README file for LTSpice_opt program

LTSpice_opt is a Python program that uses an iterative optimization approach to design analog filters. It is designed to be used in conjunction with the popular circuit simulator LTspice. It embeds an LTSpice simulation inside the powerful Python nonlinear least-squares optimizer. It works as follows;

- The user provides a target frequency and/or phase response in Python, and a circuit topology in LTspice with some initial component values.
- The user provides a list of which circuit instances the optimizer is allowed to vary.
- The optimizer then iteratively adjusts those component values, running a simulation for every pass through the least-squares algorithm, in an attempt to reduce the error between the target frequency response and the simulated response.
- Once the optimizer has finished, a new schematic is generated with the optimized component values. During the schematic generation process, each component value is quantized to a user-defined tolerance.

Why is this capability useful? Don't we already know how to design filters?

Traditional filter design uses standard circuit topologies such as Sallen-and-Key or multiple-feedback op-amp based active filters. In these cases, given a "standard" filter shape such a Butterworth or Chebychev, a high-order filter may be factored into 2nd-order sections, and an op-amp circuit can be used for each of those sections. This design procedure is quite straightforward and has not changed for many years. However, there are many cases where this approach fails;

- **The desired filter shape is not a traditional shape** such as Butterworth or Chebychev.
    - For example, the filter may need to compensate for some other part of the system that has a non-flat frequency response, while simultaneously attenuating other frequency regions.

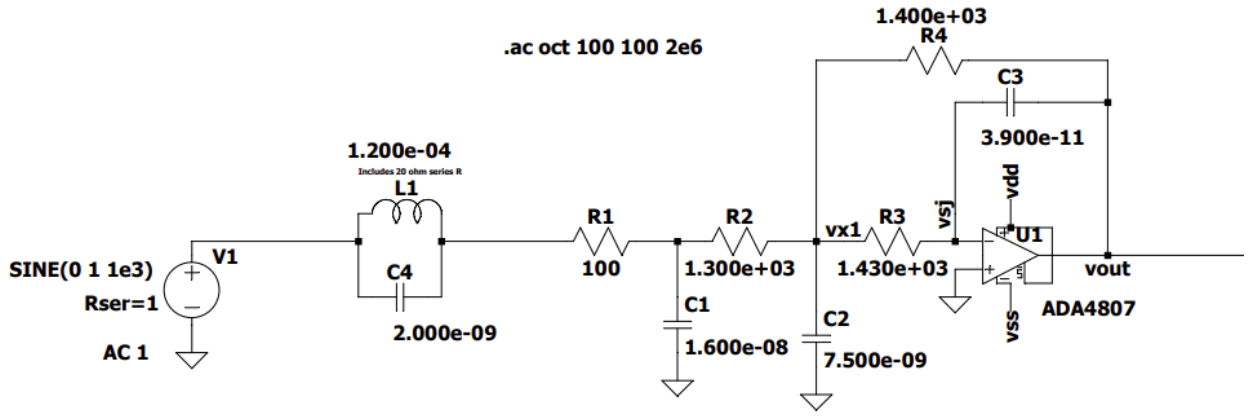
- **The need to reduce power/area by combining multiple filter sections into a single op-amp circuit.**
    - This leads to non-conventional circuit topologies that have very messy closed-loop formulas, and it becomes very difficult to solve for the component values.

- **The application operates at frequencies where finite op-amp gain-bandwidth degrades the frequency response.**
    - Calculating the effects of finite gain-bandwidth on the frequency response is quite complicated, especially in cases where the gain/phase response deviates from the traditional single-pole model. By running optimizer simulations in LTSpice, the actual target op-amp may be included in the simulation. This yields a solution that inherently attempts to compensate for finite gain-bandwidth effects.

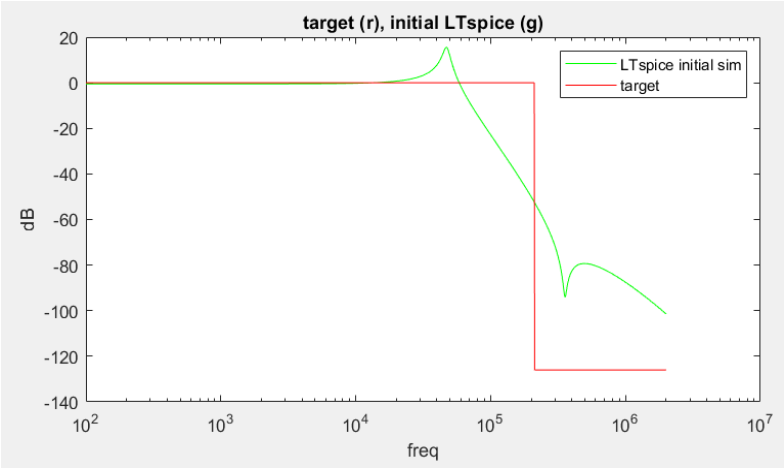
Example circuit and results

The optimized circuit of “example2.asc” (included in this distribution) is shown below. Note that while one could obtain the transfer function of this circuit by manual or automatic means, it is not in a form that would match any conventional lowpass filter shape. By using the passband/stopband/don’t-care band approach, we can still obtain a lowpass shape that is quite sharp for a single op-amp filter.

