \quad 0 < t < 2\left(\sqrt{\alpha^2+1} - \alpha\right) \tag{3.15}$$

(Note that these situations are also satisfied at $t = 0$ and at certain negative values of t, but those are considered outside the range of operation). These contours are shown in Figure 3.12. Unfortunately, unlike the Chamberlin case, these contours are not cheaply computed, and the au-

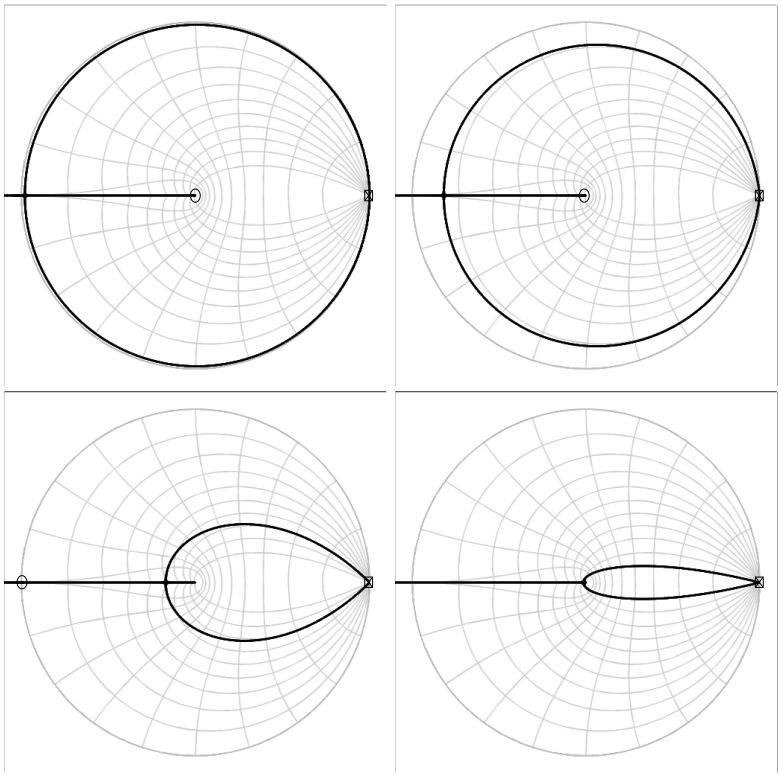


Figure 3.11: Root Loci in the t parameter, state-variable-filter variant. Left to right, top to bottom: $\alpha = 0.01, 0.1, 1.0, 4.0$. ($t > 0$)

thor has not yet found a cheap polynomial which fits well inside the real-poles region, as was done in Section 3.2.1 for the Chamberlin case. The use of a LUT for computing the boundary would appear to be the most likely solution for this design. Note that since the teardrop does wrap around $z = 0$, it should be valid to clamp t to the complex-poles boundary, unlike the Chamberlin case.

Also note that if, as is often done with the Chamberlin filter, one can still use ff = 1 ($f_c = f_s/6$) as a convenient upper limit on f_c if one does not want to compute the above boundaries, and as long as one is willing to stay in the region $0 < \alpha < 1$.

A block diagram for the variant

Here we attempt to derive a block diagram for this variant in a form similar to the Chamberlin form (Figure 2.13) diagram). As such, we will also change the coefficient names to match those of the Chamberlin form: $t \rightarrow$ ff, and $\alpha \rightarrow$ qq.

Using the Direct-Form two-pole filter's topology as a guide, replacing z^{-1} with k, and replacing the coefficients with D, N_1, and N_2, we can derive a "direct-form" implementation of a filter that

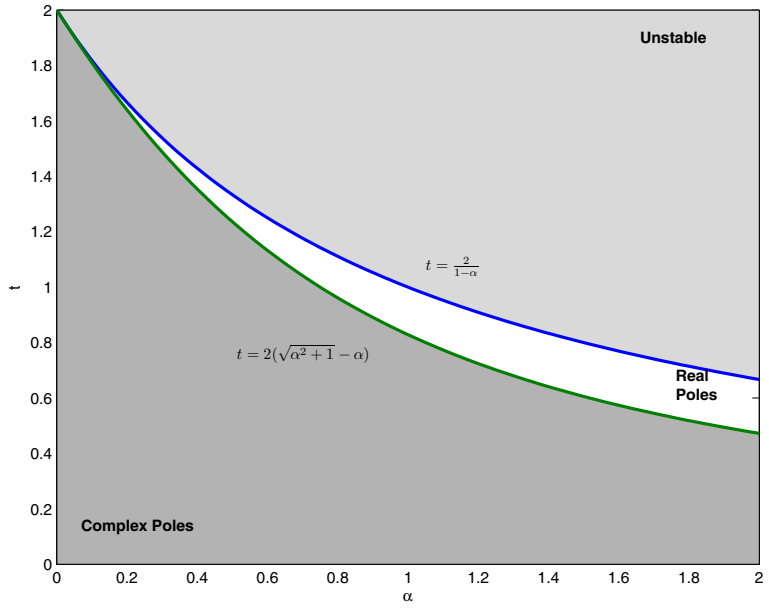


Figure 3.12: Complex-pole and stability boundaries of state-variable filter variant.

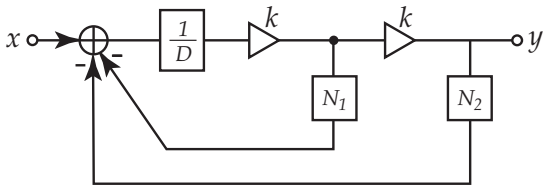


Figure 3.13: A "Direct-Form" implementation of a system which has $D + kN_1 + k^2N_2 = 0$ as its root locus.

will have a desired 2nd-order root locus in k: $D + kN_1 + k^2N_2 = 0$. Such a filter is shown in Figure 3.13.

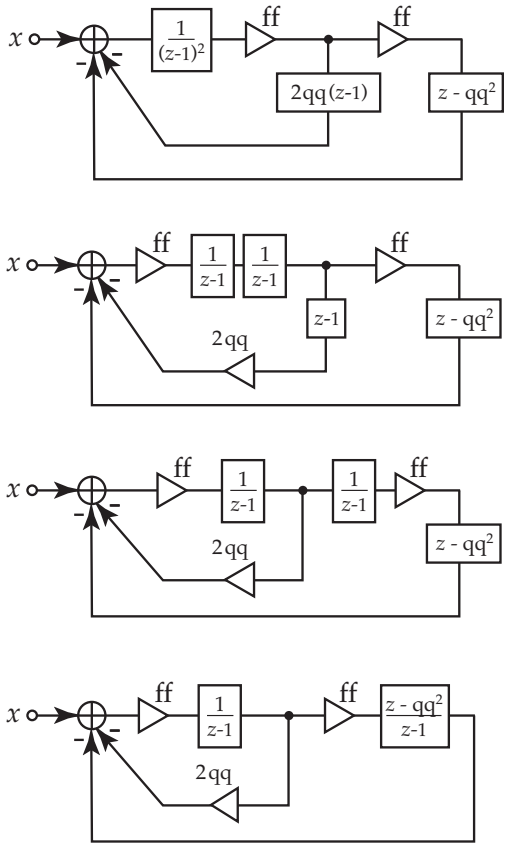


Figure 3.14: Initial steps in creating a block diagram for this state-variable-filter variant.

Now, replacing D with $(z-1)^2$, N_1 with $2\text{qq}(z-1)$, and N_2 with $(z+\text{qq}^2)$, and performing some block-diagram manipulations (Figure 3.14), we derive the form in Figure 3.15, which is quite close to Chamberlin form. Comparing this to Chamberlin form, we see two major differences: first, the inner feedback is 2qq rather than qq, and there is an additional FIR filtering in the feedback loop.

From here, one may wonder about where to pick off the outputs. Figure 3.16 shows most of the probable output locations for this filter. Their transfer functions are:

$$\begin{aligned} Y_7 &= X\frac{\text{ff}^2(z+\text{qq}^2)}{(z-1)^2 + 2\text{qq}\,\text{ff}\,(z-1) + \text{ff}^2(z+\text{qq}^2)} \\ &= \text{ff}^2(z+\text{qq}^2)(X/den) \end{aligned} \tag{3.16}$$

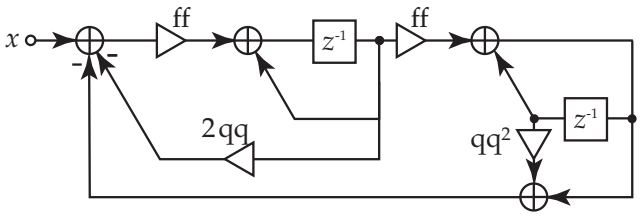


Figure 3.15: A block-diagram realization of this state-variable-filter variant.

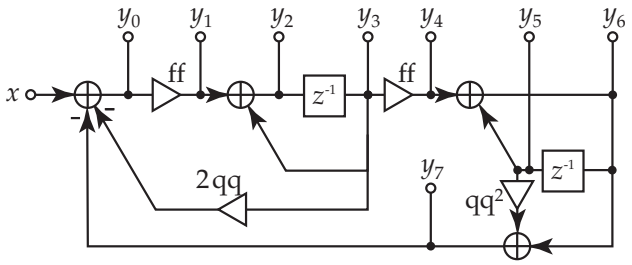


Figure 3.16: A block-diagram realization of this state-variable-filter variant, with possible outputs shown.

$$Y_4 = Y_7 \frac{z-1}{z+\mathrm{qq}^2} = \mathrm{ff}^2(z-1)(X/den) \tag{3.17}$$

$$Y_6 = Y_4 z/(z-1) = \mathrm{ff}^2 z(X/den) \tag{3.18}$$

$$Y_5 = z^{-1}Y_6 = \mathrm{ff}^2(X/den) \tag{3.19}$$

$$Y_3 = Y_4/\mathrm{ff} = \mathrm{ff}(z-1)(X/den) \tag{3.20}$$

$$Y_2 = zY_3 = \mathrm{ff}\, z(z-1)(X/den) \tag{3.21}$$

$$Y_1 = (z-1)Y_3 = \mathrm{ff}(z-1)^2(X/den) \tag{3.22}$$

$$Y_0 = Y_1/\mathrm{ff} = (z-1)^2(X/den) \tag{3.23}$$

Now, in comparison to the numerators of the Chamberlin filter outputs (Eqs 2.30-2.32), we can name the three outputs as follow:

$$Y_l = Y_6 = X\frac{\mathrm{ff}^2 z}{(z-1)^2 + 2\mathrm{qq}\,\mathrm{ff}\,(z-1) + \mathrm{ff}^2(z+\mathrm{qq}^2)} \tag{3.24}$$

$$Y_b = Y_2 = X\frac{\mathrm{ff}z(z-1)}{(z-1)^2 + 2\mathrm{qq}\,\mathrm{ff}\,(z-1) + \mathrm{ff}^2(z+\mathrm{qq}^2)} \tag{3.25}$$

$$Y_h = Y_0 = X\frac{(z-1)^2}{(z-1)^2 + 2\text{qq}\,\text{ff}\,(z-1) + \text{ff}^2(z+\text{qq}^2)} \tag{3.26}$$

And the difference equations for the filter are as follows:

$$y_l(n) = y_l(n-1) + \text{ff}\,y_b(n-1) \tag{3.27}$$

$$y_h(n) = x(n) - y_l(n) - \text{qq}^2 y_l(n-1) - 2\text{qq}\,y_b(n-1) \tag{3.28}$$

$$y_b(n) = \text{ff}\,y_h(n) + y_b(n-1) \tag{3.29}$$

Note that they differ from the Chamberlin form only in the later terms of $y_h(n)$.

Further analysis

Interestingly, qq in this filter is related to Q slightly differently than in the Chamberlin filter:

$$\text{qq} \approx \frac{1}{2Q} \tag{3.30}$$

This factor of two was first noticed experimentally. As with the Chamberlin filter, there is a breakdown in this approximation at very low Q (see Figure 1.3)

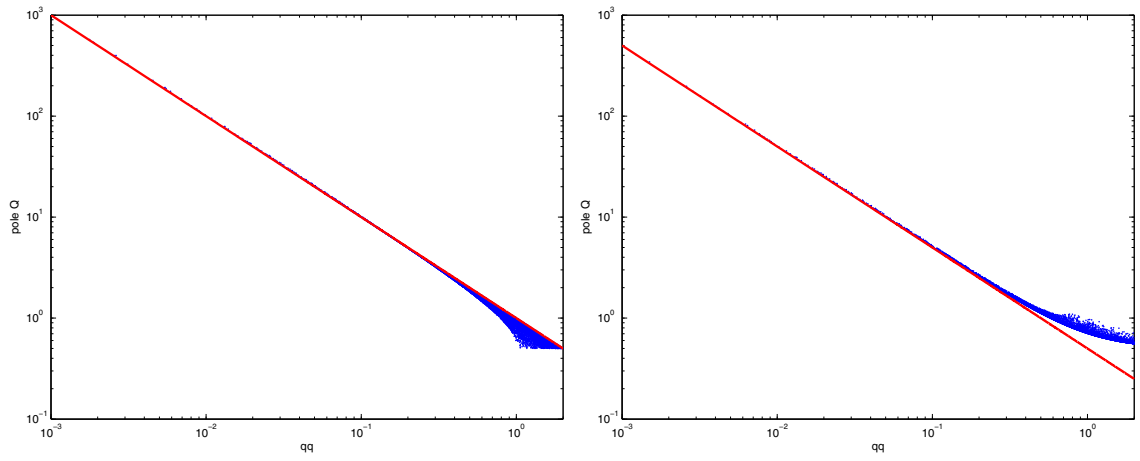


Figure 3.17: Measured pole Q vs. qq for random qq and ff values (only plotting complex poles). Left: Chamberlin Filter, line: 1/qq. Right: Variation, line: 0.5/qq.

But is it any better?

Because the Chamberlin filter is so inexpensive and does work extremely well over a range of f_c and Q that many find acceptable, the fact that this filter is actually slightly more expensive probably

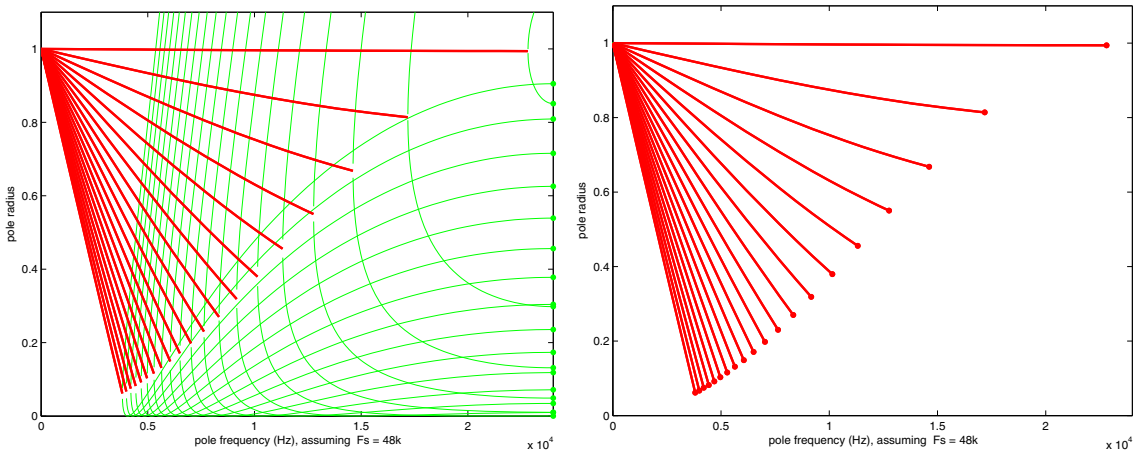


Figure 3.18: State-Variable Variation Pole Radii: Left: no clamping. Right: ff clamped to where the poles hit the real axis.

outweighs any advantages it might have. This filter is therefore primarily an example of how one might design a state-variable-like filter directly from approximations of its desired root tracks. The primary advantage of this form is that the "teardrop" of the ff locus always wraps around $z = 0$, rather than contracting in to $z = 1$ as in the Chamberlin form. From some viewpoints, this is a more reasonable pole track. It also allows the filter to be more usefully constrained to the range where the poles are complex, which was not possible in the Chamberlin form.

Finally, this design is not yet fully complete. There are still scaling issues to be figured out in the outputs, for example. Figure 3.19 shows representative lowpass, bandpass, and highpass responses for various frequencies and Qs of this filter vs. the Chamberlin form, and at low Q values, the filter gains get a little off. Of course, comparing against the Chamberlin form, which itself has issues at very low Q, may not be the best comparison. Still, this shows that there is still work to be done if this form is to become a fully usable filter type.

Higher Order?

What happens if we truncate the series in Equation 3.10 at a higher order in θ? This would theoretically give a closer approximation to $\cos(\theta)$ at the cost of a more expensive computation (and a higher-order locus). The 3rd- and 4th-order root locus equations can be extracted trivially from Equation 3.10: $N_3(z) = -\alpha z$, and $N_4(z) = -z/12$. Rearranged into a single z polynomial, these forms are:

$$\begin{aligned} z^2 + (-\alpha t^3 + t^2 + 2\alpha t - 2)z + (1 - \alpha t)^2 &= 0 \\ z^2 + \left(-\frac{1}{12}t^4 - \alpha t^3 + t^2 + 2\alpha t - 2\right)z + (1 - \alpha t)^2 &= 0 \end{aligned}$$

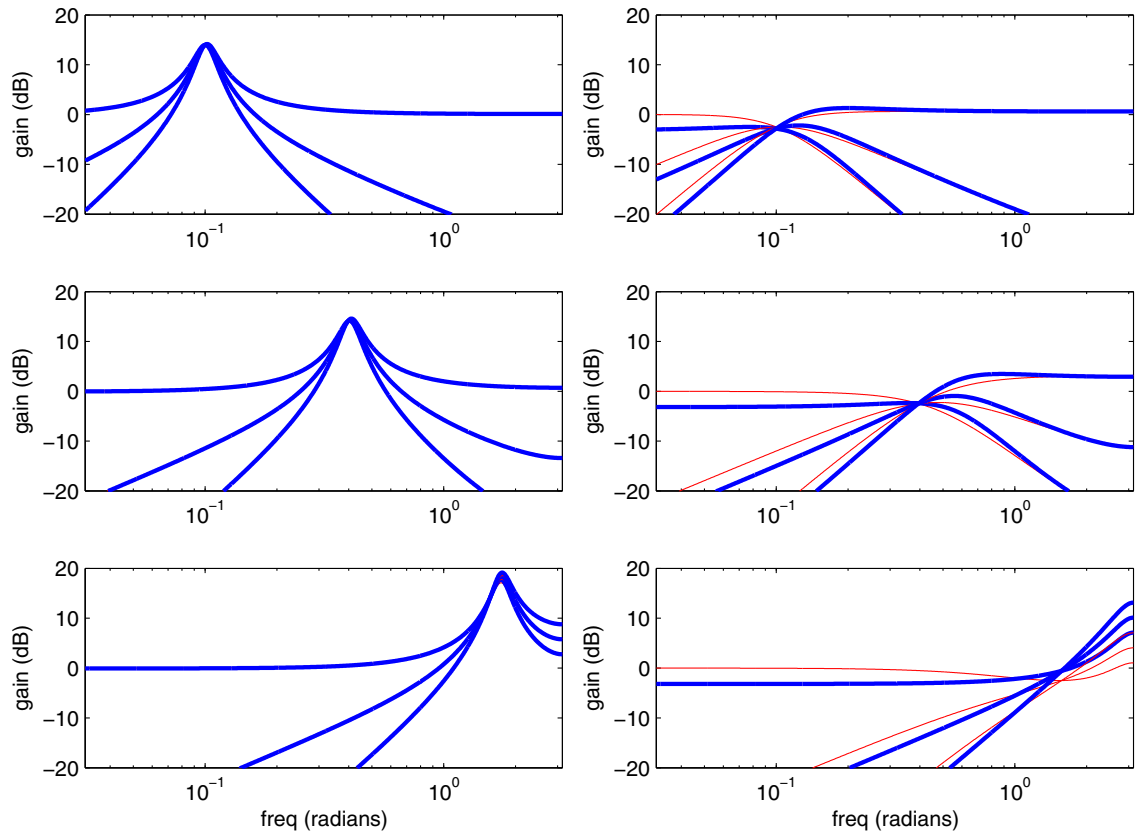


Figure 3.19: Comparing highpass, bandpass, lowpass frequency responses of this variation against those of the Chamberlin form. Thick lines: the variation. Thin lines: Chamberlin Form. Left: Q=5, Right: Q=0.75

As expected, the differences are just in the z coefficient (where the cosine term was), simply adding more terms of the series approximation of $-2(1-\alpha\theta)\cos(\theta)$. For 3rd-order, we add $N_3(z) = -\alpha z$, which has a root on $z = 0$. The loci are particularly nice looking (Figure 3.20). Note, in comparison to the 2nd-order loci in t (Figure 3.11), which tend to drift to higher Q at high frequencies (not unlike the Chamberlin form), these 3rd-order loci start drifting to lower Q at high frequency when α gets past 0.3 or so. Also, unlike the 2nd-order loci, these loci actually exhibit the behavior of the tracks "curling inward" near the real axis. The complex-pole boundary is also rather straightforward:

$$\alpha < 0.5: \quad 0 < t < 2$$
$$\alpha > 0.5: \quad 0 < t < 1/\alpha$$

(although the divide isn't necessarily cheap).

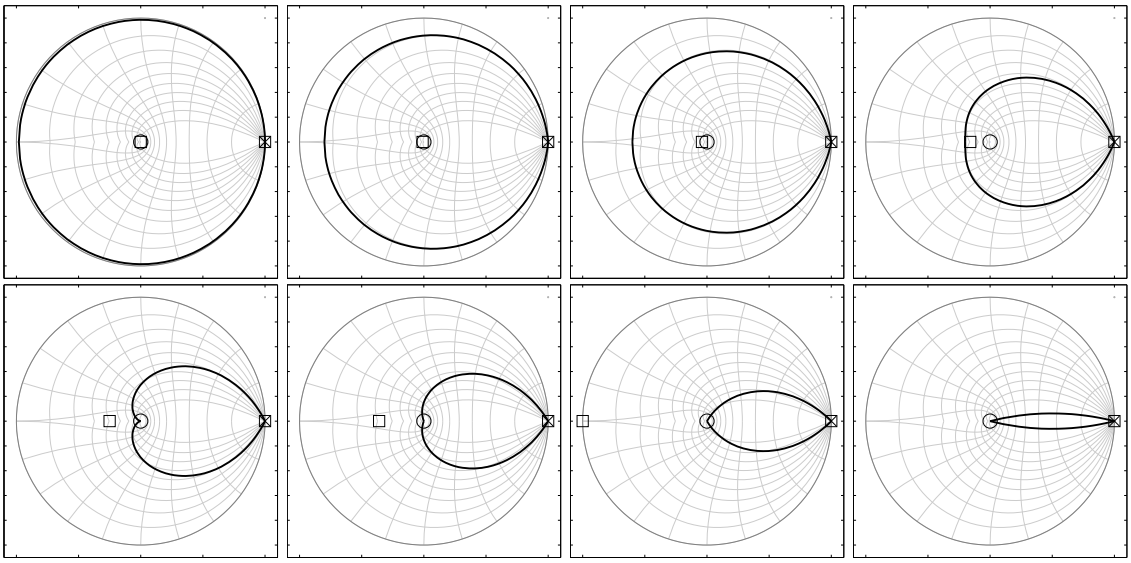


Figure 3.20: State-variable filter variant, 3rd-order in t. Left to right, top to bottom: $\alpha = 0.01, 0.1, 0.2, 0.4, 0.5, 0.6, 1.0, 4.0$. For $\alpha < 0.5, 0 < t < 2$, for $\alpha > 0.5, 0 < t < 1/\alpha$.

The 4th-order approximation of the cosine is not quite as "nice". We can see in Figure 3.21 that the locus, while following the constant-Q contours even more closely, does not actually reach the real axis near $f_s/2$, instead curving back around somewhere through the frequency range, making this variation on the design less interesting, except at low frequencies, where simpler designs are sufficient.

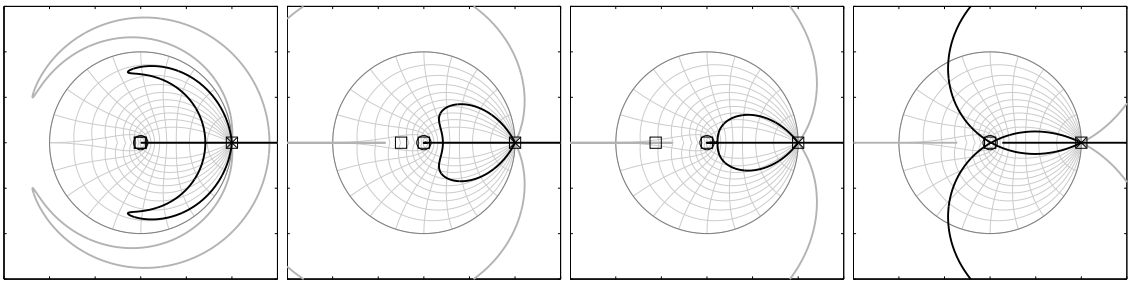


Figure 3.21: State-variable filter variant, 4th-order in t. Left to right: $\alpha = 0.1, 0.5, 0.75, 2.0$. ($-10 < t < 10$)

These higher-order variants were presented simply as explorations of further variations. However, the nice loci of the third-order cosine approximation suggest that there may be some useful further research in working out how to turn that design into a workable filter topology.

3.2.3 Derivation of SVF-like filter from Circle RL filter

The Type-1 Circle filter (Figure 2.1) looks quite similar to the Chamberlin State-variable filter. It is not a constant-Q filter, however. Interestingly, if we try to turn it into a constant-Q filter, we end up with a form not that unlike the Chamberlin form. First, however, we must note a useful method for designing tuning coefficients, which shows up in filters such as the state-variable filter.

The $\cos \rightarrow \sin$ method

As may have become obvious, it is extremely common for 2nd-order pole angles to be related to their coefficient by a cosine or one-minus-cosine relationship. This can be understood intuitively by the fact that root locations are very sensitive to their coefficients at "breakaways" (root locus situations where a pair of poles on the real axis leave the real axis and become complex), such that the poles' distances from the real axis (and hence their angle) move very quickly with changes in the coefficient. Hence the coefficient must change very slowly to achieve desired changes in the pole angle (i.e., the tuning curve must have slope near zero, which is the case in $1 - \cos(x)$ tuning relationships). Unfortunately, this means that a table lookup is probably necessary to get even gross tuning. However, we can make a very important observation from standard trigonometric identities:

$$\sqrt{2(1 - \cos(\theta))} = 2\sin(\theta/2) \tag{3.31}$$

Next we note that if we split a scale coefficient in a block diagram into two separate scale coefficients in series, then to get the same original scaling each should implement the square-root of that scaling. Therefore, if the original implemented $2(1 - \cos(\theta))$, then the split coefficients implement $2\sin(\theta/2)$. The important feature of this change is that $\sin(\theta) \approx \theta$ for small θ.[10] Since at common sample rates, much musical filtering uses pole frequencies that are quite small compared to the sampling rate. This means that one may often be able to get away with using coef $= \theta$ as a very good approximation. Therefore, a pretty ugly tuning mapping is turned into a pretty nice tuning mapping simply by splitting a coefficient into two parts. Now, if we look at the Chamberlin state-variable filter, and several other similar second-order filters, we can see that the tuning coefficient is actually in the system twice, and does actually implement the $2\sin(\theta/2)$ mapping.

This method is a very good one for a filter experimenter to keep in mind. We will use it here to help in turning the Type 1 Circle Filter into a form similar to the Chamberlin Form.

Modifying a Circle Filter

Note that since these really are just different forms of 2nd-order filter, we should theoretically be able to transform any form into any other form with sufficient block-diagram rearrangement and coefficient transformations. Hence, we could start with about any of the forms. An interested

[10] One can also note this by looking at the Taylor-series expansion: $2 - 2\cos(\theta) \approx 2 - 2(1 - \theta^2/2) = \theta^2$.

reader might try the same starting from one of the other forms discussed in this thesis, like the digital "X" filter (Figure 2.22). The real intent here is to note how similar these forms are by showing how few steps are necessary to get a similar filter (not identical), and further to present a couple interesting design methods and observations about the block diagrams which might help build intuition on how these filters operate (and in particular about how the Chamberlin form works).

As we have noted earlier, the Circle Filter forms (Figure 2.1) have much in common with the Chamberlin form (Figure 2.13). In particular, a loop around two first-order filters, one in backward-difference form and the other in forward-difference form. As such, one might ask: "If we modify the Circle filter to have some of the features of the state-variable filter (i.e., to be closer to constant-Q, etc), would we end up with a design that is like the Chamberlin form?

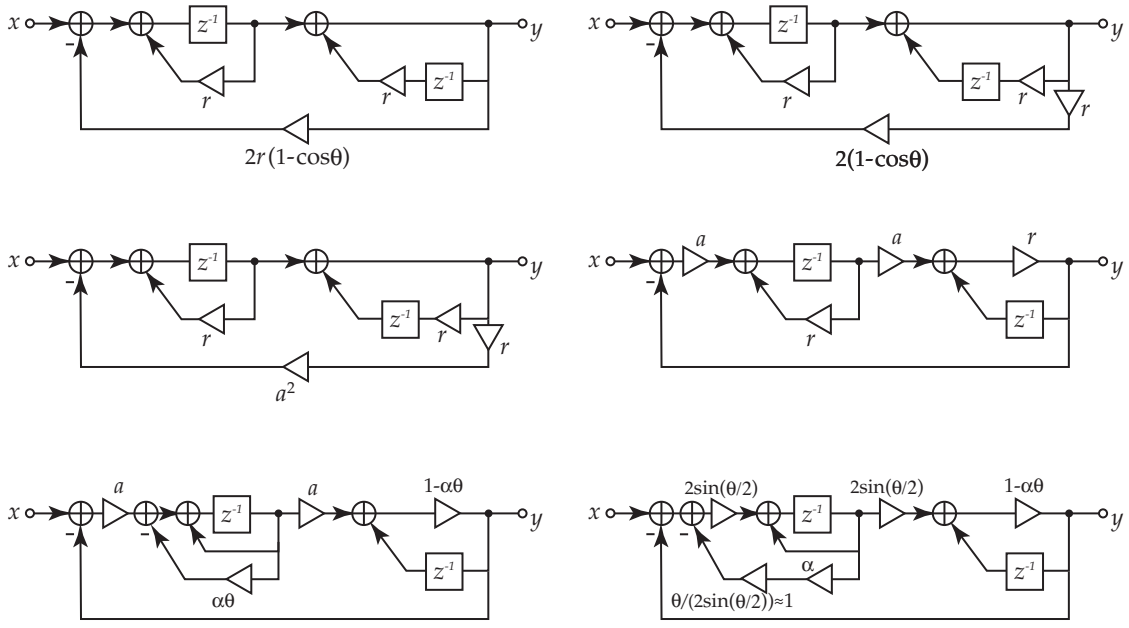


Figure 3.22: Modifying a Type 1 Circle Filter into a form similar to Chamberlin form

Let's try. Figure 3.22 shows a series of algebraic modifications to the Type-1 Circle Filter form, heading towards a Chamberlin-like form (right-to-left, top-to-bottom). First, we recall the coefficient design equations, noting that the outer feedback coefficient is $k = 2r(1 - \cos(\theta))$. If we split that coefficient into two coefficients, r, and $2(1 - \cos(\theta)$, we can combine the r with the feedback in the second onepole and rotate the scale around into the feedforward of that onepole (ignoring for now any issues with the output tap). This leaves us with the $2(1 - \cos(\theta))$ form that was just discussed, so we note that we will want to split it into two parts. Thus, let us rename the coefficient (for notational simplicity) to a^2. Now, split the coefficient into two scales by a, and then rotate them

back through the two onepole filters, placing a factor of a in front of each. This leaves the feedback gain at unity. This gets the filter looking even closer to the Chamberlin form, which also has the outer feedback at unity and has two frequency coefficient in the feedforward, placed in front of each onepole (roughly).

Next, we work on turning the filter into a constant-Q filter, as currently, it is more of a constant-radius filter. As in the previous section, we use the approximation for constant Q:

$$r = 1 - \alpha\theta \tag{3.32}$$

(which is most accurate at low frequencies or high Qs).

Side note: What if we end up with a coefficient is equal to θ (or has θ in it)? In a form like the Chamberlin form, this does not happen, θ only enters into the filter via the frequency coefficient $2\sin(\theta/2)$, and it would not be good to have to invert that to get back to θ. However, we note that if the filter is implemented with a frequency transformation, like ff = $f(\theta)$, then we *do* have θ available: the input to that transformation, and we can just use it. Note that this kind of method only works if the filter actually implements the transformation internally. If the filter is being used in "inexpensive mode", where the approximation $2\sin(\theta/2) \approx \theta$ is being used (i.e., no frequency mapping at all), then this method becomes less useful (or at least less accurate).

Back to the filter modification: Using the $r = 1 - \alpha\theta$ approximation allows us to split the first onepole into a two-loop structure with an internal forward-difference integrator and an external loop with a scaling of $\alpha\theta$. This is another step closer to the Chamberlin form, for which the first onepole is similarly split into two loops. Still, the Chamberlin form has the first frequency coefficient inside the outer of the new loops, and the feedback coefficient is not dependent on θ. Therefore, we push the a scaling forward through the sum, and in so doing push a scale factor of $1/a$ down into the feedback path, which we will combine with θ, which we have split from α. Thus we now have the a scale inside the inner feedback loop, as in Chamberlin form. To do so, we have created two scales in the inner feedback loop: α and θ/a. Now, α is directly related to Q, so in comparison to Chamberlin form, is exactly where we need it, but what about the θ/a? Here we note that $a = 2\sin(\theta/2)$. If we look at the series expansion of θ/a, we get:

$$\frac{\theta}{2\sin(\theta/2)} = 1 + \frac{\theta^2}{24} + \cdots \tag{3.33}$$

In other words, $\theta/a \approx 1$ up to 2^{nd}order, and we can thus approximate that scale away, leaving us with an inner feedback nearly identical to that of Chamberlin form.

In summary, we are left with a filter in a form almost exactly that of Chamberlin form, except for a scale of $1 - \alpha\theta$ in the second onepole loop (which, if we want, we can use "low frequency, high Q" arguments to approximate away as 1.0). As such, we have shown how one might attempt a derivation of a Chamberlin-like filter by attempting to turn a Circle filter into a constant-Q filter, and

using a few other hacks here and there (such as the $\sqrt{1-\cos(\theta)} \rightarrow 2\sin(\theta/2)$ method, and splitting a $1-x$ feedback into two feedback loops and hence turning a onepole filter into an integrator with feedback round it).

3.2.4 Deriving Chamberlin Form from 2nd-order Root-Locus Primitives

Now we approach the state-variable filter from a completely different direction. As will be mentioned in Appendix A, the author created some tools for interactively exploring root locus behavior, allowing the user to drag zeros and poles around and see the effect on the locus live. A 2^{nd}-order-locus version was created to explore 2nd-order loci, and in particular to help understand the effect of the relative scaling on N_1 and N_2 (see Section A.1.3), as it was realized that this scaling was inherent in how the "X" locus (Section 2.4.4) track angles were controlled.

Rather quickly after starting to use the explorer, a couple basic facts were easily noted:

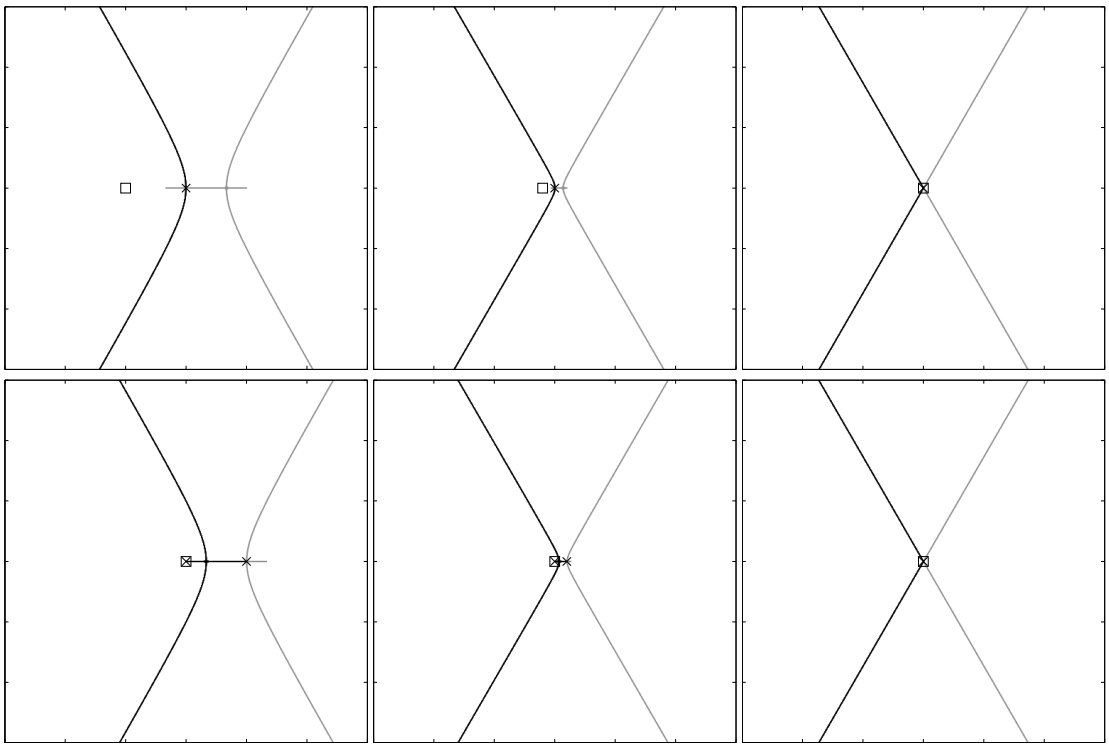


Figure 3.23: SVF-like 2^{nd}-order root-locus exploration. Top Row: moving N_1 root closer to the doubled D roots. Bottom Row: N_1 on top of one D root, moving other D root closer. Black: $k > 0$, grey: $k < 0$.

- The basic requirement for getting a locus track pair that leaves the real axis at an angle (and without extra poles heading into the opposite half plane) is for $D(s)$ to have double roots on

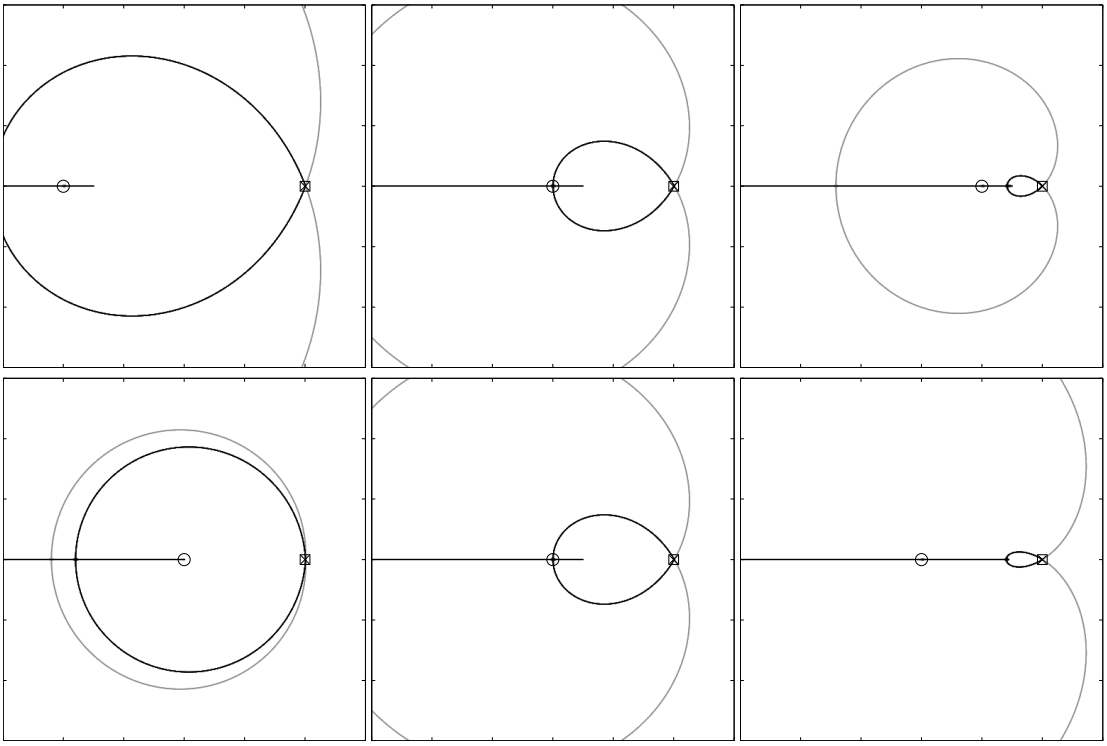


Figure 3.24: SVF-like 2nd-order root-locus exploration. Top Row: moving a finite N_2 root closer to the D roots. Bottom Row: changing the scale on N_1.

the real axis, and for $N_1(s)$ to have one root on the same location. If the N_1 root is away from the $D(s)$ roots, then the poles leave the real axis perpendicular to it, and only then bend to the desired angle. In other words, leaving the axis at an angle is a limiting behavior as the N_1 root approaches the D roots. Figure 3.23 shows the effects of various relative N_1 and D roots.

- The angle is definitely controlled by the relative gains of N_1 and N_2 (Figure 3.24).
- Giving N_2 a finite root bends the rays around the root to come back down to the real axis. More importantly, the shape they trace in so doing is quite reminiscent of the constant-Q curves in the z plane (except that they rejoin the real axis perpendicularly). Hence, the "teardrop" shape can be achieved this way (Figure 3.24).
- The size of the teardrop varies with the departure angle, such that the rejoining location pulls towards the zero (or even beyond it) as the angle moves from perpendicular to the real axis (the teardrop is a circle) towards parallel to it (the teardrop disappears)). This is also reminiscent of constant-Q shapes. By appropriately locating the N_2 root relative to the D roots, the overall scale can be controlled. To line up with the unit circle, the D roots and the N_1 root

go on $z = 1$, and the N_2 root goes on $z = 0$.

Therefore, one may be able to contemplate a discrete-time filter design based purely on these observations:

$$\begin{aligned} D(s) &= (z-1)^2 \\ N_1(s) &= \alpha(z-1) \\ N_2(s) &= z \end{aligned}$$

For a root-locus equation:

$$(z-1)^2 + \alpha(z-1)k + zk^2 = 0 \tag{3.34}$$

Where α is used to control the relative scaling of N_1 and N_2, for controlling the departure angle (and hence low-frequency Q), and the locus parameter k is a pole-angle (i.e., frequency) control. As noted previously (Section 3.2.2), we can turn this into a block diagram by analogy to a twopole filter (whose denominator is $1 + d_1z^{-1} + d_2z^{-2}$), replacing z^{-1} with k, and the coefficients with N_1 and N_2, and putting $1/D$ ahead of the first k). Figure 3.25 shows the block-diagram algebra which takes these equations and directly gives us a filter in Chamberlin form.

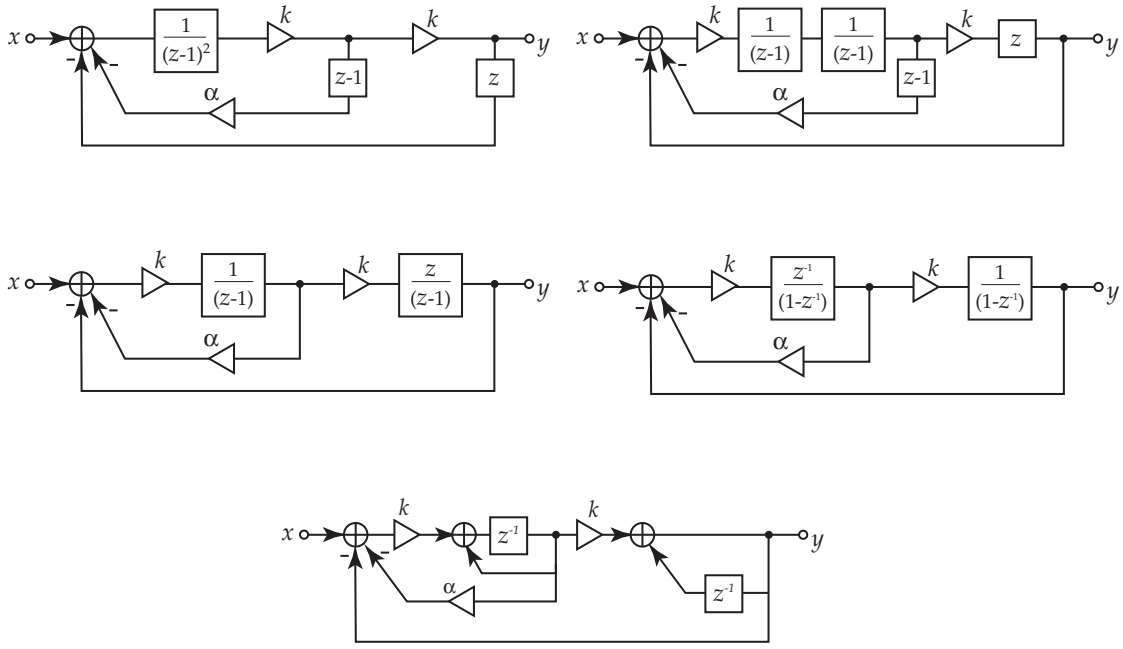


Figure 3.25: Deriving Chamberlin form from 2nd-order root-locus form.

Essentially, we have a derivation of the Chamberlin form from 2nd-order root-locus primitive

shapes, much as how the Circle Filters were derived from 1st-order primitives in Section 2.2.1.

3.2.5 Conclusions on the SVF section

Although the state-variable filter is quite useful as it stands, we have looked into a few small ways of possibly extending its useful range, and explored how similar filters may be derived from some other starting points.

3.3 Moog-Style Discrete-Time Filter Design

Here we look into designing digital filters in the style of the Moog voltage-controlled lowpass filter [173]. As hinted in the introduction to the chapter, the basic design concept will be to copy the topology of the analog filter, i.e., a cascade of first-order sections with feedback around the cascade. We will initially look at applying standard discretization techniques to the first-order filters (or in one case to the feedforward cascade as a whole). Later, we will move away from discretization and look at the digital filter directly, apart from any particular discretization of the analog filter, and explore variations of the filter directly.

A Review of the Design Philosophy

As discussed at the start of the chapter, the philosophy is to design a filter whose control mappings are as cheap as possible, in the non-oversampled, linear realm. Modulation is expected to update every sample, so that the control must be as inexpensive as possible. This philosophy treats the original Moog filter as an ideal continuous-time filter claszR(i.e., four one identical onepoles with a loop around them), rather than a particular circuit or implementation to be emulated. The intent is mainly to help understand the basic behavior of the filter and of possible digital variants. The problem of matching a particular circuit is left to other researchers (for example [110]).

Why not nonlinear? Given that the nonlinear behaviors of the Moog filter and its variants are, for some users, a primary attraction for their use, one must ask why we would bother looking into just the linear aspects of the filter behavior. The reasons are many:

- There is quite a bit to be explored in the linear domain. As we have seen and will see further, there are quite a few interesting variations of these filters to be explored within just the linear domain. Further, since the primary nonlinearity encountered in analog Moog-style filters is saturation, it is believed that in small-signal situations, the behavior of the filter is essentially linear. Hence, the basic linear behavior still needs to be designed, and we make the assumption that it can be done so (to a good approximation) separately from dealing with

the nonlinearity, and as an initial design step.[11] Note that this research is not intended to produce as a result a specific filter design, but rather to reveal directions of filter design which designers might follow to produce their own designs. Thus, the lack of inclusion of nonlinearity in the examples shown in this thesis does not imply that it cannot be included in any particular implementation, nor does it imply that such behaviors were not understood, just that they were assumed to not significantly effect the design issues under discussion.

- An implementation using nonlinearity should be oversampled to reduce the possibility and/or audibility of aliasing due to the bandwidth-spreading capabilities of nonlinearities. As noted earlier, we are attempting to explore design possibilities which do not inherently require oversampling. Luckily, such linear design does not preclude the use of oversampling, and it can always be added onto a design if necessary (for example when adding nonlinearity).

- The author did not desire to drag the whole field of Nonlinear System Dynamics into an already full thesis. For the interested reader, some recommended references on the field are: Vidyasagar [285], Slotine and Li [233], Khalil [134], Arrowsmith and Place [9], and Nayfeh, et al. [188]. It was felt that what one can implement on one's own is on a different level from what one has to be accountable for when putting it into a thesis. As such, the author preferred to leave nonlinear implementation experimentation in "the real world."

- It is unclear whether good objective measures exist for use in deciding how well a given nonlinear design approximates its continuous-time model. This makes it difficult to objectively trade off approximation fidelity versus implementation complexity. Such tradeoffs must therefore be quite subjective, which leaves the implementor open to being pushed around by the "gold connectors crowd," (those audiophiles who have convinced themselves that they can hear the effect of their favorite unmodeled circuit peculiarity, and thus declare any emulation not containing an explicit model to be obviously lacking — "Well *I* can hear it…").[12]

- Good objective measures will probably be psychoacoustically derived and verified, and that is yet another area that the author did not want to wade into for this thesis.

Before we leave the topic of nonlinearity, we must make mention of some early research performed by Harvey Thornburg. Early in his time at CCRMA (approximately 1997), Harvey was researching bandlimited nonlinearities (such as fitting a polynomial to a memoryless nonlinearity, and applying each order to appropriate frequency bands of the input signal). As a side experiment, he explored the subjective effects of placing various saturation nonlinearities into a digital

[11] In fact, most of the author's working implementations (and those for musical use) contain the capability for saturation (usually as two versions: one for speed with just clipping and another for smoother nonlinearity), either due to processor-implemented saturation (as in fixed-point DSP implementations), or some sort of explicit saturation, either polynomial or lookup-table based, usually containing a hyperbolic tangent shape (based on [23]).

[12] These issues also applies if we want to attempt to match fast time-varying behaviors such as audio-rate modulation, and as such give arguments for why such matching was not attempted.

Moog-style filter implementation, looking for the effect of placing them at various locations, and comparing the effect of a single nonlinearity versus distributing multiple nonlinearities through the loop (between one-pole stages being a simple case). He presented the results at an informal research seminar at CCRMA [268]. The results were preliminary (and unfortunately remained unpublished), but one of particular interest was that distributing a smooth nonlinearity (i.e., a polynomial or a tanh) multiple places around the loop gave a more "natural" sound than having a single such nonlinearity in the loop. The consensus explanation was that having multiple saturations reduced how "strongly" any one of them had to act on the signal to keep it in range, and therefore their actions could be more subtle. In hindsight, the seminar agreed that that result was intuitively obvious. At the time, the required oversampling for a "clean" implementation of a filter with such nonlinearities was still considered too expensive in general (though the *Nord Lead* virtual-analog synth, which did use oversampling, did come out around that time), so Harvey and the author did do some research into implementing some of the perceptual behaviors of a saturating filter with a time-varying "locally-linear" filter, using a small control system to dynamically pull back the filter's Q or gain as the signal became large ([251], also Appendix C), to simulate one or two of the effects audible in saturating filters.

Finally, due to the linear assumption, the effect of internal gains as separate from external gains are not explored in this thesis. Such issues become important in nonlinear operation and fixed-point implementation, and are good topics for further research. For example, if a loop gain of, say 4 or 16 is necessary within a loop, should it be lumped into a single gain in the loop or should it be distributed among the stages? If multiple outputs from the filter are to be taken (for example, to create highpass, bandpass, outputs) such gain-distribution questions strongly affect the output gains. Of course, if nonlinearity is implemented (smooth saturation in particular), then the issue of gain distribution becomes inextricably linked with the action of the nonlinearity. The author has heard conjectures that such issues may be the cause of some well-known analog filters sounding more nonlinear that other designs, since, due to particular ladder characteristics, a much larger feedback gain is necessary, and hence a stronger nonlinearity when saturation occurs (Hutchins [114] began discussion along these lines).

3.3.1 Basic discretizations

The first approach to creating a digital Moog-style filter is to do one of the basic discretizations. For these experiments, we will only look at certain discretizations: Backward-Difference Transform, Forward-Difference Transform, Bilinear Transform, and Pole-Zero Mapping. The Impulse-Invariant transform was not considered because it was decided that the partial-fraction expansion that is used in the method would interfere with the desired topology preservation of the resulting system. Other discretization methods (such as zero-order hold equivalents, first-order hold equivalents, etc. [86], and δ-operator methods [82] [168]) were not considered mainly due to time

restrictions.

In this section, we will sometimes refer to parts of the filter as named transfer functions, referring to Figure 3.26. The general filter topology consists of four first-order filters ($H_1(z)$, $H_2(z)$, $H_3(z)$, $H_4(z)$) in series, with negative feedback around them which contains the loop gain k and an additional linear system $G(z)$ (which is usually just a delay or a pass-through). Sometimes we will refer to the feedforward filters as a single system, which we will call $H_f(z)$. The total filter we will call $H(z)$:

$$H(z) = \frac{H_f(z)}{1 + kH_f(z)G(z)} \tag{3.35}$$

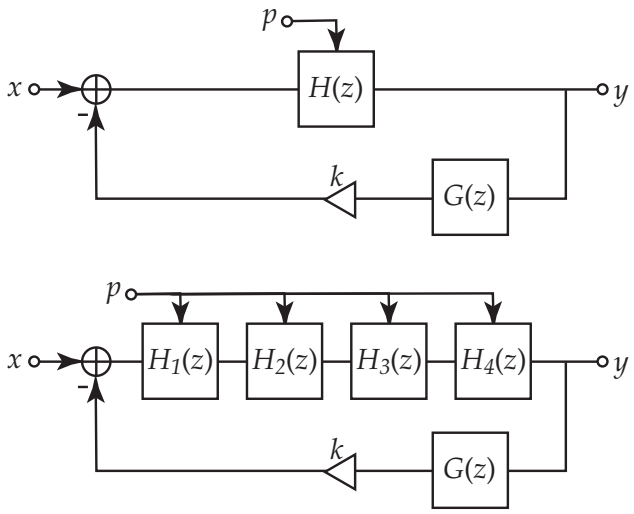


Figure 3.26: Basic block diagram terminology for the Moog-style digital filters.

As part of the analysis of each proposed filter, we will look at the family of loci in k for various values of the tuning parameter, as well as loci in the tuning parameter and other analyses. We are already familiar with the shapes of the ideal constant-Q curves (Figure 1.1, for example). For reference, Figure 3.27 shows the $z = e^{sT}$ transforms of the "X" tracks of a continuous-time Moog-style filter for different open-loop one-pole tunings.

Backward Difference

This is one of the most obvious choices for discretization if starting from the description of a Moog-style filter ("four one-pole filters in a row with feedback around them"), as the backward difference is one of the most common discrete-time one-pole implementations.

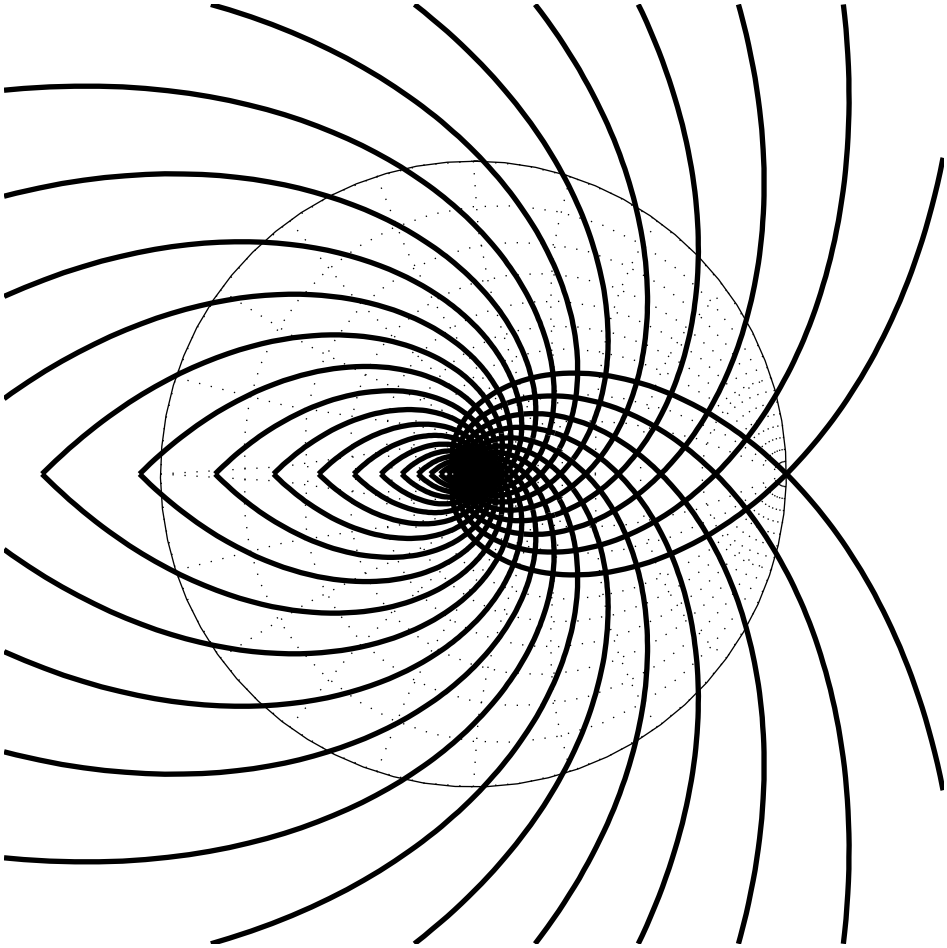


Figure 3.27: $z = e^{sT}$ transforms of the 45-degree "X" traces of a continuous-time Moog-style filter.

The backward-difference transform is:

$$\frac{1}{s} \rightarrow \frac{Tz}{z-1} = \frac{T}{1-z^{-1}} \tag{3.36}$$

For the purposes of this discussion, we are transforming an ideal filter rather than a particular implementation, so we do not need to preserve coefficient ranges, and can simplify by setting $T = 1$, so that the onepole filters are transformed as:

$$H_i(s) = \frac{a}{s+a} \rightarrow H_i(z) = \frac{a}{\frac{z-1}{z} + a} = \frac{a}{(a+1) - z^{-1}} \overset{(a+1)\rightarrow(-1/p)}{\longrightarrow} \frac{p+1}{1+pz^{-1}} \tag{3.37}$$

The transformed filter is described by:

$$H_f(z) = \left[(1+p)\frac{1}{1+pz^{-1}}\right]^4, \; G(z) = 1 \tag{3.38}$$

We will end up describing many of the Moog-style filter implementations by where their open-loop zeros lie, and this one can be described as "all the zeros on $z = 0$". Thus the root-locus equation in

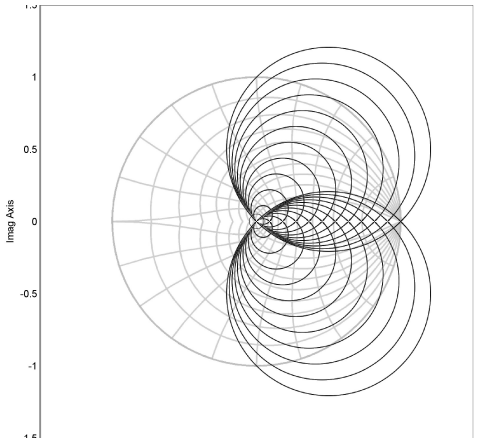


Figure 3.28: Backward-difference transform of ideal Moog-style filter: First-order loci in k for various p between $p = -1$ and $p = 0$.

k is:

$$(p+z)^4 + k(1+p)^4 z^4 = 0 \tag{3.39}$$

i.e., four open-loop poles on $z = -p$, and four open-loop zeros on $z = 0$. Figure 3.28 shows loci in k for the filter. Note that at small k, the locus shape is the X shape we remember from the continuous-time filter (because of the four coincident open-loop poles on $-p$), but then the traces curve around to come back together at $z = 0$. Remember that the backward difference is a conformal map, and as

such, lines and/or circles map to lines and/or circles. The arms of the X shape in the s plane have mapped to circles in the z plane. Further, we know that the backward difference maps the left half plane into the circle between $z = 0$ and $z = 1$, with $s \to \infty$ mapping onto $z = 0$. Unfortunately, this mapping shrinks the curve such that some of the traces don't even reach the unit circle.

The root-locus equation in p is:

$$z^4(1+k) + p(4z^3(kz+1)) + p^2(6z^2(kz^2+1)) + p^3(4z(kz^3+1)) + p^4(1+kz^4) = 0 \tag{3.40}$$

Using the notation we have used before for higher-order loci, we thus have, for the locus in p:

$$\begin{aligned}
D(z) &= z^4(1+k) && \text{(roots: } 0,0,0,0\text{)} \\
N_1(z) &= 4z^3(kz+1) && \text{(roots: } 0,0,0,-1/k\text{)} \\
N_2(z) &= 6z^2(kz^2+1) && \text{(roots: } 0,0,\pm j/\sqrt{k}\text{)} \\
N_3(z) &= 4z(kz^3+1) && \text{(roots: } 0, \text{(3 roots of -1)}/\sqrt[3]{k}\text{)} \\
N_4(z) &= 1+kz^4 && \text{(roots: (4 roots of -1)}/\sqrt[4]{k}\text{)}
\end{aligned}$$

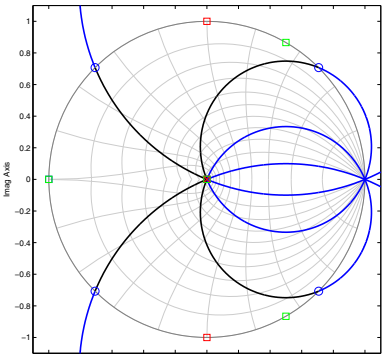


Figure 3.29: Backward-difference transform of ideal Moog-style filter: Fourth-order loci in p (full range of p) for $k = 1$. 'x': roots of $D(z)$, 'o': roots of $N_4(z)$, squares: roots of $N_1(z)$ through $N_3(z)$.

Figure 3.29 shows a locus in p. Again, since the continuous-time loci are lines in the s-plane, they transform to circles in the z-plane. This particular figure is hard to read, however, because it shows the full range of p, so it is hard to tell which traces belong to which range of p. Figure 3.30 shows two loci with p restricted to the range [-1,1], which are a bit easier to interpret.

Unfortunately, this filter is purely theoretical as described (i.e., four onepoles in a row with feedback around them), because the transformed one-pole filters end up with a delay-free path, and the resulting loop would not be directly implementable. As such, when using this transform, one normally adds a delay (possibly doing so without realizing it). The resulting filter will be examined next, and we will not look any further at the features of this implementation.

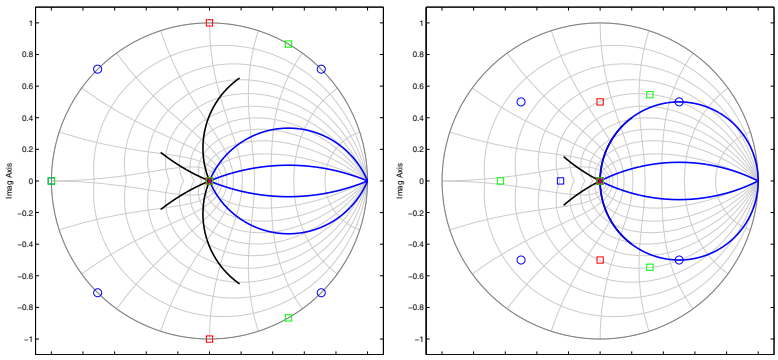


Figure 3.30: Backward-difference transform of ideal Moog-style filter: Fourth-order loci in p ($-1 < p < 1$) for $k = 1$ (Left) and $k = 4$ (Right). 'x': roots of $D(z)$, 'o': roots of $N_4(z)$, squares: roots of $N_1(z)$ through $N_3(z)$.

Backward Difference with Delay

As noted, the backward-difference-transformed one-pole filters have a delay-free loop, so a delay must be added in order to implement the filter (either intentionally or unintentionally[13]).

There is anecdotal evidence [49] that quite a few early commercial implementations of Moog-style filters used this type of implementation (especially in the years before virtual analog became popular, when the filter was mainly used as a special effect). This form (or the forwards difference of the next section) is also what designers tend to implement if they build up the filter from a set of digital onepole filters, as these are the typical forms of one-pole filter implementation.

This filter implementation is the same as the one described in the previous section, but with $G(z)$ being a unit delay:

$$H_f(z) = \left[(1+p)\frac{1}{1+pz^{-1}}\right]^4, \quad G(z) = z^{-1} \tag{3.41}$$

As such, this filter can be described as "Three zeros on $z = 0$, one at infinity." Thus the root-locus equation in k is:

$$(p+z)^4 + k(1+p)^4 z^3 = 0 \tag{3.42}$$

i.e., four open-loop poles on $z = -p$, and three open-loop zeros on $z = 0$ (note that the fourth zero has been cancelled by the unit delay). Figure 3.31 shows loci in k for the filter. Again, for small k, the locus shape is the X shape emanating from the four coincident open-loop poles, and the traces curve around to the left. However, they aren't "pulled in" as tightly towards $z = 0$ as with the pure backward-difference. The two outer traces meet back up on the real axis rather than meeting back at zero (As the inner traces still do, since there are still three coincident open-loop zeros on $z = 0$.

[13] If one were to simply implement four backward-difference onepoles discretely in code and put a feedback around them, an implicit delay would end up in the feedback loop of the resulting filter due to the order of operations. In other words, since a delay-free loop is unimplementable, any actual implementation will not be delay-free.

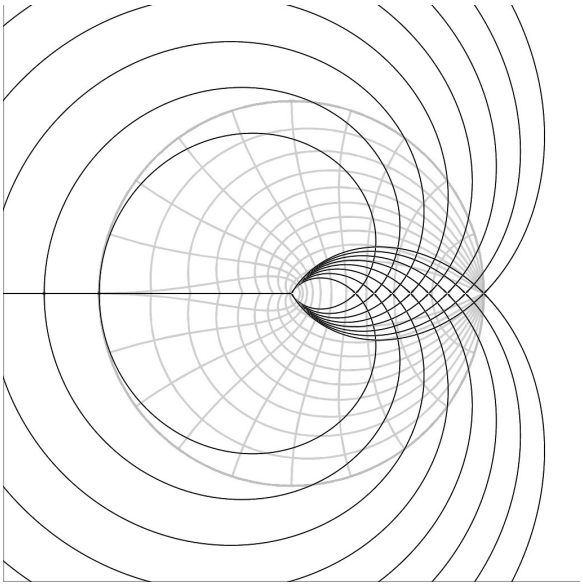


Figure 3.31: Backward-difference with a delay: First-order loci in k for various p between $p = -1$ and $p = -1/3$.

Interestingly, the traces leave $z = -p$ at 45°angles, but the traces approach $z = 0$ at 60°angles. A benefit of not being pulled in so strongly is that the curves have not shrunk as much as they did without the delay, such that the traces cross the unit circle all the way out to $p = -1/3$, and as such are able to trace the whole unit circle, which was impossible without the delay.

The addition of the delay has also caused the resulting closed-loop filter to no longer be describable by the backward-difference transform, and as such, there is no longer a conformal mapping between the continuous-time loci and the discrete-time loci. Indeed, the traces are no longer circles. This is not necessarily a bad thing, as the various desired contours in the z-plane (constant Q, constant f_c, etc) are not circles either.

Still, the fact that the $p = -1/3$ trace is tangent to the unit circle at $z = -1$ does not bode well for f_c/Q separation at high frequencies, as near $z = -1$, k appears to be more of the pole-frequency control than the Q control.

The root-locus equation in p is:

$$z^3(z+k) + p(4(1+k)z^3) + p^2(6z^2(z+k)) + p^3(4z(z+k)) + p^4(1+kz^3) = 0 \tag{3.43}$$

And the individual coefficient polynomials are:

$$\begin{aligned} D(z) = &\quad z^3(z+k) &&\text{(roots: } 0,0,0,-k\text{)} \\ N_1(z) = &\quad 4(1+k)z^3 &&\text{(roots: } 0,0,0\text{)} \end{aligned}$$

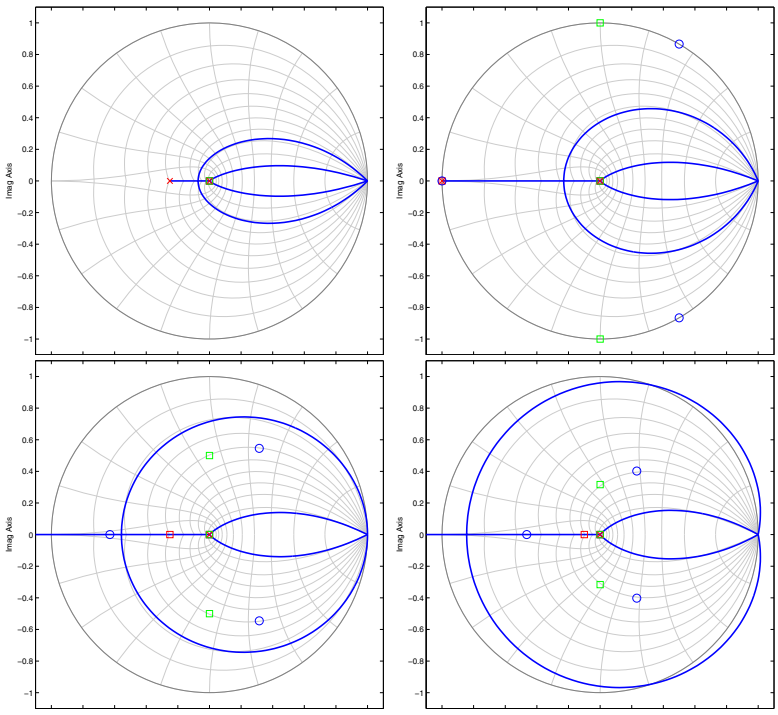


Figure 3.32: Backward-difference with a delay: Fourth-order loci in p ($-1 < p < 0$) for various k: Top Left: $k = 0.25$, Top Right: $k = 1$, Bottom Left: $k = 4$, Bottom Right: $k = 10$. 'x': roots of $D(z)$, 'o': roots of $N_4(z)$, squares: roots of $N_1(z)$ through $N_3(z)$.

$$
\begin{aligned}
N_2(z) &= 6z^2(kz+1) && \text{(roots: } 0, 0, -1/k\text{)} \\
N_3(z) &= 4z(kz^2+1) && \text{(roots: } 0, \pm j/\sqrt{k}\text{)} \\
N_4(z) &= 1 + kz^3 && \text{(roots: } (-1, (-1)^{1/3}, (-1)^{2/3})/\sqrt[3]{k}\text{)}
\end{aligned}
$$

We can see from looking at the loci in p (Figure 3.32) that the tracks are not constant-Q tracks, as we are looking for. This can be interpreted as the fact that f_c and Q are not separated in this filter topology. A useful analysis of this issue is to plot the k required for the poles to hit the unit circle, versus p (Figure 3.33). Note how it changes with p. Also included in the figure are tuning curves. The most important being the bottom-right graph, which shows the mapping from pole angle (i.e., f_c) to p. Note that this filter reaches $f_s/2$ at $p = -1/3$, rather than $p = -1$. We will see this kind of behavior quite a bit in later filters, where the f_c range maps onto a smaller range in p than [-1,1].

Forward Difference

This transformation is not likely to be used by someone specifically transforming the filter, as it is known not to have particularly good transform properties (see Section 1.3.1). However, if one is

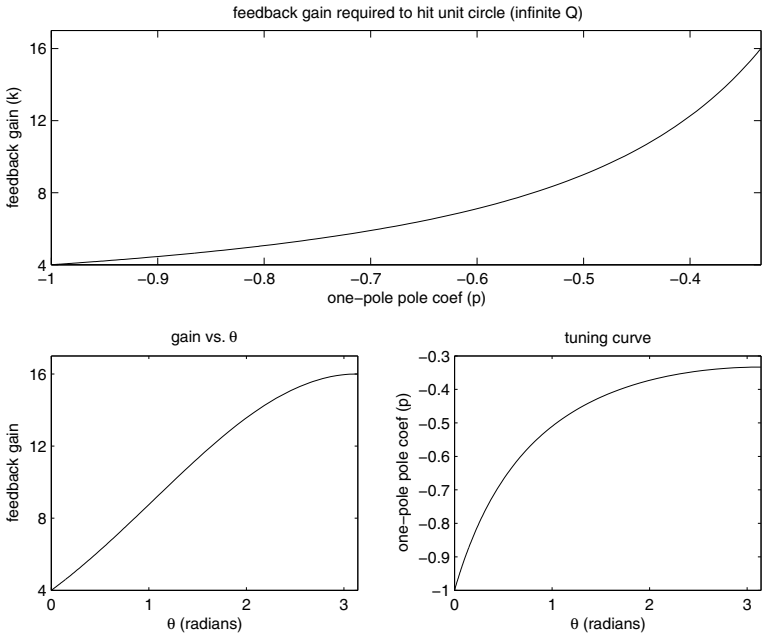


Figure 3.33: Backward-difference with a delay: Top: Loop gain (k) required to hit unit circle, vs. p. Bottom Left: Loop gain vs. pole angle at the unit circle. Bottom Right: Tuning curve (p vs. θ at the unit circle).

building up the filter from its verbal definition, one might just as easily pick a forward-difference version of the onepole as pick a backward difference version (the difference being whether to output the sum or the delay — i.e., whether the delay is in the feedforward or feedback part of the filter). As such, one may want to know what happens.

The forward-difference transform is (assuming T=1):

$$\frac{1}{s} \rightarrow \frac{1}{z-1} = \frac{z^{-1}}{1-z^{-1}} \tag{3.44}$$

so that the onepole filters are transformed as:

$$H_i(s) = \frac{a}{s+a} \rightarrow H_i(z) = \frac{a}{\frac{z-1}{1}+a} = \frac{az^{-1}}{1+(a-1)z^{-1}} \stackrel{a \rightarrow p+1}{\longrightarrow} \frac{(p+1)z^{-1}}{1+pz^{-1}} \tag{3.45}$$

The transformed filter is described by:

$$H_f(z) = \left[(1+p)\frac{z^{-1}}{1+pz^{-1}}\right]^4, \; G(z) = 1 \tag{3.46}$$

As such, this filter can be described as "All zeros at infinity." Note that since each onepole has its delay in the feedforward path, there are no delay-free paths in the outer feedback loop, so no additional delays are necessary.

The root-locus equation in k is:

$$(p + z)^4 + k(1 + p)^4 = 0 \tag{3.47}$$

i.e., four open-loop poles on $z = -p$, and four open-loop zeros at $z \to \infty$. Loci in k for this case

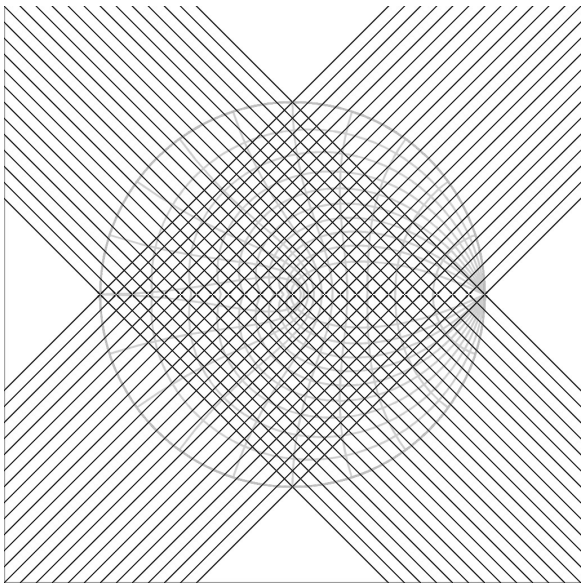


Figure 3.34: Forward-difference: First-order loci in k for various p between $p = -1$ and $p = 1$.

are shown in Figure 3.34. Since the open-loop zeros are at infinity, the X shapes are not distorted at all compared to the continuous-time filter. One drawback that becomes quickly clear is that the left-hand pair of poles is no longer at a much faster decay than the right-hand pair, and so the right-hand pair is no longer necessarily dominant. In fact, when the open-loop poles move to the left of $z = 0$, the left-hand pair of poles actually reaches the unit circle first, in essence flopping the (tenuous) dominance from one set of poles to the other (indeed, around $p = 0$, both sets are nearly the same distance from the unit circle, and so the resulting filter has two distinct peaks). What is not immediately clear from the locus is that the range $\pi/4 < \theta < 3\pi/4$ is not reachable, as in order for poles to get close to the unit circle there, the 'X' must be positioned such that the "other" set of poles is outside the unit circle. Therefore, if we look at the tuning curve for this filter (Figure 3.35), we see that the highest-radius pole frequency jumps from $\pi/4$ to $3\pi/4$. We can also see the effect of the change in the top graph, where the gain curve changes character at $p = 0$. This change and

its associated lack of clearly dominant poles makes this version of the filter undesirable (at least for our current purposes).

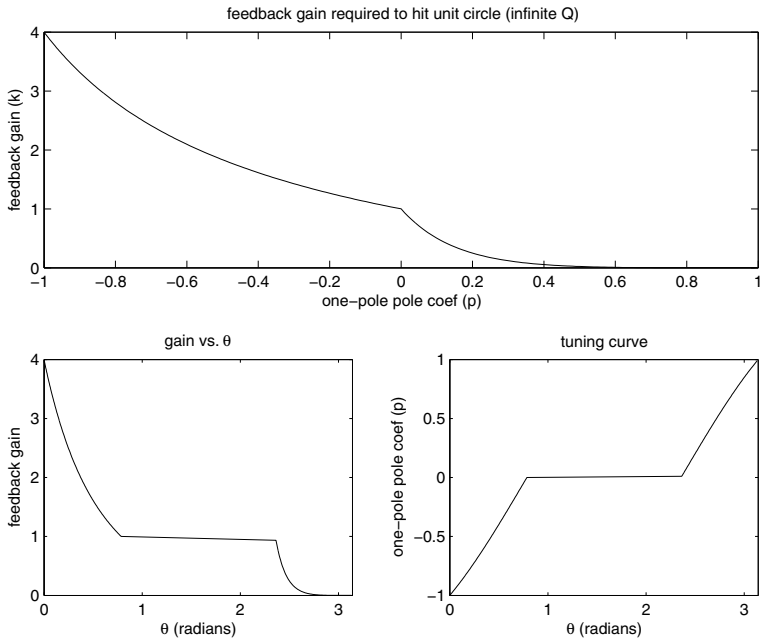


Figure 3.35: Forward-difference: Top: Loop gain (k) required to hit unit circle, vs. p. Bottom Left: Loop gain vs. pole angle at the unit circle. Bottom Right: Tuning curve (p vs. θ at the unit circle).

Bilinear Transform

For those familiar with continuous-time-to-discrete-time discretization, the bilinear transform is expected to perform much better than the ones presented so far. As discussed in Section 1.3.1, the transform is:

$$\frac{1}{s} \to \frac{T}{2}\frac{z+1}{z-1} = \frac{T}{2}\frac{1+z^{-1}}{1-z^{-1}} \tag{3.48}$$

so that the onepole filters are transformed as:

$$H_i(s) = \frac{a}{s+a} \to H_i(z) = \frac{a}{\frac{2}{T}\frac{z-1}{z+1}+a} = \left(\frac{Ta/2}{Ta/2+1}\right)\frac{1+z^{-1}}{1+z^{-1}\left(\frac{Ta/2-1}{Ta/2+1}\right)} \tag{3.49}$$

Now, we absorb T and a into p, the onepole pole location:

$$p = \frac{Ta/2-1}{Ta/2+1} \tag{3.50}$$

To get the onepole filter:

$$H_i(z) = \frac{p+1}{2}\frac{1+z^{-1}}{1+pz^{-1}} \tag{3.51}$$

As with the previous transforms, we see that the filter is scaled so that the DC gain is 0 dB, as was in the continuous-time filter. The system is thus described by:

$$H_f(z) = \left[\left(\frac{1+p}{2}\right)\frac{1+z^{-1}}{1+pz^{-1}}\right]^4, \; G(z) = 1 \tag{3.52}$$

As such, this filter can be described as "All zeros on -1."

The root-locus equation in k is thus:

$$(z+p)^4 + k\left(\frac{p+1}{2}\right)^4 (z+1)^4 = 0 \tag{3.53}$$

i.e., four open-loop poles on $z = -p$ and four open-loop zeros on $z = -1$. If we compare against the backward-difference case (Section 3.3.1), we are simply moving the zeros from $z = 0$ to $z = -1$. As such, we should expect much the same root loci in k, just scaled up and shifted over, to move $z = 0$ onto $z = -1$. Indeed, that is what we get (Figure 3.36) The nice thing about this scale and shift is

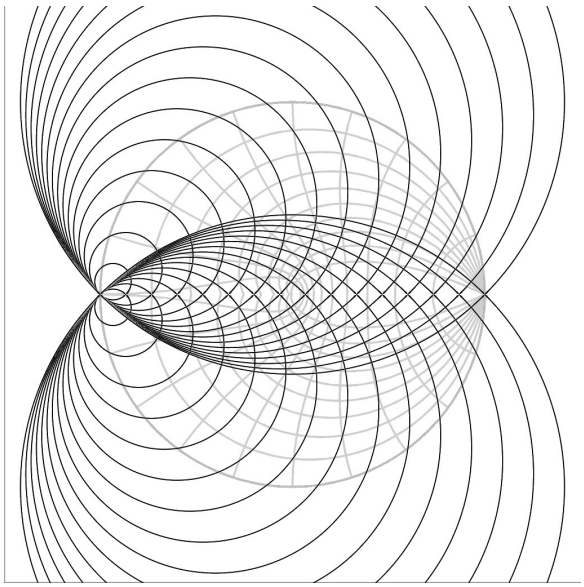


Figure 3.36: Bilinear Transform: First-order loci in k for various p between $p = -1$ and $p = 1$.

that the loci now line up very nicely with the unit circle. For example, all the circles intersect the

unit circle at an angle of 45°from the tangent of the circle.[14] Again, as in the backward difference case, this transform is a conformal map, and we see that the lines of the X's have transformed to circles.

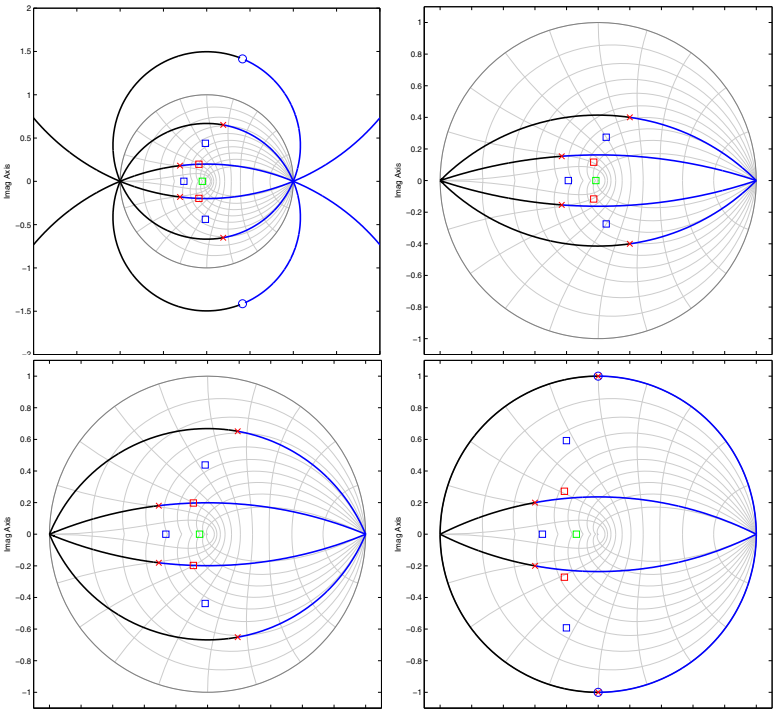


Figure 3.37: Bilinear Transform: Fourth-order loci in p ($-1 \leq p \leq 1$) for various k: Top Left: $k = 1$ (in this plot, p sweeps full range, to show that the traces are indeed circles), Top Right: $k = 1/4$, Bottom Left: $k = 1$, Bottom Right: $k = 4$. 'x': roots of $D(z)$, 'o': roots of $N_4(z)$, squares: roots of $N_1(z)$ through $N_3(z)$.

The root-locus equation in p is:

$$\begin{aligned}(16z^4 + k(z+1)^4) + p(64z^3 + 4k(z+1)^4) + p^2(96z^2 + 6k(z+1)^4) \\ +p^3(64z + 4k(z+1)^4) + p^4(16 + k(z+1)^4) = 0\end{aligned} \tag{3.54}$$

And the individual coefficient polynomials are:

$$\begin{aligned}D(z) &= 16z^4 + k(z+1)^4 \\ N_1(z) &= 64z^3 + 4k(z+1)^4 \\ N_2(z) &= 96z^2 + 6k(z+1)^4\end{aligned}$$

[14] This happens because conformal maps preserve local angles [189].

$$N_3(z) = 64z + 4k(z+1)^4$$
$$N_4(z) = 16 + k(z+1)^4$$

Unfortunately, the roots of these are no longer simple to describe.[15]

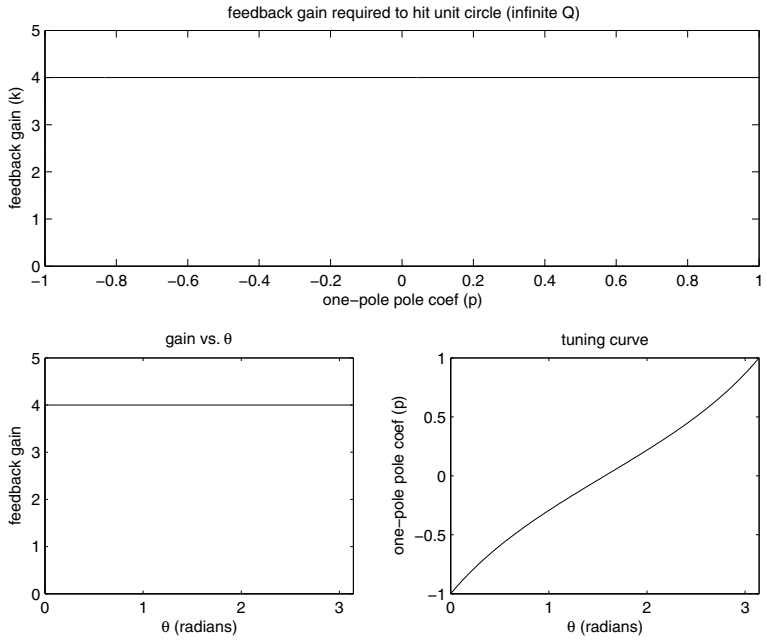


Figure 3.38: Bilinear Transform: Top: Loop gain (k) required to hit unit circle, vs. p. Bottom Left: Loop gain vs. pole angle at the unit circle. Bottom Right: Tuning curve (p vs. θ at the unit circle). Nice curves, but the filter has a delay-free loop.

Figure 3.37 shows loci in p for various k. Note again, in the top-left plot, that the traces are circles, due to the bilinear transform being a conformal map. The rest of the plots limit p to between -1 and 1. The most interesting plot is the $k = 4$ plot (the bottom-right one), because the output poles trace the unit circle exactly. As such, the f_c and Q controls are separated, at least at infinite Q (this can also be seen in Figure 3.38). This would be very good news, except for the fact that this filter, like the backward-difference, has a delay-free loop. In the next section, we look at what happens when we add a delay into the loop.

We must note that within the field of warped-filter design, the problem of delay-free loops has typically been handled by some sort of structure modification and online coefficient recomputation ([245] for example). This research did not attempt such a solution, primarily on the assumption that it would adversely affect the "physical model" nature of the "one big loop" topology, and that the required coefficient recalculation would be too expensive.

[15] See Section 3.3.3, which looks into this a bit deeper

Bilinear with delay

In order to make the filter implementable, a delay is put into the loop. Thus:

$$H_f(z) = \left[\left(\frac{1+p}{2}\right)\frac{1+z^{-1}}{1+pz^{-1}}\right]^4, \; G(z) = z^{-1} \tag{3.55}$$

We can describe this as "Four zeros on $z = -1$, and another at infinity".

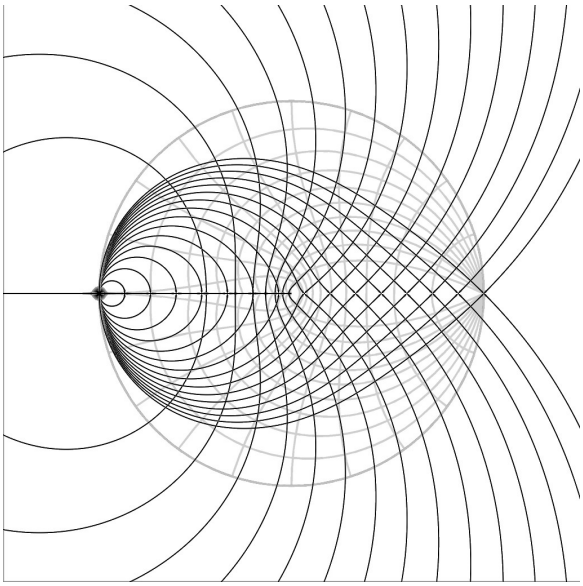


Figure 3.39: Bilinear Transform with delay: First-order loci in k for various p between $p = -1$ and $p = 1$.

The root-locus equation in k is:

$$z(z+p)^4 + k\left(\frac{p+1}{2}\right)^4 (z+1)^4 = 0 \tag{3.56}$$

i.e., four open-loop poles on $z = -p$ with one more on $z = 0$, and four open-loop zeros on $z = -1$. Note that the polynomials in z have become fifth-order because, unlike the backward-difference case, the pole in $G(z)$ can't simply cancel one of the feedforward zeros. Loci in k are shown in Figure 3.39. As when a delay was added to the backward-difference-transformed filter, the effect here is to "weaken" the pull of $z = -1$ on the loci, so that the "right-hand" traces of each 'X' don't come back together at $z = -1$, rather coming back together much further down the negative real axis (the filter is also no longer described by a conformal map of the continuous-time filter, so the traces are no longer always circles, which we also saw in the backward-difference case).

Also of interest: because of the open-loop pole at $z = 0$, when p crosses past zero, the traces emanating from the poles on $z = -p$ rotate from the 'X' departure angles to the other four possible departure angles (halfway between the 'X' angles: $0, \pm\pi/2, \pi$).[16] It will be more visible in Figure 3.40, but one can guess from examining the loci that for p near DC, equal steps in p correspond roughly to equal steps in crossing angle along the unit circle (i.e., the tuning appears very close to linear in p near DC). If we look at the tuning curve in Figure 3.40, it certainly looks linear at small θ. In fact, it looks very close to $p = 2\sin(\theta/2) - 1$. We can show experimentally that it is exactly that (Figure 3.41, with error down below 10^{-8}) Still, the gain curve is not flat, and so this filter is

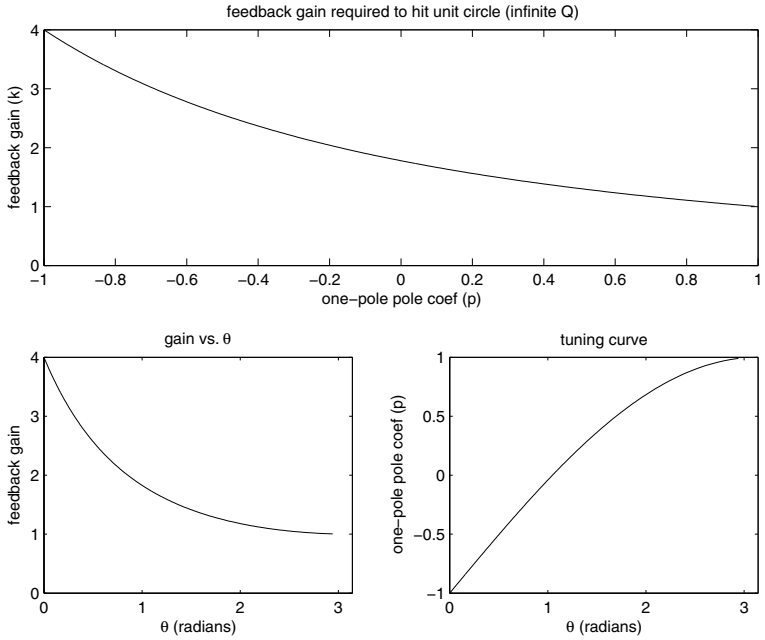


Figure 3.40: Bilinear Transform with delay: Top: Loop gain (k) required to hit unit circle, vs. p. Bottom Left: Loop gain vs. pole angle at the unit circle. Bottom Right: Tuning curve (p vs. θ at the unit circle).

no longer separable, like the ideal bilinear transformed filter was. The curve is less extreme than in the delayed backward-difference case, though. However, the curve falls rather than rises. The implication of this is that for $k > 1$, one can sweep upwards and the Q will rise, and eventually go unstable, whereas for a rising gain curve, holding k constant and sweeping to higher f_c will result in a lowering of Q, and thus the filter will at least stay stable (in other words, if the delayed backward-difference filter is stable at $f_c \to 0$ for a given k, then it will be stable for all f_c at that k, though at significantly lower Q).

[16] Also of interest: the traces for p just to the left of $z = 0$ are warped in a shape very reminiscent of the way the constant-natural-frequency contours are warped around $z = 0$. Not sure if this means anything or if it can be exploited.

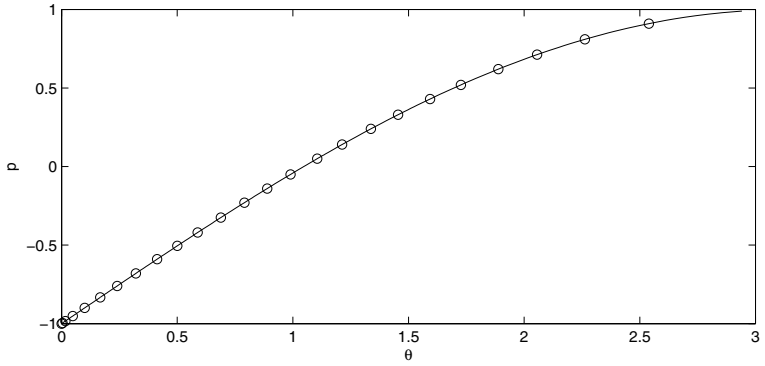


Figure 3.41: Bilinear Transform with delay: Tuning curve. Circles: $2\sin(\theta/2) - 1$, Line: experimentally-derived tuning curve.

The root-locus equation in p is quite similar to the equation for the pure bilinear-transform filter:

$$\begin{aligned}(16z^5 + k(z+1)^4) + p(64z^4 + 4k(z+1)^4) + p^2(96z^3 + 6k(z+1)^4) \\ +p^3(64z^2 + 4k(z+1)^4) + p^4(16z + k(z+1)^4) == 0\end{aligned} \tag{3.57}$$

And the individual coefficient polynomials are:

$$\begin{aligned}D(z) &= 16z^5 + k(z+1)^4 \\ N_1(z) &= 64z^4 + 4k(z+1)^4 \\ N_2(z) &= 96z^3 + 6k(z+1)^4 \\ N_3(z) &= 64z^2 + 4k(z+1)^4 \\ N_4(z) &= 16z + k(z+1)^4\end{aligned}$$

Effectively, the terms aside from the $(z+1)^4$ terms have increased order in z by one. Representative loci are shown in Figure 3.42. As was noted above, we can see that for $k > 1$, the sweeps start stable but leave the unit circle at high cutoff frequencies.

Pole-Zero Placement

To review, the rules for pole-zero placement (see Section 1.3.2) are:

- Place poles according to $z = e^{sT}$. In this case, since the poles are all real, the poles end up somewhere on the real axis.
- Place finite zeros according to $z = e^{sT}$. There are none in the ideal continuous-time Moog-style filter.

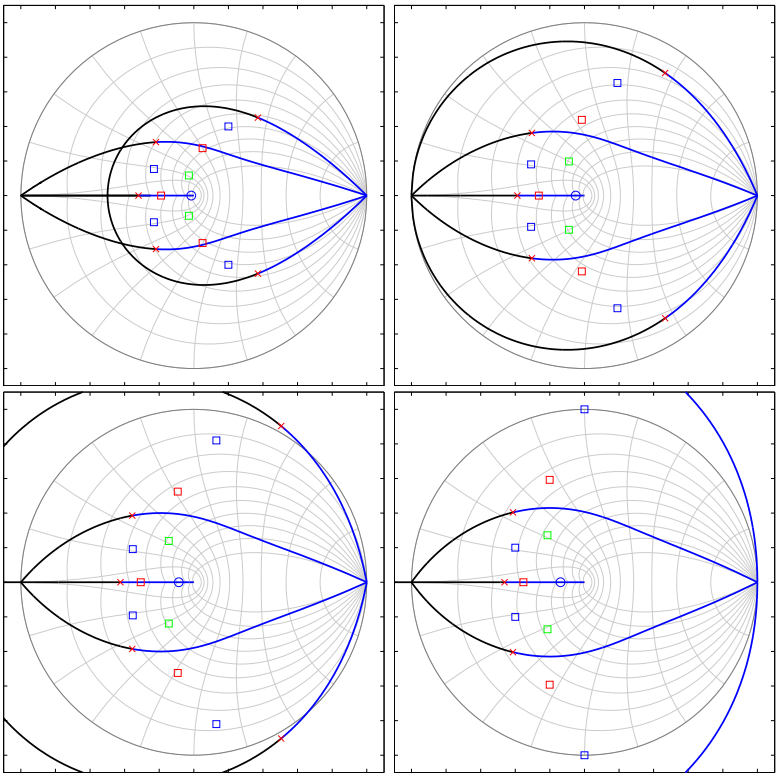


Figure 3.42: Bilinear Transform: Fourth-order loci in p ($-1 \leq p \leq 1$) for various k: Top Left: $k = 1/4$, Top Right: $k = 1$, Bottom Left: $k = 2$, Bottom Right: $k = 4$. 'x': roots of $D(z)$, 'o': roots of $N_4(z)$, squares: roots of $N_1(z)$ through $N_3(z)$.

- Place all but one of the infinite zeros at $z = -1$

- Place the remaining infinite zero at $z \to \infty$ which means that the system will have no delay-free path, and that the feedback term will not need a delay.

Thus we get:

$$H_f(z) = \left[\left(\frac{1+p}{2}\right)\frac{1+z^{-1}}{1+pz^{-1}}\right]^3 \left[(1+p)\frac{1}{1+pz^{-1}}\right], \quad G(z) = 1 \tag{3.58}$$

And the root-locus equation in k is:

$$(z+p)^4 + k\left(\frac{(p+1)^4}{2^3}\right)(z+1)^3 = 0 \tag{3.59}$$

So we can describe this filter as "Three zeros at $z = -1$ and one at infinity. To do this, we have actually made a slight change in the methodology: we have transformed the whole fourpole-feedforward section as a whole, rather than separately transforming each onepole filter (this manifests in the way the zeros were transformed: if we transformed each onepole separately, each would end up with an infinite zero, and we would have a filter basically the same as the forward-difference transformed filter, which did not work out). Note that this looks very similar to the delayed bilinear transform, but instead of getting implementability by adding a pole (and jumping up to 5th order), it is achieved by moving one of the zeros to infinity. One can also think of this case as taking the pure bilinear transform and replacing one of the first-order filters with a forward-difference version (and remapping the meaning of p, if once cares about the relation to the original continuous-time filter).

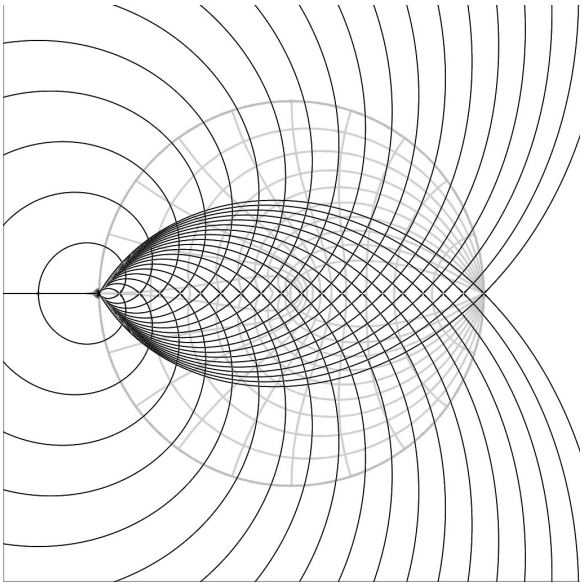


Figure 3.43: Pole/Zero Placement (3 zeros at $z = 0$, one at $z \to \infty$): First-order loci in k for various p between $p = -1$ and $p = 1$.

Loci in k are shown in Figure 3.43, and the gain/tuning curves are shown in Figure 3.44. The k-loci have features somewhere between those of the pure bilinear transform filter (Figure 3.36) and the delayed bilinear transform filter (Figure 3.39). In particular, the lack of a fifth pole in the system makes the behavior a bit more regular across the range of p. Looking at the gain curve, it is nearly a straight line (it dips a bit from a straight line, actually). On first look, the fact that gain curve actually approaches zero as $p \to 1$ is a bit worrisome. Of interest is the tuning curve, which doesn't have extreme slopes or curvatures, which bodes well for implementing inexpensive tuning transformation. Still, the gain curve is not flat, so this form is yet again not inherently separable. It

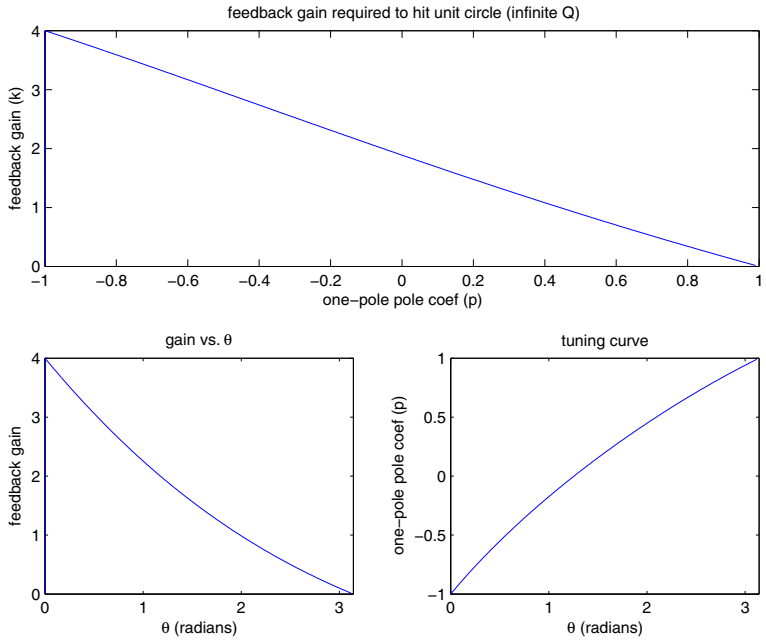


Figure 3.44: Pole/Zero Placement (3 zeros at $z = 0$, one at $z \to \infty$): Top: Loop gain (k) required to hit unit circle, vs. p. Bottom Left: Loop gain vs. pole angle at the unit circle. Bottom Right: Tuning curve (p vs. θ at the unit circle).

is a good candidate for further exploration, though, like the delayed bilinear form.

The root-locus equation in p is also similar to the equations for the two bilinear-transform-based filters:

$$\begin{aligned} 8z^4 + k(z+1)^3 + 32z^3 + 4k(z+1)^3 + 48z^2 + 6k(z+1)^3 \\ +32z + 4k(z+1)^3 + 8 + k(z+1)^3 = 0 \end{aligned} \tag{3.60}$$

And the individual coefficient polynomials are:

$$\begin{aligned} D(z) &= 8z^4 + k(z+1)^3 \\ N_1(z) &= 32z^3 + 4k(z+1)^3 \\ N_2(z) &= 48z^2 + 6k(z+1)^3 \\ N_3(z) &= 32z + 4k(z+1)^3 \\ N_4(z) &= 8 + k(z+1)^3 \end{aligned}$$

The major differences being the scales of the z^n terms, as one of the stages does not have the $1/2$ in

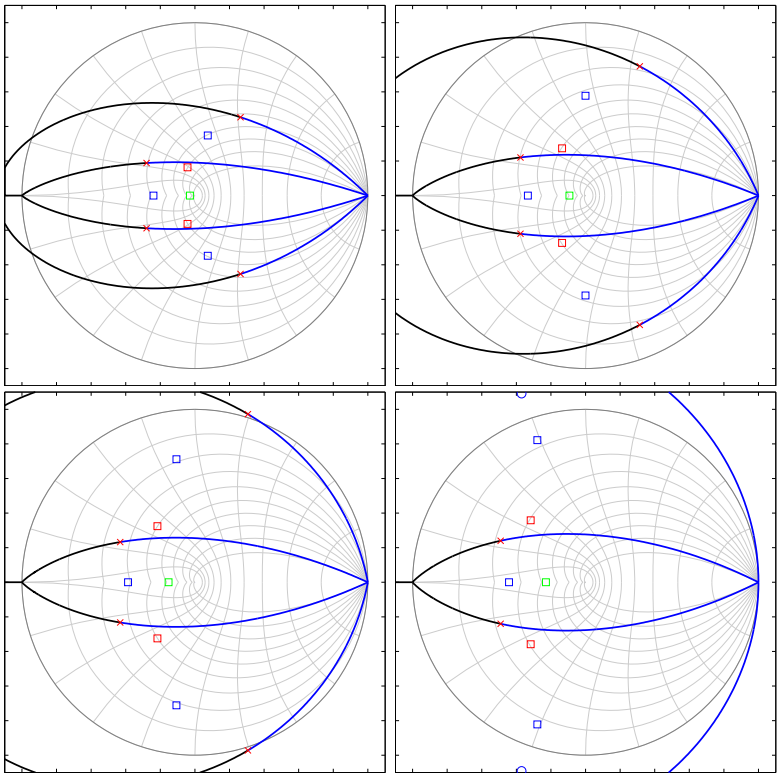


Figure 3.45: Pole/Zero Placement: Fourth-order loci in p ($-1 \leq p \leq 1$) for various k: Top Left: $k = 1/4$, Top Right: $k = 1$, Bottom Left: $k = 2$, Bottom Right: $k = 4$. 'x': roots of $D(z)$, 'o': roots of $N_4(z)$, squares: roots of $N_1(z)$ through $N_3(z)$.

its DC scaling, and the fact that the $(z+1)$ terms are reduced to third-order (again, since only three of the stages have the zero at $z = -1$). Representative loci are shown in Figure 3.45. As evidenced by the gain curve, the tracks start inside the unit circle but eventually cross outside.

3.3.2 Analyzing the Basic Discretizations

As a review:

- **Backward Difference:** "All four zeros at $z = 0$". Unimplementable. We will consider this one no further.
- **Backward Difference With Delay:** "Three zeros at $z = 0$, one zero at infinity". Common, not independent controls.
- **Forward Difference:** "All four zeros at infinity". Non-dominant poles, hole in frequency range. We will consider this one no further.
- **Bilinear Transform:** "All four zeros at $z = -1$". Independent controls, but unimplementable. We will consider this one no further, but may refer to it on occasion.
- **Bilinear Transform with Delay:** "Four zeros at $z = -1$ with an extra pole at $z = 0$". Not independent controls, but closer to independent than backward-difference with delay. Promising.
- **Pole/Zero Mapping:** "Three zeros at $z = -1$, and the other at infinity". Not independent controls, but promising "non-extreme" behaviors.

Here we will introduce another form of analysis which applies particularly well for analyzing Q/f_c separation. Since Q becomes extremely sensitive near the unit circle ($Q \approx \theta/(2(1-r))$), minor variations in Q become hard to pick out on a root locus at high Q. Thus we introduce another plot: Q vs. f_c for a number of different Q tracks. For filters with completely independent controls, this plot would have tracks that were perfectly flat, and so we can easily pick out variations from independence rather easily. In practice, we will plot $\log(Q)$ vs. log(pole angle), since both are perceived approximately logarithmically. Also, we use pole angle as the frequency axis, rather than f_c. As an example, let's present such a plot for the Chamberlin state-variable filter (Section 2.4). (Figure 3.46). The state-variable filter has highly independent Q and f_c controls for Q larger than 3 or so, and for f_c less than approximately 1 radian. For lower Q, we see a slight drop in pole frequency, and higher frequencies, we see a rise in Q, which corresponds to the fact that the tracks meet perpendicular to the real axis (see Figure2.26), whereas true constant-Q tracks would bend back inwards as $\theta \to \pi$.

The range of Q shown in Figure 3.46 might be considered excessively wide by some, and they would have a valid point. It turns out that there are two classes of usage of such filters:

- **"Pinging the filter":** In this mode of operation, the filter is used to synthesize a decaying sinusoid when pinged with an input consisting mainly of impulsive inputs. As such, the filter is often expected to ring out audibly, so it needs a decay time sometimes as large as multiple seconds, which corresponds to extremely high Q (as high as 10000 or more). For

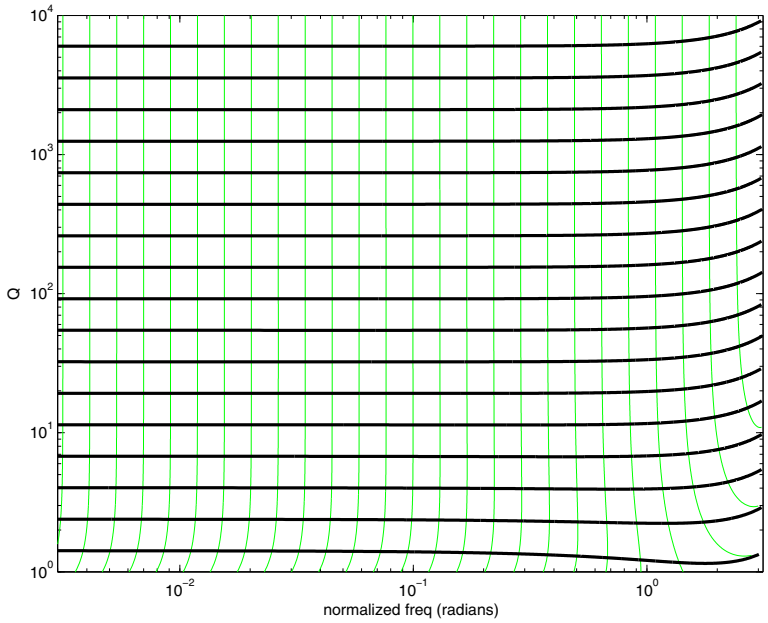


Figure 3.46: Q vs. freq plot for Chamberlin state-variable filter. Thick lines: ff sweeps for various values of qq. Thin lines: qq sweeps for various values of ff. Only the complex-poles region of operation (i.e. ff < 2 – qq) is shown, as the pole-angle definition of Q is being used.

such usage, the filter must be able to get very close to the unit circle without going unstable.[17] Keeping separability at such Q becomes difficult.

