The Filter Wizard
issue 12: Filter Design using the "Million Monkeys Method"
Kendall Castor-Perry

Previously on The Filter Wizard (Simulate Circuits in a Spreadsheet with some 'Ladderal Thinking'), we saw how to use a spreadsheet such as Excel to calculate the frequency response of a filter network with a ladder form. This form covers a wide range of common filter circuits, from loudspeaker crossover networks through to high performance communications filters.

Now we've not yet done anything you can't do for free with a good circuit simulator such as LTSpice . But writing down the expressions to do the impedance calculations for the ladder branches is good learnin'. You should never take your tools for granted, and doing something in a slightly more 'primitive' way keeps the fundaments fresh in the engineering part of your brain.

There's a further cool thing you can do with a spreadsheet that most SPICEs can't. That's to use the spreadsheet 'solver' functionality to adjust component values in the search for a better-fitting circuit – or even to find a set of component values for a circuit you can't otherwise design. Using familiar spreadsheet cell methods makes this a simple task. Let's review the basics.

In An Excelent Fit, Sir!, Excel's solver adjusted coefficients of a transfer function to minimize deviations outside a particular response "window" that we built by adding tolerance bands above and below the desired curve. There are several ways of forming the error value. For most tasks a simple, easy approach is just to add up the *squares* of the discrepancies between the response of the current solution iteration at each test frequency, and the nearest tolerance band limit. All these contributions are positive so an error in one point can't "cancel out" an error at another point. If at some frequency point in the data set the response lies inside the tolerance window, there's no error contribution. Minimizing the error value by adjusting coefficients gets you the "best fit" transfer function. Any solution for which the total error is zero completely meets your spec.

Because we build the error value row by row in a spreadsheet, it's easy to apply additional "weights" to the performance at particular frequency points. Deviations at some frequencies might be less critical than at others, so you can scale the contributions accordingly. We'll not do that here, but you'll see how easy it is as soon as you experiment with the technique yourself.

Instead of adjusting the coefficients of a transfer function, we can get the solver to minimize the error by adjusting the *components* in a circuit directly – provided that we can calculate the circuit's response using the spreadsheet. And now we can!

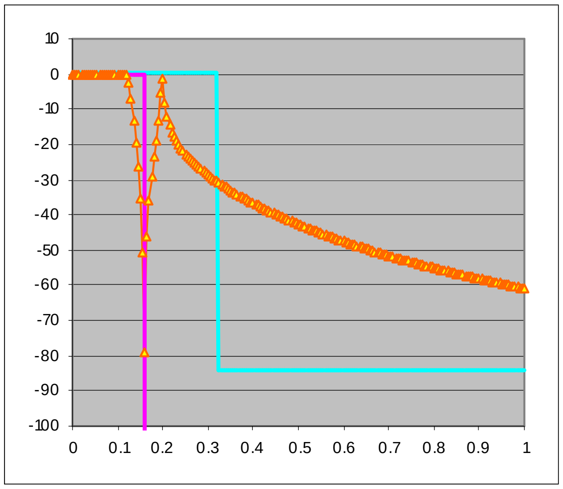
One use for this approach is to 'tweak' a particular circuit to compensate for some change or imperfection. Taking the bandpass filter from Fainting in Coils as an example, I changed the inductors to a smaller, cheaper, lower Q type, and used the solver method to adjust the capacitors and resistors to recreate a circuit with the same response shape as the original. It turns out that lowering the inductor Q causes more gain reduction than can be made up for by the other components. To make the optimization work at all, I had to introduce an extra gain parameter for the solver to tweak. This indicated an unacceptable 20dB of extra loss, no good for my application. I decided not to take up space in this article documenting that exercise.

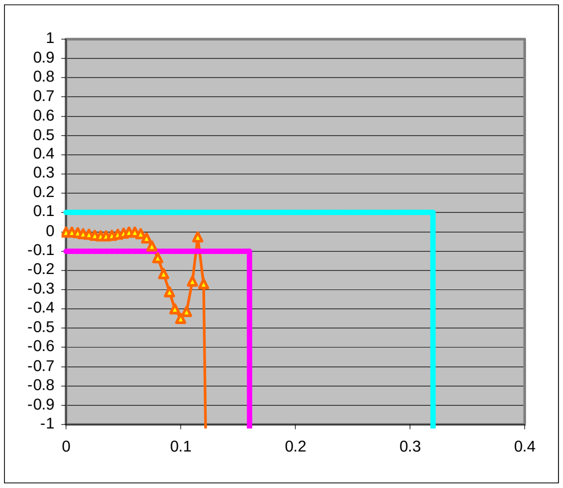
Such minor fiddling isn't very ambitious, anyhow. *Much* more exciting is the use of this technique to design filters from scratch. In this case, you use the solver capability to create the desired response by changing component values in a ladder network *whose initial response looks nothing like the desired final response*. This is useful either if you can't afford proper filter design software, or if you have a specific constraint that makes the circuit undesignable by other means.