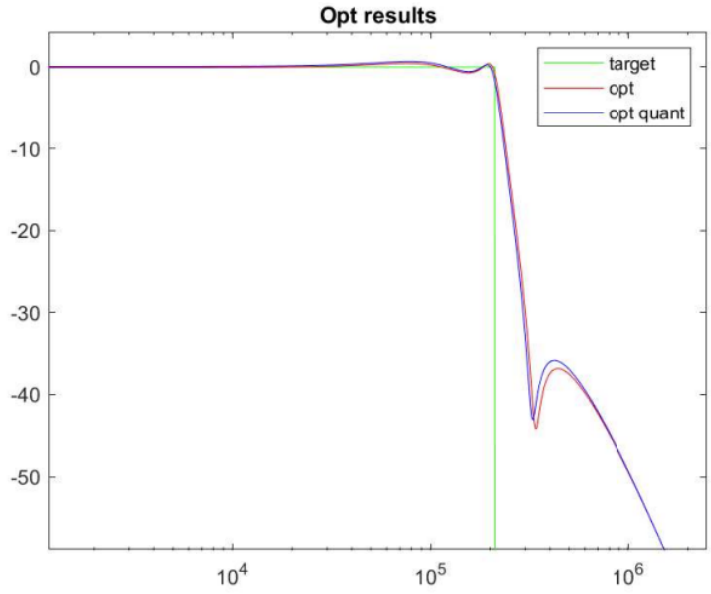
Schematic (post-optimization)



Pre-optimization simulation



Post-optimization simulation



Python files included in the gitHub repo;

LTSpice_opt.py

- the main application

-the 1st 2 lines import the user-specific information about the target circuit

example1_setup.py, example2_setup.py

Example setups for example1 and example 2 circuits

Each of these files include 2 functions;

Set_target: for setting the target response

simControl: for setting schematic and system--specific variables

These functions can be copied and modified for any particular circuit. The following information needs to be provided;

- path to the LTSpice executable
- path to the LTspice circuit simulation directory
- file name of the LTspice circuit that will be optimized
- LTSpice variable that will be matched to the target
- list of instance names in the schematic that the optimizer will adjust
- min values of the instances above
- max values of the instances above
- tolerance of the instances above
- Match Mode, amplitude only, phase only, or both ampl+phase
- A target response (amplitude, phase, or both), calculated at the same frequencies used in the LTspice sim.

LTspice Files files included in gitHub repo

Example1.asc - a 3rd-order single-op-amp inverting lowpass filter.
Example2.asc - a complicated single-op-amp filter including an LC stopband notch

Installation of the files

Place all the “.py” files in a directory where you will run Python.
Place all the .asc files in a directory where you will run LTspice.
The full-path location of the LTspice directory as well as the LTspice executable needs to be entered in the example setup files (see below).

Setting up the Python environment

The following must be installed in your Python environment;

numPy - included with most distributions
Matplotlib.pyplot - included with most distributions
PyLTSpice; from PyLTSpice · PyPI. This is used to read the LTspice .raw simulation files.

This code was developed using the popular Anaconda 3 environment, and run using the python command-line prompt.

Running the examples

The setup files must first be modified to provide the correct paths to the LTspice executable and the LTspice schematic directory. The main application can then be started from a Python command line by running “Python -i LTspice_opt.py” The first line of the main application is used to import one of the two example setup files; these can be edited later to point to a user-defined circuit setup file.

How to run your circuit;

1) Enter a schematic in LTSpice. Set the simulation command to do a frequency response sweep

example, '.ac oct 100 100 2e6'.

Run the simulation from within LTSpice to make sure there are no errors, and plot the frequency response of the output node. Note that the frequencies used in the LTSpice simulation are passed into the "setTarget" function as the variable "freqx". The target response must be computed at these same frequencies.
It’s worth noting that if you use a log frequency grid in your simulation, the optimizer will pay equal “attention” to each octave, whereas if you use a linear frequency sweep in your simulation, the optimizer will pay equal attention to each Hz. It’s worth thinking about which result you prefer; for audio circuits, a log-sweep is typically preferred.

2) Identify the component instance names that you want the optimizer to adjust. Note that making ALL the components adjustable is usually a bad strategy. For example, for most filters it is possible to scale the R's by some factor k, and also scale the C's by 1/k, without changing the frequency response. If all the R's and C's are adjustable, then the optimizer may drift off into an undesirable range of values, so it makes sense to pick one component that is not in the list of adjustable values, or alternatively restrict the range of one component to values that are acceptable.

3) Fill in the simControl function in the setup file. This file is heavily commented and it should be fairly clear how to enter your design.