- **"Shaping a spectrum":** In this mode of operation, the filter is acting as a brightness or vowel effect, or a "wah" effect. In these situations, Q values beyond about 20 cause the filter to become so narrow as to start dominating the sound that it is filtering, and reducing the effectiveness of the filter.

Therefore, we will make two versions of this plot: one with a high-Q range and one with a low-Q range, to more clearly show the low-Q behaviors of the filters. For the state-variable filter example, the low-Q plot is shown in Figure 3.47. Note that this plot does more clearly show how the filter's approximations start breaking down below $Q = 1$ or so.

As discussed elsewhere, our current goal is to find Moog-style filter topologies for which Q and f_c are independent (or "separable"): If we hold the Q coefficient constant, we can sweep the f_c coefficient around and stay on the same Q, without having to readjust the Q coefficient. We saw that the ideal continuous-time Moog-style filter had this property, and that the state-variable filter

[17] Further, if the filter is to be used in a self-oscillating state (i.e., very slightly unstable, probably with some sort of loop saturation), then having separation at very high Q will help maintain a consistent "level of instability" across the tuning frequency range.

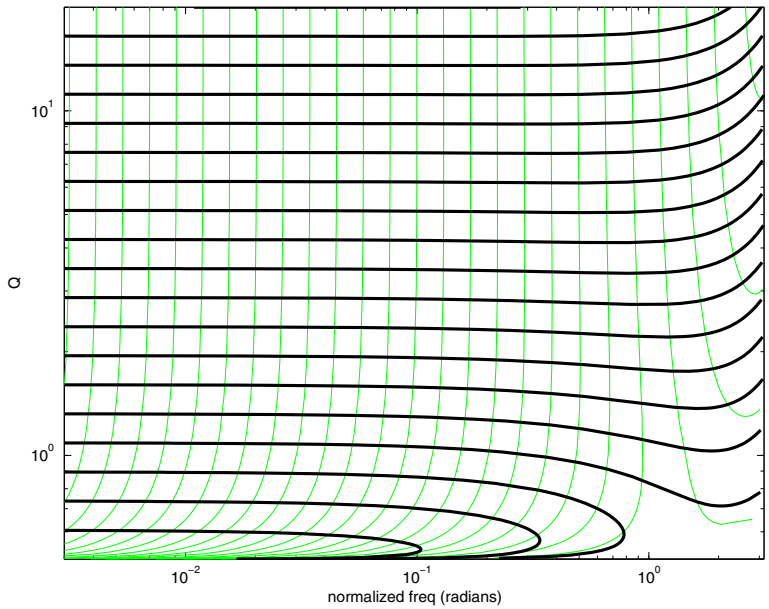


Figure 3.47: Q vs. freq plot for Chamberlin state-variable filter, lower Q range.

very closely approximates this property over a large range of Q and f_c. If, for some reason, this goal isn't achievable (and so far, we haven't found such a topology), the next goal is to make mappings which separate Q and f_c as cheaply as possible. We will mainly be considering mappings which take the form of $k = f(Q)g(p)$, where g is as cheap as possible (as is f, of course). We will call mappings such as g "Separation Functions" (or Separation Tables, or Separation Curves, etc.).

We also, of course, desire separation in the other axis (i.e., if we hold the f_c coefficient constant, we can sweep the Q coefficient around and stay on the same f_c. We are making the assumption that this separability will tend to "work itself out" if the above separability is achieved. Thus we will not be working with mappings like $p = f(k)$, or, more correctly, $k = f(Q, f_c), p = g(f_c, Q)$. As was discussed in the introduction to this chapter, we assume that the use of such 2-dimensional function/tables can achieve perfect tuning and separability, but is considered overkill.

Therefore, we postulated in the introduction that it is sufficient to use at most three lookup tables (or functions) in a design:

$$\begin{aligned} p &= f_{\text{tuning}}(f_c) \quad \text{(computed at infinite } Q\text{)} \\ k &= f_{\text{separation}}(p) \times f_{\text{Q}}(Q) \end{aligned}$$

Where, depending on the circumstances and the necessity for accurate Q selection, we may leave out f_{Q}, or replace it with an inexpensive approximation. In some special-effects situations, it may

even be possible to replace f_{tuning} with a cheap approximation. However, we will be assuming that all three lookup tables will be needed in many musical situations. We will now analyze filters, based on the basic discretizations, which make use of these lookup tables, to verify the above assumption (Figures 3.48–3.53).

In the Q-vs-p analyses, we used the following k values for the raw filters. For the high-Q range: $k = 4 * (1 - 10^x)$, where x varied between 10^{-4} and 0 in 20 steps. For the low-Q range: $k = 4 * x$, where x varied between 0 and 1 in 20 steps. The maximum of k was chosen to be 4.0 because all the filters have $k = 4$ as the infinite-Q value as $f_c \rightarrow DC$ (due to the DC gains of their onepoles being normalized to 0 dB).

For the analyses using separation tables, the maximum k was changed from 4.0 to be the separation-table value. Thus for the high-Q range, we used $k = f(p) * (1 - 10^x)$, with the same set of x values as before, and for the low-Q range, $k = f(p) * x$, with the same x values as before.

Note that logarithmic spacing of k away from 4.0 (or from $f(p)$) tends to give approximately equal spacing in Q (at least near DC). We will look into that a bit later.

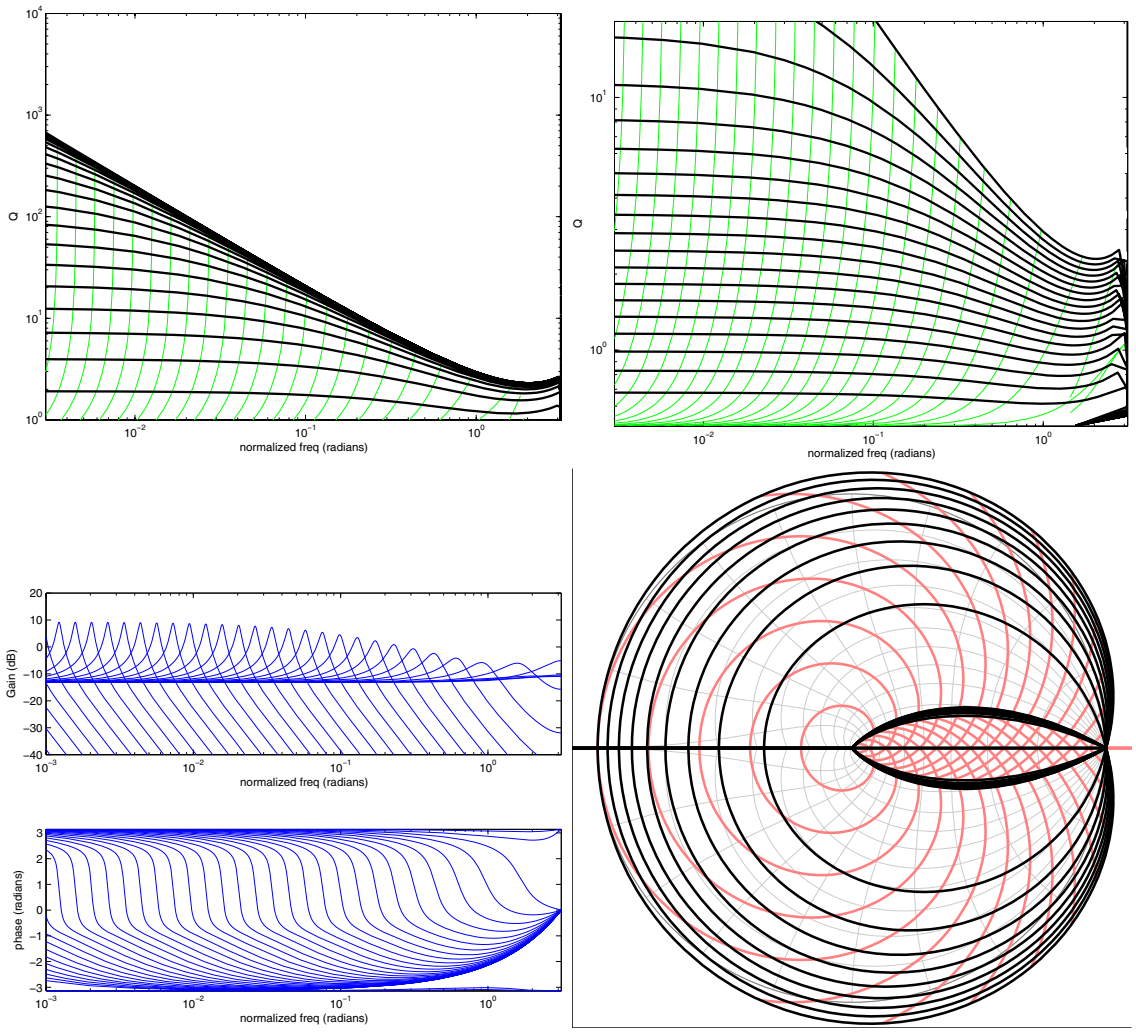


Figure 3.48: Backward Difference with Delay: Raw filter (no Lookup tables). Top Left: Q vs. p, wide Q range. Top Right: Q vs. p, narrow Q range. Bottom Left: representative frequency responses for a p sweep ($k = 3.5$). Bottom Right: Pole traces (Dark: constant k, Light: constant p) ($-1 < p < 0$).

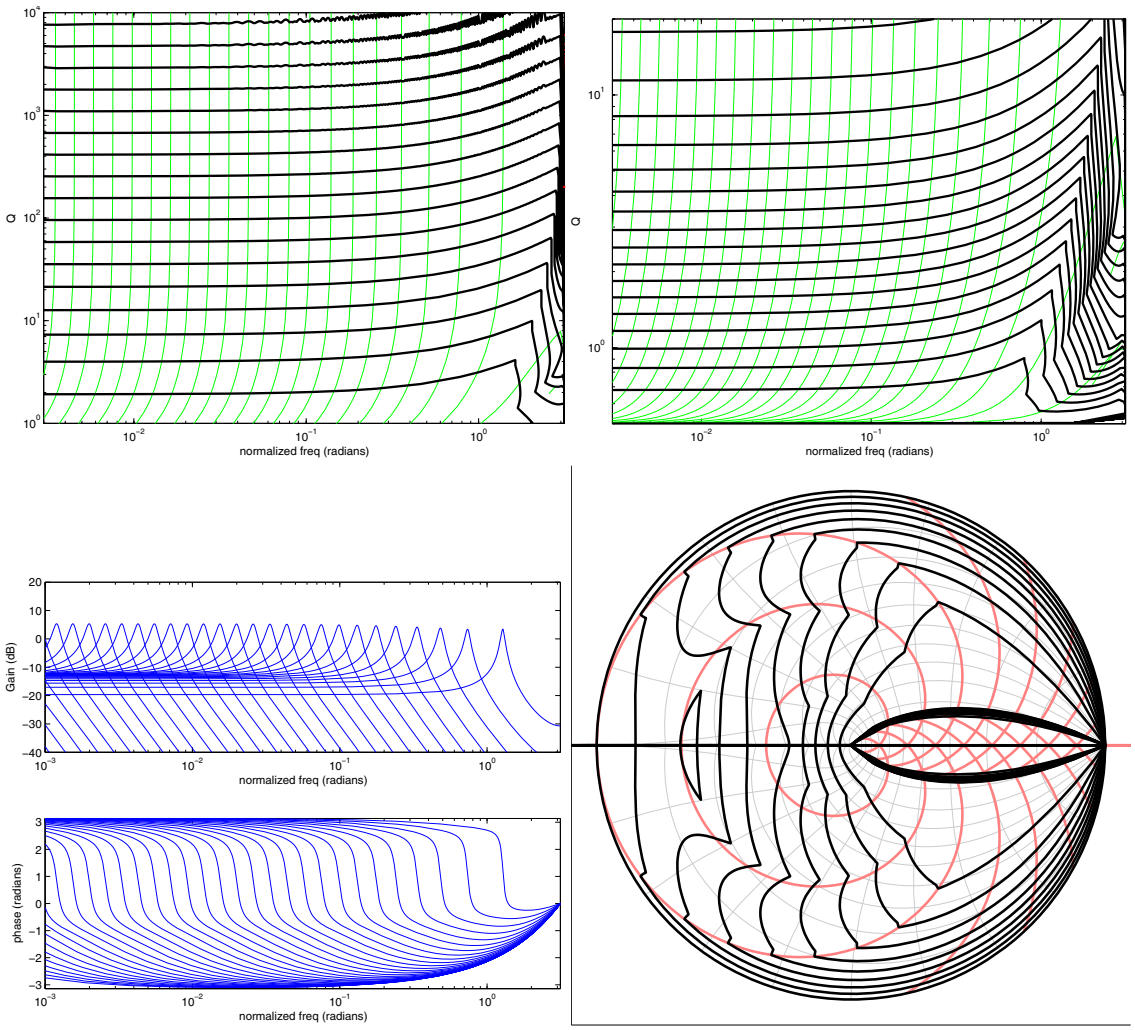


Figure 3.49: Backward Difference with Delay: Using separation table shown in Figure 3.33. Top Left: Q vs. p, wide Q range. Top Right: Q vs. p, narrow Q range. Bottom Left: representative frequency responses for a p sweep ($k = 0.8f(p)$). Bottom Right: Pole traces (Dark: constant k, Light: constant p).

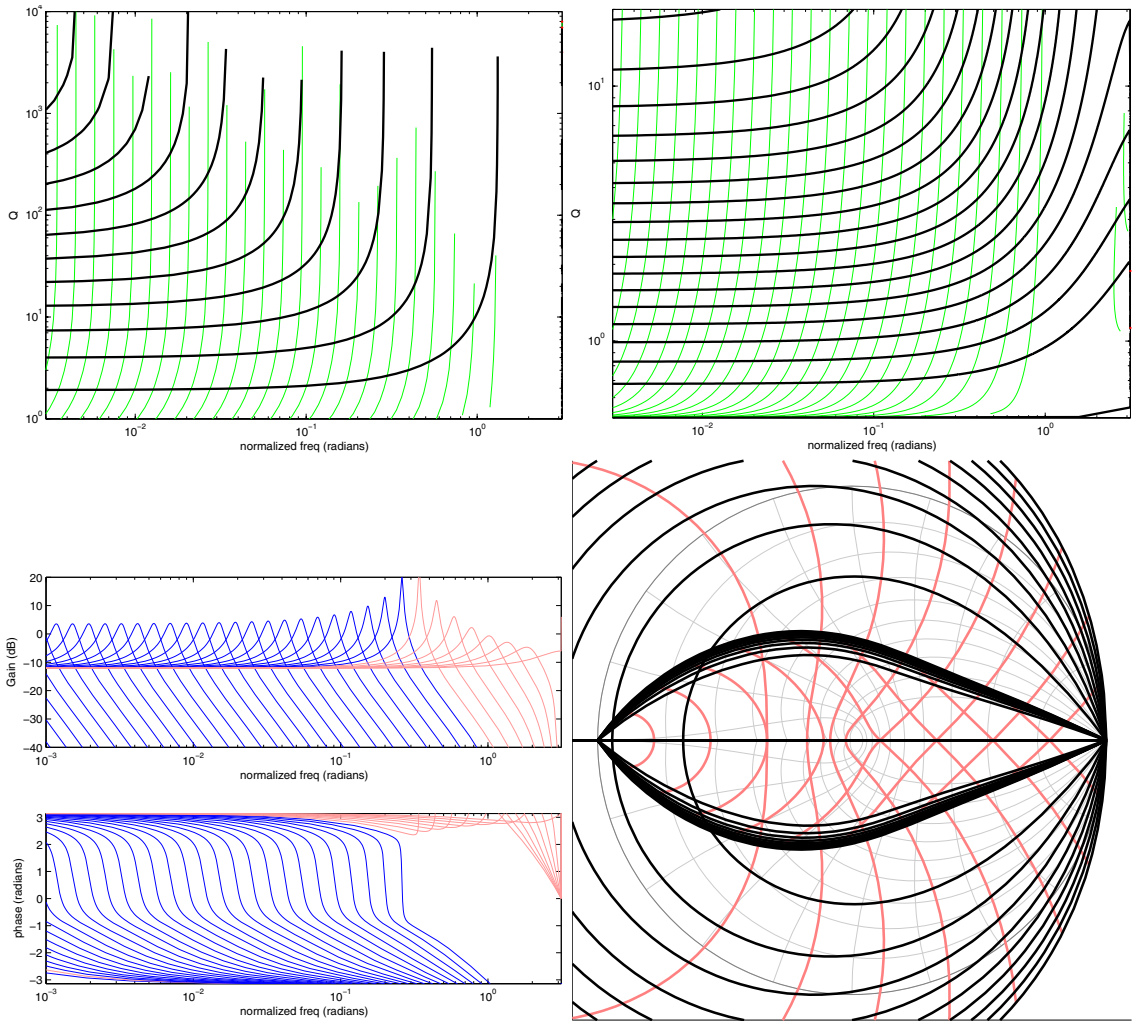


Figure 3.50: Bilinear Transform with Delay: Raw filter (no Lookup tables). Top Left: Q vs. p, wide Q range. Top Right: Q vs. p, narrow Q range. Bottom Left: representative frequency responses for a p sweep ($k = 3$) (light traces: unstable filters). Bottom Right: Pole traces (Dark: constant k, Light: constant p).

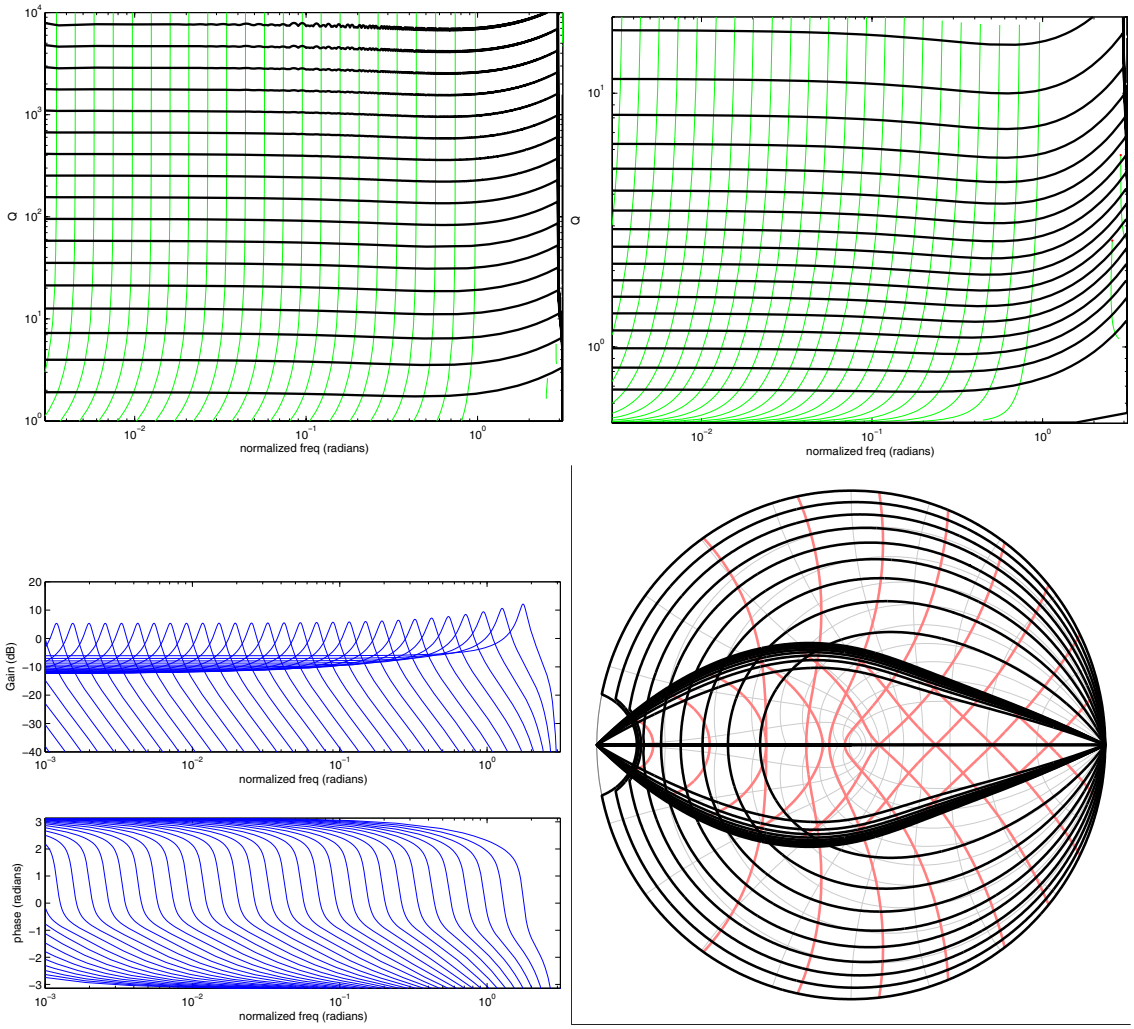


Figure 3.51: Bilinear Transform with Delay: Using separation table shown in Figure 3.40. Top Left: Q vs. p, wide Q range. Top Right: Q vs. p, narrow Q range. Bottom Left: representative frequency responses for a p sweep ($k = 0.8f(p)$). Bottom Right: Pole traces (Dark: constant k, Light: constant p).

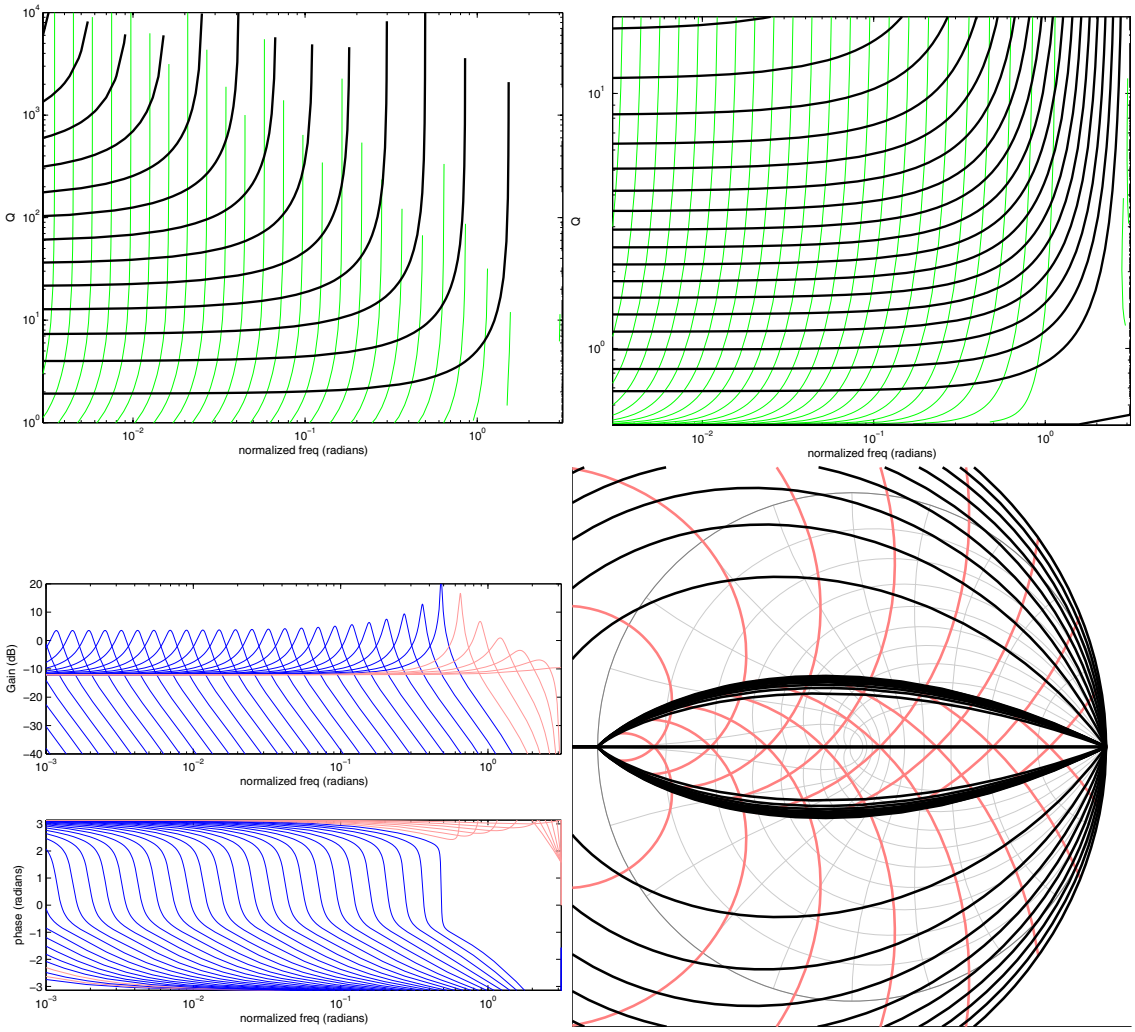


Figure 3.52: Pole Placement (Three zeros on $z = -1$, one zero at $z \to \infty$): Raw filter (no Lookup tables). Top Left: Q vs. p, wide Q range. Top Right: Q vs. p, narrow Q range. Bottom Left: representative frequency responses for a p sweep ($k = 3$) (light traces: unstable filters). Bottom Right: Pole traces (Dark: constant k, Light: constant p).

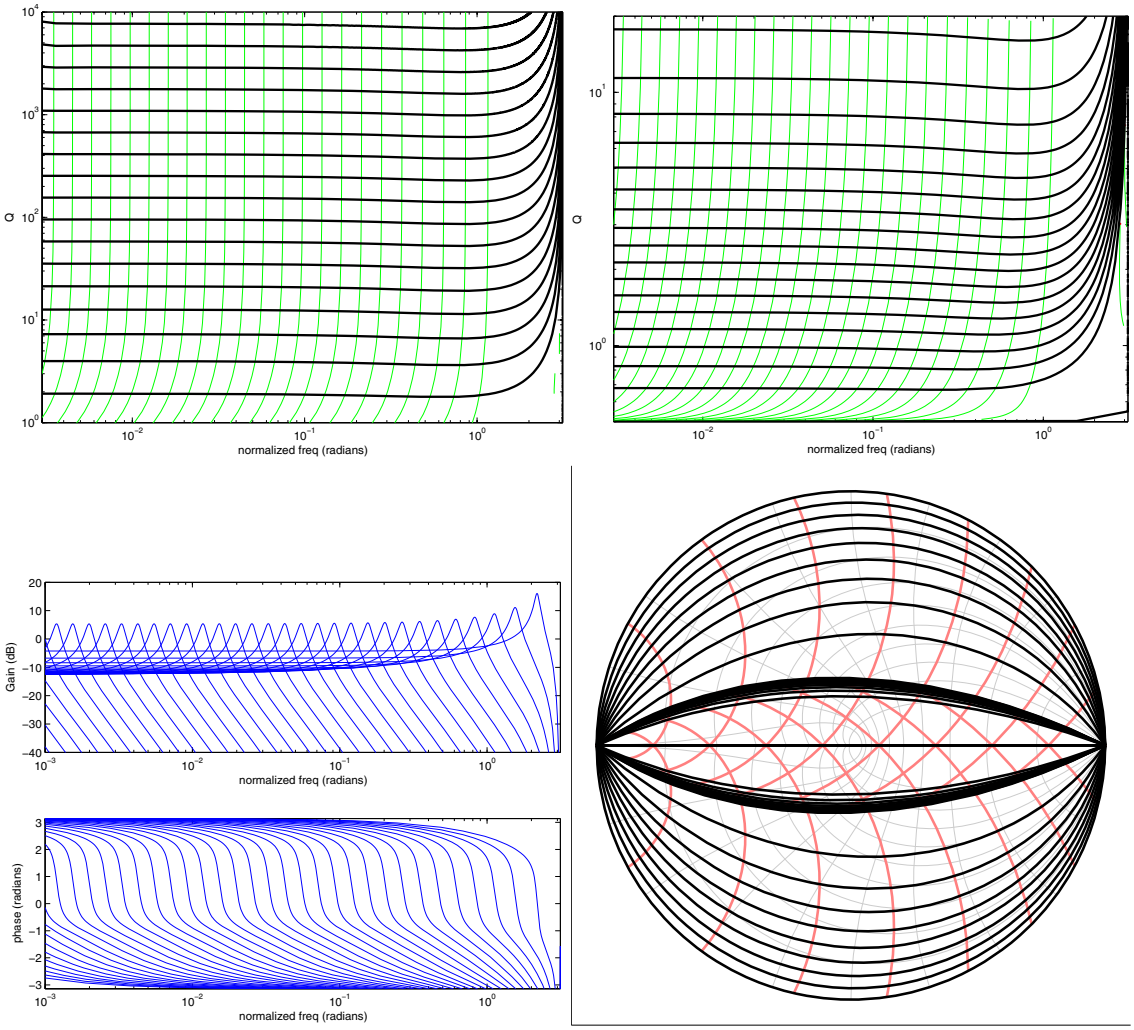


Figure 3.53: Pole Placement: Using separation table shown in Figure 3.44. Top Left: Q vs. p, wide Q range. Top Right: Q vs. p, narrow Q range. Bottom Left: representative frequency responses for a p sweep ($k = 0.8f(p)$). Bottom Right: Pole traces (Dark: constant k, Light: constant p).

Some notes:

- Backward difference with table (Figure 3.49) has some problems at low-Q/high-freq, and the freq variation with Q seems a bit larger than the others.
- Bilinear with table (Figure 3.51) has a slight dip in Q at high freqs, which extends down to low Q's too. Pole placement has a similar dip (Figure 3.53). Looking at the pole traces, we can see that the curves do dip in to lower radii than the current Q contour for a bit at around 1 o'clock to 2 o'clock, before heading out to higher radii.
- We should note that the different shapes of separation tables might imply differences in internal gains between the discretizations which may be significant in implementations using saturation nonlinearities. These may affect discretization decisions in those cases, or require the addition of scales at points in the loop to renormalize signals.
- At very high Q, the sensitivity of Q to gain gets very high. As such, limitations in separation-table sampling density and/or interpolation method can have effects. This can be seen as wiggles in the Q curves in some of these plots. In particularly sparse tables, this issue can become important: lower-order interpolated gains (such as linear) might result in significantly higher Q or instability.

Having shown experimentally that the above "three-tables" postulate is workable for several of the basic discretizations, we therefore set as a restriction to our further design work that tuning/separation be implemented using methods cheaper than the three lookup tables. Now, "cheaper" can have different meanings in different situations. For example it may refer to memory usage, such as in a design that must fit on some no-memory microcontroller, where a lookup-table array might not fit in available fast memory, or it may refer to a cycles-limited design. Therefore, we choose the following rule of thumb:

- a 2^{nd}-order polynomial will be considered the maximum complexity we will feel comfortable with for a mapping.

However, we take as a given that f_{tuning} will have to be a full lookup-table in order to get sufficient tuning precision (and f_Q when necessary). Thus, we apply this restriction mainly to the $f_{\text{separation}}$ mapping.

Admittedly, we have set ourselves a very small design target, as we have just seen that the three-lookup-table method works well, and is not outrageously expensive. We fully expect a large number of usage cases to be satisfied by the above methods, and such designs should be good enough. Still, we believe there are situations where smaller implementations may be of use, and we believe there are interesting things still to be found by looking in that direction.

From here on, we break away from the concept of discretizing the continuous-time filter, and concentrate solely on the discrete-time filter as a type in and of itself. As such, we will no longer

look for a "better discretization", but rather a "better filter" (though specifically staying with the loop topology). We will take as our starting points the discretized filters of the previous section, and look at variants of those filters, ignoring any further relation back to the continuous-time filter.

Our main design objective will be to attempt to flatten the infinite-Q gain curve, on the idea that a flat gain curve can be approximated be a very inexpensive function (a constant). The flatter the curve, the better it is approximated by the constant, and therefore the wider range of Q can be achieved using the constant approximation. Later, we will also look at a variation or two of this basic objective, but for now, we will concentrate on flattening.

If we study the results of the basic discretizations (in particular, Figures 3.48 and 3.50), we note that the gain curve for the backward-difference (with delay) case slopes upward, and the gain curve for the bilinear case (with delay) slopes downward. Remembering that the difference between these two cases is simply the locations of the zeros in the onepole filters, we wonder if there may some intermediate zero location were the gain curve may have a nearly horizontal trend — in other words, a "compromise" between the two behaviors.

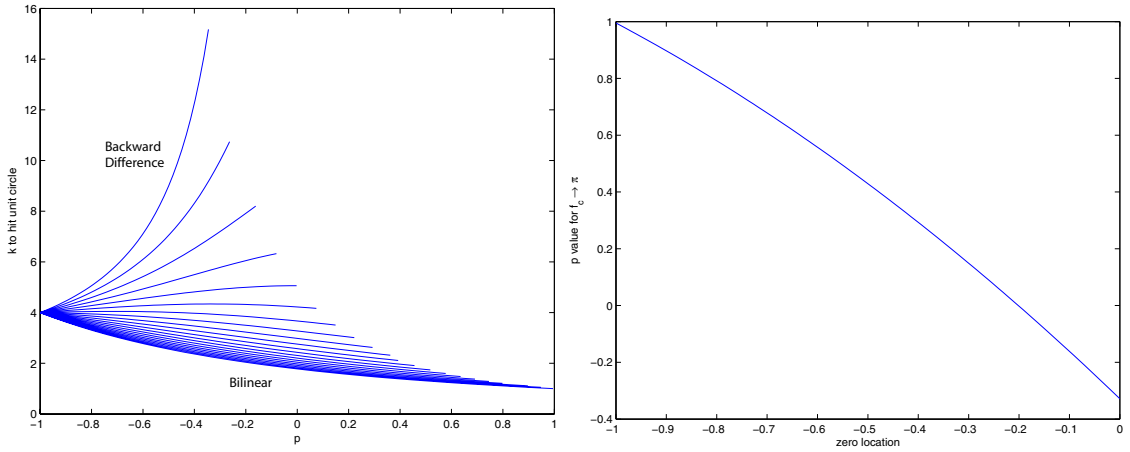


Figure 3.54: Left: The family of gain curves for filters between the Delayed Backward-Difference Filter (top curve) and the Delayed Bilinear-Transformed Filter (bottom curve). Intermediate filters have all their zeros on an intermediate location between $z = 0$ and $z = -1$. Right: p limit as a function of open-loop zero location.

Figure 3.54 is a plot of the family of gain curves. Sure enough, there are curves with a nearly flat trend (though not *exactly* flat). Also note that most of the curves to not end at $p = 1$. As we noticed when introducing the backward-difference filter, the infinite-Q poles reach $z = -1$ before p has reached 1. The intermediate filters also have this property. The right-hand plot of Figure 3.54 shows value of p at which the poles reach $z = -1$, versus zero location (this curve is pretty-well approximated by the polynomial $-0.38273z_0^2 - 1.70596z_0 - 0.32817$, where z_0 is the open-loop zero

location). Beyond these values, the poles move along the real axis. For some filters (the backward-difference included) the poles may again split off the unit circle at higher values of p. This can be seen in the top-right plot of Figure 3.68, which is the backward-difference case we have seen before, but with p allowed to go all the way out to 1 (in Figure 3.48, the p range was limited to $-1 < p < 0$ in order for the locus to be more readable, as otherwise this secondary effect obscured the $p < 0$ locus at high frequencies).

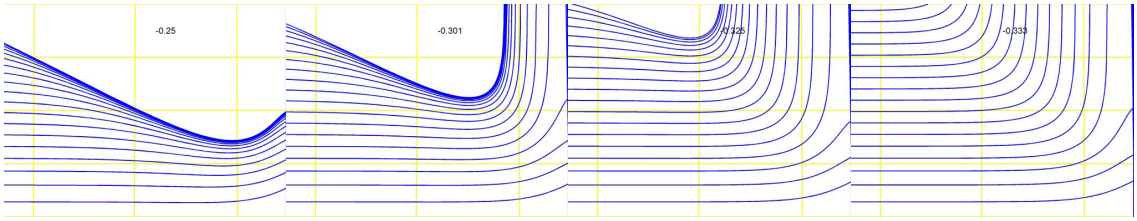


Figure 3.55: Frames from an animation of the Q/f space ("High-Q" range: vertical axis goes up to Q=10000) for various open-loop zeros (raw filters, no separation tables). Left to right: zeros at -0.25, -0.3, -0.325, -0.333. Max gain: 4.0.

Unfortunately, none of the intermediate gain curves is perfectly horizontal. As such, there is some tradeoff in choosing which zero location upon which to base a filter. Figure 3.55 shows four frames from an animation exploring the Q/f space as the zeros move between $z = -0.2$ and $z = -0.4$. $z = -1/3$ jumps out as having an interesting property: $\frac{dQ}{df}$ appears to be zero at DC for all Q tracks, whereas for zeros closer to $z = 0$, the higher Q tracks trend downward from DC for a while.[18] However, nearly all of the Q tracks above some low Q eventually go unstable, and the higher-Q tracks go unstable at lower frequencies, which is considered undesirable. Note that the above behavior can be be explained by the fact that $z_0 = -1/3$ appears to be the case for which the gain curve has zero slope at DC.,

The tradeoff becomes one of placing the bottom of the "dip" in the Figure 3.55 plots. Moving it down/right (i.e., zeros closer to $z = 0$) pushes up the frequencies at which "high-Q" tracks which start stable at DC (i.e. those with $k < 4$) go unstable, at the expense of limiting the maximum Q for which the Q traces are relatively flat from DC up to reasonably high f_c. Moving it up/left (zeros closer to $z = -1/3$) increases the maximum Q range of the "nearly flat" traces, but reduces their maximum f_c. Past -1/3, nearly all the traces go unstable, just faster and faster.

A zero location of $z = -0.3$ was chosen as a viable tradeoff [249] (Figure 3.56 shows the gain curve family with the -0.3 curve highlighted). We call the filter based on this choice the "compromise" filter, since the zero location decision was a practical compromise between a few issues. Note that any zero location in the vague range of $-0.275 < z_0 < -0.325$ probably works as a compromise.

[18]Since these filters have DC gain normalized to 0dB, $k = 4$ corresponds to infinite Q as $f_c \rightarrow DC$ for all open-loop zero locations. However, since the Q tracks trend downwards, there are regions in $f_c > 0$ where gains of $k > 4$ are stable. These regions correspond to the empty areas in the Q/f plots.

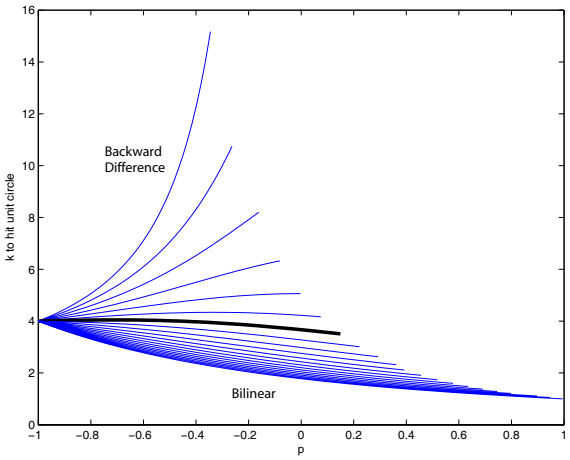


Figure 3.56: Copy of Figure 3.54, with the -0.3 curve highlighted.

The open-loop filters resulting from choosing -0.3 are:

$$H_f(z) = \left[\left(\frac{1+p}{1.3}\right)\frac{0.3+z^{-1}}{1+pz^{-1}}\right]^4, \quad G(z) = z^{-1} \tag{3.61}$$

And we can describe this as "Four zeros on $z = -0.3$, and a delay in the feedback".

We will repeat the analyses of the previous sections on this filter.

The root-locus equation in k is:

$$z(z+p)^4 + k\left(\frac{p+1}{1.3}\right)^4 (z+0.3)^4 = 0 \tag{3.62}$$

i.e., four open-loop poles on $z = -p$ with one more on $z = 0$, and four open-loop zeros on $z = -0.3$. Remember, like the bilinear-with-delay, there is an extra pole.

The root-locus equation in p:

$$\begin{aligned}(13^4 z^5 + k(10z+3)^4) + p(13^4 4z^4 + 4k(10z+3)^4) + p^2(13^4 6z^3 + 6k(10z+3)^4)\\ +p^3(13^4 4z^2 + 4k(10z+3)^4) + p^4(13^4 z + k(10z+3)^4) = 0\end{aligned} \tag{3.63}$$

And the individual coefficient polynomials are:

$$\begin{aligned} D(z) &= 13^4\, z^5 + k(10z+3)^4 \\ N_1(z) &= 13^4\, 4z^4 + 4k(10z+3)^4 \\ N_2(z) &= 13^4\, 6z^3 + 6k(10z+3)^4 \end{aligned}$$

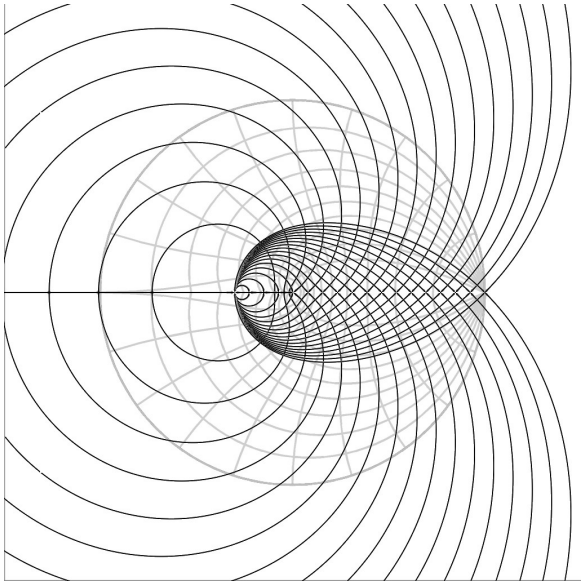


Figure 3.57: Compromise Filter: First-order loci in k for various p between $p = -1$ and $p = 0.3$.

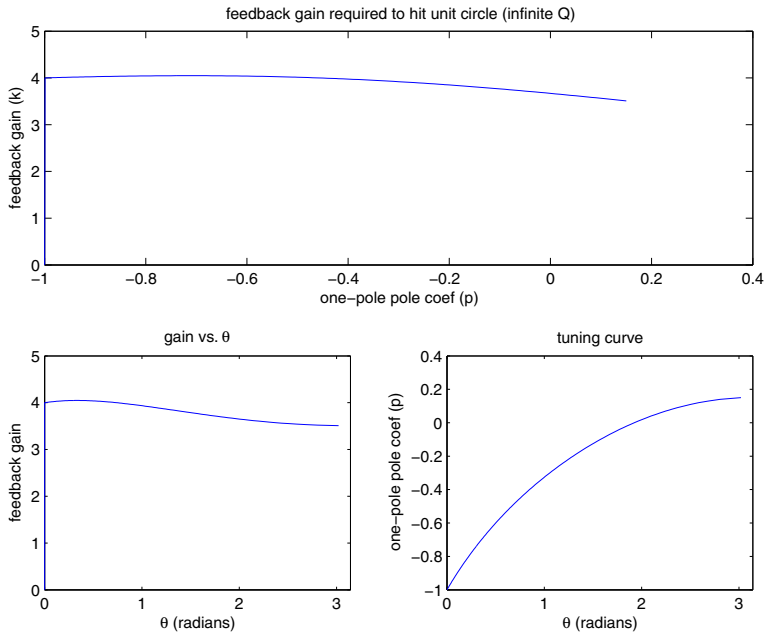


Figure 3.58: Compromise Filter: Top: Loop gain (k) required to hit unit circle, vs. p. Bottom Left: Loop gain vs. pole angle at the unit circle. Bottom Right: Tuning curve (p vs. θ at the unit circle).

$$N_3(z) = 13^4\, 4z^2 + 4k(10z+3)^4$$
$$N_4(z) = 13^4\, z + k(10z+3)^4$$

Even though this filter was designed in an attempt to make a filter which does not need a separation table, we present analysis of the filter with a separation table (Figure 3.61) as well as the analysis without (Figure 3.60).

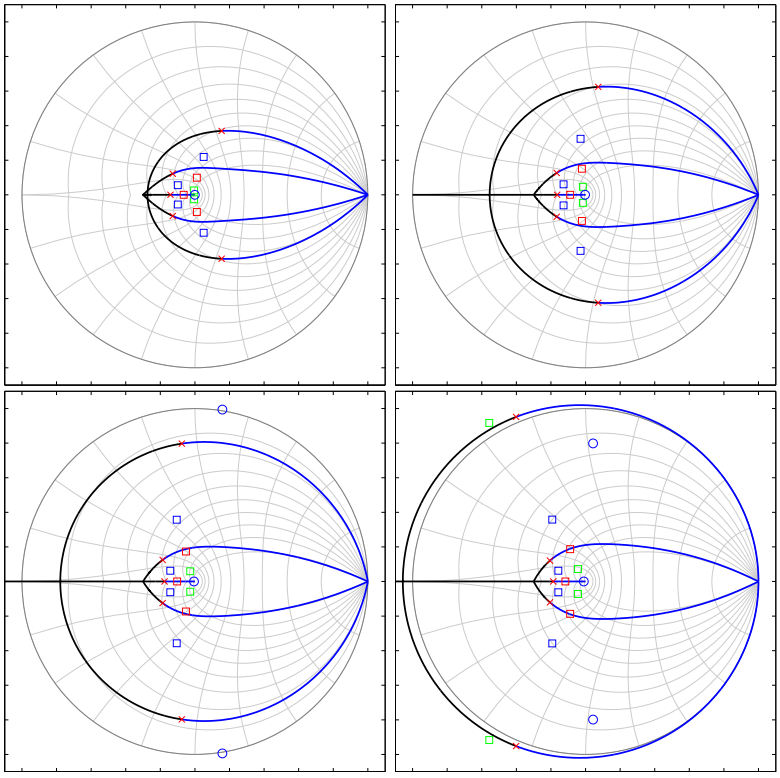


Figure 3.59: Compromise Filter: Fourth-order loci in p ($-1 \leq p \leq 0.3$) for various k: Top Left: $k = 1/4$, Top Right: $k = 1$, Bottom Left: $k = 2$, Bottom Right: $k = 4$. 'x': roots of $D(z)$, 'o': roots of $N_4(z)$, squares: roots of $N_1(z)$ through $N_3(z)$.

Pseudocode for a C implementation of this filter is given in Figure 3.62. Note that this code is not guaranteed to be the most optimal for any particular processor or compiler, and users are encouraged to further optimize as far as possible. This code also implements Direct-Form-2 filter forms, which are not expected to work as well in fixed-point number systems. Any algorithm should be modified as appropriate for the specific architecture that it will be implemented on.

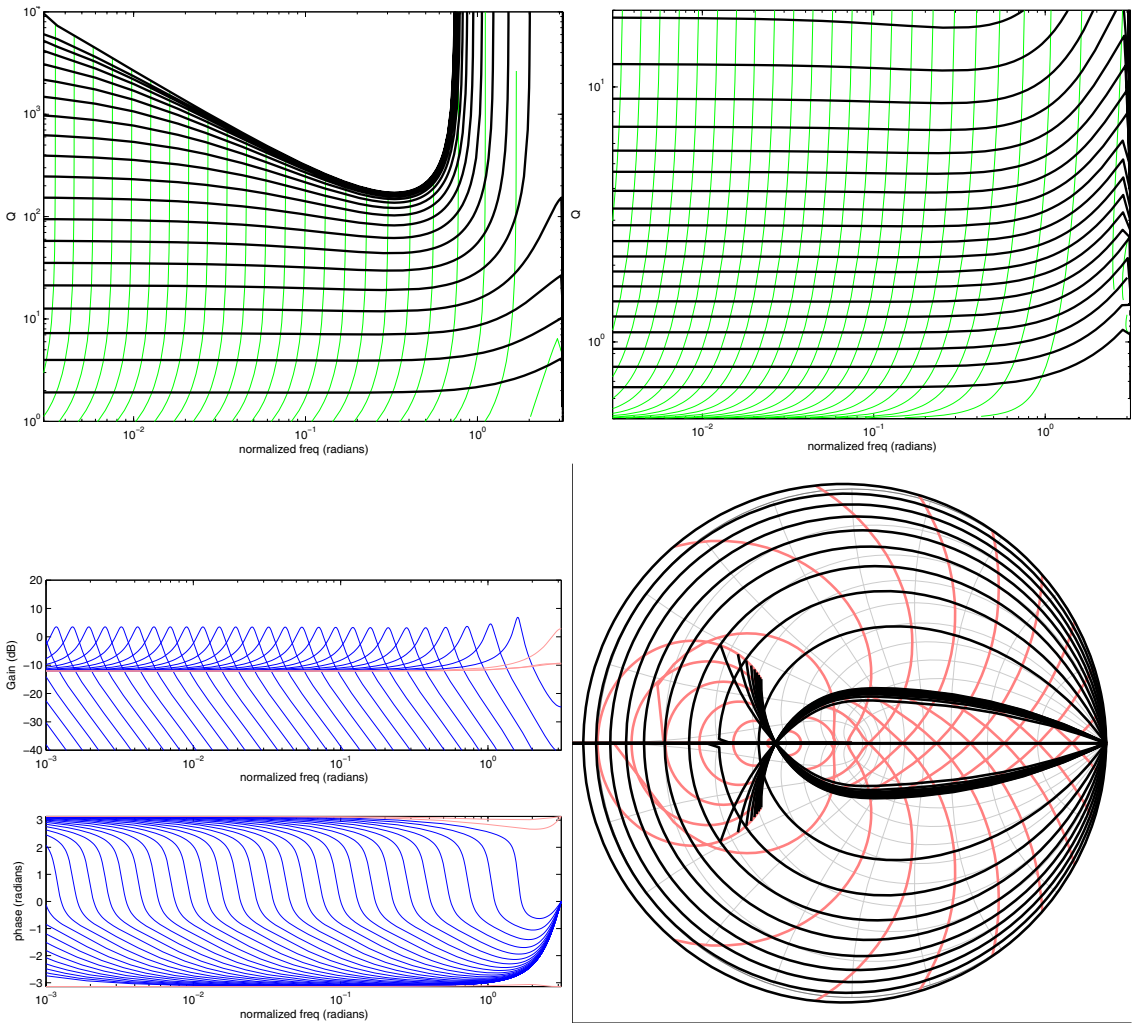


Figure 3.60: Compromise Filter: Raw filter (no Lookup tables). Top Left: Q vs. p, wide Q range. Top Right: Q vs. p, narrow Q range. Bottom Left: representative frequency responses for a p sweep ($k = 3.5$). Bottom Right: Pole traces (Dark: constant k, Light: constant p).

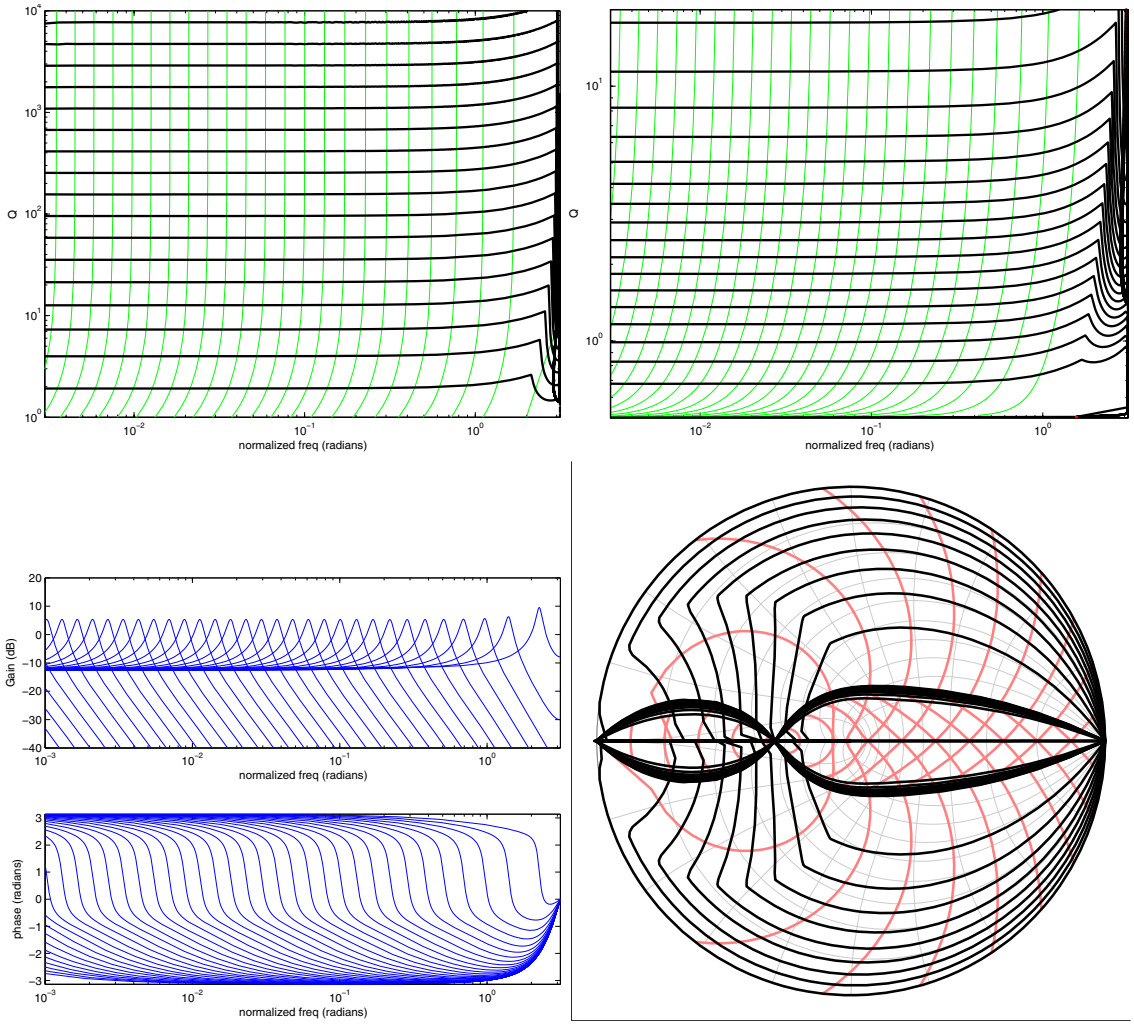


Figure 3.61: Compromise Filter, Using separation table shown in Figure 3.58. Top Left: Q vs. p, wide Q range. Top Right: Q vs. p, narrow Q range. Bottom Left: representative frequency responses for a p sweep (k = 3.5). Bottom Right: Pole traces (Dark: constant k, Light: constant p).

```
s = state->filterStateArray;
A = state->feedbackState;

for (i=0; i<vectorLength; i++) {
    pval = 0.55*tuningCoef[i] - 0.45;   // Bring (-1.0,1.0) into (-1.0,0.1) range

    A = SATURATE(0.25*(in[i] - A));   // Since loop has a DC gain of 2^4 = 16,
                                      // pull it back to 4 here

    // WARNING: DF2 filters (not recommended for fixed-point implementations)
    for (j=0; j<4; j++) {
        tmp = s[j];
        A =  SATURATE((2.0/1.3)*(1.0+pval)*A) - pval*tmp);
        s[j] = A;
        A = SATURATE(0.3*tmp + A);
    }
    out[i] = A;
    A *= gainCoef[i];
}
state->feedbackState = A;

// Deal with denormals on floating-point processors which need it:
for (j=0; j<4; j++)
    s[j] = CLAMP_DENORMAL_TO_ZERO(s[j]);
```

Figure 3.62: C pseudocode for an implementation of the Compromise Filter. This implementation assumes that any desired $f_c \rightarrow p$ and $Q \rightarrow k$ mapping has been done externally to this loop. `SATURATE()` implements whatever saturation is desired (none, hard clipping, polynomial, tanh, LUT, etc.). This implementation can be considered to contain "too many" saturations, and cheaper implementations should limit the number of expensive saturations in the loop. Note that this code demonstrates DF2 implementations for the onepole filters. One may prefer DF1 or TDF2 implementations in that case, as well as possibly modifying how gain is distributed throughout the loop.

Zeros and output combinations There is a good chance that moving the open-loop zeros to locations other than $z = 0$ (or $z \to \infty$) will effect the ways that outputs are combined to produce various gross filter shapes. However, the analysis of this effect had to be dropped for time in the course of finishing the research for this thesis.

Digression: Setting Q

For most accurate Q setting, one would measure Q vs. k at some frequency (typically a low audio frequency), and from those measurements create a $Q \to k$ table. However, a basic rule of thumb can be theorized: for any particular f_c, if the gain required to hit the unit circle (i.e., the gain curve) is k_{max} (either when implemented as $k_{max} = f(p)$ or as a constant), then:

$$k \approx k_{max}\left(1 - \frac{2}{Q}\right) \tag{3.64}$$

Such approximations tend to be best for high Q, and become less accurate as Q gets on the order of 1.0 and less. See Figure 3.76 for an experimental verification of this rule of thumb for a later filter design.

3.3.3 Beyond the Compromise Filter

In 1998 Duane Wise [293] discussed building a filter on the same basic paradigm (a number of first-order sections with feedback around them) using allpass filters as the first-order sections. From the standpoint of separation, this compelling, as the allpass nature of the feedforward cascade guarantees that the poles will reach the unit circle at $k = 1$ regardless of tuning. However, one must practically be limited to two first-order stages in the loop, as all the poles will reach the unit circle at the same time, so if there are more than two poles, the total filter will exhibit multiple peaks (which is usually not desired). As an example, Figure 3.63 shows a family of loci in loop gain for a filter consisting of two first-order allpasses in a loop (plus a delay to make the loop implementable) for various values of the allpass coefficient between −0.95 and 0.95. Note that in this example, the delay causes a third pole, which will unfortunately reach $z = -1$ at the same gain as the other poles reach the unit circle, since the loop is allpass. A workaround (discussed by Wise) is to put zeros on $z = -1$ in cascade with the filter to keep energy from this mode, as well as assist in achieving a lowpass response.

This method achieves perfect separation at infinite Q, due to the use of the allpass in the loop, and as such is another useful direction for Moog-style filter design, but since this thesis mainly concentrated on four-stage designs (for which this concept does not work well given the multiple-peak issue), this direction was not deeply explored in this research. Interestingly, this method may be roughly viewed as an intermediate filter type between state-variable designs (which also have

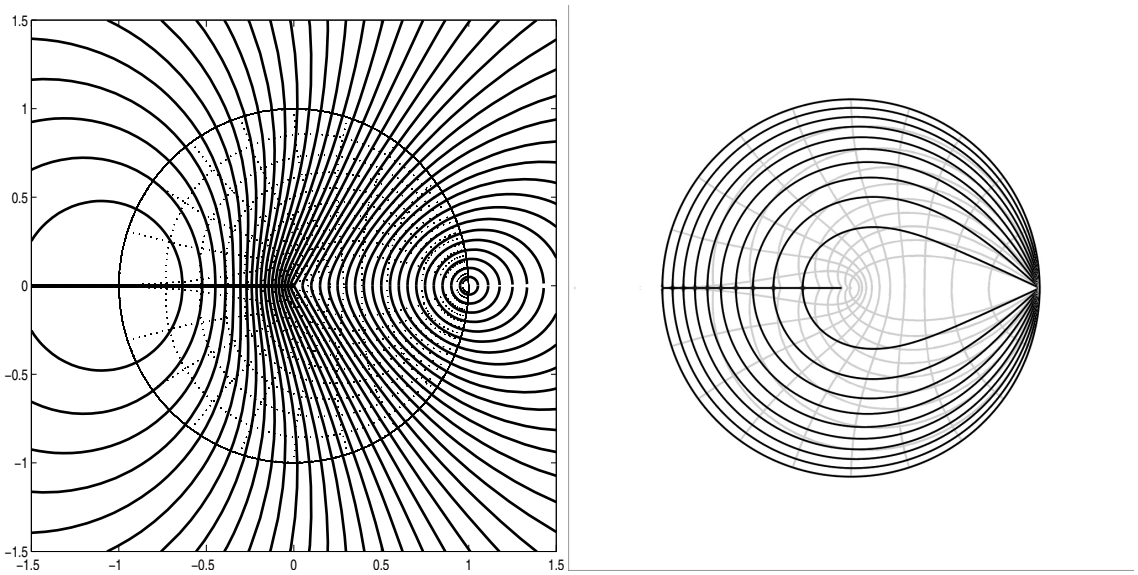


Figure 3.63: Loci families for a system consisting of two first-order allpass filters in a loop (with a delay to make the loop implementable). Left: Loci in feedback gain for various allpass poles between −0.95 and 0.95. Right: Loci in pole location for various feedback gains between 0.0 and 1.0.

perfect separation at infinite Q, but are only 2$^{\text{nd}}$-order systems) and the 4$^{\text{th}}$- and 5$^{\text{th}}$-order single-loop topologies explored in this part of this thesis.

One may ask the question "Do we need to go beyond the Compromise filter design?" For low-Q usage cases, the compromise filter works just fine for a large range of corner frequencies. However, if one needs a wider Q range without bumping up against loop instability (or inconsistent self-oscillation vs. frequency if using saturation within the loop), or if one needs less Q variation for a given k value at high-Q ranges (and one doesn't want to use separation tables), further tweaking of the design may be desired.

From the standpoint of the thesis, it was assumed that one would want a filter with a flatter separation table, simply in order to have a reason to continue exploring the filter design space.

Attempts at optimization-based root-locus track fitting

One area of interest at this time was exploring whether some sort of optimization could be used to help design filters in the Moog style. The primary concept would be: "optimize a feedforward filter to have a constant-Q locus." Practically, this meant to explore the viability of programs such as:

$$\text{minimize}_{N,D} \qquad \|\text{err}(Z(N,D), p_{des})\|_p$$

s.t. various constraints

Essentially, minimize the "distance" between the root locus $Z(N, D)$ (i.e., the zero set of $D(s) + kN(s)$ for all k) and a set of points p_{des} along a desired track, using the polynomials $N(s)$ and $D(s)$ as the free variables (or maybe their root locations). Further intended design directions would be to optimize feedforward filters more in the Moog-style topology (i.e., a chain of first-order filters with). Possible constraints may be to try to have the fitted track be the only track (or the highest radius, or that there are no unstable tracks in the same k range, etc.). Since the Moog-style filters have two major control parameters, f_c and Q, we should really optimize for loci in both parameters at once, fitting a surface in (s, Q) space (or (s, f_c) space)), which should theoretically mean just a slightly larger program. However, initial experiments were performed using a locus in just one parameter.

There is an active field of research in the computer-graphics and image-processing fields on the fitting of curves and surfaces to point data. If we look at the wide field of fitting any representation, the field is enormous. Of particular interest recently has been the fitting of parameterizable surfaces to range data, as comes from 3-D scanning devices (with pioneering work being done here at Stanford by Marc Levoy and his group, a representative paper being [145]).

Root loci can be viewed as implicit curves, so a subgenre of particular interest is that of fitting implicit curves and surfaces to data. By removing explicit k from the fitting, we would not overly restrict the fitting by requiring a particular tuning curve. Though the topic existed before, it had a bit of a revolution with Gabriel Taubin's work starting in 1991 ([261] [258] [259]) which culminated in [260], which presented the most accurate distance approximations up to that time. As discussed in that paper, though the closest point on an algebraic curve can be found exactly, the math is not efficient, effectively requiring the enumerating of the intersections between the curve in question $(f(x, y))$ and another derived implicit curve $(f'(x, y))$ which depends on the original curve and the point $(p = (u, v))$ being checked:

$$f'(x, y) = (x - u)\frac{\partial}{\partial x}f(x, y) - (y - v)\frac{\partial}{\partial y}f(x, y) \tag{3.65}$$

These intersection points are the points where the curve f is perpendicular to the point p. We note in passing that $f'(x, y)$ for a root-locus equation is another root-locus equation, but one with complex coefficients. In general, that method is taken to not be efficient. Instead, Taubin described a method for first-order, second-order, and higher-order approximations to the distance which are quite accurate within an impressive region surrounding the implicit curve or surface. As discussed in Appendix A, this leads immediately to an extremely fast and versatile root-locus rendering method as well as to fitting methods.

Since the 1994 paper, Taubin presented further work on fitting in [263]. About the same time, Baja and group described an implicit-surface fitting algorithm for use as in interpolation method.

1996 was a big year for implicit-function fitting, with Redding and Newsam presenting two papers on the subject, in the field of image understanding ([215] and [216]), Varah discussing least-squares fitting of implicit functions [281], and Lei and Cooper presenting an improved algorithm [155]. In 2000, Blane joined Lei and Cooper to present an updated algorithm which became quite popular (the "3L" algorithm). Recently, Helzer described more updates to the techniques [106] in 2004, and Sahin and Unel continued in 2005. Both of these were focused on making the fitting algorithm more stable in the presence of certain difficult situations. On a familiar note for root locus, Gao and Li described in 2004 an algorithm for fitting rational plane curves.

However, most of these types of algorithms do not fit well into the proposed problem. Two basic issues:

- In most of the methods, the implicit function's order is quite freely variable, and the best fits use rather high function orders. However, in our variable filters, we have an order in mind and do not particularly want to add further roots into the system. Further, our filter orders are rather small compared to orders often used in such fitting algorithms, so fitting to arbitrary shapes is quite limited. In practice, we need to prove to ourselves using some other method (experimentation, usually) that the filter order is capable of a particular shape, and only then try to optimize to that shape.

- It is usually fine in graphics to approximate a curve or shape with a combination of implicit functions which only fit parts of the surface, so that the total fit is from several equations. In variable filters, the concept of "handing off" to another filter part-way through the range of the control is not desirable (at least for Moog-style filtering, as it is strongly hoped that there is a single filter which can handle the whole range). It may be applicable in some areas of variable filter design, though, as multi-filter designs are not unheard of (such as the Farrow structure [77], and vector-array decomposition-based filter designs [62]).

Early-on, the author experimented with approximating a constant-Q locus curve by alternating some first-order poles and zeros on the real axis, on the concept that controlling their relative locations could be used to "push and pull" on the locus to give the desired shape. Early hand-fitted experiments, such as that shown in Figure 3.64, seemed promising, but they did not translate to automatic optimization methods very well at all,[19] and the fit quality appeared to be tied to the order of the open-loop system in an undesirable way.

Interestingly a paper on fitting root loci to desired tracks was found. Prokhorova described in 1991 a method [207] effectively using

$$\sum_{s_i \in S_{des}} \left| \mathrm{Im}\left(\frac{N(s_i)}{D(s_i)} \right) \right|^2 \tag{3.66}$$

[19] Though it may be that just not enough time was put in to working out bugs.

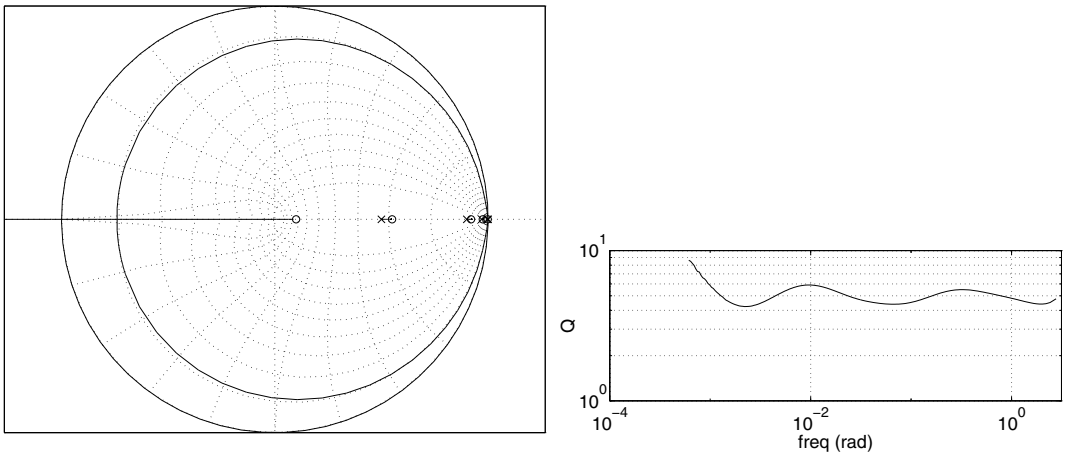


Figure 3.64: Manual attempt at fitting a constant-Q trajectory using real open-loop roots as optimization variables. Left: locus, Right: Q

As the error function. Unfortunately, this is a rather weak distance measure, and allows quite large errors in regions where the measure is quite low. Further, it is pretty badly non-convex. Still, it did provide a starting point for further exploration. A variation on this measure, vaguely inspired by Prony's method for rational-filter design, was to note that

$$\mathrm{Im}\frac{D(s)}{N(s)} = \frac{D(s)N(s^*) - D(s^*)N(s)}{2j|N(s)|^2} \tag{3.67}$$

and take the numerator, to make the program

$$\text{minimize}_{D,N} \sum_{s_i \in s_{des}} |D(s)N(s^*) - D(s^*)N(s)|^2 , \tag{3.68}$$

on the hope that this would make for a program that was closer to convex. It was still non-convex, but it appeared that the number of local minima may be small and manageable. Still, this measure exhibited the same problem with allowing large errors in locus shape where the gradient of the distance measure got small. Finally, it was noted that there is a trivial zero in this measure when $N(s) = D(s)$, which is not the desired result (so it would have to be constrained somehow).

In searching for distance measures, the brute-force method (i.e. calculating a root locus and directly finding the distance to desired points) was considered too expensive for optimization work,[20] since it would require the calculation of a full root locus at least once per optimization iteration.

There was some experimentation into performing some sort of gradient-descent on the above

[20] not to mention inelegant

function, on the hope that it would arrive at the locus at the point closest to the starting point. Unfortunately, that assumption was not true. However, some experiments in using Newton's method on the function produced an intriguing form of Newton's-Method fractal (Figure 3.65, for example), where rather than there being discrete zeros and thus discrete regions of attraction, the zero set is continuous, and the "regions of attraction" are a more amorphous concept.

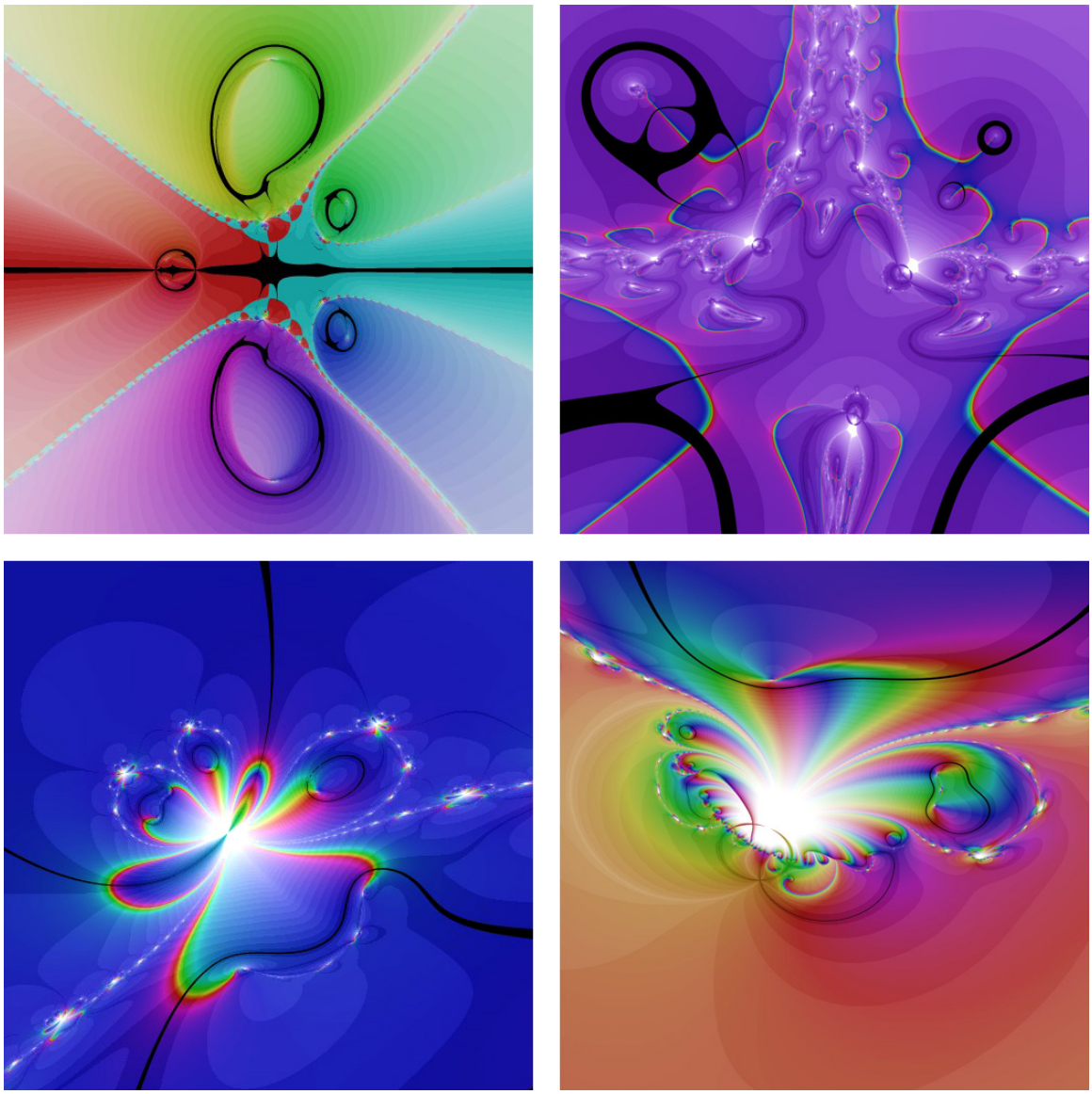


Figure 3.65: Newton-Fractals and details for various first-order root-loci.

In the end, Taubin's first- and second-order distance approximations appeared to be the best

Locus equation: $D(s)+kN(s) = 0$, hence full locus is solutions of $\mathrm{Im}(D(s)/N(s)) = 0$. Now if $D = dr + jdi$, and $N = nr + jni$, then

$$\begin{aligned}\frac{D}{N} &= \frac{(nrdr + nidi) + j(nidr - nrdi)}{dr^2 + di^2} \\ &= \frac{\mathrm{Im}(D(s)N^*(s))}{|D(s)|^2} \\ &= \frac{\mathrm{Im}(D(s)N(s^*))}{|D(s)|^2} \text{ because N has real coefs}\end{aligned}$$

Now we only need to look at the numerator in order to trace where the zeros are, so we perform Newton's method on the function

$$f(s) = D(s)N(s^*) \tag{3.69}$$

Or, in terms of the real components ($D = dr+jdi$, $N = nr+jni$, $dD/ds = ddr+jddi$, $dN/ds = dnr + j\ dni$, $s = sr + j\ si$), the Newton iteration is:

$$\begin{aligned}dr + j\ di &= D(s) \\ nr + j\ ni &= N(s) \\ ddr + j\ ddi &= \frac{dD(s)}{ds} \\ dnr + j\ dni &= \frac{dN(s)}{ds} \\ \partial_{Im} &= (dnr\ dr - dni\ di) - (nr\ ddr - ni\ ddi) \\ \partial_{Re} &= (dnr\ di + dni\ dr) + (nr\ ddi + ni\ ddr) \\ x &= \frac{(nr\ di + ni\ dr)}{\partial_{Im}^2 + \partial_{Re}^2} \\ sr &= sr - x\partial_{Re} \\ si &= si - x\partial_{Im}\end{aligned} \tag{3.70}$$

Figure 3.66: Newton's method on a root locus (1st-order in k):

upon which to base optimization measures. Still, there were quite a few issues that slowed progress.

General convexity issues In general, trying to fit locus tracks to points is not a convex problem. Aside from any issues as to the convexity of any particular optimization variables to any particular root locus shape, there are issues with the convexity of distance measures and the multiplicity of ways in which a curve might be seen as "fitting" a set of point data. Since point-distance measures effectively ignore the behavior of the curve away from the points, any combination of curve behaviors can be exhibited away from the fitting points and still have sections of the curves which pass through or near the fitting points. There can even be situations where different tracks might "trade off" the fitting, such that one track is near one subset of the fitting points but a completely different track is near another subset. As such, quite a bit of work needs to go into controlling these behaviors, such as coming up with a number of program constraints, or restricting the feedforward topology to ones which are known to generate desirable locus shapes.

Even so, some experimentation was done using global optimization packages, such as ASA, VerGO, etc. The primary results were to confirm the difficulties mentioned previously.

In general, a couple of problem areas were noticed (beyond ones mentioned previously), which make locus fitting for audio filtering a difficult problem:

- The need to fit a surface rather than a curve in order to optimize over both controls. Within the optimization community, this is not considered a problem, but it does complicate things, especially if the problem is non-convex.
- There is significant amount that must be handled in constraints, including enforcing that the fitted tracks are those of dominant poles (and thus, trivially, any other poles are not unstable). For example, some of the best fits have had other pole tracks that stayed completely outside the unit circle.
- Most fitting tends to put pressure on increasing the filter order in order to get a better fit.

Finally, it was realized that z-plane distance measures are not well-suited for Q-based filter design, as the sensitivity of Q in pole location goes to infinity at the unit circle. In other words, very small errors in z distance can imply very large errors in Q for high-Q tracks. As such, much better distance measures should be in some $(\log(f), \log(Q))$ plane. Now we will see in Appendix A that certain distance measures, such as Taubin's approximations, can warp to such planes just fine, but in so doing, the concept moves away from the original idea of root-locus track fitting, and as such became less exciting for work being done using a root-locus perspective. However, further work on fitting should probably follow this direction.

In summary, these locus-track optimization directions were not heading towards promising results in the near term, whereas other directions we will discuss in upcoming sections were proving fruitful.

Comparing the Discretizations Further

In Section 2.3, we noted that it is possible to offset k in a root locus, and for filters where the DC end of the frequency tracks do not correspond to $k = 0$ (or $p = 0$, or whatever the tuning variable is), such a transformation might make the coefficient polynomials $(D(z), N_i(z))$ more intuitively interpretable. Here we apply that notion to the discrete-time Moog-style filters we have looked at so far.

Backward Difference with Delay: If we take Equation 3.43 and change variables as $p = p_0 - 1$, such that $f_c = DC$ now corresponds to $p_0 = 0$, rather than to $p = -1$, then the root locus equation becomes:

$$(z-1)^4 + 4(z-1)^3 p_0 + 6(z-1)^2 p_0^2 + 4(z-1)p_0^3 + (1+kz^4)p_0^4 = 0 \tag{3.71}$$

As such, the roots of the coefficient polynomials are:

$$\begin{aligned} D(z): &\quad [1,1,1,1] \\ N_1(z): &\quad [1,1,1,\infty] \\ N_2(z): &\quad [1,1,\infty,\infty] \\ N_3(z): &\quad [1,\infty,\infty,\infty] \\ N_4(z): &\quad [\sqrt[-3]{k}] \end{aligned}$$

Now, while the roots of the coefficient polynomials of Equation 3.43 aren't that difficult to interpret, these are even easier to interpret (especially in light of the continuous-time locus , Section 2.5).

Bilinear with Delay: Applying the same transform $p = p_0 - 1$ to Equation 3.57, we get:

$$16z(z-(p_0-1))^4 + k(z+1)^4 p_0^4 = 0 \tag{3.72}$$

or, expanded in terms of powers of p_0:

$$16z(z-1)^4 + 64z(z-1)^3 p_0 + 96z(z-1)^2 p_0^2 + 64z(z-1)p_0^3 + 16z + k(z+1)^4 p_0^4 = 0 \tag{3.73}$$

As such, the roots of the coefficient polynomials are:

$$\begin{aligned} D(z): &\quad [0,1,1,1,1] \\ N_1(z): &\quad [0,1,1,1,\infty] \\ N_2(z): &\quad [0,1,1,\infty,\infty] \\ N_3(z): &\quad [0,1,\infty,\infty,\infty] \end{aligned}$$

$$N_4(z): \quad \text{[complicated]}$$

Now, in comparison to Equation 3.57, in which all the coefficient polynomials had complicated roots, the transformed locus only has complicated roots for N_4, all the others are very straightforward, and again we get nearly the same pattern of roots on $z = 1$ as with the backward difference. The extra roots at $z = 0$ come from the fact that this filter is actually fifth-order due to the delay that was added to make it implementable.

Pole/Zero Mapping: Again, applying the transform to Equation 3.60, we get:

$$8(z-1)^4 + 32(z-1)^3 p_0 + 48(z-1)^2 p_0^2 + 32(z-1)p_0^3 + 8 + k(z+1)^3 p_0^4 = 0 \tag{3.74}$$

And the roots of the coefficient polynomials are:

$$\begin{aligned} D(z): &\quad [1,1,1,1] \\ N_1(z): &\quad [1,1,1,\infty] \\ N_2(z): &\quad [1,1,\infty,\infty] \\ N_3(z): &\quad [1,\infty,\infty,\infty] \\ N_4(z): &\quad \left[\sqrt[3]{\tfrac{-8}{k}} - 1\right] \end{aligned}$$

Again, the roots of the coefficient polynomials of Equation 3.60 are all complicated, whereas the transformed locus equation is much more interpretable, and yet again, we get the same pattern in $D(z), N_1(z), N_2(z)$, and $N_3(z)$.

Compromise Filter: Transforming Equation 3.63, we get:

$$(13^4)\left(z(z-1)^4 + 4z(z-1)^3 p_0 + 6z(z-1)^2 p_0^2 + 4z(z-1)p_0^3\right) + 13^4 z + k(10z+3)^4 p_0^4 = 0 \tag{3.75}$$

And the roots of the coefficient polynomials are:

$$\begin{aligned} D(z): &\quad [0,1,1,1,1] \\ N_1(z): &\quad [0,1,1,1,\infty] \\ N_2(z): &\quad [0,1,1,\infty,\infty] \\ N_3(z): &\quad [0,1,\infty,\infty,\infty] \\ N_4(z): &\quad \text{[complicated]} \end{aligned}$$

This is quite similar to the bilinear-with-delay case, as it is based on it, and as such has the extra set of roots on $z = 0$.

Comparison: Comparing these transformed locus equations, we see that this transformation does indeed make the locus equations more understandable, as a common structure among the coefficient polynomials becomes clear. Further, aside from extra zeros in the fifth-order filters, all these filters have the same roots for all the coefficient polynomials except $N_4(z)$. Although not included here, we also get the same results if we transform the forward-difference version, and the two non-implementable discretizations (pure bilinear and pure backward difference). In essence, we can theorize that the only difference between these designs is where the $N_4(z)$ roots end up.

As such, this may suggest a further research direction: attempting to design a Moog-style filter by optimization, where the free variables are the roots of $N_4(z)$ (and possibly some scalings on the coefficient polynomials). This would make for a much smaller optimization space for attempting to design such filters than trying to optimize over all five polynomials.

A difficulty in working in this direction, being purely root-locus based, is the problem, once a good locus is found, of translating that back into a Moog-like filter topology, with its nice "physical-model" structure. This is an area of future research.

Further work on trying to flatten the separation curve

Here we tackle the problem of further varying the structure of the Moog-style filter in order to get an even flatter separation curve (i.e. the gain required to high infinite Q vs. p). But first, a digression:

On practical calculation of the separation curve. the author has made use of two different methods in the course of this research. First off, it is not easily found in closed-form (the equations get ugly fast), so these are numerical methods. The original method was simply to perform a binary search in feedback gain (Algorithm 1). The algorithm may be further augmented by noting if the poles have hit the real axis (or simply by storing away an additional array of outer pole locations corresponding to $k_{sep}(\mathbf{P})$), as the range of p for which the poles are complex may not be known ahead of time.[21]

This algorithm is quite robust (as long as the required gain is within the search range), but it was noted that the gain can be found via another bit of knowledge about the filter topology: the poles will only hit the unit circle where the phase angle of the open-loop transfer function (let's call it G for now) is a multiple of 2π (as long as we're only considering positive gains), thus we can find the required gain by finding all frequencies θ_i with the required phase angles, and then simply invert $|G|$ at that point (Algorithm 2). The function `find2PiPhaseFreqs` in this algorithm may use

[21] Note that a similar binary search can (usually) be used to directly find the value of p at which the poles hit the real axis.

Algorithm 1: Binary Search for Calculating Separation Table

Data: Filter $H_{p,k}(z)$, and set of p values $\mathbf{P}$.
Result: Values of $k_{sep}(\mathbf{P})$ which put poles on unit circle.

foreach $p \in \mathbf{P}$ **do**
  // Search in range $(k_0 - 2\Delta k_0, k_0 + 2\Delta k_0)$
  $k \leftarrow k_0$, $\Delta k \leftarrow \Delta k_0$;
  while $\Delta k > \varepsilon_k$ **do**
    $\mathbf{r}_i \leftarrow$ roots$(H_{p,k}(z))$;
    $r_{max} \leftarrow \max|\mathbf{r}_i|$;
    if $|r_{max} - 1| < \varepsilon_r$ **then break** ;
    if $\max|r_i| > 1$ **then**
      $k \leftarrow k - \Delta k$;
    else
      $k \leftarrow k + \Delta k$;
    end
    $\Delta k \leftarrow \Delta k/2$;
  end
  $k_{sep}(p) \leftarrow k$;
end

Algorithm 2: Phase Search for Calculating Separation Table

Data: Open-loop Filter $G_{p,k}(z)$, and set of p values $\mathbf{P}$.
Result: Values of $k_{sep}(\mathbf{P})$ which put poles on unit circle.

foreach $p \in \mathbf{P}$ **do**
  $\theta_i \leftarrow$ find2PiPhaseFreqs$(G_{p,k}(z))$;
  $k_{sep}(p) \leftarrow \min\left(\left|G_{p,k}(e^{j\theta_i})\right|^{-1}\right)$;
end

any number of methods for finding the frequencies at which the phase passes through multiples of 2π: binary search, variations on Newton's method, etc. In practice, however, it has been noted that care has to be taken to handle if the search method diverges (as has happened on some occasions with p near DC, presumably due to numerical issues). In general, this algorithm is a bit faster than the binary search algorithm. When using Newton-like search methods, this algorithm also allows the result of the previous iteration in p to be used as the starting-point of the search for the next p, possibly accelerating it by several steps. This type of acceleration is not as viable in the binary search method.

This method can be further modified if $G(z)$ is known to consist of four identical one-pole filters ($G(z) = G_0(z)^4$), and if there is no additional delay in the loop, in which case we know that $\angle G(z) = 4\angle G_0(z)$. i.e., we search for frequencies where the phase angles of the single one-pole filters pass through $\pi/2$. This problem is simple enough to be solvable in closed-form. However, since this method requires no additional delays in the loop, it can't be applied to many of the filters we

are considering. Extending this to handle a unit delay in the loop makes it such that the closed-form solutions are no longer straightforward, and the method becomes no more efficient or elegant than looking directly at the phase angles of $G(z)$.

Flattening the separation curve: Pole Placement Zeros

The Compromise Filter was based upon the bilinear-transformed filter, and hence had an extra delay in the loop to be implementable. The Pole/Zero Placement transform, on the other hand, eliminated the need for the extra delay by making one of the feedforward onepoles a forward-difference form. What if we repeat the Compromise design, but based on this form instead? The Compromise filter was designed by choosing a location for the zeros which had a nearly flat separation curve. This design would similarly choose a location for the remaining three zeros of the pole-placement form.

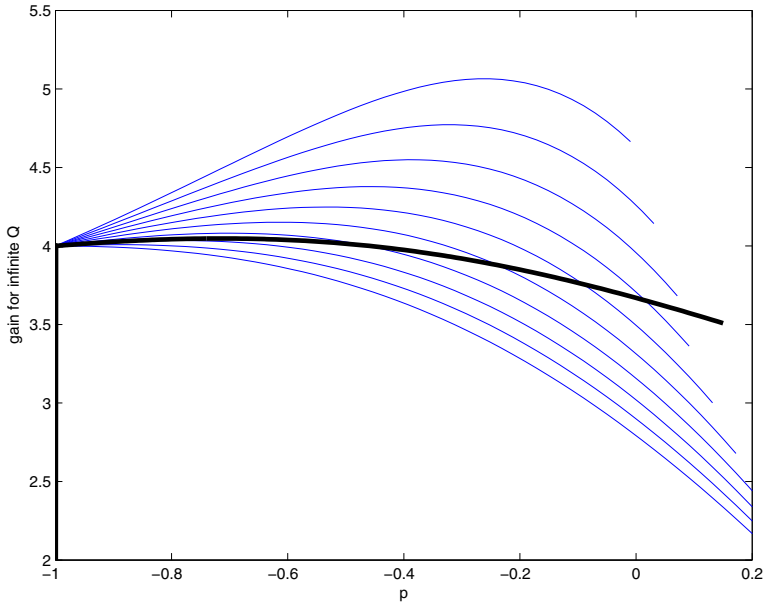


Figure 3.67: Separation curves for modified pole-placement-transform filter. Open-loop zeros ranging from 0.25 (top) to 0.5 (bottom). The separation curve for the Compromise filter is superimposed on top (thick line)

However, if we plot separation curves for prospective zero locations (Figure 3.67), we see that the curves have a much wider variance than the curve of the Compromise filter, hence "less flat," and thus less desirable.

Other pole-placement variants? Remember that the pole-placement-transformed filter had open-loop zeros at $[-1, -1, -1, \infty]$, and the backward-difference-with-delay filter had open-loop zeros at $[0, 0, 0, \infty]$. What if we try other mixtures of zeros? Figure 3.68 shows a set of loci for filters, starting with all open-loop zeros at $z = 0$ (i.e., the pure backward difference), and replacing one zero at a time until we have the forward-difference filter. There is a recognizable trend in the locus shapes as more and more of the open-loop zeros move out to infinity. The same comparison can be done between bilinear (all zeros at $z = -1$) and forward difference (Figure 3.69). Note that the loci are basically the same shape as in Figure 3.69, just scaled up such that the left-hand extent is $z = -1$ rather than $z = 0$, though the difference relative to the unit circle can be significant.

In the spirit of the Compromise Filter, we can explore the space of zero locations between the backward-difference case and the bilinear case (zeros at $z = 0$ and zeros at $z = -1$) for the two cases we haven't looked at yet (the four-finite-zeros case is unimplementable, the three-finite-zeros case we just looked at, and the no-finite-zeros case is the forward difference that we looked at in Section 3.3.1.)

Looking at the families of separation curves for the two remaining various numbers of forward-difference onepoles (Figure 3.70), it appears that neither of these variants is a promising direction for further exploration.

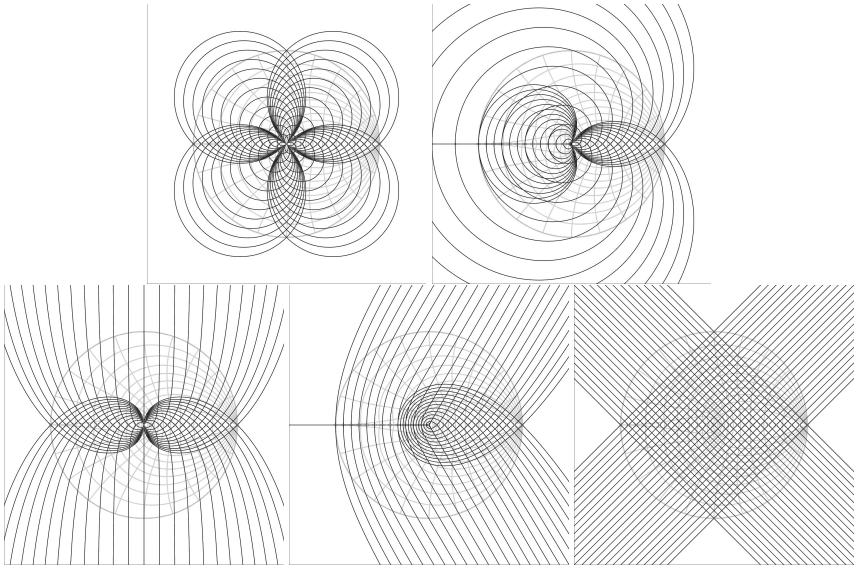


Figure 3.68: First-order loci in k for various p between $p = -1$ and $p = 1$. Left-to-right, top-to-bottom: Four zeros on $z = 0$ (backward-difference with no delay), three zeros on $z = 0$ (backward-difference with delay), two zeros on $z = 0$, one zero on $z = 0$, and no finite zeros (i.e., fwddiff).

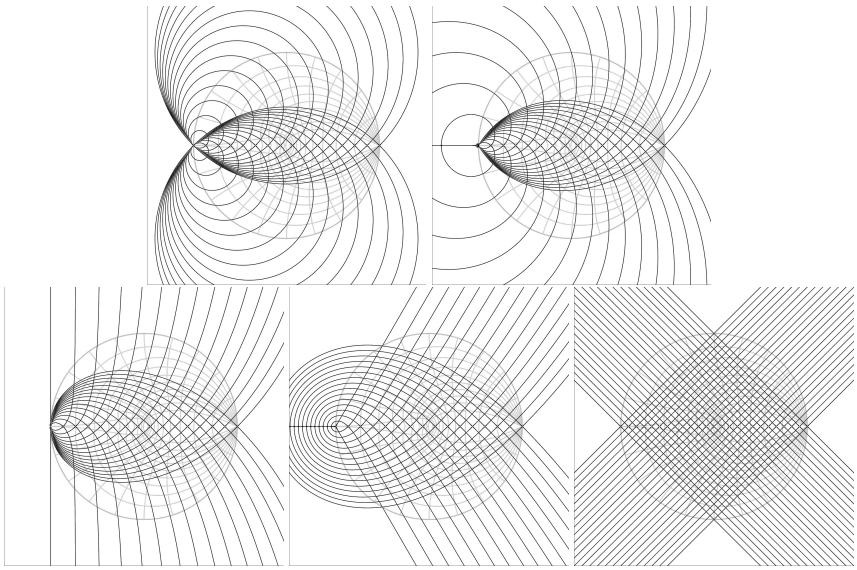


Figure 3.69: Same as Figure 3.68, but with bilinear onepoles rather than backward-difference (i.e., finite open-loop zeros at $z = -1$ rather than $z = 0$).

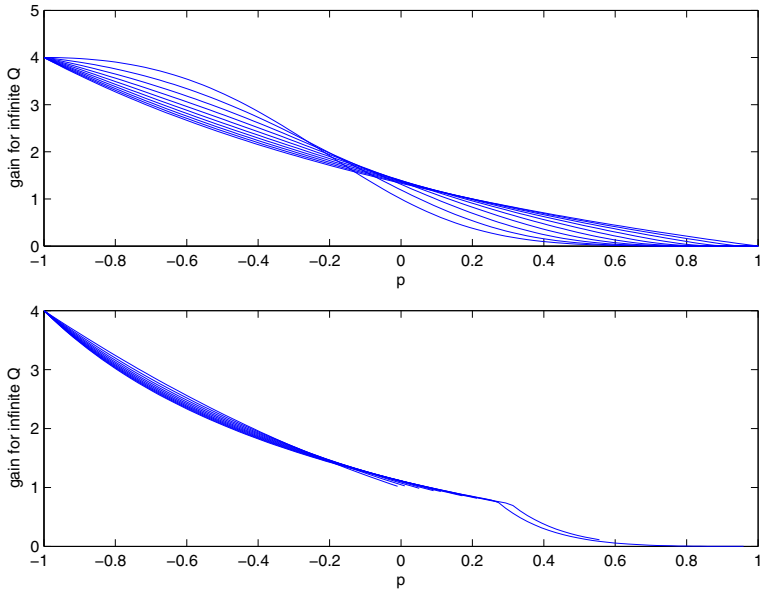


Figure 3.70: Separation-table families (various open-loop zero locations between $z = 0$ and $z = -1$). Top: Two finite zeros and two at infinity. Bottom: One finite zero and three at infinity.

Flattening the separation curve: Different Zero Locations

Here we explore modifying the Compromise Filter by allowing the zeros to be on different locations from each other. We assume that having all the zeros on the same location is somehow suboptimal, and set up an optimization to find a better set of zero locations. We admit, however, that since the compromise filter was chosen with as a subjective tradeoff, such optimization may not choose a design with the same general behavior as the compromise filter.

The optimization we set up was to minimize the weighted sum of squares of the deviation of the curve from its mean value, normalized by the number of points in the curve (this last bit is due to the fact that the curve was only calculated at values of p where the dominant poles are complex, and since p is sampled across the range (-1,1), the number of points in the resulting curve will not be known ahead of time). The weighting is accomplished by spacing the samples of p logarithmically (a la `-1+logspace(0,log10(2),N)` in Matlab), to give approximately equal weighting to each octave (which is also the reason why an L_∞ norm isn't used). In order to remove some obvious non-convexities, the search variables were chosen to be $\mathbf{z} = [z_0, \Delta z_1, \Delta z_2, \Delta z_3]$, such that the open-loop poles are:

$$\begin{aligned} z_0 &= z_0 \\ z_1 &= z_0 + \Delta z_1 \\ z_2 &= z_1 + \Delta z_2 \\ z_3 &= z_2 + \Delta z_3 \end{aligned}$$

And the Δz_i are restricted to be positive, such that the z_i are monotonic:

$$\begin{aligned} \text{minimize} \quad & \|g_{infQ}(\mathbf{z}, p) - \text{mean}(g_{infQ}(\mathbf{z}, p))\|_2 / \text{count}(g_{infqQ}(\mathbf{z}, p)) \\ \text{subject to} \quad & z_i < 1 \\ & z_i > -1 \\ & \Delta z_i > 0 \end{aligned}$$

Now we doubt that the resulting problem is actually convex, so the initial exploration optimization was performed by doing a 1000-point initial monte-carlo search (using valid search points to make full use of all 1000 points), and then using the most optimal point from that search as a starting point for a local search (in this case, Matlab's `fminsearch` function, which performs a form of simplex search). This is not guaranteed to find the global minimum, but it was felt that the result would be sufficient to decide if this design direction was worth following.

The result of the exploratory optimization was

$$\mathbf{z} = [-0.2650, -0.2784, -0.2951, -0.3091]$$

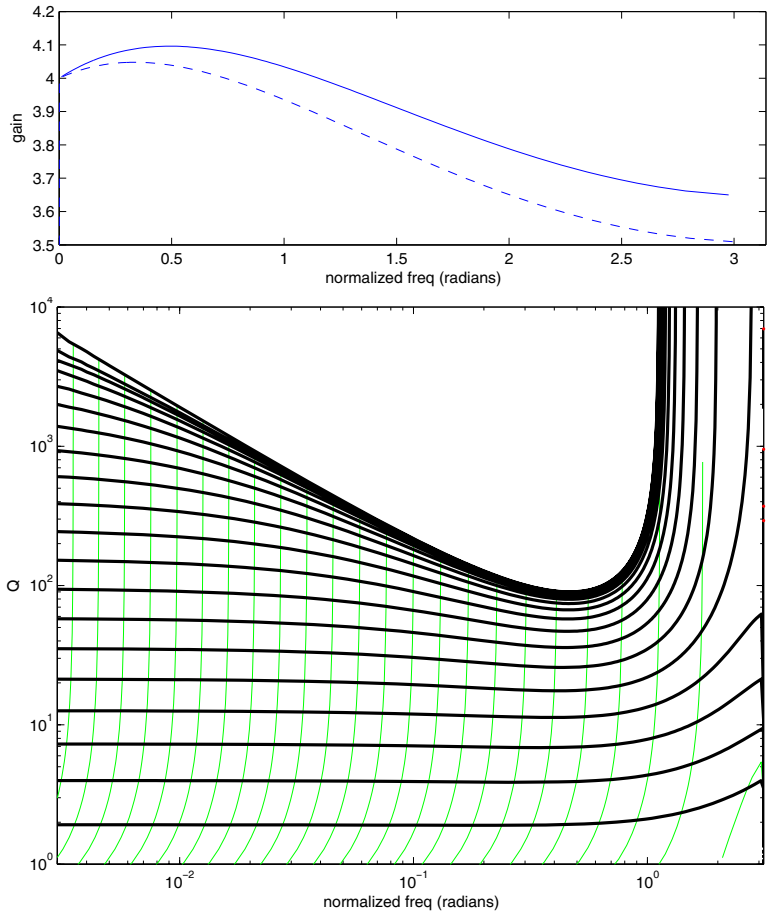


Figure 3.71: Compromise variation with optimized open-loop zero locations [-0.2650, -0.2784, -0.2951, -0.3091]. Top: Separation Curve (Compromise filter separation curve shown in dashed line). Bottom: Q vs. p (compare against Figure 3.60).

This filter is analyzed in Figure 3.71. While the separation curve can be considered to have a slightly smaller variation (and a bit more of a horizontal trend) than the compromise filter, it cannot be said to be vastly better. In fact, comparing the Q-vs-p curves against those of the compromise filter (Figure 3.60), one might still prefer the compromise filter.

As such, we decided not to explore this direction further.

Flattening the separation curve: Tilted Rather Than Flat

Looking at the separation curves, one may notice that some, while not anywhere near horizontal, to appear to be pretty "straight." As such, they could be represented as a first-order polynomial, which should be significantly cheaper than a lookup table. Thus, a viable direction should be to look for filter variations where the separation curve is as close to a straight line segment as possible. For example, the pole/zero mapping transform gives a separation curve which is quite close to a straight line (though the gain does approach zero as $f_c \to f_s/2$).

A few initial experiments were made in this direction, with promising results. However, they were not followed up due to the result of the next section, which effectively halted further work on these previous directions. As such, there is some good work left to do on these lines of inquiry.

Flattening the separation curve, "X1 Filter": Moving the zeros with p

While contemplating the tradeoff that was necessary in the creation of the compromise filter (review Figure 3.55), one begins to wish that the open-loop zero locations could be varied with frequency. For example, note that $z_0 = -1/3$ has quite nice properties at low frequencies, but has the problem of going unstable at higher frequencies, whereas z_0 closer to zero get progressively more stable at high frequencies, at the cost of being "too stable" at low frequencies.

As such, we next explore moving the open-loop zeros as a function of p. Now, we remember that we are trying to keep things inexpensive, so we try the simplest movement function we can think of: a first-order polynomial:

$$z_0 = z_{00}p + z_{01} \tag{3.76}$$

Remember that the intent is to remove the need for a separation table, so if this is successful, the addition of this polynomial would replace one lookup table.

As before, we set up an optimization to search for the best value of the optimization variable $\mathbf{z} = [z_{00}, z_{01}]$. Our first attempt will simply be a minimax optimization:

$$\begin{aligned} \text{minimize} \quad & \|g_{infQ}(\mathbf{z}, p) - \text{mean}(g_{infQ}(\mathbf{z}, p))\|_\infty \\ \text{subject to} \quad & z_0 < 1 \\ & z_0 > -1 \end{aligned}$$

It is important to note that all of the previous Moog-style filter designs have implemented one-pole filters which are normalized to have a DC gain of 0 dB:

$$H_i = \left(\frac{1+p}{1+z_0}\right)\frac{1+z_0 z^{-1}}{1+pz^{-1}} \tag{3.77}$$

Normally, this did not require a divide at runtime, as z_0 was fixed, so $1/(1+z_0)$ could be pre-calculated. However, in this case, z_0 will not be fixed, and this normalization would therefore require at at least one divide per change in p, or one divide per sample if we are allowing p to vary at the sample rate. This would be unacceptable. Therefore, for this design we will change the onepoles to no longer be fully DC normalized:

$$H_i = (1+p)\frac{1+z_0 z^{-1}}{1+pz^{-1}} \tag{3.78}$$

Again, this optimization is not expected to be convex. However, we have what we consider to be a good starting point in the compromise filter ($z_{00} = 0.3, z_{01} = 0.0$).[22] Again, the Matlab simplex search was used, and the result was found to be: $z_{00} = -0.07429, z_{01} = 0.3569$, or $z_0 = -0.07429p + 0.3569$. Thus, the pole moves in a range near the fixed zero of the compromise filter, as might be expected. What is not expected is how flat the resulting separation curve turns out to be (Figure 3.72).

This design is significantly better than the compromise filter. Whereas the compromise filter's separation curve has a range of nearly 0.5, (or about ±0.1 at low frequencies), this design (which the author has been referring to as the "X1 filter"[23]) has a variation of only about 0.001. Even if we take into account the difference in gain range due to the lack of DC normalization of the filters, this is still a factor of somewhere between 25x to 100x smaller variation (depending if we look at the whole range of the compromise filter's curve, or just the low-frequency range).

Let's analyze this filter with the same analyses we have applied to the previous designs (Figures 3.73 through 3.78).

The open-loop filters are, in the notation used to analyze the earlier filters:

$$H_f(z) = \left[(1+p)\frac{(z_{00}p + z_{01}) + z^{-1}}{1+pz^{-1}}\right]^4, \; G(z) = z^{-1} \tag{3.79}$$

And we can describe this as "Four zeros moving together as a first-order polynomial in z, and a delay in the feedback".

[22] And the upcoming result was later replicated using a minimization seeded with a monte-carlo search.
[23] For no particular reason

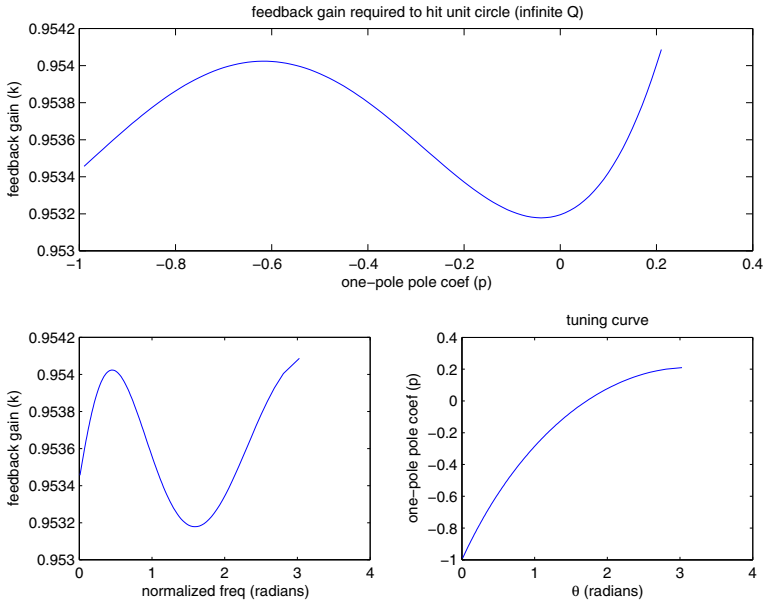


Figure 3.72: "X1" Filter. Open-loop zeros moving as a first-order polynomial with respect to p. Top: Loop gain (k) required to hit unit circle, vs. p. Bottom Left: Loop gain vs. pole angle at the unit circle. Bottom Right: Tuning curve (p vs. θ at the unit circle).

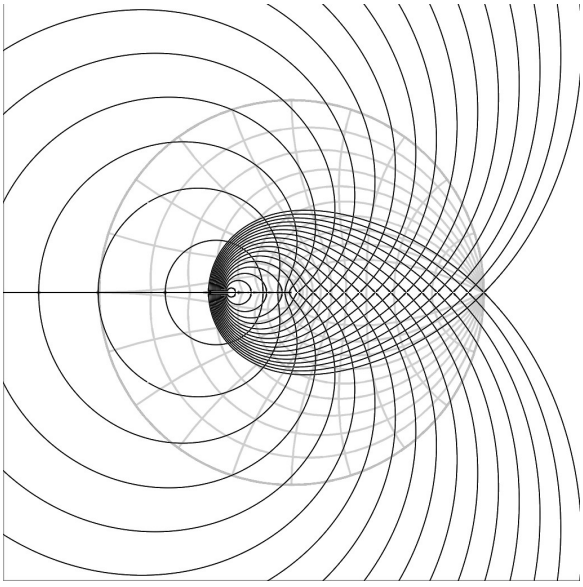


Figure 3.73: X1 Filter: First-order loci in k for various p between $p = -1$ and $p = 0.29$ (max of tuning curve is at $p = 0.21$).

The root-locus equation in k is:

$$z(z+p)^4 + k\left[(p+1)^4(z + z_{00}p + z_{01})^4\right] = 0 \tag{3.80}$$

i.e., there are four open-loop poles on $z = -p$ with one more on $z = 0$, and four open-loop zeros at some point on the real axis near $z = -0.3$. Remember, like the bilinear-with-delay, there is an extra pole. Now the root locus in k is still a basic 1st-order locus, but note that by making the open-loop zeros first-order polynomials in p, we have made the locus 8th-order in p!

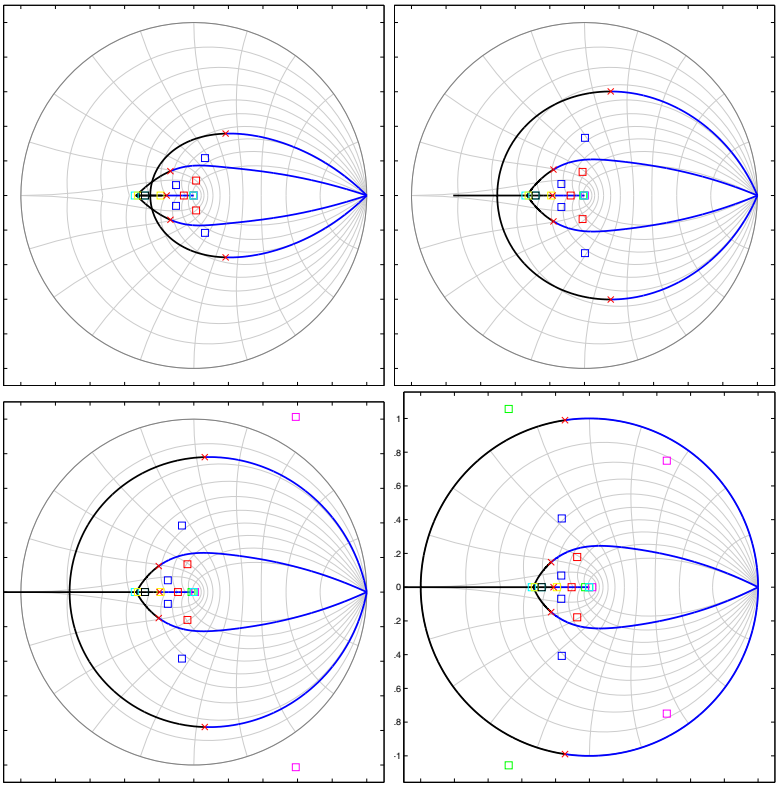


Figure 3.74: X1 Filter: Eighth-order loci in p ($-1 \le p \le 0.33333$) for various k: Top Left: $k = k_{max}/16$, Top Right: $k = k_{max}/4$, Bottom Left: $k = k_{max}/2$, Bottom Right: $k = k_{max}$. Using the DC end of the separation curve for k_{max}. 'x': roots of $D(z)$, 'o': roots of $N_8(z)$, squares: roots of $N_1(z)$ through $N_7(z)$.