Our 'tolerance window' approach to response specification is already familiar in classical filter design. Let's consider the example of a lowpass filter. A typical lowpass filter prototype spec, 'normalized' to the standard cutoff frequency of 1 radian per second, or 0.159Hz, might be:

passband attenuation 0dB +/-0.1dB from DC up to 0.159Hz
stopband attenuation >84dB for frequencies >0.318Hz.

In the passband, a data point contributes to the error value if the attenuation is >0.1dB or <-0.1dB (i.e. a gain of 0.1dB). In the stopband, we add to the error value if the attenuation is less than 84dB, but we don't mind if it goes up to any higher value. This filter prototype can be scaled later to any desired frequency and overall impedance level. Let's use these figures in an example.

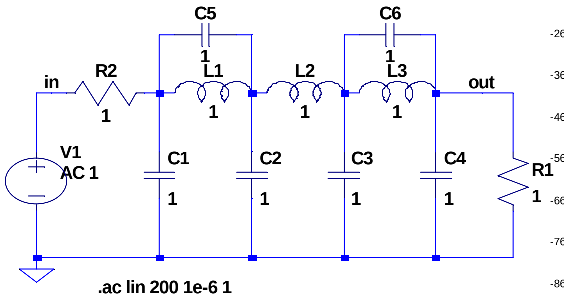


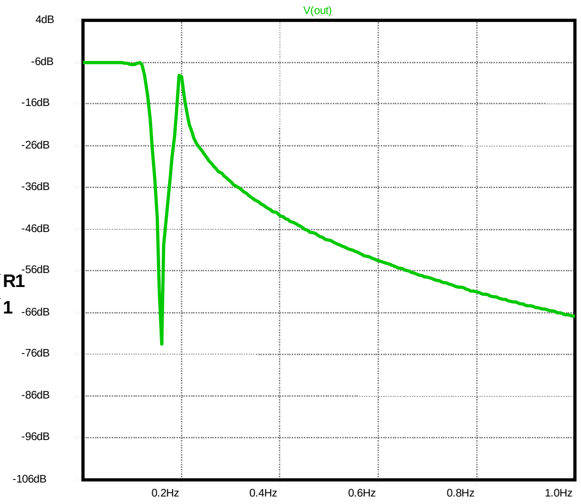


figures 1&2: example tolerance plot, plus initial response (orange) that doesn't meet it

Figures 1 and 2 show Excel plots for the response limits, and also the frequency response of an initial guess circuit that clearly does not meet the response requirements at all (it should lie entirely between the blue and pink outlines).

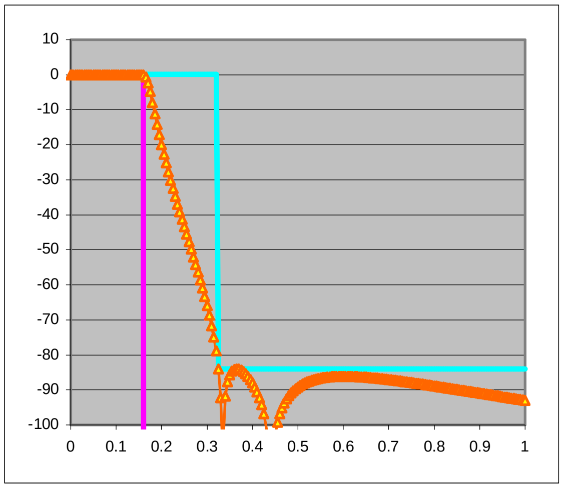
The initial guess circuit is shown in Figure 3. The topology is a standard 7th order lowpass ladder commonly used for elliptic filters, though I've omitted a capacitor across one of the inductors, making a circuit for which there are no standard design methods. Feigning complete ignorance of the likely component values, I've just set them all equal to unity. This is a good tip for users of the Excel solver – it's much better at solving problems where all the values stay within a few orders of magnitude of unity. Hence it's best to work with these normalized circuits rather than real component values.