The first step is to fill in the paths to your LTspice executable as well as the LTspice working directory (where you placed the .asc files).

Next, you need to fill in the information about which schematic instances will be adjusted by the optimizer. For each component listed that will be optimized, you must enter a min and max value. In most cases, the spread of min and max values should be fairly wide, to avoid limiting the optimizer; however, in some cases it makes sense for a particular component to have a more narrow allowable spread. For example, the min value of an input resistor may need to be large enough so it can be easily driven, but the maximum value may be limited by noise or bias current considerations.

In addition to the instance names, min values, and max values, the user should fill in the tolerance of the newly-adjusted components. This is specified in standard "E" format (number of values per decade), as defined in the comment section of the file. Note that the tolerance quantization is done "outside" the least-squares loop, as part of the schematic-generation process.

The above information is entered into Python lists. The example files can be copied and edited to minimize the chance of entry errors.

The “matchMode” parameter may be set to amplitude-only (1), phase-only (2), or both amplitude and phase (3).

4) Edit the setTarget function in the setup file. This function contains a frequency vector called "freqx" that will be imported from the main program, and will contain all the frequencies that are used in the LTSpice sim. You must generate a vector called "target" with the dimensions as freqx. "Target" should contain the desired magnitude response (note, NOT in dB) of your filter at the corresponding frequencies in freqx. In the case of matching phase-only, the

target should be set to the unwrapped phase in radians. For both magnitude and phase matching, target should be the concatenation of amplitude and phase targets.

Also in the setTarget function is a vector called "errWeights". This vector controls how much to weight the error at the corresponding frequencies in the freqx vector. It is initialized to all 1's, which may be adequate for many designs. However, when the filter design contains some combination of passbands, stopbands, and transition bands, it may be useful to set the weighting to 0 in the transition bands, and to some larger number in the stopbands. A rule of thumb is that the ratio of linear passband ripple to linear stopband ripple should equal the ratio of the stopband weights to the passband weights.

If the goal is to design a classic filter shape such as Butterworth or Chebyshev, then the built-in sciPy signal-processing functions may be used to set the target response. An example of this technique is shown in the circuit “example1.asc’ with the corresponding setup file “example1_setup.py”.

Note that the sciPy optimizer is a least-squares optimizer, and therefore cannot be expected to return an equiripple design. However, the user can experiment with running the algorithm multiple times, and for each new run, adjusting weighting factors in regions where the ripple is too high. This can eventually result in a nearly equiripple design.

5) Run the main program LTSpice_opt.. See the requirements for setting up your Python environment above.

If everything is set correctly, you will see a series of updates in the Python command window, which tracks the algorithm progress. For each iteration of least-squares, the component values are displayed, along with the rms error in the frequency response versus the target. Depending on many factors, the optimizer may run for only a few minutes, or for much longer if the design is complicated.. Go have a cup of coffee!

In addition to the updated component values, the window displays the cumulative number of simulations.

When the algorithm has finished, a plot is displayed of the target response, the optimized response with arbitrary-precision components, and the optimized response with components that were quantized according to user input as described earlier. A new schematic is also generated with the quantized optimized component values, with a schematic name the same as the original but with "opt" appended to the name. It is advisable to open this schematic and run

the simulation from LTspice to make sure that everything has worked as expected.
If the quantization of component values to their specified tolerance results in excessive error, a useful strategy is to take the worst-tolerance components (for example, 5% capacitors) and remove them from the list of instances to be optimized, and then re-run the optimizer based on the new '_opt' schematic. This will cause the more precise components (for example, 1% resistors) to adjust their value slightly to compensate for the effect of the 5% quantization of the capacitor values.

Caveats

There are a few things to bear in mind;

1) Least-squares optimizers are not guaranteed to find a global optimum. Therefore it's worth spending some time manually adjusting component values to make your response as close as possible to the target before launching the optimizer.
2) There are some circuits that have difficulty finding a DC operating point before running the AC sim. In these cases, not only is the optimizer very slow, but there may be an issue where a new DC solution is found for each pass through the optimizer, and this can affect the AC response in ways that are not linear.. This behavior is common when using op-amps with non-zero input bias currents (typically bipolar op-amps). If you encounter such a case, it is wise to use ideal op-amps with no input bias currents. Once the optimizer has found a solution with ideal op-amps, then you can put in your desired specific op–amp from the library and run the optimizer again, starting with the previously-optimized RLC values. By starting the simulation at a point that is very close to the optimum, the difficulties mentioned above can often be overcome.

Acknowledgements

This development was inspired by a program written more than 40 years ago by **Mark Davis**, who was my co-worker at the time at dbx Inc. Mark is an MIT PHd graduate, and has made extensive contributions to the field of audio during the

course of his career. His original program ran in a DOS command window and was 100% text-based. Mark used it to design crossover filters for loudspeakers (where the target response came from a speaker measurement program), while others used it to design non-standard op-amp circuits. I hope this program proves to be as useful as the original!

License