Still, with the help of Mathematica, we can note some interesting patterns in the locus polynomials, like the patterns of the roots of the coefficient polynomials after transforming p to $p_0 - 1$:

$$\begin{aligned} D(z): &\quad [0,1,1,1,1] \\ N_1(z): &\quad [0,1,1,1,\infty] \end{aligned}$$

$N_2(z)$: $[0, 1, 1, \infty, \infty]$

$N_3(z)$: $[0, 1, \infty, \infty, \infty]$

$N_4(z)$: [complicated]

$N_5(z)$: $[(z_{00} - z_{01}), (z_{00} - z_{01}), (z_{00} - z_{01}), \infty, \infty]$

$N_6(z)$: $[(z_{00} - z_{01}), (z_{00} - z_{01}), \infty, \infty, \infty]$

$N_7(z)$: $[(z_{00} - z_{01}), \infty, \infty, \infty, \infty]$

$N_8(z)$: $[\infty, \infty, \infty, \infty, \infty]$

Interestingly, we get the same pattern as with the compromise filter in the first five coefficient polynomials. There is no theory yet as to the meaning of the pattern in the later polynomials.

When calculating the Q-vs-p curves (Figure 3.75), we have a choice of values for the maximum k value (i.e.,the value that we will treat as "infinite Q"). Remember that the idea is to have made a filter whose separation curve is as close to a constant as possible. The question is, since there is still some variation, what value do we use? Three obvious candidates are: the minimum of the curve, the mean of the curve, and the max of the curve. However, since the curve does not hit an extreme (or the min) at $p = -1$ (i.e., at $f_c \rightarrow$ DC), another option is to use the $p = -1$ value. Figure 3.75, shows the plots for each case. In the author's opinion, using the DC value of the separation curve gives the most even Q distribution when using the logarithmic approximation for $k(Q)$, as opposed to using measured $k(Q)$ curves. Note that in all of these cases, one can get pretty flat Q curves up past $Q = 1000$ without needing to use a separation table (the limit for the Compromise filter is around 50 to 100).

Speaking of measured Q curves, let's look at the logarithmic approximation of k for a given Q: In Figure 3.76, we see two approximations: the upper approximation (the straight line) is:

$$k = k_{max}\left(1 - \frac{2}{Q}\right) \tag{3.81}$$

The lower approximation (the closer one) is:

$$k = k_{max}\left(1 - \frac{2}{Q + 1.5}\right) \tag{3.82}$$

Note that this approximation requires an accurate value of k_{max}. As can be seen in Figure 3.75, which uses logarithmically-spaced k values, the constant-k traces end up getting compressed together in actual Q at high values of intended Q if k_{max} corresponds to roots inside the unit circle (as in much of the lower-right graph), or they end up spreading out in actual Q if k_{max} corresponds to roots outside the unit circle (as in the upper-left graph). The Q values in Figure 3.76 were measured near DC using the DC value of the separation curve for k_{max} (i.e., the lower-left graph of Figure 3.75), so it was accurate for that measurement.

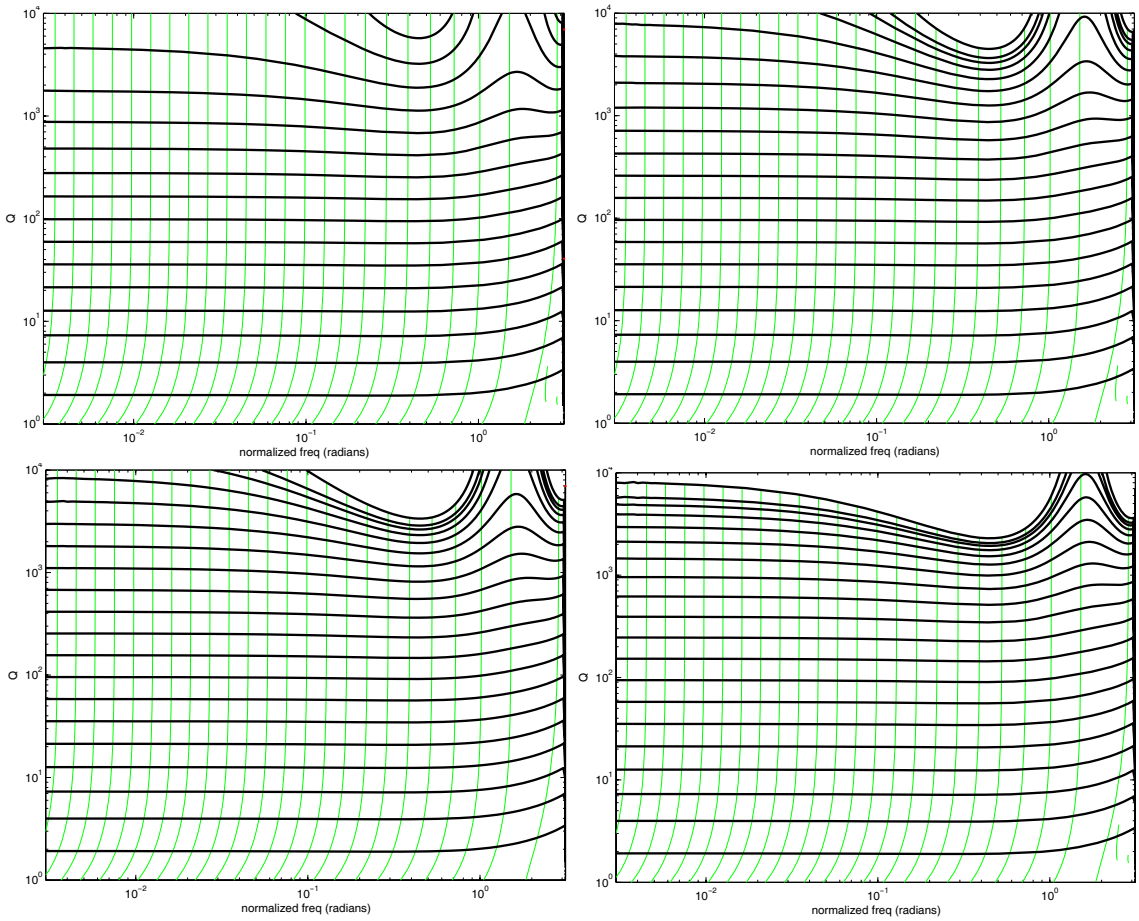


Figure 3.75: X1 Filter (raw, no Lookup tables). Q vs. p, wide range. Top Left: k_{max} set to max value of separation curve (0.9541). Top Right: using mean of separation curve (0.9536). Lower Left: using $p = -1$ value (0.95346). Lower Right: using min of separation curve (0.9532).

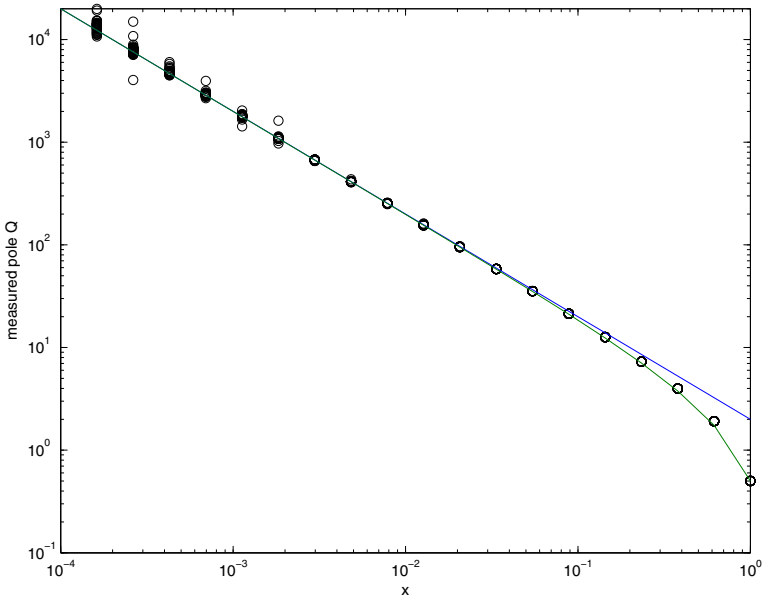


Figure 3.76: X1 Filter, $k(Q)$ approximation. Dots: Measured Q vs. x (where $k = k_{max}(1 - x)$). Straight Line: $2/x$, Lower Line: $2/x - 1.5$. This plot obtained using $k_{max} = 0.95346$, the DC value of the separation curve. Q measured at frequencies near DC.

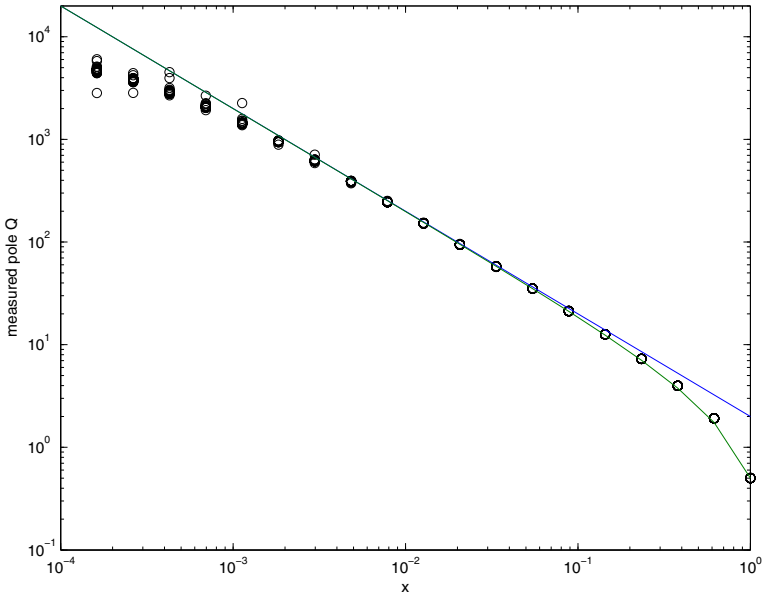


Figure 3.77: X1 Filter, $k(Q)$ approximation with inaccurate k_{max}. This plot obtained using $k_{max} = 0.9532$, the minimum value of the separation curve.

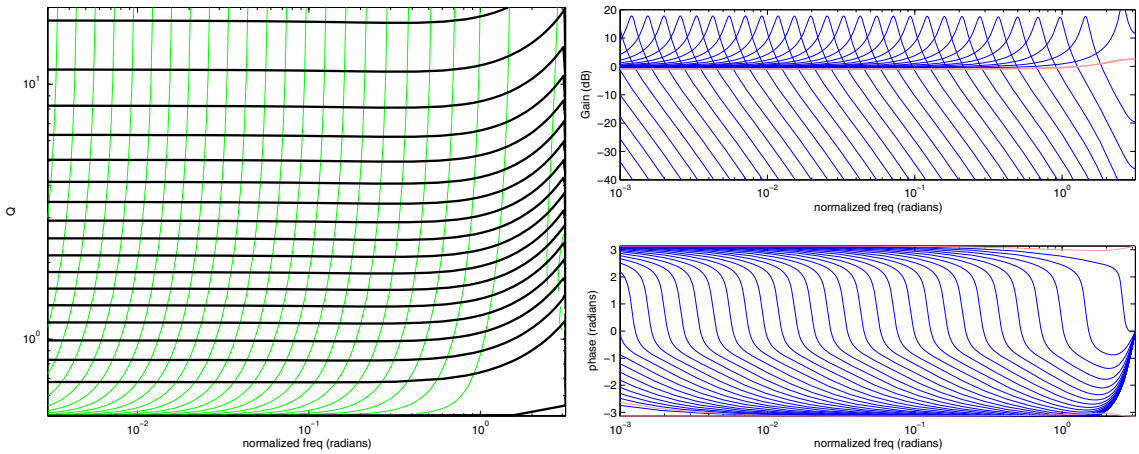


Figure 3.78: X1 Filter: Raw filter (no Lookup tables). Left: Q vs. p, narrow Q range. Right: representative frequency responses for a p sweep ($k = 0.75\ k_{max}$, using mean value of separation curve for k_{max}).

Note that a side-effect of removing the DC normalization from the feedforward one-pole filters is that the I/O gain of the filter is increased. If one compares the example frequency responses of this filter (Figure 3.78) versus those of the compromise filter and the basic discretizations, one will see that this filter has a gain approximately a factor of 4 higher. This can also be seen by the fact that the values of the separation table dropped from near 4 (at $f_c \rightarrow$ DC) to just below 1.0, so that the loop gain for unity feedback dropped by about 4x.

Finally, we do not present analyses of this filter using separation tables, because the filter is considered good enough that the use of separation tables is not expected to be necessary. Furthermore, one can be pretty sure from the analyses of the other filters when using separation tables (especially the Compromise filter, Figure 3.61) that the use of a table will have predictably good results, and therefore showing such an analysis would be redundant.

Pseudocode for a C implementation of this filter is given in Figure 3.80. Note that this code is not guaranteed to be the most optimal for any particular processor or compiler, and users are encouraged to further optimize as far as possible. This code also implements Direct-Form-2 filter forms, which are not expected to work as well in fixed-point number systems. Any algorithm should be modified as appropriate for the specific architecture that it will be implemented on.

Flattening the separation curve: Future Directions

The "X1" filter is pretty good, so it becomes difficult to justify further exploration, especially in (or outside) the realm of this thesis. However, quite a few possible directions are left open for exploration, which may produce further useful filter design methods:

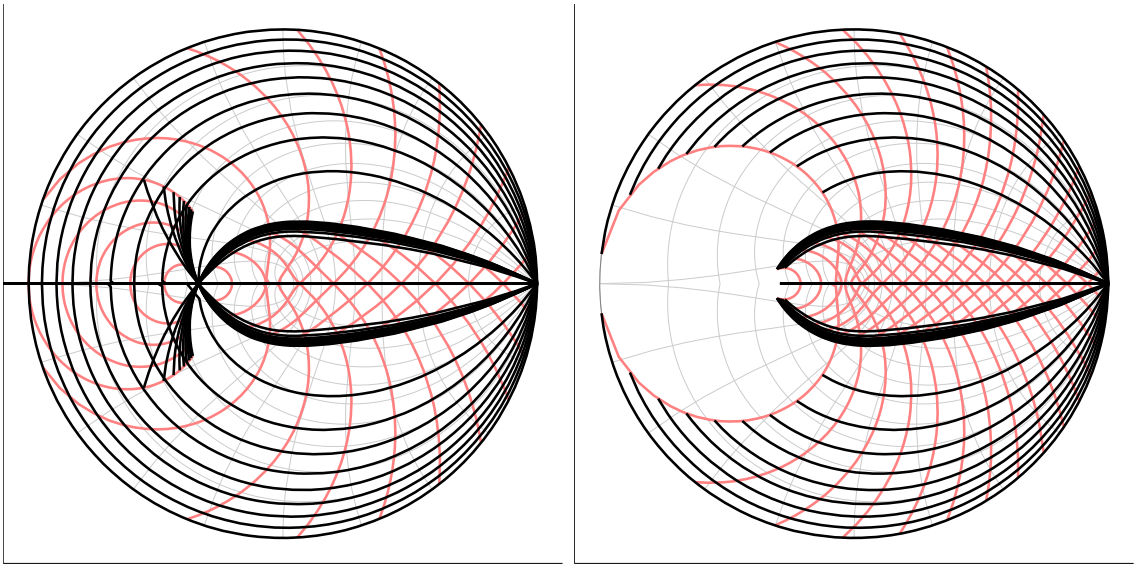


Figure 3.79: X1 Filter (raw, no Lookup tables). Pole traces. Dark: constant k, Light: constant p. Left: $-1 < p < 1$, Right: $-1 < p < 0.21$.

- Several directions involving two- or three-stage feedforward systems were ignored out of hand, due to concentration on four-stage systems. In particular, good work can be done following up Wise's work on allpass-based loops, and there is some evidence that three-stage methods may produce viable four-pole filters.
- Looking at all the combinations of (-1,0,1) zero locations across the stages.
- Taking the "X1" filter further, and moving the different zeros independently.
- Varying the $f_c \rightarrow$ pole mapping for each stage. Looking deeply into the effects of having the open-loop poles not co-located. Hutchins has discussed that in some experiments this causes the required feedback gain to grow much larger. It would be useful to have a good intuitive understanding of why this may be so.
- Variants on the onepole gains, possibly varying with some of the controls. All of the Moog-style designs except the X1 filter used onepole filters with a DC gain of 1.0, and the X1 filter only moved away from that for efficiency reasons. There may be useful directions in exploring other gain behaviors.
- Picking off outputs in various combinations for various filter types, as done in the state-variable filter, and as discussed by Hutchins and more recently by Huovilainen for the Moog-style filters.

```
s = state->filterStateArray;
A = state->feedbackState;

for (i=0; i<vectorLength; i++) {
    pval = 0.6*tuningCoef[i] - 0.4;   // Bring (-1.0,1.0) into (-1.0,0.2) range
    z0 = 0.3569 - 0.0742*p;     // choose the zero location from the pole location

    A = SATURATE(in[i] - A);   // feedback

    // WARNING: DF2 filters! (not recommended for fixed-point implementations)
    for (j=0; j<4; j++) {
        tmp = s[j];
        A =  SATURATE((1.0+pval)*A - pval*tmp);
        s[j] = A;
        A = SATURATE(z0*tmp + A);
    }
    out[i] = A;
    A *= gainCoef[i];
}
state->feedbackState = A;

// Deal with denormals on floating-point processors which need it:
for (j=0; j<4; j++)
    s[j] = CLAMP_DENORMAL_TO_ZERO(s[j]);
```

Figure 3.80: C pseudocode for an implementation of the X1 Filter. This implementation assumes that any desired $f_c \rightarrow p$ and $Q \rightarrow k$ mapping has been done externally to this loop. `SATURATE()` implements whatever saturation is desired (none, hard clipping, polynomial, tanh, LUT, etc.). This implementation can be considered to have "too many" saturations, and cheaper implementations should limit the number of expensive saturations in the loop. Note that this code demonstrates DF2 implementations for the onepole filters. One may prefer DF1 or TDF2 implementations in that case, as well as possibly modifying how gain is distributed throughout the loop. This code is not guaranteed to be the most optimal for any particular processor or compiler. Designers are encouraged to further optimize as far as possible.

3.3.4 Conclusions on the Moog-style Filter Section

For most work, we have seen that basic discretization can work fine when using two or three one-dimensional lookup tables for mapping and control separation. For situations where fewer tables are desired, two filter methods: the "Compromise," whereby the open-loop zeros are placed at 0.3, and the "X1", where those zeros are moved as a first-order polynomial in the pole location, are described. The Compromise technique is useful for low-Q situations (below 20 to 100), and the X1 technique is useful for Q up past 1000.

Future directions

Note that this research concentrated solely on the use of *four* onepole sections. There are likely some good alternative designs to be found using two or three onepole sections rather than four (Duane Wise's allpass-based concept in [293], whereby two first-order allpass stages are used rather than first-order lowpass filters, can be considered to be in these classes).

In particular, certain discretizations required an extra delay in the feedback and actually ended up with a higher-order filter. As such, using those discretizations with three one-pole stages rather than four would result in a true fourth-order filter. There is a good chance that such three-stage filters may make for particularly nice designs. For a quick preview exploration of the area, let us look at a family of k loci for a three-stage bilinear-with-delay filter (Figure 3.81) the design is the same as before, but with one onepole stage removed. If we follow the same design methodology

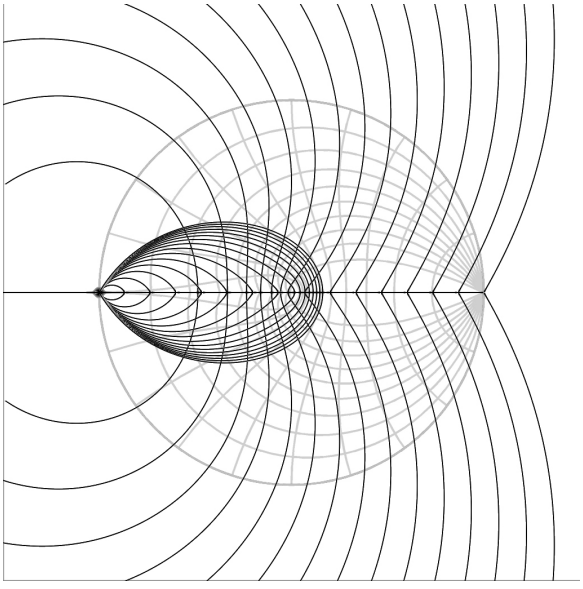


Figure 3.81: Three-stage moog-style filter, bilinear Transform with delay: First-order loci in k for various p between $p = -1$ and $p = 1$.

as the X1 filter, we get the following polynomial for open-loop zero locations:

$$z_0 = 0.154993 - 0.0632325p \tag{3.83}$$

The separation and tuning tables for this design are shown in Figure 3.82. Note that the separation curve has a bit more variation than the 4-stage case. It is as yet unclear why that may be. Q-vs.-freq analyses and an example set of frequency responses are shown in Figure 3.83.

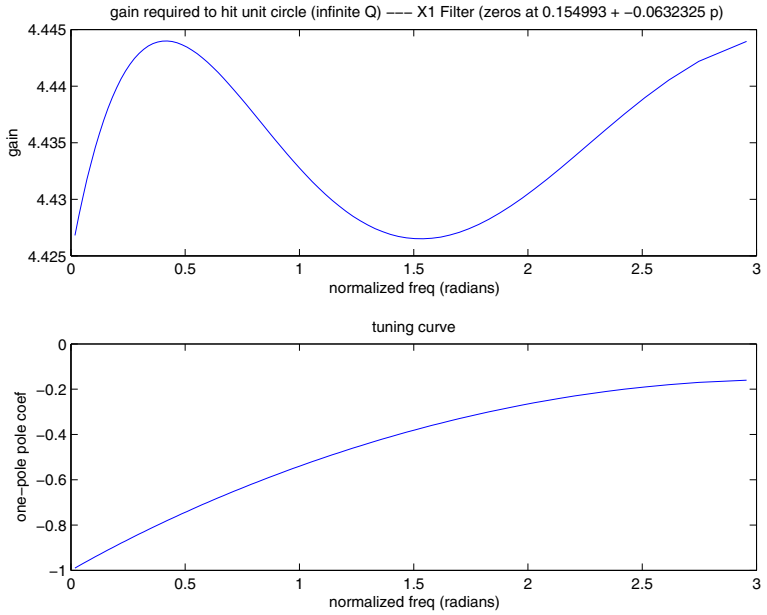


Figure 3.82: Three-stage filter following the "X1" design method. Open-loop zeros moving as a first-order polynomial with respect to p. Top: Loop gain (k) required to hit unit circle, vs. p. Bottom Left: Loop gain vs. pole angle at the unit circle. Bottom Right: Tuning curve (p vs. θ at the unit circle).

Another possible design direction which was not be followed in this discussion is the cascading of 2nd-order filters. for example: design only 2nd-order set of resonant poles (maybe even a state-variable filter), and add in the non-dominant poles in cascade, possibly as a static filter (since the motion of the non-dominant poles has lees effect). This direction was not taken because of the desire to stay with the "one big loop" concept of the filter topology. Thus other similar directions like cascading two state-variable filters were also not followed. One may argue that if one implements nonlinearity, that such cascaded designs would not display the same type of nonlinear behaviors as a single-loop implementation.[24]

[24] In other words, they would be "less physical."

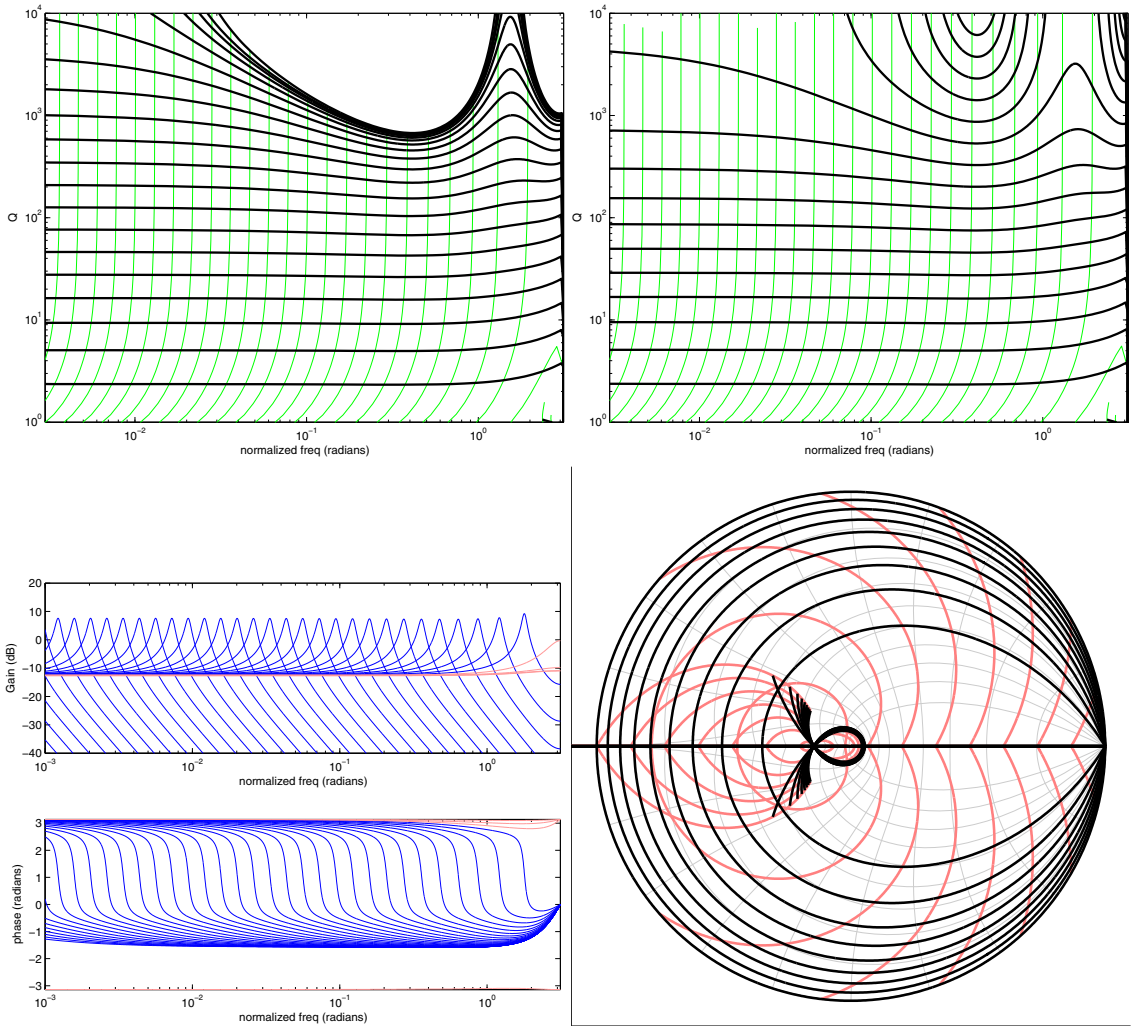


Figure 3.83: Three-stage filter following the "X1" design method: Raw filter (no Lookup tables). Top Left: Q vs. p, wide Q range, nomalized to DC value of separation curve. Top Right: normalized to maximum value of separation curve. Bottom Left: representative frequency responses for a p sweep ($k = 0.8k_{max}$). Bottom Right: Pole traces (Dark: constant k, Light: constant p).

A closing thought

It is noted that the state-variable filter seemed to discretize much more easily than the Moog-style filter. In particular, the infinite-Q contour was not a problem in the state-variable-filter's discretization. Here are a few theories as to why that may have been:

- The infinite-Q contour for the continuous-time state-variable filter occurs where the qq coefficient equals zero, whereas for the Moog-style filter, it occurs on a non-zero value of the k coefficient. It may be possible that the behaviors of a zero coefficient discretize better than those of a nonzero coefficient. Note that this may be tied into the fact that these filters were discretized by keeping the same topology in discrete time. As such, the a zero coefficient in a feedback loop will have the same effect in continuous and discrete time (to block a signal), whereas a nonzero coefficient's action may not transfer as directly.

- A more likely interpretation is thus: The state-variable filter works by taking an undamped system and damping it by applying feedback. As such, all that is necessary to recreate the behavior in discrete-time is to create an undamped system which is dampable. The infinite-Q contour is thus inherent to the filter (without touching anything). Doing anything further damps the system (or destabilizes it), but the default of doing nothing gives exactly the desired contour. In the Moog-style filter, on the other hand, we take an extremely damped system (a bunch of real poles), and by applying feedback, undamp it (essentially destabilizing it). Thus, the default behavior is the highly damped state. The infinite-Q contour must be reached by giving a very particular feedback gain, which (as we have seen) can be easily affected by many different side-effects of the discretization. Interestingly, this interpretation is further validated by noting the difficulty the state-variable has in keeping consistent behavior at very low Q, which in this way of looking at it, is at the other end of the Q control from the default state, and thus most likely to be affected by discretization side-effects. This may shed light on possible further directions for Moog-like four-pole exploration: can an inherently nondamped four-pole system be created with two poles on the unit circle and two others highly damped, which (1) can be damped by feedback (without adversely undamping the inner poles), (2) still has the loop structure and (3) saturates similarly to the original loop. The saturation question is an interesting one. In such a structure, the outer feedback would be highly attenuated in a high-Q situation, whereas in the current Moog-style topology, the primary feedback has its highest gain near approaching infinite Q. That difference alone may be enough to make this direction have strongly different saturation behavior.

Chapter 4

Bandlimited Waveform Synthesis

4.1 Introduction

Virtual Analog deals primarily with subtractive synthesis. As such, it requires harmonically-rich waveforms from its oscillators. The classic analog waveforms (sawtooth, rectangle (square), and triangle) fit this requirement by containing discontinuities either in the waveform or in a low derivative. As such, they were effectively infinite bandwidth (within the limits of the analog circuitry).

Any such signal must be *bandlimited* to less than half the sampling rate before sampling to obtain a corresponding discrete-time signal, or else images of the higher harmonics will end up in the discrete-time signal (also known as "aliasing artifacts"). Similarly, a discrete-time signal which is to be generated, if it is to model such a waveform, must be generated with the effect of such bandlimiting, or else the generated wave will contain aliasing artifacts. Trivial methods of generating such waveforms digitally contain aliasing due to having to round off the discontinuity time to the nearest available sampling instant.

Aliasing produces distortions which are typically objectionable. Such artifacts are "grunginess" (low-frequency changes in the signal character as the signal passes through various temporary integer-period states, and slight clicks are other noise on the transitions), possible beating between aliased partials which get too close to non-aliased partials, and spectral components which audibly move in the opposite direction of the primary signal's frequency motion. This last behavior is particularly noticeable in virtual analog, given the tendency to smoothly modulate frequency. Further, among many users, the more they hear aliasing, the more attuned they become to it and the more objectionable it becomes to them over time.

In this chapter, we will first review the generation of the waveforms in the analog domain, and review our design philosophies, then we will review why digitally-generated waveforms tend to alias, followed by an overview of existing methods for generating waveforms without aliasing

(or without significant aliasing). Next, we will present the concept of deriving the popular wave types from impulse trains via purely linear operations, such that if a bandlimited version of the impulse train can be generated, these other waves can be derived without introducing further harmonics, and hence staying bandlimited. Finally, we will discuss the generation of bandlimited impulse trains ("BLITs"), in the light of the previous exploration of existing bandlimited waveform synthesis methods, and look at recent developments in the field.

4.2 Review

4.2.1 Analog Synthesizer Waveforms

The classic analog synthesizer waveforms were as follows ([37][119][107][108]:

- Sawtooth. This was typically implemented as some sort of integrator of controllable slope with some method to reset it when a threshold value is reached. As such, the slope controlled the frequency.
- Rectangle. (Aka "Pulse" or "Square", though "Square" implies a 50% duty-cycle to many.). This was typically implemented by offsetting a sawtooth and running through a comparator. As such, the resulting wave's *duty cycle* (the amount of time the wave is in the high state as a percentage of the period) could be controlled via the offset amount. It is also noted that one can also derive a rectangle wave by subtracting delayed sawtooth waves of the same frequency. However, this technique was not used much in the analog days, as delay was a difficult operation in analog, so would end up requiring the implementation of two sawtooth generators instead. The comparator method was in general a much easier and smaller circuit. As the system ran in continuous time, the fact that this was a strongly nonlinear operation was of little consequence (whereas in discrete time, this is a very important fact).
- Triangle. This one was implemented by a variety of methods, including integrating a square wave, implementing an integrator which flipped its slope when it reached the limits, or full-wave rectifying a sawtooth (which was tricky, though, due to possible glitches on the sawtooth reset). Note that some called this wave a "sawtooth". In fact, some implementations allowed the up and down slopes to be varied, in which case a limiting case would be the sawtooth as discussed above.

Note that sinusoidal waves were not necessarily implemented in analog synthesizers, at least as sound sources, as the subtractive-synthesis philosophy required harmonically-rich signals, and the sine wave was the opposite. However, many users would sometimes put a filter into self-oscillation (if possible) in order to get a sinusoid. Note that many LFOs ("Low-Frequency Oscillators", which run a (usually) sub-audible frequencies to be used as control signals) also did not have sinusoidal

outputs (again, the above shapes were the most common), though some synths did implemented them as a wave-shaping of triangle wave to get an LFO with a smoother reversal than the triangle [119] [37].

Note that the various waves could be derived from the sawtooth, a concept which will come up again. However, we should note that the derivation in the analog world was usually via extremely non-linear operations, such that these exact methods are not applicable for all discrete-time work (though they can work just fine for low-frequency work, where aliasing is often not an issue).

4.2.2 Review of Design Philosophy

As discussed in Chapter 1 and Chapter 3, the general design philosophy for this work is that of inexpensive modulatability: assume that the frequency and amplitude (and possibly other parameters such as pulse-width/duty-cycle, sync, etc.) may change to any arbitrary value from sample to sample. Thus, not only should the basic waveform-generating algorithm be inexpensive, so should the tuning mappings, if any.

As we will see, the primary tradeoffs for achieving this will be the amount of aliasing and possibly modulation quality. Some of the proposed algorithms will end up trading off some efficiency in parts of the operation range for reduced aliasing. We will see other algorithms which take the other extreme, implementing extremely efficient calculations at the expense of increased aliasing.

Finally, as before, we take as a basic assumption that oversampling is too expensive. As mentioned earlier, many of the issues discussed in this chapter become almost trivially easy when oversampling (i.e. trivially-calculated waves are sufficient). As such, in a rather large-grained tradeoff, we choose to ignore oversampling in favor of calculation at "low" sample rates (typically somewhere between 22 kHz and 48 kHz), and in favor of having something interesting to work on. However, if a designer can get away with oversampling (or if some other part of the system contains nonlinearities which must be oversampled), we will not discourage them.

4.2.3 Why Trivially-Calculated Discrete-Time Pulse Trains Alias

The "obvious" way to generate a discrete-time version of an impulse train is to approximate it by a unit-sample-pulse train. The unit sample pulse $\delta(n)$ is defined as

$$\delta(n) \triangleq \begin{cases} 1, & n = 0 \\ 0, & |n| = 1, 2, 3, \ldots \end{cases} \tag{4.1}$$

However, the unit-sample pulse only exists on integer sample locations (i.e., we cannot put an impulse on a fraction of a sample, though we will later attempt to approximate doing so. Everything in discrete time must happen on integer sample locations), so we have a problem when a

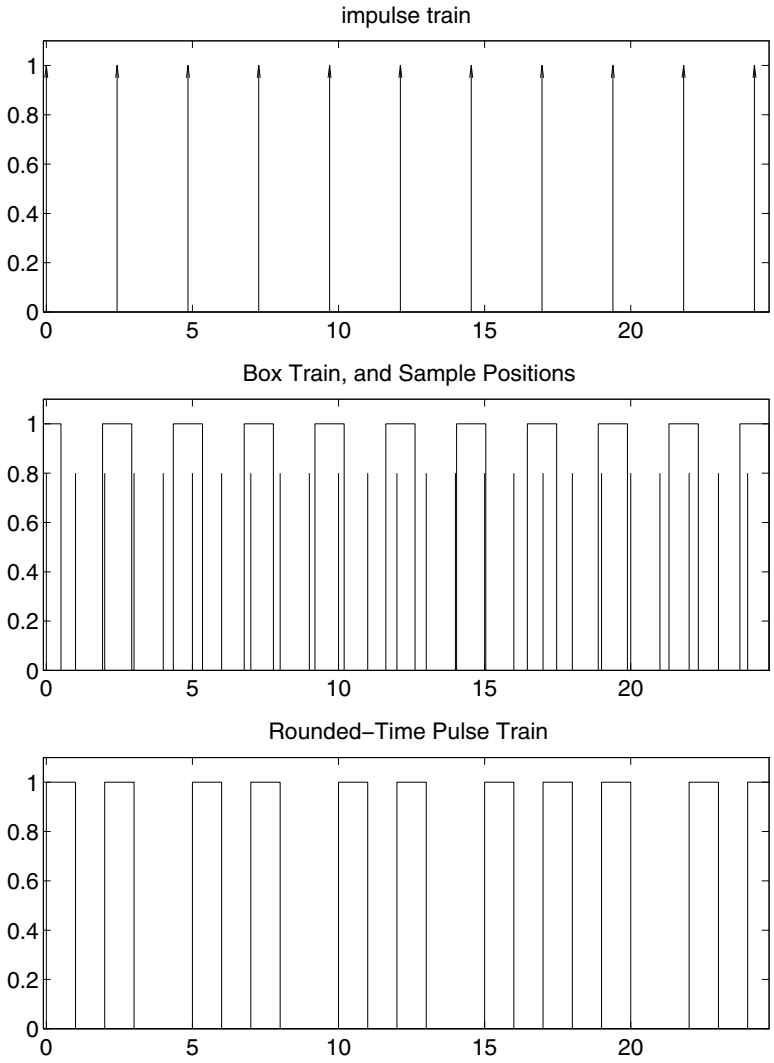


Figure 4.1: Rounded-Time Impulse Train as a Sampled Version of an Ideal Rectangular Pulse Train

periodic wave's period $1/f_1$ is not an integer, which is nearly always the case (it is only an integer if f_s/f_1 is an integer, where f_s is the sampling rate).

It does not work to round the period M to the nearest integer, except at low frequencies, since for short periods, the set of achievable frequencies f_s/M would be extremely sparse. Therefore, to make the pitch right, it is necessary to compute the impulse arrival times accurately and round each arrival time to the nearest sample instant, such that their periods average out to the correct value. This process can be thought of as phase jitter which adds noise to the signal.[1] It can also be modeled as a uniform sampling of a sequence of rectangular pulses one sample wide centered on the ideal periodic impulse-train locations (Figure 4.1) (i.e., the impulse train convolved with a one-sample-wide box function, then sampled on the sample periods).

$$box(t) \stackrel{\Delta}{=} \begin{cases} 1, & |t| \le T/2 \\ 0, & |t| > T/2 \end{cases} \tag{4.2}$$

The Fourier transform of the box is a *sinc function*

$$Box(\omega) \stackrel{\Delta}{=} \int_{-\infty}^{\infty} box(t) e^{-j\omega t} dt = T \,\mathrm{sinc}(fT) \tag{4.3}$$

where $\omega = 2\pi f$, and

$$\mathrm{sinc}(x) \stackrel{\Delta}{=} \frac{\sin(\pi x)}{\pi x}. \tag{4.4}$$

A periodic sequence of these box functions is constructed as

$$x(t) \stackrel{\Delta}{=} \sum_{l=-\infty}^{\infty} box(t + lT_1) \tag{4.5}$$

equivalent to convolving an impulse train of period of T_1 with the box function. Its Fourier transform is:

$$\begin{aligned} X(\omega) &\stackrel{\Delta}{=} \sum_{l=-\infty}^{\infty} e^{j\omega l T_1} Box(\omega) \\ &\propto Box(\omega) \sum_{l=-\infty}^{\infty} \delta(\omega - l\omega_1) \\ &\propto \begin{cases} Box(l\omega_1), & \omega = l\omega_1 \\ 0, & \omega \neq l\omega_1, \forall l \end{cases} \end{aligned}$$

where $\delta(\omega)$ denotes the discrete-time delta function, and $\omega_1 = 2\pi f_1$. Thus, the rectangular-pulse train has an infinite harmonic spectrum weighted by a sinc function having zeros at multiples of the sampling rate f_s (because the original rectangular pulse was taken to be one sampling interval

[1] Since the jitter is within one sample, the smaller the period, the larger percentage of a period is the jitter. This is one explanation for why aliasing is less audible at low frequencies: the jitter is a nearly insignificant percentage of the period.

wide). Since the sinc function has a lot of energy above its first zero crossing, sampling $x(t)$ will cause aliasing of an infinite number of harmonics whose amplitudes have a lot of energy (they fall off at 6 dB per octave (since $\mathrm{sinc}(fT)$ falls off as $1/f$)). The harmonics under the outer half of the main lobe of the sinc function will also alias. Sampling, we get

$$\begin{aligned} y(n) &\stackrel{\Delta}{=} x(nT) \quad \leftrightarrow \\ Y(e^{j\omega T}) &\propto \sum_{k=-\infty}^{\infty} X(\omega + k2\pi f_s) \\ &= \sum_{k=-\infty}^{\infty} Box(\omega + k2\pi f_s) \\ &\quad \times \sum_{l=-\infty}^{\infty} \delta(\omega + k2\pi f_s - l\omega_1) \end{aligned}$$

The desired pulse-train spectrum is only the $k = 0$ term above. Each nonzero k term contributes a string of aliased of harmonics across the entire frequency band. The amount of aliasing is highly significant and audible, as can be predicted from looking at an example spectrum (Figure 4.2, top plot).

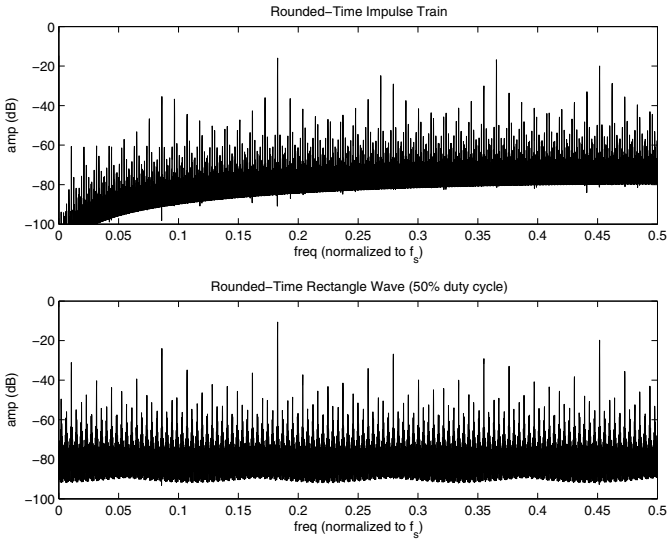


Figure 4.2: Top: Spectrum of a rounded-time impulse train with a frequency of $0.1828 f_s$. Bottom: Spectrum of a rounded-time square wave of the same frequency.

The only frequency which does not see any aliasing is DC, since the sinc spectral envelope goes through zero at all multiples of the sampling rate (which are the frequencies which would alias onto DC). For this reason, the aliasing also is reduced at very low frequencies relative to the sampling rate. This provides another explanation why the aliasing is not as objectionable for

low fundamental frequencies (where the nearest-integer roundoff is an insignificant fraction of the period): The aliased frequencies that would tend to be most noticeable — those near or below the fundamental frequency — are most attenuated.

The above analysis extends to rectangle waves: the trivially generated rectangle wave has its transitions rounded to the nearest sample. Since a rectangle wave can be viewed as the integration of a bipolar impulse train (one with pulses of alternating sign, see Section 4.3.2), we get a spectrum which has similar aliasing (Figure 4.2, lower plot). Because of the integration, there is a different tilt to the spectrum, but the aliasing is still very strong.

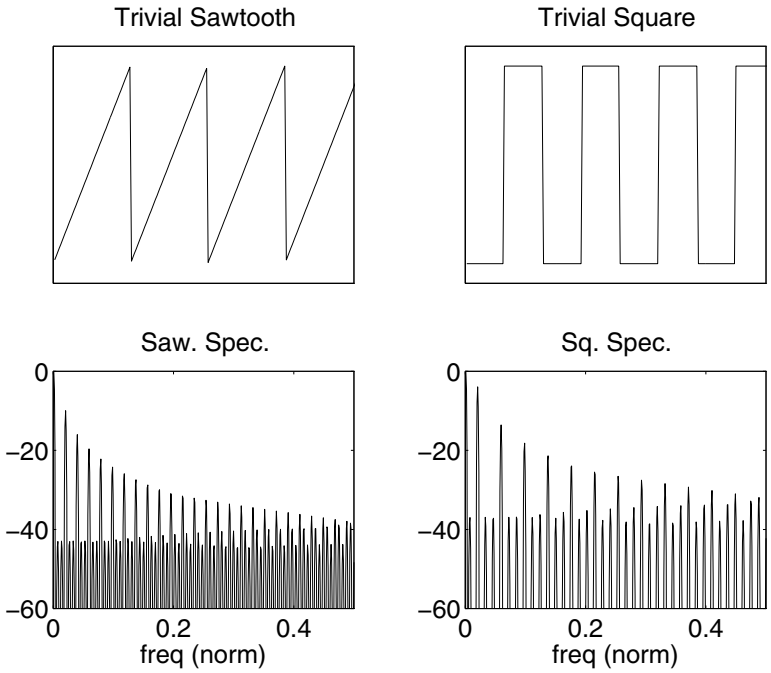


Figure 4.3: Spectra of Trivial Sawtooth and Square-Wave Signals, Low Frequency, note how the aliased harmonics wrap back and forth in the spectrum between the "correct" harmonics

A similar argument applies to explain why sawtooth signals alias: If the unit-amplitude sawtooth is generated by

$$s(n) = (f_1 nT) \bmod 1 \tag{4.6}$$

where f_1 is the desired frequency, then it can be shown to be simply a sampling of a continuous-time ("analog") sawtooth of equivalent slope. Since the analog sawtooth is not bandlimited, the sampled version will be aliased.

This is slightly different from a rounded-time retriggered sawtooth (i.e. a phase ramp which is reset to zero whenever it is determined that a new period is starting). It differs from the previous

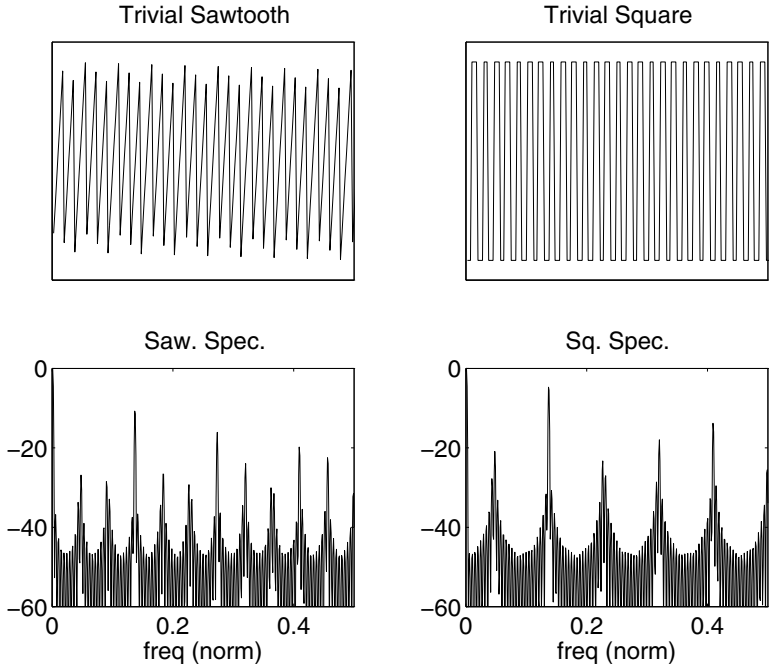


Figure 4.4: Spectra of Trivial Sawtooth and Square-Wave Signals, High Frequency. The aliased harmonics are quite prominent, and would be very audible.

sawtooth in that the reset value is a constant (i.e., zero), whereas the previous algorithm "resets" to a value that is equal to the amount by which the wave had exceeded the threshold). Such a wave can also thus be described as the convolution of a discrete-time copy of a single cycle of a sawtooth wave with a rounded-time impulse train, which was just shown to be aliased as well.[2] Since a single cycle of a sawtooth has a Fourier transform which falls off at 6 dB per octave (all discontinuous waveforms have spectral fall-offs no faster than 6 dB per octave [200]), it follows that the spectrum of the rounded-time retriggered sawtooth also falls off at 6 dB per octave. Therefore, sampling it causes aliasing in an amount similar to that in the sampled rectangular pulse train.

Note that this description of rounded-time retriggered sawtooth (i.e. convolving a rounded-time impulse train with a shape) also describes a few vocal synthesis algorithms, particularly *formant synthesis* methods, such as VOSIM [133], Chant [225] (see also [221]). These also relate to the "PET" (periodic excitation table) method of commuted synthesis of bowed-string instruments [129], whereby a violin-body impulse response is periodically played into a string, controlled by a rounded-time impulse train (when used in a non-interpolated setting). It is also related to *window-function synthesis* [92] (summarized in [221]), where a rounded-time impulse train is multiplied by

[2]With the variable addition or subtraction of the last sample of this single-cycle shape, depending on the actual rounded number of samples in a particular period.

an amplitude pattern, and then convolved by a Blackman window. In all these cases, a possibly non-integer period is convolved with the impulse response of some formant filter (or lowpass filter in the case of window-function synthesis). These perhaps overlap if the period is short. The impulse-response start times are quantized to the nearest sample, thus causing pitch-period jitter. As will be seen later on, this jitter can be reduced or removed by estimating the subsample-location of each impulse, and translate that into interpolated resampling of the impulse responses (a different subsample resampling for each pulse). When most of these methods were derived, such a resampling was considered very expensive, so it appears not to have been done in practice. We will see later, though that the interpolated version of this idea is the basis for the BLIT-SWS method.

4.2.4 Bandlimited Synthesis Review

There has been quite a bit of work done on the generation of waveforms with little or no aliasing, as from the beginning of digital music research, aliasing has been a constant problem. As such, most overview books give at least some discussion to the topic ([221], [65]) Vesa Välimäki has recently published a good review of the field as well [276], and as such mentions much in common with this review (though with a slightly different but technically similar taxonomy).

In this section, we will look primarily into methods for generating bandlimited versions of general waveforms. In a later section, we will look at some more specific methods for generating bandlimited impulse trains. Algorithms specifically for impulse trains will not be discussed in this section.

4.2.5 Bandlimited Synthesis Review: Steady-State Algorithms

In this section, various preexisting methods for synthesis of general bandlimited waveforms will be reviewed. In particular, this section looks at methods which calculate waves using a "steady-state" assumption.

For some background, imagine a truly bandlimited impulse train, which can be viewed as a train of sinc functions centered on each impulse location. Since ideal sinc functions have infinite extent in time, the value of the resulting signal is an infinite sum of such sincs: the value at any particular time is affected by the center locations and scales of all of these sincs, both in the past and in the future. This thought experiment can be extended to other waveforms as well: the bandlimited version of a cycle of a periodic waveform is its convolution with a sinc function, and hence each point in the final sum is theoretically affected by all past and future periods.

Steady-state waveform synthesis algorithms are based on the assumption of true periodicity, as that allows the past and future periods to be predicted. The algorithms generate the values of a wave as though it were exactly periodic for all time (usually by implementing some closed-form solution or otherwise recording the solution). As such, they are exact if the waveform does not

ever change. Any change in their parameters with time breaks this assumption, but it is usually assumed that the effect of such a break wold be inaudible (or hidden by the fact that it may only occur rarely). However, if changes happen repeatedly, as when the system is being modulated (FM, AM, sync), the effect may become noticeable. In Section 4.4.4, we will look into this issue.

As a restatement, steady-state algorithms generate, for any point in time, samples of the bandlimited version of a waveform as though it were exactly periodic. They can achieve bandlimiting by being able to predict the history and future of the wave and thus predict at design time and implement the required math to generate a signal that will be bandlimited.

Additive Synthesis using Discrete Oscillators

Additive synthesis is trivially bandlimited simply by not generating harmonics higher than $F_s/2$; typically this also presents a computational savings, in that those harmonics need not be calculated.

The central idea of additive synthesis — that a sound can be constructed from a superposition of a number of sinusoids — is based on the concept of the Fourier transform and its inverse, which take an infinite range of time into account. In fact, any signal which is not infinite in time corresponds in the frequency domain to localized continuua of frequencies, rather than discrete frequencies [26]. In other words, any signal represented by discrete frequencies (i.e., individual sinusoids) actually exists infinitely in time, and is hence a steady-state representation.

Additive synthesis is unlike the other steady-state methods in that it does not precalculate the harmonic superposition (rather it creates it by brute-force summation), as such, its artifacts in the presence of modulation tend to be a bit different. Since theoretically we can modulate all the component oscillators, we can tend to get what appears to be a modulated signal audibly free of artifacts. However, it is usually the case that the signal being generated does not match the ideal signal being approximated. For example, if only modeling frequencies up to a certain limit, then frequency components due to modulation of otherwise inaudible harmonics past the limit, which would otherwise come down into the audible range, would be left out of the final signal. However, such frequencies are usually not missed by the listener.

The primary drawback of this method is the cost of the calculation of the oscillators. Figure 4.18 (on p. 232) shows the number of harmonics below half the sample rate for frequencies in various ranges, and we can see that at low frequencies, the number of harmonics easily becomes unwieldy, if not inefficient. However, at very high frequencies, the number of harmonics is actually quite low, and one might consider using discrete-oscillator additive in hybrid with some other technique with an opposite cost trend (i.e., one which gets more expensive at high frequencies but is rather inexpensive at low frequencies).

Variations on additive synthesis have been proposed over the years, including Inverse-FFT synthesis (to be discussed later), group additive [143] (grouping a large number of harmonics into a

smaller number of wavetables which are combined on playback), and multirate additive [202]. Finally, just about any method can be used in additive-like ways, by simply adding together multiple waveforms that have relatively simple spectra to produce a more complicated spectrum.

Wavetable Synthesis

One of the very earliest synthesis techniques used in computer music was periodic wavetable synthesis [163] (not to be confused with sample playback synthesis which is also called wavetable synthesis these days). In this technique, a wavetable contains only one period of the desired tone, sampled at a high rate, such as $N = 512$ samples per period. Playing out the table repeatedly generates a periodic waveform with fundamental frequency f_s/N, where f_s is the sampling rate. Skipping every other sample on playback yields a fundamental frequency of $2f_s/N$ and so on. Intermediate frequencies f_0 are obtained using a non-integer skip-factor, or "phase increment," given by $\text{inc} = Nf_0/f_s$; in such cases, the wavetable address has a fractional part which is often either discarded or used to round to the nearest integer (which only works if the wavetable is sufficiently oversampled, usually requiring a very large oversampling factor), or it is used to determine an interpolated table value. By far the most common interpolation technique is *linear* interpolation. However, higher order interpolation methods, especially Lagrange [227] and bandlimited interpolation [242], have been used commercially. Moore presented a classic analysis of interpolation-error noise versus table length for sine-wave tables in [174], which may be applied to the harmonics of the table.

Interpolated wavetable synthesis is not guaranteed to be bandlimited when the phase increment is larger than one sample. In these cases, one is performing the equivalent of a decimation of the waveform, with the decimation factor being equal to the phase increment in samples. Therefore, for nonsinusoidal wavetables, an upper bound (for a particular quality or lack of aliasing) is imposed on the phase increment by the highest harmonic of the signal in the wavetable ($\max(\text{Inc}) = Nf_0/(f_s n_h)$, where n_h is the harmonic number of the highest harmonic.). In such situations, higher-order interpolation may be used to effectively perform an antialiasing filter on the waveform. However, the required filtering can become expensive. In general, one can consider the problem to be one of sample-rate-conversion, where the output sample-rate is varied to produce various pitches, with high pitches requiring a high conversion ratio and hence requiring a narrow antialiasing filter. If the restriction on exact bandlimiting is relaxed, then the phase increment can have a higher bound, based on the highest harmonic whose amplitude is large enough to be objectionable when aliased.

Interpolated wavetable synthesis is equivalent in principle to *additive synthesis* employing a digital sinusoidal oscillator on every harmonic. In practice, wavetables were generally computed as a weighted sum of harmonics up to some maximum harmonic number (though in recent years, they may simply be processed recordings of a waveform). Thus, we may either add the outputs of N

oscillators together, or we may add their effective wavetables together with the same weightings to obtain a single wavetable oscillator. If the highest harmonic frequency is well below half the sampling rate, and/or if the harmonic amplitudes decrease rapidly with harmonic number, as normally happens in natural waveforms, inexpensive interpolation techniques such as linear interpolation can give high quality results. In the presence of modulation, this precalculation of the summed harmonics can be the source of artifacting (see Section 4.4.4).

Bandlimited Interpolation

Here we look a bit deeper at the sample-rate conversion interpretation given above. We follow concepts described in [242].

For a general discrete-time signal $x(nT)$, exact bandlimited interpolation, which we call *sinc interpolation*, is carried out by placing a sinc function whose zero-crossing width is the sampling period T on each sample and summing them all up. This is interpretable as reconstructing the sequence into continuous time using a brickwall reconstruction filter (the sincs).

To resample $x(t)$ at a new sampling rate $f_s' = 1/T'$, we need only evaluate the sum at integer multiples of T', as long as the output sampling rate is higher than the sampling rate of x, otherwise, x must be filtered with an antialiasing filter to remove harmonics which would be above half the new sampling rate. This is often accomplished in many design procedures by changing the width of the sinc in the above equations to the larger of T and T'.

Now, the summation cannot be implemented in practice, because the sinc functions actually extend infinitely in time. It is necessary to window the ideal impulse response so as to make it finite. The Fourier transform of the window determines the rolloff rate of the equivalent lowpass filter, and hence the probability of aliasing. This is also the basis of the *window method* for digital filter design [212]. (See also the `fir1()` function in Matlab.) While many other filter design techniques exist (e.g., [14]), the window method is simple and robust, especially for very long impulse responses. The filter impulse response ends up being very long because for most implementations it is heavily oversampled to increase its accuracy when used as a lookup table. Interpolation techniques of this nature which approximate sinc interpolation in the frequency domain are generally referred to as *bandlimited interpolation* techniques [227, 53, 242].

In general, this level of interpolation is not used in practice for wavetable playback. Most systems resort to some low-order polynomial (first-order (i.e linear interpolation) being the most common, but orders up to 10-15 have been used in commercial synthesizers).

Exact Wavetable Interpolation Schanze showed in [228] that a periodic wavetable of length N can be ideally reconstructed with the information in just one period, by noting that the overlapping

of the periodic sinc functions would produce the *periodic sinc* (or *time-aliased sinc*) function:

$$\mathrm{Sinc}_N(t) \triangleq \frac{\sin(\pi t)}{N \sin(\pi t/N)} \tag{4.7}$$

And that therefore a periodic reconstruction can be performed by placing a Sinc_N on each sample of the period and summing these (a finite sum). A more detailed derivation can be found in [239] and [248]. However, just as the bandlimited interpolation mentioned in the previous section is rarely used in practice in favor or low-order polynomial interpolations, this method is rarely used in practice for synthesis.

"Bank of Wavetables", a.k.a. "MIP Mapping" Given that wavetable technology has become the *de facto* standard for most synthesis hardware and a large percentage of synthesis software, many systems are well-suited to wavetable-based methods. Now, the method described above has shortcomings when the system must resample the wavetable upwards at a large ratio, especially if the wavetable is not oversampled.

However, taking a technique from sampling synthesis, we can distribute multiple samples across the pitch space and transition between them as we vary frequency. Now, most sample-based synthesizers already have this capability, but they only choose which sample to use at the start of a note. However, since virtual analog can theoretically sweep over the entire frequency range at any moment (regardless of when the "note" might have started), the algorithm needs adjustments to continuously choose which wavetable(s) to use. Such modifications are often not difficult, depending on the synth architecture.

As such, one can set up a system whereby a wave bank is set up with tables containing various numbers of harmonics (for example, in an octave-spaced bank, each wavetable would have twice the number of harmonics as the previous one). These can then be switched between and the maximum upsampling ratio therefore limited, thus allowing the implementation of more inexpensive interpolation.

However, since switching instantaneously between wavetable banks as the frequency crosses certain boundaries would usually cause discontinuities in the signal (i.e., clicks), a further extension of the algorithm is to interpolate between adjacent wavetables according to the location of the fundamental frequency in the current octave, essentially "fading out" or "fading in" the wavetables, so that they do not come into and out of existence discontinuously. Note that this implicitly assumes that all the harmonics that are in common between neighboring tables are of the same amplitude, frequency, and phase, so that interpolating between them will not get any unexpected cancellations (i.e., only the harmonics that are different between tables should change with the interpolation amount).

We should note that this method has a clear parallel in the world of computer graphics, whereby surface texture maps are split into a series of increasingly lowpassed pre-filtered maps in order to

implement much simpler on-the-fly texture resampling. In that world, the technique is called "MIP Mapping", introduced in 1983 by Lance Wiliams [289]. Interestingly, some have begun calling this multi-wavetable synthesis method by the same name, since the connection is obvious once one realizes it.

This method can implement just about any static waveform, and via combinations of sawtooths can produce pulse-width modulation without requiring changes to the waveforms. However, modifications such as sawtooth slope modulation cannot be directly implemented without moving to further interpolation axes in the wavetable bank (actually quite straightforward, though memory usage grows fast with increasing interpolation dimensions).

There are some design tradeoffs that can be made in deciding how the wavetables are laid out in memory. In graphical MIP Mapping, the filtered versions of the textures are also downsampled, so that they take up less space (in other words, each level is "critically sampled"). In an octave-spaced bank, the whole bank would thus take less than twice the size of the bottom octave, as each subsequent octave's bank would be half the length. However, this would require a bit of extra pointer math to derive the pointers into each of the wavetables (though in a fixed-point-phase system, that should be just a shift). Another possibility is to have all of the wavetables the same size. This uses much more memory (N times the number of tables in the bank), but has the advantage that a single phase offset can be used to index into all the tables.

This method has been described by Phil Burk in [31] and used in his late-1990s synth JSyn [30]. Julius Smith and the author also mentioned the method in class lectures [250] around that time, and there are anecdotal mentions of it already being used by a few commercial synthesizer manufacturers by then.

In general, this technique is very useful, straightforward to understand and implement, and depending on memory cost and speed, probably quite efficient. As such, it may be the preferred implementation for many designers. However, on systems where memory is a premium, or memory access is slow, this technique is much less desirable.

DSF Synthesis

In [175] (partially summarized in [65, pp. 149–154] and [221, pp. 260–261]), Discrete-Summation Formulae (DSF) are proposed for synthesizing bandlimited periodic signals based on the identity[3]

$$\sum_{k=0}^{N-1} z^k = \frac{1 - z^N}{1 - z} \tag{4.8}$$

and replacing z with the exponential form $\sin(x) = (e^{jx} - e^{-jx})/(2j)$ (or the cosine form $\cos(x) = (e^{jx} + e^{-jx})/2$). After some algebra and various trigonometric identities, the sum of a geometric

[3]Note that Winham and Steiglitz [292] also described similar equations for use in sound synthesis

series of sinusoids can be derived:

$$\sum_{k=0}^{N-1} a^k \sin(\theta + k\beta) = \frac{\left[\sin(\theta) - a\sin(\theta - \beta) - a^N \sin(\theta + N\beta) a^{N+1} \sin\left(\theta + (n-1)\beta\right)\right]}{1 - 2a\cos(\beta) + a^2} \tag{4.9}$$

By setting θ and β to various values, a wide class of bandlimited waveforms can be generated. Bandlimiting is achieved by controlling N such that the highest frequency generated is less than $f_s/2$ (for example, for $\theta = 0$ and $\beta = f_1 t, N = \lfloor f_s/(2f_1) \rfloor + 1$). Note that Csound's `buzz` and `gbuzz` unit generators implement this method [22].

This method is a classic example of a steady-state method: it assumes a set of discrete sinusoids and uses a closed-form version of the sum to calculate the waveform.

4.2.6 Bandlimited Synthesis Review: Non-Steady-State Algorithms

As opposed to steady-state algorithms, which generate for any point in time a sample of a wave that is assumed to be exactly periodic, non-steady-state algorithms make no assumption of periodicity. Instead, they only consider the signal locally in time. The extreme of such locality is to consider only the value at the current point in time, and the "trivially-generated" wave algorithms fall into this class. These algorithms simply calculate the ideal waveform at any point in time. Of course, as we have noted, such waves are aliased. To apply bandlimiting (i.e., filtering) to such algorithms requires that their "window of locality" be made wider than just a single sample. Bandlimited versions of such algorithms thus must generate a filtered version of their ideal signal. However, since non-steady-state algorithms assume that the signal will change over time, and since the extent of the filtering is limited by processing capability, the filtering must have a finite extent. That is where this class of algorithms breaks with the ideal: by limiting the extent of the effective bandlimiting filters to some finite width in time, and hence limiting their filtering to some finite rolloff rate (thus never completely removing aliasing, rather just reducing it strongly).

By not being able to predict the history and future of the waves, as is possible in the steady-state assumption, these methods must specifically perform bandlimiting "on the fly" during playback. This usually involves implementing, either directly or via some assumption-based simplification, some sort of high-quality resampling.

Oversampling Trivially-Generated Waves

This is the most straightforward method for generating approximately bandlimited signals, and it has the desirable property of applying to any waveform that can be calculated, as the bandlimiting is implemented brute-force as part of the downsampler, and is thus independent of the wave being generated. Most other bandlimited-wave generation algorithms limit the waveform selection in some way to just those waves which apply to the particular simplifications of the bandlimiting that

are implemented. On the other hand, since oversampling implements its bandlimiting by brute-force methods, it tends to be rather expensive.

To repeat a phrase from elsewhere in this thesis: Oversampling makes things easy, but expensive.

However, the waves are usually trivial to generate in this domain, so if oversampling is necessary for other reasons, this methods is easy and works.

Note that when using oversampling, one tends to stay away from implementing impulse trains. In order to assist the downsampling antialiasing filter, one is better off generating waveforms which have some sort of rolloff in their harmonic amplitudes. Impulse trains do not have any rolloff in their harmonic amplitudes, so one stresses the downsampler most strongly with such waveforms.

Reduced-Aliasing Shaped Waves

Algorithms of this type tend to implement the simplest approximations to bandlimiting that are possible. Usually, these algorithms are derived not in terms of bandlimiting, but in terms of reducing the amount of harmonic energy.

Typical early algorithms were attempts at inexpensive sine-wave generation. For example, a triangle wave may be turned into an approximation of a sine wave by running it through a polynomial which implements the first couple terms of the Taylor-series approximation of sine, or one of any number of other approximations to that shape. Other examples include the use of a bipolar parabola as a cheap approximation to a sine.

Later, Lane et al. described a method [150] for generating a reduced-aliasing harmonically-rich waveform by taking the absolute value of a sine wave and filtering it to give the desired rolloff rate (in their case a sawtooth rolloff), and also to remove high-frequencies, where aliasing is worst. Of course, such an algorithm still has aliasing problems when the aliasing components descend into the region not being filtered out, and hence the method tends to have a low upper frequency limit.

Recently, Vesa Välimäki has approached sawtooth generation (and hence rectangle-wave generation by the method of subtracting two sawtooths) from this perspective [275] (further analyzed in [276] and [277]), applying preemphasis and noise shaping concepts to the waveform generation. The technique is called “differentiated parabolic wave” (DPW), and the general concept is that a sawtooth is the derivative of a piecewise parabolic wave. Since the parabolic wave was harmonics that fall off twice as fast as those of a sawtooth (due to the integration/differentiation relation between them), its aliasing components will have dropped to lower amplitudes by the time they wrap around back to the audio band. Thus, when this signal is re-differentiated, which boosts the high frequencies of the discrete-time signal, the aliased components in the low frequencies will stay at their low amplitudes (particularly compared to where they would be in a directly-generated sawtooth).

In the same paper, an enhanced version of the algorithm which performs the initial steps at an

oversampled rate (2X is described), which then antialiased and downsampled before performing the differentiation at the base band. Since, in general, calculating signals at oversampled rates causes them to have reduced aliasing (as long as the downsampling antialiasing filter is works well), the result is further reduced aliasing, primarily at the cost of the antialiasing filter (since the parabolic wave calculations are quite inexpensive, even oversampled). The paper discusses the pros and cons of extremely low order antialiasing filters, in particular 2-point averaging, which at 2X oversampling will notch those frequencies that would alias to and near DC. Within the goal of extreme low cost, that was considered sufficient. In [276], higher-order filters are analyzed. In [277], the extension to controllable-duty-cycle square-waves is presented (by subtracting sawtooths of different phase offset), and a triangle-wave method is presented which works very similarly to the sawtooth algorithm by differentiating a bipolar parabolic wave.

These methods take one extreme of the cost-vs.-aliasing tradeoff by implementing extremely inexpensive algorithms at the cost of some midrange aliasing.

Additive Synthesis Using Inverse FFT

Additive synthesis systems that use the inverse FFT to compute the oscillators [224] [154] [153] are bandlimited, by definition, because the inverse FFT only generates frequencies up to $F_s/2$. Similarly, frequency aliasing is trivially avoided since such frequencies would have to be specifically modelled. Instead, in inverse-FFT synthesis, *time*-aliasing is the issue to worry about.

IFFT-based additive synthesis straddles the duality between frequency and time, and as such, can in some situations be viewed as a steady-state synthesis method, and in other situations viewed as more localized in time. Since any particular frame represents a specific limited range of time, and since frames are generally crossfaded by virtue of some sort of overlap-add technique, many artifacts which might otherwise occur due to its local steady-state nature may be made relatively unnoticeable.

Note that fast modulation of the oscillator is tricky in IFFT synthesis. Modulation that must occur near or faster than the frame rate must be turned into the equivalent effect on the spectrum. Work has been done on deriving the spectral bin values necessary for various sub-frame pitch trajectories [96], but it requires special-case math depending on the shape of the modulation, and so it is difficult to add general modulation into the method.

As mentioned previously, the major drawback of additive synthesis is the amount of data that must be handled to perform the synthesis. The existence of the other inexpensive algorithms mentioned in this chapter, such as the DPW and various BLIT/BLEP algorithms, tend to make IFFT-based synthesis seem like a bit of overkill for Virtual-Analog signal generation, though it is very useful in more general sound-synthesis applications.

4.3 Deriving Waveforms from Other Waveforms

As we saw in the introduction, there is a history in analog synthesis of deriving waveforms from other waveforms. We have also seen in the review some further cases where this was done as part of a waveform derivation, in order to help reduce aliasing. It was also noted that many of the analog methods are unfortunately highly nonlinear, such that their use in digital implementations would generate significant aliasing (or exacerbate any existing aliasing), so would not be of much use, except at particularly low frequencies, where aliasing is not as audible.

4.3.1 Linear Operations

However, it is possible to modify signals via linear operations as well as by nonlinear operations. Linear operations will not generate any frequencies not already in their input signals so that, if a bandlimited signal is presented to a linear operation, its output will remain bandlimited.[4]

We have already come across the use of summation to turn two sawtooth waves (or one delayed sawtooth wave) into a rectangle wave. Summation is of course linear, and thus if the sawtooths are bandlimited, the rectangle will be bandlimited as well.

We will now discuss the concept of deriving sawtooth, rectangle, and triangle waves from impulse trains. Since impulse trains are in some ways nearly degenerate signals, it is conjectured that bandlimited versions can be more easily generated than more general signals might be (this will be the subject of Section 4.4). This method was first presented by the author and Julius Smith in [248].

4.3.2 Successive Integration of BLIT

It turns out that the three classic waveforms can all be derived from impulse trains via the application of integration in various combinations: Sawtooth and Rectangle from single integrations of impulse trains and Triangle from a further integration of the Rectangle wave.

Sawtooth from BLIT

A continuous-time sawtooth function can be generated as follows:

$$Saw_{\text{CTS}}(t) = \int_0^t \text{CIT}(\tau) - C_1 \, d\tau \tag{4.10}$$

where $\text{CIT}(\tau)$ is a continuous-time impulse train, and $C_1 = \int_0^T \text{CIT}(\tau) \, d\tau$, the DC component of

[4] However, if low-level aliasing components exist in the input, a linear operation might amplify them.

the impulse train. This converts directly to discrete-time (via the impulse-invariant transform):

$$\begin{aligned} Saw(n) &= \sum_{k=0}^{n} BLIT(k) - C_2 \\ \Leftrightarrow Saw(z) &= \frac{z}{z-1}(BLIT(z) - \mathcal{Z}(C_2)) \end{aligned}$$

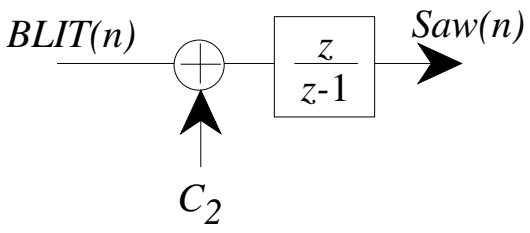


Figure 4.5: Direct Sawtooth Generation

which is trivially implementable with a single sum and one-pole digital filter. For example, see Figure 4.6. The offset C_2 is the average value of $BLIT$, which should be subtracted off to keep

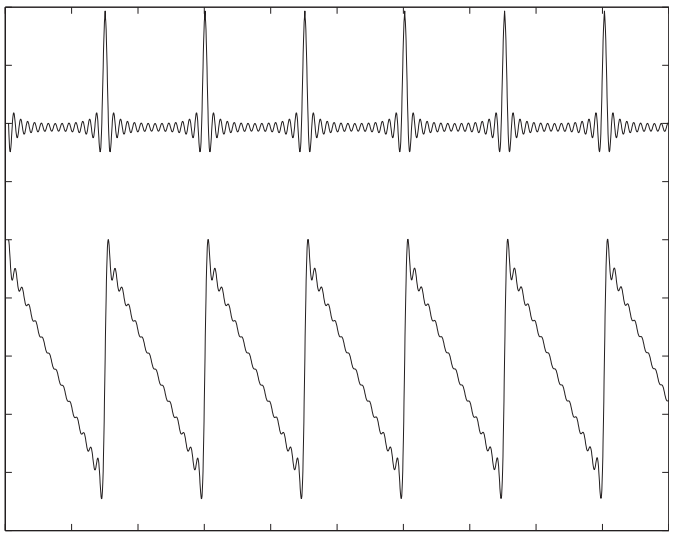


Figure 4.6: Generating a sawtooth. A BLIT (upper signal) is integrated (with the DC value removed) to get a sawtooth (lower signal).

the integration from ramping off to infinity (or saturating). C_2 is a function of frequency (for the

steady-state case, $C_2 = a/T$, where T is the period of the impulse train, and a is the amplitude of the pulses. Depending on the initial conditions of the integrator, a DC offset may end up on the output of the integrator (this will be discussed in Section 4.3.3). Alternatively, the BLIT could be passed through a DC blocking filter such as

$$\text{Block}(z) = \frac{z-1}{z-(1-\varepsilon)} \tag{4.11}$$

Which will have the effect of drifting the BLIT down by the right amount to have no DC component, and thus no offset need be calculated (though one may want to be careful about the width of the notch in the DC blocker versus the BLIT frequency, else the low harmonics of the BLIT may also be attenuated: the tradeoff is between low-harmonic attenuation and responsiveness to changes in the DC value of the BLIT due to changes in frequency).

Rectangle from BP-BLIT

A continuous-time rectangle wave can be computed as:

$$Rect_{\text{CTS}}(t) = \int_0^t \text{CIT}(\tau) - \text{CIT}(\tau - t_0) - C_3 \, d\tau$$

Which discretizes to:

$$\begin{aligned} Rect(n) &= \sum_{k=0}^{n} \text{BLIT(k)} - \text{BLIT(k} - \text{k}_0) - C_4 \\ &= \sum_{k=0}^{n} \text{BP-BLIT}_{k_0}(k) - C_4 \\ \Leftrightarrow Saw(z) &= \frac{z}{z-1}(\text{BP-BLIT}_{k_0}(z) - \mathfrak{Z}(C_4)) \end{aligned}$$

Where BP-BLIT is a "BiPolar" BLIT, whose pulses alternate sign. See Figure 4.7 for an example. See Section 4.4.2 for discussion on methods for efficiently generating BP-BLIT. It turns out that a bipolar impulse train has a DC component of zero, which means that $C_3 = 0$ (and subsequently $C_4 = 0$). The rectangle width is controlled with k_0 (the spacing between an "up" pulse and the subsequent "down" pulse, in samples), which can be varied to give PWM (pulse-width modulation). The range of k_0 in these equations is [0,Period] (or, depending on the implementation of the BLIT, the PWM control may also be in the range [0,1]). Depending on the initial conditions of the integrator, there may be a DC offset on the output. Also, time-variation of k_0 and/or the BP-BLIT frequency will cause temporary DC offsets which can get "caught" in the integrator (we will look into this in Section 4.3.3).

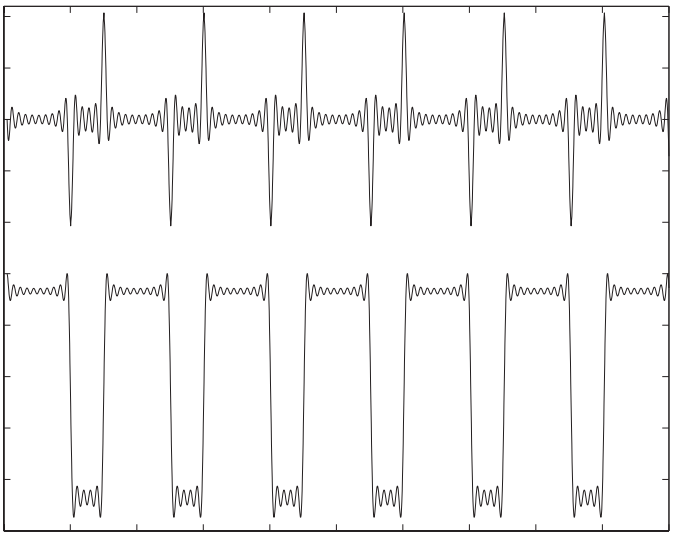


Figure 4.7: Generating a rectangle wave. A bipolar BLIT (upper signal) is integrated to get a rectangle wave (lower signal).

We can also note the relation between integrating BP-BLIT and subtracting sawtooths: essentially the integration is transposed with the subtraction:

$$\begin{aligned}
Saw(n) &= Integrate(\text{BLIT}(n) - C_1) \\
Rect(n) &= Saw(n) - Saw(n - k_0) \\
Rect(n) &= Integrate(\text{BLIT}(n) - C_1) - Integrate(\text{BLIT}(n - k_0) - C_1) \\
Rect(n) &= Integrate(\text{BLIT}(n) - \text{BLIT}(n - k_0)) \\
Rect(n) &= Integrate(\text{BP-BLIT}_{k_0}(n))
\end{aligned}$$

Since the DC offsets are dependent on the BLIT frequency and since both sawtooths have the same frequency, they would have the same internal offsets (assuming other offsets due to time-variation are taken into account).

Triangle from Rect

A triangle wave can be generated as:

$$Tri_{\text{CTS}}(t) = \int_0^t Rect_{\text{CTS}}(\tau) - C_5 \, d\tau \tag{4.12}$$

Where C_5 is the DC component of the rectangle wave: $C_5 = \int_0^T Rect_{\text{CTS}}(\tau)\, d\tau$. This discretizes to:

$$\begin{aligned} Tri(n) &= \sum_{k=0}^{n} Rect(k) - C_6 \\ \Leftrightarrow Tri(z) &= \frac{z}{z-1}(Rect(z) - \mathcal{Z}(C_6)) \end{aligned}$$

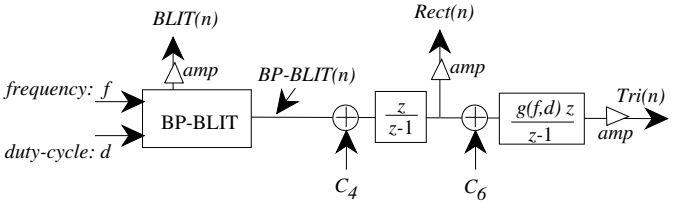


Figure 4.8: Rectangle and Triangle Generation

The offsets C_5 and C_6 are functions of the rectangle wave duty cycle and of a DC offset that arises from the initial conditions of the integration that produces the rectangle wave: $C_6 = k_0/Period + C_7$. C_7 is a function of $BLIT(0)$, the initial condition of the integration. To get appropriate amplitude on the triangle wave (so that its extrema are the same size as those of the BLIT and Rectangle wave), a frequency- and duty-cycle-dependant scaling must be performed on the Triangle integration:

$$\begin{aligned} Tri(n) &= \sum_{k=0}^{n} g(f,d)(Rect(k) - C_6) \\ g(f,d) &= \frac{2f}{d(1-d)} \end{aligned}$$

Where f is the frequency in units of (cycles/sample), and d is the duty cycle ($d \in [0,1]$).

For example, see Figure 4.9. Note that if the Rect wave has 50% duty cycle, then the triangle wave has equal up and down slopes (which corresponds to a common definition of the triangle wave). However, if the Rect wave has some other duty cycle, then the slopes will not be equal, producing what is variously referred to as a "variable-slope sawtooth", or a "variable-slope triangle" wave. Theoretically, one could take the duty cycle to a limit of almost 0% or 100% and generate the equivalent of the sawtooth, but care must be taken if the duty cycle causes either the up or down pulse to become narrower than a sample, as the required gains may become problematic. Further, since the method in Section 4.3.2 requires a single integration from the BLIT whereas this method takes two integrations, it is less susceptible to DC-offset issues (Section 4.3.3), and so is generally preferred for generation of "pure" sawtooth waves.

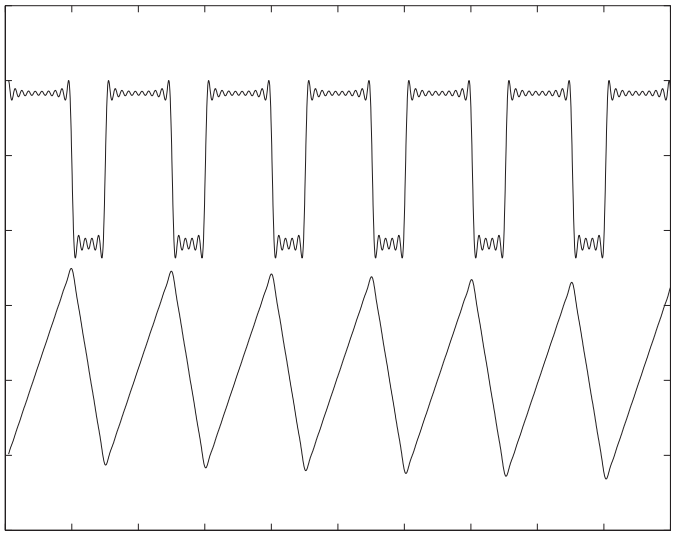


Figure 4.9: Generating a triangle wave. A rectangle wave (upper signal) is integrated to get a triange wave (lower signal).

4.3.3 Handling DC Offsets

Since even temporary DC offsets get integrated by the integrators and can cause the waveforms to drift out of range, etc., they must be dealt with for such a method to be practical.

Appropriate Scalings/Offsets and Non-Steady-State Fixups

In order to keep the integrators from ramping their outputs to infinity (or at least beyond the capabilities of the number system or the DACs), any DC offset in the input to the integrator must be avoided. As already noted, BP-BLIT has no DC offset, so there need be no special offsets for the square-wave integration. The initial conditions of the first integrator, however, can produce a DC offset on the output that must be canceled before the second integration. The value of this offset is also dependant on the duty-cycle of the signal,[5] so that the correct initial condition will change based on: (1) desired phase, and (2) desired duty cycle. There is also a frequency-dependant scaling necessary for the second integration (because the triangle slopes are proportional to frequency), whose effects must be accounted for during frequency changes. It is important to remember that the old state (right before the change) acts as a new "initial condition" for the integrator when the parameters are changed.

[5] Any amplitude dependance can be avoided by assuming unit amplitude in all the integrations and simply post-scaling the outputs. Thus the scaling gains are: on input of first integrator, 1.0; on input of second integrator $2/(Td(1-d))$ where d is the duty cycle ($\in [0,1]$) and T is the period in samples.

In practice, the accounting necessary for multiple integrations in the presence of various frequency, amplitude, phase, and pulse-width modulations (not to mention oscillator sync), make trying to subtract off the appropriate offsets either impossible or at least not viable in general. However, there is still a solution.

Leaky Integrators

The use of pure integrators can present problems in the presence of numerical errors, such as round-off. These errors accumulate in the integrators, causing unwanted (and unpredictable) offsets in the signals, which can destroy the ability to create the desired waveforms, especially in the second integration. Add to this the difficulties mentioned in keeping track of all the DC offsets in the presence of time-variation, and some more automatic method for dealing with DC is in order.

Therefore, we move the poles of the filters that implement the integration slightly in from the unit circle. These "leaky" integrators slowly forget bad initial conditions and numerical errors, so that they don't continue the build up forever. The impulse response of a leaky integrator is an exponential that slowly decays to zero. This has two more effects:

First, the decay rate places an effective lower bound on oscillator frequency (especially in cases where the signal is to be used as a control signal), as when the period gets on the order of the decay time, the 'held' output (as in a rectangle wave), is no longer even close to being constant over its portion of the cycle. Equivalently, the low harmonics of the oscillator waveform can end up with lower amplitudes than expected, as the leaky integrator's low-frequency boosting is "chopped off" by the leakiness. The faster the leak (i.e., the faster the decay of the impulse response), the higher the frequency that the integration gain is chopped at, and the higher the frequency at which signal harmonics will begin to be attenuated from their theoretical amplitudes. In audio signals, this is probably not a problem (it just attenuates the lowest-frequency harmonics, which may not be terribly audible anyway), but in control signals, where the exact shape of the signal is important, this can be a big problem. On the other hand, LFOs are often not modulated as much as audio oscillators by many users, and so DC-offset accounting may be viable there, in tandem with very slow leaks to handle numerical grit.

Second, In steady-state, the outputs of the integrators (for Rectangle and Triangle waves) will have no DC component (because BP-BLIT has none and non-BP blit can be dealt with using a DC blocker), regardless of initial conditions, since the leaky integrators eventually forget them. Thus, if one can live with occasional transient DC offsets (which decay at the leak rate), then just the presence of the leaky integrators can handle all offset cases.

Variable-Leak Integrators

However, practical implementations have shown that these "transient DC offsets" can be quite audible, as "bangs" or "bumps", or situations in which the signal temporarily gets clipped out of

existence as the temporary DC offset swings the signal past clipping limits. Therefore, we introduce a further extension to the leaky integrators: temporarily increase the leak rate of the integrators in question, to more quickly leech off the DC offset, and then slowly recover the leak rate to a steady-state location. This can be implemented by using integrators whose pole locations can be signal inputs, which are hooked up to one-pole decays, which decay towards the steady-state value of the pole location, and get "pinged" to faster pole values as necessary. In other words, the integrators can be made to temporarily "center themselves". A side benefit of this technique is that the steady-state leakiness can be placed much slower than it would have been otherwise, since most of the DC can be leaked out during the temporarily fast leakiness (some leakiness is still a good idea to handle any numerical rounding issues or the like). There is sill the problem of identifying when to speed up the leakiness. As most temporary DC offsets occur due to modulations, one could sum up the derivatives of the various modulation sources feed that in to the integrator-pole decay units, basically setting the leak rate to be proportional to the modulation rates. It may also be possible (though this hasn't been tried), to implement some sort of feedback method whereby the average of the oscillator's output over some period of time is used to determine if the oscillator has a DC offset, which can then be fed back to control the leakiness rates. This system would have to be verified to see if it can react fast enough to suppress "bumps" due to unexpected modulations.

Definitions of Amplitude

Moorer presents a discussion of amplitude compensation in his DSF paper [175]. Similar compensation is necessary in BLIT generation. The compensation to be used depends on how one defines amplitude, which depends on how the signal is to be used. If the signal is to be used as an audio signal, signal power or some psychoacoustic loudness measure is appropriate, but if the signal is to be used as a control signal, a maximum-value (Chebychev) measure is more appropriate.

4.3.4 Can DSF directly generate sawtooth, square, triangle waves?

One may wonder if DSF can be used to directly generate these waves via appropriate settings of the a parameter. The short answer is no, because (as can be shown from the integrations) the square-wave's harmonics' amplitudes fall of as $1/f$ (or, equivalently, as $1/n$, where n is the harmonic number), and the triangle's as $1/f^2$. DSF harmonics, on the other hand, fall off as a^n (i.e. exponentially in n rather than as a polynomial in n). This keeps the DSF from being able to generating exact rectangle or triangle waves, but depending on the circumstance (i.e., the signal not being used as a control signal, etc), fitting an a^n fall-off rate to approximate $1/n$ or $1/n^2$ might be close enough (perceptually, for example). Other possibilities might be the use of a post-filter to shape the spectral rolloff. This brings up the next question as to whether the DSF harmonics are in the correct phase relationship to give the desired wave shapes. This is a topic that the author has not yet looked into.

4.4 Bandlimited Impulse Train (BLIT) Generation

Some of the previously mentioned techniques can be used to generate BLITs, but can also generate other waveforms. Now we will discuss techniques designed especially to generate BLITs, without consideration for extension to other waveforms (although it may be possible in some cases). By restricting to bandlimiting impulse trains, which is mathematically much simpler than generic waveforms, these methods may be much more efficient than the other methods.[6]

4.4.1 Steady-State Synthesis

Again, these methods can be classified in terms of whether they are based on a steady-state assumption or not.

Additive

True additive synthesis methods can gain no additional efficiency when implementing impulse trains, so there is nothing new to say from the earlier discussion.

Sum of sincs and Sinc_M

As noted earlier, the standard operation before sampling is to apply an anti-aliasing filter. The ideal anti-aliasing filter has a continuous-time impulse response that is a sinc function with a zero-crossing interval of one sample:

$$h_s(t) = \text{sinc}(f_s t) = \frac{\sin(\pi f_s t)}{\pi f_s t}. \tag{4.13}$$

The ideal unit-amplitude impulse train with period T_1 seconds is given by

$$x(t) = \sum_{l=-\infty}^{\infty} \delta(t + lT_1) \tag{4.14}$$

Since this signal is nearly everywhere zero, the convolution by the antialiasing filter is a particularly simple expression (as compared to a more general convolution):

$$x_f(t) = (x * h_s)(t) \quad = \quad \sum_{l=-\infty}^{\infty} \text{sinc}(t/T + lP)$$

[6]The basic operation to be simulated is `Sample[ImpulseTrain(t) * sinc(t)]`, where "$*$" denotes convolution. Since ImpulseTrain is just impulses, the convolution is almost trivial. This operation would be much more difficult to calculate for more complex waveforms

where $P = T_1/T$ is the period in samples, (probably not an integer). The filtered impulse train is now sampled to obtain

$$y(n) = x_f(nT) = \sum_{l=-\infty}^{\infty} \mathrm{sinc}(n + lP) \tag{4.15}$$

As before, the above expression for $y(n)$ can be interpreted as a *time aliasing* of the sinc function about an interval of P samples. As with DSF, this infinite sum has a closed form solution:

$$y(n) = (M/P)\mathrm{Sinc}_M[(M/P)n] \quad (M \text{ odd}) \tag{4.16}$$

where[7]

$$\mathrm{Sinc}_M(x) \triangleq \frac{\sin(\pi x)}{M \sin(\pi x/M)} \tag{4.17}$$

Thus,

$$y(n) = \frac{\sin(\pi n M/P)}{P \sin(\pi n/P)} \tag{4.18}$$

or, in terms of frequency normalized to the sample rate ($f_n = f_1/f_s$), rather than period ($f_1 = f_s/P \rightarrow f_n =$):

$$y(n) = \frac{f_n \sin(\pi n M f_n)}{\sin(\pi n f_n)} \tag{4.19}$$

This function provides a closed-form expression for the sampled bandlimited impulse train (BLIT), and it can be used directly for synthesis in a manner similar to DSF. While P is the period in samples, M is the number of harmonics in the two-sided spectrum (i.e.,including the negative-frequency harmonics). It is always odd because an impulse train has one "harmonic" at DC, and an even number of non-zero harmonics always exists in the two-sided spectrum because the signal is real, (provided no harmonic is allowed at exactly half the sampling rate, which we assume and require). Now assuming that we have the maximum number of harmonics possible (i.e., all those out to but not including half the sampling rate), then M/P is always close to 1. When P is an odd integer, $P = M$, and $y(n)$ is simply $\mathrm{Sinc}_M(n)$. As P departs from M, the above equations implement a time scaling along with a compensating amplitude scaling. We can relate the number of harmonics M in the two-sided spectrum to the period P of the impulse train as

$$M = 2\lfloor P/2 \rfloor + 1 \tag{4.20}$$

i.e.,M is the largest odd integer not exceeding the period P in samples.

This method of synthesis is very similar to DSF, in that a series of harmonic partials are synthesized via a closed-form expression which allows the number of harmonics to be controlled[8] Note that DSF is often used with a constant number of harmonics, rather than harmonics that go all way

[7] Sinc_M has also recently been denoted as asinc_M, meaning *time-aliased sinc of period M*.

[8] One should be able to algebraically relate the two methods by setting the amplitude decay ratio of DSF to 1.0 and aligning the harmonic phases (by replacing sin with cos in the DSF formula) to derive the SincM formula.

out to $F_s/2$. This can produce a slightly different result, which can be reconciled using the above equation which relates M and P for a full-bandwidth impulse train.

Note that in normal operation, keeping M up to date while the oscillator frequency changes may involve a division (to get P if it isn't already derived for other reasons), but also may require changing M from one integer value to another from sample to sample, which can cause discontinuities in the signal (audible clicks). We will see this later in Figure 4.25 on p. 241.

4.4.2 Generating Bipolar BLITs with $Sinc_M$ and DSF

Difference of BLITs The most obvious method of generating the bipolar BLITs is simply to take the difference to two BLITs, one delayed (or otherwise phase-shifted) relative to the other. This is the technique most commonly used with BLIT-SWS (to be described later).

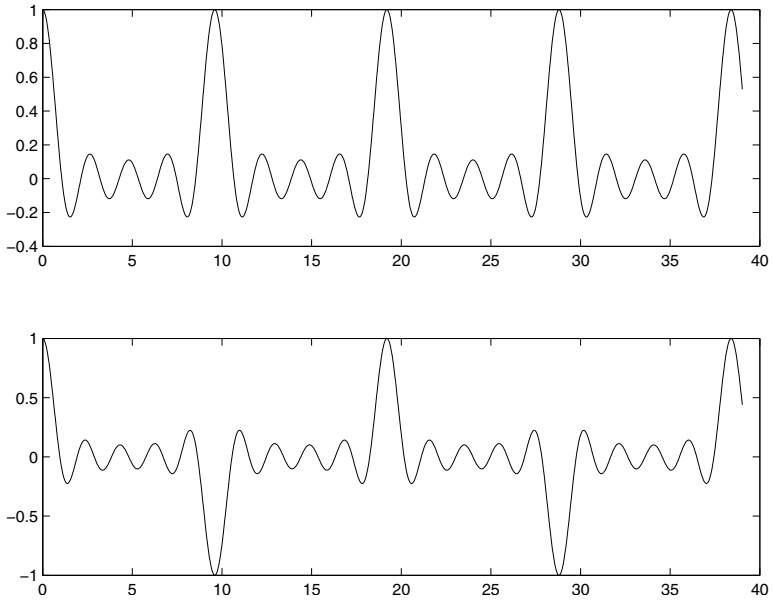


Figure 4.10: Top: SincM with M=9, P=9.6. Bottom: SincM with M=10, P=9.6.

SincM method One will note that the definition of SincM Equation 4.16 restricted M to be odd. If, instead, M is chosen to be even, then a bipolar signal is generated, at half the expected frequency (see Figure 4.10).

This behavior can be explained intuitively based on the fact that M represents the number of total harmonics in the full spectrum (i.e., negative frequencies included). See Figure 4.11. Therefore, since a unipolar signal would have a nonzero DC component, and since it is real and hence conjugate symmetric about DC, there will always be an even number of harmonics not on DC, plus the one on DC, making an odd number of harmonics. An even value of M, therefore, implies that

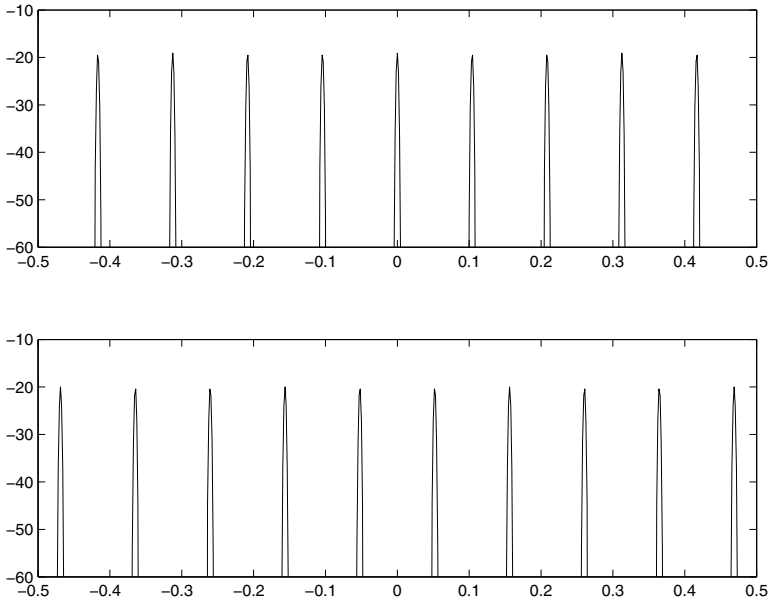


Figure 4.11: Top: Two-sided spectra of the signals in Figure 4.10. Top: SincM with M=9, P=9.6. Bottom: SincM with M=10, P=9.6.

there would be no harmonic on DC. The harmonics would still be evenly distributed through the spectrum, but would line up such that DC lands halfway between two harmonics. This leads to the harmonics landing on purely odd multiples of a frequency which is half the harmonic spacing, and hence describing a bipolar BLIT of half the frequency with a 50% duty cycle.

DSF method Since DSF has a few more degrees of freedom in its definition, we can generate a more flexible bipolar BLIT based on its equations.

First, we note that BLITs can be generated via DSF by replacing the sin by cos in the DSF formulas from [175] (this ends up essentially replacing sin by cos in the numerator of the original DSF equation, the denominator stays the same). See Figure 4.12.

50% duty cycle First, it can be shown that using a negative a in the DSF formula $\sum_{k=1}^{N} a^k \sin(0 + kf_1t)$ produces a signal that is shifted from the positive-a signal by exactly half a cycle (Figure 4.13), this gives a slightly more elegant way of producing the shifted BLIT than offsetting t. This can lead to showing that:

$$\begin{aligned} & \sum_{k=1}^{N} a^k \sin(a + bk) - \sum_{k=1}^{N} (-a)^k \sin(a + bk) \\ = \quad & 2a \sum_{k=1}^{N/2} (a^2)^k \sin(\ (a + b) + 2bk) \end{aligned}$$

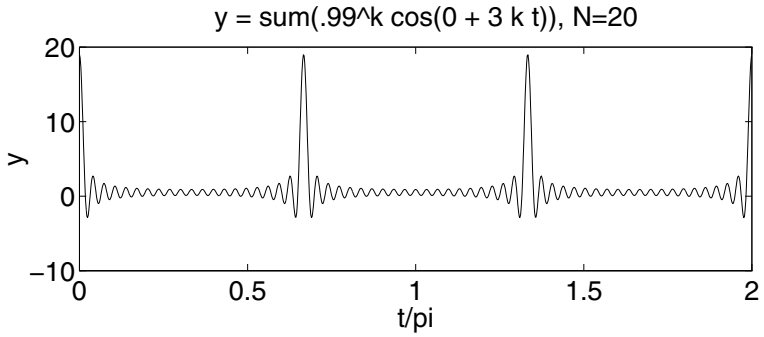


Figure 4.12: Using cosine-DSF to generate BLIT

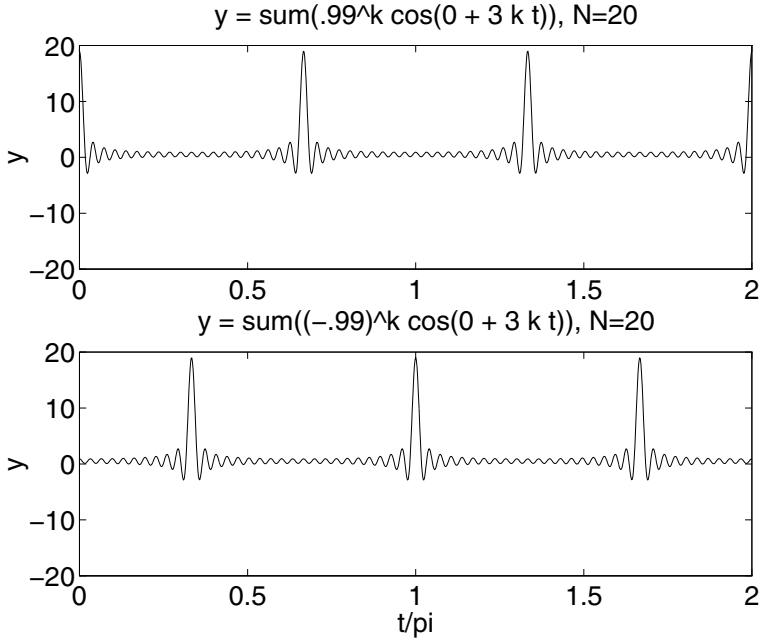


Figure 4.13: DSF: half-cycle shift

The same applies to the sum-of-cosines DSF, which is shown in Figure 4.14. Thus a 50% duty-cycle bipolar DSF BLIT can be generated almost as efficiently as a single DSF BLIT.

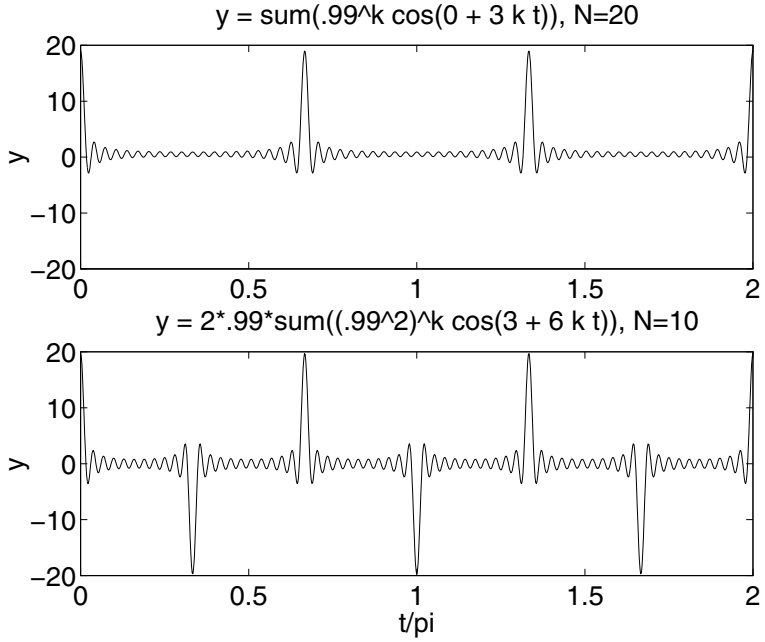


Figure 4.14: 50% duty-cycle BP-BLIT using DSF

PWM: For other duty cycles, there is another variation on DSF that is of interest. Let a be complex and take either the real or imaginary part of the DSF, this imposes a $\sin(k\angle(a))$ (or cosine) amplitude envelope onto the harmonics, which is equivalent to a comb filtering, which in turn is equivalent to summing a real DSF with a shifted version of itself (possibly with a sign flip), all of which can be shown mathematically. See Figure 4.15. This method implements BP-BLIT in essentially the same complexity as evaluating the difference of two real DSF BLITs, but with a bit more elegance. However, in synthesis systems where complex arithmetic is not already implemented, it may be considered too much work to use this method, and an implementor may just fall back to subtracting offset-phase versions.

4.4.3 Non-Steady-State Synthesis

Oversampling

As noted in the review, generating impulse trains via oversampling generally places unnecessary stress on the capabilities of the downsampler's antialiasing filter. As such, if oversampling is going to be used anyway, due to other decisions in the synthesizer, one is much better off directly

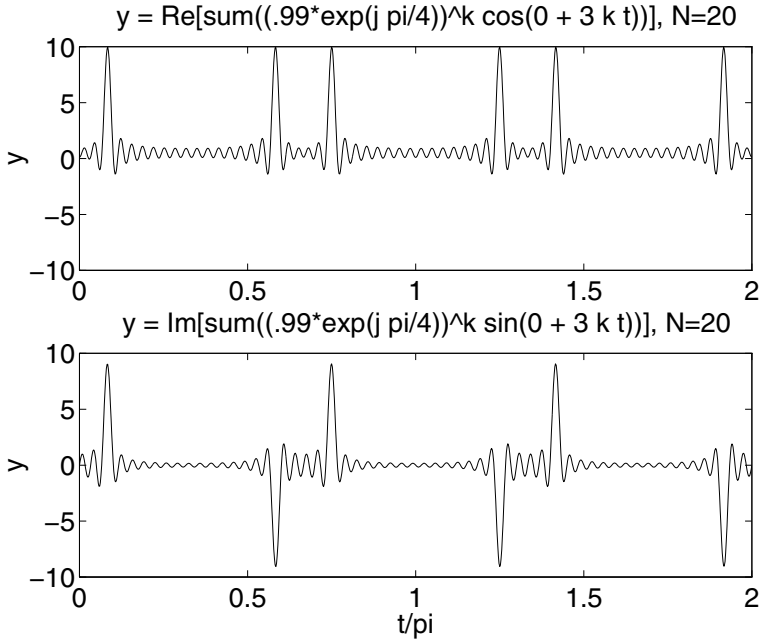


Figure 4.15: Using a complex multiplier in DSF to generate BP-BLIT

calculating the desired waveform (and all the classics are generally very cheap to calculate) rather than generating a BLIT via oversampling and deriving them from it. However, as noted, the design philosophy for this discussion is to look for situations where oversampling is not required, so we look elsewhere.

Sum of Windowed Sincs (BLIT-SWS)

An efficient method for synthesizing digital impulse trains may be based on the windowed-sinc method for general bandlimited interpolation [242]. The technique is equivalent conceptually to bandlimited periodic wavetable synthesis of an impulse train, as mentioned earlier: Bandlimited interpolation is used to convert the sampling rate of a discrete-time unit sample pulse train from a pitch which divides the sampling rate (so that the period is an integer) to the desired pitch. The rate conversion causes each unit sample pulse $\delta(n)$ to be replaced by a windowed sinc function $w(t)h_s(t)$ sampled at some phase which generally varies each period. As such, each pulse is separately generated and added into the output. If the period is shorter than the windowed sinc width, multiple sincs will overlap.

This method was first presented by the author and Julius Smith in [248].

Algorithm 3: Typical BLIT-SWS Algorithm

Data: Instantaneous frequency (phase increment) p_{inc}
Result: Audio signal $out[n]$

```
foreach n do
    p_n ← p_n + p_inc;
    if p ≥ p_thresh then
        // Find sub-sample offset of the pulse
        x ← findPulseOffset (p_{n-1}, p_n, p_0);
        startNewPulse (x);
        p_n ← p_n − p_thresh;
    end
    foreach Active Pulse Generator gen_i do
        out[n] ← out[n]+lookupSincTable (x_i);
        x_i ← x_i+sincstep ;
        if x_i >sinclen then
            removeFromActiveList (gen_i);
        end
    end
end
```

Typical BLIT-SWS Algorithm In order to reduce the amount of work the table lookup interpolation must do, we can oversample the windowed sinc table (as mentioned in [242]). the author's implementations have typically oversampled such that the table contains on the order of 128-1024 samples per zero crossing, which gives very clean results when used with linear interpolation, though that probably can be considered a bit of overkill, especially in memory-limited implementations (and there may be cache-clobbering consequences as well). the author has not deeply measured the effect of the table size on fidelity, (beyond analyses such as Figure 4.16), though Moore's analysis of sine-table noise in [174] may be used as a guide for the effect of the sinc sable size on the noise of the highest harmonics.

The choice of window can have a noticeable effect on the noise floor. Figure 4.17 shows spectra for integrated BLITs (i.e. sawtooths) using the same windows width (32 sinc zero crossings), table oversampling factor (1024 samples per zero crossing), and corner frequency ($0.9 f_s/2$). The difference is the window. The top spectrum is for a Blackmann-Harris window, the middle is a Kaiser windows with β = 3.5, and the bottom is a Kaiser window with β = 20. Note how the wider Kaiser window whose sidebands do not fall off very far, raises the general noise floor. Such a design would be at the limit of usefulness for a system with a 16-bit output, and as in [276] we see how the additional boosting of the noise can cause it to rise above the theoretical 16-bit noise floor.