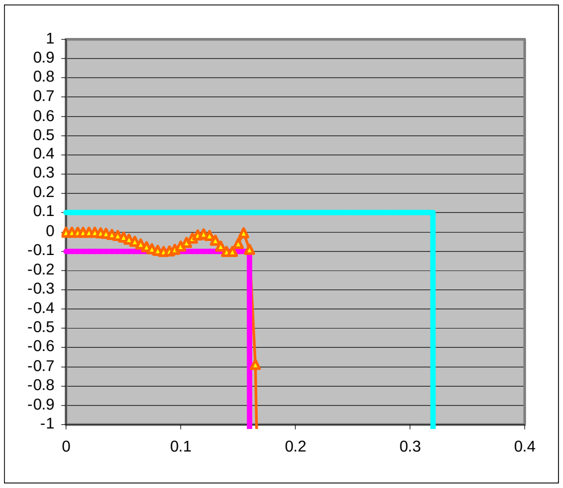




figures 3&4: the starting circuit, and response from LTSpice (compare figure 1)

I'm going to allow all the components in the network to be adjusted with the exception of the source and load impedances. There's a good reason why I want them to stay equal and unity – I'll write an article about that. Meanwhile, bear with me on that choice – and note that this means that there is a 6dB fixed attenuation always contributed by the presence of these source and load resistors, which we allow for in our error calculations. Now we press the 'solve' button, and see what happens.

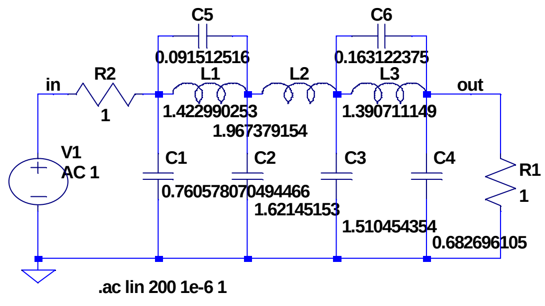


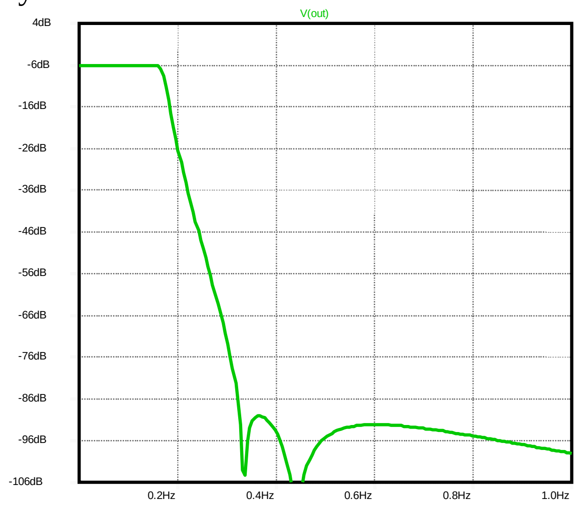


figures 5&6: The response of our filter after being Excelently optimized

And, believe it or not, we get the results shown in figures 5 and 6. Because I initially provided a size and topology of network that could meet the specification (with suitable component values), the solver was able to find a solution with a zero error value, at which point it terminated straight away. These results are unretouched from an actual run. Starting so far away from a good result, it does take a few minutes to converge. If you set the initial components to more ‘plausible’ values, it converges much faster.

The passband and stopband response look spookily similar to that given by standard analytical filter design techniques (one homework question for you: how could I modify the calculation to make use of *all* that gap between the upper and lower passband tolerance limits?). The circuit values and a corroborating LTSpice response plot (again, notice the 6dB flat loss compared to the Excel graphic) are shown in figures 7 and 8. No analysis more complicated than working out the impedance of an inductor and capacitor in parallel was needed in this process. You can immediately see the potential to add extra constraints; for example, that all the inductors should be the same value. Verily, you can produce work worthy of a filtery Shakespeare using only the uncomprehending labours of the millions, nay billions, of monkeys inside your PC.





figures 7&8: the circuit as calculated, and its response according to LTSpice

You’re not limited to optimizing just magnitude response. By manipulating the complex result from our in-worksheet simulation, we can work with phase shift, group delay, or the real and imaginary parts of the response. We can even optimize the time response to a stimulus such as a step function, or any other input waveform for which you can write down a Fourier series. The Fourier series for the circuit’s output is trivially produced by multiplying the coefficients of the original series by the calculated complex gain of the circuit at the corresponding frequency. The time waveform is recreated by summing up all the resulting gain- and phase-shifted sinewaves. The error can then be formed from the fit of the time response to a specified time function. This has some interesting uses that I’ll write about sometime.

The ladder algorithm readily extends to calculating network input and output impedances, enabling matching network design and loudspeaker impedance compensation. You could even simultaneously optimize a transfer function and an impedance function by combining two sets of deviations into a single error.

The technique can be used to design non-classical prototype topologies that give excellent filtering performance when transformed into active filters using remarkably few op-amps. Yet another article! Meanwhile, try this technique, to get practice with using spreadsheet solvers. They really deserve a place in your signal-management toolbox. Let me know how it goes. *Optimus est!* best – Kendall

P.S. Check out my Cypress blog at www.cypress.com/go/thefilterwizard - you can also email me from there.