Note that the algorithm shown is derived from a PET/Granular paradigm, where each pulse is a separate interpolated wave to be played out and overlapped. It is also possible to interpret the algorithm more from a sample-rate conversion standpoint and note that the system can be implemented via a polyphase structure. However, to be implemented in a pure polyphase bank, the

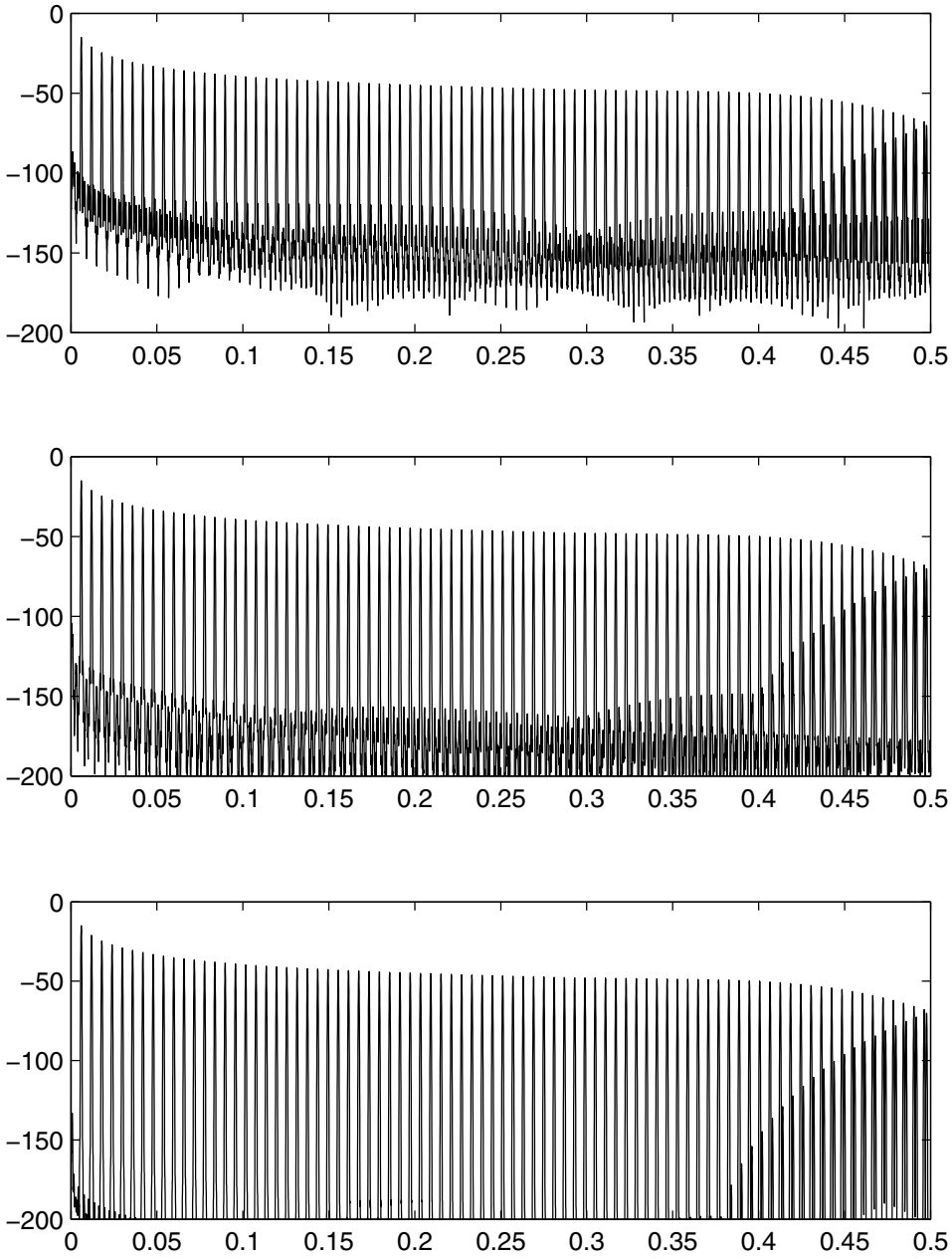


Figure 4.16: Effect of number of samples per sinc zero crossing on BLIT-SWS noise floor (linearly interpolated). 32-zero crossings, corner pulled back to $(0.9F_s/2)$, Kaiser window (β = 20 to get extremely low noise floor from the window). Top: 32 samples per zero crossing, Middle: 128 samples, Bottom: 1024 samples.

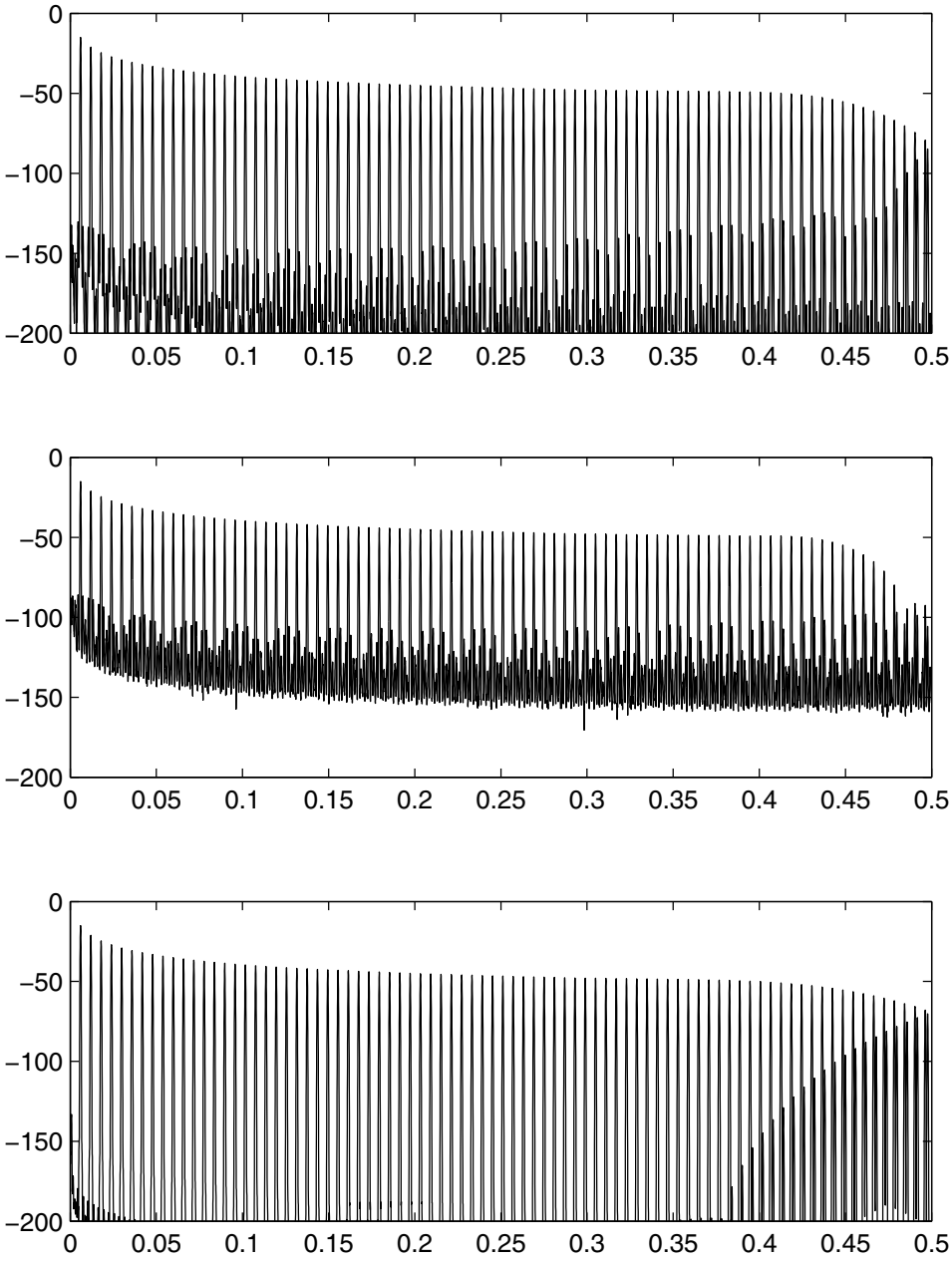


Figure 4.17: Effect of window (i.e., filter design) on BLIT-SWS noise floor. 32-zero crossings, 1024 samples per zero crossing, corner pulled back to $0.9f_s/2$. Top: Blackmann-Harris window, Middle: Kaiser Window, $\beta = 3.5$, Bottom: Kaiser Window, $\beta = 20$ (note slower rolloff of the aliasing components due to the large β).

subsample impulse offsets must be rounded to the nearest subphase of the polyphase bank, and hence some time-rounding aliasing artifacts will creep back in (though attenuated). Instead, some interpolation between polyphase outputs should be used to approximate a smooth continuum of subsample offsets for the impulses. Note that interpolating polyphase outputs can be seen to be equivalent to interpolated lookup into the windowed sinc table (this fact is partially described in [13], from an asynchronous sample rate conversion standpoint). Also, when implemented with a polyphase structure, the effective number of points in the table tends to be more restricted by efficiency and memory concerns (see the discussion of cost Section 4.4.3), and so can end up implementing lower-quality BLITs.

Relation of BLIT-SWS to Existing Synthesis Methods

CHANT/Vosim/PSOLA/etc As mentioned in Section 4.2.3 (p. 198), CHANT, Vosim, and the like are methods which play back an impulse-response wave pseudo-periodically as a filter impulse-response to produce (typically) vocal formants. BLIT-SWS uses the same concept: reading out a filter response (this time a bandlimiting filter) at a periodic rate. However, BLIT-SWS implements what was considered too expensive at the time those algorithms were implemented: subsample offsets. This allows the virtual impulse train to not be rounded to sample boundaries. Further, by using a highly oversampled table for the filter response, lookups into it do not have to do as much work as they otherwise might (linear interpolation works fine, whereas if the table were less oversampled or not oversampled, much higher interpolation order would be necessary).

PET Periodic-Excitation-Table synthesis [129] is basically equivalent to the above methods, but was applied to implement commuted synthesis of bowed strings, by periodically playing a body impulse response into a string. It was this conceptual framework upon which the author based the BLIT-SWS implementations.

Granular Implementations as in Algorithm 3 (as opposed to polyphase implementation) are also very similar philosophically to implementations of Granular synthesis systems, except with completely deterministic grain-generation timings, and the sub-sample offset capability (which is not always implemented in Granular systems, as the randomness in most of their implementations can help break up the effects of time-rounding in a dithering-like manner).

Phase Format Note that one shortcoming of this algorithm is that the calculation of the subsample location of the impulse is most inexpensive if the phase of the oscillator is in units of samples, and therefore the frequency is defined by its period rather than its frequency. As such, a divide is necessary to compute the period if the frequency is the known value. Note that if the frequency is itself being derived from some other information (like MIDI note number), then the

transformation to frequency can often be replaced by a transformation to period with no additional cost. Still, if one is modulating the frequency (either at low rates as with an LFO or at audio rates), one is typically expecting that the modulation be either to the linear frequency (or to the exponential frequency), not to the period. As such, the divide creeps in again. Small modulations may be fine in the period domain, however.

Of course, one can keep the phase in radians (or scaled radians), such that the frequency input is linear, in which case the divide gets moved to the calculation of the sub-sample impulse location: The subsample offset x from sample $n-1$ of the intersection of the phase line between p_{n-1} and p_n and the threshold phase p_0 is given by:

$$x = \frac{p_0 - p_{n-1}}{p_n - p_{n-1}} \tag{4.21}$$

Though this calculation is more complicated than if phase were in radians (in which case the calculation of the subsample offset is simply a subtraction), it need only be calculated once per period, so it should happen much less frequently.

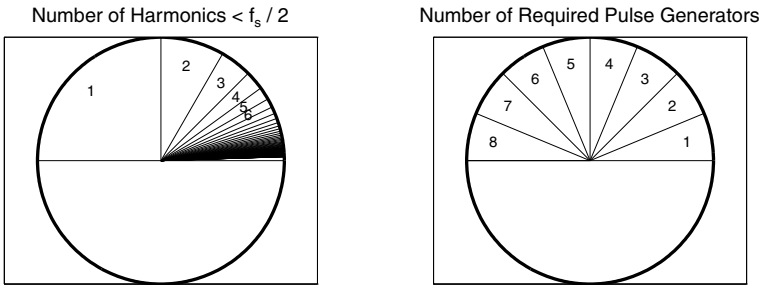


Figure 4.18: Comparing number of harmonics to number of overlapped pulse instances for a pulse 8 samples long.

Number of Harmonics vs. Sinc Overlap A further optimization comes from comparing the number of sincs that must be overlapped in the BLIT-SWS method to the number of harmonics of the BLIT that land below $f_s/2$. At very high frequencies, the number of bandlimited harmonics becomes quite small. Indeed, in the top octave, only the fundamental is in band. Thus for a large percentage of the frequency range, it is quite likely that it may be more efficient to generate a BLIT (or any other harmonic waveform) by simple summation of sines. At lower frequencies, we can again revert to the BLIT-SWS method because it is obviously more efficient at low frequencies, where the number of harmonics is very large. the author has not prototyped such a system, and it is not known if this has been done by any one else.

Note that the variable cost puts this algorithm at a different point in the cost-vs-aliasing tradeoff

curve than, say, the DPW algorithms. This algorithm attempts to strongly reduce aliasing at the cost of a more expensive algorithm, which unfortunately can get quite expensive at very high pitches. This fact can make this algorithm less attractive.

In [276], a variation on the BLIT algorithm was implemented which used a polyphase filter to implement the BLIT. This makes the filter a constant-cost filter, though the constant cost is a mid-range cost (higher than the low-frequency cost of the method described above, but lower than its maximum cost). However, because the size and cost of a polyphase implementation increases with the number of branches (equivalent to the number of samples per zero crossing in the sinc table of the method described above), it is normally limited to a rather small number ([276] used 32, whereas the author's implementations of the variable-cost algorithm typically use the equivalent of 256-1024), and as such the noise floor is raised significantly. This becomes a particular problem when these BLITs are integrated, as the raised noise floor is boosted even further at the low frequencies by the integration. When using polyphase implementations, the arguments for BLEP methods (see Section 4.5) become even stronger, as implied in [276].

Precisely where the tradeoff occurs depends on the system in which the algorithm is to be implemented. For an extreme example, in an IFFT system the tradeoff frequency may move all the way down to $f_s/(2N)$, because the system implements sums sines so efficiently. On the other hand, in a system where sine generation is significantly more expensive (say in a system where memory accesses are expensive enough to make even table lookup mildly expensive), the tradeoff frequency can easily be on the order of $f_s/8$. Of course, the amount of accuracy desired in the BLIT-SWS algorithm also affects the tradeoff, because it will affect the choice of the window length, thus affecting the number overlapping sincs at a given frequency.

Harmonic (Aliasing) Fall-Off Rate vs. Number of Zero Crossings Because the windowing imposes a finite fall-off rate in the harmonics, some aliasing is inevitable. We can, however, control this by our choice of window, and by carefully choosing the corner frequency of the virtual sinc function. We will see below the effect of choosing various window widths on the sharpness of the filter rolloff.

But first, a side note on the birth and death of harmonics during frequency slews or modulations.

In practice, the BLIT-SWS method has demonstrated another advantage over other, more exact bandlimited BLIT algorithms. In cases where the frequency is sweeping, such as vibrato or portamento, high harmonics of the signal will disappear or appear (depending on the direction of the frequency sweep) during the sweep. In an exactly bandlimited system, the highest harmonic transitions between full amplitude and zero amplitude (or vice versa) in the period of one sample. This causes an audible transient, especially at low sampling rates, where $f_s/2$ is well within the audible region. These transients are usually unwanted and distracting.

Like additive synthesis and bandlimited wavetable synthesis, and unlike DSF, in BLIT-SWS synthesis the highest harmonic need not audibly "pop" in or out as it comes down from or gets up to half the sampling rate, since the window function can be chosen to exhibit any amount of attenuation at $f_s/2$.

This effect has not been a historical problem in recorded digital music because of the use of non-ideal anti-aliasing filters. The finite fall-off rate of these filters allows harmonics to die out more slowly in upward sweeps (and appear more slowly in downward sweep), which avoids the above-mentioned transients. The BLIT-SWS method ends up implementing slower fall-off rates as an artifact of the windowing, and therefore gets this effect for free.

Some of the above-mentioned BLIT generation methods cannot easily implement a slower rolloff. SincM and DSF, for example, have control only over the existence (or lack thereof) of harmonics via N, so can only implement abrupt transitions in the number of active harmonics. DSF can implement a harmonic fall-off via the parameter a, but this fall-off must start at the first harmonic and increase through all the harmonics, rather than beginning only near $f_s/2$. A few hacks can be used to overcome this. For example, a second DSF group of harmonics can be placed at the top few harmonics with a small a to implement the fall-off rate. Another possibility is to calculate two oscillators with differing numbers of harmonics and interpolated between the oscillators to allow the highest harmonics ramp to zero before they are removed. Another hack would place a post-filter after the DSF to implement the high-frequency fall-off. This hack may have problems handling the harmonics' on-off transients, however.

Example Spectra Figure 4.19 shows the spectrum of a rounded-time impulse train and of a frequency sweep up across the whole frequency range. In this algorithm, unit samples are put out at the sample time nearest the ideal time, and there is massive aliasing.

Figure 4.20 shows the spectrum of a discrete-time impulse train using linear interpolation to interpolate the unit sample pulse locations. The aliasing is reduced, but it is still very strong.

Figure 4.21 shows the spectrum of a bandlimited impulse train generated using the BLIT-SWS method with 8 sinc zero-crossings under a Blackman window. While there is still considerable aliasing at high frequencies, at low frequencies it is down 90 dB or so.

Figure 4.22 shows the same thing as Fig. 4.21 with the number of zero crossings raised from 8 to 16. This roughly halves the transition bandwidth of the window transform, and as a result, the aliasing is down 90 dB over approximately 60% of the spectrum.

Figure 4.23 shows the same thing again with the number of zero crossings doubled again from 16 to 32. Again the transition width is halved, and now only the upper 20% of the spectrum is heavily aliased.

Fig. 4.24 shows the previous case (32 zero crossings) with the cut-off frequency of the sinc function lowered below half the sampling rate. This means the transition band of the window transform

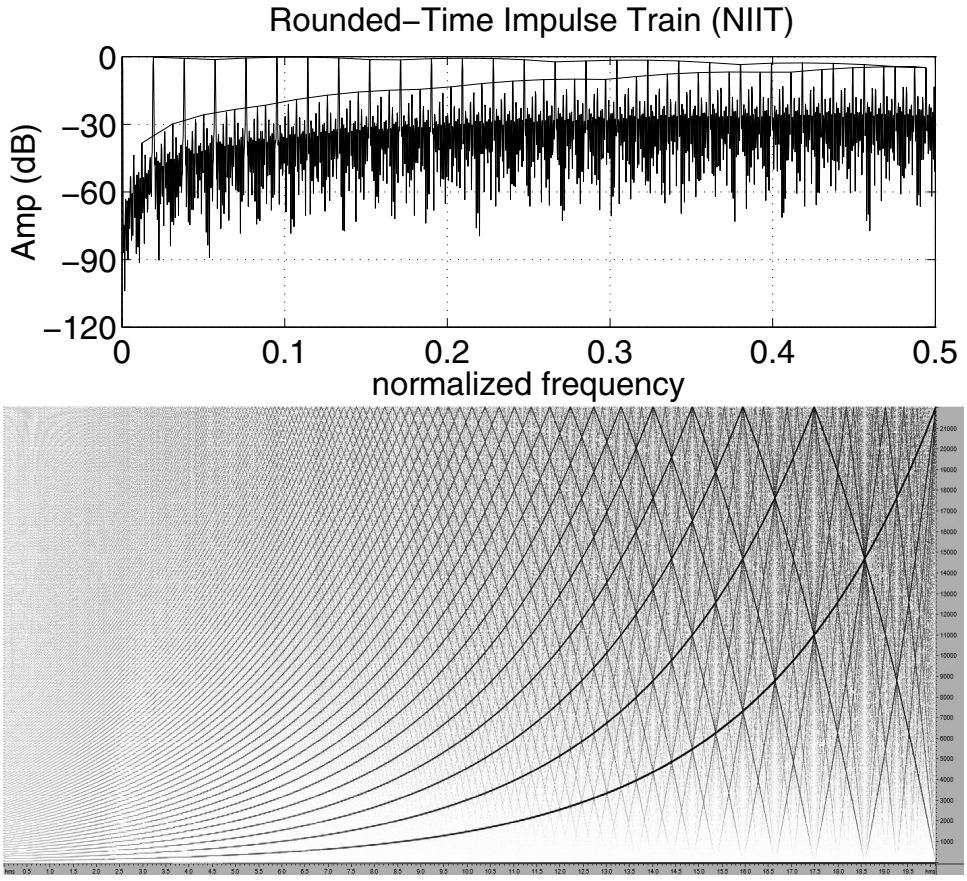


Figure 4.19: Top: Spectrum of rounded-time impulse train with a line drawn connecting the peaks of both the desired harmonics and the first string of aliased harmonics ("NIIT" stands for "Nearest-Integer Impulse Train"). Bottom: Spectrogram of a logarithmic frequency sweep.

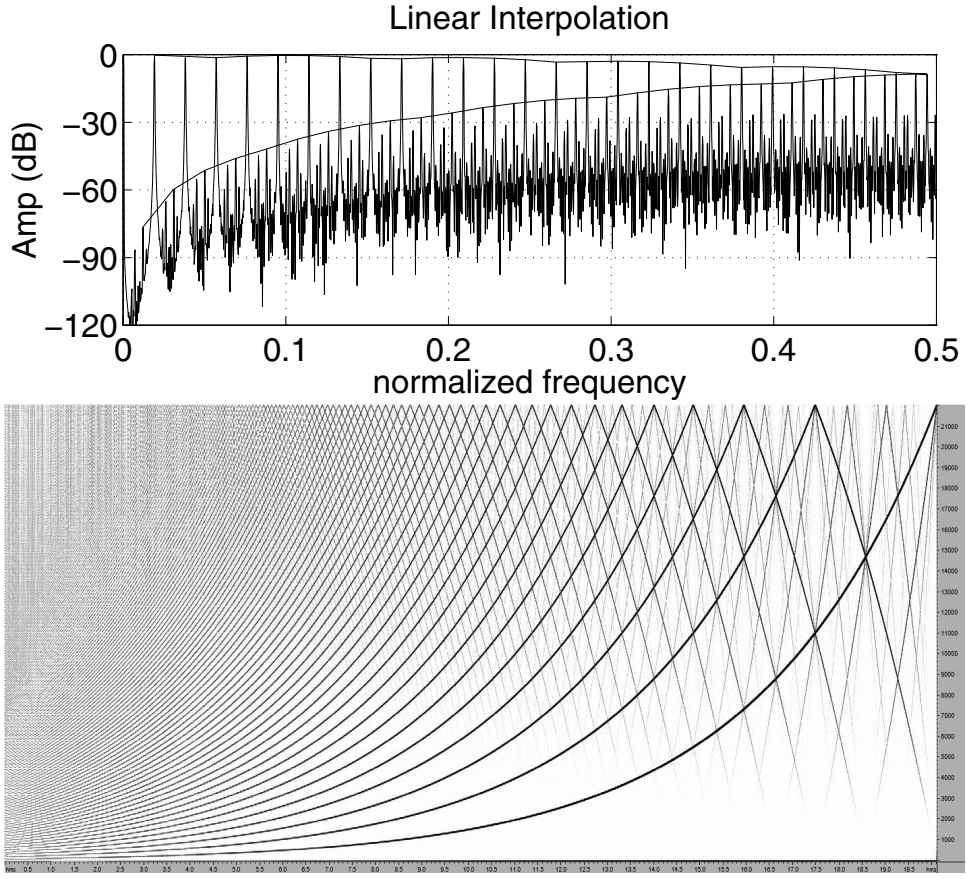


Figure 4.20: Top: Spectrum of linear-interpolation impulse train with a line drawn connecting the peaks of both the desired harmonics and the first string of aliased harmonics. Bottom: Spectrogram of a logarithmic frequency sweep.

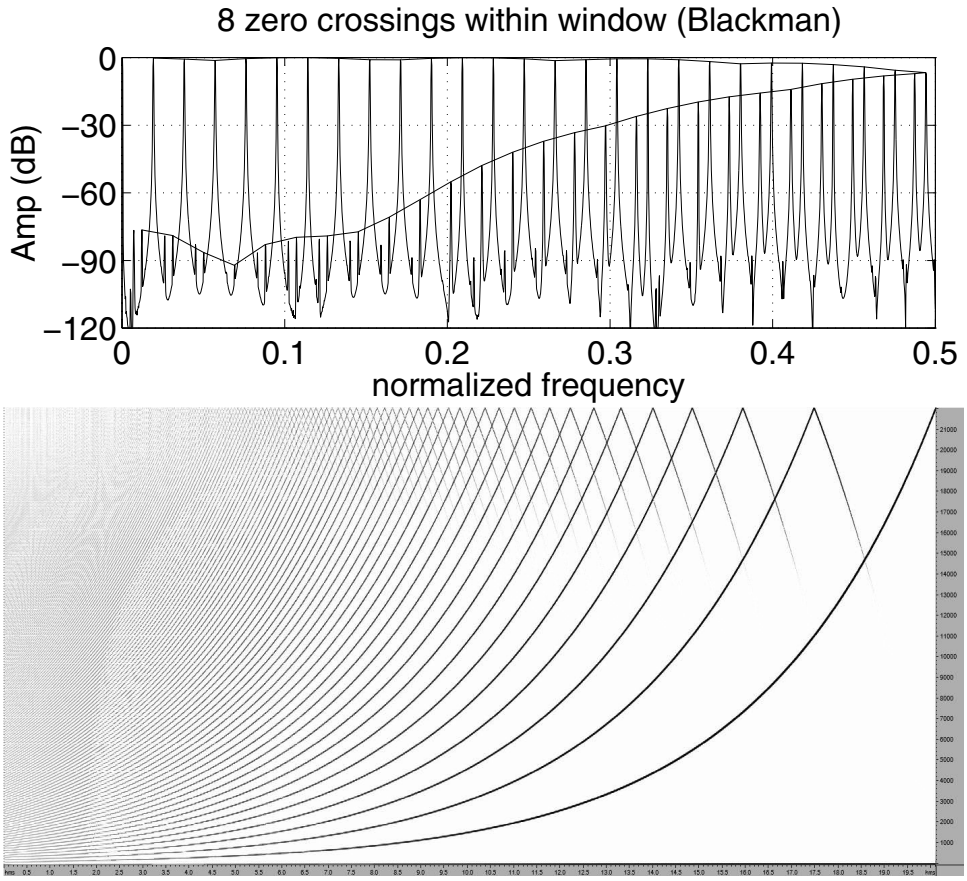


Figure 4.21: Top: Spectrum of windowed-sinc interpolated, window 8 sinc zero-crossings wide. Bottom: Spectrogram of a logarithmic frequency sweep.

is folded in half as it falls into half the sampling rate and reflects. The result is another halving of the aliased region to about 10% of the highest frequencies. If the limit of human hearing is 20 kHz, this means we need a 2 kHz guard band, so the sampling rate should be at least 44 kHz.

Finally, Figure 4.25 Shows the behaviors for the SincM algorithm and for bank-of-wavetables algorithm with 8 octave-spaced wavetables, and interpolation between adjacent tables as the fundamental frequency moves through the octaves.[9]

[9]In practice, the author has heard (in an octave-spaced impulse-train bank at 44.1 kHz) clear variations in brightness across long-distance sweeps as the amount of energy in the top octave varied with the fundamental-frequency's location within each octave (see Figure 4.25 and note the variation). As such, finer spacings than octave spacing may be required in some implementations. This of course would be most obvious in an impulse-train wave bank, and as such may be less of an issue in general.

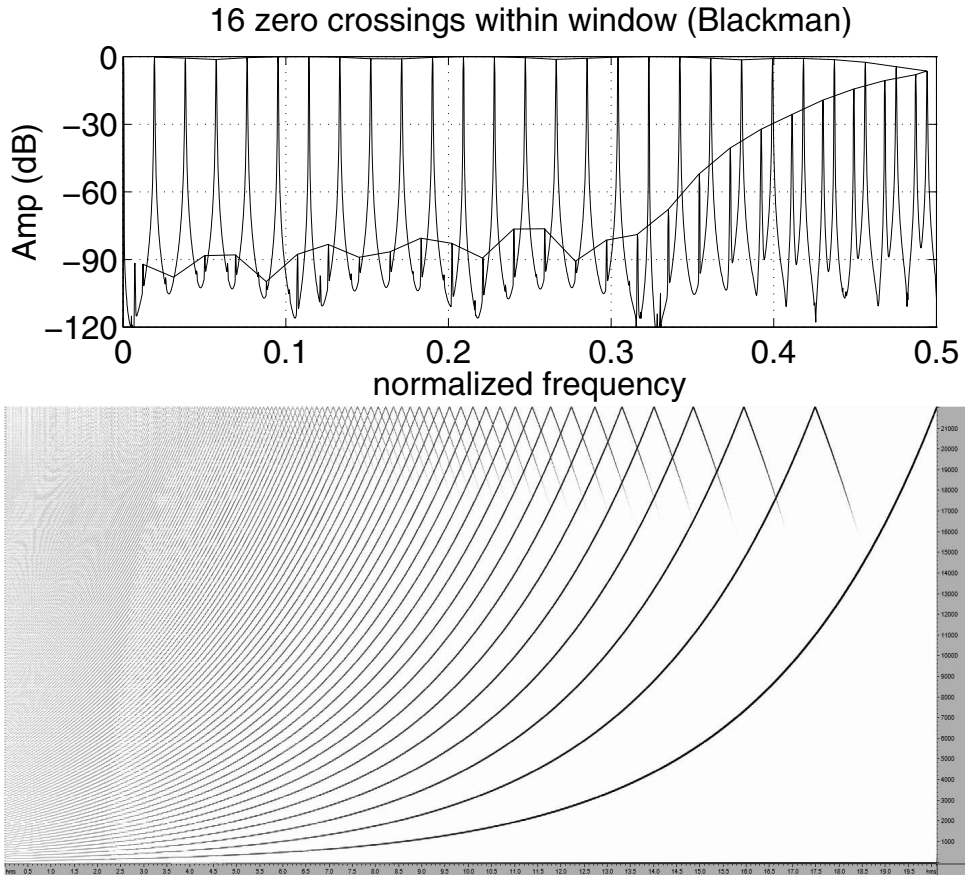


Figure 4.22: Top: Spectrum of windowed-sinc interpolated, window 16 sinc zero-crossings wide. Bottom: Spectrogram of a logarithmic frequency sweep.

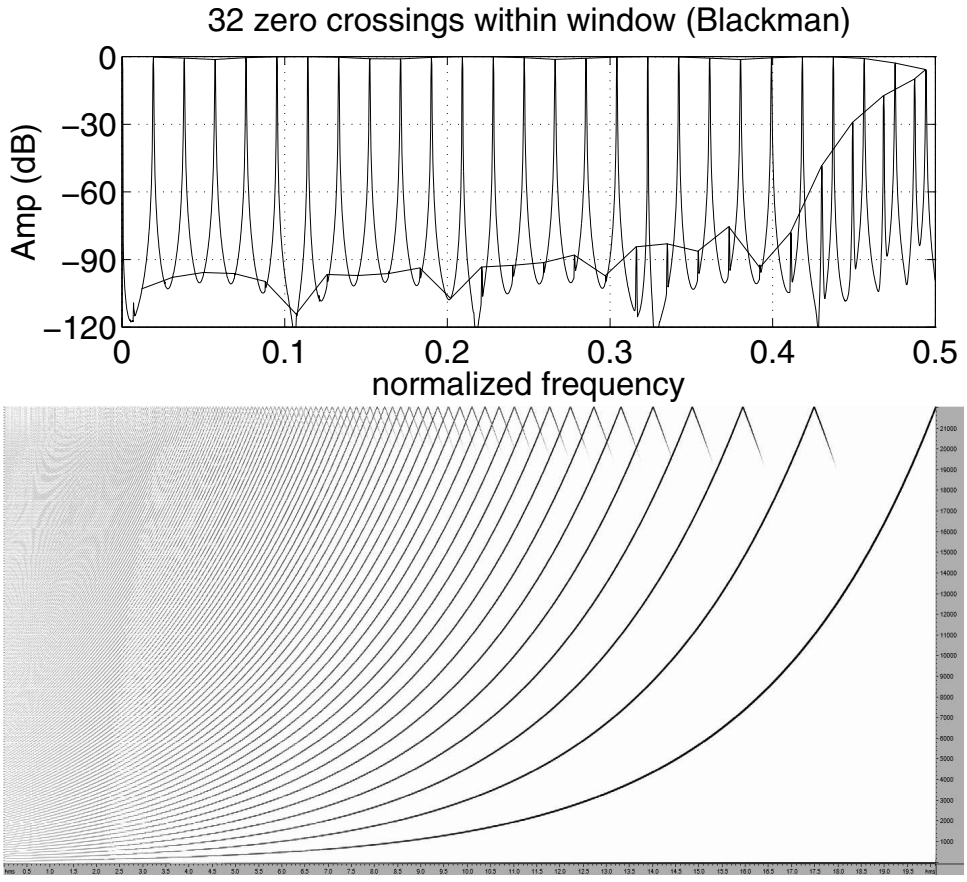


Figure 4.23: Top: Spectrum of windowed-sinc interpolated, window 32 sinc zero-crossings wide. Bottom: Spectrogram of a logarithmic frequency sweep.

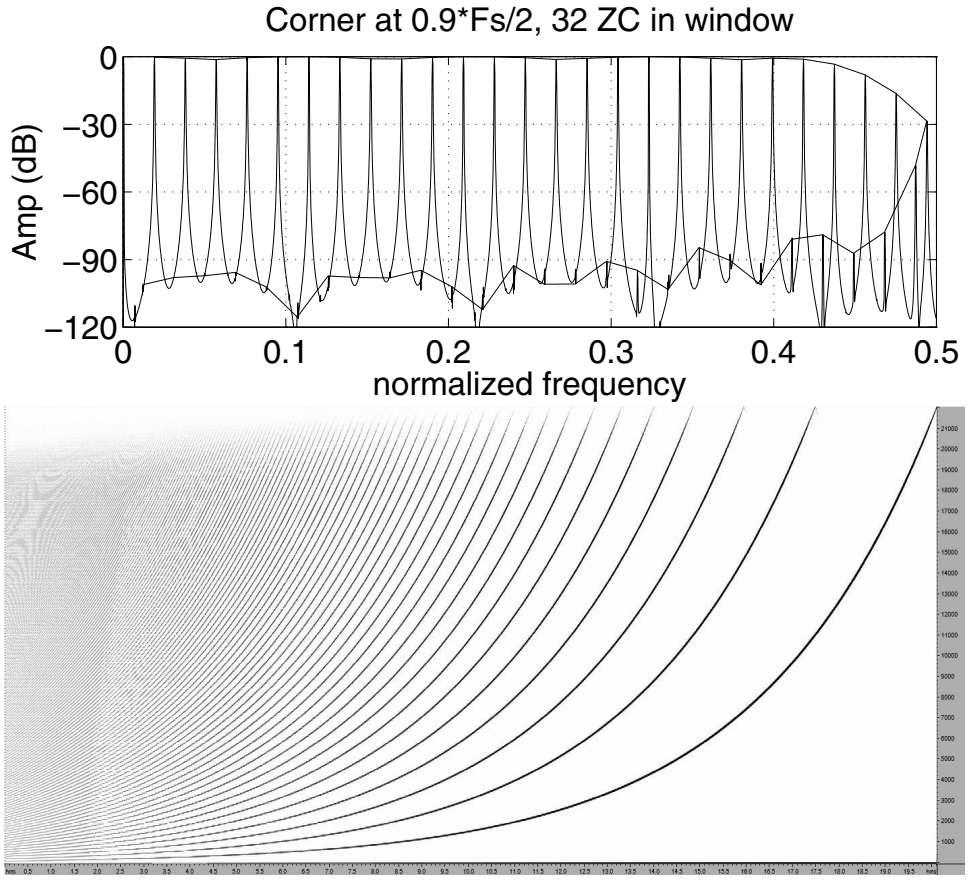


Figure 4.24: Top: Spectrum of windowed-sinc interpolated, window 32 sinc zero-crossings wide, sinc function dilated to lower cutoff frequency to $0.9fs/2$. Bottom: Spectrogram of a logarithmic frequency sweep.

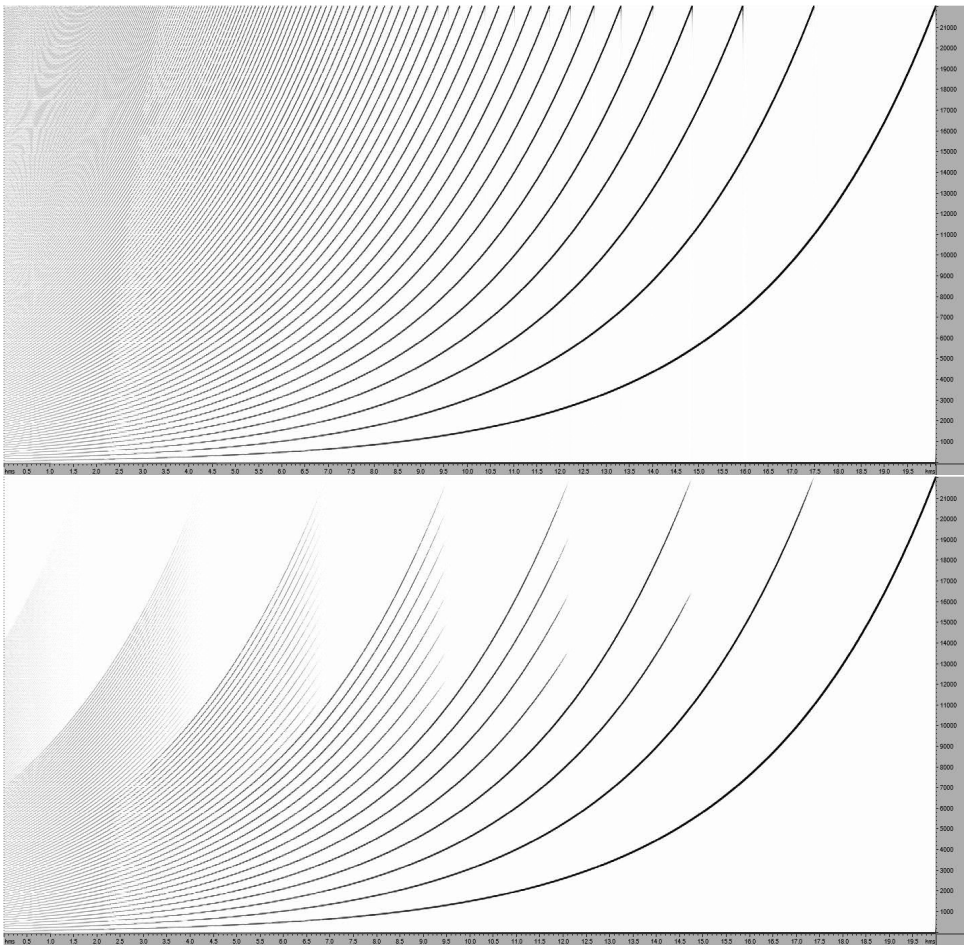


Figure 4.25: Spectrograms of logarithmic frequency sweeps. Top: SincM: Note clicks when harmonics disappear. Bottom: Octave-spaced bank of wavetables, linear interpolation between tables across top octave.

4.4.4 Modulatability

As discussed in the introduction to "steady-state" algorithms (Section 4.2.5), such algorithms implicitly assume that at any point in time, the current settings of the oscillator are constant, infinitely forward and backward in time. For an impulse train, this means that such algorithms assume an infinite number of impulses spaced at the current period with the current amplitude, on either side of the current point in time. Since the ideal bandlimiting sinc which these methods assume is also infinite in time, that means that the value of the oscillator at the current point in time is theoretically affected by the location and amplitude of each of the infinite number of impulses, even those a large distance away. Now, in practice, the effect becomes negligible beyond some distance in time (beyond some number of zero crossings of the bandlimiting sinc function), still in high-frequency situations, where the period is short enough for there to be significant contributions from multiple neighboring impulses, the implications of the steady-state assumption can effect the modulatability of the oscillator.

Let's consider for a moment a worst-case situation. Figure 4.26 shows a simulation of sudden transition in a pulse train from a period of 9 to a period of 21. The top plot shows the "theoretically correct" version of the transition, consisting of the sum of a very large number of sincs spaced by 9 on the left and spaced by 21 on the right. The middle plot shows the transition using $Sinc_M$. The problem is that the $Sinc_M$ wave was about to produce another pulse, and so was ramping the "ringing" up for the pulse (this "predicting the future" is part of the behavior of a steady-state waveform, since in steady state, the pulse *will* arrive). However, the system was changed, and the pulse in fact did not happen, so that the steady-state method has errors around the transition. A non-steady-state method is shown in the bottom plot (a BLIT-SWS-style algorithm using triangle-windowed sincs with a window width of 20). The non-steady-state algorithm trades of inaccuracies in the sinc shape (i.e., a finite rolloff rate for its harmonics) against placing pulses precisely and not having "prediction errors". Remember that a BLIT-SWS algorithm actually achieves causality by operating at a delay of half the window width.[10]

A similar effect can be seen in the modulation of amplitude (Figure 4.27). Amplitude modulation as a post-process can simply "clamp" off the ringing of an impulse. As a second example, the sudden change in Figure 4.26 could easily have been an amplitude transition — the fact that the steady-state method was ramping up to a pulse that will actually be different would still be a source of artifacts.

Now consider that such changes are happening every sample due to audio-rate modulation. Of

[10] which may suggest an unexplored direction for a steady-state method: operate at a delay, and right after producing the center of a pulse, check to see if the next pulse will arrive on time (or even early), and ramp the phase increment towards the coming change. By placing the ramping right after pulses, the effect of misprediction might be minimized.

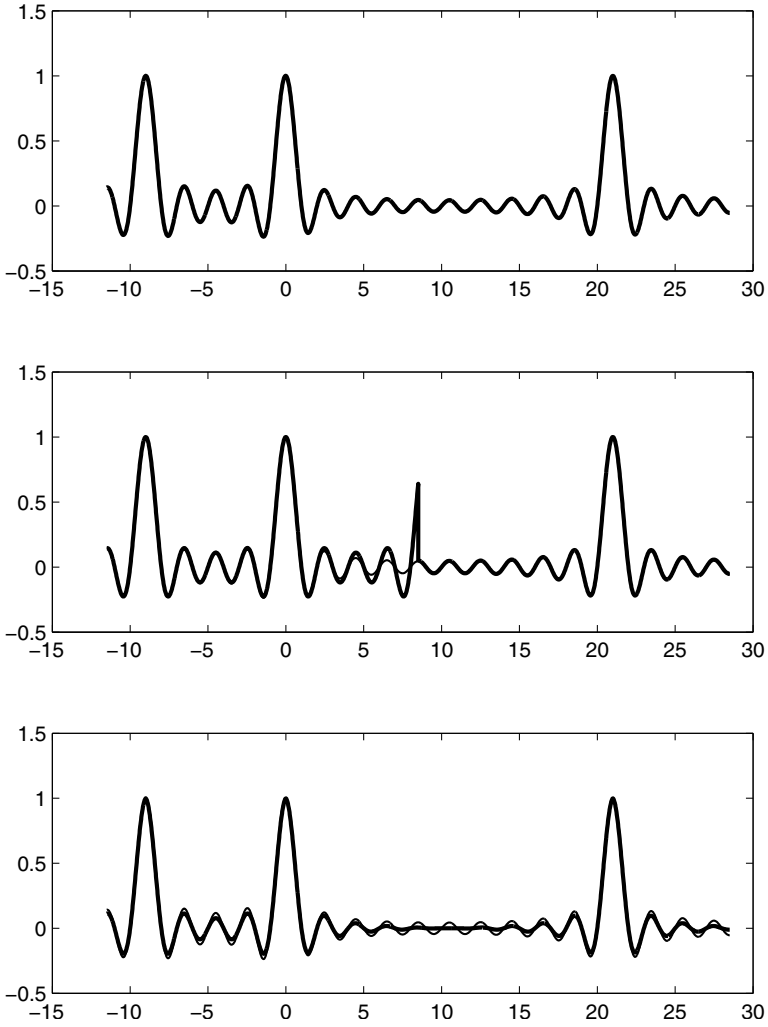


Figure 4.26: A particularly bad modulation for a steady-state waveform: a sudden transition from a period of 9 to 21. Top: Theoretical sum of actual sincs (approximated by 1000 sincs on each side at the correct locations), Middle: $Sinc_M$ (thin line is the theoretical curve of the top graph), Bottom: Sum of triangle-windowed sincs with a width of 20.

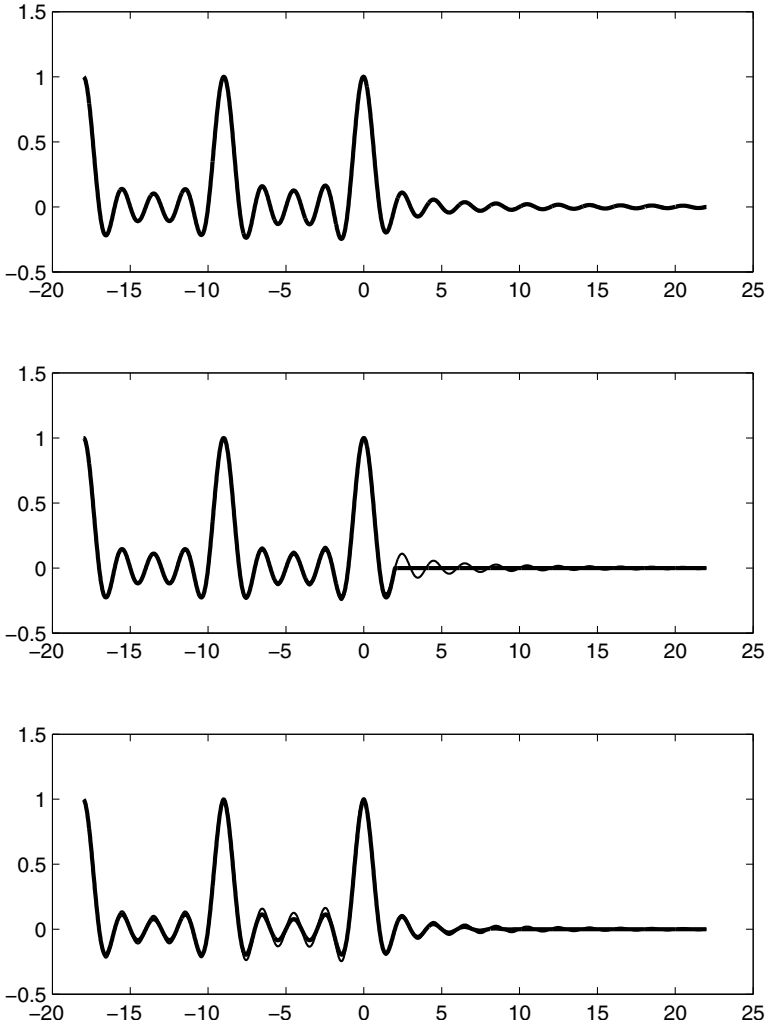


Figure 4.27: A sudden amplitude transition from 1 to 0. Top: Theoretical sum of actual sincs (approximated by 1000 sincs on each side at the correct amplitudes), Middle: $Sinc_M$, post-modulated (thin line is the theoretical curve of the top graph), Bottom: Sum of triangle-windowed sincs.

course, the changes are likely to be much less drastic than these worst-case examples, but the problem remains: modulating a steady-state algorithm results in modulating the steady-state waveform's bandlimited signal, whereas modulating a non-steady-state algorithm bandlimits the modulated ideal signal. In other words, the bandlimiting and modulation operations can be transposed in a non-steady-state algorithm, whereas they cannot be with a steady-state algorithm.

Therefore, consider a case where a high-frequency impulse train is being synthesized (i.e., a wide harmonic spacing), and the highest harmonic is very close to $f_s/2$, and therefore the next harmonic of the theoretical signal would be well beyond $f_s/2$. Now we amplitude modulate this situation with a sinusoidal modulator which sweeps upward in frequency. In the continuous-time case, each harmonic would develop sidebands whose spacing was the pitch of the modulator. As such, since the next higher harmonic is well beyond $f_s/2$, we would expect that its lower sideband would not extend below $f_s/2$ until the modulator reached a particularly high frequency. The upper sideband of the harmonic just below $f_s/2$ would move up past $f_s/2$ of course. An ideal bandlimiting of this modulated system would not reflect the upper sideband of the harmonic just below $f_s/2$, and would only exhibit frequencies coming downward through $f_s/2$ when the modulation frequency got high enough for the lower sidebands of the next harmonic to reach past $f_s/2$.

However, a discrete-time post-modulation of a bandlimited oscillator (i.e. multiplication by the output of the oscillator, which is also the only possible modulation in a steady-state method) would result in the upper sidebands of the harmonic just below $f_s/2$ reflecting off of $f_s/2$, and thus exhibiting an aliasing artifact, even if the original signal were bandlimited, since in discrete time, the output of the oscillator will contain a mirror-image harmonic just above $f_s/2$, whose lower modulation sidebands will cross below $f_s/2$ at a rather low modulation frequency.

On the other hand, a non-steady-state method can actually modulate the pulse heights before applying the bandlimiting, and hence produce a bandlimited version of the theoretical signal (within the limits of the rolloff rates of the bandlimiting approximation used by the method).

There is a rather large caveat to this point for amplitude modulation: the modulating signal must be calculated at the impulse locations rather than rounded to the sample in which the impulses occur. Unfortunately, if the modulation is being calculated externally from the oscillator, this means that the oscillator will have to perform some sort of interpolation on the modulating signal to estimate its value at the calculated impulse instant. Such interpolations may have imprecisions which decrease the fidelity of the resulting signal (i.e., the noise floor can rise significantly if the modulation signal is badly interpolated). Compare Figure 4.29 with the lower spectrum in Figure 4.28, to see the effect of not interpolating the sinusoidal amplitude modulating signal. Note that even the interpolated version does get some artifacting from the low order of the interpolation (which would of course get worse at higher modulation signal frequencies).

In summary, since non-steady-state BLIT methods such as BLIT-SWS actually calculate using estimates of the pulse heights and locations (at a higher precision than the nearest sample), they

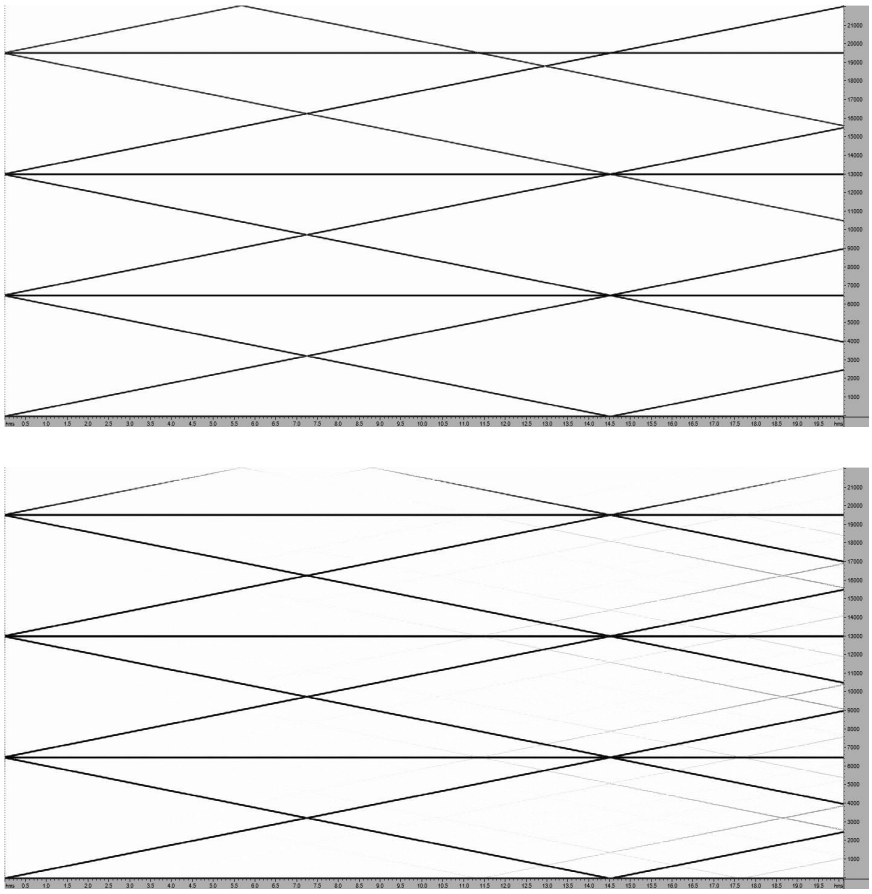


Figure 4.28: Spectrograms of amplitude modulation of a 6500 Hz impulse train with a sweeping modulation frequency (f_s = 44.1kHz). Comparing amplitude modulation as a post-process (i.e., what must be done with steady-state methods), top, versus modulating the individual pulse amplitudes in a BLIT-SWS method, bottom. Note how in the bottom spectrum the upper harmonic of the top harmonic does not bounce off $f_2/2$ with significant energy, but rather the lower sideband of the next (out of band) harmonic comes into the band, whereas in the top spectrum the upper sideband of the top harmonic bounces off $f_s/2$ and no sidebands of out-of-band harmonics come into band.

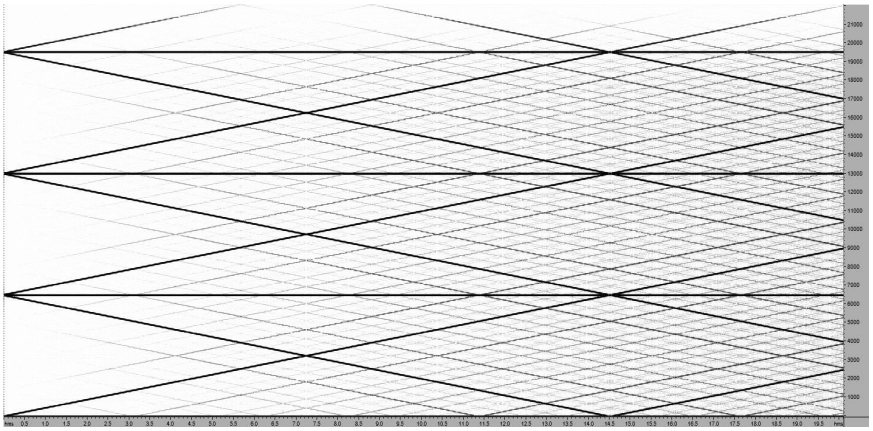


Figure 4.29: The effect of not interpolating the modulating signal to the subsample pulse locations. Note the increased noise floor.

can theoretically bandlimit any modulation which simply moves the pulse locations or changes the pulse heights, as the methods make no assumptions or predictions about the pulse locations (or heights), simply taking them as they come and bandlimiting them.

As such, popular techniques such as hard sync (resetting the phase based on the reset of another oscillator), which when applied to an impulse train simply move the pulse locations, can be synthesized without aliasing [250]. However, when integrated, this method cannot trivially produce exact sync for the other wave shapes, since in those situations sync implies resetting the integrators, which can't trivially be done in a bandlimited fashion.[11] On the other hand, as was mentioned in the discussion of leaky integrators, in many situations where a signal is in the audio range, the exact waveform shape is not actually important, just the overall spectral shape. In such cases, it may be sufficient to perform the sync on the the impulse train and feed that into the integration system as normal.

On the other hand, there are extensions to the integrated-BLIT concept which have appeared recently which can overcome a few of these shortcomings.

4.5 Beyond BLIT

It can be noted that one of the shortcomings of integrating BLITs to get sawtooths and rectangle waves is that the noise floor near DC is boosted by the integration [276]. Further, multiple integrations in series become increasingly sensitive to DC offsets. We have discussed the use of leaky

[11] Although it should be possible to calculate a good approximation of a negative pulse to feed into the integrator to very nearly zero out the state, either based on information about the last reset time (trickier when leaky integrators are being used, and even worse when time-varying leakiness is used), or based on the integrator state (though that would be a little less accurate as the integrator state would also contain "ringing" from previous events which should not be zeroed out)

integrations, and even time-varying leakiness, but in general, an argument can be made along the lines of "the fewer integrations the better". As such, a few authors have worked on directly synthesizing the integrated bandlimited pulse rather than integrating it at runtime. Therefore, not only can any noise-floor and DC-offset issues be reduced, but the wave-shape issues discussed above for hard sync can be reduced.

4.5.1 BLEP, MinBLEP and PolyBLEP

The first work on bandlimited step synthesis (or integrated-pulse synthesis) was published by Brandt in 2001 [27] ([276] describes the usage of the algorithm and some of its issues in more detail). The primary concept, as noted, is to remove integrations in the algorithm by directly synthesizing bandlimited steps rather than impulses. Whereas in BLIT, an oversampled and windowed sinc function is precalculated, in BLEP, this function in integrated at the precalculation stage. This is then subtracted from an ideal step to get a waveform that has a limited range in time (a step is unlimited, since it never comes "back" to zero). Thus a bandlimited step can be generated by adding appropriately-offset samples of the subtracted wave to a trivially-integrated step. In other words: integrate as normal (in a time-rounded fashion), and add in the subsample-shifted "ringing" function to fixup the time rounding in the step integration.

As noted in [276], this can reduce the noise-floor boost which occurs when integrating BLITs. Further, as was the topic of Brandt's paper, this method allows the direct implementation of hard sync in sawtooth and rectangle oscillators while maintaining the correct waveform shape (whereas BLIT, as mentioned above, can only implement sync with correct wave shape for impulse trains).

For some, the bulk delay of BLIT/BLEP methods is disturbing (they must delay the impulses by half the window length in order to give the symmetric windows time to "ramp up"). As such, Brandt also derives "minBLEP", which is a minimum-phase version of the BLEP, which moves the energy as far forward in time as possible. However, it is noted in [276] that this is not completely successful, and that the calculation of the alignment of the minBLEP with the rounded-time step can actually become more complex.

In [276], Välimäki and Huovilainen discuss replacing the bandlimiting filter used to bandlimit the step function with a polynomial approximation. They start with the triangle window, which when integrated and turned into BLEP form gives a system of two 2^{nd}-order polynomials as opposed to the precalculated filter responses used earlier. They name this method "PolyBLEP", and note that for mid-quality work, it may be sufficient, and can be significantly cheaper.

4.6 Summary

In this chapter, we have explored methods of generating bandlimited (reduced aliasing) waveforms for use in virtual analog subtractive synthesis, concentrating on inexpensive, modulatable

oscillators and trying to balance that against aliasing-reduction and other quality factors. We described a method for deriving the standard virtual-analog waves by linear operations from an impulse train, and explored methods for generating bandlimited impulse trains in particular, including the BLIT-SWS technique, which the author described in [248]. We noted more recent developments to these algorithms, particularly the BLEP, which reduces the number integrations from the BLIT-based methods, and its recent variations. We also noted the usefulness of other algorithms, including crossfading amongst pre-filtered wavetables (now being known by the term "MIP-Mapping" in relation to the graphical algorithm of that name), and aliasing-reduction methods like the Differentiated Parabolic Wave method.

As in the exploration of filtering, several methods are of use, and each designer's situation and personal aesthetics will govern which methods they consider most applicable to their design.

Appendix A

Root Locus Review and Rendering Methods

A.1 Introduction

As discussed in the introduction to Chapter 2, the root loci of a variable filter along its various control axes can give insight into the behaviors of the filter's variation. The concept of a root locus is one we borrow from the field of classical control-systems design. Evans first described a method for graphically analyzing [73] and designing [74] standard plant/controller/gain feedback loops via a set of straightforward rules on how to sketch the trajectories of the total system poles as the feedback gain varied between zero and (usually) some point where the system had gone unstable. The primary goals at the time where (1) to find at what gain the system would go instable or would pass through various contours describing desired behaviors (overshoot, reaction times, etc.), (2) to give intuition on adding further controller dynamics to change the gain at instability and to shift the dynamics (i.e., the dominant pole tracks) to more desirable behaviors.

Through the decades of its most prominent use, root-locus design was primarily a sketching method (possibly with the assistance of a spirule, a device invented by Evans specifically for that purpose [72]), though some analytic methods did crop up, as we will see later. As computing power became more accessible, root loci could finally be drawn numerically and accurately, though unfortunately, the world of control-systems was already starting to move away from root locus towards newer methods which gave the designer even more power to generate more optimal controllers. As such, root-locus became relegated to a teaching tool in introductory classes, though some designers would fall back to it now and then for quick analyses.

During this time a number of algorithms for drawing root loci on a computer were developed, from the most basic brute-force methods through more elegant techniques making use of various

properties of the locus equations. From a graphical algorithms perspective, there is actually quite an interesting variety among the possible ways that loci can be drawn. Different methods are based on different combinations of the locus-equation properties, and so some can be more applicable than others for drawing loci of systems that are slightly outside the traditional mould in some way or another.

In this chapter we will first give an overview of the literature on root-locus, then review basic concepts of the standard locus (first-order in k) and higher-order loci. We next will discuss a short taxonomy of the types of systems we may try to draw loci of. That will be followed by some explorations into directly deriving a few systems to have certain desired root-locus shapes. Finally, we will look at the various ways one can draw root-loci, including several techniques which, while not necessarily useful at the technical level, produce particularly interesting graphical results.

A.1.1 Root Locus Overview

Literature Review

Root-Locus literature can be considered to start with Evans' two papers in 1948 and 1950 [73], [74]. The first introduced the concept for analysis of control-system behavior, and the second described its uses in designing controllers. As noted in [72], Evans would eventually write a textbook on the subject [75], but due to delays, several textbooks describing the method had already been written. From there, it was mainly in the domain of textbooks to describe the method. These days, well-known textbooks such as Franklin and Powell [88], Dorf and Bishop [66], or D'Azzo Et Al [59] are the standard references for many students. These days, it is rare to see a book purely on root-locus, but back in its hey-day such books could be found, such as Dransfield and Haber [67].

Not Algebraic Geometry Although on the forms of the root locus equation, such as

$$1 + kG(s)H(s) = 0$$

or

$$D(s) + kN(s) = 0,$$

are of the basic type explored in the field of Algebraic Geometry:

$$F(x, y, \cdots) = 0,$$

the two fields look at strongly different issues. Root-locus is a very practical, primarily numerical area, whereas algebraic geometry is interested in much more abstract properties of such equations and their solutions. An analogy may be to say that root locus is to algebraic geometry as the quadratic equation is to modern algebra. Still, although we will not look into the area in this thesis,

algebraic geometry is quite an interesting field. Some classic texts are Griffiths and Harris [100], Shafarevich [230] [231], Cox Little and O'Shea [51], and Harris [105]. Slightly closer to the interests of this thesis are texts which concentrate more directly on the properties of curves and surfaces, such as Kirwan [142], Bruce and Giblin [29], and McLeod and Baart [164]. Still, since the problem of drawing root loci is primarily a numerical one, even these texts are a bit beyond the needs of this discussion, though it is certain that one can glean applicable insights from them.

Teaching and Intuition-Building As root-locus is mostly a teaching technique these days, much of the more interesting recent literature is aimed in that direction. Yang [296] [295] relates the understanding of root locus to electric field potentials and flows, most interestingly describing the addition of poles and zeros as putting forces on the locus. Loci in phases other than π and zero are also discussed. Similarly, Cavicci [35] stresses the understanding of loci at phases other than π, and discusses loci in phase directly (rather than gain) to help visualize phase margin. Further, Lundberg [160] recommends the plotting of the vector field of the phase of the transfer function as a way of building intuition on the root locus.

Tsiotras has a recent educational paper [271] giving a good overview of the geometry of the locus in the plane, how the locus relates to the magnitude and phase of the open-loop transfer function, and how the locus relates to a 3-D version of the Bode plot. Kurfess and Nagurka wrote a similar earlier (and slightly less engaging) paper [148] , which also discusses 3-D Nyquist diagrams as well as the root locus and 3-D Bode plots.

Basics and Locus Compendia In the early days of root locus as a technique, a few papers were published simply exploring all the possible loci of a particular class of system. A good example is Williams' and Lovering's 1968 paper [290], which is a comprehensive exploration of all four-pole loci. More recently, D. Pierre [203] looked into dipole and doublet loci and robustness issues related to them, and Charles Neuman [190] proved the locus for a particular class of 2-pole, 2-zero systems is a circle and explored it's stability.

Analytic Methods Although Evans proposed the technique as primarily a mixture of intuition and sketching, some authors preferred a more analytic approach to the understanding and drawing of loci. The first such paper was by Bendrikov and Teodorchik in 1959 [15], in which they arrive at some equations which analytically describe the locus. A few later authors refer to their work, and especially those equations ([83] and [207]). At about the same time (the early 1960s), Steiglitz [244] Chang [38] and Krishnan [146] also worked on analytical methods for drawing loci. Later, Spencer and Philipp [243] would apply the work of mathematician Dickson to the root locus and end up with another similar analytic approach.

Loci in Other Axes or Variables From time to time, papers appeared proposing drawing root loci in other axes, such as log frequency [90], etc. Interestingly, Power [205] proposed using a bilinear transform on s as a way of compressing the frequency range of s, as opposed to using log axes for continuous-time RL. Some argue that the dependence of the roots on k is hidden by the locus, and argue for plotting pole amplitude and/or phase vs. k in order to see the gain dependency explicitly [147].

Complementary Loci Evans' original rules only applied to plotting so-called "180-degree" loci (loci in positive k with negative feedback, such that positive feedback occurs at frequencies where the loop transfer function (not including k or the feedback sign) has a phase of 180 degrees). In situations where the plant has a negative overall gain, it turns out that one ends up plotting a zero-degree locus instead, and the rules must be modified a bit. Teixeiria et al [265] discussed how the sketching rules change when attempting to draw a complementary root locus, and Eydgahi [76] discussed certain findings about complementary loci of not-strictly-proper loop transfer functions. Such issues mainly affected rule-based drawing methods, and most numerical and analytic methods do not need changes to draw complementary loci.

Loci of Time-Delay/Continuous-State Systems Evans' method is specifically for rational systems, but there are extensions to certain non-polynomial systems (such as time delays) that have been discussed. Two of the earlier papers on the subject Yeung and Wong [297], and Suh and Bien [255], both in 1982. Typically, analytic or purely numerical methods were employed, as the plotting rules generally did not apply (though, luckily, much of the intuition still applied). Later, Gribble [99] described a locus variation for time-delay systems: plotting various pole properties vs time delay. In 1994 Byrnes, Gilliam and He worked on a rigorous extension of root-locus into distributed-parameter systems [33] (most people up to then who looked into distributed-system RL (like strings), including the author, hadn't worried about being rigorous about this). This continued earlier work of theirs [32] on similar problems.

Other As an interesting side note, Vanecek and company [279] [280] applied root locus to create a chaotic system by defining a linear system that is stable at low and high loop gain, but not in the middle, in feedback with a memoryless nonlinearity whose dynamic gain covers the whole range of stable-unstable-stable gains.

Root Locus Topics We Will Not Discuss

As the primary focus of the discussion is the rendering of root loci of certain single-input, single-output audio filters, certain areas of root-locus research and usage are effectively ignored for this discussion:

- Robust root locus or otherwise analyzing further parameter variations. This is actually a very interesting area from a graphical-algorithms standpoint, but this thesis doesn't look deeply into it. Modern techniques involve using properties of polytopes of polynomials to find the boundaries of regions that represent the robust locus. A particularly interesting paper graphically is Tong and Sinha [269] which demonstrated a significantly better identification of a robust locus using such methods than the typical techniques of randomly sampling the coefficient space. Other literature on this subject includes ([12] [266] [208] [85] [40] [122] [123]).

- RLs directly in more than one control parameter. Even though most filters this thesis looks at do have multiple parameters, they are currently only analyzed one parameter at a time or in some sort of gridding (i.e., brute-force methods).

- Locus equivalents for MIMO (multiple-input-multiple-output) systems and/or loci of eigenvalues of convex combinations of matrices, though [218], which discusses a multivariable extension of the root locus has quite visually-interesting figures.

- Loci at other angles than 0 and π.

- Root-locus issues specifically about control-system design (like k picking, controller synthesis, phase margin, etc).

A.1.2 Review First-Order Locus

The standard root locus, as described by Evans and traditionally used, describes the motion of poles of a system as in Figure A.1. The transfer function of the whole system is

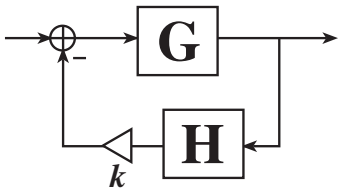


Figure A.1: Linear feedback system drawn in Root-Locus form.

$$\frac{G(s)}{1 + kG(s)H(s)} \tag{A.1}$$

Hence the root-locus drawing problem is one of finding the roots of

$$1 + kG(s)H(s) = 0 \tag{A.2}$$

for all k of interest.[1] Typically, this will be re-written as follows: assume that $G(s)H(s)$ is a rational function which can be written as $N(s)/D(s)$, where $N(s)$ and $D(s)$ are polynomials. As such, the root-locus equation becomes

$$D(s) + kN(s) = 0 \tag{A.3}$$

Where the roots of $D(s)$ are usually interpreted as the "open-loop poles", i.e., the poles of the transfer function of the loop from the output of the feedback sum around to the negative-feedback input of the loop, but with the loop not closed (hence "open-loop"). Similarly, the roots of $N(s)$ are the "open-loop zeros."

Some properties of the locus that are immediately clear are:

- If $k = 0$, the poles of the system are those of $D(s)$, the open-loop poles.
- As $k \to \infty$, the poles of the system become the roots of $N(s)$, the open-loop zeros.

Further, since k is assumed to be real, then if we solve for k:

$$k = -\frac{D(s)}{N(s)} \tag{A.4}$$

we can note that the imaginary part must be zero:

$$\mathrm{Im}\left(\frac{D(s)}{N(s)}\right) = 0 \tag{A.5}$$

This gives us an implicit equation independent of k, and hence the ability to draw the full locus using implicit-function drawing methods and not have to explicitly sample at various values of k (if we don't want to).[2] Note that the standard way of thinking about the locus is to only consider $k > 0$ or $k < 0$, not the whole range of k. The above implicit equation describes curves for the whole range of k, and so rendering methods often have to have special-case rules for not drawing part of the implicit curves.

Equation A.5 is useful for a numerical algorithm, but let us first note properties and rules of the root locus which are well suited for sketching.

First, note that Equation A.5 can also be interpreted [88] as: The root locus is the locus of all points that satisfy:

$$\sum \text{angles to roots of N(s)} - \sum \text{angles to roots of D(s)} = n\pi, n \text{ odd} \tag{A.6}$$

Now let us review the set of rules which Evans and others have built up over the years for

[1] The rules for root locus in s apply exactly to the z plane, as the problem is still one of finding the roots of a combination of two polynomials, only the interpretations of the root locations are different.

[2] Note that if we want to draw a locus in other angles, the above equations can simply be modified to put a phase rotation inside the Im() function.

sketching root loci (assembled from Dorf and Bishop [66], Franklin and Powell, [88], Dransfield and Haber [67], and Spencer, et al [243]):

1. A real-coefficient locus will be symmetrical about the real axis (a complex-coefficient locus need not be, but the sketching rules were only for real-coefficient loci).

2. The number of root tracks (branches, curves) is the larger of the orders of $N(s)$ and $D(s)$. Let $n = ord(N)$ and $m = ord(D)$. Historically, some sort of "properness" rule is added to require that $n \leq m$, and some sets of sketching rules may assume such a restriction. With that restriction, all roots will be finite when $k = 0$ (i.e., no additional roots will "come in from infinity" in the process of drawing the locus).

3. The tracks start at the roots of $D(s)$ when $k = 0$ and end at the roots of $N(s)$ as $k \to \infty$.

4. If $m > n$ there will be $m - n$ tracks which head to infinity as $k \to \infty$, these will follow asymptotes on angles
$$\frac{l\pi}{m-n},\ l = \pm 1, \pm 3, \cdots$$
and whose extensions come together on the real axis at
$$\frac{\sum roots(D) - \sum roots(N)}{m-n}$$
i.e., the asymptotes are equally distributed around $(0, 2\pi)$. For example: if $m - n = 2$, the asymptotes are $\pm\pi/2$ (i.e., the two vertical angles), if $m - n = 4$, the asymptotes are the 45-degree angles ($\pm\pi/4$ and $\pm 3\pi/4$), etc. If $n > m$, then $n - m$ poles will "arrive from infinity" on similar asymptotes.

5. The 0-degree locus version of the previous rule are:
$$\frac{l\pi}{m-n},\ l = \pm 0, \pm 2, \pm 4, \cdots$$
and
$$\frac{\sum roots(D) - \sum roots(N)}{m-n}$$
These asymptotes are "halfway between" the 180-degree asymptotes.

6. The 180-degree root locus occupies the regions of the real axis which have an odd-number of real open-loop poles and zeros to the right.

7. The 0-degree root locus occupies the regions of the real axis which have an even-number of real open-loop poles and zeros to the right. Hence the real axis is completely covered by the "full locus in k".

8. If there is a section of locus between consecutive open-loop poles (i.e., with no open-loop zeros between them), then there will be a "breakaway" somewhere between them. A breakaway is where two real poles come together and split off to become a complex pole pair as k increases.

9. If there is a section of locus between consecutive open-loop zeros, then there will be a "breakin" somewhere between them (where a complex pole pair comes together at the real axis and splits off to become two real poles as k increases).

10. The angle at which a pole leaves a complex open-loop pole of multiplicity one (the "departure angle") is determined by:

$$\theta = \sum(\text{angles to open-loop zeros}) - \sum(\text{angles to open-loop poles}) - \pi$$

The angles at which poles of multiplicity l leave are given by

$$\frac{\theta + n\pi}{l},\ n = \pm 1, \pm 3, \cdots$$

It is assumed that the graphical accounting of angles to poles and zeros was one of the tasks which was assisted by the spirule.

11. As with asymptotes, the departure angles for a 0-degree locus are halfway between the departure angles of the 180-degree locus (for multiplicity one, that would end up being the opposite direction as the 180-degree departure angle).

12. The arrival angles at open-loop zeros are given by the negative of the same equation (evaluated at the zero location, of course).

13. Locations for where the locus crosses the imaginary axis (of most interest for s-plane loci, but possibly of use in drawing loci in z as well) can be found by the *Routh-Hurwitz criterion* ([226][112]).

14. The locus will have multiple roots at points on the locus where

$$\frac{d}{ds}(G(s)H(s)) = 0$$

or equivalently

$$D(s)\frac{dN(s)}{ds} - N(s)\frac{dD(s)}{ds} = 0$$

This is often used primarily to locate breakaway/breakin points on the real axis, as at such points the breaking poles are on top each other right at the break. It can also be used to analyze other situations where poles come together off the real axis. As with open-loop poles or zeros

with multiplicity, the poles will approach equally-spaced around the angles according to the number of approaching poles, and leave at angles halfway between.[3] For example, two poles which meet will approach each other at exactly opposite angles ("head-on"), and will depart at opposite angles, rotated 90 degrees from the arrival angles.

15. The derivative of a pole's location on the locus, with respect to k (i.e.,its local departure direction) is:

$$\frac{ds}{dk} = \frac{-\frac{\partial}{\partial k}(D(s) + kN(s))}{\frac{\partial}{\partial s}(D(s) + kN(s))} = \frac{-N(s)}{D'(s) + kN'(s)}$$

The magnitude of this function is also known as the *sensitivity function* and describes how sensitive the locus poles are to k, in other words, how fast the poles move as k changes. We can also derived the sensitivity from $1 + kN/D$ rather than from $D + kN$ and we get:

$$\frac{ds}{dk} = \frac{-\frac{\partial}{\partial k}(1 + kN(s)/D(s))}{\frac{\partial}{\partial s}(1 + kN(s)/D(s))} = \frac{N(s)D(s)}{k[N(s)D'(s) - D(s)N'(s)]}$$

These two functions, though they are different, have the same effective behavior (as can be seen in Figure A.12). It is commonly known that pole sensitivity goes to infinity when multiple roots meet (for example at breakaways). Some drawing methods may use this function to modulate the step size in k to not miss details at such points. Some methods might make use of this function (or its denominator) to help pinpoint the k at the center of such a feature.

This list is more extensive than some, as some authors do not necessarily include all the latter rules.

Some interesting notes:

2-pole breakouts are perpendicular to real axis. In keeping with the rules for arrival and departure angles for two-pole meetings, a two-pole breakout or breakin will always be perpendicular to the real axis. A breakout or breakin on the real axis involving *more* than two poles will always involve poles entering and leaving from *both* sides of the meeting point (i.e., if the meeting point is on the stability boundary ($z = 1$ or $s = 0$), then at least one entering and departing pole must be unstable.

Non-intersection. Pole tracks will not self-intersect, or intersect each other. They may *touch* at points of multiplicity, but otherwise, if a point in the s plane is a member of the locus for for some k_0, it will not be a member of the locus for any other k_0.

[3] This can be viewed in terms of earlier rules by offsetting k to the point where the poles meet, at which time this secondary locus can be considered to have "open-loop" poles of the current multiplicity at that point, with the "arriving" poles coming in along the 0-degree locus and the "departing" poles leaving along the 180-degree locus [88].

Swapping D and N. (i.e., swapping the open-loop poles and zeros) gives the same full locus shape. This can be seen from a few angles. First, if $\mathrm{Im}(D/N) = 0$, then $\mathrm{Im}(N/D) = 0$ too. Second,

$$D(s) + kN(s) = 0 \Rightarrow N(s) + \frac{1}{k}D(s) = 0$$

Thus, the k mapping is all that is different. For filter design, this can make for some possible confusion about what should be poles and what should be zeros in the open-loop, as they can effectively be swapped, though the direction that the poles move with k should be able to clear up any such confusion.

Offsetting k. Offsetting the locus parameter keeps the same locus shape, simply moving the open-loop poles:

$$\begin{aligned} D(s) + kN(s) &= 0 \\ k &= \hat{k} + a \\ D(s) + (\hat{k} + a)N(s) &= 0 \\ (D(s) + aN(s)) + \hat{k}N &= 0 \end{aligned}$$

The new open-loop poles are at the location of $k = a$ in the original locus. Similar analysis can be done for any other mapping of k. For example, mapping $k = \hat{k}^2$ gives the same locus shape, but only the $k > 0$ locus, since (for real $\hat{k}$) $\hat{k}^2 > 0$.

Feedback around an FIR filter. This examines the question about whether an FIR filter has poles. If take an FIR filter: $H(z) = n_0 + n_1 z^{-1} + \cdots + n_N z^{-N}$ and put negative feedback around it with a loop gain of k, we get:

$$\frac{n_0 + n_1 z^{-1} + \cdots + n_N z^{-N}}{1 + k(n_0 + n_1 z^{-1} + \cdots + n_N z^{-N})} = \frac{n_0 z^N + n_1 z^{N-1} + \cdots + n_N}{z^N + k(n_0 z^N + n_1 z^{N-1} + \cdots + n_N)} \tag{A.7}$$

As such, the closed-loop system acts as though it has N poles at $z = 0$, which can also be seen if we write the FIR in terms of z rather than z^{-1}:

$$\frac{n_z^N + n_1 z^{N-1} + \cdots + n_N}{z^N} \tag{A.8}$$

The locus of such a system starts with all poles at $z = 0$ with $k = 0$, and the system simply having the FIR response. As k increases, the poles split out in the standard distribution out the angles (N branches), depending on the sign of k, and eventually at high enough gain reach the open-loop zeros. A typical example is shown in Figure A.2. A typical behavior is that one sign of k gives root tracks which head nearly directly to the zeros, hence giving a filter with whose zeros get narrower

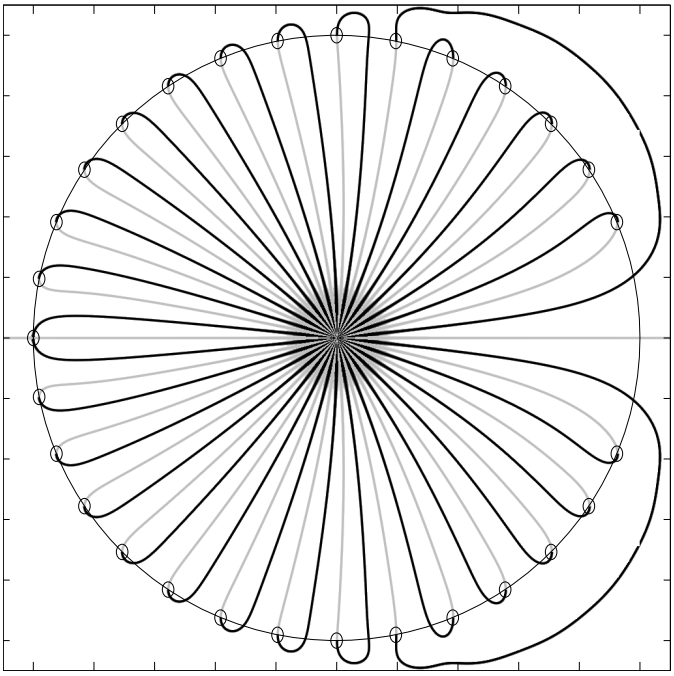


Figure A.2: Root Locus for a system consisting of feedback around an FIR filter whose taps are a 31-point Hann window.

and narrower, and whose transfer function otherwise ends up pretty flat (much like how a DC blocker works, by putting a zero on the unit circle and bringing a pole up close to it at a slightly lower radius). The other sign of k sends the poles outward at angles between the open-loop zeros, which has the effect of making peaks between the notches of the zeros.

Karplus-Strong Filter and Flanger. These are degenerate cases of the FIR example, where the FIR filter is just a delay, and hence has all infinite zeros (if the delay is an integer):

$$H(z) = z^{-N}.$$

The locus is simply the N-way branching from $z = 0$ out to infinity (see Figure A.3). As one would expect, the poles reach the unit circle when $|k| = 1$. The sign of k determines which rotation of the branching the poles take. As is known in the use of Karplus-Strong models, one sign of k gives an impulse response with all harmonics (pulse-like) (in this case $k < 0$, because of the assumed negative feedback), and the other sign of k gives only odd-harmonics (pulses with alternating sign). This behavior can be interpreted in terms of the different rotations of the locus branches.

What about delays of non-integer lengths? A common interpolation is linear. Figure A.4 shows

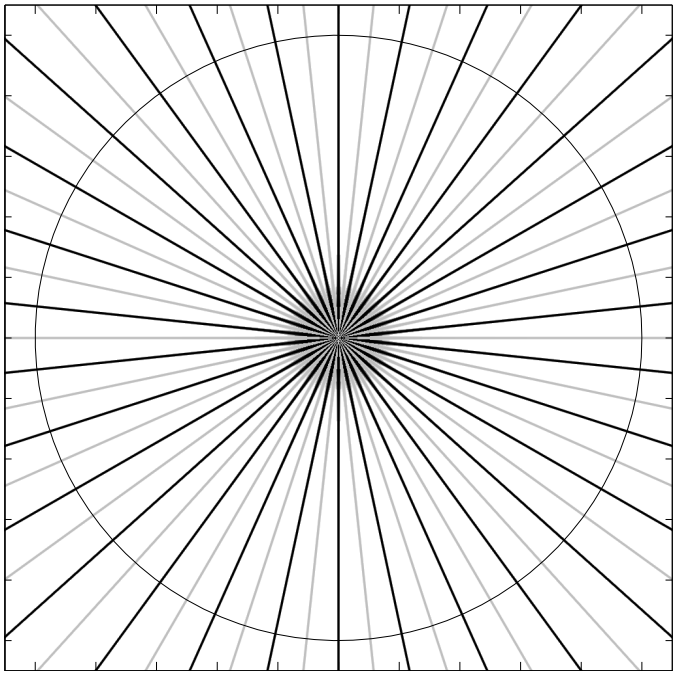


Figure A.3: Root Locus for a system consisting of feedback around an integer-length delay of 30 samples (as in a Karplus-Strong string model or a Flanger.

the locus for a common situation, with linear interpolation for 1/2 sample:

$$H(z) = 0.5z^{-N} + 0.5z^{-(N+1)}.$$

An open-loop zero is added due to the interpolation, which has the affect of decreasing the radius of the higher "harmonics" for $k = -1$, and also slightly warping their frequencies.

Another common interpolation is allpass interpolation. Figure A.5 shows a 5-sample delay loop with an additional one-pole allpass in the loop (such that when the allpass coefficient is 0, the total loop delay is 6 samples):

$$H(z) = z^{-5}\frac{c + z^{-1}}{1 + cz^{-1}} \tag{A.9}$$

The frequency-warping effect of strong allpass filtering is quite obvious for "strong" allpass coefficients, and the effect is visible in these loci, as the allpass pulls the frequencies downward, and in the extreme of $c = -1$ would have the effect of swapping the effective sign of k. Note how the branches emanating from $z = 0$ change from 6-way branching to 5-way branching.

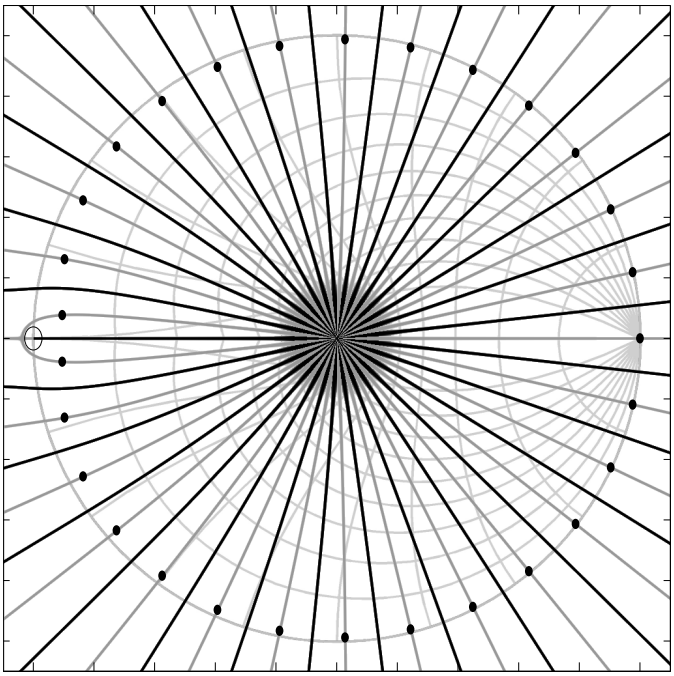


Figure A.4: Root Locus for a system consisting of feedback around an delay of 29.5 samples, implemented using linear interpolation. Dots: $k = -1$ (which together with the negative feedback gives a positive loop gain)

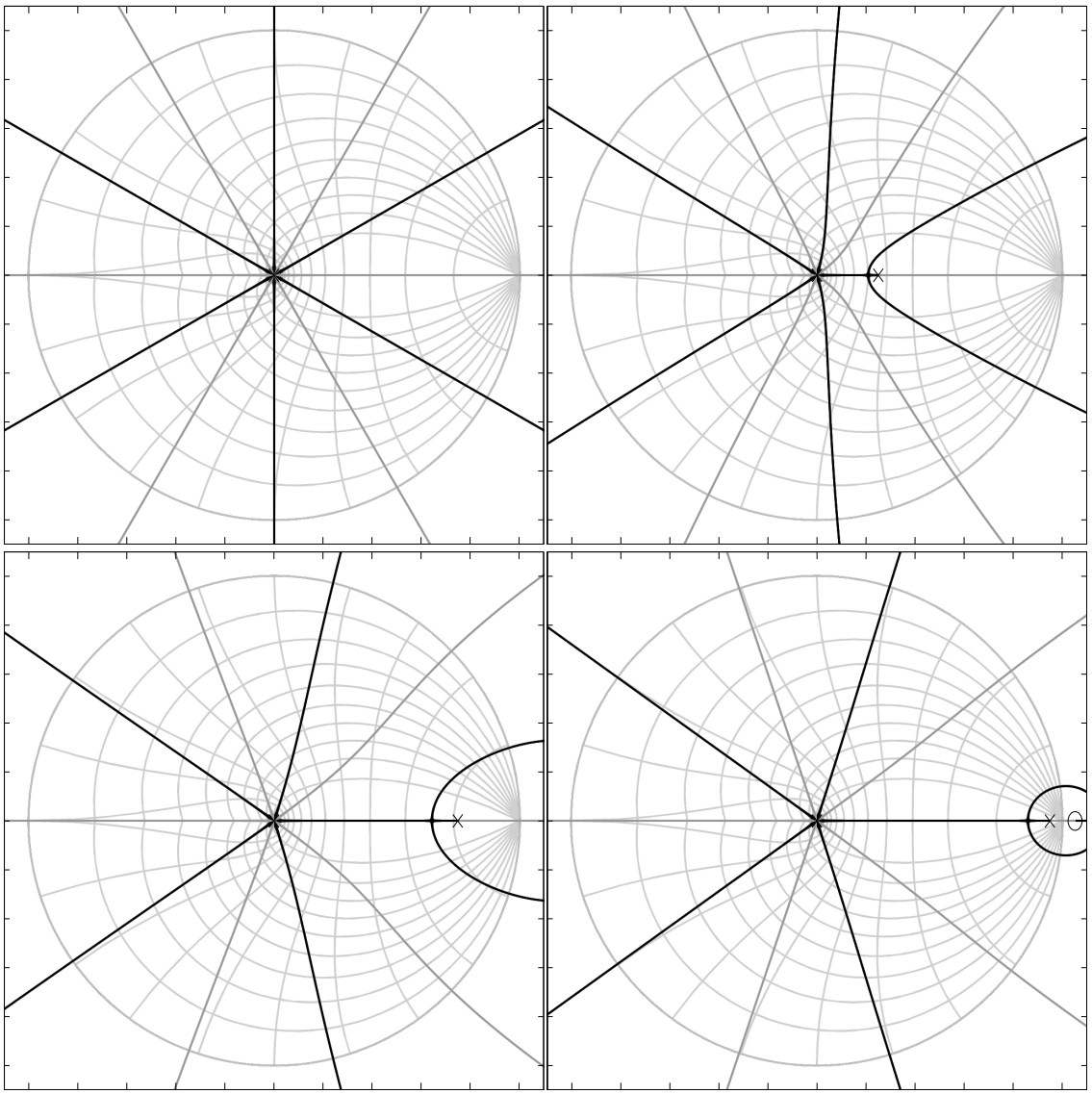


Figure A.5: Root Loci for a system consisting of feedback around delays between 5 and 6 samples, implemented using allpass interpolation. left-to-right, top-to-bottom: allpass coefficient = 0, -0.25, -0.75, -0.95

A.1.3 Higher-Order Loci

In the analysis of the state-variable filter and Moog-style filters, locus equations that are higher-order polynomials in the "gain" variable are often encountered. We will call these "higher-order" loci. For example, a 2^{nd}-order locus in k would be:

$$D(s) + kN_1(s) + k^2N_2(s) = 0 \tag{A.10}$$

We name the polynomials in this manner in comparison with the first-order locus. In the state-variable filter, the root-locus is 2^{nd}-order in the frequency coefficient (but only 1^{st}-order in the Q coefficient). In Moog-style filters, the locus is 4^{th}-order in the tuning coefficient (and again 1^{st}-order in the Q coefficient). For higher orders, we continue the naming scheme:

$$D(s) + kN_1(s) + k^2N_2(s) + \cdots + k^NN_N(s) = 0 \tag{A.11}$$

Not much literature can be found on higher-order root locus, though a book on the subject was found, written in 1981 by Hauberk Kahn [102].

A Look at 2^{nd}-Order Locus

What can we quickly say about a 2^{nd}-order locus?

First, in comparison to a first-order locus, the roots start a those of D, and end at the roots of N_2. N_1 has the effect of "pulling" on the locus in some way.

Next, we can think of it in terms of the 1^{st}-order loci of its components. The 2^{nd}-order locus lies within the envelope of the first-order loci:

$$\begin{aligned} D(s) + kN_1(s) &= 0 \\ D(s) + kN_2(s) &= 0 \\ N_1(s) + kN_2(s) &= 0 \end{aligned}$$

Figure A.6 shows the 2^{nd}-order locus for the polynomials:

$$\begin{aligned} D(s) &= s^4 - s^3 + 4 \\ N_1(s) &= s^4 + 2s^2 + s - 2 \\ N_2(s) &= 0.2 * (2s^4 - 4s^2 + 3s) \end{aligned}$$

The roots of D are shown by 'x', the roots of N_1 by squares, and the roots of N_2 by 'o'. The roots start at the roots of D when $k = 0$, and at low k, the roots head "towards" the roots of N_1 near the $D + kN_1$ locus, then pull away and head towards the roots of N_2, arriving at high k along the

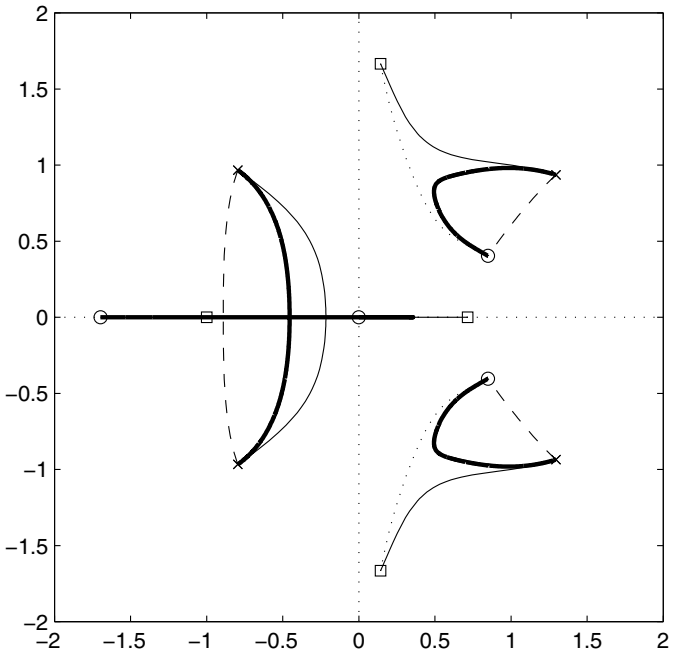


Figure A.6: A 2nd-order locus lives within the envelope of three 1st-order loci.

$N_1 + kN_2$ locus.[4] We will look into the strength of the pull of N_1 later on in this section.

On first look, the method whereby we derived a 2D implicit equation for the full 1st-order locus which didn't depend on k wouldn't appear to have an extension to higher-order loci. This is initially a worry, since some of the nicer rendering methods use the implicit function form of the locus. However, it is still possible to use implicit-function methods. First, one could think of the locus equation as an implicit function in three dimensions (s plus k) and project 3D implicit-function renderers down to the s plane (we will see some of that later on in this appendix). Alternatively, we can solve the locus equation for k:

$$k = \frac{-N_1(s) \pm \sqrt{N_1^2(s) - 4N_2(s)D(s)}}{2N_2(s)} \tag{A.12}$$

and apply the same requirement that it be real:

$$\mathrm{Im}\left(\frac{-N_1(s) + \sqrt{N_1^2(s) - 4N_2(s)D(s)}}{2N_2(s)} \right) = 0 \tag{A.13}$$

OR

[4]This behavior is reminiscent of spline curves (Breezier splines in particular)

$$\text{Im}\left(\frac{-N_1(s) - \sqrt{N_1^2(s) - 4N_2(s)D(s)}}{2N_2(s)}\right) = 0 \tag{A.14}$$

This system of two implicit equations defines the locus, which can also be interpreted as: "The locus is the union of the solutions of the two implicit equations." For example, let us look at the polynomials:

$$\begin{aligned} N_1(s) &= 11s^5 - 5s^4 + 10s^3 + 5s^2 - 13s \\ N_2(s) &= s^5 + 4s^4 + 3s^3 + 3s^2 + s + 2 \\ D(s) &= 8s^5 + 3s^4 - s^3 + 3s \end{aligned}$$

The full 2^{nd}-order locus is shown in Figure A.7, along with the solutions to the two sub-problems shown separately.

This concept generalizes to loci of higher order in k:

$$P(k) = D(s) + kN_1(s) + \cdots + k^N N_N(s) = 0 \tag{A.15}$$

Each root of $P(k)$ in k generates an implicit equation in s, and the root locus in k is the union of the solutions to those equations:

$$\text{Im}\left[\text{rootk}_i P(k)\right] = 0, i \in (1, \cdots, N) \tag{A.16}$$

where rootk_i denotes the i^{th} root in k of $P(k)$.

Scaling and Sign Issues In first-order root locus, the raw scaling of D and N do not affect the *shape* of the locus:

$$\begin{aligned} aD(s) + kbN(s) &= 0 \\ D(s) + k(b/a)N(s) &= 0 \\ D(s) + \hat{k}N(s) &= 0 \end{aligned}$$

In other words, changing the scales simply results in a remapping of k (though if plotting only one sign of k, the signs of the scalings can effect which half of the locus is implied).

However, if we do the same manipulation to a 2^{nd}-order locus equation, we get the following:

$$\begin{aligned} aD(s) + kbN_1(s) + k^2cN_2(s) &= 0 \\ D(s) + k(b/a)N_1(s) + k^2(c/a)N_2(s) &= 0 \\ D(s) + \hat{k}N_1(s) + \hat{k}^2(ca/b^2)N_2(s) &= 0 \end{aligned}$$

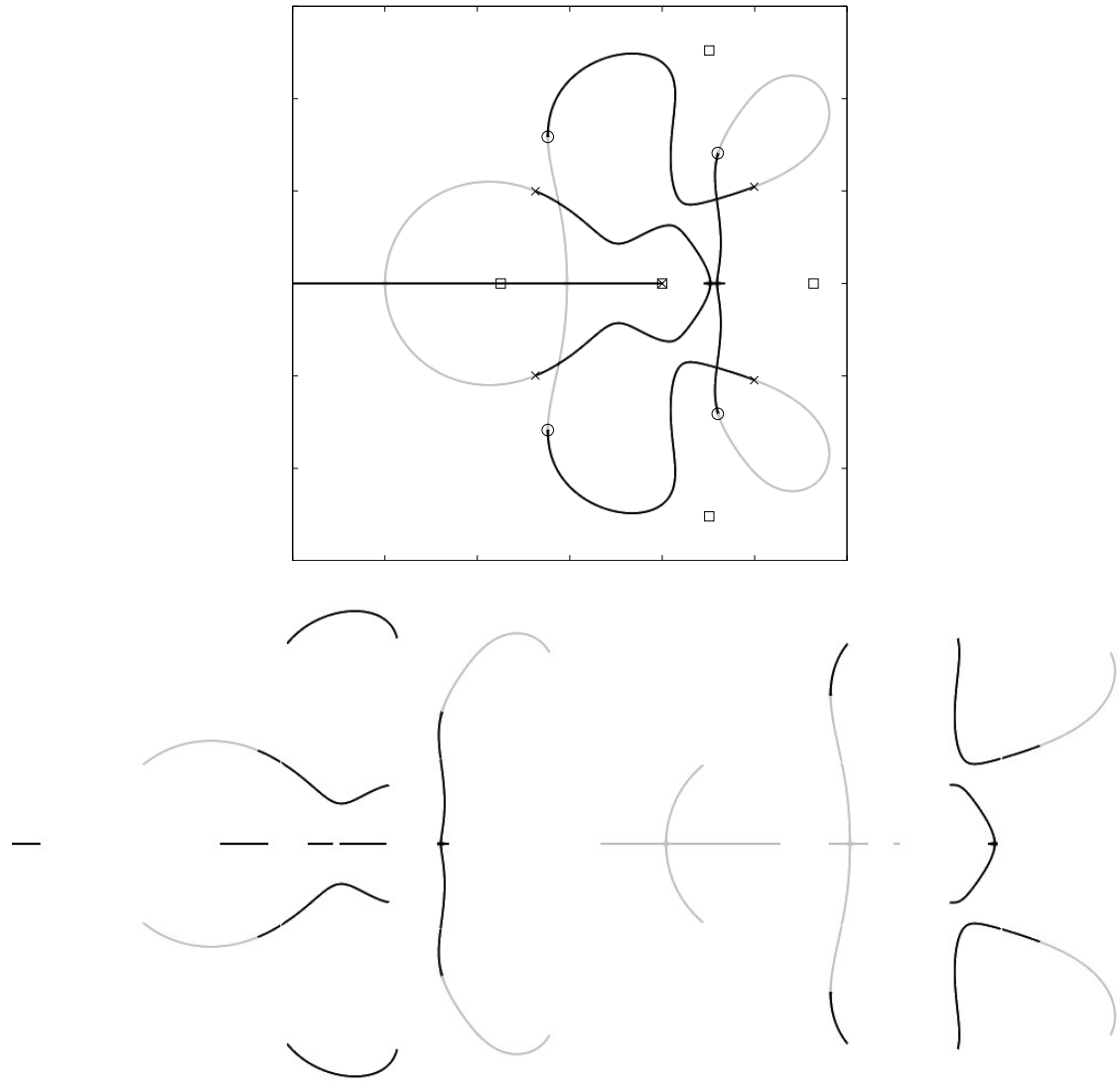


Figure A.7: Top: A 2nd-order locus, dark: $k > 0$, light: $k < 0$. Bottom Right: one of the solutions of quadratic equation. Bottom Left, the other solution.

or, with slightly different algebra:

$$\begin{aligned} aD(s) + kbN_1(s) + k^2cN_2(s) &= 0 \\ D(s) + k(b/a)N_1(s) + k^2(c/a)N_2(s) &= 0 \\ D(s) + \hat{k}\sqrt{b^2/ac}N_1(s) + \hat{k}^2N_2(s) &= 0 \end{aligned}$$

such that we do not end up with a simple remapping of k. As such, there is an extra degree of freedom in in the shape of a 2nd-order locus. By analogy, an N^{th}-order root locus has $N-1$ further degrees of freedom in the locus shape, related to the relative gains (and signs) of the polynomials.

Effect of the extra degree of freedom . To answer an earlier question: this degree of freedom affects the pull that the roots of $N_1(s)$ have on the locus. Figure A.8 shows a set of loci for the

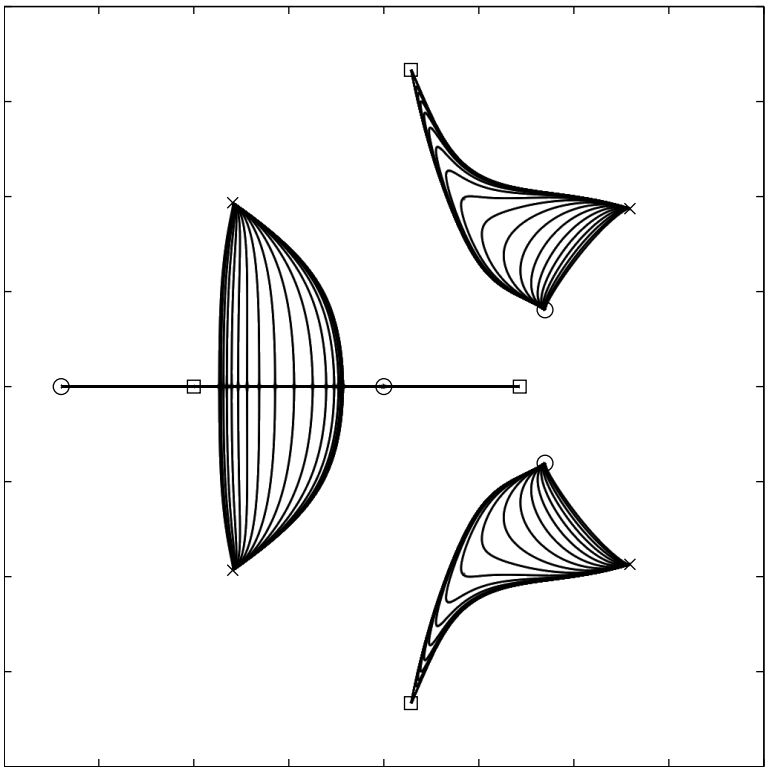


Figure A.8: Locus of Figure A.6 with various scales on N_2 between 10^{-6} and 10^{6}.

system of Figure A.6, where the scale on N_2 varies over a range of 10^{-6} to 10^{6}. Therefore, not only does the 2nd-order locus live within the envelope of the three first-order loci, depending on the relative scale of N_1 and N_2, it can visit any point within that envelope.

Let's look again at the system of Figure A.7 with different scales on its N_2. Looking at Figure A.9, we can see some trends:

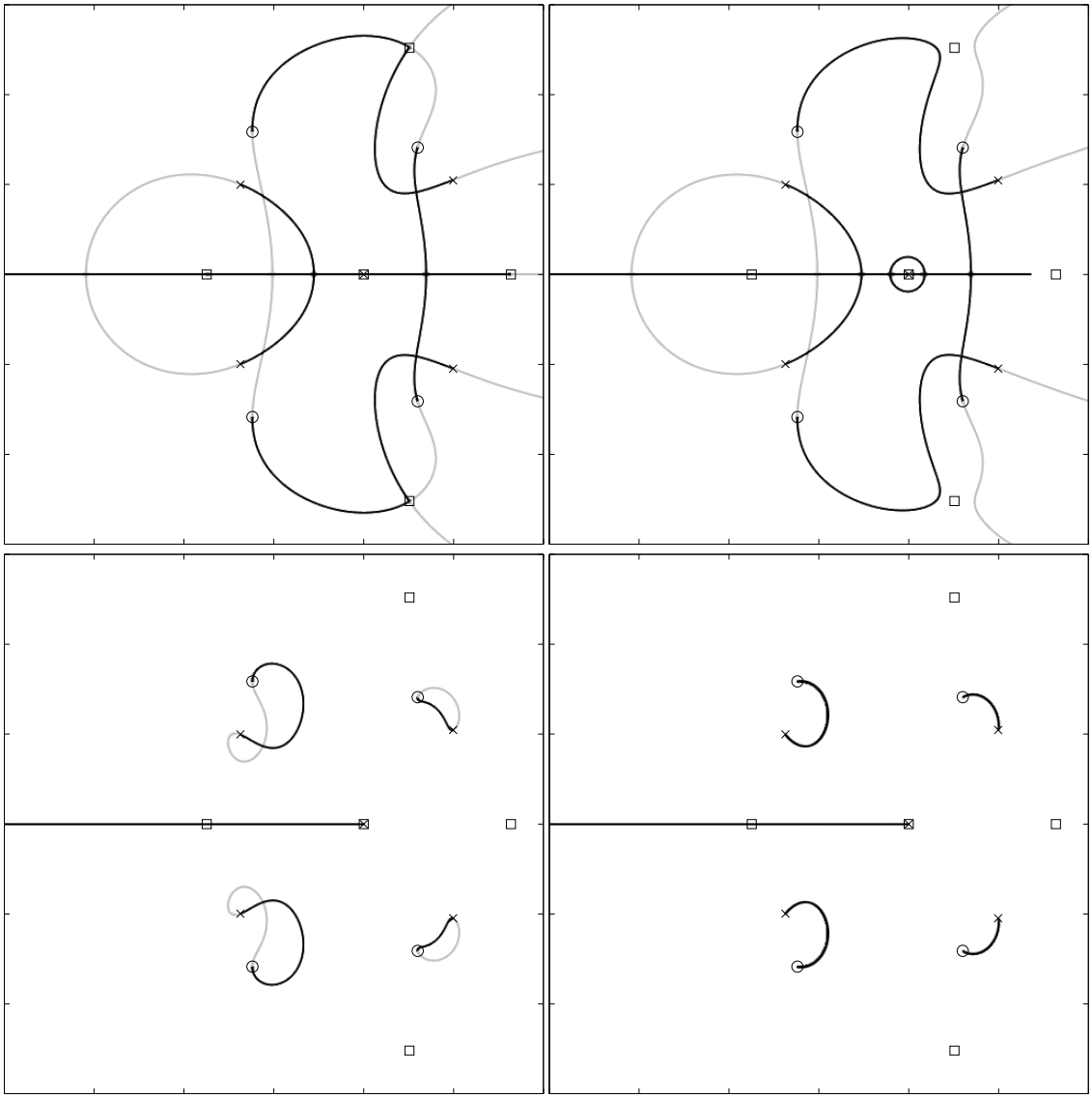


Figure A.9: Locus of Figure A.7 with various scales on N_2: Top Left: 10^{-5}, Top Right: 10^{-1}, Bottom Left: 10^{2}, Bottom Right: 10^{5}.

- As the scale on N_2 gets small, the "pull" from the zeros of N_1 gets progressively stronger.
- As the scale approaches zero (but still nonzero), the locus approaches the union of the 1^{st}-order loci of $D + kN_1 = 0$ and $N_1 + kN_2 = 0$. The low-$|k|$ range traces the $D + kN_1 = 0$ locus,

ending up at the roots of N_1, and then the very high range of $|k|$ traces the $N_1 + kN_2 = 0$ locus, eventually ending up at the roots of N_2.

- A scale of zero is a degenerate system from the standpoint of the second-order locus, and only represents the $D + kN_1 = 0$ locus.
- As the scale gets large (and is positive), the positive-k and negative-k halves of the locus start approaching each other, until they meet on the 1^{st}-order locus $D + kN_2 = 0$, but only tracing the $k > 0$ side of it (regardless of the actual sign of k, because it is squared on the N_2 term.)
- Similarly, as the scale gets large but negative, the halves approach the $k < 0$ side of the first-order locus $D + kN_2 = 0$. This case is not pictured in Figure A.9.

A dual set of rules can be described if N_1 is getting scaled rather than N_2.

Remember that the scaling is actually about the relative scales of N_1 and N_2. With a small bit of algebra, one can show:

$$D + akN1 + k^2N2 \Rightarrow D + \hat{k}N1 + (1/a^2)\hat{k}^2N2 \tag{A.17}$$

Hence scaling N_2 up can give the same shape as scaling N_1 down an appropriate amount, though the k mapping would be different. Remember that the signs of the polynomials are part of their scales, and they affect how the sign of k traces out the various "sides" of the loci.

Some further ways in which higher-order loci can differ from first-order loci

- Whereas any particular first-order locus track cannot self-intersect, such intersection is possible with higher-order loci. Locus tracks can hit the same points in the s plane at different values of k (either the same track or a different track). See Figure A.7, for example. Backtracking along the same track is also possible. This was seen in the analysis of the state-variable-like derived filter in Section 3.2.2, where a real pole headed towards $z = 1$, but then reversed and headed back towards $z = -1$, so that the locus appeared to "stop" at some point.
- As we will see in Section A.2, higher-order loci can be set up to exhibit breakaways which do not have to be perpendicular to the real axis, and can, in fact, have controllable angle.

A.2 A Derivation of the "X" Locus

In an earlier chapter, we posed the question of how one might derive a locus with constant-Q lines. We also saw in Section 2.4.4 that the continuous-time state-variable filter possess such a locus in the s-plane. Here we present a derivation of such a system from the standpoint of desiring an X-shaped locus with variable-slope lines.

Problem Statement: Derive the equation for a (real-coefficient polynomial) root locus whose shape is a ray (and its complex conjugate) of controllable slope starting at $s = b$ on the real axis when $k = 0$, and heading "up and to the left" with a slope of a as k increases in the positive direction.

The desired ray can be described by:

$$\begin{aligned} x(k) &= (b - k) \\ y(k) &= ak \end{aligned}$$

Or, alternately, a point on the ray can be described in the s-plane by:

$$s_0(k) = (b - k) + jak \tag{A.18}$$

Thus, a polynomial system with roots on s_0 and its conjugate, parameterized by k, would be:

$$\begin{aligned} (s - s_0)(s - s_0^*) &= 0 \\ (s - \mathrm{Re}(s_0))^2 + \mathrm{Im}(s_0)^2 &= 0 \\ (s - (b - k))^2 + a^2k^2 &= 0 \\ ((s - b) + k)^2 + a^2k^2 &= 0 \\ (s - b)^2 + 2(s - b)k + (1 + a^2)k^2 &= 0 \end{aligned} \tag{A.19}$$

Or, if we wish to control slope by scaling N_1 rather than N_2, we can transform k (as described in a previous section) and get:

$$(s - b)^2 + \frac{2}{\sqrt{1 + a^2}}(s - b)k + k^2 = 0 \tag{A.20}$$

As we have seen before, this locus is a 2^{nd}-order locus in k, and it has both of its "open-loop poles" (roots of $D(s)$) on $s = b$, and also one of the $N_1(s)$ roots on $s = b$ as well, with both roots of $N_2(s)$ and the other root of $N_1(s)$ at infinity.

As such, we have a plausible direction from which one might derive such a locus if one had not already come across it through analysis of the continuous-time state-variable filter. It also presents a method one might use when attempting to derive other root-locus-based constant-Q filters.

Note that if $N_1(s)$ has its finite root elsewhere on the real axis than $s = b$, then the locus is a hyperbola, with the above "X" lines being its asymptotes. The above "X" is the degenerate hyperbola for when the two foci come together at the same point. Only in the degenerate case of the roots on top of each other do the locus tracks actually leave the axis at an angle, otherwise they leave perpendicularly.

A.2.1 Deriving a Moog-style constant-Q Locus

We can also apply the above derivation method to deriving a Moog-style locus in the s-plane. As seen in Section 2.5, the locus in p for the ideal continuous-time Moog-style filter is a complex-conjugate pair of rays at some angle to the left of the imaginary axis, together with a second pair at an angle closer to the real axis, all originating at $s = 0$.

On first thought, one might use:

$$s_0(p) = (b - p) + ja_0p$$
$$s_1(p) = (b - p) + ja_1p$$

Where a_0 and a_1 are the desired slopes. However, since these ray equations are parameterized such that the horizontal location along the ray is parameterized directly to p, that would cause s_0 and s_1 for any particular p to be offset vertically from each other. However, we saw in Figure 2.29 that the poles for a particular p are actually offset horizontally (which makes the dominant poles much more dominant). Therefore we need to use a different line parameterization, such as:

$$s_0(p) = c_0(b - p) + jp$$
$$s_1(p) = c_1(b - p) + jp$$

Where $c_0 = 1/a_0$, and $c_1 = 1/a_1$. This parameterization will cause s_0 and s_1 to have the same imaginary part for any particular p.

Following the above derivation, we get the 4^{th}-order locus equation

$$D(s) + N_1(s)p + N_2(s)p^2 + N_3(s)p^3 + N_4(s)p^4 = 0$$

where

$$\begin{aligned}
D(s) &= (s - c_0b)^2(s - c_1b)^2 \\
N_1(s) &= 2(s - c_0b)(s - c_1b)((c_0 + c_1)s - 2c_0c_1b) \\
N_2(s) &= b^2(c_0^2 + c_1^2 + 6c_0^2c_1^2) - 2b(c_0 + c_1 + 3c_0^2c_1 + 3c_0c_1^2)s + (2 + 4c_0c_1 + c_0^2 + c_1^2)s^2 \\
N_3(s) &= -2b(c_0^2 + c_1^2 + 2c_0^2c_1^2) + 2(c_0 + c_1 + c_0^2c_1 + c_0c_1^2)s \\
N_4(s) &= (1 + c_0^2)(1 + c_1^2)
\end{aligned}$$

Now the Moog filter rays start at the origin, so $b = 0$, and things simplify significantly:

$$\begin{aligned}
D(s) &= s^4 \\
N_1(s) &= 2(c_0 + c_1)s^3
\end{aligned}$$

$$\begin{aligned} N_2(s) &= (2 + 4c_0c_1 + c_0^2 + c_1^2)s^2 \\ N_3(s) &= 2(c_0 + c_1 + c_0^2c_1 + c_0c_1^2)s \\ N_4(s) &= (1 + c_0^2)(1 + c_1^2) \end{aligned}$$

These equations allow us to specify the slopes of both rays.[5] However, in the Moog-style filter, the ray angles are constrained. If we refer again to Figure 2.29, we see that the poles for any particular p form a square. Hence, if the dominant-pole ray slope is a_0, then the non-dominant-pole ray slope is restricted be a function of a_0. With a bit of geometric construction, we can derive that if $a_0 = c/d$, then $a_1 = c/(d + 2c)$, or

$$a_1 = \frac{a_0}{1 + 2a_0} \tag{A.21}$$

Or, in terms of c_0 and c_1:

$$c_1 = c_0 + 2 \tag{A.22}$$

And the 4th-order locus equations further simplify to:

$$\begin{aligned} D(s) &= s^4 \\ N_1(s) &= 4(1 + c_0)s^3 \\ N_2(s) &= 6(1 + c_0)^2s^2 \\ N_3(s) &= 4(1 + c_0)^3s \\ N_4(s) &= 4 + (1 + c_0)^4 \end{aligned}$$

Which compares well to Equation 2.66, and one can derive a relation between k and the dominant-pole ray slope a_0.

As noted elsewhere, and in comparison with the "X" locus, we have the interesting pattern that the roots of each coefficient of p has one fewer root at $s = 0$ than the previous one (starting at $D(s)$, the zeroth-order coefficient of p)

A.2.2 An interesting extension of the "X" locus

It turns out that some of the properties of the "X" locus (in the locus domain, not necessarily in the filter domain) extend to locus equations of the form:

$$(s - x)^{2n} + \alpha k(s - x)^n + k^2 = 0 \tag{A.23}$$

where n is a positive integer.

[5] There may be some interesting further exploration into the filter-design implications of the ray angle of the non-dominant poles. Also, of course, there is the research problem of inverting from these equations back to a similar filter topology as the ideal Moog-style filter.

Remember that the "X" locus (Section A.2) (which corresponds to $n = 1$), when scaled on N_1 as in this equation, ranges between the trivial locus $(s - x) + k = 0$ when $\alpha \to \infty$ at one extreme, and the other trivial locus $(s - x)^2 + |k| = 0$ when $\alpha = 0$ on the other extreme on the extremes of scaling in a 2nd-order locus). The interesting thing was that between these extremes, the locus is always a set of rays originating from the origin, their angles controllable by α.

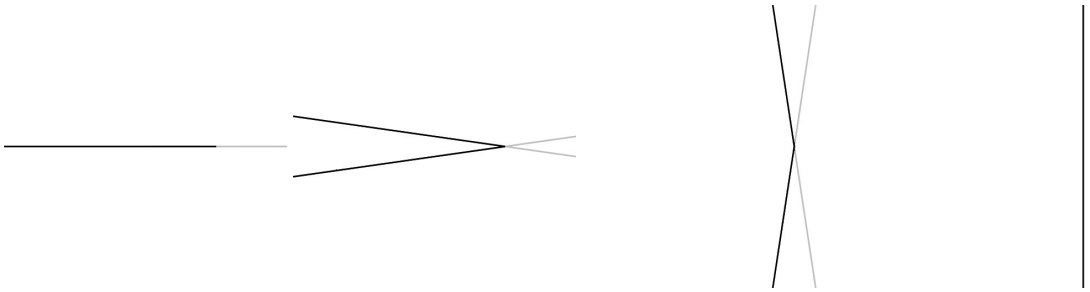


Figure A.10: Loci of $(s-1)^2 + \alpha k(s-1) + k^2 = 0$ for (left to right) $\alpha = 2$, $\alpha = 1.98$, $\alpha = 0.3$, $\alpha = 0$. Range: $-2 \le \mathrm{Re}(s), \mathrm{Im}(s) \le 2$

Figure A.10 shows that as α ranges from 2 down to zero, the $(s - x) + k = 0$ locus (i.e., the positive and negative real axis, depending on the sign of k) "splits" into pairs of rays, which rotate as α decreases until meeting with one of the rays from the opposite sign of k at lines parallel to the imaginary axis (forming the $(s - x)^2 + |k| = 0$ locus).

Now for higher values of n, the same thing happens, just replicated n times around the angles 0 to 2π, as we will see below. If we let $b = (s - x)^n$, then Equation A.23 becomes:

$$b^2 + akb + k^2 = 0 \tag{A.24}$$

Solving for b:

$$b = k\left(-\frac{\alpha}{2} \pm \sqrt{\left(\frac{\alpha}{2}\right)^2 - 1}\right) \tag{A.25}$$

Which, if $\alpha < 2$, corresponds to:

$$\begin{aligned} b &= k(-\cos(\phi) \pm j\sin(\phi)) \\ b &= ke^{\pm j(\phi+\pi)} = -ke^{\pm j\phi} \quad \left(0 \le \phi \le \frac{\pi}{2}\right) \end{aligned}$$

Now substituting the definition for b:

$$\begin{aligned} (s - x)^n &= -ke^{\pm j\phi} \\ s &= x + \sqrt[n]{-ke^{\pm j\phi}} \end{aligned} \tag{A.26}$$

Remember that the n^{th} roots of a complex number z are distributed in a circle at equally-spaced angles away from $\angle z$ (i.e., at $[\angle z/n, \angle z/n \pm 2\pi/n, \angle z/n \pm 4\pi/n, \cdots]$), thus the $n = 1$ shape is replicated n times at angles around $s = x$. See Figure A.11.

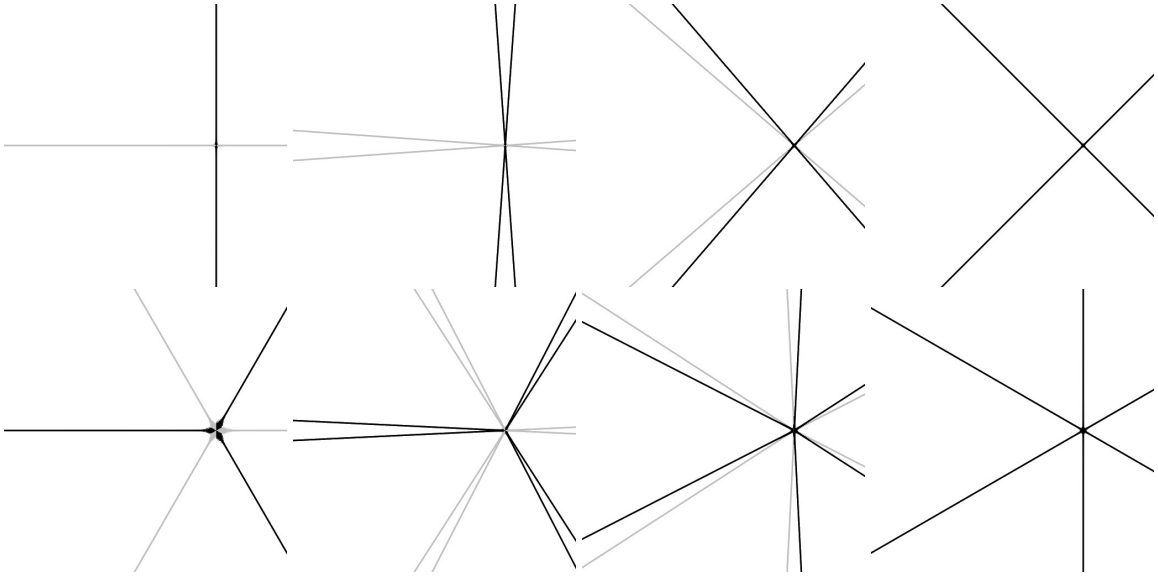


Figure A.11: Loci of $(s-1)^{2n} + \alpha k(s-1)^n + k^2 = 0$ for (left to right) $\alpha = 2$, $\alpha = 1.98$, $\alpha = 0.3$, $\alpha = 0$. Top: $n = 2$, bottom: $n = 3$. Range of plots: $-2 \leq \mathrm{Re}(s), \mathrm{Im}(s) \leq 2$

A.3 Locus Drawing Methods

While researching the interpretation of variable filters via root locus concepts, it quickly became necessary to be able to draw them. While MATLAB certainly contained a root locus renderer, the version that existed at the time (early 1990s) was not perfect, particularly from the standpoint of someone desiring to produce "neat pictures." The loci were more than sufficient for use in control-system design, but artifacts were common. As such, a secondary track of research was begun exploring methods by which loci are drawn, from a graphics-algorithm perspective, with the basic goal to come up with algorithms which could produce "clean pictures" of the loci.

At about the same time, attempts at fitting loci to desired tracks were going on, and one of the major issues in that research was finding good-but-fast distance measures (i.e., how far is a track from a set of points?). That research ended up dovetailing with the locus-rendering research, particularly in the form of Taubin's method (Section A.3.4 upcoming), which was originally encountered due to its good approximations of the distance form a point to the solution set of an implicit function.

A Short History. The first root-locus drawing method was Evans' own method, of course [73]. It was the primary method for most until computer-based methods became common. However, various papers did propose other methods, mostly based on analytic approaches to the locus. One method of particular interest, which the author only has anecdotal evidence of [54], was the use of a rubber sheet and Tinker-Toys: Dowels of various heights (related to pole gains in a partial-fraction expansion?) were anchored to a table top using various methods, and then a latex sheet was lowered over them and anchored to the table top at some distance from the poles (one story mentions the use of a large sewing hoop), and zero locations were pressed down to the table with further dowels and/or weights. Since it was known that the locus followed steepest-descent paths down the amplitude of the loop transfer function in the s plane, and it was also known that a rubber sheet stretched in this way relaxed to a surface quite similar to that of the loop transfer function's amplitude, the rubber surface would be a good helper for visualizing a root locus. Apparently, this view, together with the standard locus rules, helped visualize the locus (and probably encouraged live experimentation with controller variations).

A.3.1 A Short Root-Locus-Variant Categorization

While root locus is normally discussed for rational-function systems, the algorithms we will look at can often handle much more general situations. For this discussion, we will consider locus equations in either the form $1 + kG(s) = 0$ or $D(s) + kN(s) = 0$. Some possible categorizations:

Rational vs. Non-Rational — i.e., whether a finite number of discrete open-loop poles and zeros can be determined. Non-rational systems typically encountered are continuous-time systems

with time delay, such that there are an infinite number of poles and zeros. More theoretical situations may be cases where some exponent is a function of k, such as in a variable discrete-time delay, which may have a term z^k or the like, in which case the number of poles might be variable with k (a difficult situation for most algorithms).

Real-coefficient vs. Complex-coefficient The basic difference is whether the locus will be symmetric about the real axis.

Locus Order This mainly determines how easily an implicit equation (or a set of equations) not containing k can be derived.

A.3.2 A Short Taxonomy of Computer Locus Drawing Methods

Vector-Based The output is a set of points on the locus, and the associated k values at each point. Such methods explicitly follow the path of the roots for various values of k, and must have some algorithm for determining which k to evaluate at.

Root-Finder-Based The most common method. Use is made of a separate root finder (usually a polynomial solver). As such, usually restricted to systems where a simple polynomial can be calculated for any value of k. Thus not typically applicable to continuous-time time-delay systems, but maybe applicable to variations on the z^k example like $z^{\lfloor k \rfloor}$, where there is still a simple polynomial for each k.

Predictor-Corrector Root finding is integrated into the path following. As such, can theoretically follow zeros of almost any kind of function as long as good starting points can be determined. All that is necessary is the ability to evaluate the function whose roots are being tracked (and possibly derivatives, though those can be derived numerically if necessary). Thus theoretically not limited by non-rational, complex-coefficient, or high-order loci. Might be limited by numerics on occasion, and would need special heuristics to handle cases where tracks may start "out of nowhere" or similarly stop (as opposed to appearing or dying by bifurcation or turning, as those are standard issues for these algorithms).

Pixel-Based (the output is picture of the locus as a pixel array) Such methods assign a pixel color to all pixels within some distance of the locus in the s plane. Such methods tend to work best using an implicit function without k in it, as having to deal with k can increase sampling difficulty for some brute-force methods. Antialiasing is usually possible, either as a directly consequence of the algorithm (as in Taubin's algorithm), or via (usually adaptive) supersampling.

Brute Force Evaluate some algorithm at each pixel to determine its color. If an implicit function without k is not available, sample k to see if the locus gets close. This level of brute

force is rarely implemented. More typical is some sort of "Ray Tracing" whereby some knowledge of the locus equation is used to simplify finding whether solutions are near the pixel. Implementations discussed in this thesis are limited to situations where a real polynomial in k can be calculated for any particular point in the plane, and are thus limited to real-coefficient locus equations, but are not necessarily limited by locus order or rationality of G. Some implementations may only implement low-order polynomial solvers, however, and would thus be limited in locus order.

Space Subdivision Use some approximation of distance to the solution set to remove portions of the plane from further consideration and hence accelerate the rendering. Taubin's method (Section A.3.4) is the best example of this type. Such methods work best when an implicit equation can be derived which does not contain k, and hence are more complex for higher-order loci. Otherwise, rationality should not be an issue, though implementations may only expect rational inputs.[6]

Hybrid Output is vector data, but it is derived via some sampling of the the space.

Contour Plotting Requires a real-valued implicit function (usually $\mathrm{Im}(1/G)$). Evaluates the function on a grid and uses standard contour-plotting methods to plot the contours for the value of zero. Since an implicit function is required, higher-order loci are more difficult (though not impossible). Rationality is generally not required, nor are real coefficients.

Slicing Derives simple 1D equations indexed by either $\mathrm{Re}(s)$ or $\mathrm{Im}(s)$, and solves these to find intersections of the locus along the slices. The slice is scanned across the region of interest. Root locations are very accurate along the slices, though the slice locations are usually quantized. Note that this thesis does not present any examples of such an algorithm. The reader is referred to [39] for further information on such an algorithm. Note that the concept of solving for solutions on a 1D scanned line is also the basis for the pixel-based ray-tracing methods as well: the 1D slice is simply in a different direction, and the slice intersections are treated a bit differently.

We will look at examples of these methods next.

A.3.3 Vector-Based Methods: Pole Tracking

Vector-based methods (as opposed to pixel-based, which we will look at in the next section) all tend to be some sort of pole-tracking algorithm, though we will see some hybrid algorithms in a later section that output vector data which are not pole-tracking algorithms.

[6] This is typical for any renderer which doesn't have theoretical limitations on system type: any particular implementation may not accept general functions as input and may thus have additional restrictions which are not inherent limitations of the algorithm.

The basic algorithm in a pole-tracking method can be summarized as

- Choose a value of k
- Find the roots for that value of k
- Repeat until desired range of k is covered.

The two major steps listed above correspond to the two major issues for pole-tracking methods: (1) how to choose k, and (2) how to find the poles. We will classify the algorithms based on how they handle case (2), as methods for handling case (1) can often apply to any pole-tracking algorithm. Note that how k is chosen can depend on various factors. Often graphical considerations lead to a much finer k sampling than would otherwise be necessary, in order to generate smooth locus curves.

The two pole-following types we will look at are:

- Using a numerical root finder. This is the most common method.
- Using a Predictor/Corrector method.

Root-Finder-Based Methods

These methods are the most common and straightforward root-locus calculation methods many MATLAB implementations have been of this type (at least as of MATLAB 5). They have at their heart the use of some root finder (usually a polynomial root finder), which is used to find all the roots for each value of k. A typical algorithm looks something like:

1. Choose the next k value k_i
2. Find the roots at $k_i : r_{0i}, r_{1i}, \ldots r_{Ni}$.
3. Assuming the root finder will not always return roots in the same order across every k transition, compare against the roots found on the previous iteration and match up the closest ones to each other.[7] This will keep the roots from 'jumping' between curves, and allows the curves to be drawn with lines connecting the roots. It is necessary for almost all root finders, since the roots will tend to move through various spatial configurations relative to each other across the locus, and any particular root-ordering method is not likely to maintain the same order for all such configurations.
4. Repeat back to Step 1 until end value of k is reached.

[7] Matlab's `vsort()` function performs such a matching

Note that the third step is there specifically to allow the curves to be drawn with lines connecting the roots (also, any bifurcation/ high-sensitivity-handling in the first step can also be considered partly necessary to make the connecting lines look right). If connecting lines are not necessary, the algorithm could actually reduce to iterating the first two steps, though later steps are still of interest if one does not want to miss locus features.[8]

The first step is usually much more complicated than it appears. Due to the fact that the sensitivity of the locus to k is related to the inverse of the distance of the poles from each other, an "ideal" sampling of k depends on the locus itself, and thus cannot necessarily be predicted. Any method which pre-defines the k locations must necessarily be conservative and grossly oversample k in some areas. The most common "open-loop" choice of k is some sort of logarithmic spacing: $k = 10^x$, where x has equally-spaced values from some minimum (usually somewhere between -5 and -2) to some maximum (usually the negative of the minimum). Again, this is usually only useful for getting a gross view of the locus, and artifacts are common, if not missed features in the worst cases.

The most common artifact is at a breakaway or a breakin, whereby the poles jump from the incoming curve to the outgoing curve well before reaching the break point. Poles move extremely fast in the vicinity of a breakaway, and unless the breakaway k value is determined by some other means, it is essentially impossible to predict a sampling of k that will catch the breakaway in any "nice-looking" way.

However, it is possible to calculate the sensitivity at a point on the locus:

$$\left|\frac{ds}{dk}(D(s) + kN(s))\right| = \left|\frac{N(s)}{D'(s) + kN'(s)}\right| \tag{A.27}$$

or

$$\left|\frac{ds}{dk}(1 + kN(s)/D(s))\right| = \left|\frac{D(s)N(s)}{k[N(s)D'(s) - D(s)N'(s)]}\right| \tag{A.28}$$

Figure A.12 shows the sensitivity functions for an example first-order locus. Note that the two derivations of sensitivity are effectively equivalent.

This equation (for first-order loci) is only accurate for a point that is known to be on the locus (and for its corresponding k value), so again cannot be used to predict a sampling of k. However, it can be used to adaptively vary the rate at which an algorithm moves along k. A possible stepping algorithm may be based on the maximum sensitivity among the current pole tracks:

$$\Delta k = \frac{\alpha}{\max(ds/dk)^\beta} \tag{A.29}$$

[8]There are also philosophical arguments that could be made about whether a collection of points represents a "locus" as well as line-connected points, and can degenerate into discussion about if straight-line connections might actually be misleading in some cases…

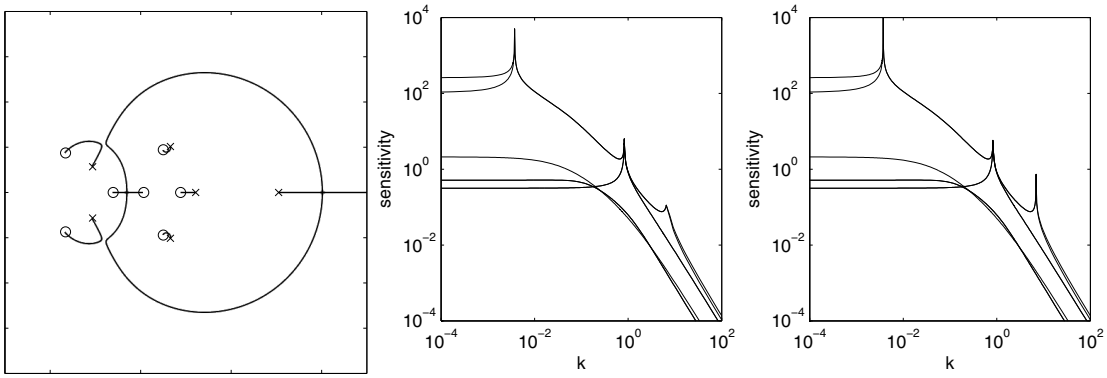


Figure A.12: Root locus sensitivity in k. Left: a first-order locus, Middle and Right: two derivations of the sensitivity functions, plotted for each of the tracks vs. k.

The choice of α and β would be made experimentally, trading off excessive oversampling of the curve vs. the likelihood of missing features and breakaway artifacts. Another possibility is:

$$\Delta k = \frac{\alpha}{(\varepsilon + \max(ds/dk))^{\beta}} \tag{A.30}$$

Which puts a limit on the step sizes when the sensitivity gets very low.

A problem to worry about with this kind of step-size control is "hopping", where when entering a feature where more than three or four poles come together (thus the arrival and departure angles are pretty close to each other), the poles will "hop" to the departing tracks rather than continue into the middle of the singularity. When β is near 1.0, this tends to happen approximately where the distance to the departing track is approximately the same as the current step distance in s. Thus, by making β a bit larger, the step sizes can be made to get smaller faster, and thus get further in towards the center of such singularities.

However, a problem that can crop up if β gets too high is "stopping", whereby Δk approaches too close to zero when the sensitivity gets high, such that the algorithm appears to stop progressing for a large number of iterations. Δk might be limited to some minimum value, or equivalently the sensitivity might be limited to some maximum value.

An alternative stepping algorithm, which the author is particularly happy with, is to use a graphical measure for determining the sampling of k: sample k such that the curves are "smooth". This is usually implemented via adaptive subdivision of k:

1. Choose next k by some algorithm (sensitivity, etc), which does not need to be very fine, just enough to not completely miss small features.

2. Find all roots at the new k, and sort them to line up with the previously-found roots.

3. For each curve, calculate the angle between the line segment consisting of the new point and the previous point, and the segment consisting of the previous point and the one before.

4. If the angle is too far from 180 degrees (according to some threshold), assume that it will be a visible "kink", so recursively pull k back towards the previous value until the angle ("curvature estimate") is within the threshold, or if the line segment length falls below some other threshold, or if some recursion depth is reached.

Not only will such an algorithm tend to produce only as many samples of k as are necessary to achieve smooth curves, it will automatically recursively track down breakaways and the like (since they are hard corners, the recursion will keep narrowing down to the exact location of the corner, until interrupted by one of the other recursion-breaking limits), giving clean, artifact-free renderings of those difficult features.

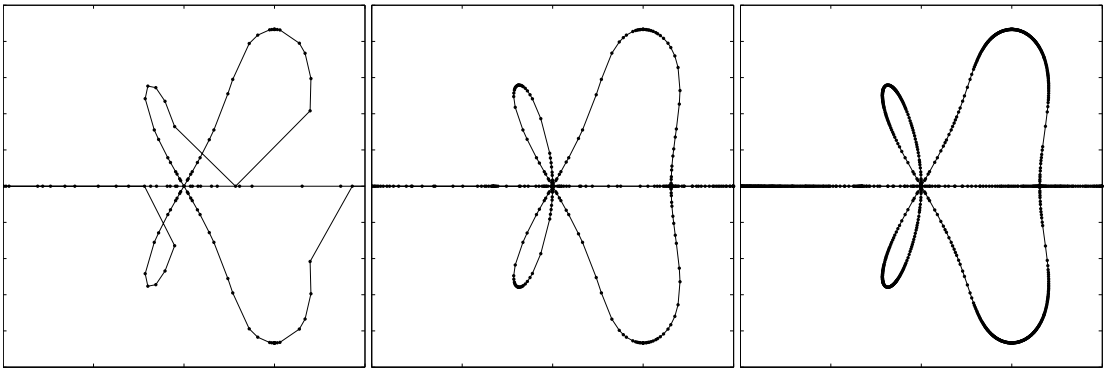


Figure A.13: A 2^{nd}-order locus rendered with a root-finder-based method using adaptive k subdivision, with three different settings of the adaptation tolerance. Left: very loose, Middle: medium, Right: very tight. Visited/rendered points are marked with dots.

Figure A.13 shows three settings of subdivision tolerance in a root-finder-based rendering method, with the visited points marked to show the tradeoff in smoothness versus the number of points.

Note that the problem of parameter-stepping is also an issue in general predictor-corrector path-following (homotopy and continuation, etc). As such, further discussion of step-adaption algorithms can be found in much of the literature (see the intro to the predictor-corrector section, next).

One potential drawback of root-finder-based methods would arise if the root finder had numerical instabilities, due to extremely high-order polynomials or the like. Depending on the numerics of the algorithm, a predictor-corrector-based algorithm, which is based purely on "forward evaluations" of the polynomials rather than repeated polynomial divisions (as in some root-finders) may have a higher probability of giving cleaner results.

Predictor-Corrector Methods

Predictor-Corrector is the most common technique in a field called *numerical continuation*, which is the problem of taking a solution for a simple problem and numerically evolving it to a solution to a difficult problem through a number of steps. This can also describe in general the concept of homotopy, where the solutions (or roots) of a difficult system are to be found by creating a function (the homotopy) which smoothly interpolates between a simple system and the complex system, and then numerically following the solutions of the homotopy as the interpolation parameter moves from describing the simple system to describing the complex system.

As the general problem is one of following solutions/roots of a system as a parameter changes, these techniques apply directly to root-locus drawing. The primary difference being that whereas the goal in homotopy continuation is the determination of the final roots, the goal in root-locus rendering is the determination of the whole path. As such, whereas the algorithm may be able to successfully track the solutions at rather large step sizes, we may intentionally want to pull back the stepping to produce a clean rendering of the path (indeed, the curvature-based adaptive step subdivision algorithm described in the previous section applies directly in predictor-corrector as well).

Some good references on numerical path following can be found in the works of Allgower and Georg ([7] [6] [8]). A typical paper on step control is Kearfott and Xing [139].

Some of the uses of continuation which end up somewhat similar to root locus are the solution of nonlinear equations [136], and the solutions of systems of polynomial equations ([177] [179] [178]).

An alternative method for continuation is called pivoting, whereby the solution space is tiled by triangles, and the path following occurs by determining which vertices the solution crosses as the continuation parameter increases, following the path around by tracking the triangles which are visited. From the standpoint of the root-locus-method taxonomy, pivoting is more of a hybrid technique, since the triangles are effectively a sampling of the space. Gunji et al ([101] [141]) apply a pivoting method to the solution of polynomial systems.

Stonick and Alexander [253] have applied a homotopy continuation to the design of IIR filters (there is some interest in the use of homotopy in non-convex optimization in general). In fact, while we're on the subject of optimization, interior-point methods in convex optimization are a form of homotopy where a simple-to-minimize error surface is morphed into the desired error surface while the minimum is tracked with predictor-corrector methods[25].

Use in root locus As mentioned, numerical continuation has an obvious application to root locus, and several papers on the subject have been published.

In 1968 Ash [10] described an implementation of a rather successful predictor-corrector locus renderer, and later O'Donell and Frederick [192], [193] described some extensions to the algorithm

and created a MATLAB port.

In 1970, Mitchel et al [169] described following the path of a locus, though it appears as though the intent was more for control-system analysis than for drawing the locus.

In 1978 Pan and Chao [199] described a predictor-corrector root-locus renderer, but approached from the concept of turning the locus into a differential equation and numerically integrating it using a predictor-corrector method.

Nishiolka et al [191] described a pivoting-based path follower for root locus.

Alexander and Stonick [5] applied continuation to track the zeros of time-varying polynomials, which is not far from tracking a root locus. In an earlier paper [252], the same authors relate an RLS adaptive-filter update to a homotopy continuation.

Algorithm: The basic predictor-corrector algorithm is as such:

1. Assume we are on the path at the current k
2. Step k to the next value
3. Foreach active path: predict, from the previous samples along the path, an estimate of the next sample location, and then correct it numerically, using a local zero/minimum finder (Newton-Raphson is most common). The prediction step is used because Newton-type finders are more accurate and converge faster the more accurate their starting points are: so a good prediction will make the algorithm run faster and allow larger steps in k.
4. Add corrected points to the active paths and repeat until k_{max} is reached.

Since Predictor/Corrector is itself a root-finding method, one might ask why it is presented as separate from the root-find-based methods. The main idea is that the methods from the previous section start the root-finding from scratch for every k, so that the algorithm is purely one of choosing the k and applying some heuristics to make sure we associate the correct roots to each other from k to k. Predictor/corrector, however, uses the results of the previous k iteration explicitly to assist in the finding of the roots for the next iteration. The algorithm makes use of the fact that the roots move in a more or less continuous fashion with respect to k to follow them around as k changes.

As such, the root-finding step in a predictor/corrector algorithm is generally much simpler than in the general algorithm, as in a sense the whole algorithm is the root finder. On the other hand, if properly initialized,[9] a predictor/corrector can track roots of systems for which there may not be a general root finder (many non-polynomial systems, for example).

[9] And if the number of roots stays constant, or new roots only appear via bifurcation of existing roots rather than appearing arbitrarily in space

One can think of a predictor/corrector method as "bringing the locus-tracking algorithm inside the root finder," whereas the root-finder-based methods have the locus-tracking algorithm outside the root-finder.

A predictor/corrector can even be applied to systems with infinite numbers of (distinct) roots (such as continuous-time delay systems), if the user is content with tracking a finite subset of the roots (presumably this would work best when there is a finite number of roots within the region of interest, as in the case of continuous-time delay systems, where the infinite number of roots exist as series of distinct roots heading out to infinity, so that there are a finite number of roots within any finite region of the s plane.).

The choice of k is integral to a predictor/corrector method: it must step roughly monotonically through space, and it must not step too far so as to cause problems with the corrector. Thus the visiting of k values cannot be completely random, as it theoretically could be in a generic root-finder-based method. However, as mentioned earlier, there may be graphically-influenced situations where k may be required to be sampled even finer than the predictor/corrector needs to maintain its tracks. Such situations may even require backtracking of k (as might occur with an adaptive-subdivision curve-smoothness algorithm). If the algorithm remembers root and k values, it could be temporarily restarted at some other point in "spacetime" to refine the locus there, or backtrack the history of some newly-fond root track, etc.

Another interesting feature that can be implemented in predictor/corrector methods (though not all methods do this) is that the k values do not have to be locked together for all the tracks, since the root-finding is actually happening independently at each root. A common annoyance for implementors of root-finder-based algorithms is when a 'boring' track gets a bunch of extra roots on it because another root was dong something interesting at the time (such as encountering a split, or even heading out to infinity). A predictor/corrector can theoretically independently adapt k along each track. It becomes tricky, however, when tracks meet, if they are on different k schedules. Such an algorithm would need some good computational geometry algorithms to identify meetings in such situations. More likely, an algorithm will tend to step all tracks together, but then possibly do any adaptive subdivisions or backtrackings locally to specific tracks.

An implementation for root-locus rendering can easily become "a stack of heuristics", though how far that goes is up to the implementor. Some possible added heuristics:

- The previously-mentioned adaptive subdivision of k for curve smoothness can be considered a heuristic.
- Handling of bifurcations (breakaways, breakins, other multiplicity situations). Ash [10] detects them by catching sudden changes in the motion angle of a track, then applies a special newton search for the exact break value of k. Other more heuristic methods may simply note that multiple tracks have corrected to the same point (which typically happens around multiplicities) and take some corrective action (such as placing prospective starting points at

perpendiculars and re-trying the corrections). It has already been noted that as long as the roots can stay on the tracks, a smoothness-based k subdivision algorithm will do the rest of the work in tracking down to the center of a multiplicity.

- Simple bifurcation tests such as checking a circle of prediction points around a test point to see if all correct to the same location or not.
- Letting tracks which leave some expansion of the area of interest die rather than continuing to track them. The intent of such a heuristic is to not have to track asymptotes out to infinity. A downside is that a track may leave the ares of interest but later come back, which makes the next heuristic necessary.
- Periodically testing for "new" tracks which aren't currently being tracked. A typical method would be to randomly seed predictor points in space and see if any correct to locations that are not on currently active tracks. Such a technique may be of use in tracking systems where the number of active poles may vary with k, or in tracking infinite-root non-rational systems, where additional roots might come in from outside the region of interest. Since k can be unlinked from tracks, a newly-recognized track can be traced backward to narrow down its point of arrival or to see if it joins up with any previously-killed tracks (or any active tracks). This, however, presents a tricky problem if the k values don't line up exactly: how to decide if two sets of points are members of the same curve or not... (this will not be discussed in detail in this thesis).
- A "reality check" heuristic that checks to see if corrected points lie abnormally far from the predicted point and the current track. This can help catch potential problems with the unpredictability of Newton's method when starting points are "between" roots.

Newton's method is the typical corrector in such algorithms. However, there can be one situation where it can pose problems: breakaways from the real axis. Let's say two roots at step $n-1$ were real, but at the next step they are complex. The predicted values for the roots are likely to still be on the real axis (or very near it). However, in such situations, the Newton steps for those locations with the new coefficients can very easily lead to a completely different area of the space. This can be illustrated with a Newton Fractal of such a situation (i.e., a map of which roots a newton iteration ends up at), as in Figure A.14: The image color-codes (or in gray) the roots at which various starting-points in the plane end up after a number of iterations of Newton's method. Very little of the area on or the real axis actually ends up either of the two complex poles (at the top center and bottom center of the image. The most likely ending point is denoted by the medium-sized area in the middle of the image, which corresponds to a root that is neither of the two roots that "should" be corrected to.

This can be understood also from the fact that in the Newton's method iteration, if the polynomial coefficients are real, and if the starting point is real, then the iteration is going to stay real: there

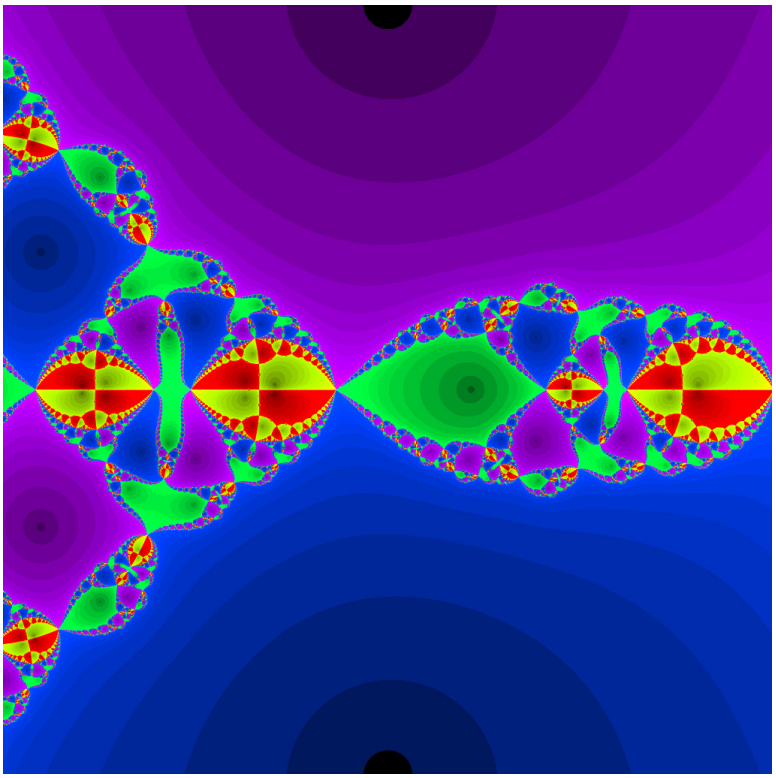


Figure A.14: Detail on the Newton-fractal of a situation where two complex poles have just split off the real axis. The previous locations (and hence their predicted locations) are still on the real axis. This image color-codes the roots at which various starting-points in the plane end up after a number of iterations of Newton's method. Very little of the area on or the real axis actually ends up either of the two complex poles (at the top center and bottom center of the image).

is no way to get off the real axis! if there is any other root on the real axis, it will be the most likely one found. Only some heuristic which specifically moves the starting point off the real axis can ever find the roots which just broke away. Such heuristics may be the bifurcation-test noted above, or some heuristic that recognizes that two poles are on a head-on path, and hence determines that a breakaway must happen, and creates predicted points off the axis (probably at perpendiculars) in order to look for the breakaway points. Such a heuristic can be aided by the tendency of the velocities of the departing poles to be similar to the velocities of the arriving poles, as a function of their distance from the breakaway point, so by looking at the trends in the points leading up to the breakaway, prospective breakaway points can be predicted sufficiently far from the real axis as to give a high probability of correcting to the desired roots.

Note that there is no such difficulty with breakins to the real axis, only with breakouts from the axis.

A.3.4 Pixel-Based Methods

Whereas pole-tracking methods find actual pole locations and plot them, pixel-based methods directly calculate an image of the root-locus, not necessarily determining actual root locations at any point in the rendering. This drawback is often offset by desirable features such as fast or consistent-time rendering, simpler algorithms, or attractive results.

Chen and Chen [39] introduce a method for rendering loci by scanning a line across the region of interest and finding the intersections of the locus with the line as a one-dimensional real root-finding problem. However, since the intersections can be considered exact, their method is more correctly placed in the Hybrid methods category.

Many methods can be considered to be some form of implicit-curve or implicit-curve rendering algorithm. This topic (especially that of rendering implicit surfaces) has received quite a bit of attention in the computer-graphics field. An introduction to the field can be found in Jules Bloomenthal's book *Introduction to Implicit Surfaces* [20]. The most common areas of application are the rendering of level sets in measured or computed volumetric data, such as medical CAT scans, PET scans, MRI's etc., or the results of large-scale fluid dynamics simulations.

On a related note, Tupper applied interval math to the problem of rendering implicit curves and areas (defined implicitly by inequalities rather than equalities) in his Master's thesis [272].

Taubin's Method

In 1994, Gabriel Taubin, who had been working on the problem of fitting implicit curves and surfaces to point data [261] [258] [259]), derived a set of approximations of the distance from a point to the curve/surface which worked quite well and were not terribly expensive. As such, he realized that these approximations could be used in an adaptive space-subdivision rendering method for drawing the curves.

In [260] and [262], Taubin presented a fast algorithm for rendering an image of an implicit function in the plane. This method turns out to be very efficient for rendering root loci, especially when arriving at an image of the locus is more important than calculating the exact locus values. Implementations of this algorithm have proved fast enough to animate in real-time, and to use as the basis for an interactive root-locus explorer that the author has written, whereby users can drag open-loop poles and zeros about the plane and see the effect of the changes live as the root is dragged, along with other live exploration capabilities.

Taubin's method is applied to first-order root locus rendering by using it to render the implicit function $\mathrm{Im}(D/N) = 0$. Or if explicit numerator and denominator polynomials are not available, $\mathrm{Im}(GH) = 0$, where GH is the total loop transfer function (not including k), from the basic loop shown in FigureA.1, which has H in the feedforward path, and G in the feedback path. As such, the method can be directly used to compute any locus that is first-order in k, and for which GH or D/N can be calculated. Therefore they do not necessarily have to be polynomial.

Taubin's method is to recursively subdivide the plane, using an approximation of the distance to the solution curve to decide when to stop subdividing a particular region. Then to paint a pixel as being on the curve when the recursion gets down to the individual pixels and the distance from the center of the pixel is within half the desired line width (note that since the distance (estimate) is known at the pixel level, antialiased images are extremely easy to create, simply by painting gray scales which ramp down from "full-on" to "zero" over about one pixel at the edge of the linewidth.

In [260], Taubin derives 1st-order, 2nd-order, and nth-order approximations of the distance, and discusses the vailidity of the approximations by deriving bounds, etc. Practically, it is demonstrated that using the 2nd-order approximation works very well in practice, is not terribly expensive to calculate, and only badly overestimates quite rarely (overestimation of the distance could cause a subset of the space to be incorrectly discarded from consideration in the recursive subdivision, and thus possibly cause missing pieces in the rendering of the implicit curve). In the author's own implementations of the algorithm, such gaps when they occur are not considered to be fatal.

The 1st and 2nd-order distance approximations [260] are:

$$\delta_1(p, Z(f)) = \frac{|f(p)|}{\|\nabla f(p)\|} \tag{A.31}$$

$$\delta_2(p, Z(f)) = \sqrt{\frac{\|F_1\|^2}{4\|F_2\|^2} + \frac{|F_0|}{\|F_2\|}} - \frac{\|F_1\|}{2\|F_2\|} \tag{A.32}$$

Where p is the point in question, f is the implicit function, $Z(f)$ is the zero-set of f (i.e., the solution of $f(p) = 0$). F_0, F_1, and F_2 are the vectors of n^{th}-order partial derivatives of f. If $p = (x, y)$, then:

$$\begin{aligned} F_0 &= f(x,y) \\ F_1 &= \nabla f(x,y) \\ F_2 &= \left[\frac{\partial^2 f(x,y)}{\partial x^2}, \frac{\partial^2 f(x,y)}{\partial y^2}, \frac{\partial^2 f(x,y)}{\partial x \partial y}, \frac{\partial^2 f(x,y)}{\partial y \partial x}\right] \end{aligned}$$

(See [260] for discussion of higher-order distance estimates).

Now, instead of symbolically working out these partial derivatives for $\mathrm{Im}(D/N)$, we estimate them numerically using finite differences. Further, we use non-symmetric differences to reduce the number of function evaluations, both of which work fine in practice (when using double-precision floating-point math):

$$\begin{aligned} v &= f(x,y) \\ v_x &= f(x+\delta, y) \\ v_{xx} &= f(x+2\delta, y) \\ v_y &= f(x, y+\delta) \end{aligned}$$

$$
\begin{aligned}
v_{yy} &= f(x, y + 2\delta) \\
v_{xy} &= f(x + \delta, y + \delta)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial f}{\partial x} &\approx (v_x - v)/\delta \\
\frac{\partial f}{\partial y} &\approx (v_y - v)/\delta \\
\frac{\partial^2 f}{\partial x^2} &\approx (v_{xx} - 2v_x + v)/(4\delta^2) \\
\frac{\partial^2 f}{\partial y^2} &\approx (v_{yy} - 2v_y + v)/(4\delta^2) \\
\frac{\partial^2 f}{\partial x \partial y} &\approx (v_{xy} - v_x - v_y + v)/(4\delta^2)
\end{aligned}
$$

The use of numerical estimates for the partial derivatives allows the algorithm to be quickly and easily applied to a variety of functions without requiring the implementor to work out all the derivatives symbolically.

Further, in order to have a less expensive $f(x, y)$, we use in the first-order locus case just the numerator of the expansion of $\mathrm{Im}(D(s)/N(s))$, which is:

$$f(x, y) \to f(s) = D(s)N(s^*) - D(s^*)N(s) = 2\mathrm{Im}(D(s)N(s^*)) \tag{A.33}$$

Where the star denotes the conjugate operation.

The algorithm thus as shown in Algorithm 4. In practice, there are a number of places the algorithm can be tweaked. For example, the factor x used in the recursion decision can be varied to make the decision more conservative and lower the probability of holes in the image. Since the approximate distance to the curve is directly available, and it is usually quite accurate once we get down to the pixel level, we can use the distance to antialias the image, simply by calculating some ramp of pixel intensities between full-on at some distance to full-off approximately half a pixel further away. Pixels on the edge thus get "feathered" pixel values which give a nice smoothing of the curve.

This algorithm is particularly nice in comparison to the vector algorithms due to the fact that it automatically renders locus features that the other algorithms have difficulty with (like breakaways, multiplicities, and smooth curves) without requiring any special heuristics (aside for any added to reduce gap probability). As such it is a very "elegant" algorithm, and the output tends to look very clean.

The algorithm is also of interest because the algorithm is very fast. As Taubin notes in [260], the number of visited point approaches $O(n)$ in the horizonal (or vertical) resolution, rather than $O(n^2)$. Further, this cost is relatively unaffected by the complexity of the locus being rendered.

Algorithm 4: Taubin's Method Root Locus Renderer

Data: Polynomials *num*, *den*, rendering range $[x_{min}x_{max}y_{min}y_{max}]$, image width N (in pixels), line width δ (in pixels)
Result: Image of the rendered locus

```
function recursiveDraw (c_x,c_y,width)
begin
    d = estDistance (c_x,c_y,num,den);
    if d < x width then
        if width ≤ pixelwidth then
            drawPixel (toPixelCoords (c_x,c_y));
        else
            recursiveDraw (c_x − width/4,c_y − width/4,width/2);
            recursiveDraw (c_x − width/4,c_y + width/4,width/2);
            recursiveDraw (c_x + width/4,c_y − width/4,width/2);
            recursiveDraw (c_x + width/4,c_y + width/4,width/2);
        end
    end
    // else recurse no further in this quadrant
end
```

In particular, areas of high curvature or root-crossings, which take longer to handle in the vector algorithms due to the need to supersample k, are rendered at the same cost as more mundane loci. While it is true that higher-order polynomials take a little longer to evaluate than low-order polynomials, the difference in rendering time is often unnoticeable between low-order and high-order polynomials.

In [260], Taubin discussed extensions to the algorithm, mainly to further reduce the probability of gaps in the curves. The locus renderers used in this thesis based on Taubin's method have not kept pace with his extensions, mainly because the extensions cannot be implemented in a depth-first recursion (they require breadth-first implementation, because they use information from neighboring regions for the decision to recurse, which can only exist at the correct time in a breadth-first algorithm). Since the author's implementations used a depth-first implementation, and occasional gaps were not considered to be a terrible problem, the root-locus renders have not made use of Taubin's later algorithm extensions.

Extending Taubin's Method to Higher-Order Loci On first look, it would appear that this method can only be used for loci which are first-order in k. As such, the author did some exploration into interpreting the locus in three-dimensional space $(\mathrm{Re}(s), \mathrm{Im}(s), k)$, which produced some interesting results, which we will look at shortly, but not great "working" locus plots. However, as discussed in Section A.1.3, the locus is also the union of the solutions of $r_i(s) = 0$, where r_i are the individual roots of the higher-order locus equation in k. We will follow that up after first looking at the

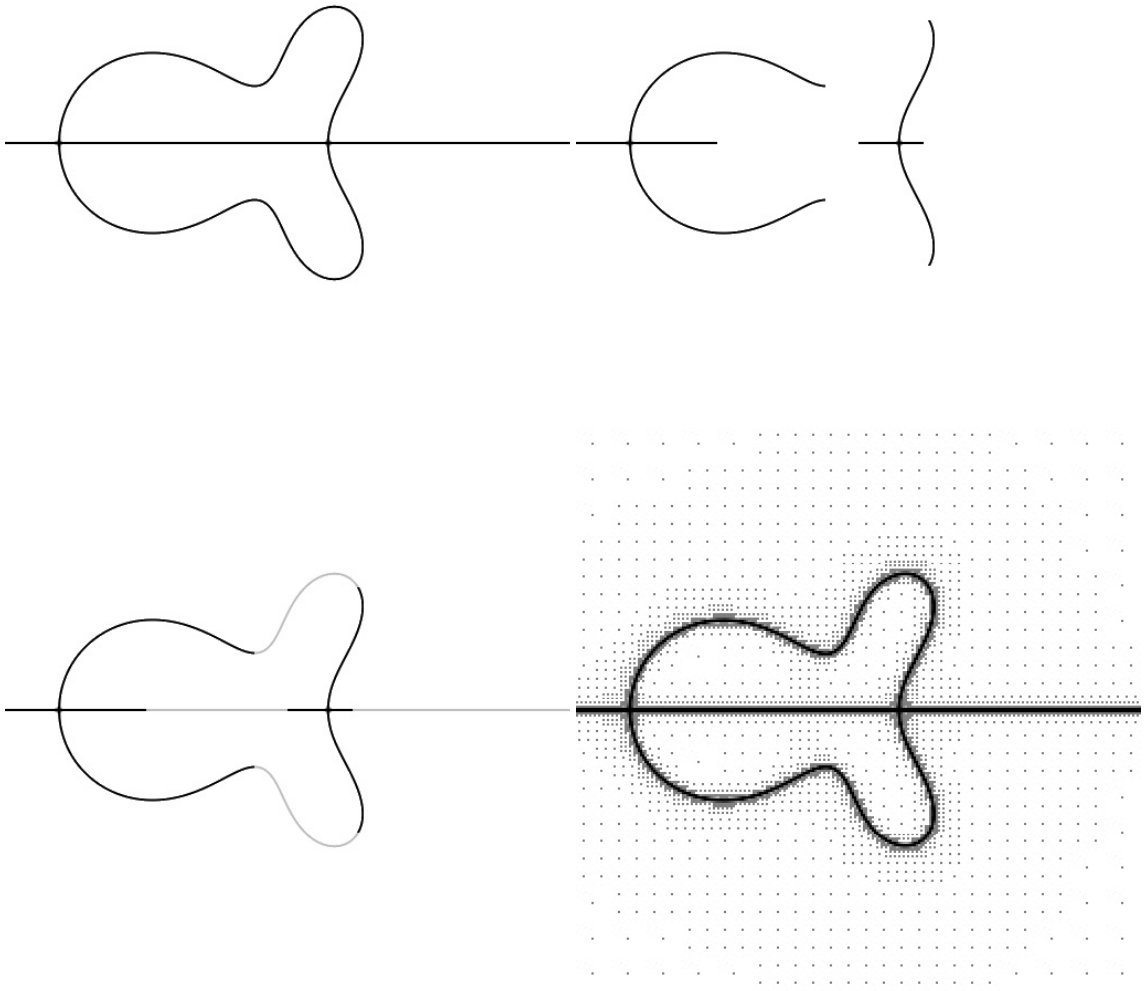


Figure A.15: First-order locus $(s^4 - s/10) + k(s^3 + 1) = 0$ rendered using Taubin's-method renderer $(-2 < \mathrm{Im}(s), \mathrm{Re}(s) < 2)$. Top Left: Full locus (both signs of k). Top Right: 180° locus. Bottom Left: positive and negative k denoted by gray levels. Bottom Right: debugging plot showing visited points in the recursion (note that this particular renderer is conservative about discarding space, so it visits more points than might be necessary).

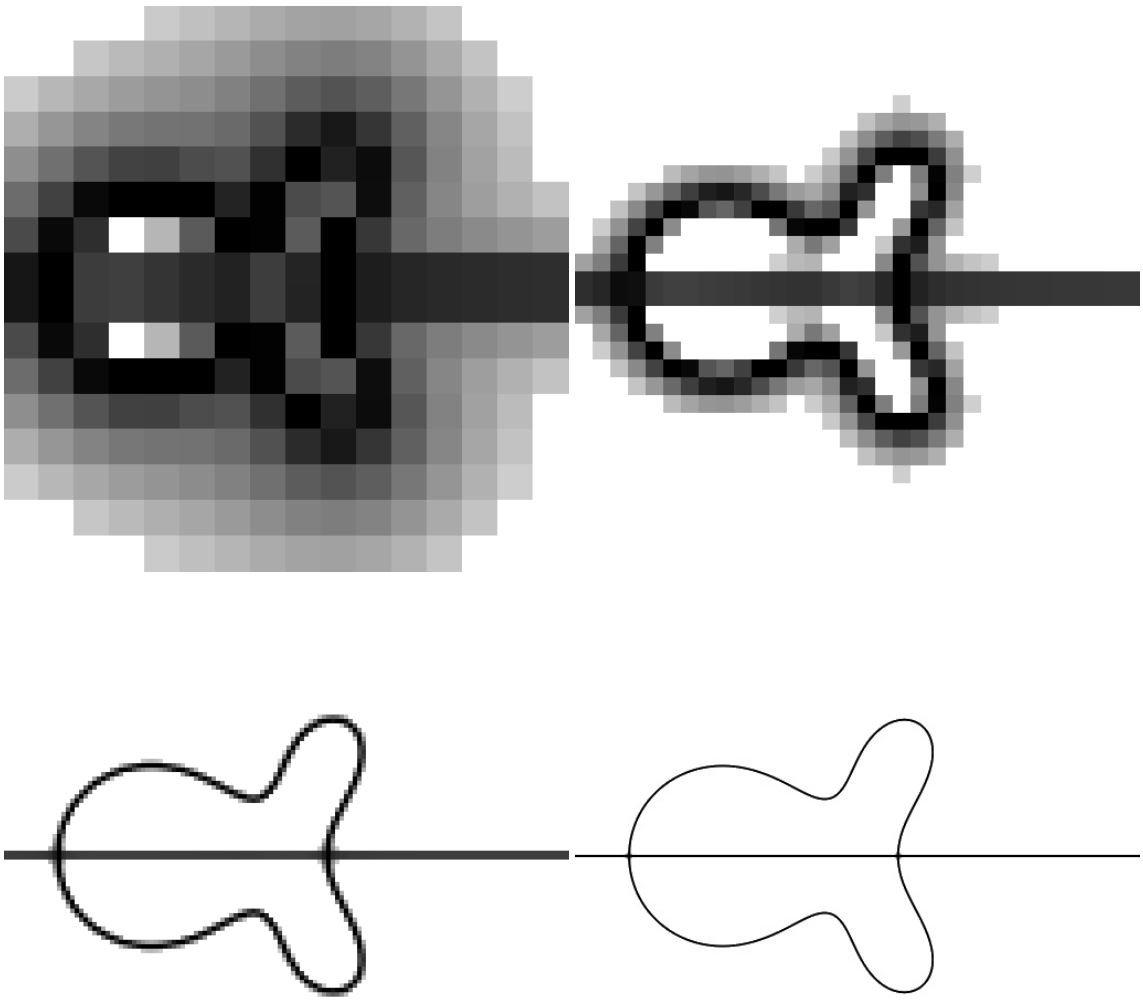


Figure A.16: Taubin's Method: various resolutions. Top Left: 16x16. Top Right: 32x32. Bottom Left: 128x128. Bottom Right: 512x512. Note how distance approximation start breaking down at lower resolutions (especially noticeable is the antialiasing spreading out beyond one pixel's width).

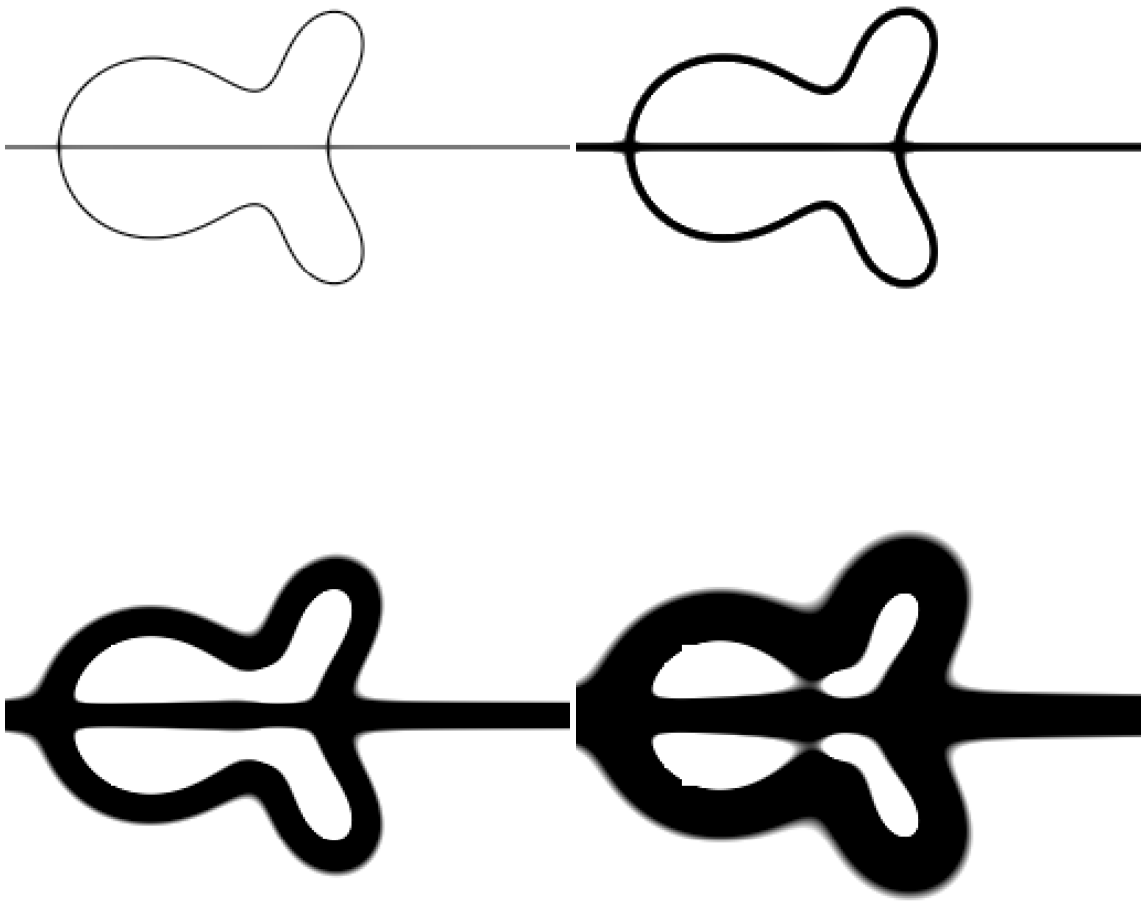


Figure A.17: Taubin's Method: various line thicknesses (256x256). Top Left: 0.1 pixels, Top Right: 5 pixels. Bottom Left: 20 pixels. Bottom Right: 30 pixels. The previous two figures used the default linewidth of 2. Note how distance approximation also breaks down as the thickness gets larger (especially noticeable: antialiasing spreading out). Also note that the algorithm does not render extremely thin lines as disappearing, due to the particulars of the implementation of the antialiasing algorithm.

three-dimensional explorations.

Taubin's Method in 3D A root locus of any order in k can be considered an implicit function in three dimensions, simply by making k the third dimension. Taubin's method easily extends to three dimensions [262], by subdividing the space in "oct-tree" fashion [84].

Unfortunately, the method works best when dealing with a region in space which covers nearly the same range in each dimension, and when rendering a function which does not get extremely sensitive to one of the dimensions. Unfortunately, root-loci break both of these: k theoretically ranges between $\pm\infty$, and even in a more practical range, it tends to cover a significantly larger range than the range of the s plane we tend to be looking in. As such, it is hard to define a visually correct "pixel" in (s, k) space (in other words, when subdividing the space, when do we decide that we have a "small enough" range of k). Further, due to the binary subdivision of the space, we would prefer that we reach our minimum k at the same recursion depth as we reach the pixel size in the s axes. The result of breaking this is that the distance to the center of a "pixel" becomes increasingly inaccurate (or at least unusable) as the "depth" of the pixel (in the k dimension) gets much larger than the width of the pixel in the s plane, and so the decision on when to end the recursion becomes inaccurate.

Further, in high-sensitivity regions (like where roots come near each other), we know that the roots move very quickly with k, hence the implicit curves become nearly perpendicular to the k axis in these regions. The outcome of which is that the distance approximations (or the ways that they are interpreted) start to break down, regardless of the issues in the previous paragraph. The resulting renderings no longer resemble a family of curves, but rather a family of (sometimes very large) "blobs", and the algorithm turns out to not be particularly useful as a locus plotter for these situations. However, the images that are produced are extremely interesting visually (Figures A.18 and A.19). In the process of exploring this rendering method, the rendering was tweaked to treat each "blob" as though it were an approximation to a sphere or ellipsoid, and to shade it as though illuminated with a light source (using the gradient of the distance approximation as the surface normal), then draw it transparently, which led to the most interesting images (also see the Gallery at the end of this thesis).

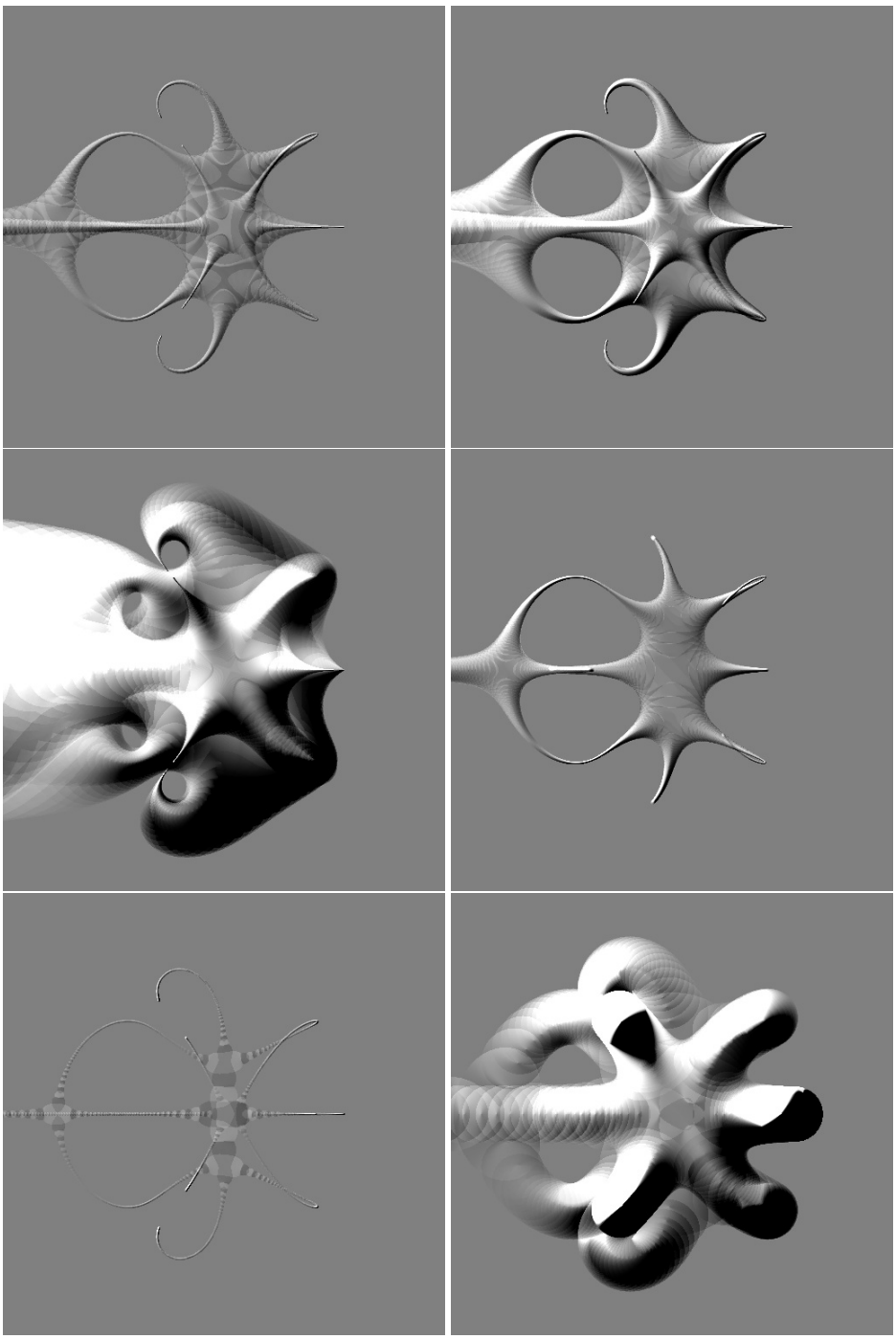


Figure A.18: Attempting Taubin's method in 3D on the 2^{nd}-order locus $(s^3 - s^2/2 + 1) + k(s^3 - s^2/2 - 1) + k^2(s^4 + 2) = 0$. Various tradeoffs on scaling the k range versus the s range and resolution.

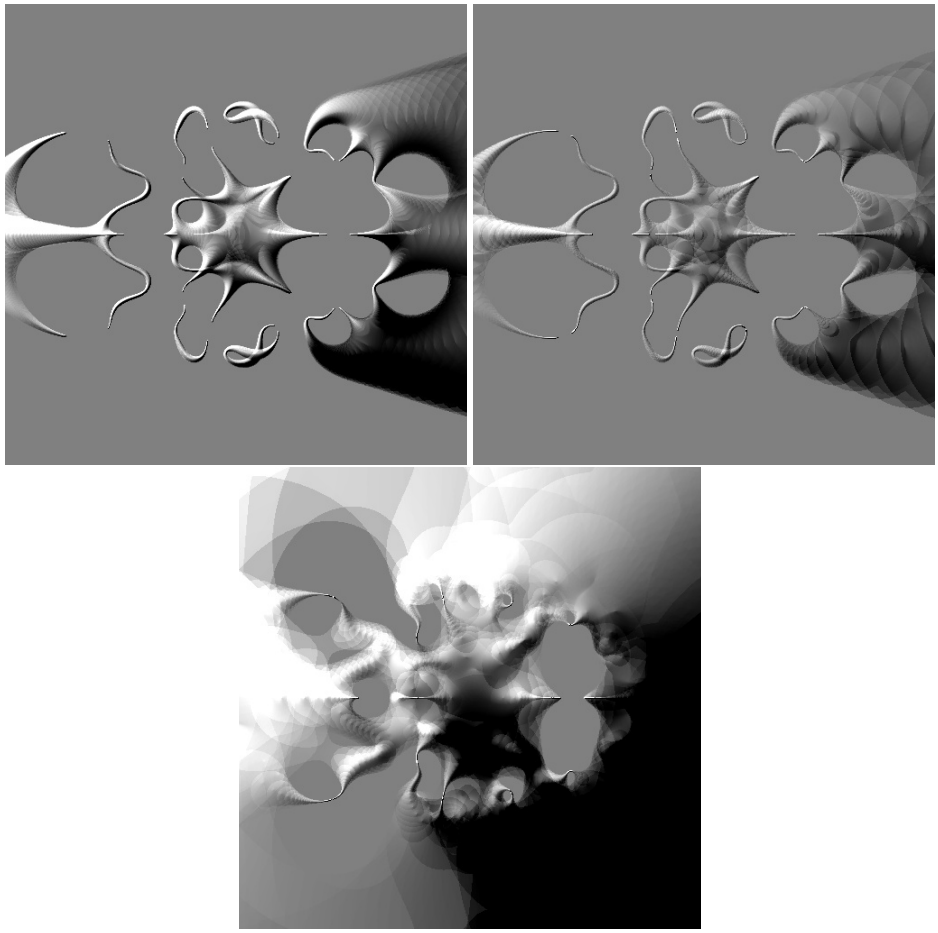


Figure A.19: Attempting Taubin's method in 3D on the 2^{nd}-order locus, D, N_1, N_2 set to a particular random 15^{th}-order locus. Note how the "blobs" can end up filling the plane.

Back to 2D However, it is not necessary to try to render in 3D at all. As noted in Section A.1.3, a 2^{nd}-order locus can be defined as the union of the solutions to two 2D implicit equations. As such, Taubin's method need only take the minimum of the distance approximations for the two implicit equations Eq. A.13 and Eq. A.14, and it will render the union of their solutions. In fact, most the locus renderings in that discussion were done using Taubin's method modified in just this way. A 2^{nd}-order version of the Root-locus explorer has been implemented, which was quite handy in developing intuition on the different behaviors of 2^{nd}-order loci, and directly helped in the understanding of the way the Chamberlin state-variable filter varies its Q.

As discussed later in Section A.1.3, this concept extends to higher-order loci as well. A fourth-order renderer using this method has been implemented and it's output is as expected (for example, Figure G.9 in the Gallery). However, due to the necessity of including calls to a general polynomial root finder (ass opposed to using the closed-form versions of the quadratic roots as for the 2^{nd}-order locus rendering), it runs significantly slower (as much as a factor of 10x), and hence could not be used to make a 4^{th}-order root-locus explorer (instead a version based on root-finder-based path follower with adaptive subdivision of k was used, with a bit less success).

Note that when using numerically-calculated estimates of the derivatives, the 2^{nd}-order version of Taubin's method evaluates $N_1(s), N_2(s)$, and $D(s)$ for each test point, such that all that is left algebraically is the 2^{nd}-order equation in k which is solved using the quadratic formula. Thus, the orders of N_1, N_2, D have a rather small effect on the algorithm, as they are merely numerically evaluated rather than solved. Similarly, when going to higher order loci in k, such as 4^{th}-order, $D(s)$ and $N_1(s)$ through $N_4(s)$ are merely evaluated at the test points, and the implicit functions are based purely on the roots of the 4^{th}-order polynomial in k, which now need only be numerically solved.

Examples of where the method breaks down. There are systems where Taubin's method breaks down. For example, the 2^{nd}-order locus:

$$D(s) = s^2$$
$$N_1(s) = s$$
$$N_2(s) = 1/2$$

One may recognize this as an "X" locus from earlier in this appendix. Taubin's method can render this locus as in Figure A.20 Often, just a slight modification of the system removes the problem, as seen in the Figure. The theory for this behavior is that in this case, the denominator terms of the 2^{nd}-order distance approximation that is being used in the rendering are all nearly zero (actually theoretically they are zero, but the numerical approximations to the derivatives give numbers that are slightly nonzero), such that theoretically, the distance approximation is zero everywhere

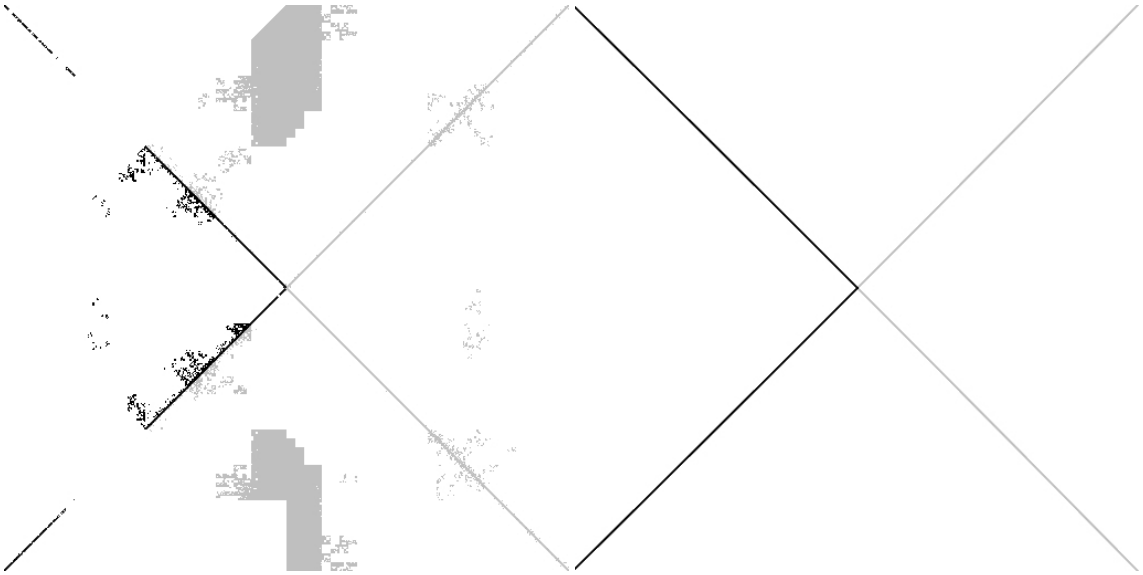


Figure A.20: Problems with certain loci in Taubin's method. Left: $D(s) = s^2, N_1(s) = s, N_2(s) = 0.5$. Right: $N_2(s)$ changed to $.001s + 0.5$.

(in which case the whole plane should be painted). Slightly shifting the polynomials breaks up the symmetry that caused the problem and hence the result is much cleaner. It is also noted that performing a change of variables on k (as in $k \leftarrow k + a$) can also clear up the drawing in some cases).

Rendering Warped Loci Using Taubin's Method Since our implementations are using numerically-approximated derivatives in the distance approximations, we are free to implement variations which would otherwise be quite difficult. For example, we can draw loci with warped axes. Rather than rendering

$$f(x, y) = 0 \tag{A.34}$$

We can render

$$f(g(x, y), h(x, y)) = 0 \tag{A.35}$$

All the chaining of the partial derivatives can be taken care of by the numerical approximations.[10]

As such, we can render a locus with $(\log_{10}(f), \log_{10}(Q))$ axes, as opposed to $(\mathrm{Re}(z), \mathrm{Im}(z))$ axes:

$$\begin{aligned} \theta &= \frac{2\pi 10^f}{f_s} \\ r &= e^{-\theta \tan(\sin^{-1}(10^{-Q}/2))} \end{aligned}$$

[10] although a numerical analyst would surely protest.

$$z = re^{j\theta}$$

where the derived z is used to in the locus distance test rather than f and Q. Examples of such transformed loci are shown in Figure A.21 and Figure A.22. Of course, numerical problems can be

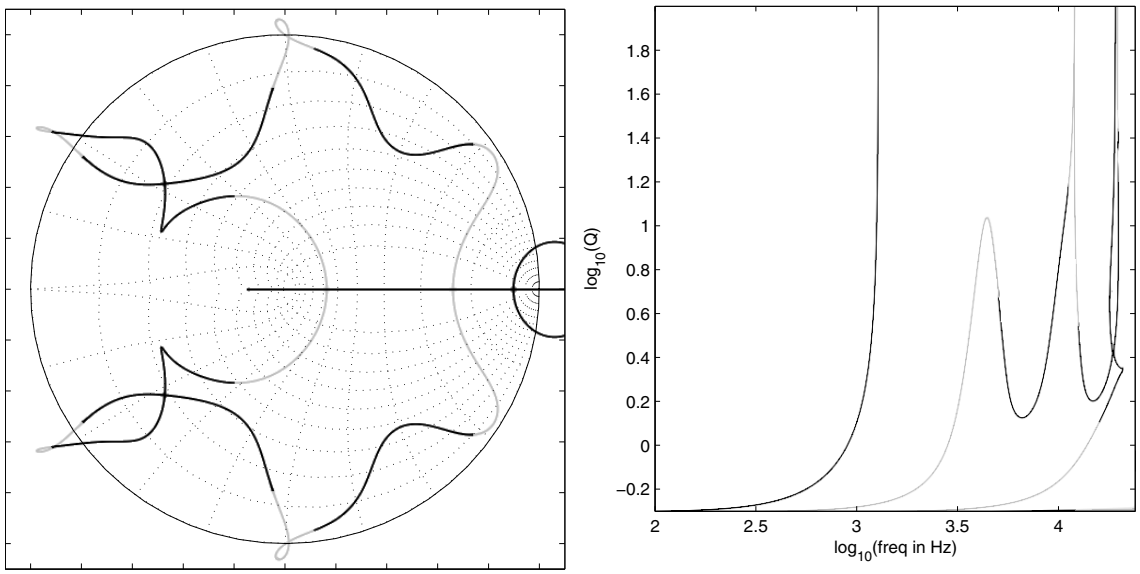


Figure A.21: Left: a random 2nd-order root locus. Right: the same locus rendered using warped axes (freq, Q) (assuming $f_s = 48kHz$).

exacerbated by such mapping. Figure A.23 shows an example of a locus that has problems in the distance approximation, together with the (freq,Q) rendering of the same locus. As one can see, the numerical problems near $z = 1$ (i.e., the whole left-hand side of the (freq,Q) graph), which are not particularly visible in the s-plane locus, (which has its obvious problems elsewhere), are quite visible in the warped locus.

Ray Tracing Distance Approximations

These are more brute-force methods, but for a while it seemed like they would be necessary (until it was realized how to handle higher-order loci in 2D with Taubin's method). These methods work by casting rays parallel to the k axis for each pixel and looking for intersections with the locus (or more correctly, intersections with an implicit surface defined by some measure being equal to some value, thus some surface surrounding the locus, hopefully closely and accurately).

Ray tracing is a well-known graphical rendering technique, and an interested reader can find good introductions in books such as Glassner [91], Foley and van Dam [84], or Watt and Watt [287]. The basic concept is to trace light backward from the eye until it intersects with some object,

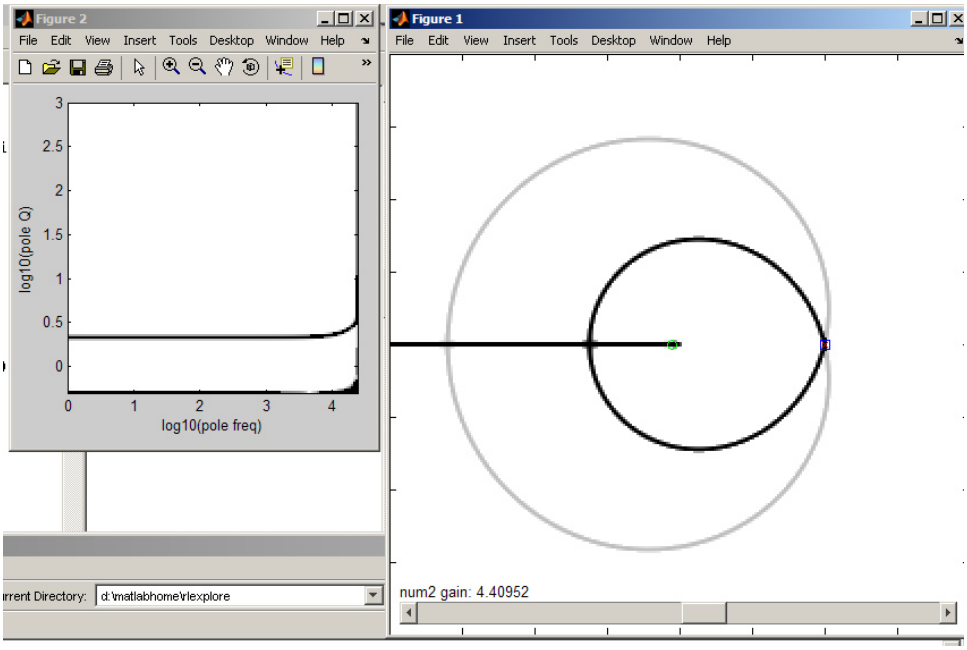


Figure A.22: Screenshot of second-order root-locus explorer with additional live (freq,Q) display, rendered using Taubin's method.

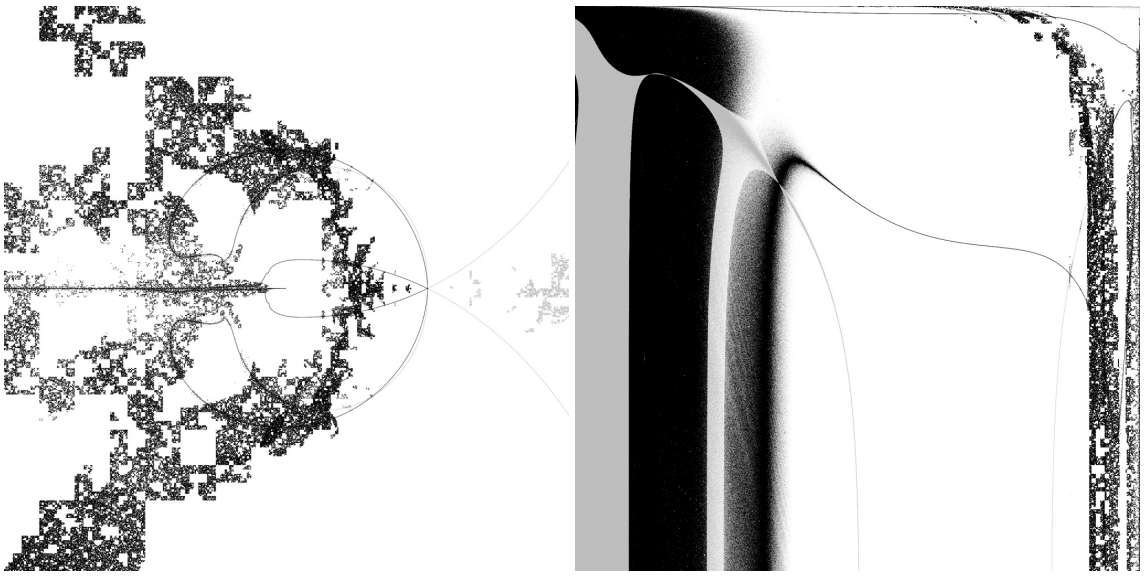


Figure A.23: Fourth-order locus with numerical problems in rendering. Left: z plane, Right: (freq,Q) plane. Note how issues near $z = 1$ are magnified, as $z \to 1$ occupies the whole left edge of the (freq,Q) plane.

and then figure out the color of the object at that point, possibly recursively spawning other rays to determine lighting, shadowing, reflections, transmissions, etc. For objects that have implicit algebraic description, the intersection test is done simply by substituting the ray equation into the implicit equation and solving for the distance along the ray. Any real positive solutions represent ray/object intersections. For other types of objects, there are methods which step along the ray, testing for intersection using other methods such as distance approximations or inside/outside tests.

Since the locus is an implicit function in a three-dimensional space (For example, the 2nd-order locus $D+kN_1+k^2 = 0$), then there is also an isosurface in the space based on some distance estimate to the locus: $\delta_{est} = \varepsilon$. If we could get a good distance approximation, we might be able to render this isosurface as a set of "tubes" which follow the locus and thus make a rendering of the locus if viewed parallel to the k axis.

Now within the field of computer graphics, there are also other methods for rendering implicit surfaces (see Bloomenthal [20] for an introduction). It should be noted, that the contour-plotting method described later has a 3D equivalent in an implicit-surface rendering, known as "Marching Cubes" [158]. Research for this thesis did not look much further beyond ray casting methods, except for an initial experiment or two. Most interesting was the idea of applying methods for ray-tracing implicit surfaces.

Ray Stepping Theoretically, we could directly ray-trace the implicit function, "simply" solving $\delta_{est}(p_0 + tp_{dir}) = \varepsilon$ for t (where $\delta_{est}(p)$ is the distance approximation at some point p, p_0 is the start of the ray, p_{dir} is the direction of the ray, and t is the distance along the ray). Any real solutions define the ray intersections. However, in general this function is as much as 2x to 8x the order of the polynomials involved in the root locus (i.e., the orders of $D(s)$ and $N_i(s)$). As such, this was not considered a direction to follow.

A well-known method of tracing generic implicit surfaces [20], which does not need to directly solve the above function is: Foreach ray: step along the ray using an approximation/bound of the distance to the surface to modulate the step length, declaring an intersection whenever the distance falls below some threshold.

Since we already have a good distance approximation in Taubin's 2nd-order approximation (which extends to three dimensions in the obvious manner), we can trace a locus rather straightforwardly (Figure A.24). As with all the previous algorithms, the distance approximation (and gradient) were calculated using numerical approximations to the partial derivatives.

This method has a large drawback: it is still terribly slow. There are a huge number of function evaluations due to stepping along the ray. However, it was quickly realized that if we wanted to restrict the rays to be parallel to the k axis (which we have been doing anyways), we could make a much simpler problem.

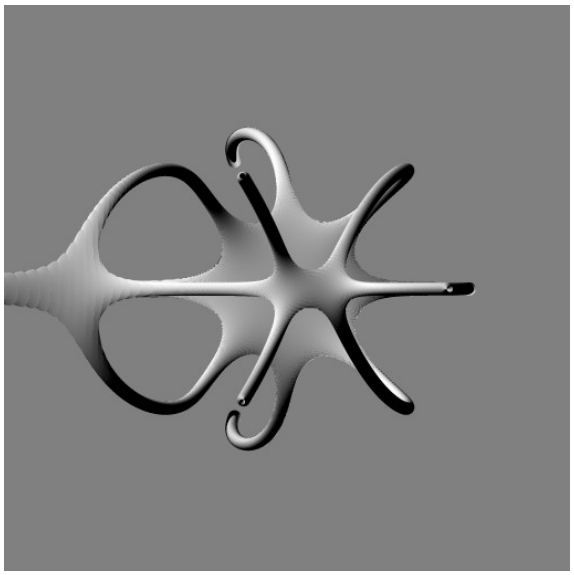


Figure A.24: Ray tracing implicit function $\delta_2 = \varepsilon$ for 2nd-order locus $(s^4 + 2) + k(s^3 - s^2/2 + 1) + k^2(s^3 - s^2/2 - 1) = 0$, using Taubin's 2nd-order distance approximation and a k range $-10 < k < 10$

Ray Tracing Parallel to k axis Whereas it would be numerical tricky and rather expensive to try to calculate intersections of general rays with such surfaces (as the order of polynomial which would need to be solves would be based on the orders of N_1, N_2 and D as well as on the order of the distance measure), it turns out that if we keep the rays parallel to the k axis (i.e., implement an orthographic projection along the k axis), then the polynomials to be solved become much simpler, as the contributions of N_1, N_2 and D reduce simply to being evaluated at the current point s, and the solution is only in a polynomial in k (usually only 2x or 4x the order of the locus, and independent of the orders of $N_1(s)$, etc.).

Deriving the $|D + kN_1 + k^2N_2|_\infty$ case: In this situation, we desire to ray-trace the surface

$$\|f\|_\infty = \|D(s) + kN_1(s) + k^2N_2(s)\|_\infty = \varepsilon \tag{A.36}$$

Now, $\|f\|_\infty = \varepsilon$ turns into a logical combination of a set of separate inequalities:

$$\begin{aligned} \mathrm{Re}(f) &= \varepsilon \text{ AND } |\mathrm{Im}(f)| \leq \varepsilon \\ &\text{OR} \\ \mathrm{Re}(f) &= -\varepsilon \text{ AND } |\mathrm{Im}(f)| \leq \varepsilon \\ &\text{OR} \end{aligned}$$

$$\begin{aligned} \mathrm{Im}(f) &= \varepsilon \text{ AND } |\mathrm{Re}(f)| \leq \varepsilon \\ \text{OR} & \\ \mathrm{Im}(f) &= -\varepsilon \text{ AND } |\mathrm{Re}(f)| \leq \varepsilon \end{aligned}$$

The inequalities can be further written as:

$$\begin{aligned} |\mathrm{Re}(f)| \leq \varepsilon &= \mathrm{Re}(f) \leq \varepsilon \text{ AND } -\mathrm{Re}(f) \leq \varepsilon \\ |\mathrm{Im}(f)| \leq \varepsilon &= \mathrm{Im}(f) \leq \varepsilon \text{ AND } -\mathrm{Im}(f) \leq \varepsilon \end{aligned}$$

and each of these as:

$$\begin{aligned} \mathrm{Re}(f) \leq \varepsilon &\Rightarrow (\mathrm{Re}(D) - \varepsilon) + k\mathrm{Re}(N_1) + k^2\mathrm{Re}(N_2) \leq 0 \\ -\mathrm{Re}(f) \leq \varepsilon &\Rightarrow (-\mathrm{Re}(D) - \varepsilon) + k\mathrm{Re}(N_1) + k^2\mathrm{Re}(N_2) \leq 0 \\ \mathrm{Im}(f) \leq \varepsilon &\Rightarrow (\mathrm{Im}(D) - \varepsilon) + k\mathrm{Im}(N_1) + k^2\mathrm{Im}(N_2) \leq 0 \\ -\mathrm{Im}(f) \leq \varepsilon &\Rightarrow (-\mathrm{Im}(D) - \varepsilon) + k\mathrm{Im}(N_1) + k^2\mathrm{Im}(N_2) \leq 0 \end{aligned}$$

These can be combined through the previously-mentioned logic to create a set of quadratic equations to find intersections along k. Note that this case doesn't involve solving any equations of order higher than the order of the locus in k. Renderings of loci using this technique are shown in Figures A.25 and A.26. While not too useful as loci, they are interesting as abstract 3D shapes.

The above equations simplify in a straightforward manner for a first-order locus.

As a side note: with a bit of algebra, one can derive that the projections of the hard edges of these 3D shapes (the corners of the L_∞ norm) are actually complex-coefficient root-loci in their own right. For first-order loci they are:

$$D(s) + kN(s) = \varepsilon(\pm 1 \pm i) \Rightarrow (D(s) \pm (1 \pm i)) + kN(s) = 0 \tag{A.37}$$

And similarly for second-order loci, the constant term of $D(s)$ is simply offset by the corners of the L_∞ unit ball. This is demonstrated in Figure A.25.

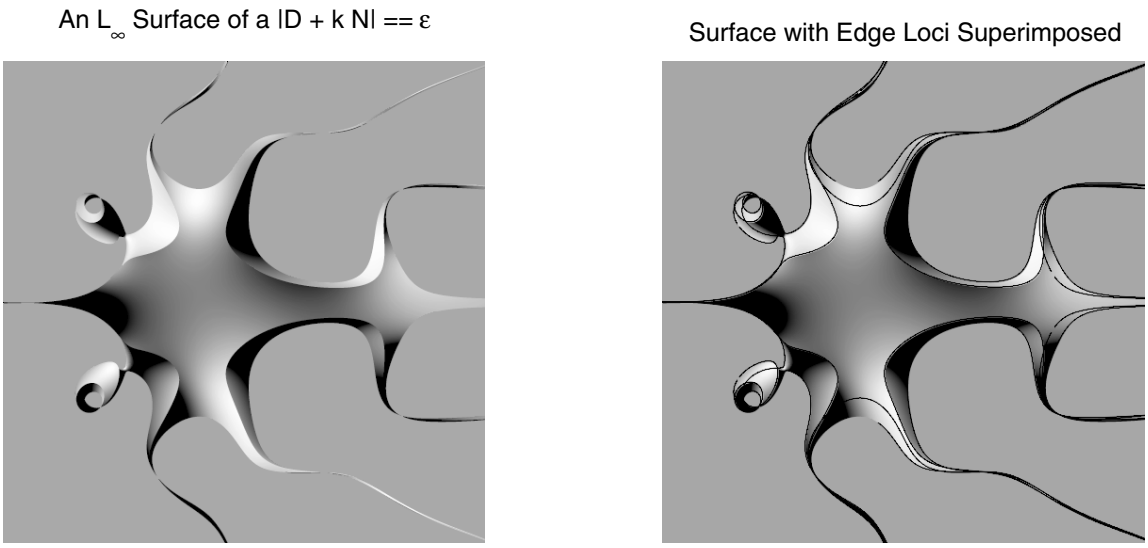


Figure A.25: The "hard edges" of the L_∞-norm surface project down to certain complex-coefficient root loci.

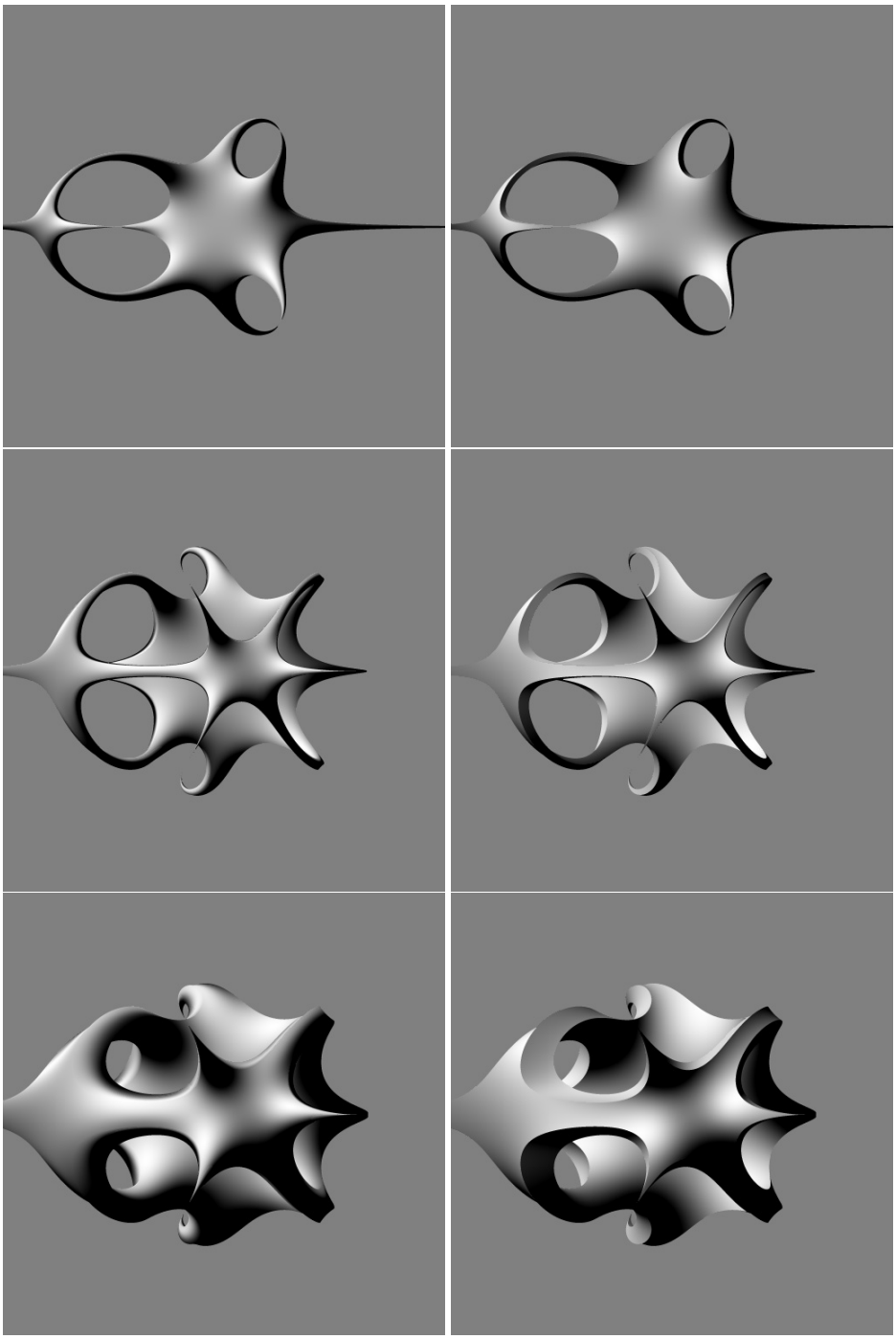


Figure A.26: Ray Tracing zeroth-order distance approximation ($|D + kN_1 + k^2N_2|_\infty = \varepsilon$) using rays parallel to k axis: Top: First-Order in k (Left: Surface normal estimate based on L_2 norm, Right: Surface normal based on L_∞ norm). Middle: Second-order in k. Bottom: larger value of ε. (same loci as previous examples).

Deriving the first-order Taubin's approximation case: In this case, we want to find intersections with the surface

$$\delta_1 = \frac{|f|}{\|\nabla f\|} = \varepsilon \tag{A.38}$$

where $f = D(s) + kN_1(s) + k^2N_2(s)$. This is most easily accomplished by multiplying both sides by their complex conjugate and noting the following:

$$\begin{aligned}
\frac{\partial f}{\partial s} &= D'(s) + kN_1'(s) + k^2N_2'(s) \\
\frac{\partial f}{\partial k} &= N_1(s) + 2kN_2(s) \\
\|\nabla f\|_2^2 &= \left|\frac{\partial f}{\partial s}\right|^2 + \left|\frac{\partial f}{\partial k}\right|^2
\end{aligned}$$

Thus $\delta_1^2 = \varepsilon^2$ becomes:

$$|D + kN_1 + k^2N_2|^2 - \varepsilon^2\left[|D' + kN_1' + k^2N_2'|^2 + |N_1 + 2kN_2|^2\right] = 0 \tag{A.39}$$

For rays parallel to the k axis, we therefore solve this for k and find all real solutions. With some algebra, this turns into solving for k in the polynomial

$$A + Bk + Ck^2 + Dk^3 + Ek^4 = 0 \tag{A.40}$$

where

$$\begin{aligned}
A &= |D|^2 - \varepsilon^2(|D'|^2 + |N_1|^2) \\
B &= 2\mathrm{Re}(DN_1^*) - 2\varepsilon^2\mathrm{Re}(D'N_1'^* + 2N_1N_2^*) \\
C &= |N_1|^2 + 2\mathrm{Re}(DN_2^*) - \varepsilon^2\left(|N_1'|^2 + 4|N_2|^2 + 2\mathrm{Re}(D'N_2'^*)\right) \\
D &= 2\mathrm{Re}(N_1N_2^*) - 2\varepsilon^2\mathrm{Re}(N_1'N_2'^*) \\
E &= |N_2|^2 - \varepsilon^2|N_2'|^2
\end{aligned} \tag{A.41}$$

evaluated at the current point in s (remember that the star denotes conjugation).

For a first-order locus, this turns into solving for k in the polynomial

$$A + Bk + Ck^2 = 0 \tag{A.42}$$

where

$$\begin{aligned}
A &= |D|^2 - \varepsilon^2(|D'|^2 + |N_1|^2) \\
B &= 2\mathrm{Re}(DN_1^*) - \varepsilon^2\mathrm{Re}(D'N_1'^*)
\end{aligned}$$

$$C = |N_1|^2 - \varepsilon^2 |N_1'|^2$$

evaluated at the current point in s.

Renderings of loci using this technique are shown in Figure A.27. These are a bit more useful as loci compared to the previous case, but probably still overkill as locus renderings.

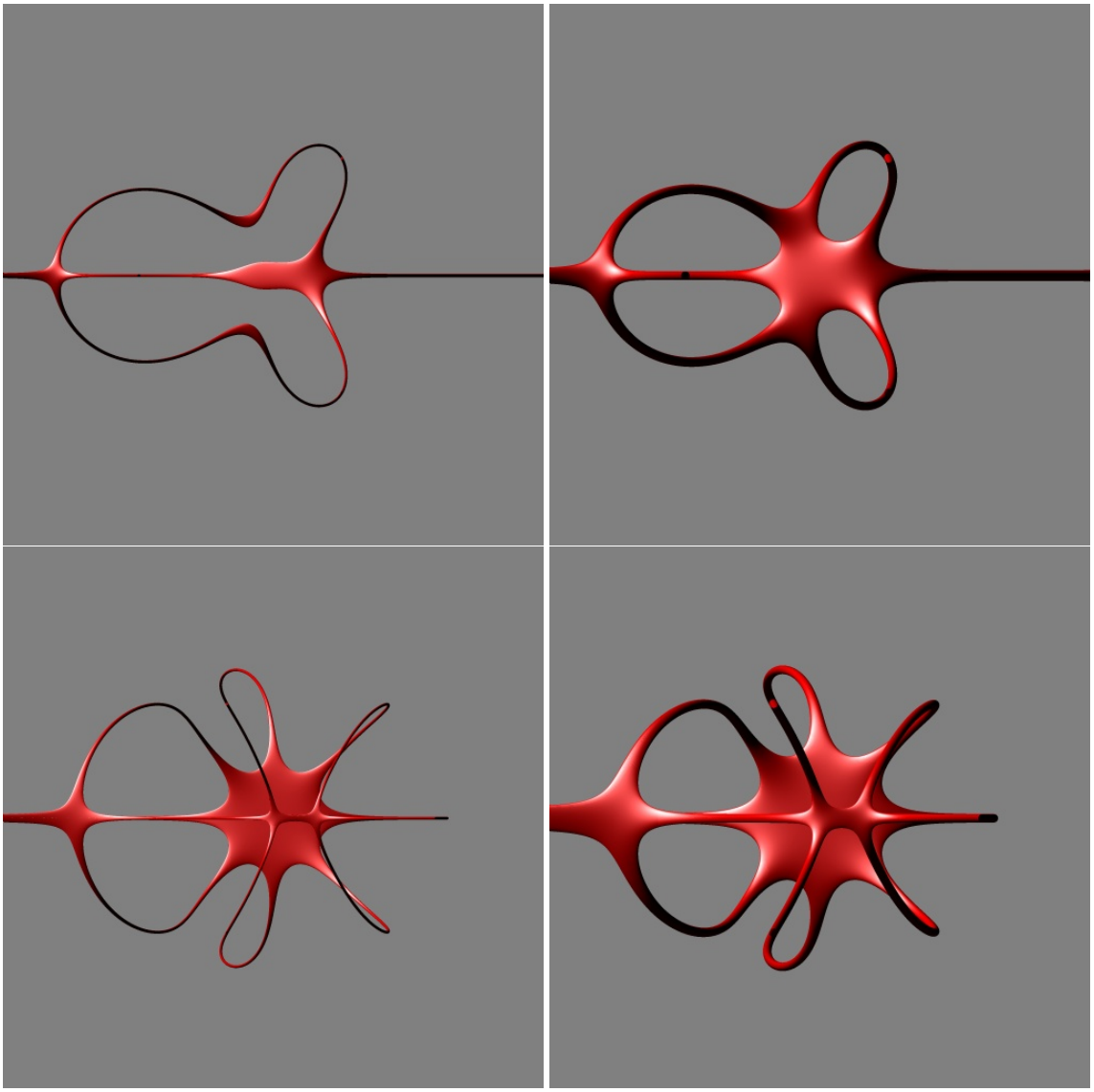


Figure A.27: Ray Tracing Taubin's distance approximation using rays parallel to k axis: Top: First-Order in k, different ε. Bottom: Second-order in k, different ε. (same loci as previous examples).

A.3.5 Hybrids

Grid and Contour

This is a very straightforward algorithm (on the surface): evaluate $y = \mathrm{Im}(D/N)$ or $y = \mathrm{Im}(GH)$ on a grid in the plane, and use standard contour-plot methods to plot the contour $y = 0$. As this only involves 'forward evaluation' (i.e., no root-finding), it can be used to render loci of non-polynomial systems, since a root finder for the system is not required. the author used it to render some of the string loci in Appendix B, for example.

This method is an implicit-function renderer, except that instead of giving an pixel image as output (as in the renders of the previous section), it gives vector output. Of course, there are still sampling issues associated with the calculation of the grid, which is why this algorithm is called a 'hybrid' algorithm, as it has both vector and pixel properties.

Being an implicit-function renderer, this method can effectively apply to the same classes of loci as, say, Taubin's method: loci whereby we can find an implicit function for the locus (i.e., which no longer has k in it). Thus it trivially applies to first-order loci, with no restrictions on D and N being polynomial or not. This method can also use the same extensions as described for Taubin's method to render 2nd-order and higher-order loci in k.

There are two complications to the simplicity implied above: (1) as with the pixel-based methods, the method inherently finds both the 0° and 180° loci, as both satisfy $\mathrm{Im}(D/N) = 0$; and (2) the In the case of the pixel-based renderers, one could simply check the sign of $\mathrm{Re}(N/D)$ and choose not to render if the sign was not the required one. This option is available in the case that one has the ability to modify the contour-calculation algorithm. However, if one is using a 'black-box' contour plotter, things get tricky. the author is not familiar with a general solution to this problem, but Matlab's contour plotter has a feature which allows us to solve this problem: like other Matlab plotting functions, array elements containing NaNs (special "Not A Number" values in the IEEE floating-point number system, normally used to flag the results of undefined operations, like ∞/∞, $\infty - \infty$, $0/0$, etc.) are not plotted at all (as opposed to plotting some fallback value). Therefore, one can plot just one side of the locus by taking the array containing $\mathrm{Im}(D/N)$, and setting all elements for which $\mathrm{Re}(D/N) < 0$ to NaN (or > 0, depending on which locus you want to plot). Doing a contour plot on this array will cause the contour calculator to only calculate contours of the desired locus.

Here is an example of rendering the full locus (both positive and negative k) of $(s^4 - s/10) + k(s^3 + 1) = 0$ in the range $-2 < \mathrm{Re}(s) < 2$, $-2 < \mathrm{Im}(s) < 2$ in Matlab:

```
xax=linspace(-2,2,200);
yax=linspace(-2,2,200);
[xx,yy]=meshgrid(xax,yax);
zz = xx+i*yy;
```

```
y = polyval([1 0 0 -.1 0],zz)./polyval([1 0 0 1],zz);
contour(xax,yax,imag(y),[0 0]);
```

And to plot just the 180° locus:

```
y1 = imag(y);
y1(find(real(y)>0))= NaN*ones(size(find(real(y)>0)));
contour(xax,yax,y1,[0 0])
```

The resulting plots are shown in Figure A.28

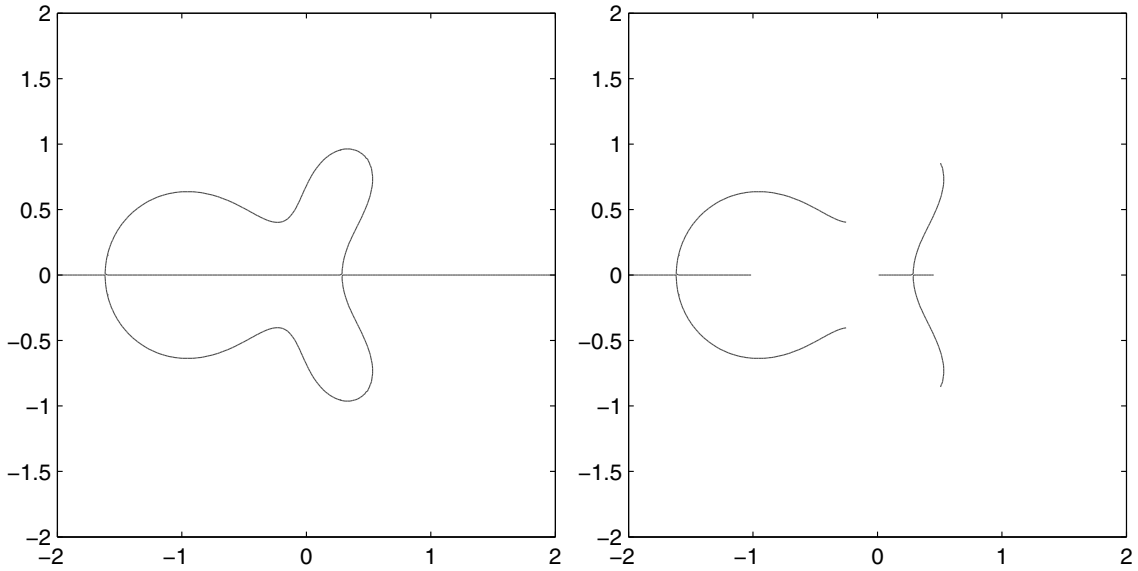


Figure A.28: Root Loci rendered using Matlab's `contour` function. Left: full locus. Right: 180° locus.

This method does not work with all contour plotters. For example, the one in Mathematica requires that all contour traces exist on parts of a surface with defined values, and since the loci for either angle "butt up" against each other, all the contours will run up against NaN values in the course of tracing the contours. As such, the algorithm removes them from the list of valid contours, and one ends up with no contours at all.

Path Following via Pivoting

Within the field of homotopy continuation this method is commonly discussed aside from predictor-corrector. It is based on the concept of being able to easily figure out which vertex of a triangle the path crosses as it exits out of the triangle. By tiling the plane with sufficiently small triangles, the

path can be followed by keeping track of which triangles are visited (the movement from one triangle to the next is called "pivoting" in this technique). As such, space is sampled in a way, which puts this in Hybrid class of techniques.

We do not give examples of this technique in this thesis, but interested readers are referred to the numerical continuation literature listed in the introduction to predictor-corrector algorithms, and to papers such as [191].

Appendix B

Root Locus Interpretation of Piano-String Linear Mode Coupling

This research was originally presented at the 1996 Acoustical Society of America Conference, and later as a poster at the 1997 International Computer Music Conference [247].

B.1 Introduction

Previous work in the study of coupled piano string behaviors, in particular that of Weinreich, Nakamura, and Hundley ([288] [187] [109]) has either focused analytically on the interaction of a pair of coupled modes, noting that the rest of the string modes couple similarly, or has studied the coupling experimentally. The equations describing three-string coupling (along the lines of [187]) become extremely complex and one can easily become lost in interpreting them. In this paper, we will show how one can interpret the coupling behavior in terms of the *Root Locus* analysis method, which analyzes the location of the poles of a closed-loop linear system according to the pole and zeros of the open-loop system and under variation of the feedback gain. Root Loci were for decades drawn by hand, so that a lore was developed about patterns that appear in root loci; this lore represents an intuition that can be acquired and applied to get a feel for the behaviors of closed-loop systems — coupled strings, in this case.

In this appendix, the coupling of two and three modes, as occurs in sets of unison piano strings, is studied, along with the coupling behavior of multiple groups of modes, such as the multiple harmonics of the strings. Many effects, for example, beating and two-stage decay, can be understood in terms of the system pole locations (since for impulsively driven systems such as pianos, the normal behavior of the system is dominated by the impulse response of the system), which makes the root-loci directly interpretable. Furthermore, rather complex, frequency-dependent coupling

impedances can easily be included in the analysis, allowing an understanding of the variation in coupling behavior at different string harmonics.

Note that most of the behaviors that we note in this appendix have already been discovered and explained in the acoustics literature (in Weinreich, etc., and by others). What is new to this research is the application of Root-Locus ideas to the understanding of the physical phenomena, and in helping build up intuition about the physics.

B.2 Piano Strings

The coupling that is of interest comes about due to the fact that modern pianos have sets of strings in three groups:

- The Bass Strings: Heavily wound single strings.
- The Midrange Strings: Lighter wound strings in pairs.
- The Treble Strings: Even lighter wound strings in triples.

For the notes which have multiple strings, the string are tuned to near unison. This implies that the dynamical system representing the vibrating strings for a particular note will have multiple poles in close proximity at each of the harmonics of the note. Because the strings are not tuned exactly equal, their corresponding harmonic frequencies will presumably get further apart (in raw Hz) at higher harmonics. However, the strings do not ring completely independently, instead, they affect each other (they are "coupled"). As such, the behavior of the total system is different than that of the independent strings. In particular, there is an effect called "two-stage decay", which is in fact very important to the sound of the piano, and the piano tuner specifically tunes the strings to optimize this effect. In two-stage decay, the string does not sound like two detuned strings beating against each other, but instead sounds like a single string with a two-part decay.[1]

It is believed (through anecdotal evidence) that the discovery of this behavior was integral to the development of the piano. It allowed note to sound loudly (i.e., lose energy to the air quickly), yet still have a long late decay. A single-string in the midrange which sounds loudly would decay too quickly, whereas one a single string with a long decay would not sound sufficiently loudly. Further, the slow decay achieved in the two-stage decay can be even slower than the decay of the individual strings, allowing the piano to have a longer ring-out than previous stringed keyboard instruments. The two-stage decay gives a loud initial ring during the fast stage, and then settles into a pleasingly long decay. Piano tuners tweak the detuning of the strings so as to balance the amount of energy lost in the initial decay stage versus the loudness of the second decay stage.

The piano-string dynamical system is a bit more complex than described, as one might expect. As such, the coupling behavior may not be the same across all the string's harmonics. We will see at

[1] In three-string notes, this all gets even more complicated, of course.

the end of this discussion an analysis of how the coupling may be different between low harmonics and high harmonics due to detuning effects and loss effects.

B.3 Root Locus Analysis

This technique, which originated in the analysis of linear feedback control systems, analyzed the poles of a closed-loop feedback system in terms of the open-loop transfer function and the (variable) loop gain. In general, root locus interprets a linear system with feedback as though it were in the form in Figure B.1, a feedback loop, split up with a feedforward part (G), and a feedback part (H), with a variable loop gain k. Traditionally, the feedback is summed with a negative sign. Positive feedback (poles) occurs at frequencies (or s or z locations) where the phase angle of the feedback loop is some multiple of 2π, thus, taking the negative feedback sign into account, traditional root locus looks for phase angles in the rest of the feedback loop of $\pi \pm n(2\pi)$, and is called the "180° root locus".

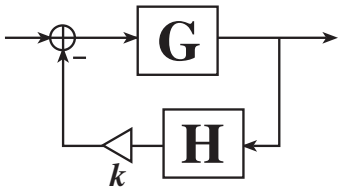


Figure B.1: Linear feedback system drawn in Root-Locus Form

As a review (This partially duplicates discussion in Appendix A), the transfer function from input to output of this system is

$$\frac{G}{1+kGH}$$

Now, if we assume the G and H have rational transfer functions (in s for now, though root locus analysis works exactly the same way in the z plane), then we can define two polynomials $D(s)$ and $N(s)$ as such:

$$\frac{N(s)}{D(s)} = G(s)H(s)$$

In particular, $N(s)$ is the numerator of GH, and $D(s)$ is the denominator. This will allow us to work on the root locus with simple polynomials rather than having to deal with rational functions all the time. Thus the denominator of the transfer function is $1 + kN(s)/D(s)$, or is we multiply the transfer function through by $D(s)$, we get

$$\frac{G(s)}{1+kG(s)H(s)} = \frac{G(s)H_{den}(s)}{D(s)+kN(s)}$$

Now in Root locus analysis, we are only worried about the location of the roots, so we only look at the denominator, and get our primary equation:

$$D(s) + kN(s) = 0 \tag{B.1}$$

The roots of this equation are the poles of the system. Solving for k we get:

$$k = -\frac{D(s)}{N(s)} \tag{B.2}$$

Now we define that k is real, which thus constrains the fraction:

$$\text{Im}(-D(s)/N(s)) = 0, \text{ or } \angle(D(s)/N(s)) = \pi \Rightarrow \angle D(s) - \angle N(s) = \pi \tag{B.3}$$

And gives us an implicit equation in s for the locus. Any point s_0 which satisfies this is a point on the locus, and the k value for that point is given by Equation B.2, which we know will be real. Traditionally, root locus analysis only looks at $k \geq 0$, though there are many situations where looking at $k < 0$ (the "zero-degree locus") is also of use. As such, the above implicit equation in s is usually augmented as:

$$\angle(D(s)/N(s)) = \pi, \ \text{Re}(-D(s)/N(s)) > 0 \tag{B.4}$$

Textbooks on classical control systems, such as [88], provide rules on drawing root-loci, and Appendix A gives a short introduction to the topic. The most important rule to note at the moment is that the closed-loop poles coincide with the open-loop poles (roots of $D(s)$) when $k = 0$, and move to coincide with the open-loop zeros (roots of $N(s)$ as $k \to \infty$. Thus we gather quite a bit of information simply by plotting the open loop poles and zeros. Another rule is that the paths of the poles are given by the equation $\angle(GH) = \pi$. We can use this rule to draw root-loci of non-rational linear systems, which show up in the analysis of continuous-time string coupling (for example, the "Grid-and-Contour" method See Section A.3.5 in Appendix A, which was used to render many of the string-coupling loci later on in this discussion).

B.4 Coupled Modes

We will be analyzing the piano string dynamics by starting from existing models, rather than deriving all the way from the physics, as the models we will use have been well-derived and allow us to work with simpler equations. Our model is the waveguide model of N strings coupled at a junction [237], [239] (Figure B.15). As such, this analysis assumes rather ideal strings, though features such as dispersion, frequency-dependent losses, and harmonic stretching, can be included

in the loop and load filters if desired.

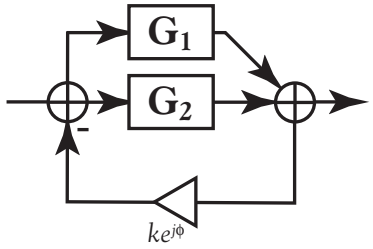


Figure B.2: Simple Coupling of Two Systems

But first, let us step back and look at a much simpler system: two complex poles coupled through a coupling phase (Figure B.2): Let $G_1(s) = \frac{1}{s-p_1}$ and $G_2(s) = \frac{1}{s-p_2}$. We can interpret k as the magnitude of the coupling between the two open-loop modes p_1 and p_2. For simplicity, we will not make this a real-coefficient system (as such, the locus will not necessarily be symmetric about the real axis).

Thus, our system transfer function is:

$$\frac{\left(\frac{1}{s-p_1}+\frac{1}{s-p_2}\right)}{1+ke^{j\phi}\left(\frac{1}{s-p_1}+\frac{1}{s-p_2}\right)} = \frac{2s-(p_1+p_2)}{(s-p_1)(s-p_2)+ke^{j\phi}(2s-(p_1+p_2)}) \tag{B.5}$$

And our root-locus equation is:

$$(s-p_1)(s-p_2)+ke^{j\phi}(2s-(p_1+p_2)=0 \tag{B.6}$$

Thus the locus starts at $s = p_1$ and $s = p_2$, leaving at angles influenced by ϕ, and ends at $s \to \infty$ and $s = (p_1+p_2)/2$. Representative loci are shown in Figure B.3. The locus shapes for ϕ near zero are what we will be most interested in in this discussion, as they directly describe the basics of mode coupling.

Let's look closer at such a locus (Figure B.4), and representative impulse responses at selected gains along the locus (Figure B.5). In this case, $p_1 = 50j$ (rad/sec), $p_2 = 60j$ (rad/sec), and $\phi = \pi/1000$.

Note that with no coupling, the poles would be on the imaginary axis and not decay (this is an ideal example, of course), and that there would be some beating between the two pole frequencies. As k starts increasing, the poles pull away from the imaginary axis and thus start to have some decay, whose rate is initially proportional to k (see the '+' case). As k continues to increase, the poles curve towards each other, making the beat frequency get slower (this can be seen comparing $k = 2$ and $k = 7$ in Figure B.5). Note that both poles have the same decay rate in this range of k.

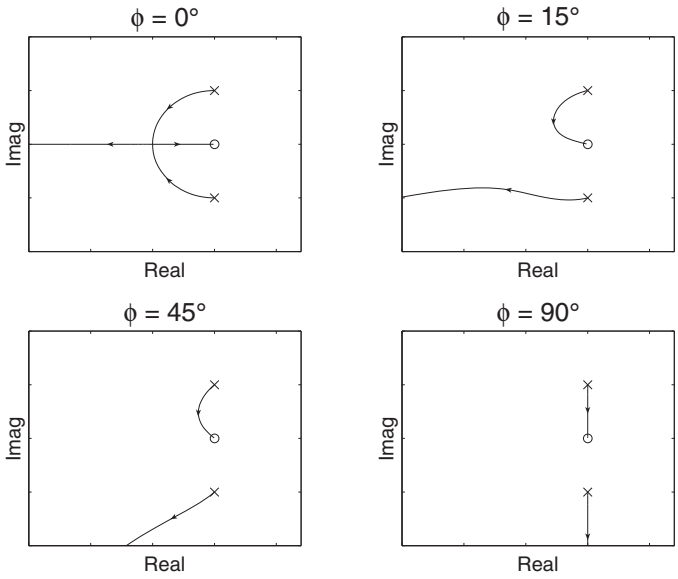


Figure B.3: Sample 2-Coupled-Mode Loci

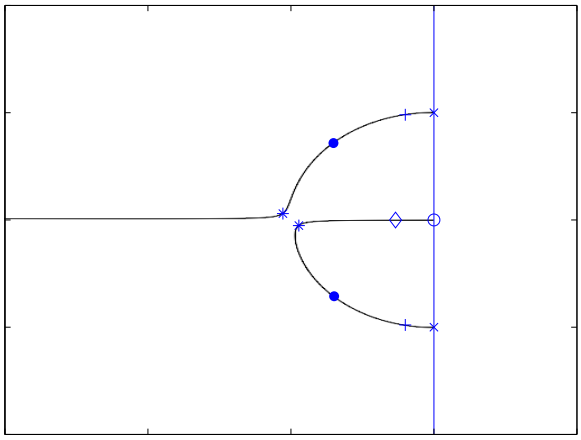


Figure B.4: Pole locations at various gains along a locus, for the system of Equation B.5. plus: k = 2, dot: k=7, star: k=10, diamond: k=20 ($p_1 = 50j,\ p_2 = 60j,\ \phi = \pi/1000$)

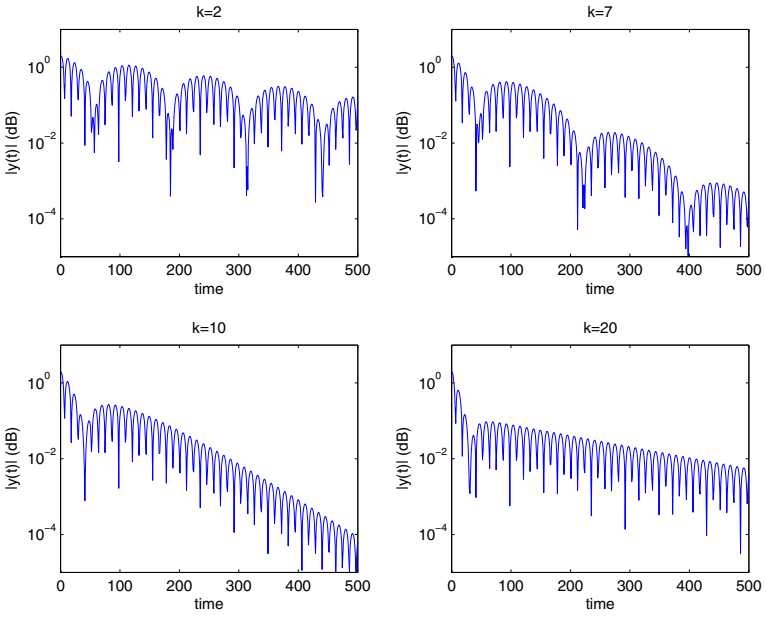


Figure B.5: Impulse responses for the pole sets in Figure B.4.

At some value of k, the pole get as close as they will get (the starred pole locations in Figure B.4). They can become co-located if ϕ is exactly zero in this example. At this point, the total impulse response is at its fastest decay rate. From here on, one pole starts moving back towards the open-loop zero (halfway between the open-loop poles) and a slower decay, and the other heads away towards ∞ and a faster decay. As such, the poles no longer have the same decay rate, and the resulting impulse response has the "two-stage decay" that is associated with mode coupling. Both move asymptotically onto a line passing through the open-loop pole location, at an angle based on ϕ. When ϕ is small, this causes both to be essentially on the same frequency, and the poles appear to have “locked” frequency. Further increase in k simply increases the decay rate of the fast pole and decreases the decay rate of the slow pole. A rule of thumb that has been developed to describe this type of 2-mode coupling is that before locking, there is beating and a single decay rate, and that after locking there is a single frequency and a two-stage decay.

Later, we will revisit this coupling and see how it acts as a function of the detuning amount of the open-loop poles, and compare with Weinreich’s well-known coupling and locking diagrams.

Now, looking back at loci for different coupling angles (Figure B.3), we see that the phase angle at the coupling frequency has the effect of “rotating” the locus. An alternative interpretation, based on looking back at Equation B.6 and noting that the phase in this case can be interpreted as modifying k, and thus essentially giving a root locus in some angle other than 180°. An effect of this rotation at small rotations is to detune the two poles at strong coupling, so that the 2^{nd}stage of

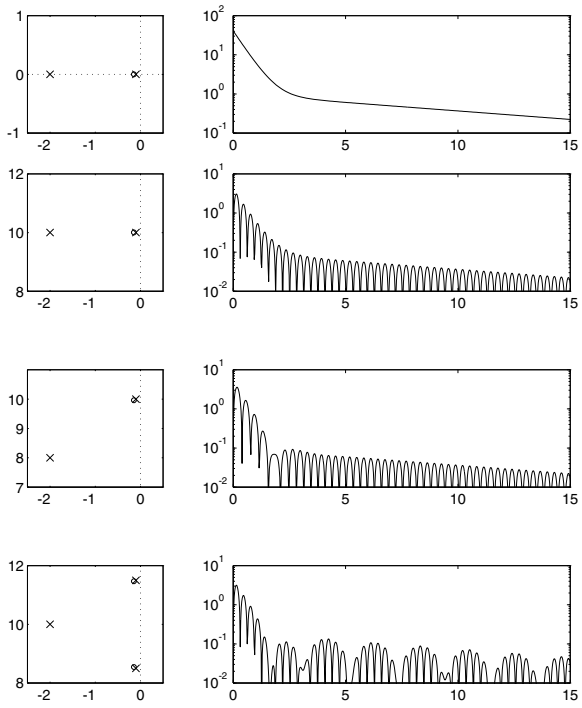


Figure B.6: Rectified impulse responses for various two-stage closed-loop pole configurations. Top: poles on real axis, two stage exponential decay; Upper Middle: poles off real axis (with a complementary pair assumed), two stage decay at the pole frequencies; Lower Middle: one pole offset in frequency, some cancellation as the decay amplitudes cross; Bottom: Three poles, one fast, two slow, note beating in 2$^{\text{nd}}$stage.

decay rings out at a slightly different frequency than the first stage. At strong rotation, the poles become so far removed that there is little noticeable difference between weak and strong coupling frequency-wise. The poles still start at slow (no) decay, and increase their decay rate with coupling, then at some point one pole heads back to slow decay and the other heads to faster and faster decay. However, as the rotation approaches 90° (reactive impedance), the poles are no longer pulled towards faster decay, and the coupling simply moves the pole frequencies around. The rotation of the locus also explains the "mode repulsion" effect which Weinreich noticed when moving the open loop poles with reactive impedances [288] (lower-right plot: the 90-degree coupling case), as the pole never come together like they do with the lesser rotations. More on this later in Section B.5.

It is important to note that the *shapes* of the loci are relatively independent of the actual mistuning of the modes (Figure B.7). These loci will look the same for any vertical spacing of the two modes. However, the distance between the open-loop poles *does* affect the range of k over which different behaviors will occur.

The closed loop poles land at the intersections of the $\angle(GH) = \pi$ and $|GH| = 1/k$ contours

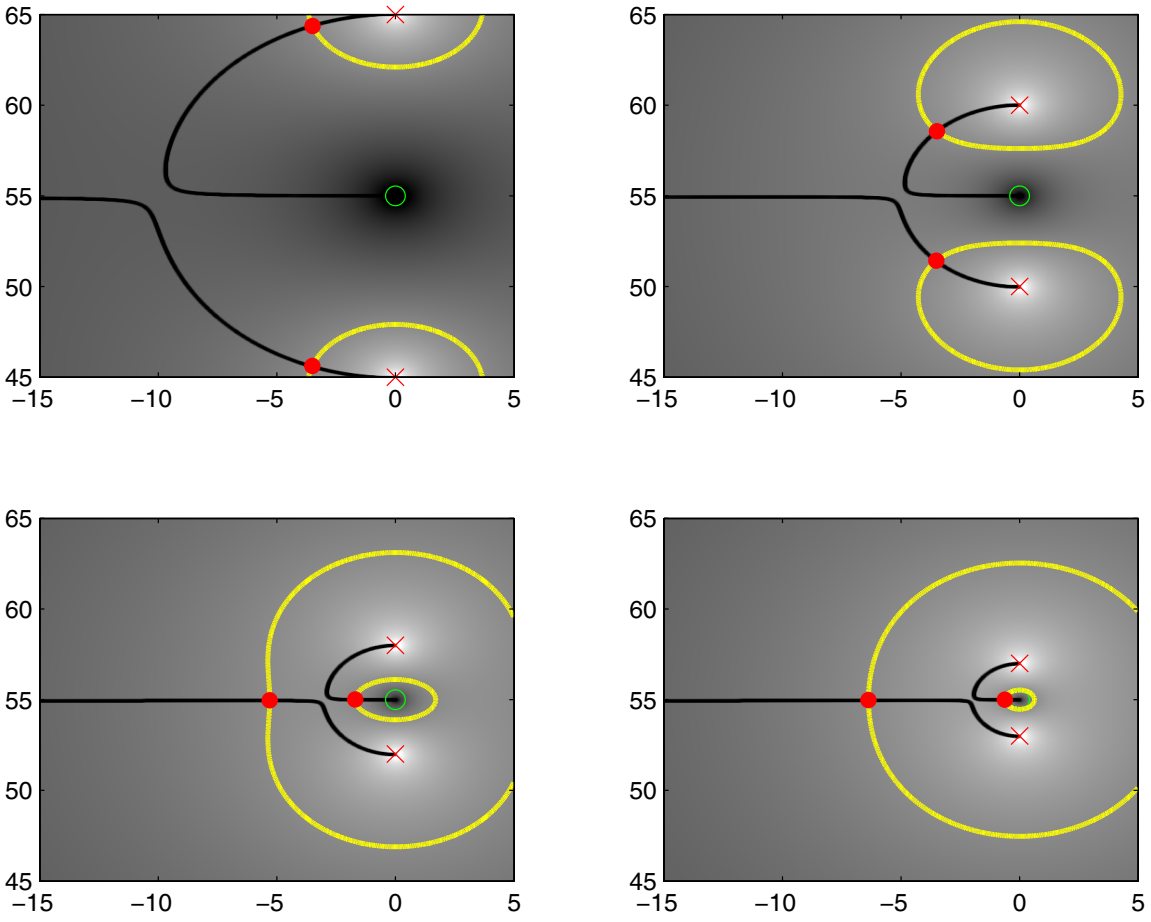


Figure B.7: The shape of the coupling locus is essentially independent of the mistuning, but the coupling strength is not. All the above cases have the same coupling gain $k = 7$. The coupled poles lie at the intersection of the root locus and the magnitude contour $|GH| = 1/k$ (or $|D(s)/N(s)| = k$).

(see Figure B.7). When the poles are closer together, the magnitude of GH is larger in the root-locus "circle" region, and so the closed-loop poles for a particular k are further along the coupling behavior. This can also be seen quite well in Figure B.25, where the open-loop poles of higher string harmonics are further apart, and so the same coupling gain results in a "weaker" coupling. The main result of this is that one can also move the system between beating and two-stage decay by fixing k and adjusting the mistuning of the modes (which is how Weinreich did it in [288], and what we will explore in the next section).

B.5 Comparing with Weinreich's Diagrams

In [288] Weinreich plotted the closed-loop pole frequencies versus the mistuning (in this case, fixing one open-loop pole and moving the other past it). As we have seen above, the behaviors noted there can be interpreted in terms of the behaviors of root loci. In Figure B.8, we look at the loci for a couple of mistunings, and note how the coupled poles trace a locus in the mistuning variable. At the start, the moving pole is well below the fixed pole, and the coupled poles are quite detuned. As the moving pole gets closer to the fixed pole, the k-locus circle gets smaller (as noted in the previous section). Interestingly, although the coupling gets stronger relative to the shape of the locus, the couple-pole decay rates (their real parts) stay relatively constant. When the mistuning gets small enough, the coupled poles move into two-stage decay, or are "locked". Again, it is interesting to note that the poles stay symmetric about their "unlocked" decay rate. As has been noted several times elsewhere in this thesis, the circle is a common shape in root loci, and here again we see a circle show up in the shape traced by the locked poles (we will see later that as the coupling angle increases, this circle becomes more distorted). Finally, as the moving pole moves past the fixed pole, the process repeats in reverse (i.e., the behavior is symmetric about the fixed pole when there are no other influences on the system).

In this case, the locus in mistuning is a 2^{nd}-order root locus when the conjugate poles are put into the system to make it real:

Let the open-loop pole locations be $p_1 = -a + jb$ and $p_2 = -a + j(b + \delta)$, and the coupling phase be provided by a zero at $s = 0$ so that

$$\frac{N(s)}{D(s)} = \frac{s\left(2(s+a)^2 + b^2 + (b+\delta)^2\right)}{((s+a)^2 + b^2)((s+a)^2 + (b+\delta)^2)} \tag{B.7}$$

and the root locus equation in k is:

$$((s+a)^2 + b^2)((s+a)^2 + (b+\delta)^2) + ks\left(2(s+a)^2 + b^2 + (b+\delta)^2\right) = 0 \tag{B.8}$$

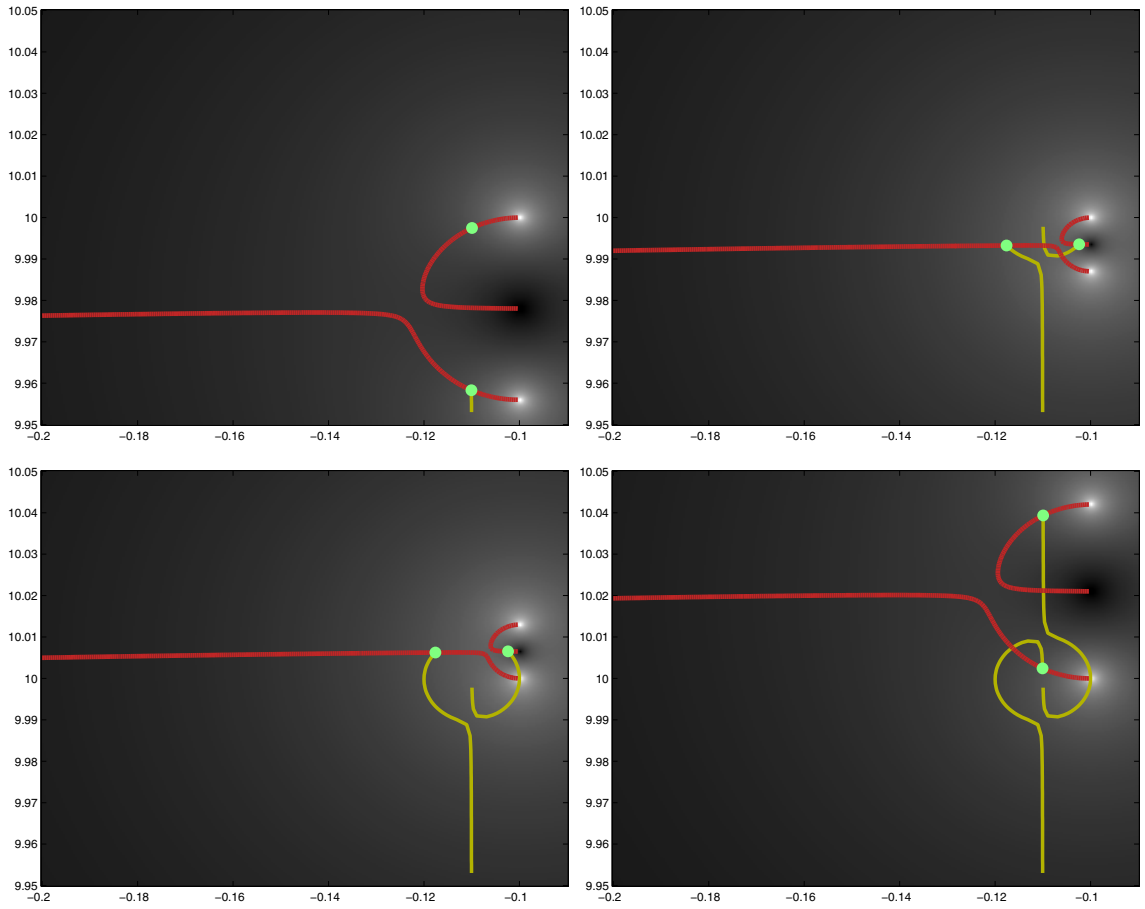


Figure B.8: *k*-Loci as mistuning changes, in the style of Weinreich's mistuning experiments. The roots for this particular value of k trace a 2^{nd}-order root locus in the mistuning amount.

But the root locus equation in δ is:

$$[((s+a)^2+b^2)((s+a)^2+b^2+2ks)] + \delta[2b((s+a)^2+b^2+ks)] + \delta^2[(s+a)^2+b^2+ks] = 0 \quad \text{(B.9)}$$

We can simplify this a bit by defining $C(s) = (s+a)^2+b^2 = (s-(-a-jb))(s-(-a+jb))$ (i.e., the polynomial for the fixed pole pair):

$$[C(s)(C(s)+2ks)] + \delta[2b(C(s)+ks)] + \delta^2[C(s)+ks] = 0 \quad \text{(B.10)}$$

A plot of this locus with representative a,b, and k is shown in Figure B.9. Compare it with the

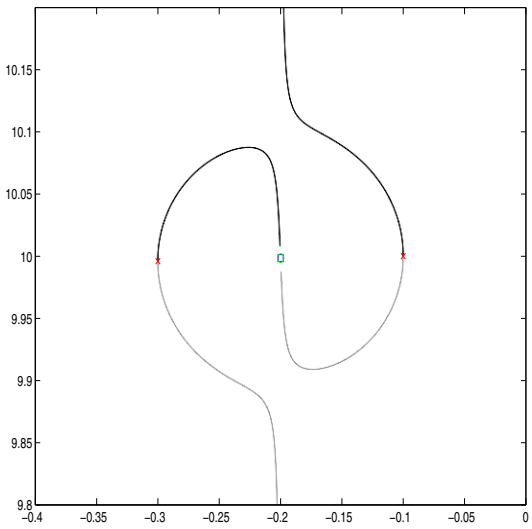


Figure B.9: A 2nd-order Root Locus in mistuning (Equation B.10). 'x': roots of $[C(s)(C(s)+2ks)]$, squares: roots of $[2b(C(s)+ks)]$, 'o': roots of $[C(s)+ks]$. Black: $\delta > 0$, Gray: $\delta < 0$

locus in Figure B.8. Note that the δ-coefficient polynomial and the δ^2-coefficient polynomial have the same roots. Also note that one of the root pairs of the "denominator" coefficient polynomial is the fixed pole pair. In this case, we can also note that the other pole pair in that polynomial will be at approximately $(-a+jb)-k$, by looking at the first two terms of the series approximation in k about $k=0$ of that root location:

$$-a-k-\sqrt{-b^2+2ak+k^2} \approx (-a+jb) - (1+\frac{ja}{b})k + O(k^2) \approx (-a+jb) - k \quad \text{(B.11)}$$

since we are assuming that our open-loop poles are very close the imaginary axis, so that a/b will be very small.

Figures B.10 through B.14 show Weinreich-style tuning-vs-mistuning diagrams for various open-loop configurations, number of poles and strengths of tuning, including loci in mistuning for comparison. In these diagrams, another axis of information is shown on the diagrams as the grayscale color of the frequency track. White represents very fast decay, and black represents very slow decay. As such, the two stage decay can be seen in the fact that one of the locked poles gets a slow decay ("goes black"), whereas the other one gets a fast decay ("goes white").

B.6 Coupled Strings

Referencing Figure B.15, we can set up the root-locus for a system of coupled strings by letting:

$$G_{\text{Forward}} = \sum_{i=1}^{N} \frac{-G_{loop_i}\, e^{-s\,T_i}}{1 - G_{loop_i}\, e^{-s\,T_i}} \tag{B.12}$$

Where the T_i are the round-trip delays of each string, and G_{loop_i} are the lumped round-trip loss of the strings. For now, let $G_{loop_i} = -a_i$. The effect of a filter for G_{loop} would be mainly to give a frequency-dependent damping (i.e., $a_i(\omega)$), and to add some frequency-dependent phase. The poles of each string are thus:

$$s = \frac{1}{T_i}(\ln a_i + jm\pi) \quad \text{for } m = 0, \pm 2, \pm 4, \ldots \tag{B.13}$$

We can draw the root locus vs. k of the coupled system by evaluating the contour $\angle\,(G_{\text{Forward}}(s)\,G_{load}(s)) = 0$ in the s-plane.[2] We can determine the closed-loop pole locations for a given k by evaluating the contour $|G_{\text{Forward}}(s)\,G_{load}(s)| = 1/k$ and locating its intersections with the root locus.

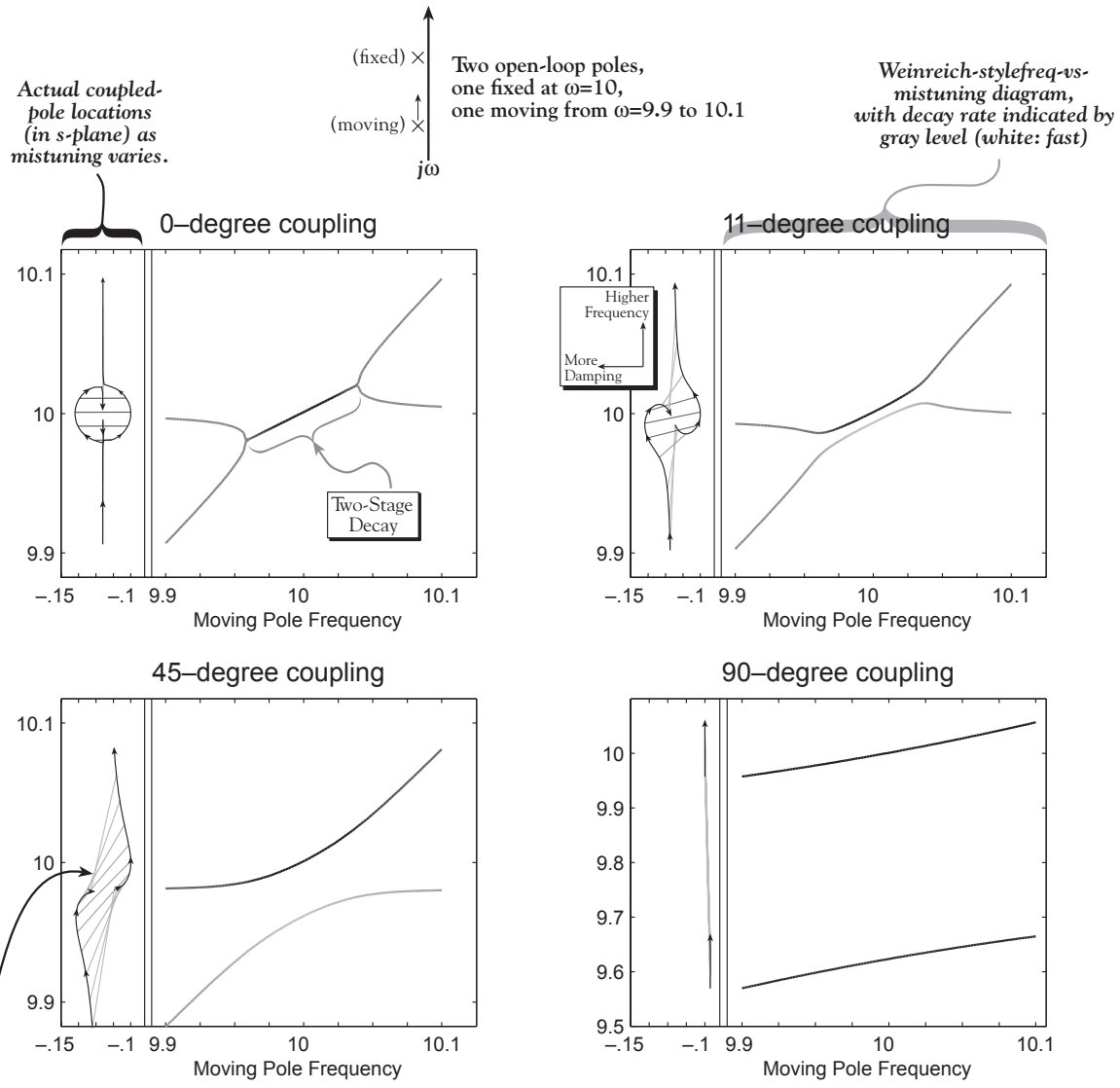
Note that this system is drawn without input or output, as we are only dealing with the system poles. The 'zeros' we will talk about are the zeros of the loop transfer function.

B.6.1 Diversion: Where are the zeros?

Since the coupling we are dealing with involves summing systems together, we expect that summation to create a set of zeros in the loop transfer function. Those seeking to build intuition on coupling might then ask "Where do the zeros land?"

The quick answer, based on what we saw in the two-pole coupling earlier, is "Between the poles." However, that is only partially true, and only really appears that way in certain circumstances, like there being only two poles, or the poles lying in certain patterns (like in a line), and with no extra phase involved. Further the gain of the poles affects how close the zeros are to the respective poles.

[2] We evaluate at $\angle(\bullet) = 0$ instead of $\angle(\bullet) = \pi$ because the system is missing the extra sign inversion in the loop that the system in Figure B.1 has due to the subtraction in the loop.

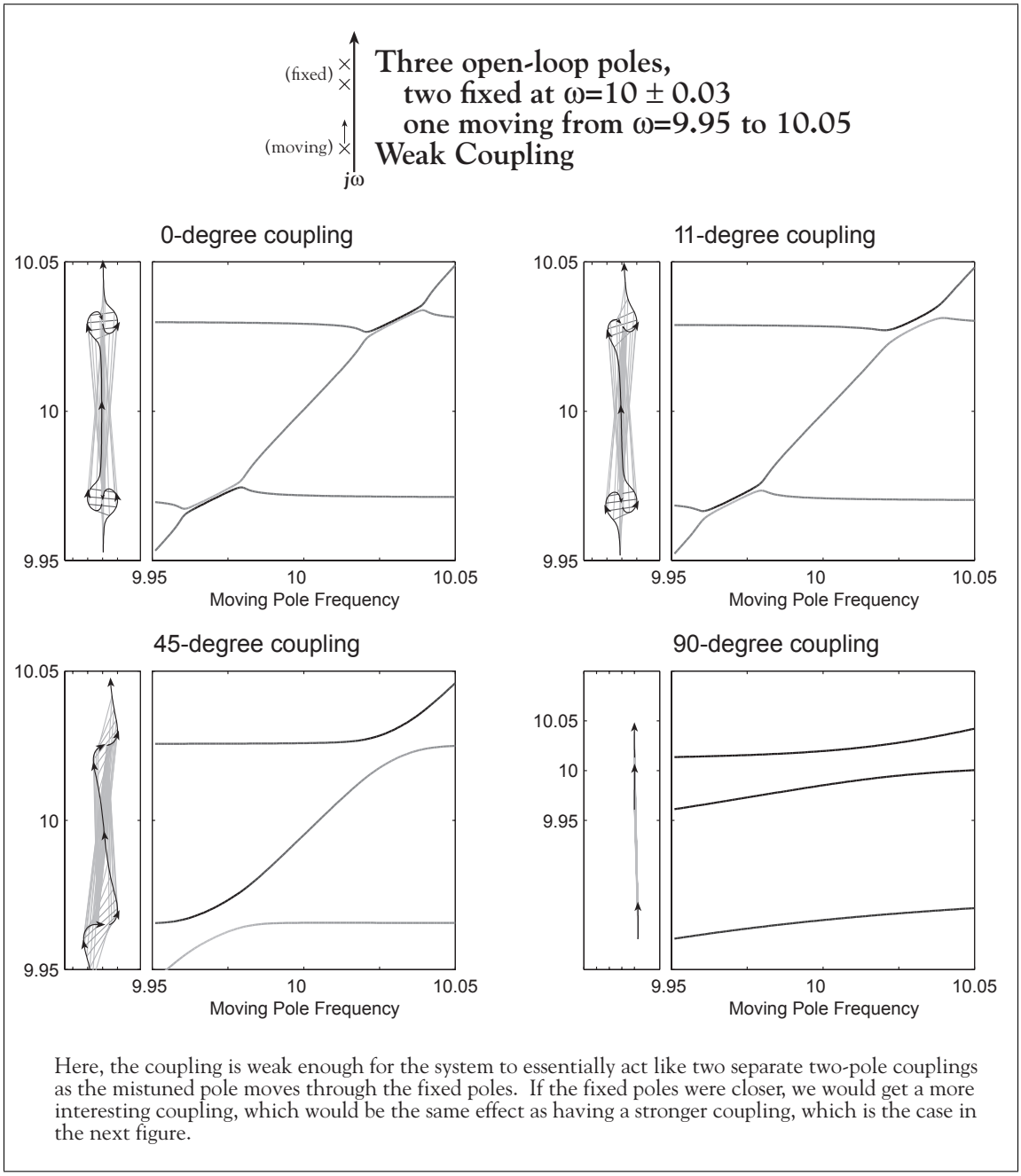


These diagrams (the ones on the left of each plot) correspond exactly to root-loci with mistuning as the free variable rather than coupling magnitude. In these root-loci, the coefficients of the closed-loop transfer function are not affine in the variable (as they are in k), so we will tend to see different patterns than in the 'standard' root loci.

It is interesting to note that the range of decay rates is independant of the coupling angle. The angle merely changes the relative freqencies of the poles at 'full coupling', keeping their rates the same. In terms of two-stage decay, this keeps the envelope of the two stage decay the same, but effects the beating pattern under the envelope.

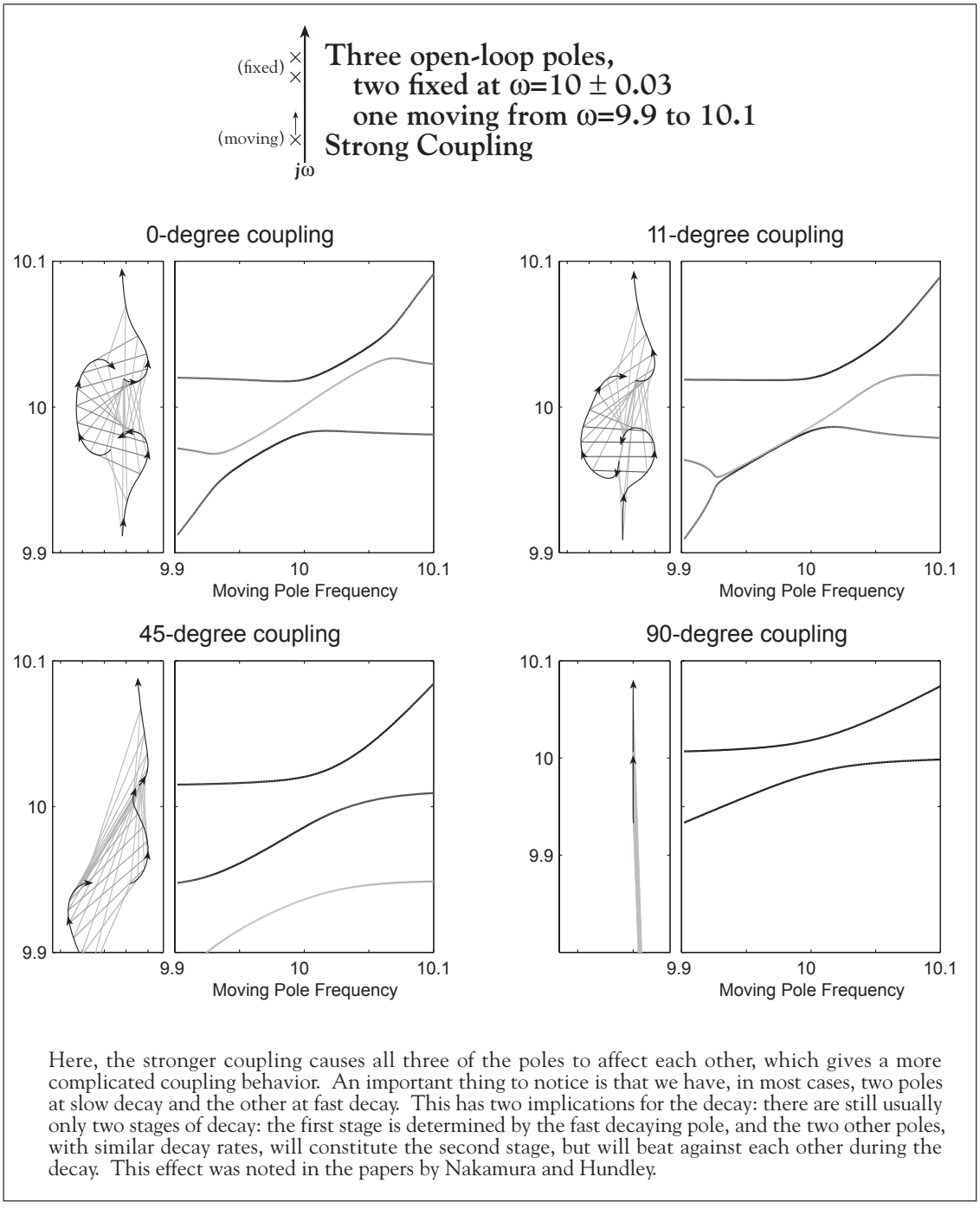
Next: we will look at 3-pole coupling. Unlike the two-pole case, where changing the distance between the poles has the same effect as changing the coupling magnitude, the presence of a third pole complicates the situation. Now, there will be more cases to look at. We will look at the following cases: (1) we will compare weak and strong coupling, and (2) we will compare the mistuning of just one pole and mistuning two poles at once. This gives us four combinations on the next four pages.

Figure B.10: Comparison with Weinreich's Frequency vs. Mistuning Diagrams. Two poles, one moving.



Here, the coupling is weak enough for the system to essentially act like two separate two-pole couplings as the mistuned pole moves through the fixed poles. If the fixed poles were closer, we would get a more interesting coupling, which would be the same effect as having a stronger coupling, which is the case in the next figure.

Figure B.11: Comparison with Weinreich's Frequency vs. Mistuning Diagrams. Three poles, one moving, weak coupling.



Here, the stronger coupling causes all three of the poles to affect each other, which gives a more complicated coupling behavior. An important thing to notice is that we have, in most cases, two poles at slow decay and the other at fast decay. This has two implications for the decay: there are still usually only two stages of decay: the first stage is determined by the fast decaying pole, and the two other poles, with similar decay rates, will constitute the second stage, but will beat against each other during the decay. This effect was noted in the papers by Nakamura and Hundley.

Figure B.12: Comparison with Weinreich's Frequency vs. Mistuning Diagrams. Three poles, one moving, strong coupling.

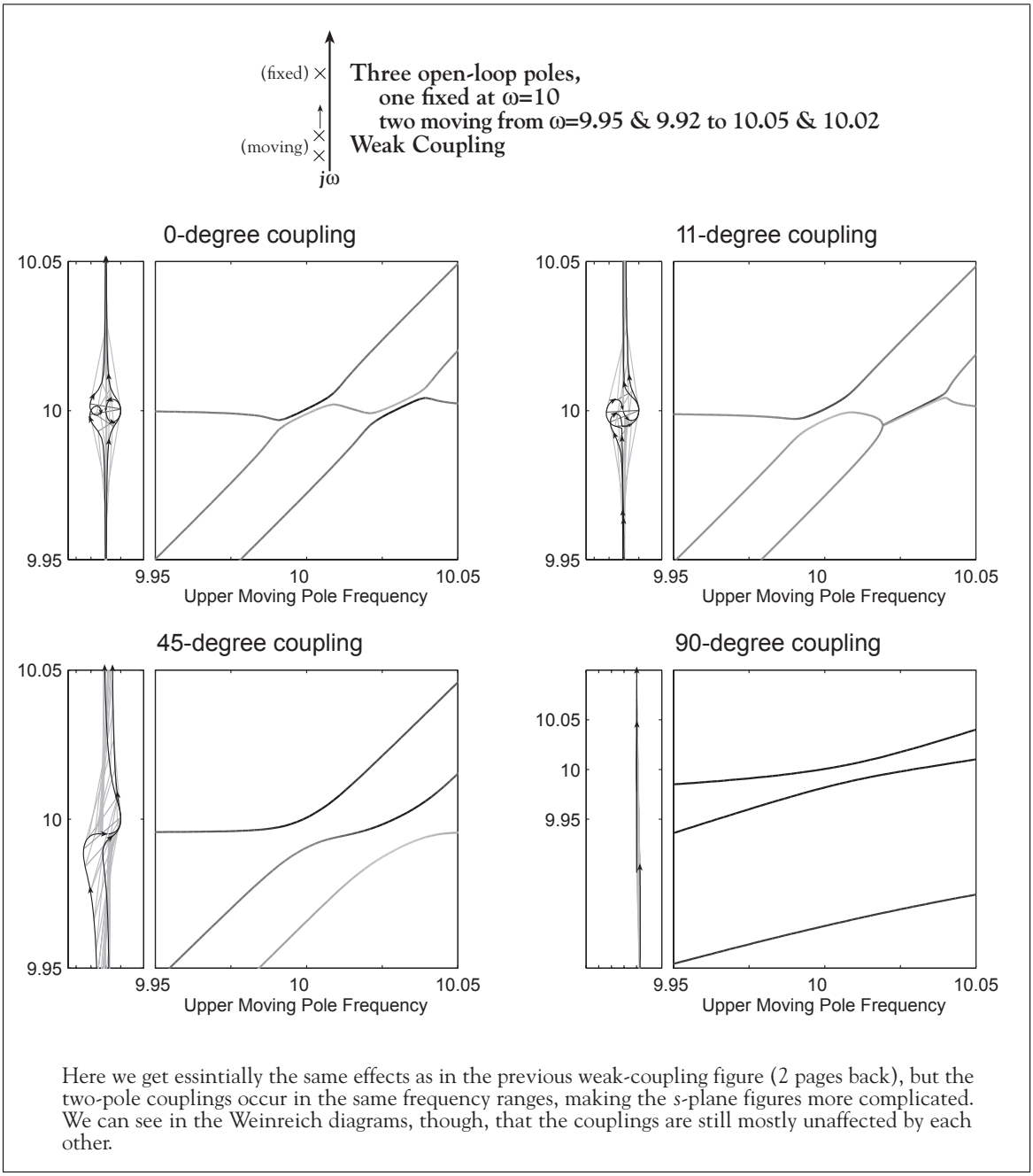


Figure B.13: Comparison with Weinreich's Frequency vs. Mistuning Diagrams. Three poles, two moving, weak coupling.

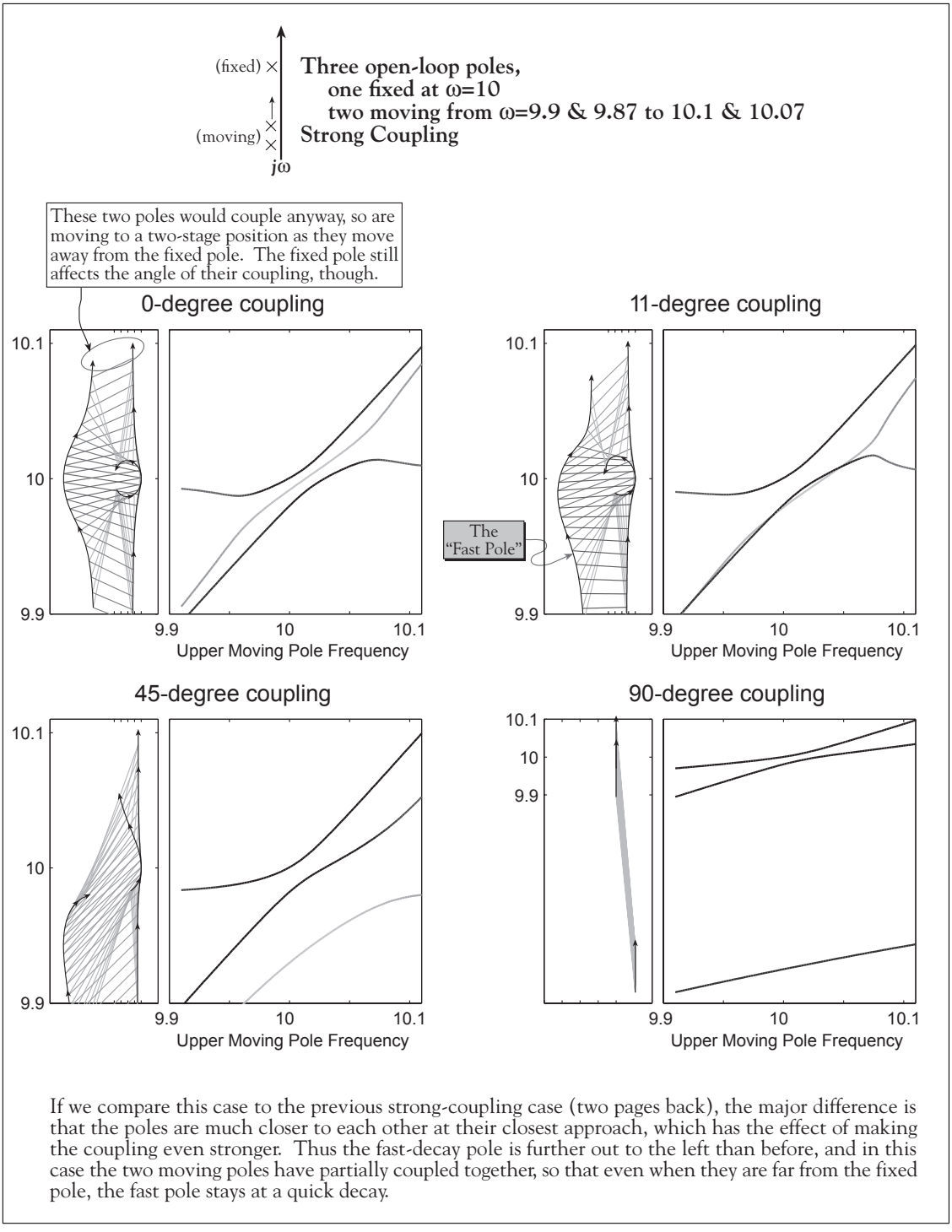


Figure B.14: Comparison with Weinreich's Frequency vs. Mistuning Diagrams. Three poles, two moving, strong coupling.

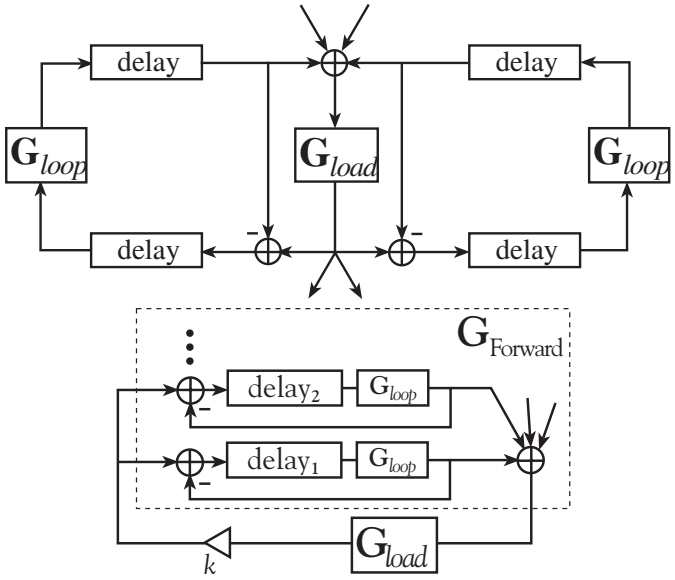


Figure B.15: Waveguide representation of N coupled strings, and rearranged into Root-Locus form

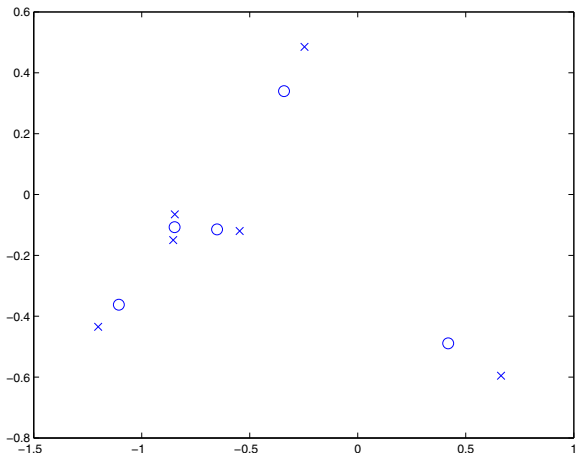


Figure B.16: Zero locations resulting from the sum of 6 random complex poles.

Let's look at the zeros of summing 6 random poles:

$$G(s) = \sum_{i=1}^{6} \frac{1}{s - p_i}$$

As we see in Figure B.16, the zeros are indeed somewhere between the poles, though because there are several poles, the zeros are in some sort of "average" locations.

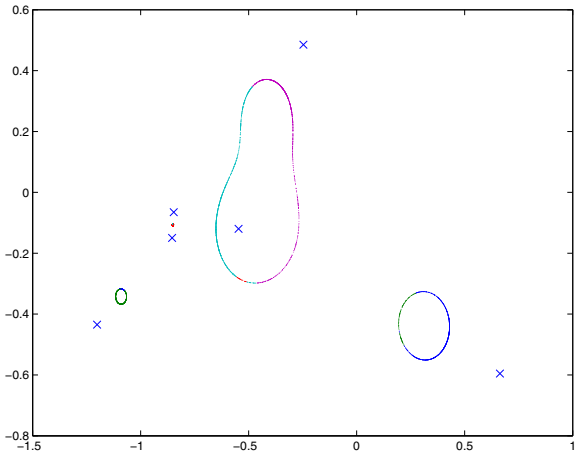


Figure B.17: Zero locations resulting from the sum of 6 complex poles, one of the poles varying phase.

Now as noted above, pole phase affects where the zeros land. Let's now look at:

$$G(s) = \frac{e^{j\phi}}{s - p_1} \sum_{i=2}^{6} \frac{1}{s - p_i}$$

And let ϕ vary between 0 and 2π (Figure B.17). Note that the zeros can move around with extra phase on just one of the poles.

If we let another pole's phase vary, then we get an interesting space (Figure B.18) over which the zeros may vary. (Varying more phases than two at a time is not easily visualizable).

Now one thing to note is that as set up, there are always one fewer zeros than poles, as a consequence of summing one-pole sections which do not have finite zeros of their own. As such, when we couple the poles together, there will be one extra pole. Most of the poles will head towards the generated zeros, but one will head away to ∞ in the direction of the overall extra phase (this can be seen in Figure B.19).

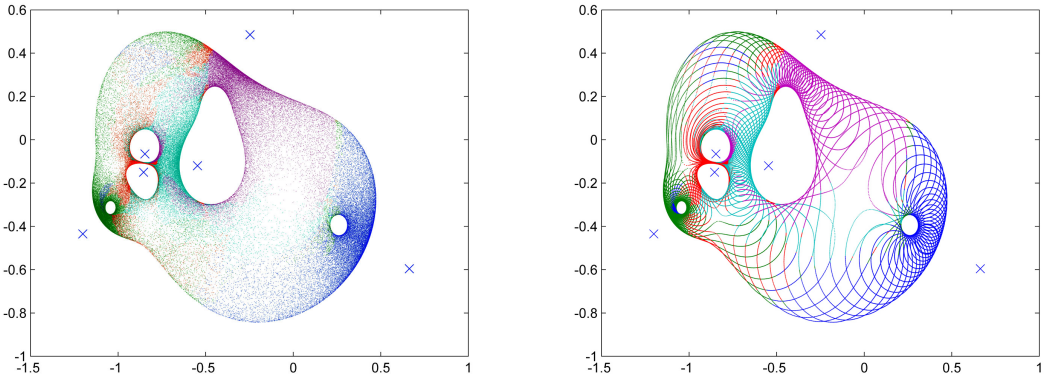


Figure B.18: Zero locations resulting from the sum of 6 complex poles, two of the poles varying phase. Left: random values. Right: gridded

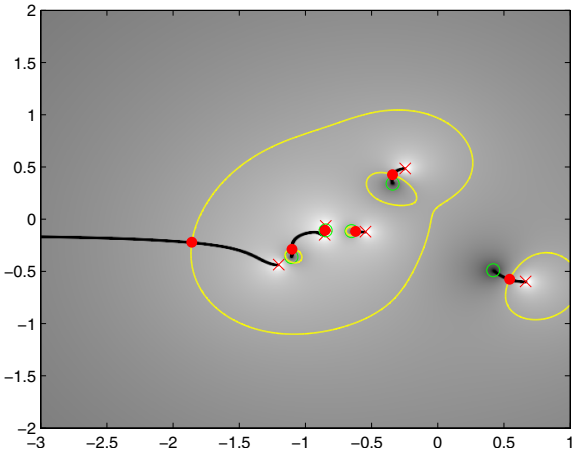


Figure B.19: Coupling root locus for a sum of poles with no extra phase, and the coupled pole locations for a particular coupling gain.

Coupled-String zeros

Now, if we again refer back to the coupled-string model (Figure B.15 and Equation B.12), we ask where the zeros will land for this situation. Now the case we are interested in consists of two (or three) strings of nearly the same tuning. As such, we would expect a series of pole pairs (or triples) at the string harmonics.[3]

Where do zeros land for a simple sum of harmonics?, like:

$$\sum_i \frac{1}{s-p_i} + \sum_i \frac{1}{s-q_i}$$

Where the p_i are the harmonic poles of one string, and the q_i are the poles of the other string. This case, shown in Figure B.20, has zeros interleaved between all the poles.

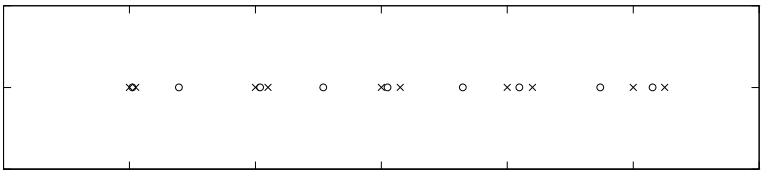


Figure B.20: Zero locations for a sum of a series of discrete poles, on the locations of harmonics of two detuned strings. (The axes are rotated so that the positive imaginary axis points to the right)

However, a string feedback section as drawn in Figure B.15 acts more like the product of its poles rather than the sum (this is verified in Figure B.23), so:

$$\frac{1}{1-e^{-sT_1}} + \frac{1}{1-e^{-sT_2}} \approx \prod_i \frac{1}{s-p_i} + \prod_i \frac{1}{s-q_i}$$

In which case, the zeros only show up between the pairs for each harmonic. None appear between the harmonics (Figure B.21).

Now, as drawn, the string delays are in their feedforward paths, so each string as an additional phase term:

$$\frac{-e^{-sT_1}}{1-e^{-sT_1}} + \frac{-e^{-sT_2}}{1-e^{-sT_2}}$$

The effect of these is to rotate the zeros to be on a different line than the poles (Figure B.22). Given that the axes are rotated with the positive imaginary axis pointing right, this means that the zeros are rotated in the positive real direction. Now if the poles themselves are offset from the axis (due

[3]Whether the harmonics are stretched or not is not an issue at the moment. We do assume, though, that if they are stretched, they are stretched more or less the same in each string, as they are made from the same material, have the same length, and nearly the same tension. As such, we would not expect stretching to have a significant effect on the relative locations of the poles within each pole pair (or triple).

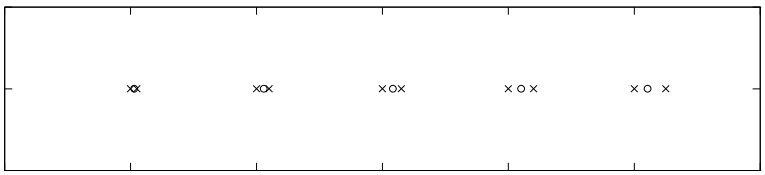


Figure B.21: Zero locations for a sum of two products of poles, on the locations of harmonics of two detuned strings.

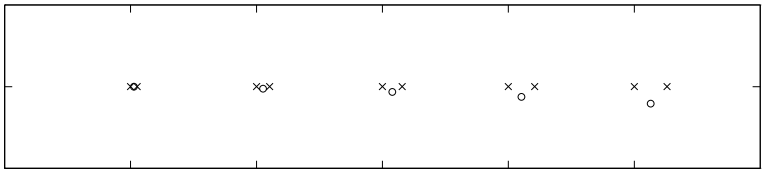


Figure B.22: String feedforward phase delays taken into account.

to damping) or rotated (due to higher damping at higher frequency), then this means that the zeros correspond to less damped locations than the poles. Therefore, with strong two-stage coupling, one of the coupled poles could end up at a slower decay than the open-loop poles. As mentioned in the introduction to this chapter, this behavior is integral to the character of the piano, because the piano sound is based partially on the "even slower" 2nd-stage decay. Otherwise, if the zeros stayed exactly between the poles, the slowest possible coupled decays would be exactly as slow as the individual strings.

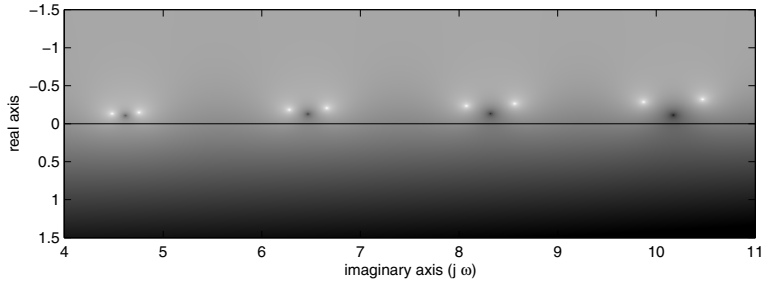


Figure B.23: Numerical experiment showing poles and zeros of summed string-like transfer functions.

As a verification of the above, Figure B.23 is a height-field image (white is more positive, black is more negative) of $\log|H_{\text{sum}}|$ from the following calculation:

$$H_{\text{del1}}(s) \quad = \quad e^{(-3.3+0.1j)s}$$

$$
\begin{aligned}
H_{\text{del2}}(s) &= e^{(-3.5+0.1j)s} \\
H_{\text{loop1}}(s) &= \frac{-H_{\text{del1}}(s)}{1 - H_{\text{del1}}(s)} \\
H_{\text{loop2}}(s) &= \frac{-H_{\text{del2}}(s)}{1 - H_{\text{del2}}(s)} \\
H_{\text{sum}}(s) &= H_{\text{loop1}}(s) + H_{\text{loop2}}(s)
\end{aligned}
$$

B.6.2 Loci

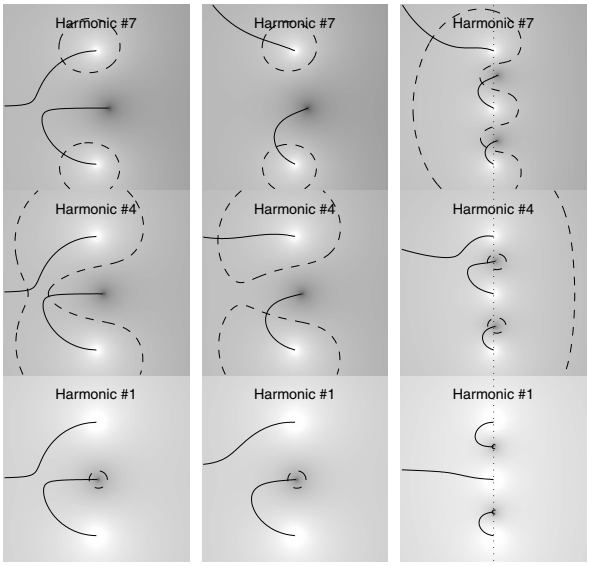


Figure B.24: String Loci, *left:* two strings, real coupling, *middle:* two strings, one-pole G_{load}, *right:* three strings, one-pole G_{load}.

In Figure B.24, we have plotted select sections of the root loci for three different systems. In the left system, we see three harmonics for the case of two-string coupling with a real scalar G_{load}. This case shows the fact that the distance between the poles will change the 'effective' coupling strength: the low harmonics, being closer, are strongly into two-stage decay, with the fast mode out of the picture to the left; whereas at the higher harmonics, the detuned open-loop poles end up further apart, so that at the seventh harmonic, the poles are still in a beating configuration. This emphasizes that different harmonics will couple differently with the same coupling constant, simply due to their pole spacing.

The middle case has G_{load} set to a onepole lowpass filter $\frac{a}{s+a}$. This coupling is now frequency dependant, so that each harmonic will see a different coupling factor. Since the phase of the onepole filter approaches –90 degrees at high frequencies, we would expect that higher harmonics will display coupling patterns that are more and more rotated, along with the distance-induced reduction

in coupling, which can be seen in the figure.

The right-hand case has three strings, along with the one-pole frequency dependent coupling. An important effect can be seen at the first harmonic, where the coupling is nearly real: the sum of the three modes has two zeros "in-between" the poles, this causes *two* of the poles to stay at slow decay when the system is strongly coupled, so that only the third pole goes into fast decay. Thus, during the second stage of decay, the two slow modes will beat with each other, an effect that is seen in ([109] and [187]).

Figures B.25 through B.28 examine several harmonics and comment on various behaviors for a couple example systems.

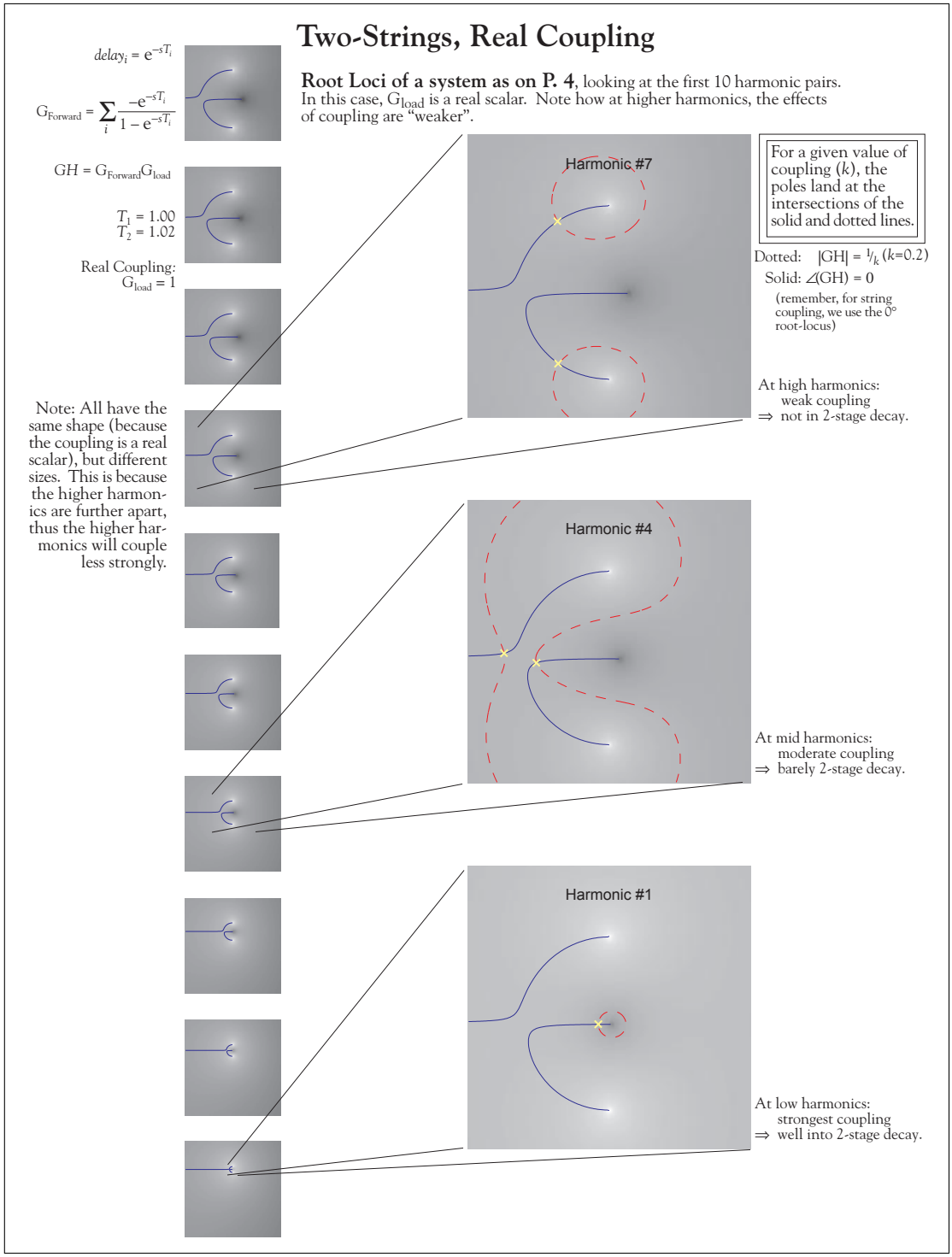


Figure B.25: Coupling Loci of two coupled strings, real coupling, no damping. Note how the low harmonics are strongly coupled, but high harmonics are not.

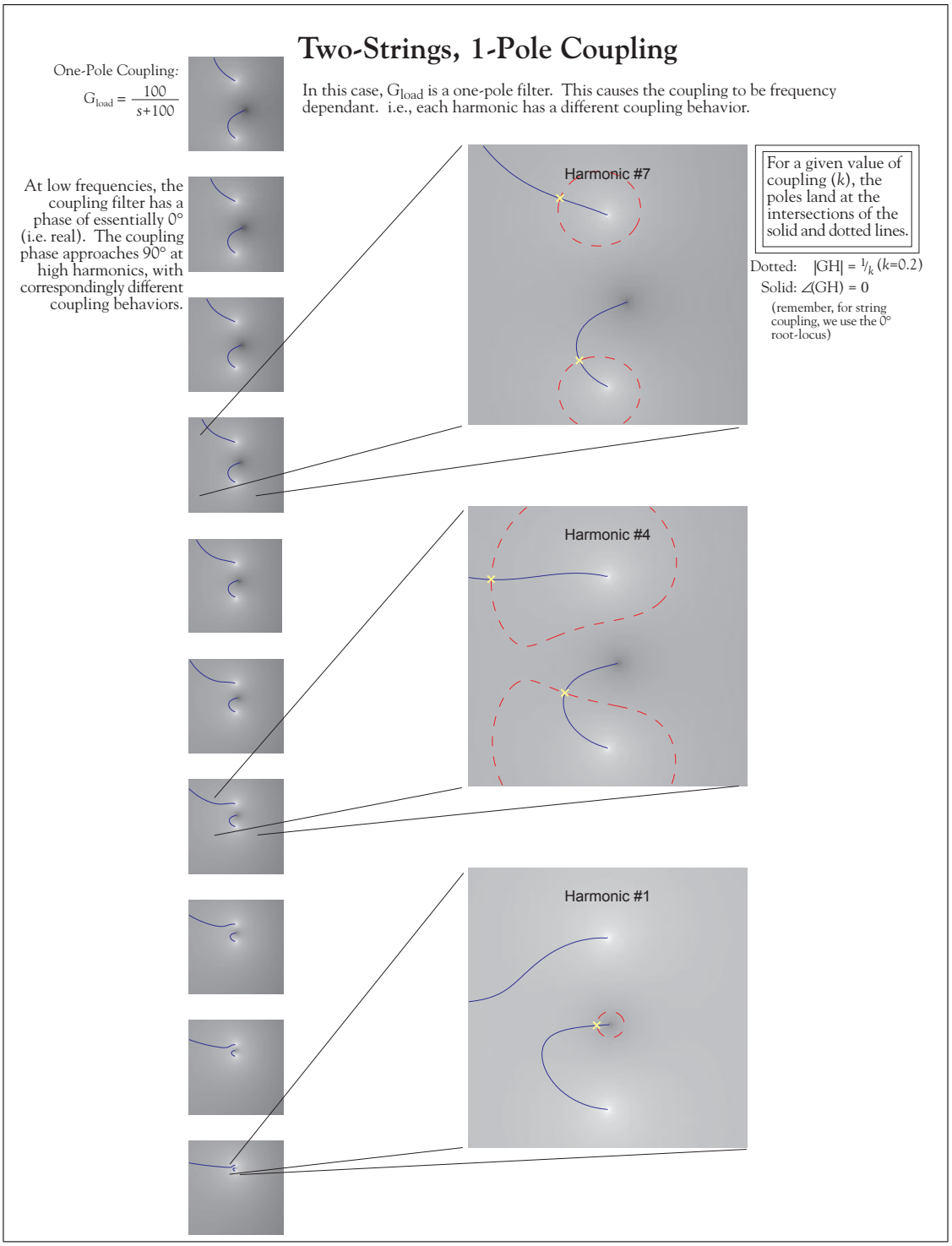


Figure B.26: Two strings, frequency-dependent coupling phase and gain.

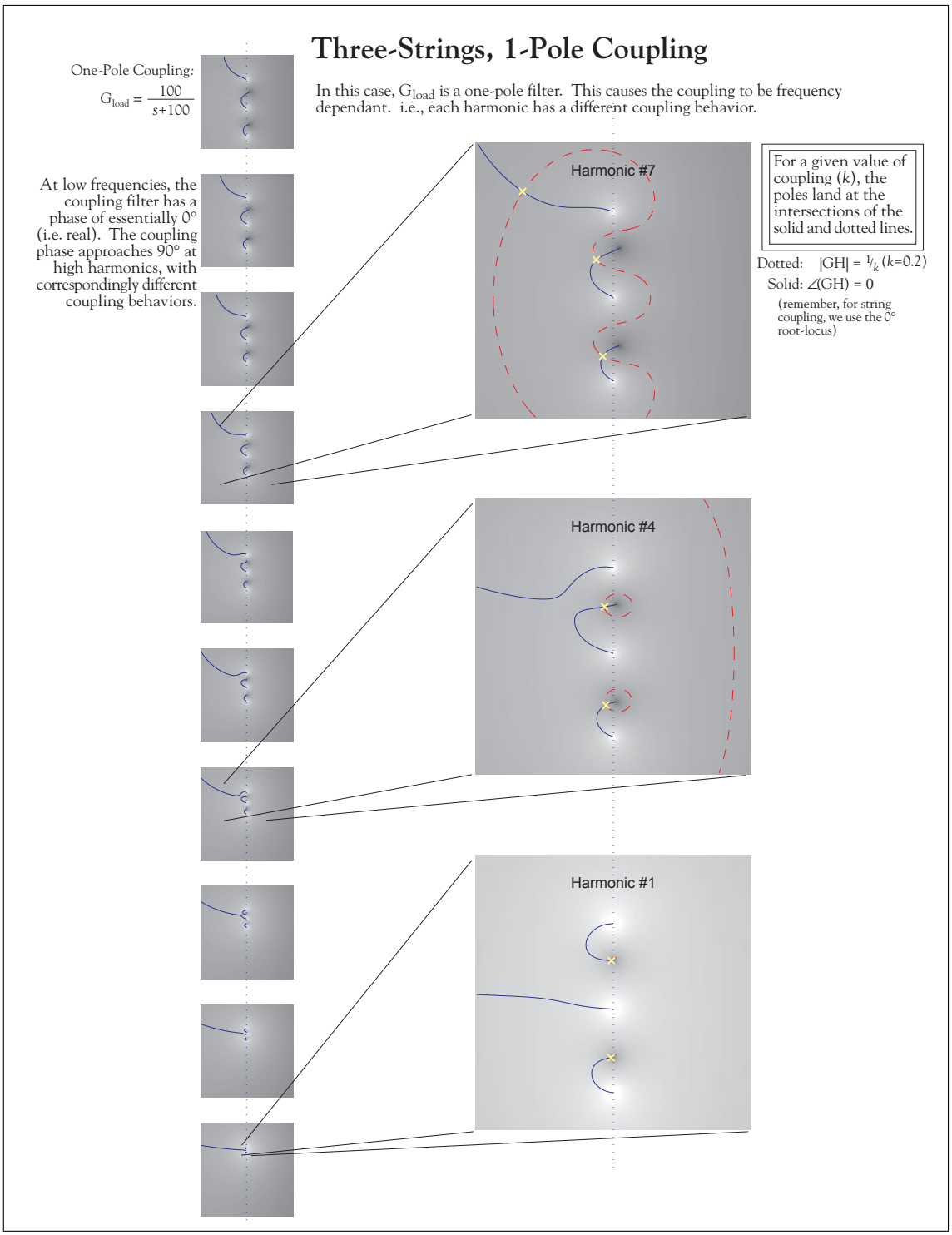


Figure B.27: Three strings, frequency-dependent coupling phase and gain.

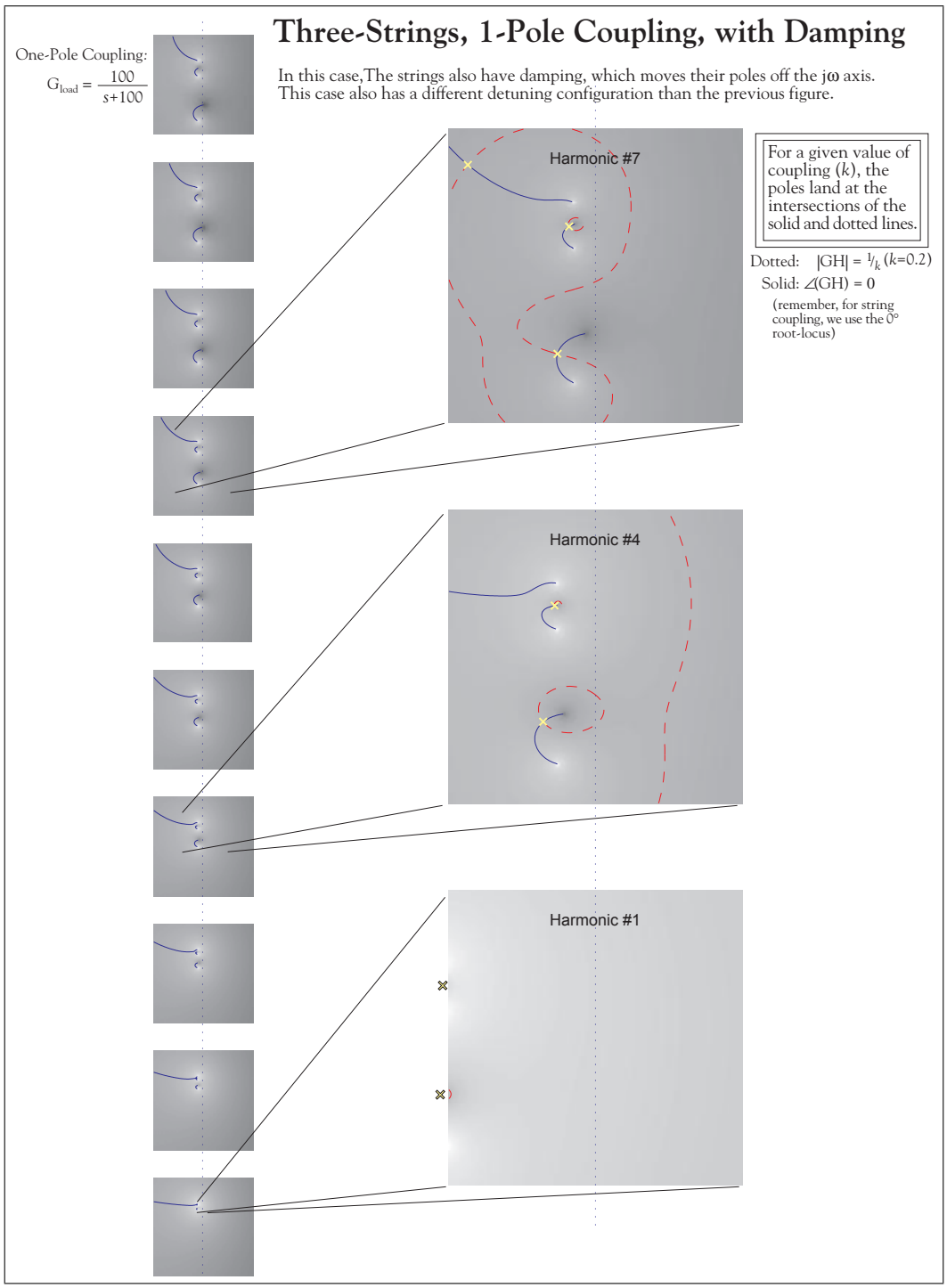


Figure B.28: Three strings, frequency-dependent coupling phase and gain, and more complicated damping.

B.7 Conclusions

The Root Locus visualizes the process of mode coupling in systems, and can provide intuition on the coupling behavior. Some important facts about multi-string coupling are deduced from the root loci:

- In 3-mode coupling, two modes stay at slow decays, this gives the beating in the second stage that has been noticed.
- Different string harmonics couple differently:
  - Higher harmonics are more detuned (absolute detuning, not relative), this makes the higher harmonics "less coupled". This can be another explanation for late-decay beating, as high harmonics may not have locked.
  - Frequency-dependent coupling causes each harmonic to couple differently. For example, each harmonic couples with a different coupling phase angle, which affects the nature of the coupling differently at each harmonic.
  - The phase of the strings causes the open-loop zeros due to the parallel combination of the string systems to be closer to imaginary axis than the open-loop poles. This allows the coupled poles in two-stage decay to have slower decay rates than the open-loop poles (i.e., than the strings individually).

Appendix C

Amplitude Control Systems in Physical Modeling and Virtual Analog

The following paper, co-authored with Harvey Thornburg, was presented as a poster at the International Computer Music Conference, Ann Arbor, Michigan, 1998.

C.1 Examples of Using Amplitude Control Systems in Music Synthesis

Feedback control systems have been used in the past to keep physical models in tune; similarly, other variables can be controlled, such as amplitude, which is explored in this paper. There can be various interesting behaviors; affecting, for example, stability and attack transient characteristics. Examples are shown for a simple waveguide model. The nonlinearity introduced by amplitude control can also approximate various characteristics of popular nonlinearities, such as saturation. In one example, a control system is applied to a linear Moog-style filter to emulate some saturation effects of the analog filter under high-amplitude conditions.

C.1.1 Introduction

Certain parameters in physical models can be difficult to control, as they are in the physical instruments they are intended to model. Such parameters can be tuning, loudness, timbre, etc. In traditional music performance, the performer must often be continually adjusting their performance

to overcome variations in the instrument over time, temperature, and various playing ranges. In trying to model these instruments for simple controls, such as MIDI, the algorithm designer must either try to remove these variations by adjusting the model according to *a priori* knowledge of how the instrument will deviate from desired performance (via derived formulae for varying certain parameters across the problematic variations, or by experimental tweaking, placed in a multitude of lookup tables), or they may try to build a dynamic system designed to react to the variations in a way similar to an actual performer. These systems are *feedback control systems*, and share much in common with more traditional control systems, such as flight control systems, and temperature control systems.

The use of a feedback control system to improve a physical model's performance with respect to difficult-to control parameters has been done previously by Perry Cook to control the intonation (among other parameters) in a brass physical model [50]. In this paper, we will explore the control of a model's amplitude. This idea can be traced at least back to Michael McNabb, in an unpublished clarinet model [165]. The control of amplitude can help simplify the algorithm designer's job, because not only can it reduce stability problems inherent in cost-reduced models, but it can assist in the implementation of a more realistic attack phase of the instrument's notes.

The amplitude control system is also explored in conjunction with a digital VCF model. Adding correct nonlinear behaviors to a digital VCF is difficult without resorting to gross oversampling. This paper describes some experiments done in an attempt to add some effects observed in overdriven analog filters to linear filters, without the bandwidth expansion (and thus aliasing) typically associated with saturation. Possible behaviors that can be modeled include stable self-oscillation and reduced effective gain at resonance, which in analog filters are by-products of a saturation nonlinearity.

Since the measurement of a signal's amplitude is a nonlinear operation, the complete system will be nonlinear, even if the underlying controlled system is linear. Furthermore, amplitude control is implemented by varying some coefficient in the controlled system (typically a simple gain), rather than by summing a control signal into the input, which would also make the complete system nonlinear. However, since amplitude measurement is usually a slow process (occurring over at least one period of oscillation), the control signal moves slowly, thereby allowing the system to be loosely thought of as locally linear. This allows the system to be understood a little more easily than most nonlinear systems.

C.1.2 Waveguide Systems

Amplitude control is of particular interest when one tries to produce extremely efficient nonlinear physical models, where absolute physical accuracy has been sacrificed in favor of cheaper computation. A typical problem encountered in working with these models is instability. The mechanisms

that self-regulated a model's amplitude can be easily broken while reducing the model's complexity. Additionally, the models were often intentionally made temporarily unstable in attempts to speed up the attacks of the notes, but the instability was not well controlled.

Early models often used simple numerical saturation to limit the effects of instability. This worked acceptably only in models which already produced a waveform that was harmonically similar to a clipped signal, such as the clarinet. A side effect was that many models ended up sounding like clarinets after simplification. The work presented here is intended to remove the need for numerical saturation in the handling of unstable models, which should better preserve the timbre of the model.

The basic implementation [246] is to set up a servo system whereby the amplitude of the system is servoed to a desired amplitude level. The control was chosen to be the loop gain (see figure), with which the controller could affect the system's amplitude via control of its stability.

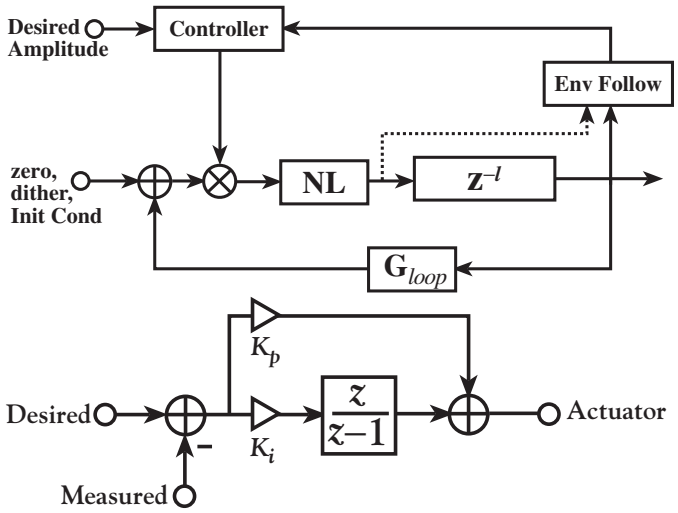


Figure C.1: Diagram of amplitude control of a typical model, along with a typical controller.

The controller shown is based on the standard *PI* (Proportional + Integral) controller [Franklin 1994]. The pole a is typically at 1.0, though using smaller values is sometimes useful. This controller has worked in practice, though it is not exactly suited for this system. See [Stilson 1996] for a short discussion of an alternative controller.

As with all control-system design, care must be taken with the controller gain. Setting the gain too high will cause the whole system to become unstable (though often still bounded). This usually manifests as a violent amplitude modulation. Thus in the design, one must trade off response speed against amplitude instability. Also note that the regions near instability are sonically interesting, as the amplitude response will 'ring' a bit before settling down, which gives the envelope a brass-like 'blip'.

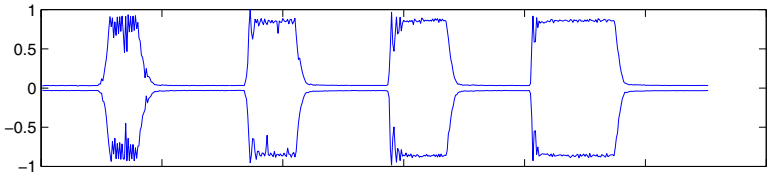


Figure C.2: Example envelopes from an amplitude-controlled waveguide model.

C.1.3 Pseudo-Nonlinear VCF

One of the more audible effects of saturation in resonant filters is the reduction of the effective resonant peak gain during resonance, due to the limited range of the filter states; therefore, a frequency-swept harmonically rich signal will pass through the filter with significantly reduced amplitude variations.

We can set about to recreate this particular effect by implementing some sort of automatic gain control. In one method, we may take the amplitude of the filter and run it through a compressor. In another method, we may use an amplitude control system. There are two choices of control variable: filter gain and filter Q. If we choose filter gain, we have simply implemented a type of compressor, but if we choose the Q control, we get a more interesting system.

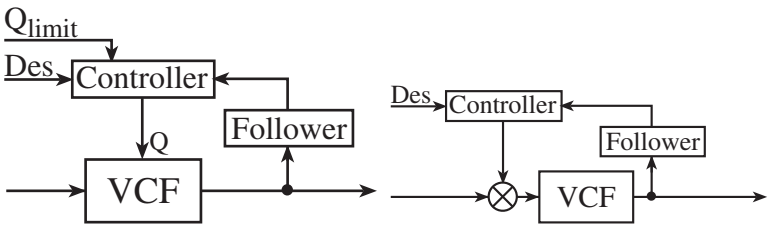


Figure C.3: Various amplitude controls applied to a linear VCF.

These amplitude-controlled filters reduce the extreme amplitude variations of the linear filter, while retaining the long impulse response of the filter for impulsive inputs, as happens in some nonlinear filters. In a linear filter, it is not possible to have both a long impulse response and a "non-peaky" frequency response in the same filter. Furthermore, a nonlinear filter can self-oscillate, due to state saturation. The amplitude control system, when using Q control, can easily cause a linear system to stably self-oscillate, and to do so without significantly adding harmonics to the signal, thereby reducing the chance of aliasing.

In Figure 4, the gain-controlled and Q-controlled systems look rather similar, but if we study Figure 5, we see that the gain control acts by attenuating the whole signal (visible in the figure by the high frequencies disappearing), whereas the Q control just reduces the peak gain, leaving the

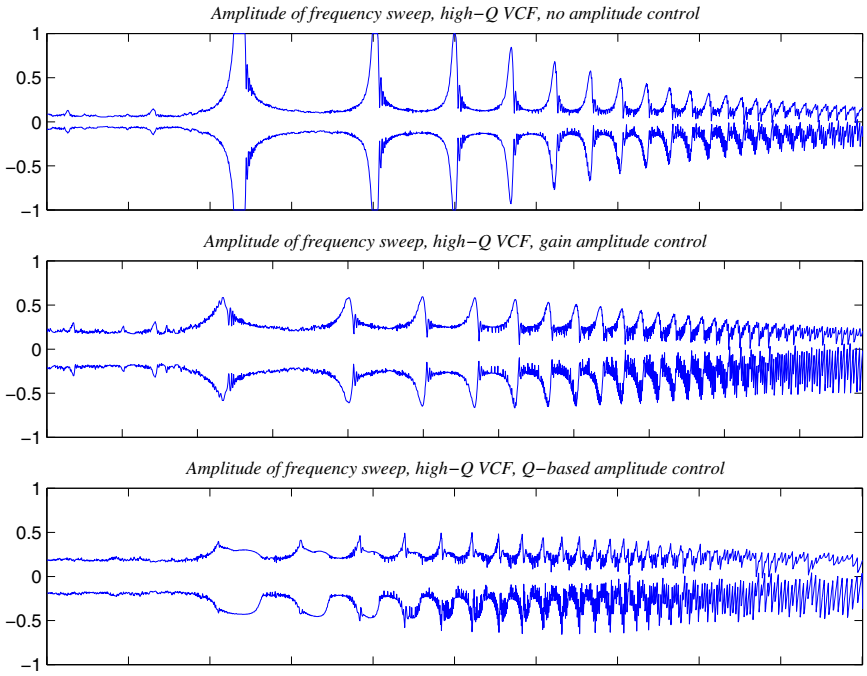


Figure C.4: Sawtooth frequency sweeps through (top to bottom): VCF, gain-controlled VCF, Q-controlled VCF.

rest of the filter relatively untouched, so that the spectral variations are less visible in the figure.

C.1.4 Summary

Amplitude control systems are an interesting addition to the palette of tools with which algorithm designers can create instruments. They can be used to bridge the gap between purely linear systems and strongly nonlinear systems, by creating a "slowly nonlinear" system, which can be easier to understand in some cases. They can also be used to help bridge between physically correct physical models and cheap, Machiavellian models. Finally, these systems open up a range of new directions for exploration in synthesis algorithms.

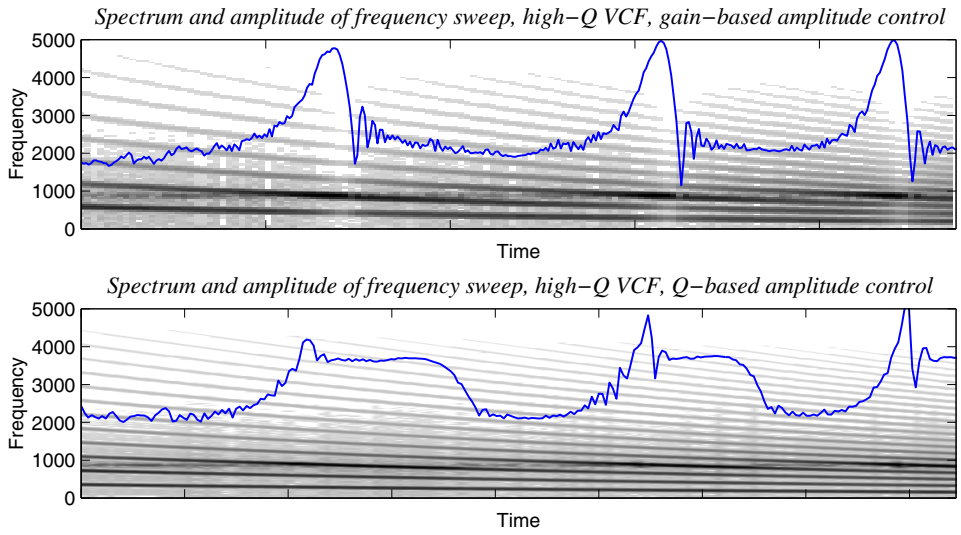


Figure C.5: Comparison of spectral effects of Q control vs. gain control.

Appendix D

Local Cosine-Coefficient Modulation

The following paper, co-authored with Scott Van Duyne, was presented as a poster at the International Computer Music Conference, Ann Arbor, Michigan, 1998.

D.1 Implementing Efficient Frequency Variation in Coupled-Mode Synthesis and Other Cosine-Frequency Systems

Many oscillator and filter algorithms are tuned via coefficients that are of the form $\cos(2\pi f/f_s)$. This can make implementing frequency modulation, and pitch-bend expensive, at least one table lookup per frequency modification. In this paper, an approximation to the cosine is explored which allows for direct modulation without cosine re-computation or table lookup. With this ability, fast and/or smooth time-variation of mode frequencies is possible without significant increase in expense, while retaining good low-deviation tuning accuracy.

D.1.1 Introduction

Quite a few oscillator and filter algorithms, including the direct-form second- order filter, the second-order digital waveguide oscillator [241] and, more recently, the coupled-mode synthesis filter [71], are tuned via coefficients that are related to frequency by a cosine operation. This can make implementing frequency variation, such as FM, vibrato, and pitch-bend expensive, requiring at least a good table lookup per frequency modification. In this paper, approximations to the cosine are explored, which allow for direct modification of the coefficients without cosine re-computation or table lookup.

With this ability, fast and/or smooth time-variation of mode frequencies in Coupled-Mode Synthesis is possible, allowing for some interesting effects, both physical and non- physical. The range

of non-physical effects is quite wide, while possible physical effects may include: simulating amplitude dependent modal frequencies in drums; simulating the pitch sliding effects of moving the pedal on a kettle drum; and simulating the sounds of water-filled shells, whose modal frequencies change as the water moves about.

In Section 2, the approximation will be derived. In Section 3, its frequency-deviation characteristics will be explored. In Section 4, more trivial deviations will be compared.

D.1.2 Derivation

Our goal is to implement the cosine coefficient in such a way as to be able to modify the coefficient to implement frequency variations in as cheap a way as possible while still keeping as much useful control over the exact variation as possible (i.e., the cosine approximation should be good). In keeping with the cheapness goal, it will be assumed that the frequency variation is based on a signal input to the system, so that is changes as fast as once per sample (this creates a context for the definition of "cheap"). It is also assumed that a table lookup is too expensive (either due to interpolation computation or memory access), so as to motivate a search for another method.

First we will define the frequency variation. We have chosen a multiplicative deviation, as it fits musical requirements: new freq $= \alpha f$, where f is the base frequency at which the original coefficients are computed (f is assumed to be change "rarely", so that exact cosine computation can be done when it changes), and α is the frequency deviation ratio. Thus our base coefficient c is defined as $c = \cos(2\pi f/f_s)$, and our deviated coefficient $\tilde{c}$ should approximate the value $\cos(\alpha\, 2\pi f/f_s)$. For simplicity, let $\theta = 2\pi f/f_s$, so that $c = \cos(\theta)$. θ varies by the same ratios as f, so that αf corresponds to $\alpha\theta$ (i.e., $\tilde{c}$ must approximate $\cos(\alpha\theta)$).

First, we look at the polynomial expansion of cosine ([2] for example):

$$c = \cos(\theta) = 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \frac{\theta^6}{6!} + \frac{\theta^8}{8!} - \ldots$$

The deviated true coefficient is:

$$\cos(\alpha\theta) = 1 - \frac{\alpha^2\theta^2}{2!} + \frac{\alpha^4\theta^4}{4!} - \frac{\alpha^6\theta^6}{6!} + \frac{\alpha^8\theta^8}{8!} - \ldots$$

Define a new coefficient $d = c - 1$, so that $c = 1 + d$, thus:

$$d = -\frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \frac{\theta^6}{6!} + \frac{\theta^8}{8!} - \ldots$$

We now introduce our approximation:

$$\tilde{c} = 1 + \alpha^2 d = 1 - \frac{\alpha^2\theta^2}{2!} + \frac{\alpha^2\theta^4}{4!} - \frac{\alpha^2\theta^6}{6!} + \frac{\alpha^2\theta^8}{8!} - \ldots$$

This approximation deviates first all the way at the fourth order, and if α is near 1.0 (i.e., small deviations), $a^n \approx a^2$ for small n, thus increasing the accuracy of the approximation a further. Also, this approximation is better for small θ, which works well because most useful frequencies correspond to rather small θ at f_s = 44100Hz.

We can use this approximation by updating c via $1 + \alpha^2\, d$, or if updating coefficients is difficult (as it can be in some situations), by replacing all multiplications by c with an add and another multiply (see Figure D.1). Note that if α is updated every sample, there is no reason to use the former method over the latter, since in the former case, the coefficients would have to be stored to memory. This storage is a useless operation since the new value is useful only in the current sample, because it will be recomputed in next sample anyways.

In multi-mode systems, like coupled-mode synthesis, the same value of α is likely to be used for all modes, which means α^2 need only be computed once per sample rather than once per mode per sample.

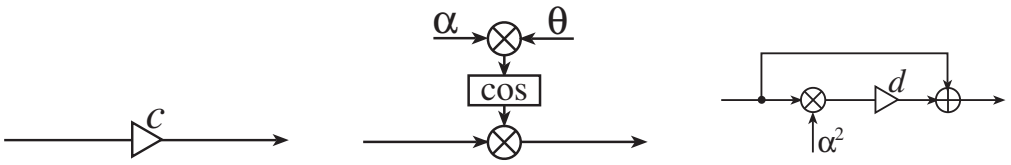


Figure D.1: Modulating c, l to r: no modulation, exact modulation, $1 + \alpha^2\, d$ approximation.

When used to modulate high-Q filters, care must be taken in the modulation, because the time-varying filter could become unstable (even if all intermediate states are stable in steady state). If this is likely to be a problem, the filter should have safeguards implemented, such as state clipping. Also, when performing large deviations near $c = 1$ or $c = -1$, $|\tilde{c}|$ may get larger than 1.0. In many systems, this will represent an unstable system, so coefficient clipping may also be necessary in certain situations.

D.1.3 Evaluating the Approximation

The accuracy of the approximation depends mainly on two parameters: θ and α. At low base frequencies (small θ), and/or at low frequency deviations (α near 1.0), the approximation will be very good, getting progressively worse as the base frequency increases and the modulation deviation increases.

In Figure D.2, we compare the approximation to the desired values for α in the range [0,2], corresponding to the range from DC to up one octave. Since the approximation also depends on θ, we show the approximations at various base frequencies. The first set of graphs show compare c and $\tilde{c}$, the next compare actual tuning based on $\tilde{c}$ against desired tuning, and the third set shows

the deviation in the actual tuning from the desired tuning, in cents.

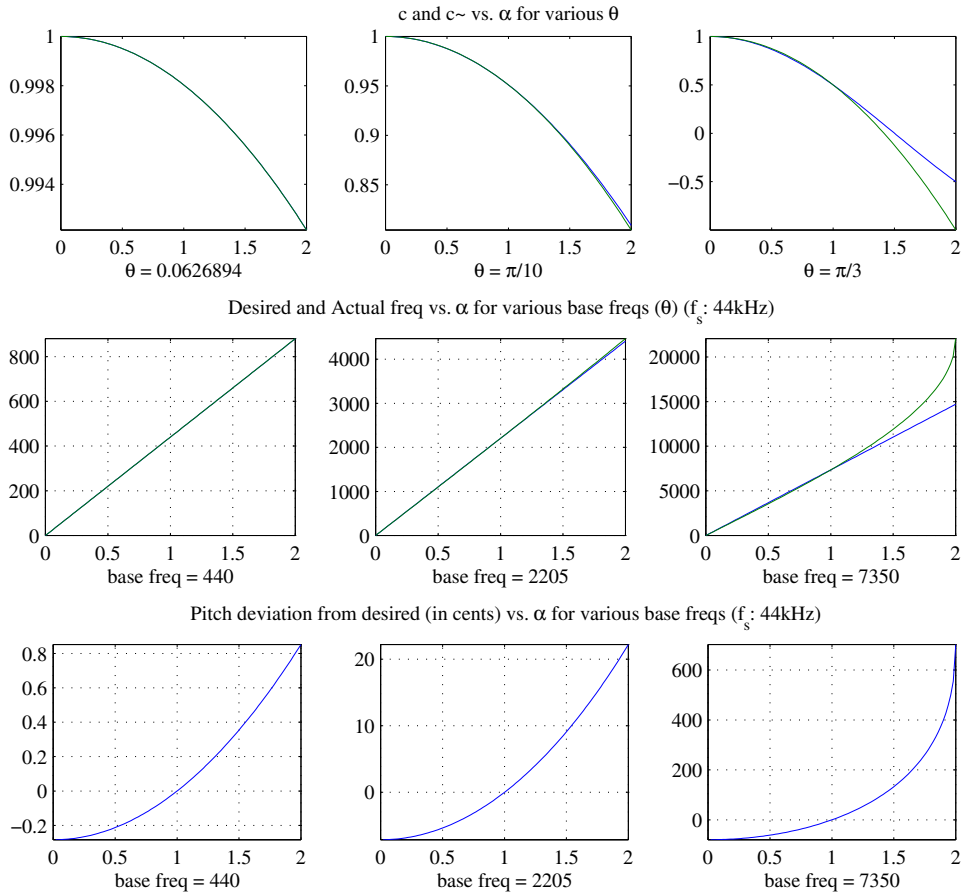


Figure D.2: Evaluating the approximation across frequency ratios (α) from 0 to 2 (up one octave)

The figure shows that for musically useful fundamental frequencies, the approximation works well, especially for frequency deviations of less than an octave. If, however, we are using this approximation to deviate a harmonic of a musical signal, the approximation may be less useful. As an example, if we take a set of partials that are harmonically related at the base frequency, and deviate them, they could end up significantly inharmonic at the extremes of the deviation.

D.1.4 Other Methods

In most of the algorithms mentioned at the top of the paper, the tuning coefficient enters into the system in such a way that any value of $-1 \leq c \leq 1$ will represent a valid frequency. Therefore, even more trivial ways of varying c will produce frequency deviation, such as $\tilde{c} = c + \alpha$ or $\tilde{c} = \alpha\, c$ (with appropriate clipping to [-1,1]). These variations will be much more approximate than the above

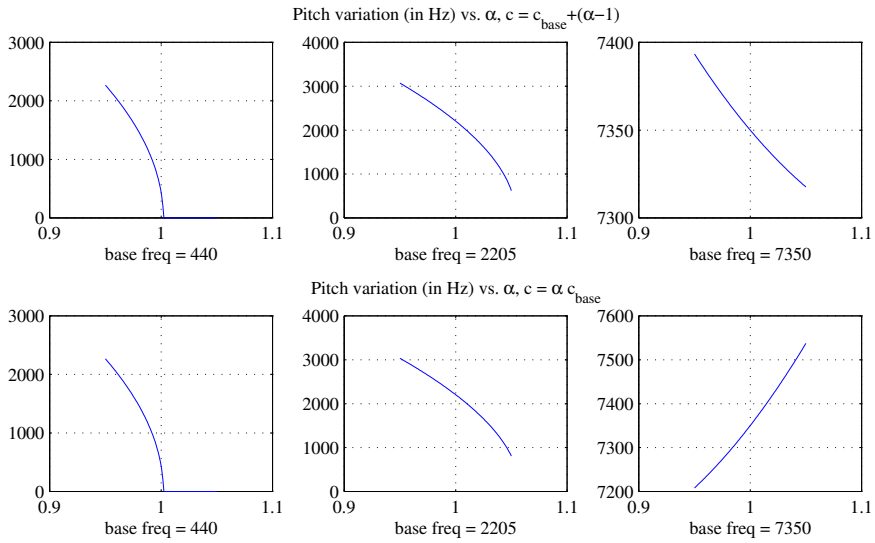


Figure D.3: Frequency deviation characteristics for more trivial coefficient variations.

approximation, as shown in Figure 3. Note, in particular, that the range of useful α now becomes a function of the base frequency, which makes the use of these variations more tricky. Certain applications don't require accurate tuning, though, so these trivial variations need not be rejected offhand.

D.1.5 Summary

An approximation of cosine, useful for small-ratio deviations from precomputed values, was presented. The approximation is intended for use in implementing inexpensive frequency modulations in cosine-tuned oscillators and filters. Numerical evaluation of the approximation shows its usefulness for most musically useful frequencies and variations (such as vibrato).

Appendix E

On The Classic Allpass Filter Forms

E.1 Relating Standard AP Forms and IIR Direct Forms

It is well known ([195], [170], [238]) that there are four standard "Direct Form" implementations of IIR filters: DFI, which corresponds directly to the simplest way of writing the transfer function, DFII which uses the minimum number of delays, and their transposes (Figure E.1). What is less

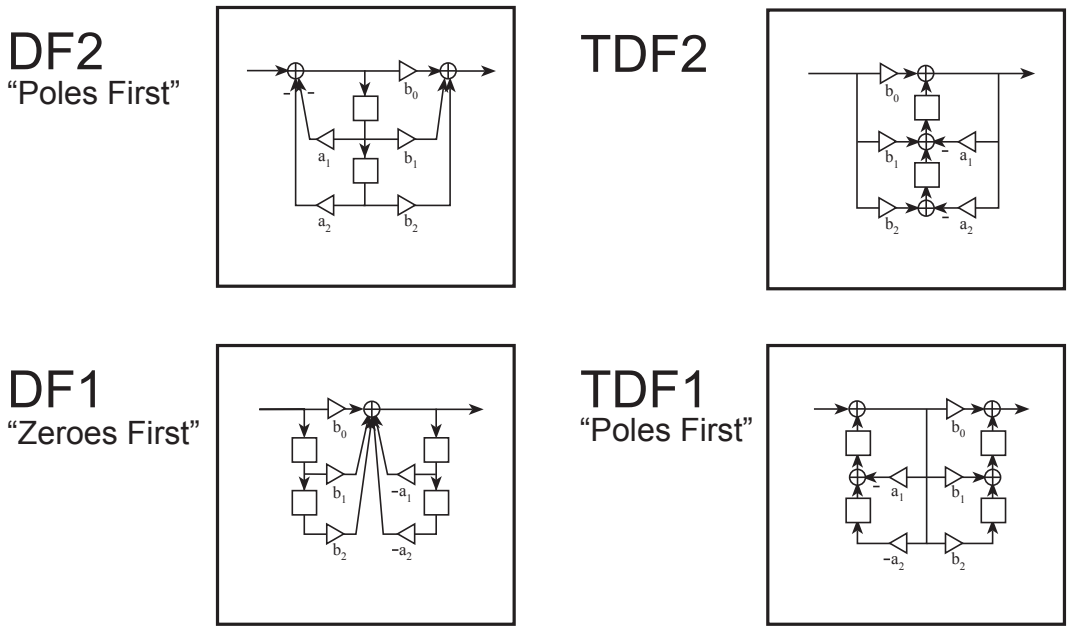


Figure E.1: The Standard Direct Form Biquad Implementations

well-known is that there are also four ways of drawing a basic allpass filter (among others, like the various lattice forms, though those forms often correspond to these forms as well), as seen in Figure E.2.[1] These are essentially the four combinations of the two following choices:

- What goes on the center line, a single scale or a single delay? If a scale is on the center line, there will be delays in the feedforward and feedback. If a delay is on the center line, there will be scales in the feedforward and feedback.

- Do we pick off the feedforward and feedback from inside the sums or outside the sums?

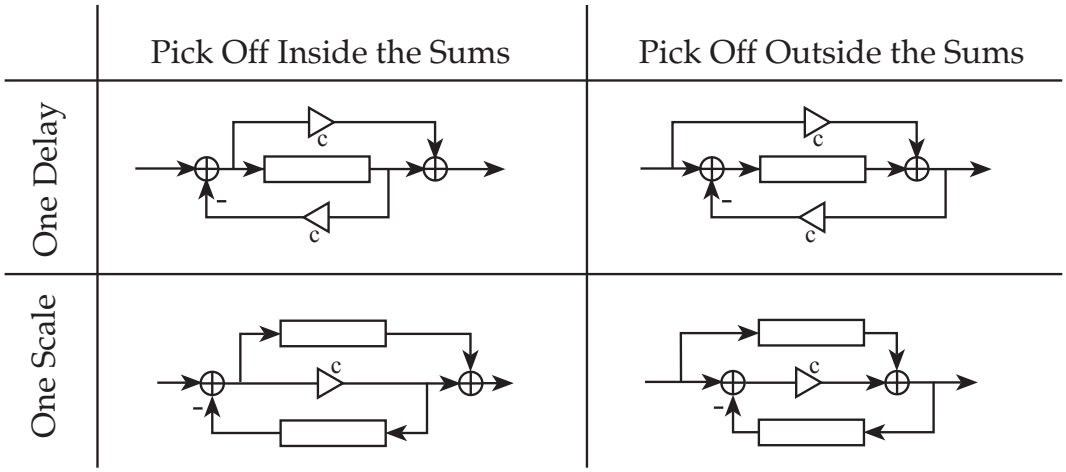


Figure E.2: Four Standard ways of Implementing a "First-Order" Allpass Filter.

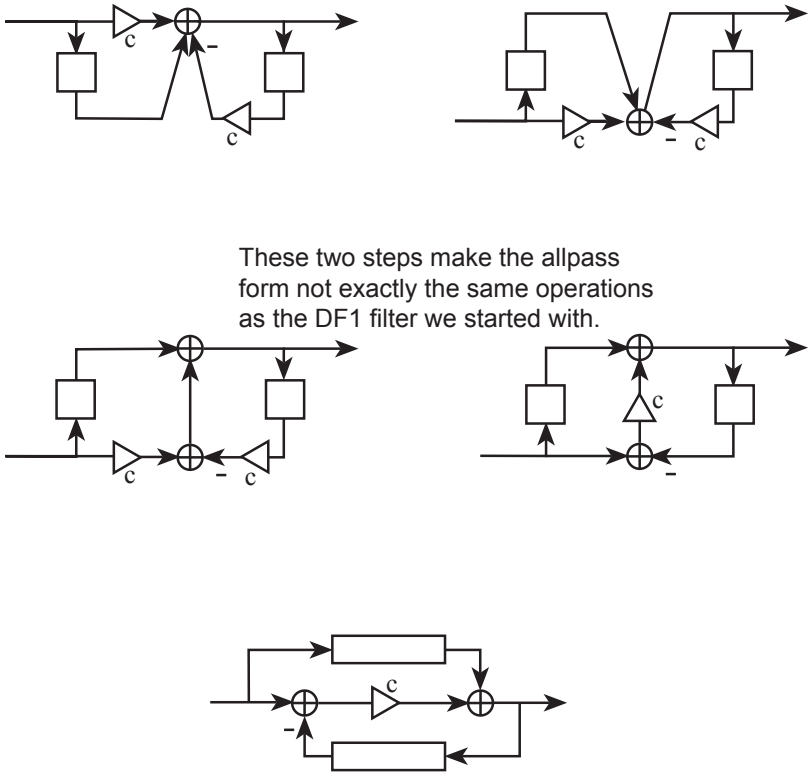
These allpass forms can be related to the IIR direct forms, though only two of them are exactly equivalent. We will note the equivalence by taking each of the direct form implementations of an allpass filter, and by block-diagram algebra, turning it into one of the allpass forms.

E.1.1 DF1

First-order DF1 corresponds to putting the scale on the midline and picking off outside the sums.

(Remember, for an allpass filter, we flip the denominator around to get the numerator. Hence, $N1 = 1, D1 = N0 = c$.)

[1] There are also one-multiply versions for all four of these [Ref Mitra textbook 3rd ed, sec 8.6.1]

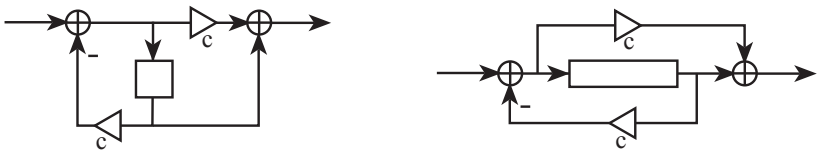


Note that there are two algebraic steps which make the allpass form a slightly different implementation than the DF1 filter, though the 'zeros first' property still holds. The difference is basically the difference between $w = x + cy - cz$ and $w = x + c(y - z)$ (i.e.,two sides of the distributive property).

E.1.2 DF2

First-order DF2 corresponds to putting the delay on the midline and picking off inside the sums.

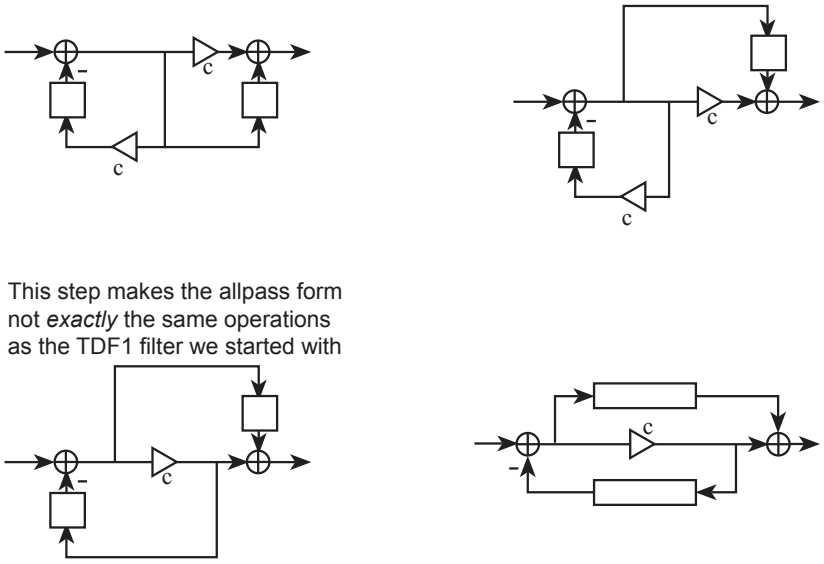
Note that The DF2 allpass form is exactly equivalent to the DF2 IIR form, there are no algebraic differences:



E.1.3 TDF1

First-order TDF1 corresponds to putting the scale on the midline and picking off inside the sums.

Like the DF1 case, there is a algebraic small difference, though in this case, it is extremely minor.



The difference here is between $w = x + cy$ and $v = cy, w = x + v$. That may only show up in architectures which have accumulators:

```
acc = x; // acc has extended precision compared to the variables
acc += c*y;
w = acc;
```

versus:

```
v = c*y; // v, not an accumulator, may lose precision in the product
w = x + v;
```

Of course, one could always reorder the operations as such:

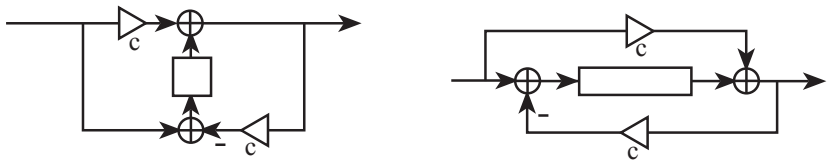
```
acc = c*y;
acc += x;
w = acc;
```

Which reinforces that the difference between the TDF1 IIR form and its corresponding allpass form is very small.

E.1.4 TDF2

First-order TDF2 corresponds to putting the delay on the midline and picking off outside the sums.

As with the DF2 allpass form, the TDF2 IIR and allpass forms are equivalent:



Reviewing, we get some rules of thumb for the first-order allpass forms:

- Transposing is effectively the same as switching between picking off inside and outside the sums.
- If we pick off inside the sums, we get "poles first", if we pick off outside the sums, we get "zeros first".
- DF1/TDF1 have the scale on the midline, DF2/TDF2 have the delay on the midline.

E.2 Allpass Internal Gains

It is well-discussed that the "zeros-first" direct-form filters (DF1 and TDF2) have less probability of internal overflow relative to the output level in fixed-point implementation than the "poles-first" forms (DF2 and TDF1) ([56][238]). This applies to allpass filters as well, though it does not appear to have been discussed as much (Dattorro does discuss it in [57]).

Remember that when analyzing filters for overflow, the L_1 norm of the filter is the norm of choice, as it measures the worst-case instantaneous gain of the filter, over all possible inputs. On the other hand, the L_2 norm (which is the peak of the frequency response) only gives us the maximum gain for sinusoidal inputs. It can be shown that the signal which will produce the maximum instantaneous gain

$$x_n, |x_n| \leq 1 \; s.t. \; \max_n(|y_n|) \text{ is maximized.} \tag{E.1}$$

is the impulse response of the filter, time-reversed, and run through a sgn() function (assuming the signals and coefs are real, otherwise the inputs would be unit-amplitude phase rotations $e^{-j\angle h_i}$). At the end of this signal being played into the filter, the output of the filter will become for one sample:

$$y_n = \sum_{i=0}^{N} |h_i| \tag{E.2}$$

Which is the definition of the L_1 norm of filter's impulse response (where N in this case is the length of the impulse response h_n).

E.2.1 Allpass I/O Gain (first-order)

Let us now look at the L_1 gain from the input to the output of a first-order allpass filter (we will be assuming real signals and coefficients for the rest of this section):

$$h_i = \begin{cases} 0 & : \quad i < 0 \\ c & : \quad i = 0 \\ (1-c^2)(-c)^{i-1} & : \quad i \geq 1 \end{cases} \tag{E.3}$$

Thus:

$$\begin{aligned} \sum_{i=0}^{\infty} |h_i| &= |c| + (1-c^2) \sum_{i=1}^{\infty} |c|^{i-1} \\ &= |c| + \frac{1-c^2}{1-|c|} \\ &= 1 + 2|c| \end{aligned} \tag{E.4}$$

Thus, the maximum instantaneous gain from input to output of a first-order allpass filter start at 1 for $c = 0$, and grows to 3 as $|c| \to 1$. This, by itself, can have implications on fixed-point implementation, if one desires absolute absence of overflow.

Now let's look at the internal gains for the DF2 and TDF2 allpass filters. The internal gain will be defined as the gain from the input to the input of the delay.

E.2.2 DF2 internal gain (first-order)

Let u_n be the input to the delay, then:

$$\begin{aligned} u_n &= x_n - cu_{n-1} \\ y_n &= cu_n + u_{n-1} \end{aligned} \tag{E.5}$$

Hence,

$$\frac{U}{X} = \frac{1}{1+cz^{-1}} \tag{E.6}$$

Which has the simple impulse response:

$$\begin{aligned} h_i &= \begin{cases} 0 & : \quad i < 0 \\ (-c)^i & : \quad i \geq 0 \end{cases} \\ \sum_{i=0}^{\infty} |h_i| &= \sum_{i=0}^{\infty} |c|^i = \frac{1}{1-|c|} \end{aligned} \tag{E.7}$$

This gain goes to infinity as $|c| \to 1$ (see Figure E.3). In other words, as well discussed: internal

saturation can happen easily in DF2 for "strong" poles.

Of course it is an accepted rule of thumb that designing for such worst-case gains often results in filters whose gains are reduced so far as to have significantly reduced SNR as well. Even so, if one designs their filter to handle lower gains than the L_1 worst-case, the use of a "zeros-first" form like TDF2 will reduce the probability of overflow. As usual, in these cases, one should implement with saturating arithmetic (as opposed to letting wraparound happen), so that if overflow does occur, it will be handled with much less annoying artifacts.

E.2.3 TDF2 internal gain (first-order)

$$\begin{aligned} u_n &= x_n - cy_n \\ y_n &= cx_n + u_{n-1} \end{aligned} \tag{E.8}$$

Hence,

$$\frac{U}{X} = \frac{1-c^2}{1+cz^{-1}} \tag{E.9}$$

Which has the impulse response:

$$\begin{aligned} h_i &= \begin{cases} 0 & : \quad i < 0 \\ (1-c^2)(-c)^i & : \quad i \geq 0 \end{cases} \\ \sum_{i=0}^{\infty} |h_i| &= (1-c^2) \sum_{i=0}^{\infty} |c|^i = \frac{1-c^2}{1-|c|} = \frac{(1-|c|)(1+|c|)}{1-|c|} = 1+|c| \end{aligned} \tag{E.10}$$

Thus, the worst-case internal gain of a first-order TDF2 allpass starts from 1 at $c = 0$, and goes up to 2 at $|c| \to 1$. Note that this is actually less than the worst-case external gain of the filter (Figure E.3). As such, this allpass form is particularly well suited for fixed-point implementations. If implementing hierarchical allpass filters (allpass inside another allpass, etc.), one may still want to take this gain into account, as the internal gains of the inner filters should grow as $(1+|c|)^{\text{depth}}$ (if they all have the same coefficient, or $(1+|c_1|)(1+|c_2|)\ldots$.

This analysis also directly applies to 'first-order' allpass filters which have the unit delay replaced by longer delays (as commonly used in some reverberation algorithms), as their responses are the same as first-order responses, but with runs of zeros between each sample. As such, the infinite sums of their responses (i.e.,their norms) are identical.

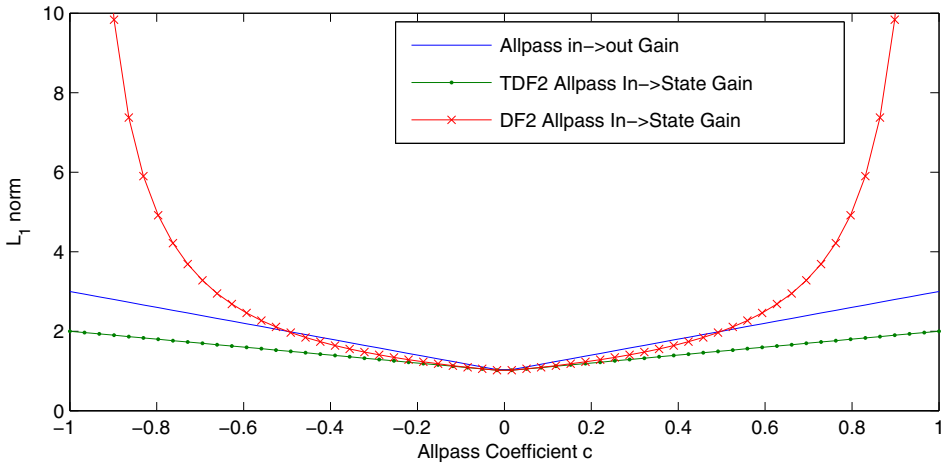


Figure E.3: Worst-case gains vs. allpass coefficient, first-order DF2 and TDF2 forms.

E.3 TDF2 Lattice Form

The result of the previous section has application in the field of Lattice Filters,[2] as it turns out that the most common lattice form is very closely related to the DF2 IIR form. In particular, a terminating stage is exactly a DF2 first-order allpass filter (see Figure E.4).

Note that the middle filter in Figure E.5 is no longer in direct form; rather, it is a different form made up of direct-form parts.

Given that the lattice stages are DF2, there is a chance of a very large gain to the delay when the k_i get "close" to 1.0.[3] In fact, the gain calculated in the previous section is the gain from the system input to the input of the delay, which in a lattice, is the input to the next stage. Thus, this gain is presented directly to the next stage, and its gain will be compounded upon the first stage's internal gain, and so forth for all the stages. Internal overflow can thus be expected to be a significant issue in fixed-point lattice-filter design using the standard lattice form, and it is indeed known to be [152].

By analogy to the previous section, we can imagine a lattice section based on the TDF2 allpass. Such a form is presented in Figure E.6. The most obvious change in the Lattice form is that the feedforward and feedback are "flipped" to the other side of the sums.

Lattice filters implemented in this form would be significantly more resistant to internal overflow than the standard DF2 form, as discussed previously. Note that this form was also derived by Jean Laroche [152], though apparently not by analogy to the direct forms. When used to make

[2]This section derives in part from discussions of reverb implementation with Sean Costello.
[3]How close is considered a problem depends on the situation and the requirements.

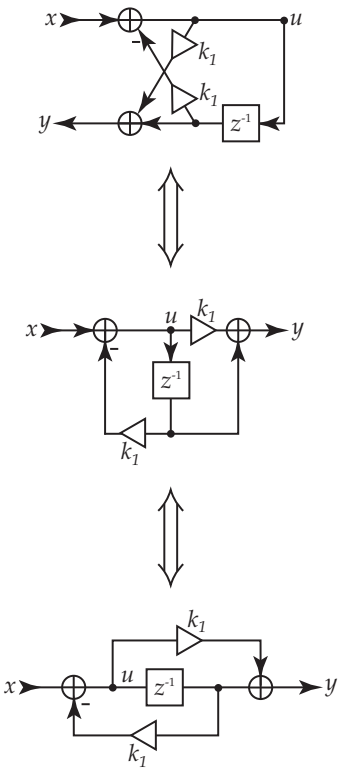


Figure E.4: These three forms are exactly equivalent. Top: The standard IIR lattice section, Middle: DF2 IIR allpass filter, Bottom: Drawn in a common allpass style.

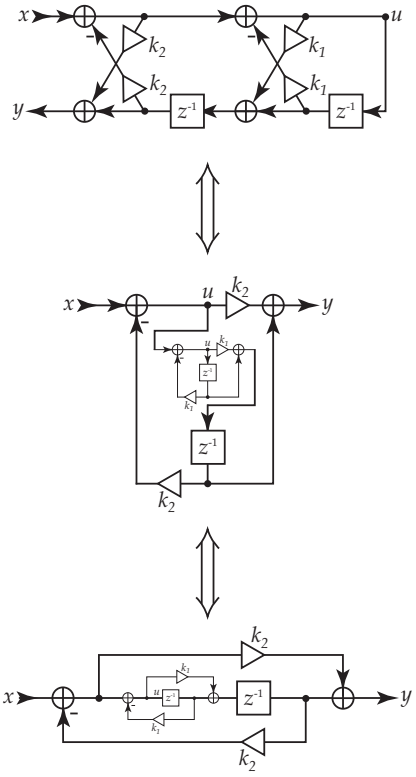


Figure E.5: Second-order Lattice, drawn in the same forms. Note where "inner" sections are placed in the IIR and allpass forms.

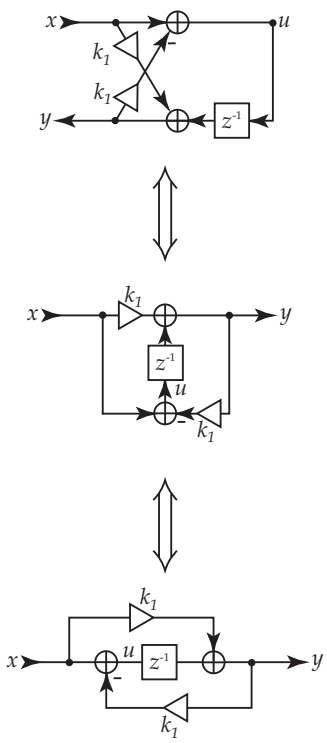


Figure E.6: These three forms are exactly equivalent. Top: TDF2 IIR lattice section, Middle: TDF2 IIR allpass filter, Bottom: Drawn in a common allpass style.

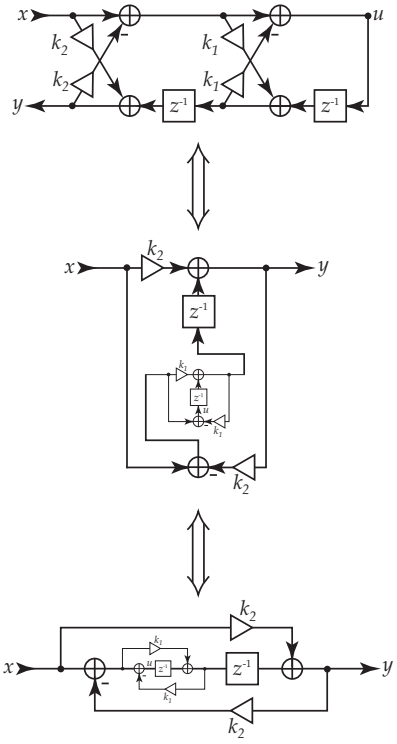


Figure E.7: Second-order Lattice, drawn in the TDF2 forms.

higher-order allpass filters (i.e.,if the only output used is the left-hand output), then the TDF2 lattice acts equivalently to the DF2 form, but with reduced internal signal levels. However, if used as a lattice filter with taps taken from the internal stages, then the fact that the TDF2 has a lower internal gain will cause all the internal taps to be scaled relative to the DF2 versions. Laroche showed that the internal states are scaled relative to the DF2 forms by $\prod(1 - k_i^2)$, for all the stages to the left of (and including) the stage in question. This can also be seen by comparing the gains of Equations E.6 and E.9.

Appendix F

Fitting Parametric EQs

This appendix describes research the author did as part of a project at Staccato Systems in 1999, and which was independently derived by Jonathan Abel and Dave Berners [1].

F.1 Problem Statement

> *Problem:* Find the parameters for a bank of parametric EQ sections in order to fit the total magnitude response to a desired shape.

This problem arises in many situations. One case has been in the design of filters to flatten speaker-enclosure frequency responses. These problems have been tackled extensively with algorithms which automatically and/or adaptively design correction filters ([182], [138]). Such filters tend to be "black boxes" from the user's standpoint, and there may be cases where user would like to tweak the design beyond that which the algorithm designed. The use of parametric EQs as the basic sections gives a final design that the user can understand and easily modify as desired.

A sub-problem, which we will solve on the way to tackling the first problem is as such:

> *Sub-Problem:* Find the gains for a bank of fixed EQ sections (i.e., a Graphic EQ) in order to fit the total magnitude to a desired shape. In particular, the curve represented by the knobs of a Graphic EQ.

This problem arises due to the fact that in order to create Graphic EQs to have the ability to smoothly shape a range of the frequencies (i.e., not have "holes" in the response whereby the EQ would have no (or limited) effect), the sections must be designed to have overlapping frequency ranges (see Figure F.1). As such, when adjacent sections are set to boost or cut all in the same direction, their responses will add up to a stronger effect than expected (Figure F.2). In fact, if the designer wants to reduce wiggles in the total response, they must make the sections have wider

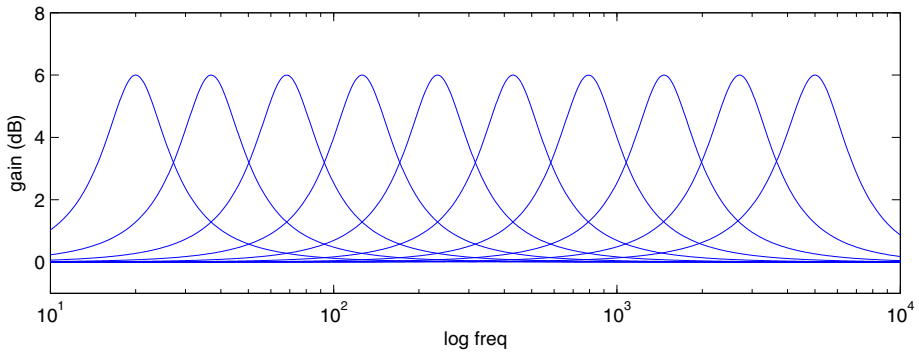


Figure F.1: Stage frequency-response overlap in typical Graphic EQ

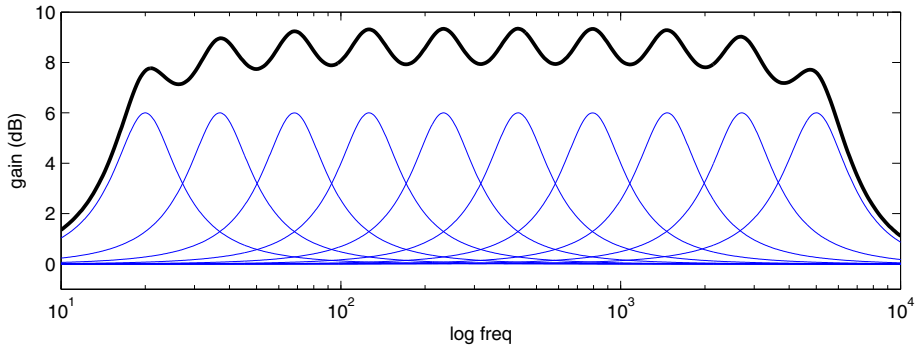


Figure F.2: Total response, showing the over-gain effect of overlap.

bandwidths However, that causes the effect of combined response to get even worse. (Figure F.3). Therefore, the designer historically must make a tradeoff decision between this effect and ripple. One can argue that most users simply adjust a Graphic EQ by ear anyway, so the actual shape of the curve represented by the knobs doesn't matter. That is probably true in most cases (as long as the Qs aren't too extreme in either direction), but it turns out that this problem can be resolved so that no such arguments are necessary.

The idea is to interpret the knob values as samples of a desired response, and put some computation between the knobs and the EQ sections which computes the actual stage gains to make the total response have the same shape as the knobs (or even pass through those gains). With such computation in place, tweaking by ear is still a valid way to use the EQ, but the shape of the curve "displayed" on the front sliders would be "more correct".

This sub-problem is simpler than the main problem. This is because the sub-problem is restricted by the EQ sections having fixed center frequency and bandwidth, and so the problem is

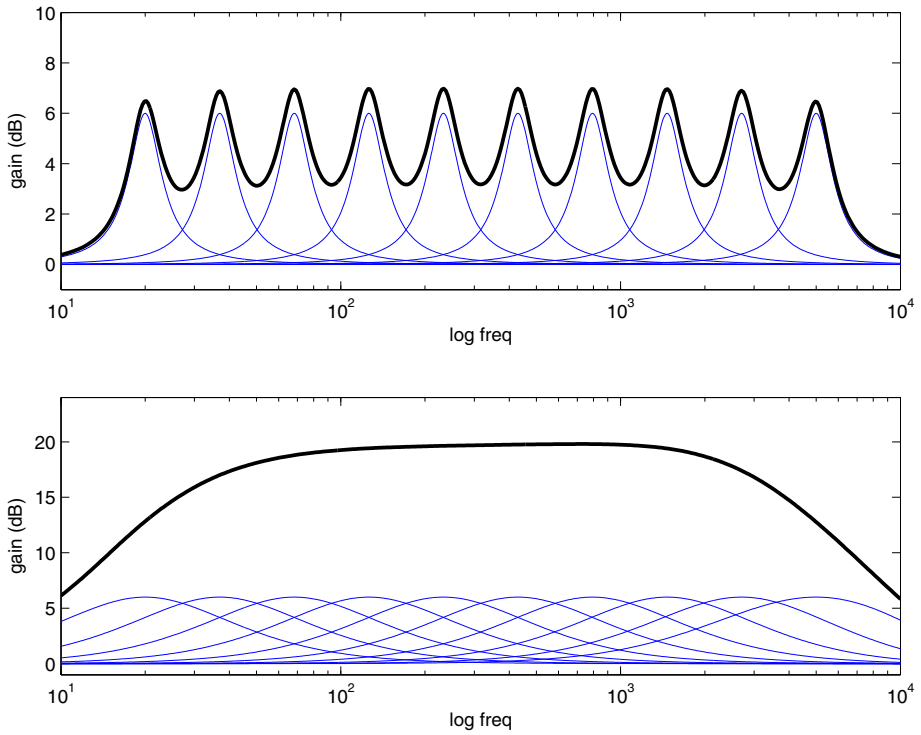


Figure F.3: Tradeoff in stage Q between "wiggle" and how far the overlap extends the total gain.

simply a problem of fitting the gains, (which ends up having a straightforward least-squares solution in certain circumstances) whereas in the main problem, the center frequencies and bandwidths are also optimization parameters and present therefore a much larger optimization space (i.e. more possible choices) and a more difficult design.[1] It will turn out, in fact, that solution of the subproblem is used as a step in the proposed solutions to the main problem.

F.1.1 Assumptions

We will take as the filter architecture for our system a set of N digital 2nd-order filter sections in series, each implementing a peaking EQ stage [28], with one or two stages optionally implementing first-order shelf filters.

This discussion will only use these shapes, but we note that adding other stages with other shapes to this technique is straightforward, *as long as the shapes meet a certain requirement*. The main concepts of this method apply to any filter shape which meets this requirement.

This discussion will use peaking EQ shapes which have gains of 0 dB at both DC and $f_s/2$.

[1] Solutions to the sub-problem, however, should not be expected to produce as close fits as good solutions to the main problem.

There are well-known techniques [198] for creating discrete-time EQ sections which are not required to have a gain of 0 dB at $f_s/2$, approximating the shapes of continuous-time peaking EQ sections whose skirts cross past $f_s/2$, and Abel and Berners use such shapes in their discussion [1]. As noted above, these differences can be considered to be implementation details relative to the algorithm itself.

F.2 Derivation

The response of each EQ stage will be called $H_i(z, g_i, f_{c_i}, Q_i)$. g_i is the peaking gain, f_{c_i} is the center frequency, and Q_i is the Q of the peak. When it is clear by the context, we will often refer to the response as just a function of z: $H_i(z)$, or just certain controls: $H_i(z, g_i)$. The total response of the system will be called $H_{tot}(z)$. Sometimes we will refer to the whole set of stage controls: $\mathbf{g} = \{g_1, g_2, \dots g_N\}$, $\mathbf{f_c} = \{f_{c_1}, f_{c_2}, \dots f_{c_N}\}$, $\mathbf{Q} = \{Q_1, Q_2, \dots Q_N\}$. As such, we may sometimes note the total response for a particular set of controls as $H_{tot}(z, \mathbf{g}, \mathbf{f_c}, \mathbf{Q})$, or just $H_{tot}(z, \mathbf{g})$ in Graphic EQ case, where the stages have fixed f_c and Q, which can be assumed to be known. The desired shape will be $H_{des}(z)$, and will assumed to only be defined at a set of frequencies $z_i \in z_{des} = e^{j\omega_i}, \omega_i \in \omega_{des}$. The number of EQ stages is N. The number of points in H_{des} is M. $\omega = 2\pi f/f_s$ is a normalized frequency in radians (i.e., an angle along the unit circle).

F.2.1 The Gain-Shape Requirement

The total response of the system is:

$$H_{tot}(z) = \prod_{i=1}^{N} H_i(z) \tag{F.1}$$

where the $H_i(z)$ are the transfer functions for each stage (as yet unspecified). Converting the amplitudes to decibels, this becomes:

$$20 \log_{10} |H_{tot}(z)|_{\text{dB}} = \sum_{i=1}^{N} 20 \log_{10} |H_i(z)|_{\text{dB}} \tag{F.2}$$

For simplicity in the representation, let us define the amplitude of a transfer function $H(z)$, in decibels, as $G(z)$:

$$G(z) = 20 \log_{10} |H(z)| \tag{F.3}$$

So that:

$$G_{tot}(z) = \sum_{i=1}^{N} G_i(z) \tag{F.4}$$

Thus we see that G_{tot} is a linear combination of the stage responses, in log space (with the combination scales being unity).

Let us look at graphic EQ example (i.e., the subproblem), where the stage center frequencies and Qs are fixed. As such, the problem of fitting the total response to a desired shape is one of choosing the correct stage gains $\mathbf{g}$. The particular problem to be solved is chosen to be:

$$\min_{\mathbf{g}} \|G_{tot}(z_{des}, \mathbf{g}) - G_{des}(z_{des})\| \tag{F.5}$$

for some distance measure (L_1, L_2, L_∞, etc.).

Now, as we noted above, G_{tot} is a linear combination of responses. We know that the pseudo-inverse solution $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$ of the over-determined system $\mathbf{Ax} = \mathbf{b}$ gives an $\mathbf{Ax}$, a linear combination of the rows of $\mathbf{A}$, which fits $\mathbf{b}$ in a least-squares sense [254]. As such, we know that there a least-squares fit to G_{des} using a linear combination of the G_i:

$$\exists\{x_1, x_2, \ldots x_N\} \; s.t. \left\| G_{des}(z_{des}) - \sum_i x_i G_i(z_{des}) \right\|_2 \text{ is minimized.} \tag{F.6}$$

for any particular $\mathbf{g}$, $\mathbf{f_c}$, and $\mathbf{Q}$.

However, the above does not have a straightforward interpretation in terms of designing filters, as $xG(z)$ corresponds to $|H(z)|^x$, which is not a scaling that can easily be done on a filter in general, especially if x is not an integer (as it surely will not be).

One may ask: "Yes, $xG(z)$ may not have a straightforward interpretation, but $xH(z)$ does. Why not just fit in linear space?"

First, fitting in linear space implies a parallel architecture for the 2nd-order sections rather than a series architecture. A parallel combination of filters will have zeros in locations that are not necessarily intuitive, and hence the output of a design algorithm using a parallel architecture may not be as "user tweakable" as a series architecture. For example, in a series EQ, one can attenuate a frequency range by adding a stage that has a cut in that range, but in a parallel EQ, one must attenuate by adding a stage that "subtracts" in that region (hence interacting with the combined phase of the other stages, and probably needing to be re-tweaked if another stage is added or removed). Second, the generally-understood shape of an EQ stage is understood in the log-log domain, not the linear domain. Again, this means that the design is more tweakable in the log-log domain, hence $G(z)$ is a likely choice for the working representation.

Now although $xG(z)$ is not easy to implement directly, let's look at its effect on the response of an peak or shelf filter. Remember: an ideal peaking filter has a response of "nearly" 0 dB for all frequencies but a range around f_c. Similarly, a high- or low-shelf filter has a response of nearly 0 dB for all frequencies below or above some frequency, respectively. As such, scaling of the response in dB space will not affect the 0 dB regions. Further, the basic shape will be a scaling of the original

shape. Together, these imply that the resulting shape is of the same type as the original shape: a

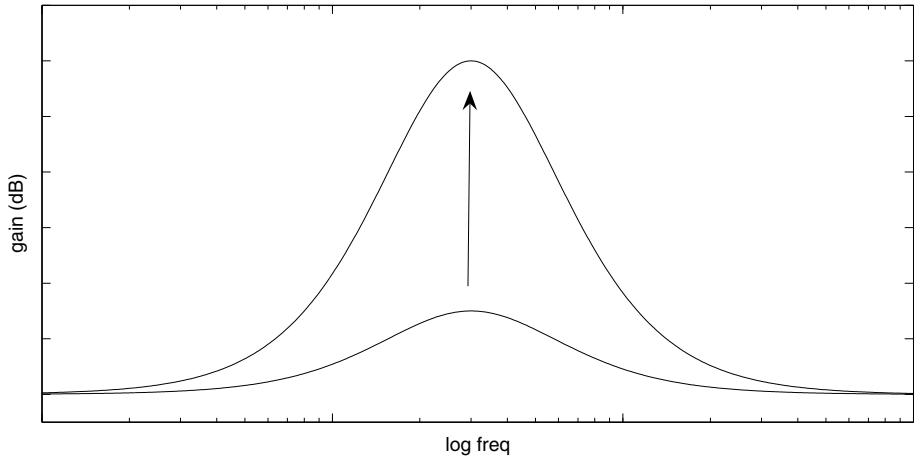


Figure F.4: Scaling an EQ section up, resulting in what looks like another EQ section (maybe different Q).

scaled peak EQ still looks like a peak EQ (See Figure F.4), maybe of another formulation and of different parameters, but still a peak-EQ shape. Similarly, a scaled shelf still looks like another shelf. Hence, one may expect that

$$xG(z, g, f_c, Q) = \bar{G}(z, \bar{g}, \bar{f}_c, \bar{Q}) \tag{F.7}$$

Where $\bar{G}$ is some other EQ filter, maybe with some other formulation and other parameters. One may thus theorize that although $xG(z)$ does not have a simple implementation in general, there might exist some EQ-filter formulation for which it *does*. Further, the peak gain of $xG(z)$ will be the peak gain of $G(z)$ scaled by x, and the peak frequency will not be changed, since the scaling would not change the shape of the response curve.[2] So, we expect that

$$xG(z, g, f_c, Q) = \bar{G}(z, xg, f_c, \bar{Q}) \tag{F.8}$$

Now, since the intent is to use the g_i as the fitting parameters, and in order to not have to deal with changing filter types and Q's in the course of the optimization, we decide to fix Q and the EQ formulation, then ask if an EQ formulation can be found for which the following is true:

$$xG(z, g) = G(z, xg) \tag{F.9}$$

[2]The peak will not change in a peak filter. However, it might shift in a shelf filter, depending on how corner frequency is defined

Let's name the above "The Gain-Shape Requirement". In words, the requirement is that the shape of the EQ peak does not change with gain: for all gain parameters, the response for a scaled version of the gain is the same as scaling the response for that gain.

The result of this requirement is that if we have a total system made up of filters that satisfy the requirement, we can exactly solve (in one step) for the gains required to get a least-squares fit to a desired response by measuring the individual filter responses (at any known nonzero gains), and use them as basis vectors for a (pseudo)inverse fitting algorithm, the result of which scales the original filter gains to get the final stage gains. We will demonstrate this in action after we look for a peak EQ formulation that satisfies the Gain-Shape Requirement.

F.2.2 Looking for an EQ formulation

There are a number of different EQ formulations. A good review is given by Robert Bristow-Johnson [28], where he shows that all 2nd-order digital peaking-EQ formulations are equivalent in all aspects except their bandwidth definitions (and hence Q definitions). The basic idea is that since a 2nd-order digital filter (a 'biquad') has five coefficients, there can be no more or less than five design constraints. Four of the constraints are the same for all formulations: (1) the response has a peak (i.e., it is horizontal) at the center frequency; (2) the response has a given gain at the center frequency; (3) gain at DC is 0dB; and (4) gain at $f_s/2$ is 0 dB.[3] The remaining constraint is used to design the width of the peak, and this is where various techniques differ: how do you define which gain levels to measure the bandwidth of the filter? Design methods have defined bandwidth variously at: (1) 3dB down from max gain (either the peak or 0dB, whichever is higher), (2) 3dB towards 0dB from the peak (either boost or cut), (3) halfway (in dB) between the peak gain and 0dB, (4) various combinations of the above using 3dB, 6dB, and halfway, etc. The paper compares which definitions which formulations have used, and derives a generic formulation which can be used to design peaking EQs of any desired bandwidth definition.

In essence: a peak EQ formulation (usually) implicitly defines its bandwidth to be measured at a particular gain: the gain at which the peak is the specified width (the 'bandwidth'). The formulations differ in their definition of what that gain is (see Figure F.5).[4]

Since the peak is a shape whose width increases monotonically towards 0dB, then formulations with bandwidth definitions defined at gains closer to 0dB will give "narrower" peaks when compared (at the same "bandwidth") to ones with bandwidth defined closer to the peak gain (Figure F.5).

Now, it turns out that most of the formulations do not at all preserve the peak shape as we

[3] The paper did not include design methods which allow gain other than 0 dB at $f_s/2$ (that was not described until a later paper by Orfanidis [198], but the concept that the principle differentiator is bandwidth definition still applies.

[4] Some formulations may even define separate gains for the upper and lower bandedges, particularly in situations where a filter is being designed which is not symmetric about f_c (such as a highpass or lowpass filter). A good example is ([57], pp.674-675)

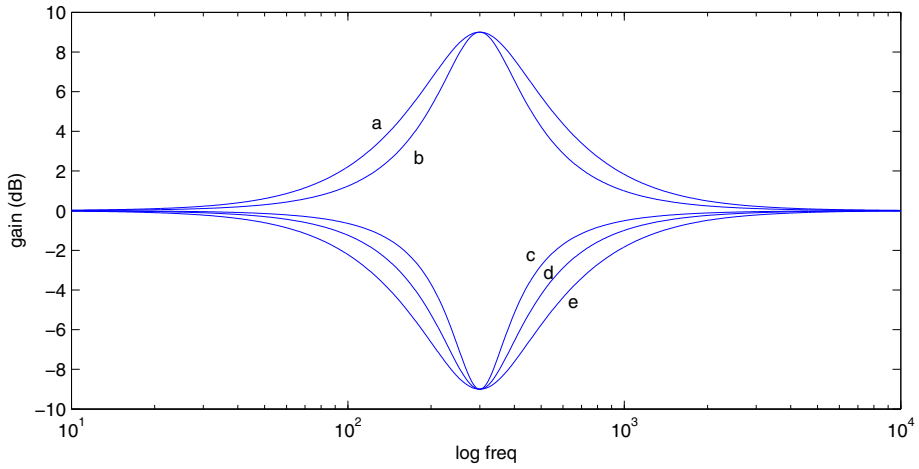


Figure F.5: Same Q, different bandwidth definitions: (a) 3dB down from peak, (b,d) halfway (in dB) to 0dB, (c) 3dB down from max gain (0 dB in this case), (e) 3dB towards 0dB from peak ((a) also fits this definition)

desire. For example, note how the shape of the peak changes drastically with a "3-dB from the peak" definition for bandwidth (Figure F.6).

Looking back at the Gain-Shape requirement, we can note that a good idealization of the situation would be if the peak EQ shape could simply be a general scaling and stretching of a single prototype shape, in which case as long as the horizontal scaling worked out, then the requirement would be guaranteed to be satisfied, as there would be only one shape (only the wrong horizontal scaling could break the requirement). Let's say this shape is defined in log-freq/dB space by some symmetric peak-shaped function

$$y(\omega), y(0) = 1, y(\omega) = y(-\omega), \text{monotonically decreasing away from } \omega = 0 \tag{F.10}$$

Hence

$$G_{ideal}(\omega) = g_{ideal} y\left(\frac{\omega - f_{c_{ideal}}}{b}\right) \tag{F.11}$$

Where b is some stretch factor based on Q_{ideal} and our bandwidth definition. We can simplify the following without loss of generality by letting $f_{c_{ideal}} = 0$, so that

$$G_{ideal}(\omega) = g_{ideal} y\left(\frac{\omega}{b}\right) \tag{F.12}$$

Let's now see how b relates to bandwidth in a 3dB formulation: Let $Q = \frac{f_c}{\text{bw}} = \frac{f_c}{2\omega_0}$, where $\pm\omega_0$ are the frequencies where the curve passes through the required gain. Now, if the required gain is

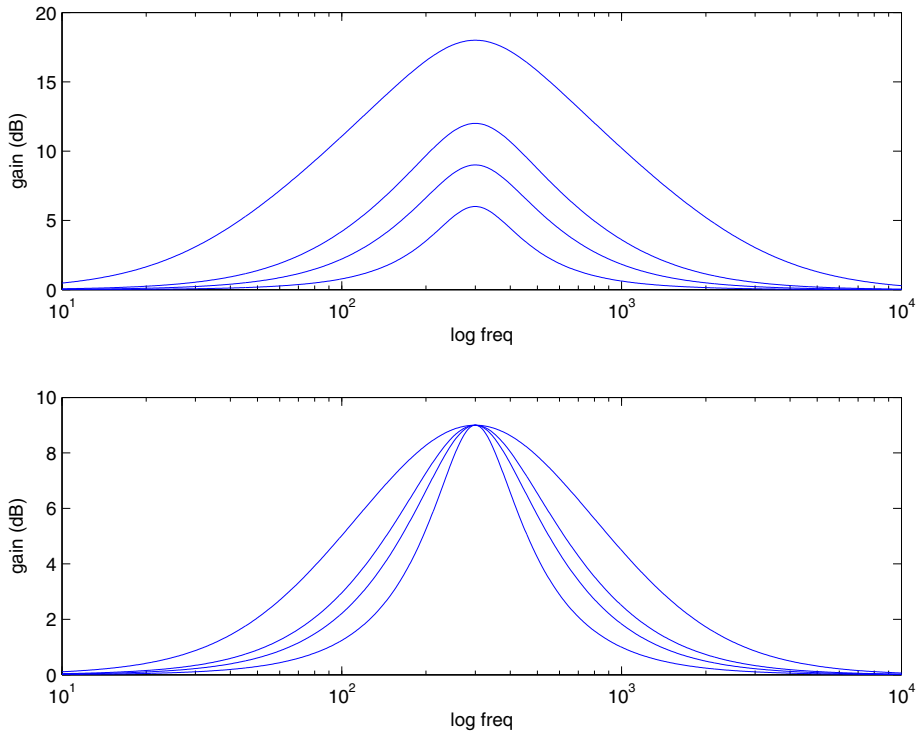


Figure F.6: Upper: Peaking EQ sections using the 3dB bandwidth definition, all have Q of 1, peak gains: 6dB, 9dB, 12dB, 18dB. Lower: All of the above responses scaled to have their peaks on 9dB.

(peak-3dB), then we solve $gy(\omega_0/b) = g - 3$ for b, and get:

$$b = \frac{\omega_0}{y^{-1}\left(\frac{g-3}{g}\right)} \tag{F.13}$$

Thus the horizontal scaling factor b is dependent on g, meaning that the shapes for two different gains, but the *same* ω_0 (i.e., the same Q), will have two different horizontal stretch factors, and hence not be appropriate for the Gain-Shape requirement.

On the other hand, if we use a formulation where the bandwidth is defined at some fraction a of the peak gain (in dB space), we get the following:

$$gy(\omega_0/b) = ag \Rightarrow b = \frac{\omega_0}{y^{-1}(a)} \tag{F.14}$$

So that b is not related to the gain, only the fraction a! Thus the stretch factor will be the same for a given Q, regardless of g, which fits the Gain-Shape requirement.

Now, in terms of bandwidth definitions, the only fraction-based definition in common use is

$a = 1/2$. We will call this definition the "Halfway Definition" of bandwidth (where it is assumed to be understood that this is defined in dB space).[5]

The above was discussed using a hypothetical ideal filter shape, but the result still applies to real-life shapes: bandwidth definitions which are defined at gains which are not fixed fractions of the peak gain will tend to have widths which depend on the peak gain, as can be seen in the experimental results (Figure F.6).

This leaves the Halfway Definition as the leading contender, as we have just shown that it that it would be a workable bandwidth definition for an ideal peaking shape, and all other standard bandwidth definitions are not based on a fraction of the peak gain.[6] Thus we ask: How closely does a peak EQ using the halfway definition match the Gain-Scaling Requirement?

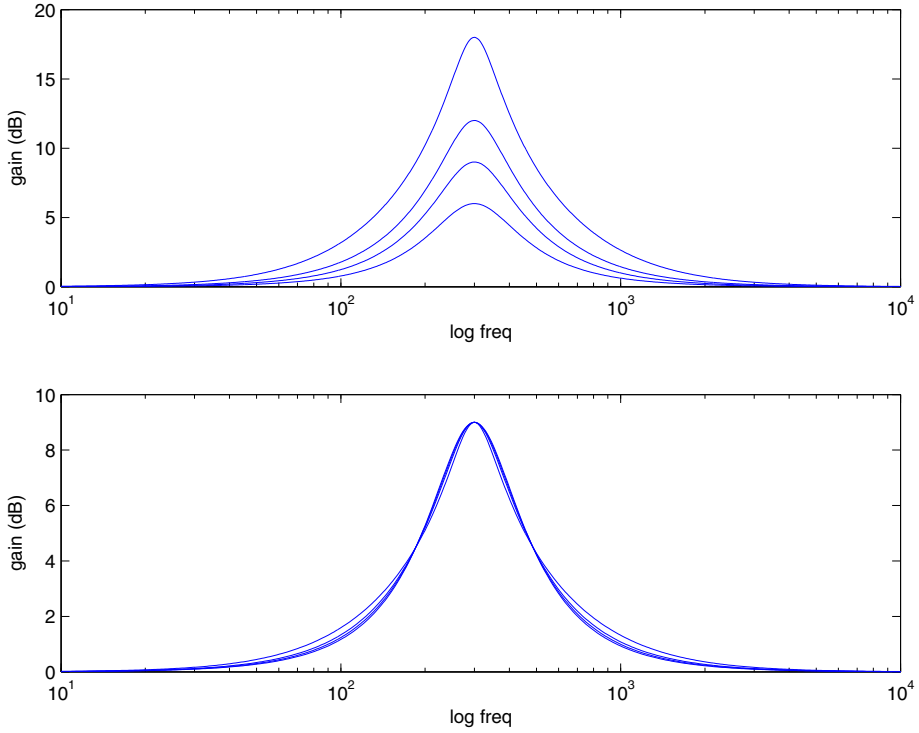


Figure F.7: Upper: Peaking EQ sections using the halfway bandwidth definition, all have Q of 1, peak gains: 6dB, 9dB, 12dB, 18dB. Lower: All of the above responses scaled to have their peaks on 9dB.

The answer: "Close, but not exact" (Figure F.7). Note that the shape stays much more consistent

[5] Further research might explore the effect of a on the fitting, using actual non-ideal formulations, like Equation F.15. There is a possibility that other values might fit better for certain values of Q, etc.

[6] [28] also comes to a similar conclusion about the usefulness of the halfway definition, and [198] further discusses the properties of bandwidths defined as arithmetic or geometric means.

than the 3dB definition, but as the difference between unscaled and scaled gain gets large, differences in the fine shape do become obvious, though they do have arguably the same widths (see Figure F.8, in particular, all the shapes intersect at their respective halfway gains). This difference

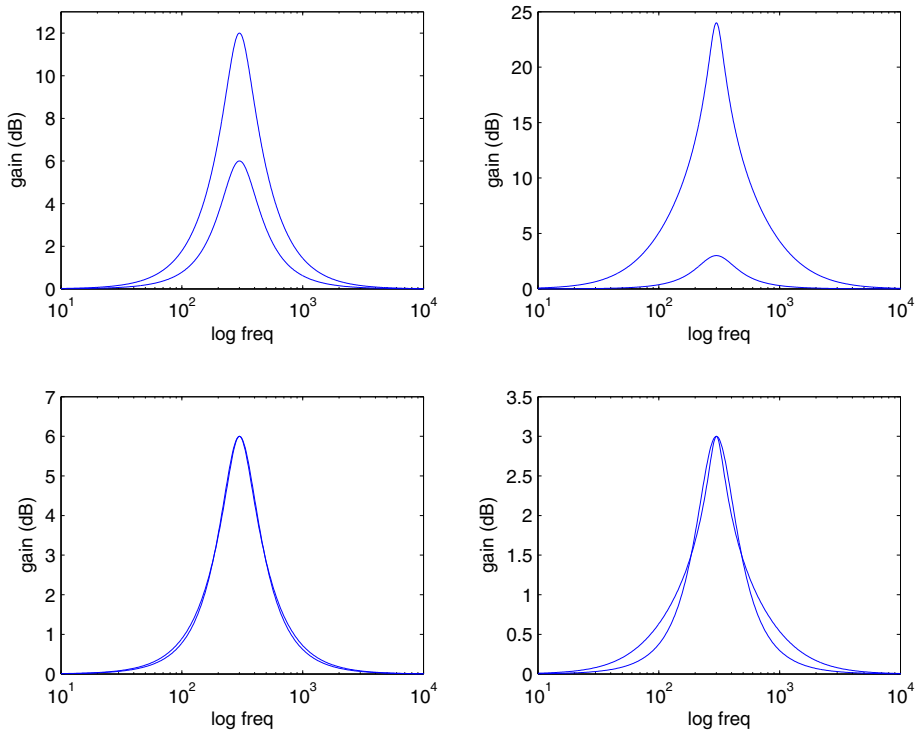


Figure F.8: Differences in Shape with the Halfway Definition, small scale difference compared to large scale difference: Left: 12dB scaled onto 6dB, Right: 24dB scaled onto 3dB

is considered sufficiently small for the purposes of the fitting algorithm, and there is a way to deal with this lack of exactness, which we will see in the next section. We therefore choose the Halfway Definition for our Peak EQ.

In the terminology of [28] (where F is the bandwidth-defining gain), our Peak EQ design equations are:

$$\begin{aligned} k_i &= 10^{g_i/20} \\ F_i &= 10^{(g_i/2)/20} \quad \text{(The Halfway Definition)} \\ \gamma_i &= = \sqrt{k_i}\sqrt{\frac{{F_i}^2 - 1}{{k_i}^2 - {F_i}^2}} \tan\left(\frac{\pi f_{c_i}}{Q_i f_s}\right) \\ n_{0i} &= (1 + \gamma_i\sqrt{k_i})/(1 + \gamma_i/\sqrt{k_i}) \end{aligned} \tag{F.15}$$

$$
\begin{aligned}
n_{1i} &= -2\cos(2\pi f_{c_i}/f_s)/(1+\gamma_i/\sqrt{k_i}) \\
n_{2i} &= (1-\gamma_i\sqrt{k_i})/(1+\gamma_i/\sqrt{k_i}) \\
d_{1i} &= n_{1i} \\
d_{2i} &= (1-\gamma_i/\sqrt{k_i})/(1+\gamma_i/\sqrt{k_i})
\end{aligned}
$$

$$
H_i(z) = \frac{n_{0i} + n_{1i}z^{-1} + n_{2i}z^{-2}}{1 + d_{1i}z^{-1} + d_{2i}z^{-2}}
$$

Remember, Abel and Berners instead use an EQ design based on [198] which doesn't have to go back to 0dB at $f_s/2$, but they still use the Halfway Definition for bandwidth, and it still keeps the EQ shape close to constant for changes in gain.

We also note that the halfway definition also works for shelving filters. By analogy to the above, we can derive formulations for first-order shelf filters using the Halfway Definition:

$$
\begin{aligned}
T &= 1/f_s \\
g_{dc} &= \text{gain at DC in dB} \\
g_{ny} &= \text{gain at } f_s/2 \text{ in dB} \\
k_{dc} &= 10^{g_{dc}/20} \\
k_{ny} &= 10^{g_{ny}/20} \\
F &= 10^{(g_{dc}+g_{ny})/40} = \sqrt{k_{dc}k_{ny}} \\
\omega_c &= 2f_s\tan(\pi f_c/f_s) \\
A &= (1/\omega_c)\sqrt{(F^2 - {k_{dc}}^2)/({k_{ny}}^2 - F^2)} = (1/\omega_c)\sqrt{k_{dc}/k_{ny}} \\
n_0 &= (Tk_{dc} + 2Ak_{ny})/(T+2A) \\
n_1 &= (Tk_{dc} - 2Ak_{ny})/(T+2A) \\
d_0 &= (T-2A)/(T+2A)
\end{aligned}
\tag{F.16}
$$

$$
H(z) = \frac{n_0 + n_1 z^{-1}}{1 + d_1 z^{-1}}
$$

F.3 Graphic EQ Algorithm

As noted above, we can fit to a desired response by finding the required linear combination of fixed-(f_c,Q) EQ stage shapes:

$$
\min_{x_1,x_2,\ldots x_N} \left\| G_{des}(z_{des}) - \sum_i x_i G_i(z_{des}) \right\|_2 \tag{F.17}
$$

By assuming that the Gain-Shape Requirement (Equation F.9) holds, solutions to the above can be used to solve the actual design problem:

$$\min_{\mathbf{g}} \|G_{des}(z_{des}) - G_{tot}(z_{des}, \mathbf{g})\|_2 \tag{F.18}$$

To achieve this, we follow the following steps:

1. Initialize the stage gains, typically: $g_i = 1$, or $g_i = G_{des}(\text{nearest } \omega_{des})$
2. Measure the stage frequency responses $G_i(\omega_{des})$.[7]
3. Form matrix $\mathbf{A}$ from the G_i, each column being one of the responses[8]
4. Solve $\mathbf{A}\mathbf{x} = {G_{des}}^T$ for $\mathbf{x}$. Since the system is usually over-determined, this requires a pseudoinverse ($\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T{G_{des}}^T$), and will determine an $\mathbf{x}$ which fits $\mathbf{A}\mathbf{x}$ to G_{des} in a least-squared sense. In the Graphic EQ case, where G_{des} is the knob settings, then the system will not be over-determined and the final response should pass through the knob settings.
5. Since the Gain-Shape Requirement is assumed to be satisfied, adjust the stage gains as $g_{i_{new}} = x_i g_{i_{old}}$.
6. If the Gain-Shape Requirement is not exactly satisfied, as we have noted in the previous section, or for any further reasons, like coefficient quantization in the measurement of the stage responses, etc., then iterate from stage 2 as desired.

Since the difference between the pre- and post-scaled shapes with the Halfway Definition goes away as the difference in the scales gets small, even the second iteration will calculate a very accurate $\mathbf{A}$ matrix, and the iterations will converge very quickly (just a few iterations). In fact, if the difference is known to be "small enough" for the first iteration (below 6 to 12 dB, depending on how finicky the requirements are), then only one iteration can be considered sufficient. Note that in the actual Graphic-EQ example, one could use the Graphic-EQ knob settings as the initial stage gains (rather than unity) to initialize the gains in step 1, which probably helps make the first iteration even more accurate.

However, if features such as coefficient quantization are also put into the response measurement, then the iteration may take longer to converge (or may not actually end up fully converging, as the quantization causes the responses to be discontinuous in $\mathbf{g}$). Also, due to limits on the capability of a digital 2nd-order filter, the shape of an EQ stage gets grossly distorted as the stage

[7] In order for the problem to be feasible with the chosen EQ equations (Equation F.15), the design should not place a nonzero G_i directly on $fs/2$ or DC. However, if shelving stages are included in the design, this restriction is not necessary.

[8] An important note: if a stage gain goes to zero dB, then its response is no longer a valid basis function and step 4 will have problems. Therefore, it is typical to replace a $g_i = 0$ with $g_i = 1$ at stage 2. If the actual gain should be zero, the pseudoinverse should calculate $x_i = 0$ for this case, and the gain will end up at zero after step 5.

gain gets larger than ±40 to 50 dB. As such, fits which require extremely large stage gains may not converge, as the stage shapes no longer satisfy the Gain-Shape requirement at those gains.

Note that the sampling points ω_{des} should be evenly distributed in log-frequency space in order for the pseudoinverse fits to be well weighted. If desired, explicit weighting can be added to step 4 in standard ways, and Abel and Berners derive the algorithm using weighting matrices.

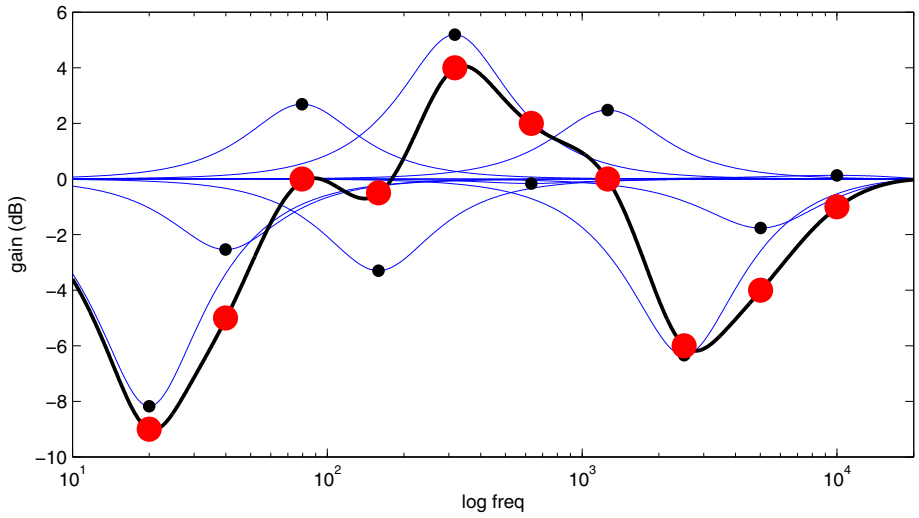


Figure F.9: A Graphic-EQ design

Figure F.9 shows a Graphic EQ design. The large dots represent the front-panel knob locations, whereas the small dots represent the internal stage gains g_i. The heavy line is G_{tot}, and the thin lines are the individual G_i.

Figure F.10 shows a design where several of the desired gains all boost together, which, as seen in Figure F.2, would produce too much gain in a basic graphic EQ. Here we see that the fitting algorithm has pulled back the central gains so that the total gain has the desired shape.

One might ask "If we have to iterate anyway, could we relax back to using an EQ formulation which does not satisfy the Gain-Shape Requirement as nicely as the Halfway Definition, such as a 3dB definition?" The answer, based on attempts to implement such an algorithm, is that the iteration does not easily converge if the initial condition is not very close to the final gains. There may be extensions to the algorithm to get it to work (or more likely: a formula to convert between halfway-definition Q and equivalent 3dB-definition Q, such that one could design with the halfway definition and convert the result to 3dB definition stages), but that is not explored here.

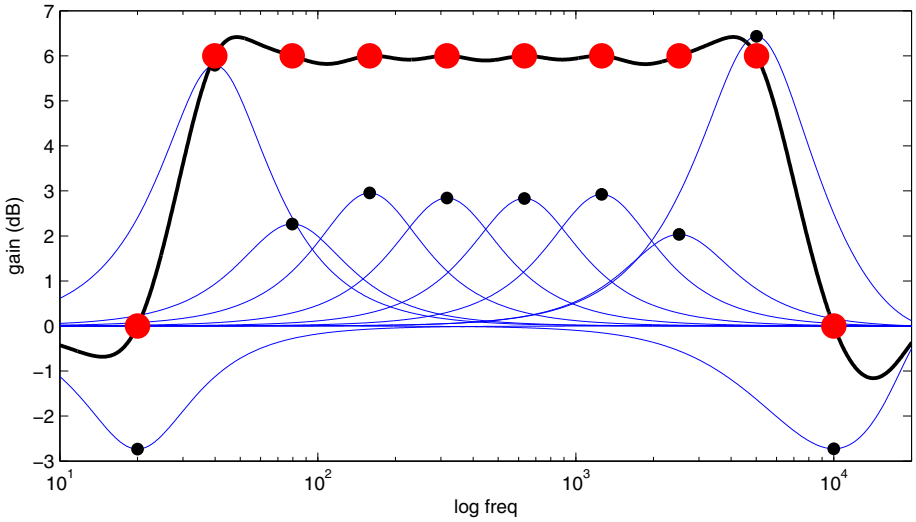


Figure F.10: A Graphic-EQ fit for a range of knobs all boosting together.

F.3.1 How do stage Qs affect the fit?

In the introduction, we noted that an EQ designer has to trade off a "wiggly" frequency response against the responses adding up badly (Figure F.3). The above algorithm can alleviate that problem in most cases, but there is still a tradeoff that must be understood: the **A** matrix becomes less well-conditioned as the stage Qs get wider. This translates into the individual stage gains getting larger, often well beyond the ranges of the final $G_{tot}(\omega)$ (See Figure F.11). On the other hand, if the Qs are too narrow, the total response "droops" too much between the stage centers (Figure F.12) Interestingly, fits with wide Qs often look very tight, it is just the extreme stage gains that hint towards problems. However, as noted above, if the stage gains start getting larger than 40dB or so, the filters lose their ability to keep the same shapes as the lower gains, and the iteration may no longer converge.

F.4 Parametric EQ Algorithm

Now we attack the original problem: fitting with the stage Qs and center frequencies also being optimization variables. The previous section derived an algorithm for fitting using fixed stages, now we let them vary in order to try to get a better design for any particular number of stages. However, this extended problem is non-convex, and so finding the global optimum would be quite difficult. Here, we will instead present heuristic methods for choosing these parameters, with the admission that they are not guaranteeing the best solutions, just 'good' ones.

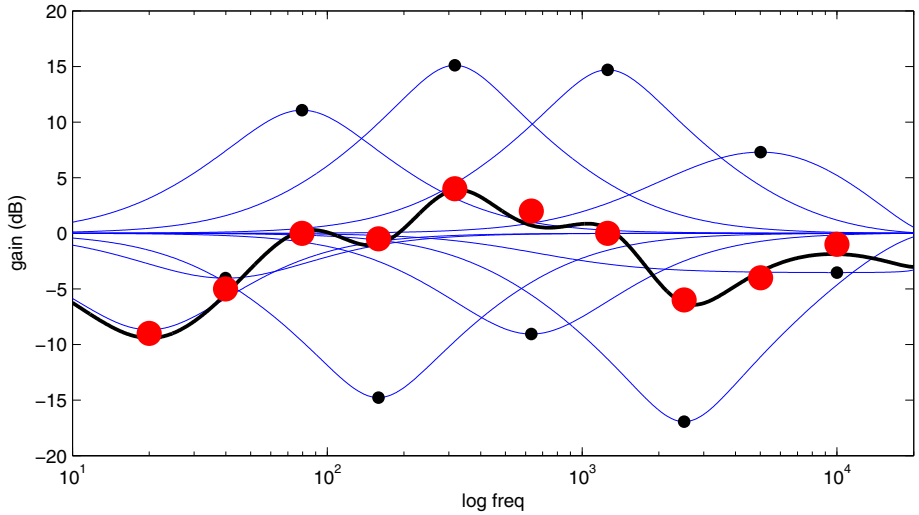


Figure F.11: A Graphic-EQ fit with stage Qs that are too wide. Note how the stage gains become much larger.

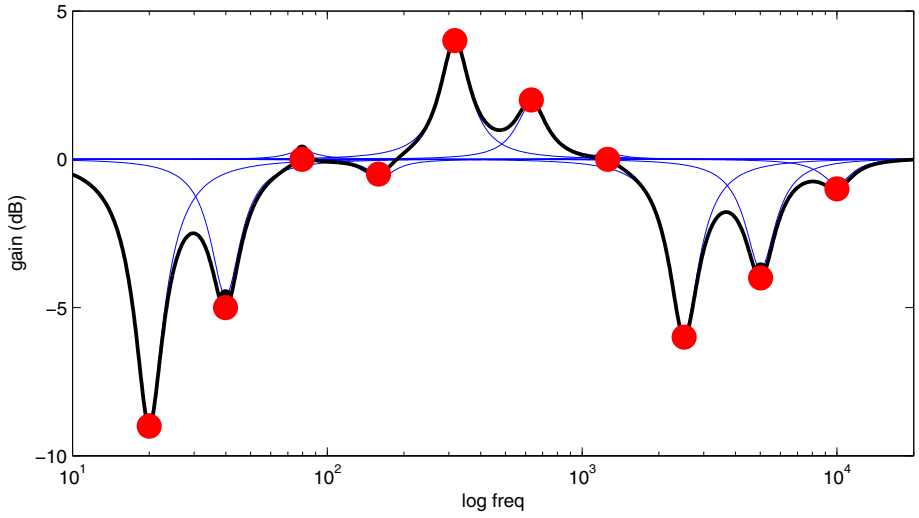


Figure F.12: A Graphic-EQ fit with stage Qs that are too narrow. Note how the total response droops between the stages.

At this stage, the author's derivation diverges from that of Abel and Berners. We will first look at their algorithm.

F.4.1 Selecting f_c and Q, Algorithm 1

1. Optionally perform a Bark smoothing [234] on the desired curve to perceptually reduce the complexity to just what is needed.
2. Locate all local extrema in the curve.
3. Place stages on each extremum, with center frequencies on the extrema frequencies. Bandwidths can be chosen either based on the halfway points between adjacent extrema, or based on the locations of inflections points in the curve. Place shelves at the end points, using a similar heuristic for the cutoff frequencies. Give each stage an initial gain (1dB, or the curve gain, anything but 0dB).
4. Perform one or more fits of the gains to the desired curve.

This method has a slight drawback in that the shape of the curve directly determines the number of required EQ stages. There is no built-in method for dealing with a situation where fewer stages must be used, more rules or heuristics would become necessary to handle that case.

F.4.2 Selecting f_c and Q, Algorithm 2

This algorithm is inspired by a verbal description once given for the Parks-McClellan [201] FIR filter design algorithm: "Find the largest error, increase the order and use the new degree of freedom to zero out the error at that point, and repeat with the new error." That description also effectively describes this algorithm:

1. Start with no stages, hence G_{tot} = 0dB. Or, if the EQ architecture has a total gain parameter, let $G_{tot} = g_{tot}$, and it can be set in a variety of ways:
   (a) Set it to some average or median of G_{des}, such that the starting G_{tot} = mean(G_{des}).
   (b) If the design does not have shelf filters and $G_{des}(fs/2)$ is well away from 0dB, the gain can be set there so that the high-frequency stages don't have to quickly jump back to 0dB at $f_s/2$ or DC.
   (c) Treat the total-gain parameter as a filter stage with a flat response, and add it into the fitting algorithm described above.
2. Choose a location for a new EQ stage
   (a) Compute $err(\omega_{des}) = G_{tot}(\omega_{des}) - G_{des}(\omega_{des})$.

(b) Identify the peak error location: $\omega_{pk} = \max_{\omega_{des}} |err|$

(c) Decide g and Q for a new stage centered on ω_{pk}. In this case, we use $g = 1dB$[9] and search locally around the peak in $err(\omega)$ for halfway points or reversals, and assign Q based on these frequencies.

3. Add an EQ stage with parameters decided in step 2.

4. Perform a fit of the gains of all existing stages to $G_{des}(\omega_{des})$ (the "Graphic EQ Algorithm" from the previous section)

5. repeat from Step 2 until out of stages (or until $|err(\omega_{des})|$ falls below some threshold if the number of stages is not fixed).

Figure F.13 shows the first three and iterations and the final result of this algorithm fitting a representative curve. Figure F.14 shows final fits for some other G_{des}.

Looking at Figure F.13, one may conclude that this algorithm fits much like Algorithm 1, as the stage centers are separated by inflection points. However, that does not always happen. As we can see in the top graph in Figure F.14, there are two cases where multiple inflections occur between stage centers (the pair just below 100 Hz, and the pair just below 1000 Hz). In this case, the wide stage up around 25 dB is “pulling everything up”, so that these narrower downward-pulling stages can work together, letting the “droop” between them catch a feature without requiring an extra stage. Note that this behavior isn't expressly designed into this algorithm as described, and it will not attempt to specifically find these situations. Instead, these behaviors may fall out of the fit depending on what order it chooses stages and how wide they are. As such, that indicates that this is probably not an optimal stage-choosing heuristic, and a better algorithm might be designed which does try to find and make use of such capabilities.

In the lower graph of Figure F.14, the wide upward stage is “off center”. This is probably due to how the algorithm defined a “peak” in the presence of a flat-top error. It is probably an implementation artifact, and other implementations might choose the first stage differently.

Step 2, and in particular Step 2c, are the most heuristic parts of this algorithm, and hence most open for experimentation for trying to enhance the algorithm. As described, this algorithm tries to knock out the peak error, and hence acts like an ∞-norm optimization. Variant algorithms could instead try to identify an EQ which minimizes the total mean-squared error, or covers up the "largest area region" in the error curve, thus attempting to first capture the overall shape of G_{des}, then heading in to the finer details. The algorithm as shown instead relies on the fitting in stage 4 to push the algorithm in a direction whereby the "large features" are eventually identified, but not necessarily at the start.

[9] The “correct” value of g will be determined in step 4

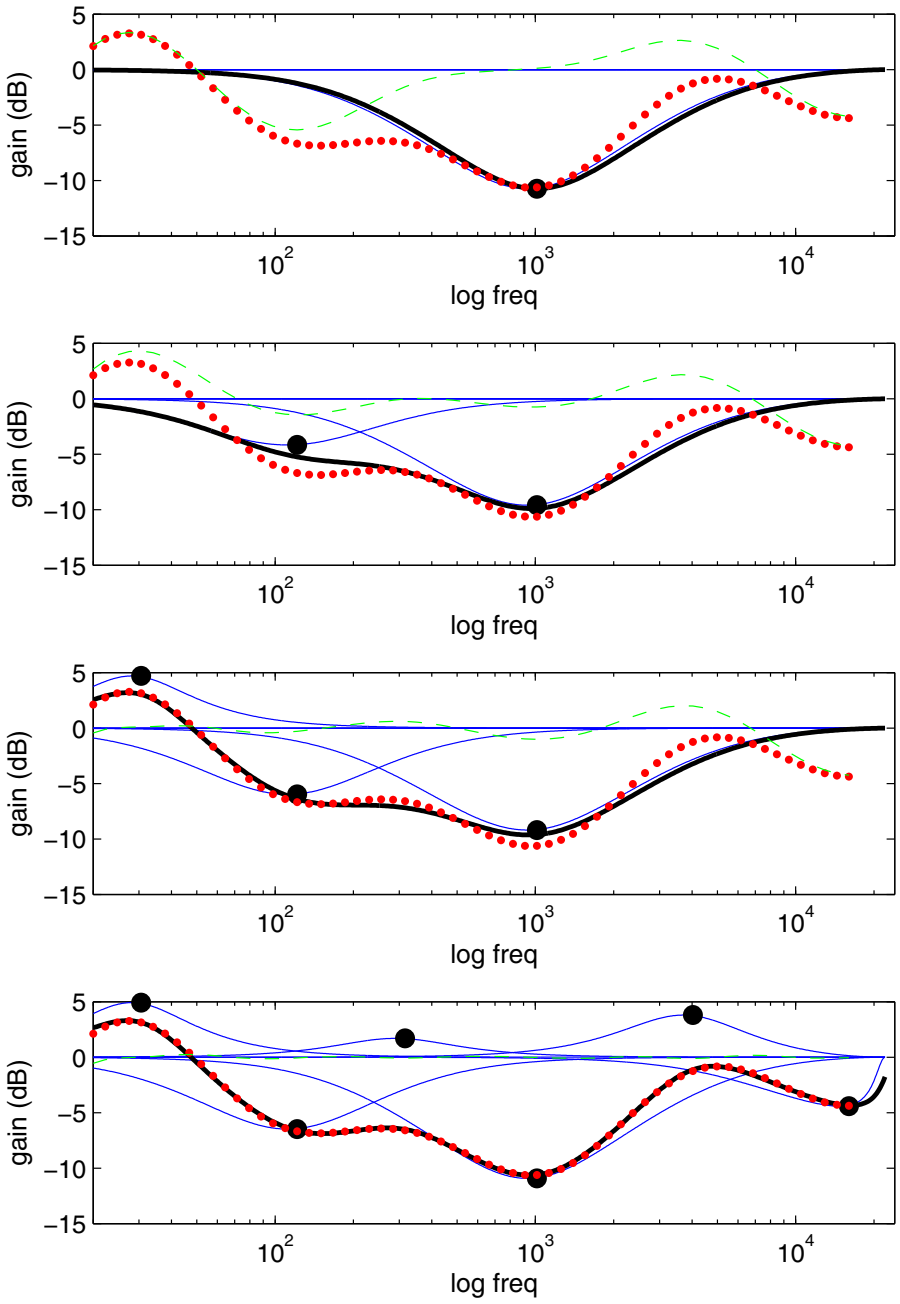


Figure F.13: The first three steps and final result for the Parametric EQ fitting algorithm 2 for a particular G_{des}. Dotted line: G_{des}, thick line: G_{tot}, dashed line $err(\omega)$, thin lines: G_i, dots: g_i. Six EQ stages.

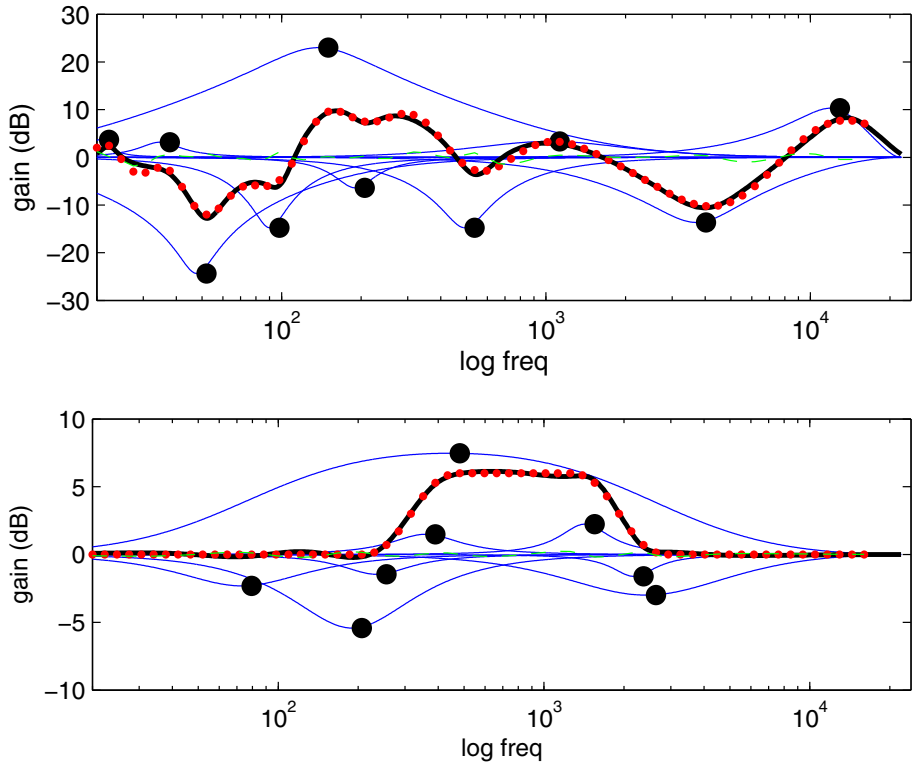


Figure F.14: Example fits for Algorithm 2

As described, Step 2 applies only to peaking-type EQ stages. Special-case identification of shelf stages would be added somewhere in the iterations (better early or better late is an open question) when those types of stages are available. Typically in such designs, there would only be one or two shelf stages, and all other stages would be peaking stages.

In cases such as "noisy" G_{des}, or other such difficult situations, the algorithm may sometimes attempt to put one stage on (exactly) the same center frequency as an existing stage. If not caught, this could end up with a lowered rank of the **A** matrix, which interferes badly with the working of the algorithm. As such, it is typical to add error-checking heuristics to the algorithm, such as: (1) not allowing two stages with same f_c, (2) adjustments for badly-calculated Qs if ω_{des} is sparse, etc.

F.5 Conclusions

This chapter has explored the problem of fitting standard EQ sections to desired curves, both in "graphical EQ" situations (i.e., all stages have predefined center frequencies and Qs), and in "parametric EQ" situations (where the frequencies and Qs can be designed as well). The "halfway in

dB" definition was shown to be the most useful for fitting, as it satisfied the "gain-shape requirement" that the shape of the EQ peak not change significantly with gain. With this definition, it turns out that the EQ shapes can be viewed as basis functions and a standard pseud-inverse can be used to perform least=squares fitting of the EQ gains to desired shapes. Two heuristics for choosing EQ-stage center frequencies and Qs were also presented.

We must note that Ramos and Lopez [213] described in 2005 an algorithm that is similar to the above algorithms in several ways. Like advocated in [1], it makes use of a psychoacoustic pre-processing to simplify the desired-gain data (though the pre-processing is different). Also, as in Algorithm 2 above, it attempts to place stages via an iterative "put it where it is most needed" method. It differs in the following ways:

- It optimizes the filters by first designing a "guess" filter using standard EQ design equations, then performs a local optimization directly on the filter coefficients (using a random-walk method) using the fit quality as the optimization measure.
- It appears to only optimize one EQ stage at a time, and leaves it alone once optimized and placed (though an optional re-optimization of all the filters at the end of the whole process is mentioned).

Appendix G

Gallery

A selection of the nicer images encountered through this research.

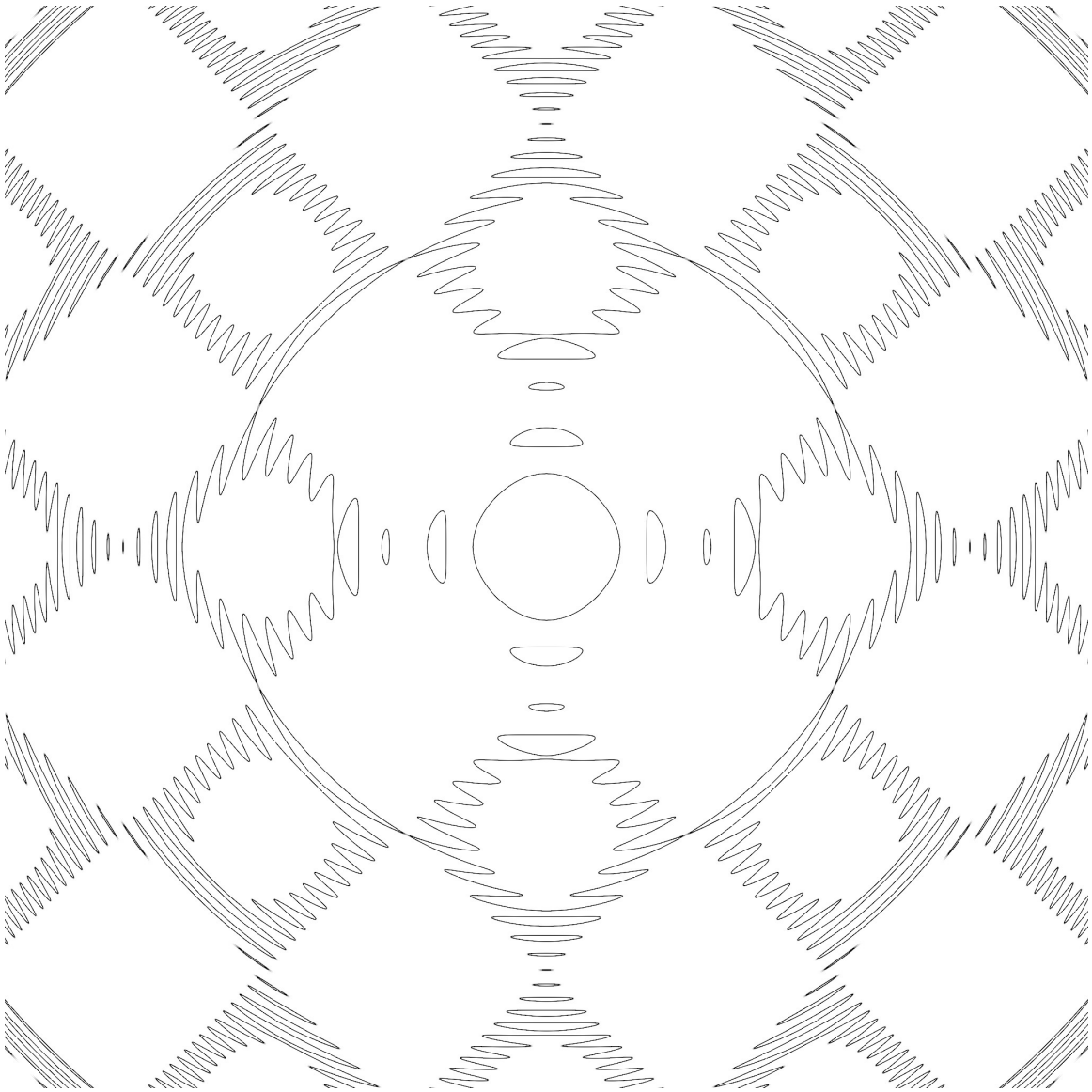


Figure G.1: Testing Taubin's Implicit-Function Rendering Algorithm

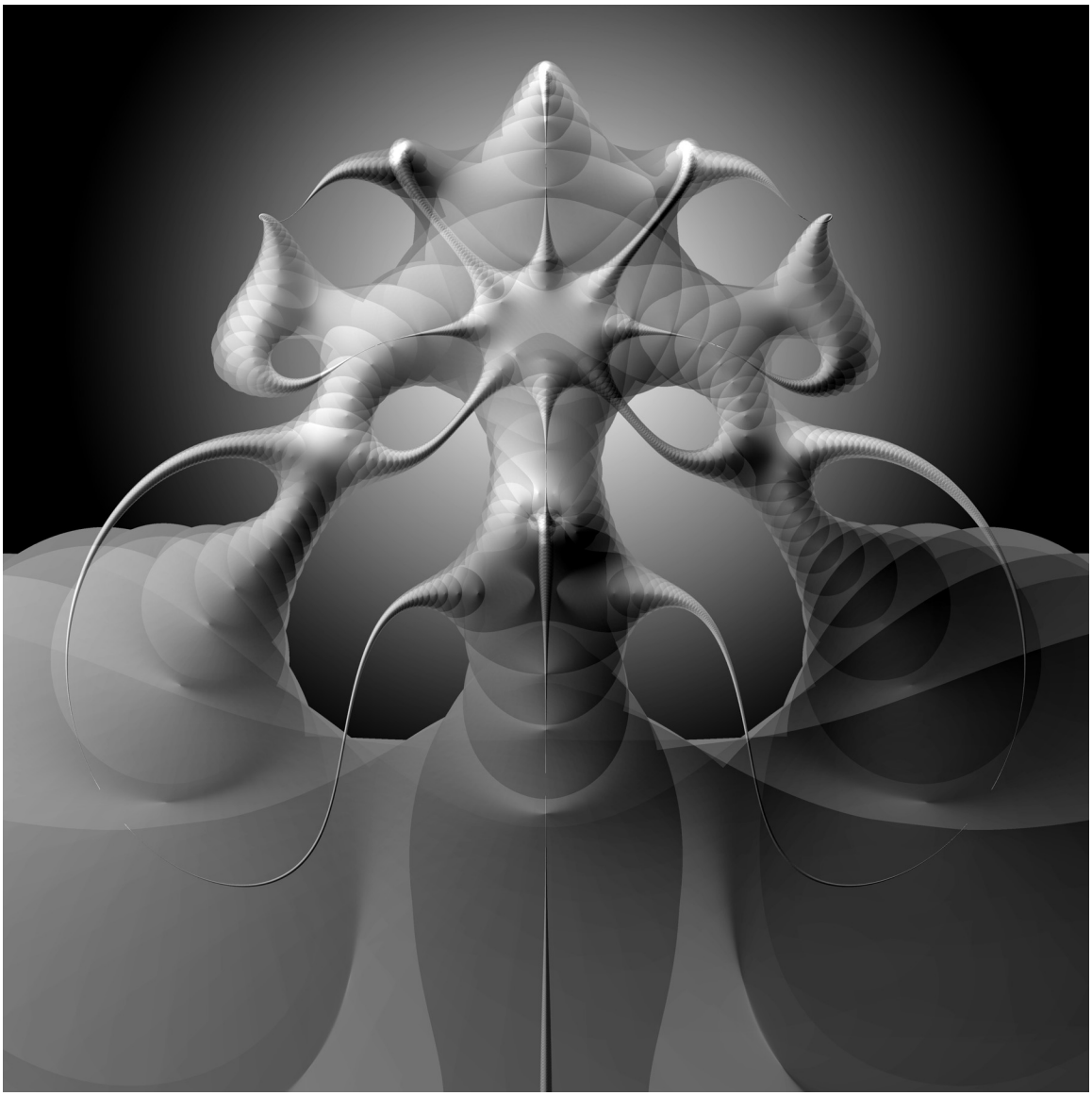


Figure G.2: "The Alien." A result of incorrectly taking Taubin's algorithm to 3D on a 2^{nd}-order root locus.

Figure G.3: Numerical problems in an implicit-function ray-tracer.

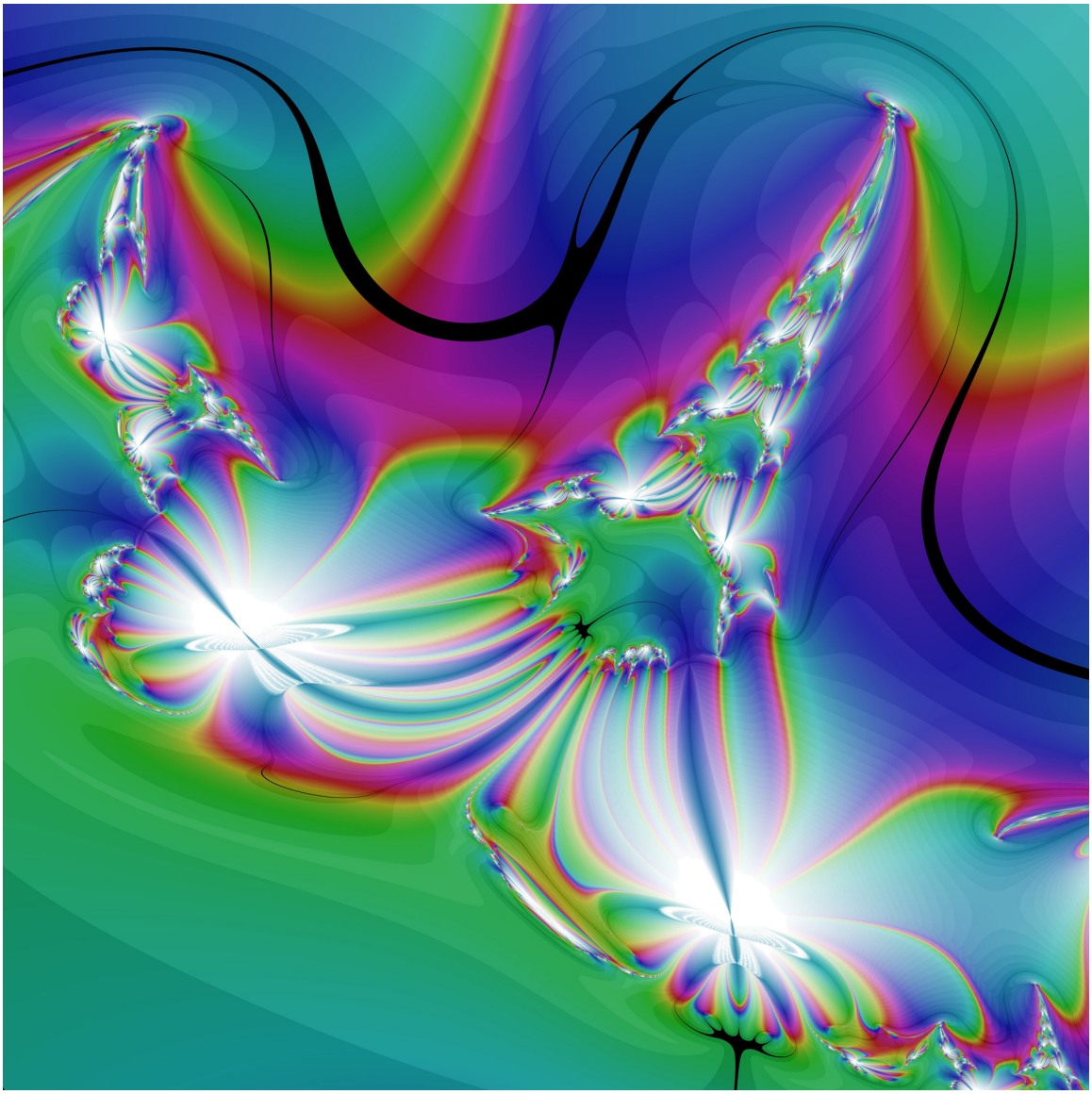


Figure G.4: Detail on a Newton's-Method fractal on $f(s) = \mathrm{Im}(D(s)N(s^*))$, which is the numerator of the expansion of $\mathrm{Im}(D(s)/N(s))$.

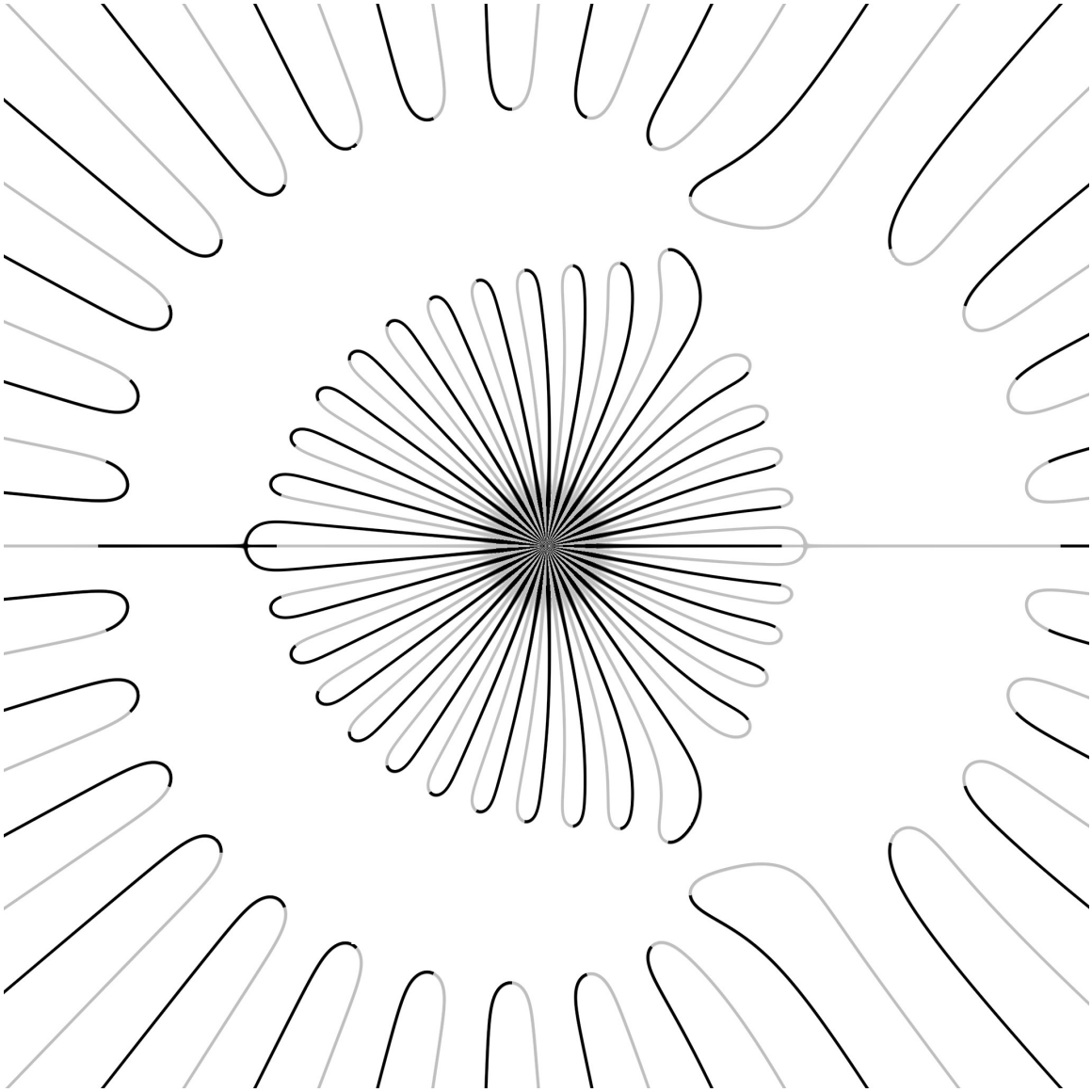


Figure G.5: Locus of feedback around an FIR filter

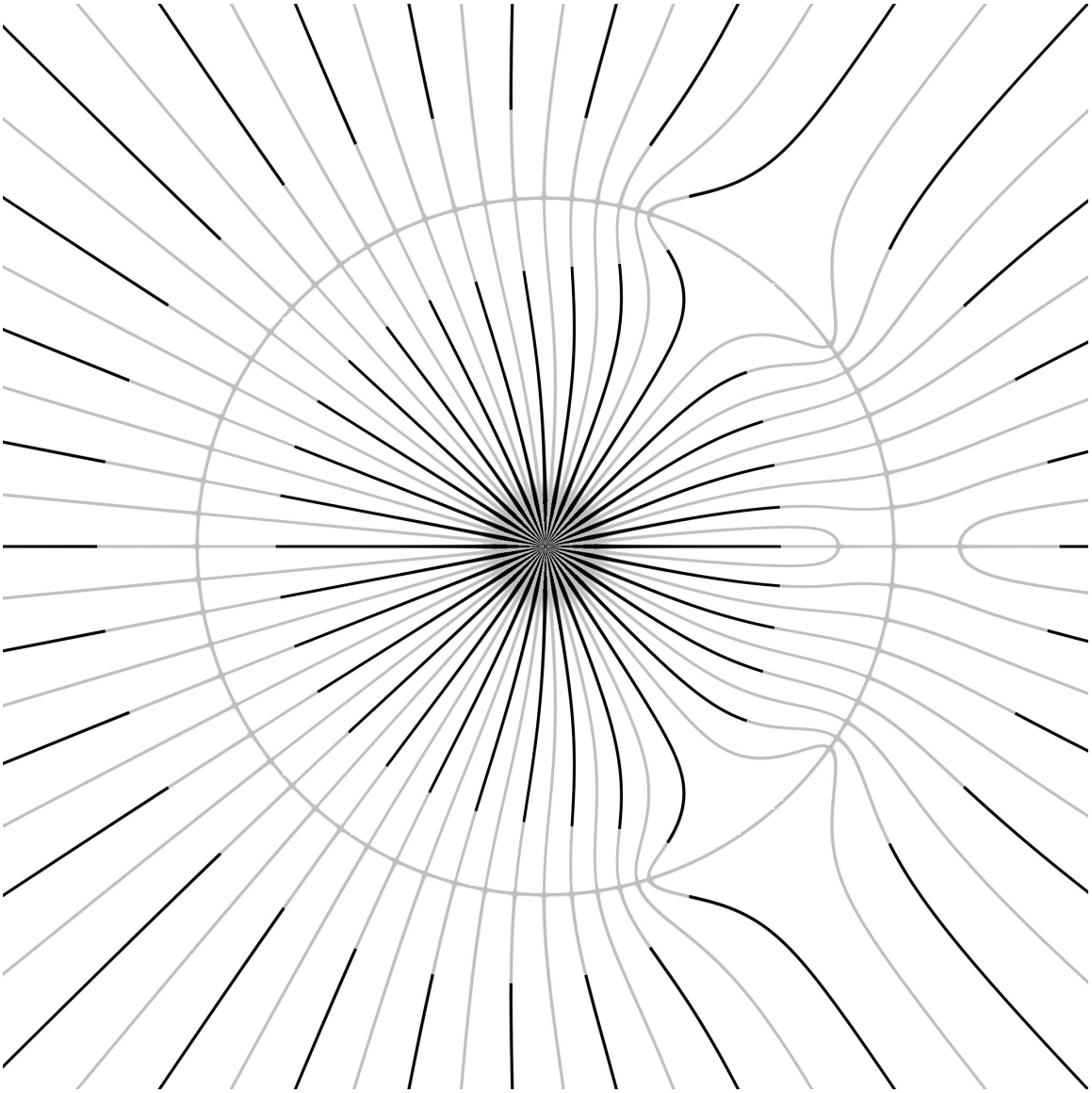


Figure G.6: Locus of feedback around an FIR filter, one more denominator root than Figure G.5.

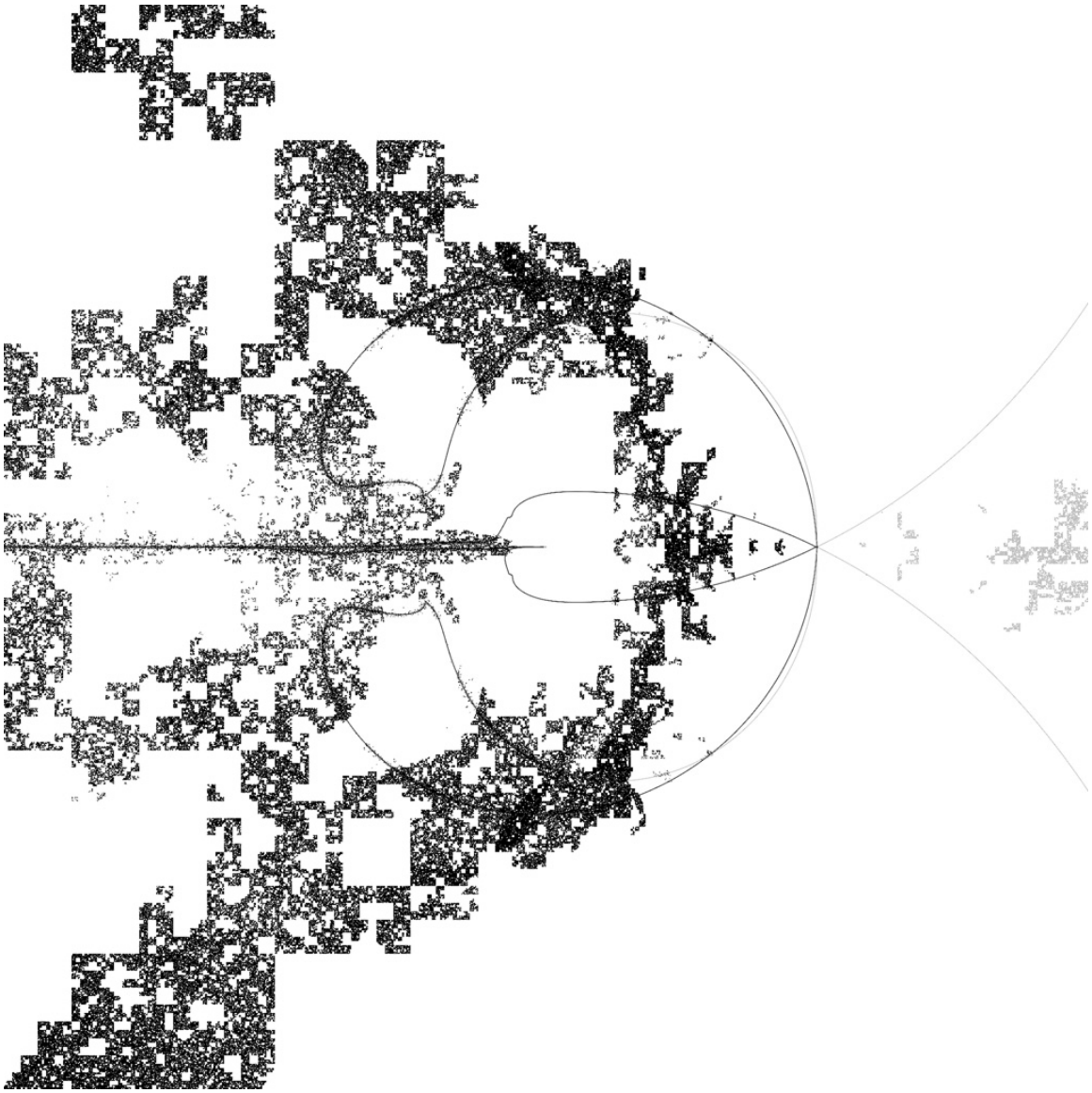


Figure G.7: Numerical problems in a 4^{th}-order locus rendered in Taubin's method

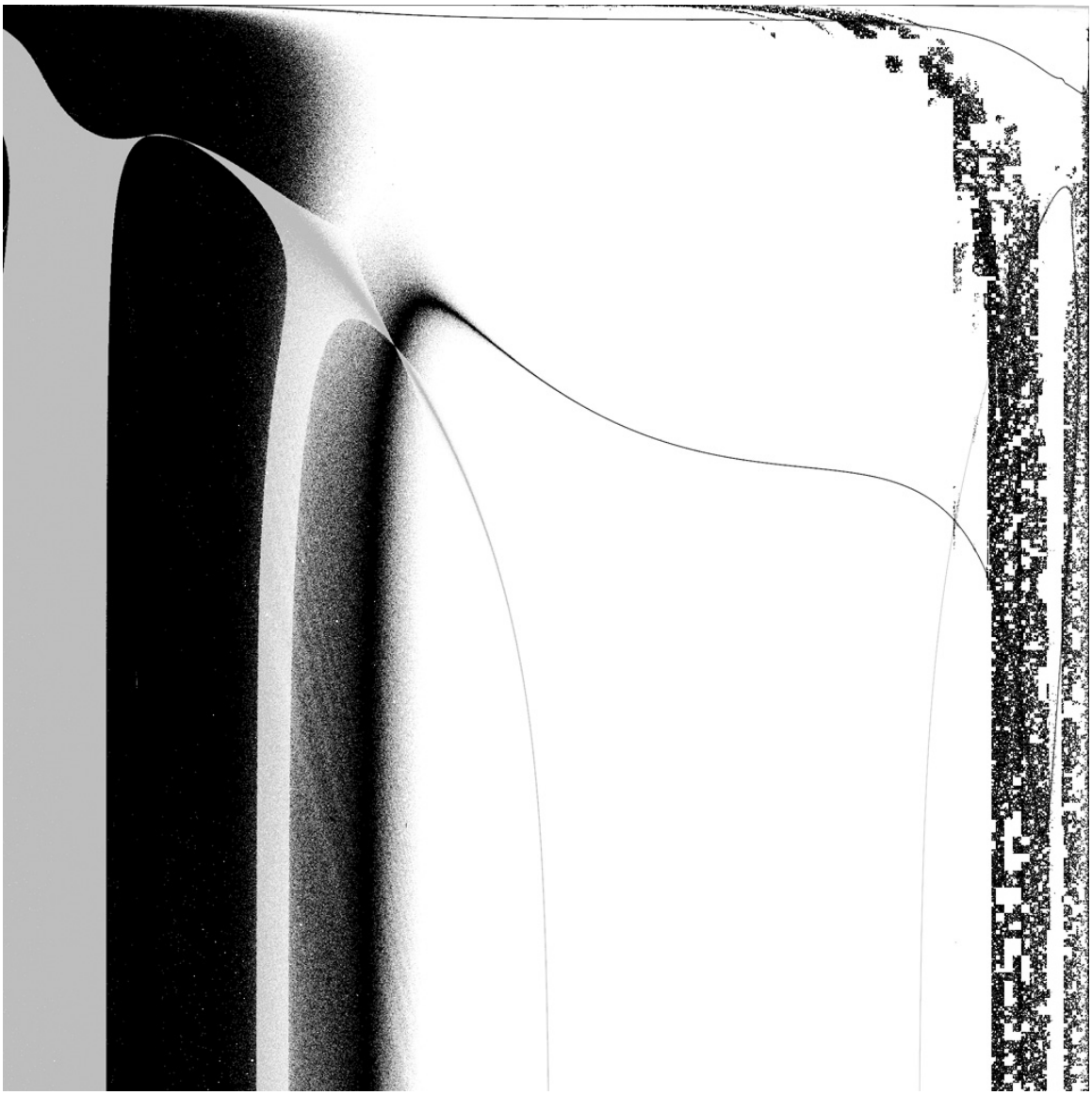


Figure G.8: Numerical problems in a 4^{th}-order locus rendered in Taubin's method, warped axes to (freq,Q). Same system as Figure G.7.

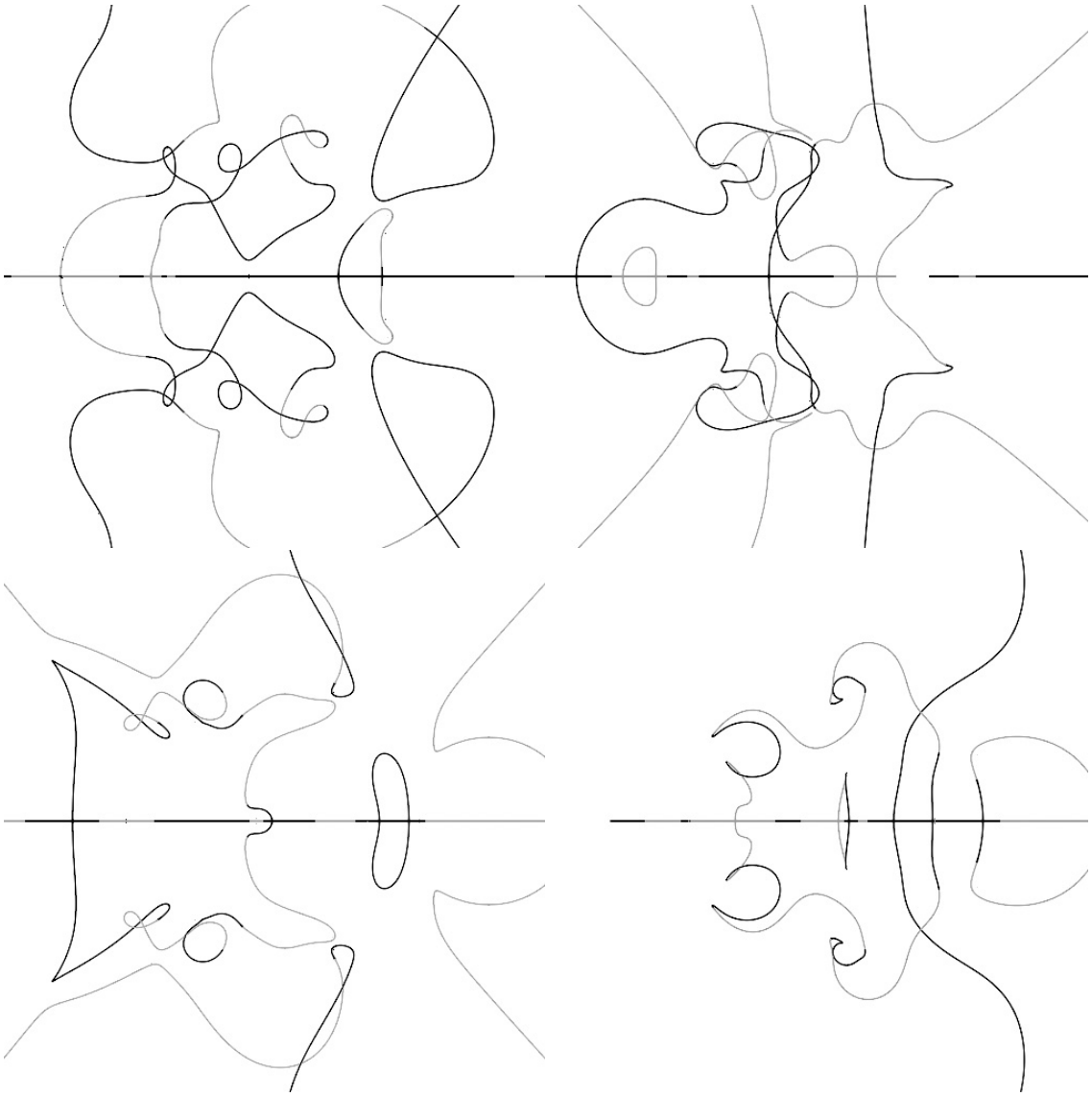


Figure G.9: Some random 4^{th}-order loci

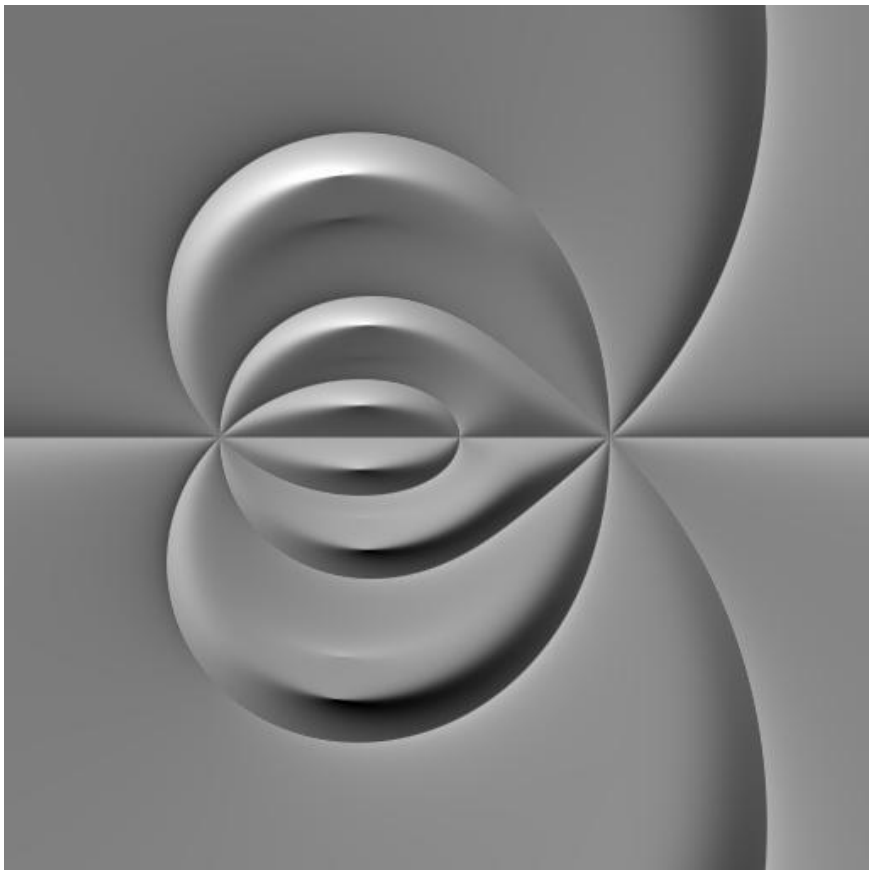


Figure G.10: A visualization of Taubin's 2nd-order distance approximation to a root locus.

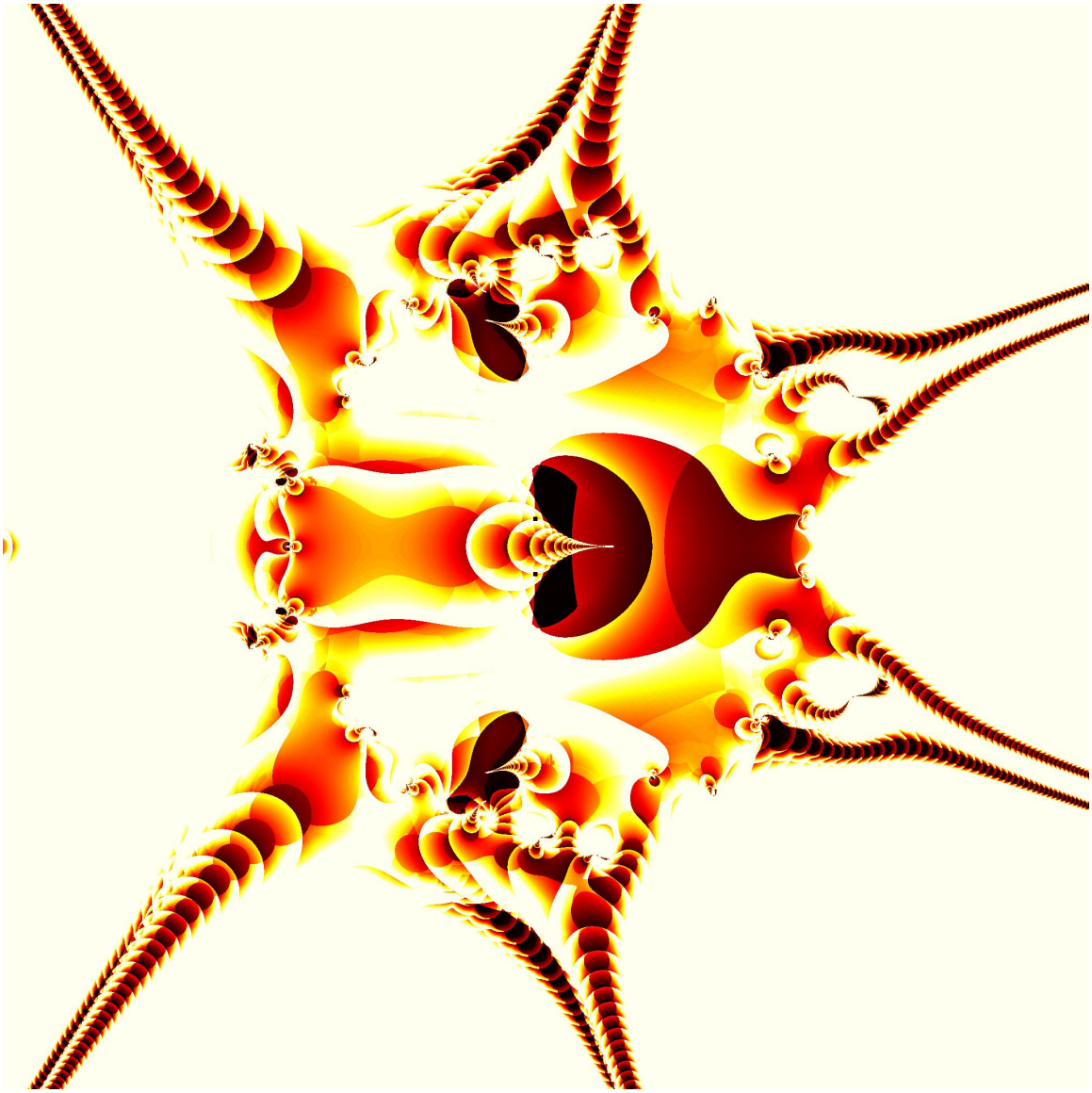


Figure G.11: A variant coloring scheme on the same technique as used in Figure G.2: incorrectly trying to take Taubin's method into 3D on a 2^{nd}-order root locus.

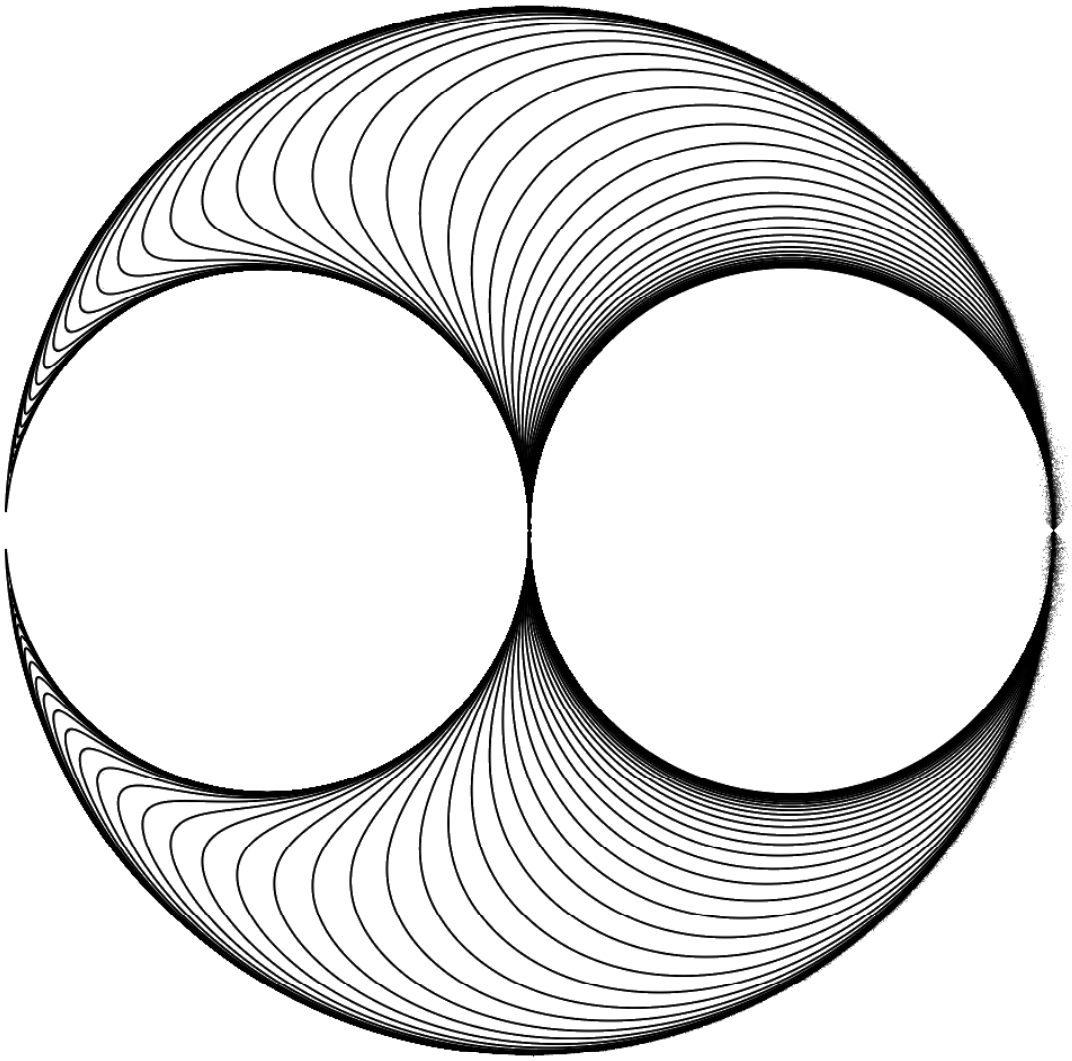


Figure G.12: A family of 2^{nd}-order loci.

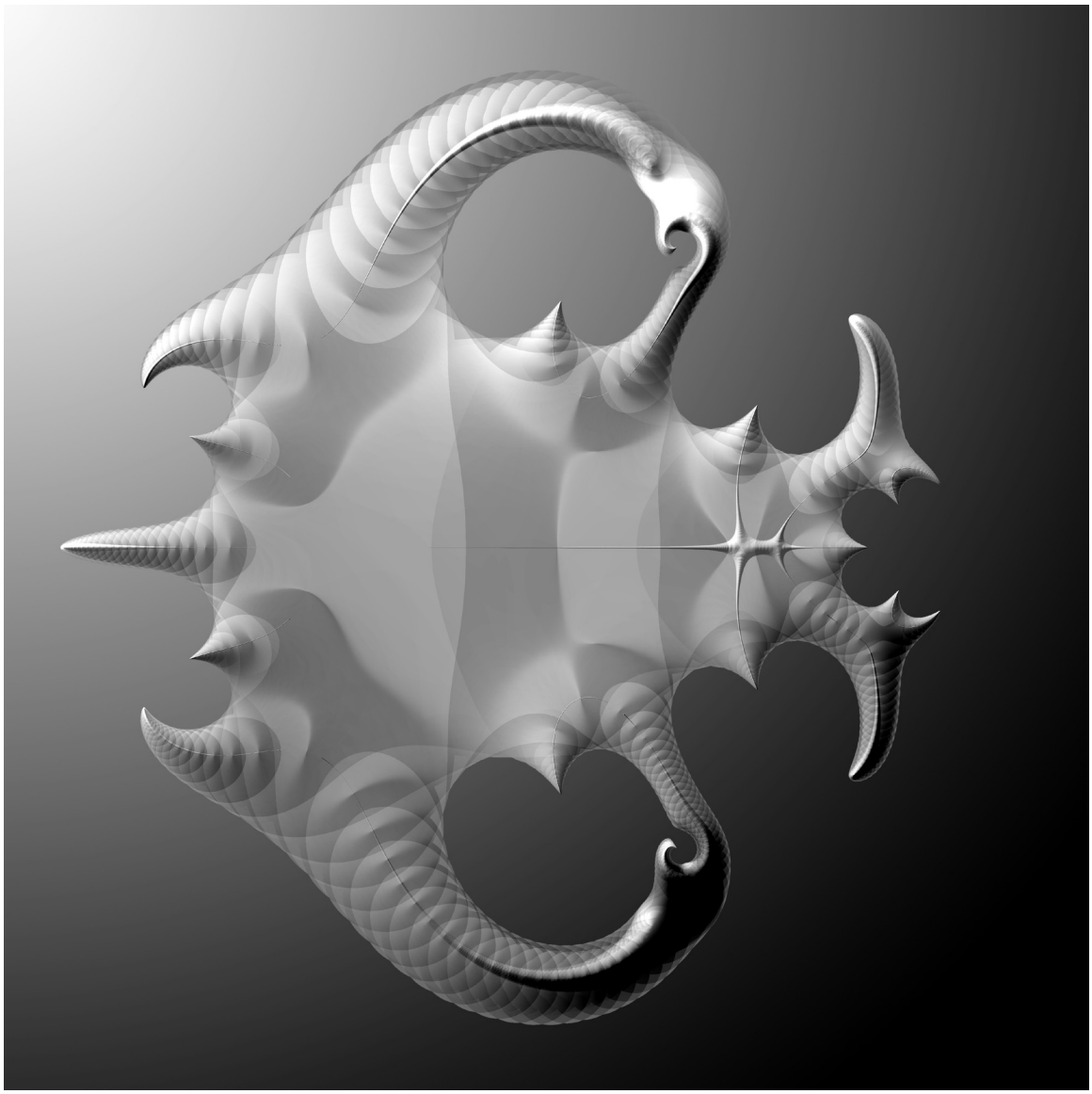


Figure G.13: “The Scarab”

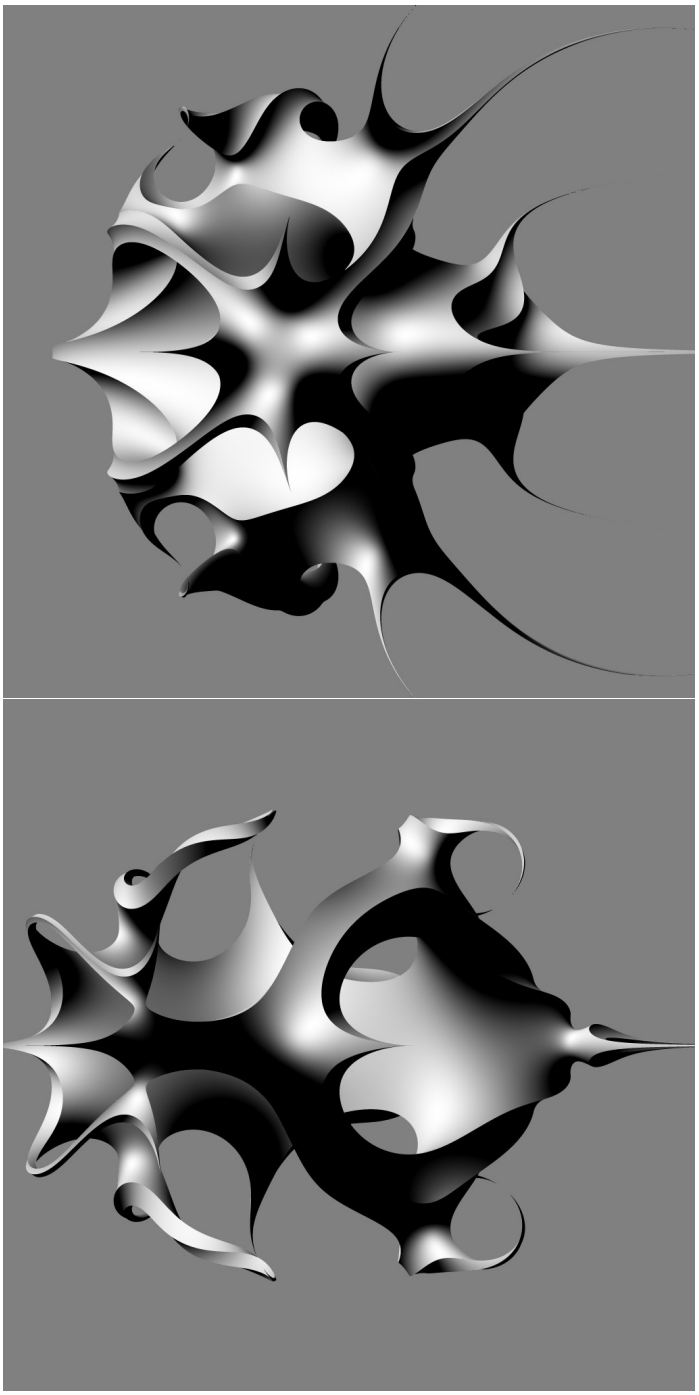


Figure G.14: 2^{nd}-order loci rendered using ray-tracing of $|D + kN_1 + k^2N_2|_\infty = \varepsilon$. Reminiscent of spacecraft from the TV series "Babylon 5".

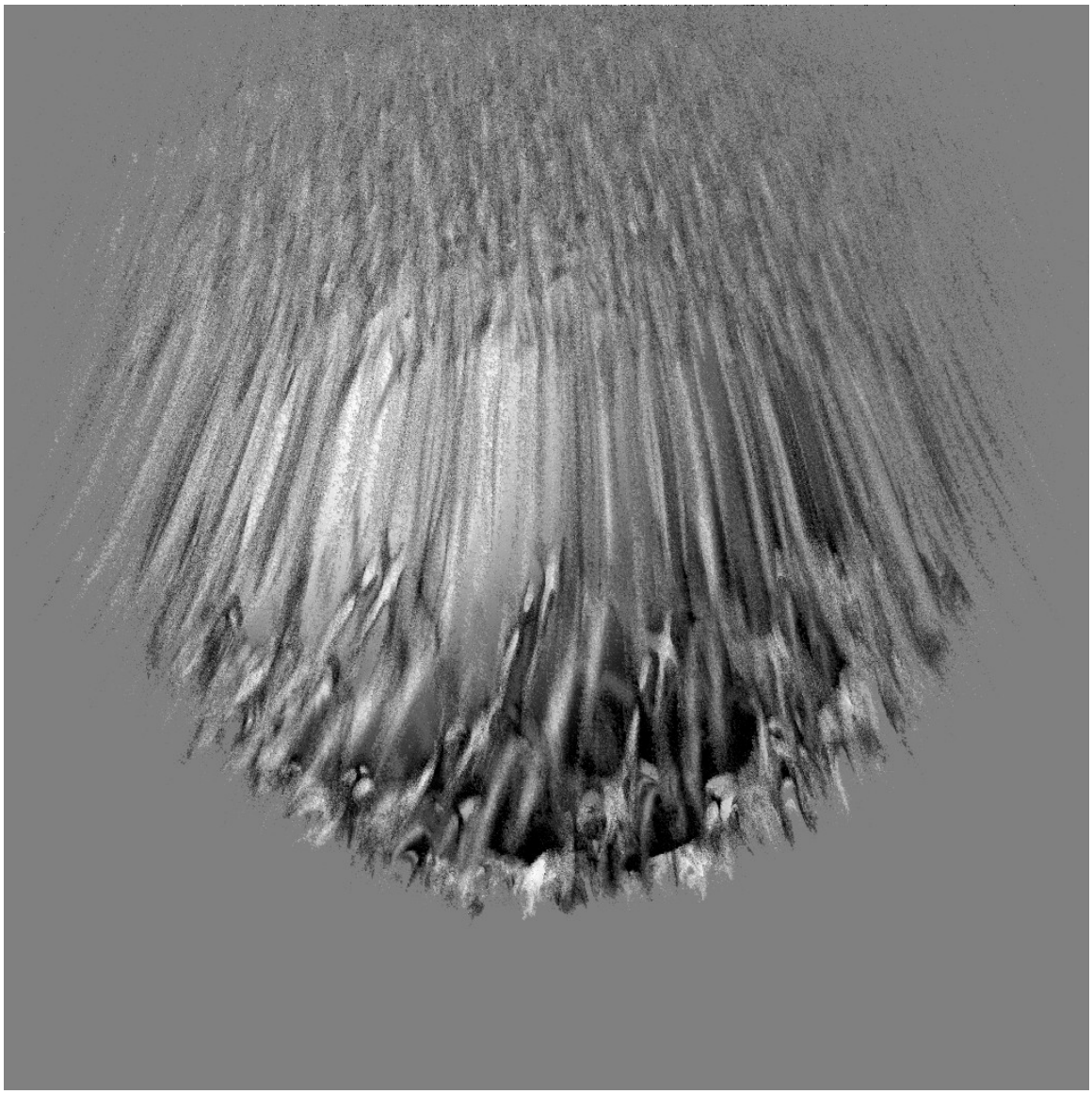


Figure G.15: Another implicit-surface ray-tracing bug.

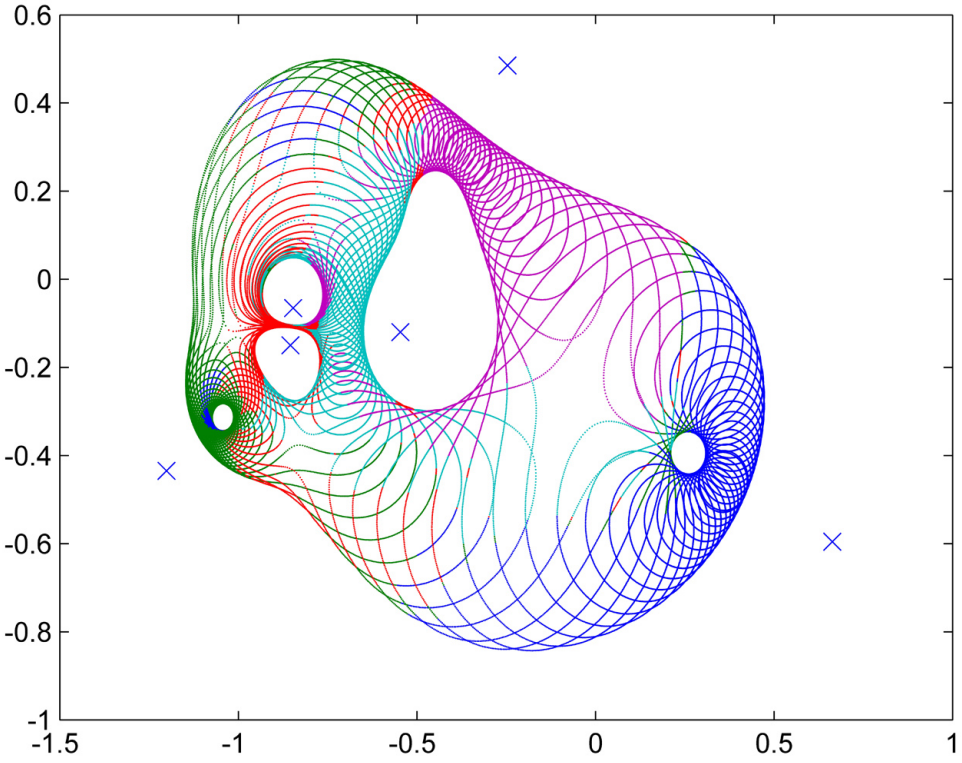


Figure G.16: A gridding of possible zero locations from summing 6 poles, with two poles allowed to vary their phase.

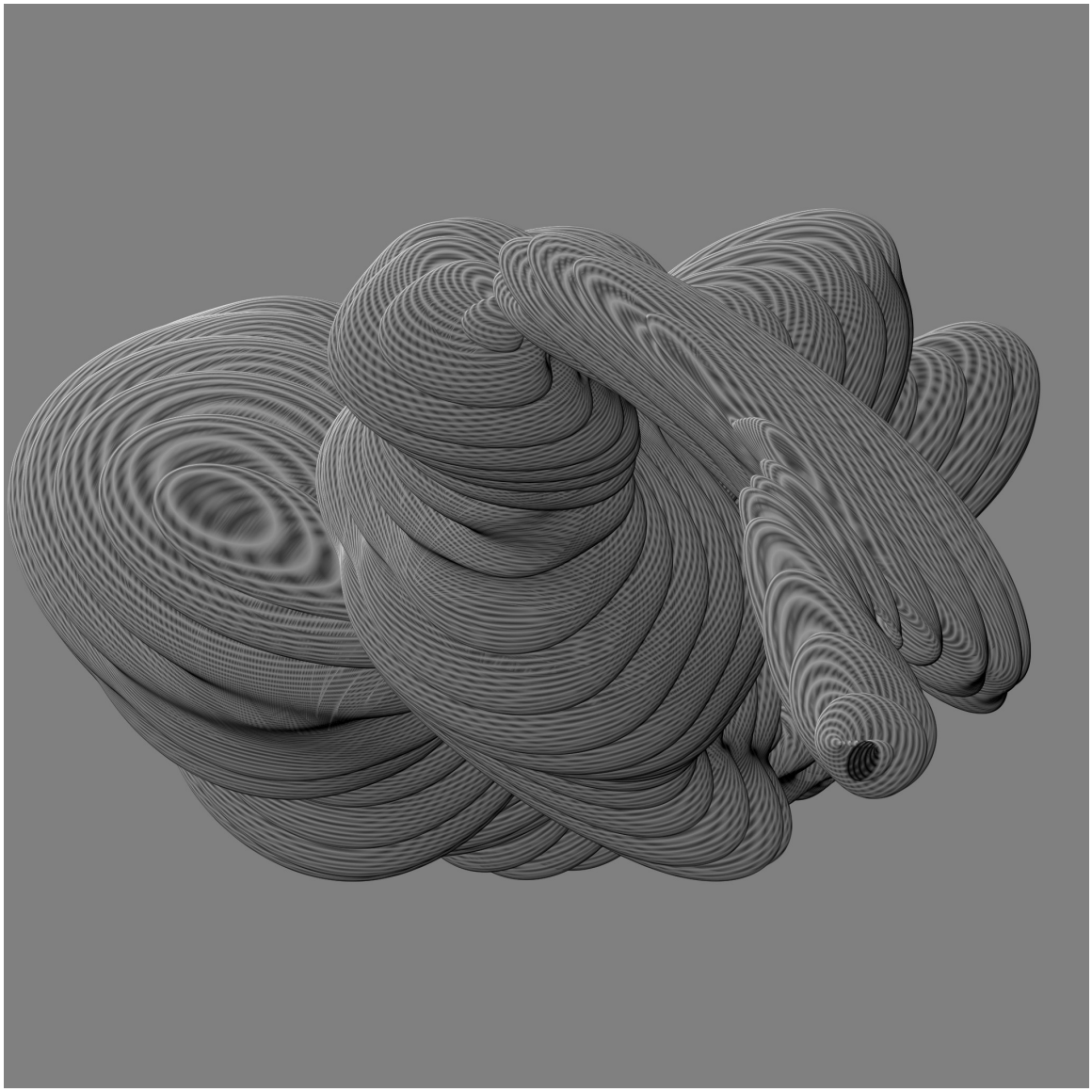


Figure G.17: A test image from attempting to take Taubin's method to 3D in a slightly incorrect manner.

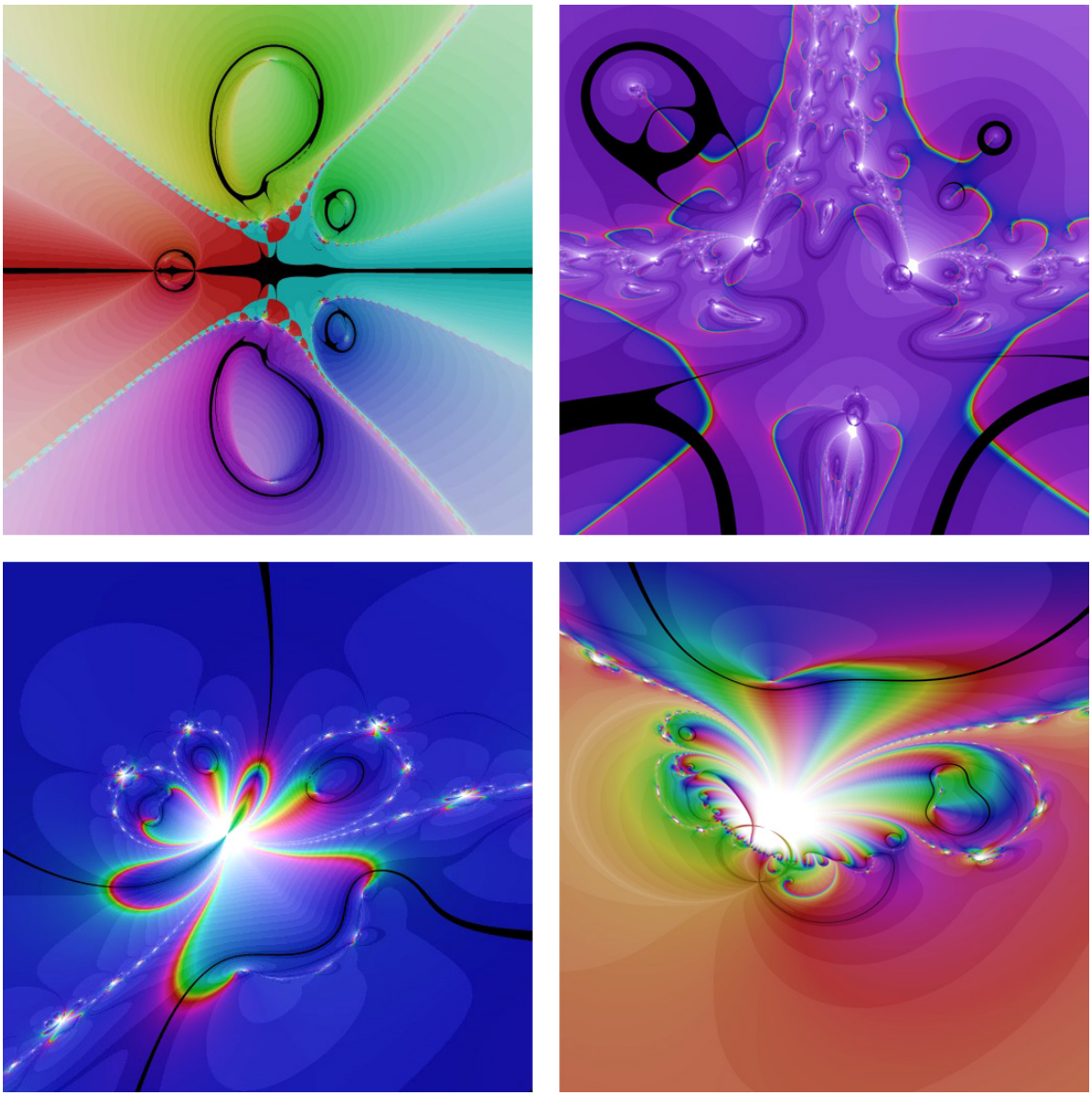


Figure G.18: Newton-Method fractals on root loci rather than discrete-root polynomials.

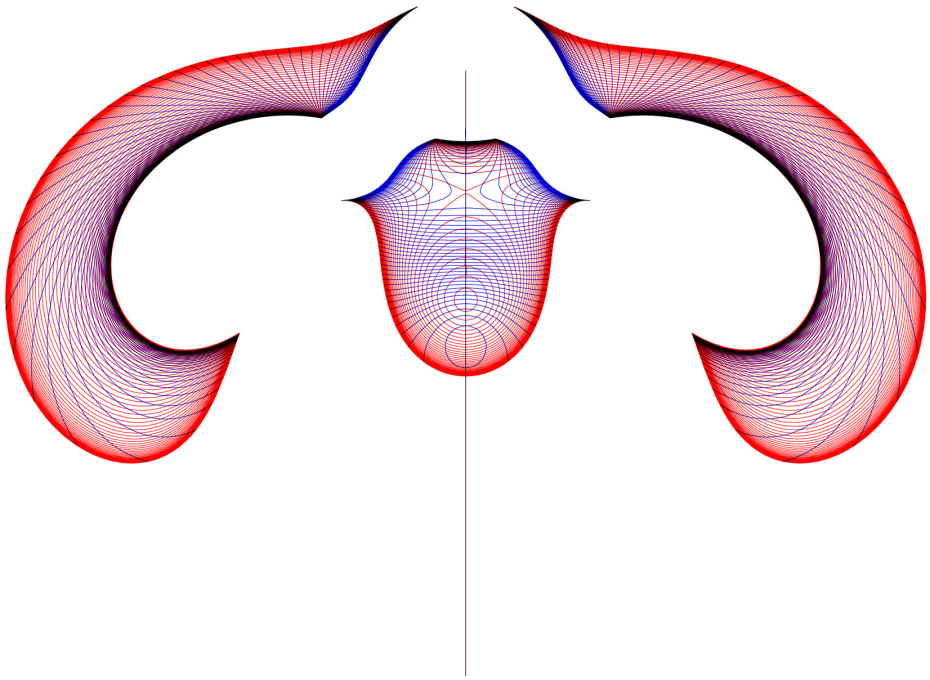


Figure G.19: A family of loci.

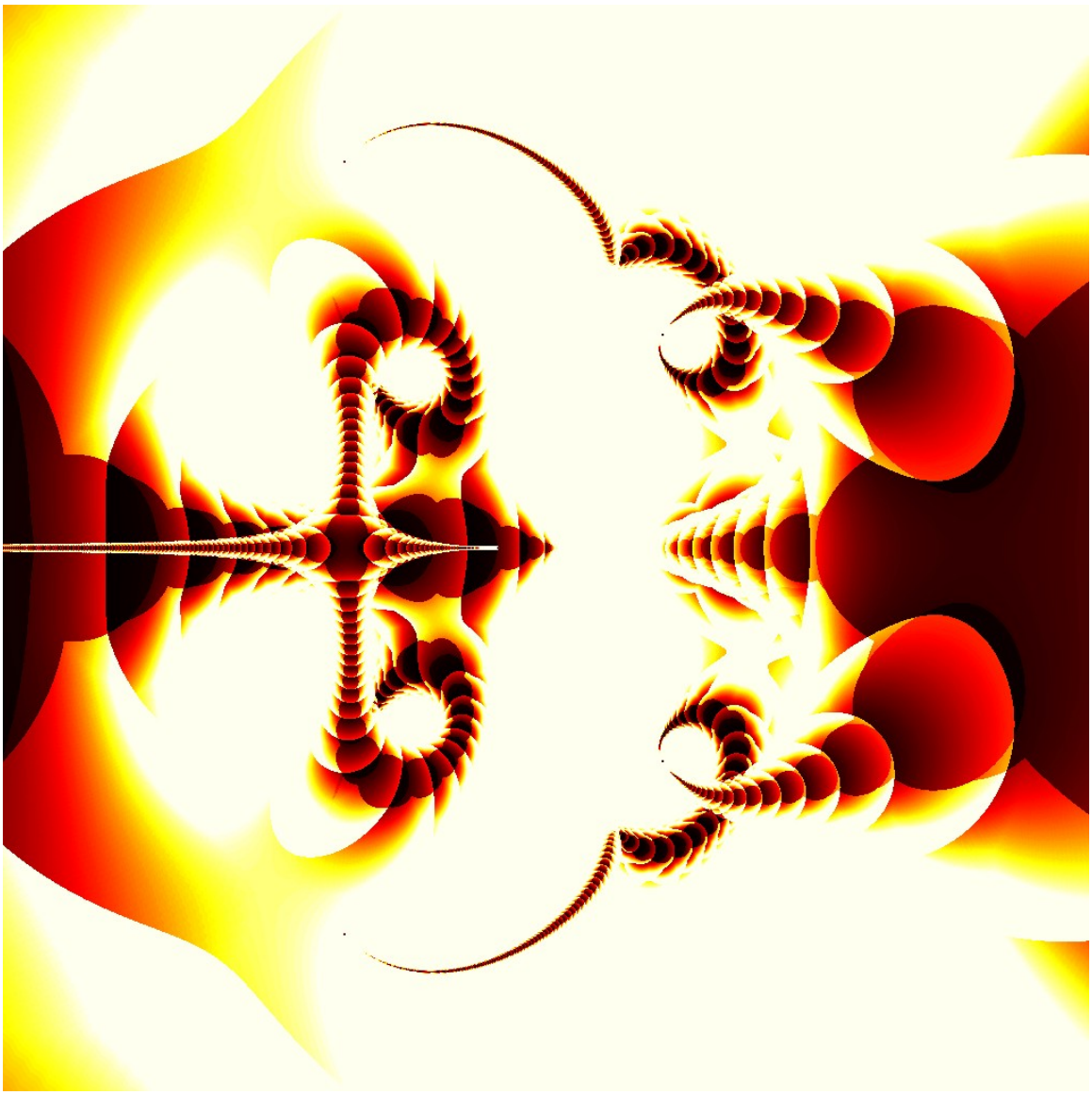


Figure G.20: More 3D attempts at Taubin's method.

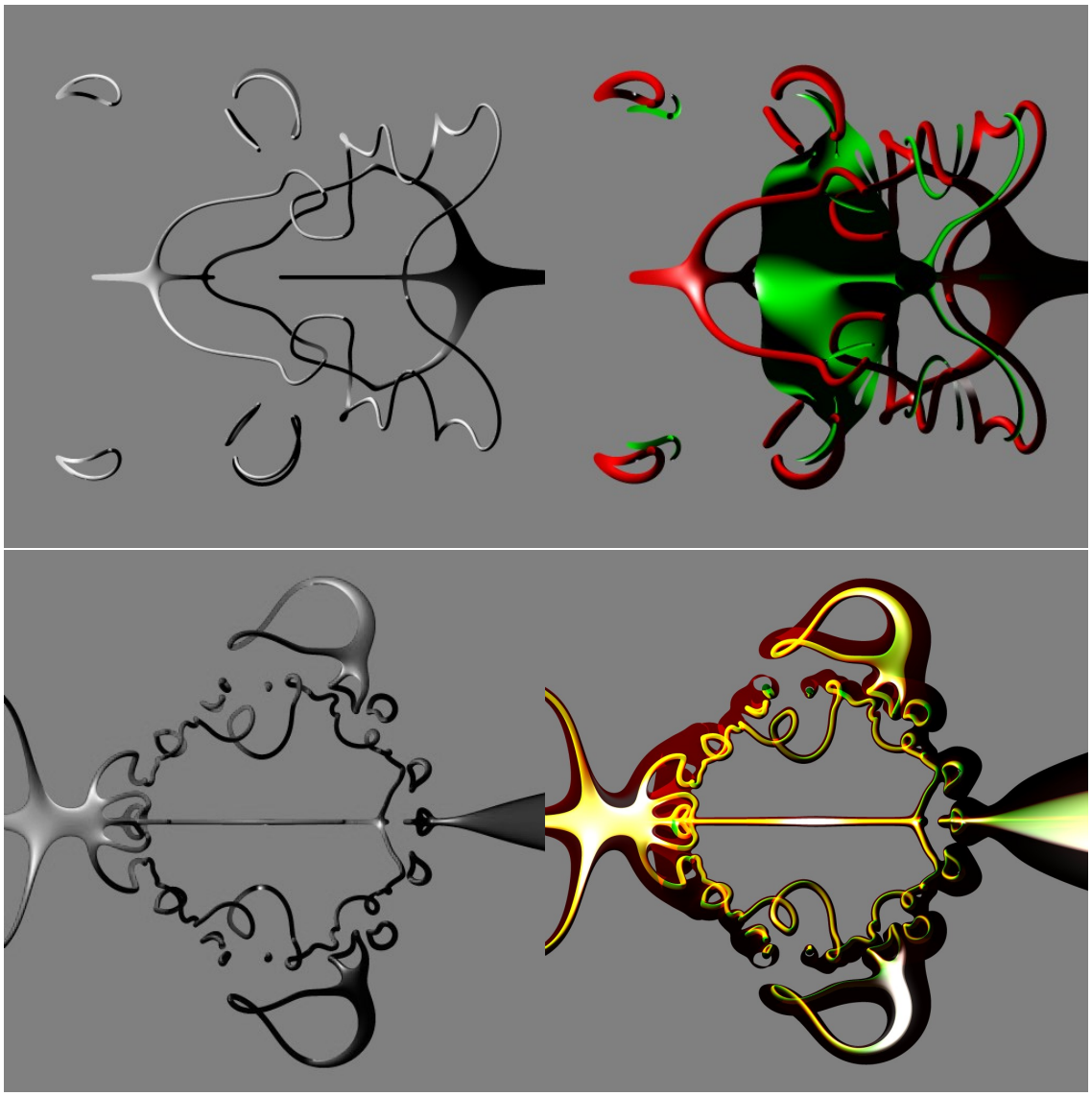


Figure G.21: Ray-tracing $\delta_1 = \varepsilon$ using Taubin's 1st-order distance estimate to various root loci. Left-hand images trace two different ε values at once, giving two different surfaces.

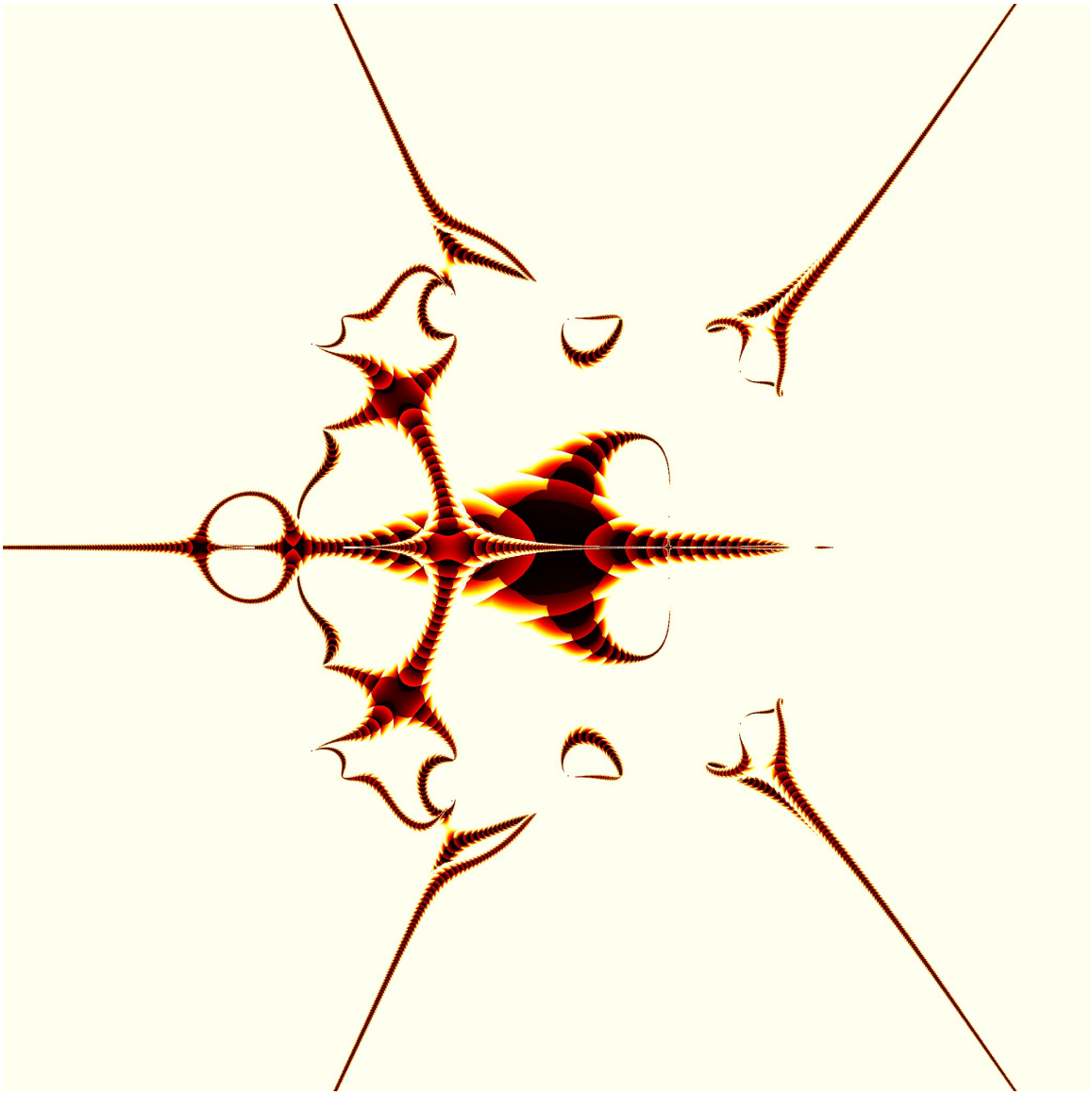


Figure G.22: More 3D attempts at Taubin's method.

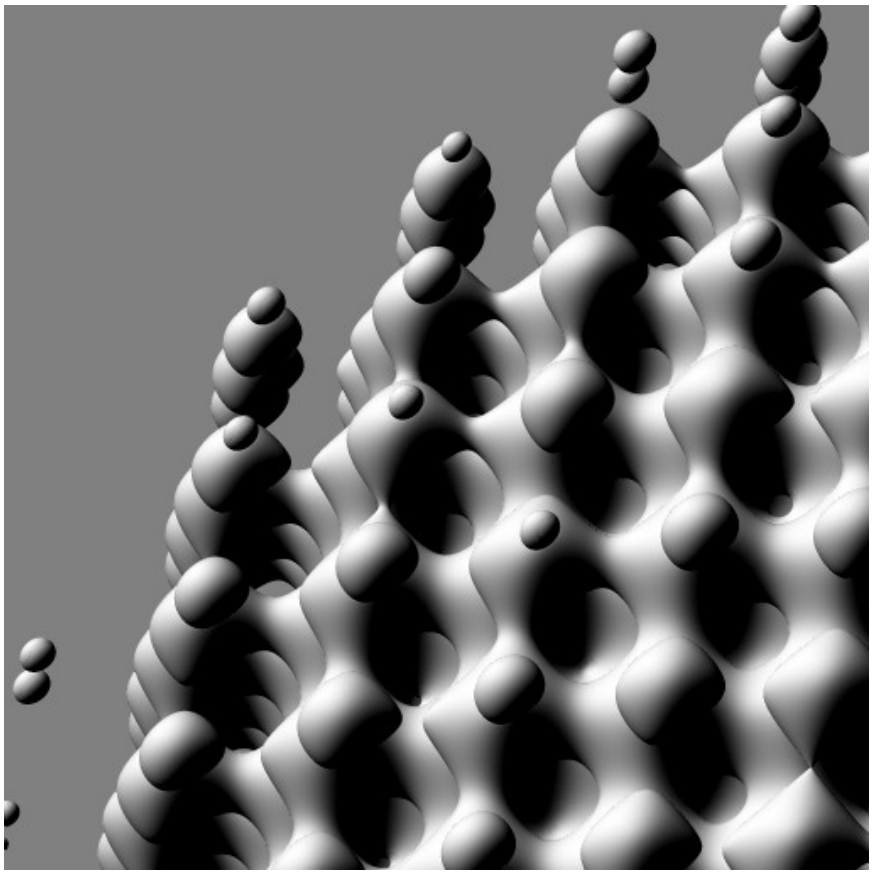


Figure G.23: Implicit-surface ray-tracing test.

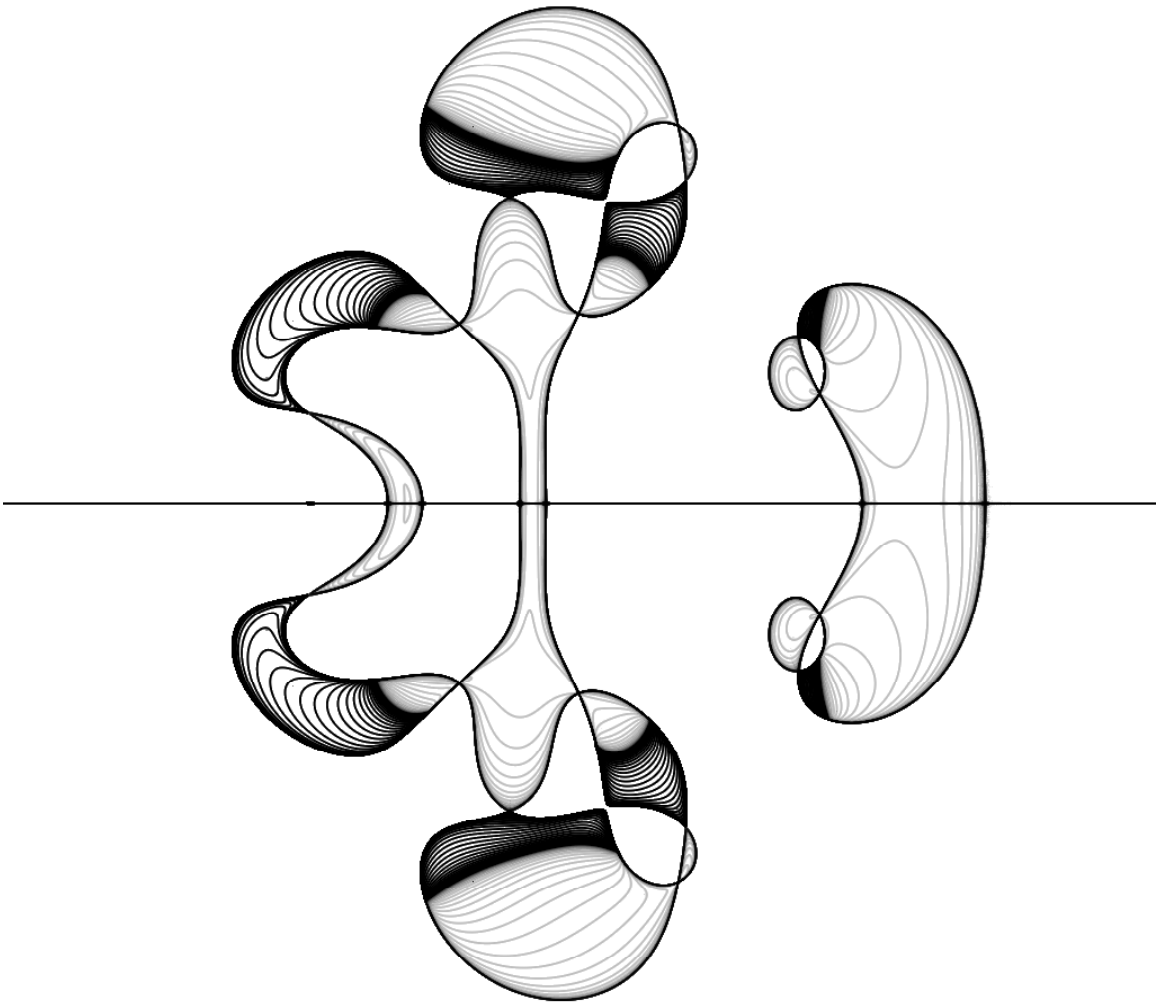


Figure G.24: A family of 2nd-order loci.

Appendix H

Directions for Further Research

H.1 Variable Filters

There are many design variations, especially in the world of the Moog-style filters, which are still unexplored. The research in this thesis just touches the surface on the behavior of such filters.

- Work can be done on a general theory of constant-Q filters and root-locus filters in particular.
- The fact that the various designs of the fourth-order Moog-like filters had such similar p_0-coefficient polynomial sets (after the $p = p_0 - 1$ change of variables) probably implies more than was touched on here.
- Moog-style filters with three feedforward onepole filters and a delay in the feedback are truly fourth-order, and may provide fruitful designs.
- There are many other unexplored combinations and patterns for varying controllable parameters with f_c and Q to try to achieve separation, and maybe achieve further design goals, such as desired nonlinear behaviors.
- Exploration and classification of basic higher-order Root-Locus-Filter block diagram types, beyond the Moog topology, the State-Variable topology and the first-order Root-Locus topology. This would be central to the ability to truly "design" a general root locus filter, with the ability to let the design indicate appropriate topologies, as opposed to designing for a particular topology.
- Continue exploring optimization-based Root-Locus-Filter design. Work on optimizing in spaces other than the z plane, which have better Q sensitivity.
- Explore from a psychoacoustic standpoint. See if there are JNDs for Q variation across an f_c sweep, or for f_c variation across a Q sweep.

- Nonlinear Study, beyond just using it.
  - See if/how Nonlinear System Dynamics applies to the use of musical filters in overdrive.
  - Find objective measures for nonlinear filter performance, probably including psychoacoustics (how close does a filter design sound to a reference filter?), but maybe including non-psychoacoustic measures.
  - Explore how finely must a nonlinear design model various aspects of the target filter. Quantify tradeoffs on what can be left unmodeled (or less accurately modeled).
- As noted in the introduction, there are several filter structures which are quite close to the State-Variable Filter structure. There are probably some very interesting comparisons that can be done between these structures, from the standpoint of musical filtering. In particular, comparative exploration in terms of the ease of Constant-Q design, and in terms of noise and quantization properties.[1]
- An alternative proposal would be to attempt a comprehensive comparative study of all 2^{nd}-order digital filter forms, both known types and types which are suggested by comparisons of known types.
- It was noted that the state-variable filter is commonly used in regions where the poles are real. The work on Moog-style filters was done assuming all complex poles. There is probably some work to be done exploring whether a design needs to extend into real-pole regions similar to those of the state-variable filter, and if/how various designs might achieve that.

H.2 Bandlimited Waveform Synthesis

Vesa Välimäki's inclusion of the frequency-loudness curve in his paper on a reduced-aliasing waveform algorithm [275] points to the need for more psychoacoustic research into the audibility of waveform aliasing. It is not clear in my mind that existing masking and loudness concepts, which in my understanding were derived and measured under the assumption that the masker and the test were independent signals, can fully apply to the understanding of the audibility of different parts of the same signal. Until there are good quantifiable measures of aliasing audibility, design in this area must be done using whatever measures the designer feels like using. As we have seen, there is a spectrum of designs between those which attempt to merely reduce the aliasing and those which attempt to eliminate most or all aliasing. The design space in between these poles is essentially a tradeoff space, and needs objective measures upon which to base the tradeoffs. Vesa's comparison of DPW techniques in [276] does appear to be heading in this direction.

[1] It appears that most of these structures were originally proposed specifically because of their noise and quantization properties.

There is probably good research to be done in hybridizing between various design techniques. For example, using the linear-transform technique with other bandlimited wave generator. As noted in the text, this is already in progress, with the BLEP work being an example.

H.3 Root Locus Drawing

My implementations of Taubin's implicit-function rendering have so far used numerical approximation for deriving the derivatives that the algorithm uses. While this in some cases allowed the rendering of non-trivial functions, one does wonder if the algorithm may be sped up in some cases by the direct calculation of closed-form derivatives. In particular, the requirement of using a numerical root finder for drawing loci of order higher than 2 really slows the algorithm down, and work on deriving expressions for spatial derivatives of imaginary parts of the roots might be able to significantly speed up rendering.

As seen, Taubin's method has numerical problems in certain ranges of operation. This may be partially due to the numerical approximation of the derivatives, but not completely. Taubin's papers do discuss handling such issues, but I have not yet done more than the most basic implementation of the algorithm and further development and research can probably enhance the rendering significantly.

In later papers, Taubin describes variations on the algorithm which have heuristics for reducing the probability of holes in the curve. As noted, I have only made use of the basic algorithm, and there is quite a bit of further work that can be done implementing such variations.

The Root-Locus Explorers which I have implemented have turned out to be very nice tools for gaining root-locus intuition, and there are many directions in which they could be extended. These may have use as teaching aids in classical control-systems courses, as well as in variable-filter work.

There is a chance that some of the research in rendering algorithms may have application in the area of drawing robust root loci. I have run across a paper [269] which presents a new algorithm for drawing robust loci which doesn't rely purely on sampling the parameter space, and which produces much cleaner looking loci. I have a feeling there is a conceptual connection between it and some of ideas behind the ray-tracing-based renderers.

The adaptive-step-size algorithms have quite a few areas of further research (this includes the predictor-corrector methods as well as the root-finder methods) Currently, I have implemented quite a few heuristics to get around various ways in which the algorithms have problems, which should be replaced with non-heuristic rules. For example, it should be possible to use the local root-sensitivity function to identify the location of singularities in fewer subdivisions than the current blind subdivision algorithms I am using.

Bibliography

[1] J. S. Abel and D. P. Berners, "Filter design using second-order peaking and shelving sections," in *Proceedings of the 2004 International Computer Music Conference, Miami, Florida*, Computer Music Association, 2004.

[2] M. Abramowitz and I. A. Stegun, eds., *Handbook of Mathematical Functions*, New York: Dover, 1965.

[3] J.-M. Adrien, "The missing link: Modal synthesis," in *Representations of Musical Signals* (G. De Poli, A. Picialli, and C. Roads, eds.), pp. 269–297, Cambridge, MA: MIT Press, 1991.

[4] R. Agarwal and C. S. Burrus, "New recursive digital filter structures having very low sensitivity and roundoff noise," *IEEE Transactions on Circuits and Systems—I: Fundamental Theory and Applications*, vol. 22, pp. 921–927, Dec. 1975.

[5] S. T. Alexander and V. L. Stonick, "Fast adaptive polynomial root tracking using a homotopy continuation method," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, Minneapolis*, vol. 3, (New York), pp. 480–483, IEEE Press, Apr. 1993.

[6] E. L. Allgower and K. Georg, "Continuation and path following," *Acta Numerica*, pp. 1–64, 1992.

[7] E. L. Allgower and K. Georg, "Numerical path following," in *Handbook of Numerical Analysis* (P. G. Ciarlent and J. L. Lions, eds.), vol. 5, pp. 3–207, North-Holland, 1997.

[8] E. L. Allgower and K. Georg, *Introduction to Numerical Continuation Methods*, SIAM Classics in Applied Mathematics 45, SIAM, 2002, Reprint, originally published in 1979 by John Wiley and Sons.

[9] D. K. Arrowsmith and C. M. Place, *An Introduction to Dynamical Systems*, Cambridge University Press, 1990.

[10] R. H. Ash and G. R. Ash, "Numerical computation of root-loci using the Newton-Raphson technique," *IEEE Transactions on Automatic Control*, pp. 576–582, Oct. 1968.

[11] A. Askenfelt, ed., *Five Lectures on the Acoustics of the Piano*, Stockholm: Royal Swedish Academy of Music, 1990, lectures by H. A. Conklin, Anders Askenfelt and E. Jansson, D. E. Hall, G. Weinreich, and K. Wogram. Sound example CD included. Publication number 64. `http://www.speech.kth.se/music/5_lectures/`.

[12] B. R. Barmish and R. Tempo, "The robust root locus," in *Proceedings of the 27th IEEE Conference on Decision and Control*, pp. 1386–1391, 1988.

[13] P. Beckmann and T. Stilson, "An efficient asynchronous sampling-rate conversion algorithm for multi-channel audio applications," *Audio Engineering Society Convention*, Preprint 6553, Oct. 2005.

[14] M. Bellanger, "Improved design of long FIR filters using the frequency masking technique," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, Atlanta*, (New York), IEEE Press, May 1996, Paper DSP1.1.

[15] G. A. Bendrikov and K. F. Teodorchik, "The analytic theory of constructing root loci," *Avtomatika i Telemekhanika (Russia)*, vol. 20, no. 3, pp. 355–358, 1959.

[16] S. Bilbao, *Wave and Scattering Methods for the Numerical Integration of Partial Differential Equations*, PhD thesis, Stanford University, June 2001, `http://ccrma.stanford.edu/~bilbao/`.

[17] S. Bilbao, *Wave and Scattering Methods for Numerical Simulation*, New York: John Wiley and Sons, Inc., July 2004.

[18] S. Bilbao, "Time-varying generalizations of all-pass filters," *IEEE Signal Processing Letters*, vol. 12, pp. 376–379, May 2005.

[19] R. Bjorkman, "A brief note on polygon filters," *Electronotes*, vol. 97, pp. 7–9.

[20] J. Bloomenthal, *Introduction to Implicit Surfaces*, Morgan Kaufmann, 1997.

[21] O. J. Bonello, "Modular parametric equalizer-filter, a new way to synthesize the frequency response," *Audio Engineering Society Convention*, Preprint 1170, Oct. 1976.

[22] R. C. Boulanger, ed., *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming*, MIT Press, Mar. 2000.

[23] P. Bowron, M. R. J. Motlagh, and A. A. Muhieddine, "Harmonic characterisation of feedback systems incorporating saturation nonlinearities," *Electronics Letters*, vol. 27, pp. 1865–1867, Sept. 1991.

[24] S. Boyd, "Personal communication," 2006, Description of a method for multivariable rational fitting via convex optimization, and its application to variable filter design.

[25] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, Feb. 2004, `http://www.stanford.edu/~boyd/cvxbook.html`.

[26] R. Bracewell, *The Fourier Transform and its Applications*, New York: McGraw-Hill, 1965.

[27] E. Brandt, "Hard sync without aliasing," in *Proceedings of the 2001 International Computer Music Conference, Havana*, 2001, URL = `http://www-2.cs.cmu.edu/~eli/papers/icmc01-hardsync.pdf`.

[28] R. Bristow-Johnson, "The equivalence of various methods of computing biquad coefficients for audio parametric equalizers," *Audio Engineering Society Convention*, 1994, `http://www.harmony-central.com/Effects/Articles/EQ_Coefficients/-EQ-Coefficients.pdf`.

[29] J. W. Bruce and P. J. Giblin, *Curves And Singularities*, Cambridge University Press, 2 ed., 1992.

[30] P. Burk, "JSyn — audio synthesis API for Java and C," in *Proceedings of the 1998 International Computer Music Conference, Michigan*, Computer Music Association, 1998.

[31] P. Burk, "Bandlimited oscillators using wave table synthesis," in *Audio Anecdotes II* (K. Greenbaum and R. Barzel, eds.), Wellesley, MA: A. K. Peters, Ltd., 2004.

[32] C. I. Byrnes and D. S. Gilliam, "Asymptotic properties of root locus for distributed parameter systems," in *Proceedings of the 27th Conference on Decision and Control, Austin, TX*, vol. 3, pp. 45–51, Dec. 1988.

[33] C. I. Byrnes, D. S. Gilliam, and J. He, "Root-locus and boundary feedback design for a class of distributed parameter systems," *SIAM Journal of Control and Optimization*, pp. 1364–1427, Sept. 1994.

[34] C. Cadoz, A. Luciani, and J. L. Florens, "Cordis-anima: A modeling and simulation system for sound and image synthesis. the general formalism," *Computer Music Journal*, vol. 17, pp. 19–29, Winter 1993.

[35] T. J. Cavicci, "Phase-root locus and relative stability," *IEEE Control Systems*, pp. 69–77, Aug. 1996.

[36] J. Chadabe, *Electric Sound - The Past And Promise of Electronic Music*, Upper Saddle River, NJ: Prentice Hall, 1997.

[37] H. Chamberlin, *Musical Applications of Microprocessors*, New Jersey: Hayden Book Co., Inc., 1980.

[38] C. S. Chang, "An analytical method for obtaining the root locus with positive and negative gain," *IEEE Transactions on Automatic Control*, pp. 92–94, Jan. 1965.

[39] C. F. Chen and M. M. Chen, "A scanning method for drawing root loci for sampled-data feedback systems," in *Proceedings of the 1985 American Control Conference, Boston, MA*, vol. 2, pp. 1020–1025, June 1985.

[40] J.-J. Chen and C. Hwang, "Characterization of robust root loci of polytopes of polynomials," *Journal of Scientific Computing*, vol. 11, no. 2, pp. 155–166, 1996.

[41] B. Chidlaw, "A fourth-order state-variable filter," *Electronotes*, vol. 99, pp. 3–12.

[42] J. M. Chowning, "The synthesis of complex audio spectra by means of frequency modulation," *Journal of the Acoustical Society of America*, vol. 21, no. 7, pp. 526–534, 1973, reprinted in [223].

[43] I. D. Chusid, editor, *Manhattan Research Inc.*, Basta Records, 2000.

[44] A. G. Constantinides, "Frequency transformations for digital filters," *Electronics Letters*, vol. 3, pp. 487–489, 1967.

[45] A. G. Constantinides, "Frequency transformations for digital filters," *Electronics Letters*, vol. 4, pp. 115–116, 1968.

[46] A. G. Constantinides, "Spectral transformations for digital filters," *Proceedings of the IEE*, vol. 117, pp. 1585–1590, Aug. 1970.

[47] P. Cook, "Physically informed sonic modeling (PhISM): Percussive synthesis," in *Proceedings of the 1996 International Computer Music Conference, Hong Kong*, 1996.

[48] P. Cook, "Physically informed sonic modeling (PhISM): Synthesis of percussive sounds," *Computer Music Journal*, vol. 21, no. 3, 1997.

[49] P. R. Cook, 1995, Personal discussion.

[50] P. R. Cook, "A hierarchical system for controlling synthesis by physical modeling," in *Proceedings of the 1995 International Computer Music Conference, Banff*, pp. 108–109, Computer Music Association, 1995.

[51] D. Cox, J. Little, , and D. O'Shea, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry And Commutative Algebra*, New York: Springer Verlag, 2 ed., 1997.

[52] R. E. Crochiere and L. R. Rabiner, "On the properties of frequency transformations for variable cutoff linear phase digital filters," *IEEE Transactions on Circuits and Systems*, pp. 684–686, Nov. 1976.

[53] R. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1983.

[54] C. Cross, "Verbal story of using rubber sheet to assist root locus." Part of the presentation of root-locus in a control-systems class at Walla Walla College, 1991.

[55] T. Darter and G. Armbtuster, *The Art of Electronic Music, The Instruments, Designers, and Musicians Behind the Artistic and Popular Explosion of Electronic Music*, A KEYBOARD Book, New York: GPI Books, 2002.

[56] J. Dattorro, "The implementation of recursive digital filters for high-fidelity audio," *Journal of the Audio Engineering Society*, vol. 36, pp. 851–878, Nov. 1988, Comments, ibid. (Letters to the Editor), vol. 37, p. 486 (1989 June); Comments, ibid. (Letters to the Editor), vol. 38, pp. 149-151 (1990 Mar.).

[57] J. Dattorro, "Effect design, part 1: Reverberator and other filters," *Journal of the Audio Engineering Society*, vol. 45, pp. 660–684, September 1997.

[58] J. Dattorro, "Effect design, part 2: Delay-line modulation and chorus," *Journal of the Audio Engineering Society*, vol. 45, pp. 764–788, Oct. 1997.

[59] J. J. D'Azzo, C. H. Houpis, and S. N. Sheldon, *Linear Control System Analysis and Design with MATLAB*, Control Engineering Series, New York: Marcel Dekker, 2003.

[60] T.-B. Deng, "Design of recursive 1-D variable filters with guaranteed stability," *IEEE Transactions on Circuits and Systems—II: Analog and Digital Signal Processing*, vol. 44, pp. 689–695, Sept. 1997.

[61] T.-B. Deng, "Variable fractional-delay filter design using weighted least-squares singular-value decomposition," in *Proceedings ICSP'04: 2004 7th International Conference on Signal Processing*, vol. 1, pp. 54–57, 2004.

[62] T. Deng, "Design of complex-coefficient variable digital filters using successive vector-array decomposition," *IEEE Transactions on Circuits and Systems*, vol. 52, pp. 932–942, May 2005.

[63] T.-B. Deng and T. Soma, "Variable digital filter design using the outer product expansion," *IEE Journal of the Vis. Image Signal Processing*, vol. 141, pp. 123–128, Apr. 1994.

[64] Y. Ding and D. Rossum, "Filter morphing of parametric equalizers and shelving filters for audio signal processing," *Journal of the Audio Engineering Society*, vol. 43, pp. 821–826, Oct. 1995.

[65] C. Dodge and T. A. Jerse, *Computer Music*, New York: Schirmer, 1985.

[66] R. C. Dorf and R. H. Bishop, *Modern Control Systems*, New Jersey: Prentice Hall, 10 ed., 2005.

[67] P. Dransfield and D. F. Haber, *Introducing Root Locus*, Massachusetts: Cambridge University Press, 1973.

[68] P. Dutilleux, "Simple to operate digital time varying filters," *Audio Engineering Society Convention*, Preprint 2757, Mar. 1899.

[69] P. Dutilleux, *Vers la Machine à Sculpter le Son Modification en Temps Réel des Caractéristiques Fréquentielles et Temporelles des Sons*, PhD thesis, Université d'Aix-Marseille II, Institut de Mécanique de Marseille, May 1991.

[70] S. C. Dutta Roy and S. S. Ahuja, "Frequency transformations for linear phase variable-cutoff digital filters," *IEEE Transactions on Circuits and Systems*, vol. CAS-26, no. 1, pp. 73–75, 1979.

[71] S. A. V. Duyne, "Coupled mode synthesis," in *Proceedings of the 1997 International Computer Music Conference, Greece*, Computer Music Association, Oct. 1997.

[72] G. W. Evans, "Bringing root locus to the classroom," *IEEE Control Systems Magazine*, vol. 24, Dec. 2004.

[73] W. R. Evans, "Graphical analysis of control systems," *Transactions of the AIEE*, vol. 67, pp. 547–551, 1948.

[74] W. R. Evans, "Control system synthesis by root locus method," *Transactions of the AIEE*, vol. 69, 1950.

[75] W. R. Evans, *Control-System Dynamics*, New York: McGraw-Hill, 1954.

[76] A. M. Eydgahi and M. Ghavamzadeh, "Complementary root locus revisited," *IEEE Transactions on Education*, vol. 44, pp. 137–143, May 2001.

[77] C. W. Farrow, "A continuously variable digital delay element," in *Proceedings of the IEEE Conference on Circuits and Systems, Espoo, Finland*, vol. 3, pp. 2641–2645, June 7–9 1988.

[78] A. Fettweis, "Digital filters related to classical structures," *AEU: Archive für Elektronik und Übertragungstechnik*, vol. 25, pp. 79–89, Feb 1971, see also U.S. Patent 3,967,099, 1976, now expired.

[79] A. Fettweis, "Some principles of designing digital filters imitating classical filter structures," *IEEE Transactions on Circuit Theory*, vol. 18, pp. 314–316, Mar. 1971.

[80] A. Fettweis, "Pseudopassivity, sensitivity, and stability of wave digital filters," *IEEE Transactions on Circuit Theory*, vol. 19, pp. 668–673, Nov. 1972.

[81] A. Fettweis, "Wave digital filters: Theory and practice," *Proceedings of the IEEE*, vol. 74, pp. 270–327, Feb. 1986.

[82] A. Feuer and G. C. Goodwin, *Sampling in Digital Signal Processing and Control*, Systems and Control: Foundations and Applications, Boston: Birkhäuser, 1996.

[83] I. M. Filanovsky, "The root loci equation and its application," *International Journal of Electrical Engineering Education*, vol. 29, no. 2, pp. 133–138, 1992.

[84] J. D. Foley, A. van Dam, *et al.*, *Computer Graphics: Principles and Practice*, Reading MA: Addison-Wesley, 2 ed., 1995.

[85] Y. K. Foo and Y. C. Soh, "Characterization of zero locations of polytopes of real polynomials," *IEEE Transactions on Automatic Control*, vol. 37, no. 8, pp. 1227–1230, 1992.

[86] G. F. Franklin and J. D. Powell, *Digital Control of Dynamic Systems*, Reading MA: Addison-Wesley, 1980.

[87] G. F. Franklin, J. D. Powell, and M. L. Workman, *Digital Control of Dynamic Systems, Third Edition*, Englewood Cliffs, NJ: Prentice-Hall, 1998.

[88] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, New Jersey: Prentice Hall, 4 ed., 2002.

[89] D. R. Frey, "Low-cost alternatives in high-quality state-variable filters," *Journal of the Audio Engineering Society*, vol. 27, pp. 750–756, Oct. 1979.

[90] J. Glaría, R. Rojas, and M. Salgado, "Logarithmic root loci for continuous-time loops," *IEEE Control Systems*, pp. 47–52, Apr. 1994.

[91] A. S. Glassner, *An Introduction to Ray Tracing*, The Morgan Kaufmann Series in Computer Graphics, Morgan Kaufmann, 1989.

[92] T. E. Goeddel and S. C. Bass, "High quality synthesis of musical voices in discrete time," *IEEE Transactions on Acoustics, Speech, Signal Processing*, vol. 32, pp. 623–633, June 1984.

[93] B. Gold and C. M. Rader, *Digital Processing of Signals*, New Yotrk: McGraw Hill, 1969.

[94] R. M. Goodall and B. J. Donoghue, "Very high sample rate digital filters using the δ operator," *Proceedings of the IEE G*, pp. 199–206, June 1993.

[95] G. C. Goodwin, R. H. Middleton, and H. V. Poor, "High-speed digital signal processing," *Proceedings of the IEEE*, pp. 240–259, June 1993.

[96] M. Goodwin and A. Kogon, "Overlap-add synthesis of nonstationary sinusoids," in *Proceedings of the 1995 International Computer Music Conference, Banff*, Computer Music Association, 1995.

[97] A. H. Gray and J. D. Markel, "Digital lattice and ladder filter synthesis," *IEEE Transactions on Audio and Electroacoustics*, vol. AU-21, pp. 491–500, Dec. 1973.

[98] A. H. Gray and J. D. Markel, "A normalized digital filter structure," *IEEE Transactions on Acoustics, Speech, Signal Processing*, vol. ASSP-23, pp. 268–277, June 1975.

[99] J. J. Gribble, "Modified root locus plots for SISO systems with time delay," *IEEE Control Systems*, pp. 54–56, Feb. 1993.

[100] P. Griffiths and J. Harris, *Principles of Algebraic Geometry*, Wiley Classics Library, New York: John Wiley and Sons, Inc., 1994, Reprint, 1st ed. 1978.

[101] T. Gunji, S. Kim, M. Kojima, *et al.*, "PHoM — a polyhedral homotopy continuation method for polynomial systems," tech. rep., Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, Dec. 2002.

[102] H. Hahn, "Higher-order root-locus technique with applications in control system design," in *Advances in Control Systems and Signal Processing, Vol. 2* (I. Hartmann, ed.), Braunschweig, Weisbaden: Vieweg, 1981.

[103] C. Hanna, "Real-time control of DSP parametric equalizers," *Audio Engineering Society Convention*, Nov. 1994.

[104] F. Harris and E. Brooking, "A versatile parametric filter using imbedded all-pass subfilter to independently adjust bandwidth, center frequency,and boost or cut," *Audio Engineering Society Convention*, Preprint 3757, Oct. 1993.

[105] J. Harris, *Algebraic Geometry*, Graduate Texts in Mathematics, New York: Springer Verlag, 1992.

[106] A. Helzer, M. Barzohar, and D. Malah, "Stable fitting of 2d curves and 3d surfaces by implicit polynomials," *IEEE Transactions on Pattern Analysis and Machine Intellignece*, vol. 26, pp. 1283–1294, Oct. 2004.

[107] D. T. Horn, *The Beginner's Book of Electronic Music*, Blue Ridge Summit, PA: Tab Books, 1982.

[108] D. T. Horn, *Music Synthesizers: A Manual of Design and Construction*, Blue Ridge Summit, PA: Tab Books, 1984.

[109] T. Hundley, H. Benioff, and D. W. Martin, "Factors contributing to the multiple rate of piano tone decay," *Journal of the Acoustical Society of America*, vol. 64, pp. 1303–1309, Nov. 1978.

[110] A. Huovilainen, "Nonlinear digital implementation of the Moog ladder filter," in *Proceedings of the Conference on Digital Audio Effects (DAFx-04), Naples, Italy*, pp. 61–64, 2004, `http://-dafx04.na.infn.it/`.

[111] A. Huovilainen and V. Välimäki, "New approaches to digital subtractive synthesis," in *Proceedings of the 2005 International Computer Music Conference, Barcelona, Spain*, Computer Music Association, 2005.

[112] A. Hurwitz, "On the conditions under which an equation has only roots with negative real parts," in *Selected Papers on Mathematical Trends in Control Theory* (I. Hartmann, ed.), pp. 70–82, New York: Dover, 1964, Originally Published in Mathematische Annalen 46, 1895, pp. 273–284.

[113] B. Hutchins, "Additional design ideas for voltage-controlled filters," *Electronotes*, vol. 85, pp. 5–17.

[114] B. Hutchins, "The effect of feedback on four-pole filters with differing pole frequencies," *Electronotes*, vol. 105, pp. 3–12.

[115] B. Hutchins, "A few more notes on polygon filters," *Electronotes*, vol. 97, pp. 9–10.

[116] B. Hutchins, "A few more notes on polygon filters," *Electronotes*, vol. 97, pp. 10–17.

[117] B. Hutchins, "The migration of poles as a function of feedback in a class of voltage-controlled filters," *Electronotes*, vol. 95, pp. 3–15.

[118] B. Hutchins, "A four-pole voltage-controlled network; analysis, design, and application as a low-pass filter and a quadrature VCO," *Electronotes*, vol. 41, pp. 1–7, July 1974, Reprinted in [119, ch. 5d].

[119] B. Hutchins, *Musical Engineer's Handbook*, Ithaca, New York: Electronotes, 1975.

[120] B. Hutchins, "A four-pole variable digital filter with fixed clock rate," *Electronotes*, vol. 15F, pp. 70–79, 1984.

[121] B. Hutchins, "On the implementation of variable digital filters with fixed clock rate," *Electronotes*, vol. 15E, pp. 18–34, 1984.

[122] C. Hwang and J.-J. Chen, "Plotting robust root loci for linear systems with multilinearly parametric uncertainties," in *Proceedings of the American Control Conference, Philadelphia, Pennsylvania*, pp. 1958–1962, June 1998.

[123] C. Hwang and S.-F. Yang, "The robust root locus of polynomial families with multilinear parameter dependence," in *Proceedings of the 2001 IEEE International Conference on Control Applications*, pp. 847–852, Sept. 2001.

[124] M. Ikehara, S. ichi Takahashi, and M. Kato, "Construction of variable cuttof digital filter based on changeover of taps," *Electronics and Communications in Japan, Part 3*, vol. 72, no. 12, pp. 13–21, 1989.

[125] L. B. Jackson, "Roundoff-noise analysis for fixed-point digital filters realized in cascade or parallel form," *IEEE Transactions on Audio and Acoustics*, vol. AU-18, pp. 107–122, June 1970.

[126] L. B. Jackson, "A correction to impulse invariance," *IEEE Signal Processing Letters*, vol. 7, pp. 273–275, Oct. 2000.

[127] L. B. Jackson, "A narrowband IIR digital filter with low sensitivity an roundoff noise," *IEEE Signal Processing Letters*, vol. 10, pp. 164–167, June 2003.

[128] D. A. Jaffe and J. O. Smith, "Extensions of the Karplus-Strong plucked string algorithm," *Computer Music Journal*, vol. 7, no. 2, pp. 56–69, 1983, Reprinted in [220, pp. 481–494].

[129] D. A. Jaffe and J. O. Smith, "Performance expression in commuted waveguide synthesis of bowed strings," in *Proceedings of the 1995 International Computer Music Conference, Banff*, pp. 343–346, Computer Music Association, 1995.

[130] P. Jarske, Y. Neuvo, and S. K. Mitra, "A simple approach to the design of linear-phase FIR digital filters with variable characteristics," *Signal Processing*, vol. 14, pp. 313–326, 1988.

[131] H. Johansson and P. Löwenborg, "On the design of adjustable fractional delay FIR filters," *IEEE Transactions on Circuits and Systems—II: Analog and Digital Signal Processing*, vol. 50, pp. 164–169, Apr. 2003.

[132] D. H. Johnson, "Variable digital filters having a recursive stucture," *IEEE Transactions on Acoustics, Speech, Signal Processing*, vol. 27, Feb. 1979.

[133] Kaegi, W., and S. Tempelaars, "VOSIM—a new sound synthesis system," *Journal of the Audio Engineering Society*, vol. 26, no. 6, pp. 418–24, 1978.

[134] H. K. Kahlil, *Nonlinear Systems*, New Jersey: Prentice Hall, 2 ed., 1996.

[135] T. Kailath, *Linear Systems*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1980.

[136] R. Kalaba and L. Tesfation, "Solving nonlinear equations by adaptive homotopy continuation," *Applied Mathematics and Computation*, vol. 41, pp. 99–115, 1991.

[137] M. Karjalainen, A. Harma, U. K. Laine, and J. Huopaniemi, "Warped filters and their audio applications," in *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, New Paltz, NY*, (New York), IEEE Press, Oct. 1997, Session 11, paper 2, 4 pages.

[138] M. Karjalainen, E. Piirilä, A. Jävinen, and J. Huopaniemi, "Comparison of loudspeaker equalization methods based on dsp techniques," *Journal of the Audio Engineering Society*, vol. 49, pp. 14–31, Jan. 1999.

[139] R. B. Kearfott and Z. Xing, "An interval step control for continuation methods," *SIAM Journal of Numerical Analysis*, vol. 31, pp. 892–914, 1994.

[140] W. J. Kerwin, L. P. Huelsman, and R. W. Newcomb, "State-variable synthesis for insensitive integrated-circuit transfer functions," *IEEE Journal of the Solid-State Circuits*, vol. SC-2, pp. 87–92, Sept. 1967.

[141] S. Kim and M. Kojima, "Numerical stability of path tracing in polyhedral homotopy continuation methods," tech. rep., Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, Mar. 2003.

[142] F. Kirwan, *Complex Algebraic Curves*, London Mathematical Society Student Texts 23, Cambridge University Press, 1992.

[143] P. Kleczkowski, "Group additive synthesis," *Computer Music Journal*, vol. 13, no. 1, pp. 12–20, 1989.

[144] J. Konopacki, "The frequency transformation by matrix operation and its application in IIR filters design," *IEEE Signal Processing Letters*, vol. 12, pp. 5–8, Jan. 2005.

[145] V. Krishnamurthy and M. Levoy, "Fitting smooth surfaces to dense polygon meshes," in *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics*, pp. 313–324, Aug. 1996.

[146] V. Krishnan, "Semi-analytic approach to root locus," *IEEE Transactions on Automatic Control*, vol. AC-11, pp. 326–332, Jan. 1966.

[147] T. R. Kurfess and M. L. Nagurka, "Understanding root locus using gain plots," *IEEE Control Systems*, pp. 37–40, Aug. 1991.

[148] T. R. Kurfess and M. L. Nagurka, "Geometric links among classical controls tools," *IEEE Transactions on Education*, pp. 77–83, Feb. 1994.

[149] T. I. Laakso, V. Välimäki, M. Karjalainen, and U. K. Laine, "Splitting the Unit Delay—Tools for Fractional Delay Filter Design," *IEEE Signal Processing Magazine*, vol. 13, pp. 30–60, Jan. 1996.

[150] J. Lane, D. Hoory, E. Martinez, and P. Wang, "Modeling analog synthesis with DSPs," *Computer Music Journal*, vol. 21, pp. 23–41, Winter 1997.

[151] J. Laroche, "Using resonant filters for the synthesis of time-varying sinusoids," *Audio Engineering Society Convention*, Preprint 4782, Sept. 1998.

[152] J. Laroche, "A modified lattice structure with pleasing scaling properties," *IEEE Transactions on Signal Processing*, vol. 64, pp. 3423–3425, Dec. 1999.

[153] J. Laroche, "Synthesis of sinusoids via non-overlapping inverse fourier transform," *IEEE Transactions on Speech and Audio Processing*, vol. 8, pp. 471–477, July 2000.

[154] J. Laroche and M. Dolson, "New phase-vocoder techniques for pitch-shifting, harmonizing, and other exotic effects," in *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, New Paltz, NY*, (New York), pp. 91–94, IEEE Press, Oct. 17–20, 1999.

[155] Z. Lei and D. B. Cooper, "New, faster, more controlled fitting of implicit polynomial 2D curves and 3D surfaces to data," *ACM Transactions on Graphics*, vol. 12, pp. 327–347, Oct. 1993.

[156] E. Lindemann, "Music synthesis with reconstructive phrase modeling," *IEEE Signal Processing Magazine*, submitted for publication.

[157] B. Liu, "Effect of finite word lenth on the accuracy of digital filters — a review," *IEEE Transactions on Circuit Theory*, vol. CT-18, pp. 670–677, Nov. 1971.

[158] W. Lorenson and H. Cline, "Marching cubes: A high-resolution 3D surface construction algorithm," *Computer Graphics*, vol. 21, pp. 163–169, 1987.

[159] W.-S. Lu and T.-B. Deng, "An improved weighted least-squares design for variable fractional delay FIR filters," *IEEE Transactions on Circuits and Systems—II: Analog and Digital Signal Processing*, vol. 49, pp. 1035–1040, Aug. 1999.

[160] K. H. Lundberg, "Pole-zero phase maps," *IEEE Control Systems Magazine*, pp. 84–87, Feb. 2005.

[161] M. Makundi, T. Laakso, and V. Valimaki, "Efficient tunable IIR and allpass structures," *Electronics Letters*, vol. 37, pp. 344–345, Mar. 2001.

[162] D. C. Massie, "An engineering study of the four-multiply normalized ladder filter," *Journal of the Audio Engineering Society*, vol. 41, pp. 564–582, July 1993.

[163] M. V. Mathews, *The Technology of Computer Music*, Cambridge, MA: MIT Press, 1969.

[164] R. J. Y. McLeod and M. L. Baart, *Geometry and Interpolation of Curves and Surfaces*, Cambridge University Press, 1998.

[165] M. McNabb, "Clarinet," 1994, Unpublished clarinet model.

[166] W. F. G. Mecklenbrauker, "Remarks on and correction to the impulse invariant method for the design of IIR digital filters," *Signal Processing*, vol. 80, pp. 1687–1690, Aug. 2000.

[167] R. H. Middleton and G. C. Goodwin, "Improved finite word length characteristics in digital control using delta operators," *IEEE Transactions on Automatic Control*, pp. 1015–1021, Nov. 1986.

[168] R. H. Middleton and G. C. Goodwin, *Digital Control and Estimation: A Unified Approach*, New Jersey: Prentice Hall, 1990.

[169] J. R. Mitchell and W. L. McDaniel, Jr., "A generalized root locus following technique," *IEEE Transactions on Automatic Control*, pp. 483–486, Aug. 1970.

[170] S. K. Mitra, *Digital Signal Processing: A Computer-Based Approach*, New York: McGraw Hill, 3 ed., 2006.

[171] S. K. Mitra, Y. Neuvo, and H. Roivainen, "Design of recursive digital filters with variable characteristics," *International Journal of Circuit Theory and Applications*, vol. 18, pp. 107–119, 1990.

[172] R. A. Moog, "Voltage-controlled electronic music modules," *Journal of the Audio Engineering Society*, vol. 13, pp. 200–206, July 1965, see also [173].

[173] R. A. Moog, "A voltage-controlled low-pass high-pass filter for audio signal processing," *Audio Engineering Society Convention*, Preprint 413, Oct. 1965.

[174] F. R. Moore, "Table lookup noise for sinusoidal digital oscillators," in *Foundations of Computer Music* (C. Roads and J. Strawn, eds.), pp. 326–334, Cambridge, MA: MIT Press, 1985.

[175] J. A. Moorer, "The synthesis of complex audio spectra by means of discrete summation formulae," *Journal of the Audio Engineering Society*, vol. 24, pp. 717–727, Dec. 1975, Also svailable as CCRMA Report no. STAN-M-5.

[176] J. A. Moorer, "The manifold joys of conformal mapping: Applications to digital filtering in the studio," *Journal of the Audio Engineering Society*, vol. 31, pp. 826–841, Nov 1983.

[177] A. Morgan, *Solving Polynomial Systems Using Continuation for Engineering and Scientific Computations*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1987.

[178] A. Morgan and A. Sommese, "Computing all solutions to polynomial systems using homotopy continuation," *Applied Mathematics and Computation*, vol. 24, pp. 115–138, 1987.

[179] A. Morgan and A. Sommese, "A homotopy for solving general polynomial systems that respects m-homogeneous structures," *Applied Mathematics and Computation*, vol. 24, pp. 101–113, 1987.

[180] P. M. Morse, *Vibration and Sound*, American Institute of Physics, for the Acoustical Society of America, 1948, 1st edition 1936, last author's edition 1948, ASA edition 1981.

[181] J. N. Mourjopoulos, E. D. Kyriakis-Bitzaros, and C. E. Goutis, "Theory and real-time implementation of time-varying digital audio filters," *Journal of the Audio Engineering Society*, vol. 38, pp. 523–536, July 1990.

[182] J. N. Mourjopoulos, "Digital equalization of room acoustics," *Journal of the Audio Engineering Society*, vol. 42, pp. 864–900, Nov. 1994.

[183] C. T. Mullis and R. A. Roberts, "Roundoff noise in digital filters: Frequency transformations and invariants," *IEEE Transactions on Acoustics, Speech, Signal Processing*, vol. ASSP-24, Dec. 1976.

[184] D. C. Munson, Jr. and B. Liu, "Low-noise realizations for narrow-band recursive digital filters," *IEEE Transactions on Acoustics, Speech, Signal Processing*, vol. 28, Feb. 1980.

[185] N. Murakoshi, E. Watanabe, and A. Nishihara, "A synthesis of variable IIR digital filters," *IEICE Transactions on Fundamentals*, vol. E75-A, pp. 362–368, Mar. 1992.

[186] N. Murakoshi, A. Nishihara, and E. Watanabe, "Synthesis of variable IIR digital filters with complex coefficients," *Electronics and Communications in Japan, Part 3*, vol. 77, no. 5, pp. 46–56, 1994.

[187] I. Nakamura, "Fundamental theory and computer simulation of the decay characteristics of piano sound," *Journal of the Acoustical Society of Japan (E)*, vol. 10, no. 5, pp. 289–297, 1989.

[188] A. H. Nayfeh and B. Balachandran, *Applied Nonlinear Dynamics*, Wiley Series in Nonlinear Science, New York: John Wiley and Sons, Inc., 1995.

[189] Z. Nehari, *Conformal Mapping*, New York: Dover, 1952.

[190] C. P. Neuman, "The two-pole two-zero root locus," *IEEE Transactions on Education*, pp. 369–370, Nov. 1994.

[191] K. Nishioka, N. Adachi, and K. Takeuchi, "Simple pivoting algorithm for root-locus method of linear systems with delay," *IEEE Transactions on Automatic Control*, pp. 576–582, Oct. 1968.

[192] J. R. O'Donnell and D. Frederick, "Extensions to the Ash algorithm for finding root loci," in *Proceedings of the 1987 American Control Conference, Minneapolis, MN*, vol. 3, pp. 1943–1944, June 1987.

[193] J. R. O'Donnell and D. Frederick, "MATLAB implementation of the extended Ash algorithm for finding root loci," in *Computer Aided Design in Control Systems. Selected Papers from the IFAC Symposium*, pp. 341–346, July 1991.

[194] H. F. Olson and H. Belar, "Electronic music synthesizer," *Journal of the Acoustical Society of America*, vol. 27, pp. 595–612, May 1955.

[195] A. V. Oppenheim, *Discrete-Time Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1989.

[196] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1989.

[197] A. V. Oppenheim, W. F. G. Mecklenbräuker, and R. M. Mersereau, "Variable-cutoff linear-phase digital filters," *IEEE Transactions on Circuits and Systems*, vol. CAS-23, no. 4, pp. 199–203, 1976.

[198] S. J. Orfanidis, "Digital parametric equalizer design with prescribed Nyquist-frequency gain," *Journal of the Audio Engineering Society*, vol. 45, pp. 444–455, June 1997.

[199] C. T. Pan and K. S. Chao, "A computer-aided root-locus method," *IEEE Transactions on Automatic Control*, vol. AC-23, pp. 876–860, Oct. 1978.

[200] A. Papoulis, *Signal Analysis*, New York: McGraw-Hill, 1977.

[201] T. W. Parks and J. H. McClellan, "Chebyshev approximation for nonrecursive digital filters with linear phase," *IEEE Transactions on Circuit Theory*, pp. 189–194, Mar. 1972.

[202] D. K. Phillips, "Multirate additive synthesis," *Computer Music Journal*, vol. 23, no. 1, pp. 28–40, 1999.

[203] D. A. Pierre, "Robustness issues concerning dipole and doublet root loci," in *Proceedings of the 1993 American Control Conference*, vol. 3, pp. 2750–2754, June 1993.

[204] T. Pinch and F. Trocco, *Analog Days: The Invention and Impact of the Moog Synthesizer*, Cambridge, MA: Harvard University Press, 2002.

[205] H. Power, "Application of bilinear transformation to root locus plotting," *IEEE Transactions on Automatic Control*, pp. 693–694, Dec. 1970.

[206] M. Prendergast, *Ambient Century: From Mahler to Trance — The Evolution of Sound in the Electronic Age*, New York: Bloomsbury, 2000.

[207] O. Prokhorova and I. M. Filanovsky, "Multivariable system design and optimization using root locus method," in *33rd Midwest Symposium on Circuits and Systems, Calgary, Alberta*, vol. 1, pp. 523–526, Aug. 1991.

[208] F. D. Pryashnikov, "Distribution of the roots of a characteristic polyonomival with respect to a polygonal domain," *Tekh. Kibern. (Russia)*, vol. 6, pp. 67–75, 1991.

[209] B. Pšenička and F. Garcia-Ugalde, "z transform from lowpass to bandpass by Pascal matrix," *IEEE Signal Processing Letters*, vol. 11, pp. 282–284, Feb. 2004.

[210] B. Pšenička, F. Garcia-Ugalde, and A. Herrera-Camacho, "The bilinear z transform by Pascal matrix and its application in the design of digital filters," *IEEE Signal Processing Letters*, vol. 9, pp. 368–370, Nov. 2002.

[211] C. K. S. Pun, S. C. Chan, K. S. Yeung, and K. L. Ho, "On the design and implementation of FIR and IIR digital filters with variable frequency characteristics," *IEEE Transactions on Circuits and Systems—II: Analog and Digital Signal Processing*, vol. 49, pp. 689–703, Nov. 2002.

[212] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1975.

[213] G. Ramos and J. J. Lopez, "Low order IIR parametric loudspeaker equalization, a psychoacoustic approach," *Audio Engineering Society Convention*, Preprint 6454, May 2005.

[214] J. Rauhala and V. Välimäki, "Tunable dispersion filter design for piano synthesis," *IEEE Transactions on Acoustics, Speech, Signal Processing*, vol. 34, pp. 1557–1564, Nov. 1987.

[215] N. J. Redding, "Fitting implicit polynomials to use as features in image understanding," in *Proceedings of the 1996 Australian New Zealand Conference on Intelligent Information Systems, Adelaide, Australia*, Nov. 1996.

[216] N. J. Redding and G. N. Newsam, "A merging procedure for connecting fitted implicit polynomials for features," in *Proceedings of the 1996 Australian New Zealand Conference on Intelligent Information Systems, Adelaide, Australia*, Nov. 1996.

[217] P. A. Regalia and S. K. Mitra, "Tunable digital frequency response equalization filters," *IEEE Transactions on Acoustics, Speech, Signal Processing*, vol. ASSP-35, p. 118, 1987.

[218] C. M. Richter and C. P. Bottura, "Riemann k-surfaces in multivariable control systems," in *Proceedings of the 1999 American Control Conference, San Diego, CA*, vol. 4, pp. 2869–2870, June 1999.

[219] J. H. F. Ritzerfeld, "Noise gain expressions for low noise second-order digital filter structures," *IEEE Transactions on Circuits and Systems—II: Analog and Digital Signal Processing*, vol. 12, pp. 5–8, Jan. 2005.

[220] C. Roads, ed., *The Music Machine*, Cambridge, MA: MIT Press, 1989.

[221] C. Roads, *The Computer Music Tutorial*, Cambridge, MA: MIT Press, 1996.

[222] C. Roads, *Microsound*, Cambridge, MA: MIT Press, 2001.

[223] C. Roads and J. Strawn, eds., *Foundations of Computer Music*, Cambridge, MA: MIT Press, 1985.

[224] X. Rodet and P. Depalle, "Spectral envelopes and inverse FFT synthesis," *Proc. 93rd Convention of the Audio Engineering Society, San Francisco*, 1992, Preprint 3393 (H-3).

[225] X. Rodet, Y. Potard, and J. Barrière, "The CHANT Project: From the synthesis of the singing voice to synthesis in general," in *The Music Machine* (C. Roads, ed.), pp. 449–465, Cambridge, MA: MIT Press, 1989.

[226] E. J. Routh, *Dynamics of a System of Rigid Bodies*, New York: MacMillan, 1892.

[227] R. W. Schafer and L. R. Rabiner, "A digital signal processing approach to interpolation," *Proceedings of the IEEE*, vol. 61, pp. 692–702, June 1973.

[228] T. Schanze, "Sinc interpolation of discrete periodic signals," *IEEE Transactions on Signal Processing*, vol. 43, pp. 1502–1503, June 1995.

[229] W. Schüssler and W. Winkelkemper, "Variable digital filters," *Arch. Elek. Übertragung*, vol. 24, pp. 524–525, 1970.

[230] I. R. Shafarevich, *Basic Algebraic Geometry I*, New York: Springer Verlag, 2 ed., 1994.

[231] I. R. Shafarevich, *Basic Algebraic Geometry II*, New York: Springer Verlag, 2 ed., 1997.

[232] P. Shapiro, *Modulations: A History of Electronic Music — Throbbing Words on Sound*, New York: Caipirinha Productions, 2000.

[233] J.-J. E. Slotine and W. Li, *Applied Nonlinear Control*, New Jersey: Prentice Hall, 1991.

[234] J. O. Smith, *Techniques for Digital Filter Design and System Identification with Application to the Violin*, PhD thesis, Elec. Engineering Dept., Stanford University (CCRMA), June 1983, CCRMA Technical Report STAN-M-14, `http://ccrma.stanford.edu/STANM/stanms/stanm14/`.

[235] J. O. Smith, "Music applications of digital waveguides," Tech. Rep. STAN-M-39, CCRMA, Music Department, Stanford University, 1987, CCRMA Technical Report STAN-M-39, `http://ccrma.stanford.edu/STANM/stanms/stanm39/`.

[236] J. O. Smith, "Physical modeling using digital waveguides," *Computer Music Journal*, vol. 16, pp. 74–91, Winter 1992, special issue: Physical Modeling of Musical Instruments, Part I. `http://ccrma.stanford.edu/~jos/pmudw/`.

[237] J. O. Smith, "Efficient synthesis of stringed musical instruments," in *Proceedings of the 1993 International Computer Music Conference, Tokyo*, pp. 64–71, Computer Music Association, 1993, incorporated into [240].

[238] J. O. Smith, *Introduction to Digital Filters*, `http://ccrma.stanford.edu/~jos/filters05/`, September 2005.

[239] J. O. Smith, *Physical Audio Signal Processing — for Virtual Musical Instruments and Digital Audio Effects*, `http://ccrma.stanford.edu/~jos/pasp05/`, Dec. 2005.

[240] J. O. Smith, *Physical Audio Signal Processing: Digital Waveguide Modeling of Musical Instruments and Audio Effects*, `http://ccrma.stanford.edu/~jos/pasp/`, Dec. 2005.

[241] J. O. Smith and P. R. Cook, "The second-order digital waveguide oscillator," in *Proceedings of the 1992 International Computer Music Conference, San Jose*, pp. 150–153, Computer Music Association, 1992, `http://ccrma.stanford.edu/~jos/wgo/`.

[242] J. O. Smith and P. Gossett, "A flexible sampling-rate conversion method," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, San Diego*, vol. 2, (New York), pp. 19.4.1–19.4.2, IEEE Press, Mar. 1984, expanded tutorial and associated free software available at the Digital Audio Resampling Home Page: `http://-ccrma.stanford.edu/~jos/resample/`.

[243] D. L. Spencer, L. Philipp, and B. Philipp, "Root loci design using Dickson's technique," *IEEE Transactions on Education*, vol. 44, pp. 176–184, May 2001.

[244] K. Steiglitz, "An analytical approach to root loci," *IRE Transactions on Automatic Control*, pp. 326–332, Sept. 1961.

[245] K. Steiglitz, "A note on variable recursive digital filters," *IEEE Transactions on Acoustics, Speech, Signal Processing*, vol. ASSP-28, pp. 111–112, Feb. 1980.

[246] T. Stilson, "Use of a simple control system for the improvement of simplified physical-modeling synthesis algorithms," 1996, Presented at the 1996 Acoustical Society of America Conference, Waikiki, Hawaii. Annotated viewgraphs are available at`http://-ccrma.stanford.edu/~stilti/`.

[247] T. Stilson, "Applying root-locus techniques to the analysis of coupled modes in piano strings," in *Proceedings of the 1997 International Computer Music Conference, Greece*, pp. 462–464, 1997, A version of this paper was also presented at the Acoustical Society of America Conferece, December 1996, Hawaii. An expanded version of the paper can be found at `http://ccrma.stanford.edu/~stilti/`.

[248] T. Stilson and J. O. Smith, "Alias-free synthesis of classic analog waveforms," in *Proceedings of the 1996 International Computer Music Conference, Hong Kong*, Computer Music Association, 1996, `http://ccrma.stanford.edu/~stilti/`.

[249] T. Stilson and J. O. Smith, "Analyzing the Moog VCF with considerations for digital implementation," in *Proceedings of the 1996 International Computer Music Conference, Hong Kong*, Computer Music Association, 1996, `http://ccrma.stanford.edu/~stilti/`.

[250] T. Stilson and J. O. Smith, "Virtual analog," May 1997, Lecture Overheads for Music 421, Spring 1997, `http://ccrma.stanford.edu/~stilti/`.

[251] T. Stilson and H. Thornburg, "Examples of using amplitude control systems in music synthesis," in *Proceedings of the 1998 International Computer Music Conference, Michigan*, Computer Music Association, 1998.

[252] V. L. Stonick and S. T. Alexander, "A relationship between the recursive least-squares update and homotopy continuations methods," *IEEE Transactions on Signal Processing*, vol. 39, pp. 530–532, Feb. 1991.

[253] V. L. Stonick and S. T. Alexander, "Globally optimal rational approximation using homotopy continuation methods," *IEEE Transactions on Signal Processing*, vol. 49, pp. 2358–2361, Sept. 1992.

[254] G. Strang, *Linear Algebra and Its Applications*, Orlando: Harcourt Brace Jovanovich, Inc., 3 ed., 1988.

[255] I. H. Suh and Z. Bien, "A root-locus technique for linear systems with delay," *IEEE Transactions on Automatic Control*, vol. AC-27, pp. 205–208, Feb. 1982.

[256] M. N. S. Swamy and K. S. Thyagarajan, "Digital bandpass and bandstop filters with variable center frequency and bandwidth," *Proceedings of the IEEE*, vol. 47, pp. 1642–1634, Nov. 1976.

[257] A. Tarczyński, G. D. Cain, E. Hermanowicz, and M. Rojewski, "WLS design of variable frequency response FIR filters," in *IEEE International Symposium on Circuits and Systems, Hong Kong*, 1997.

[258] G. Taubin, "Constrained implicit function fitting," in *ICPR '92: International Conference on Pattern Recognition, The Hague, Holland*, Sept. 1992.

[259] G. Taubin, "An improved algorithm for algebraic curve and surface fitting," in *ICCV '93: International Conference on Computer Vision, Berlin, Germany*, May 1993.

[260] G. Taubin, "Distance approximations for rasterizing implicit curves," *ACM Transactions on Graphics*, vol. 13, pp. 3–42, Jan. 1994.

[261] G. Taubin, "Estimation of planar cuvrves, surfaces, and nonplanar space curves defined by implicit equations with applications to edge and range image segmenataion," *IEEE Transactions on Pattern Analysis and Machine Intellignece*, vol. 13, pp. 14–23, Mar. 1994.

[262] G. Taubin, "Rasterizing algrebraic curves and surfaces," *IEEE Computer Graphics and Applications*, vol. 13, pp. 14–23, Mar. 1994.

[263] G. Taubin, F. Cukierman, S. Sullivan, J. Ponce, and D. J. Kriegman, "Parameterized families of polynomials for bounded algebraic curve and surface fitting," *IEEE Transactions on Pattern Analysis and Machine Intellignece*, vol. 16, pp. 287–303, Mar. 1994.

[264] P. A. Taylor, A. Black, and R. Caley, "The architecture of the Festival speech synthesis system," in *The Third ESCA Workshop oin Speech Synthesis, Jenolan Caves, Australia*, pp. 147–151, 1998.

[265] M. C. M. Teixeira, E. Assunção, and E. R. M. D. Machado, "A method for plotting the complementary root locus using the root-locus (positive gain) rules," *IEEE Transactions on Education*, vol. 47, pp. 405–409, Aug. 2004.

[266] S. W. Tento, "The implicit function theorem and robust root locus," in *Proceedings of the American Control Conference, 1991*, vol. 1, pp. 869–870, 1991.

[267] J.-P. Thiran, "Recursive digital filters with maximally flat group delay," *IEEE Transactions on Circuits and Systems*, vol. 18, pp. 659–664, Nov. 1971.

[268] H. Thornburg, "Digital Moog filter distortion experiments." CCRMA DSP Seminar Presentation, 1997.

[269] Y. Tong and N. K. Sinha, "A computational technique for the robust root locus," *IEEE Transactions on Industrial Electronics*, vol. 41, pp. 79–85, Feb. 1994.

[270] C. Tseng, "Digital integrator design using Simpson rule and fractional delay filter," *IEE Journal of the Vis. Image Signal Processing*, pp. 189–194, Mar. 1972.

[271] P. Tsiotras, "The relation between the 3-D Bode diagram and the root locus," *IEEE Control Systems Magazine*, pp. 88–96, Feb. 2005.

[272] J. Tupper, "Graphing equations with generalized interval arithmetic," Master's thesis, Graduate Department of Computer Science, University of Toronto, 1996.

[273] C. W. Ueberhuber, *Numerical Computation: Methods, Software and Analysis*, vol. 2, New York: Springer Verlag, 1997.

[274] V. Välikmäki and T. Laakso, "Suppression of transients in variable recursive digital filters with a novel and efficient cancellation method," *IEEE Transactions on Signal Processing*, vol. 46, pp. 3408–3414, Dec. 1998.

[275] V. Välimäki, "Discrete-time synthesis of the sawtooth waveform with reduced aliasing," *IEEE Signal Processing Letters*, vol. 12, no. 3, pp. 214–217, 2005.

[276] V. Välimäki and A. Huovilainen, "Antialiasing oscillators in subtractive synthesis," *IEEE Signal Processing Magazine*, submitted for publication.

[277] V. Välimäki and A. Huovilainen, "Oscillator and filter algorithms for virtual analog synthesis," *Computer Music Journal*, pp. 21–33, 2006.

[278] V. Välimäki, T. I. Laakso, and J. Mackenzie, "Elimination of transients in time-varying allpass fractional delay filters with application to digital waveguide modeling," in *Proceedings of the 1995 International Computer Music Conference, Banff*, pp. 327–334, Computer Music Association, 1995.

[279] A. Vaněček and S. Čelikovský, "Chaos synthesis via root locus," *IEEE Transactions on Circuits and Systems—I: Fundamental Theory and Applications*, pp. 59–60, Jan. 199.

[280] A. Vaněček and S. Čelikovský, "Root locus of chaotic attractors," in *UKACC International Conference on CONTROL '96*, pp. 1220–1225, Sept. 1996.

[281] J. M. Varah, "Least squares data fitting with implicit functions," *BIT*, vol. 36, no. 4, pp. 842–854, 1996.

[282] Various, "Virtual analog page at musicdsp.org's wiki." `http://www.musicdsp.org/phpWiki/index.php/VirtualAnalog`.

[283] J. Vesma, "A frequency-domain approach to polynomial-based interpolation and the Farrow structure," *IEEE Transactions on Circuits and Systems—II: Analog and Digital Signal Processing*, vol. 47, Mar. 2000.

[284] J. Vesma and T. Saramäki, "Design and properties of polynomial-based fractional-delay filters," in *ISCAS 2000 — IEEE International Symposium on Circuis and Systems, Geneva, Switzerland*, (New York), pp. I–104 – 107, IEEE Press, May 2000.

[285] M. Vidyasagar, *Nonlinear Systems Analysis*, New Jersey: Prentice Hall, 2 ed., 1993.

[286] R. Vijayan, H. V. Poor, J. B. Moore, and G. C. Goodwin, "A Levinson-type algorithm for modeling fast-sampled data," *IEEE Transactions on Automatic Control*, pp. 314–321, Mar. 1991.

[287] A. Watt and M. Watt, *Advanced Animation and Rendering Techniques: Theory and Practice*, Reading MA: Addison-Wesley, 1992.

[288] G. Weinreich, "Coupled piano strings," *Journal of the Acoustical Society of America*, vol. 62, pp. 1474–1484, Dec 1977, see also [11] and *Scientific American*, vol. 240, p. 94, 1979.

[289] L. Williams, "Pyramidal parametrics," *Computer Graphics*, vol. 17, pp. 1–11, July 1983, (Proc. SIGGRAPH '83).

[290] S. E. Williamson and W. F. Lovering, "The root-loci of four pole systems," *International Journal of Control*, vol. 10, pp. 625–643, Dec. 1969.

[291] R. Wilson, "Filter topologies," *Journal of the Audio Engineering Society*, vol. 41, Sept. 1993.

[292] G. Winham and K. Steiglitz, "Input generators for digital sound synthesis," *Journal of the Acoustical Society of America*, vol. 47, no. 2, pp. 665–666, 1970.

[293] D. K. Wise, "The recursive allpass as a resonance filter," in *Proceedings of the 1998 International Computer Music Conference, Michigan*, Computer Music Association, 1998.

[294] G.-T. Yan and S. K. Mitra, "Modified coupled-form digital-filter structures," *Proceedings of the IEEE*, vol. 70, July 1982.

[295] C.-D. Yang and C.-H. Wei, "Root-locus dynamics," in *2005 American Control Conference, Portland, OR*, pp. 63–68, June 2005.

[296] C.-D. Yang and F.-B. Yeh, "On modeling root-locus behavior," in *Proceedings of the 33rd Conference on Decision and Control, Lake Buena Vista, Florida*, pp. 2151–2156, Dec. 1994.

[297] K. S. Yeung and W. T. Wong, "Root-locus plots of systems with time delay," *Electronics Letters*, pp. 480–481, May 1982.

[298] T. Yoshida, A. Nishihara, and N. FUjii, "A design method of variable FIR filters using multidimensional filters," *IEICE Transactions on Fundamentals*, vol. E75-A, pp. 964–971, Aug. 1992.

[299] R. Zarour and M. M. Fahmy, "A design techinique for variable digital filters," *IEEE Transactions on Circuits and Systems*, vol. 36, pp. 1473–1477, Nov. 1989.

[300] L. H. Zetterberg and Q. Zhang, "Elimination of transients in adaptive filters with application to speech coding," *Signal Processing*, vol. 15, no. 4, pp. 419–428, 1988.

[301] U. Zölzer and T. Boltze, "Parametric digital filter structures," *Audio Engineering Society Convention*, Preprint 4099, Oct. 1995.