This is a very basic example of using *scipy's* brute force optimizer to optimize the z domain poles and zeros to match an s domain prototype filter. Since the optimizer starts by doing an exhaustive search over a range of values for any number of variables, this becomes very slow rapidly as the number of variables or fineness of the search grid grows.

This example is set up for a peaking equalizer where the gain , Q, and frequency of interest (fo) are specified. I suggest the reader familiarize themselves with the optimizer by reading the online description as well as *The Audio EQ Cookbook.*

The function *fun(x)* is minimized by the optimizer via the array (x) of variables. Here x[0] is fo, x[1] is A (gain), and x[2] is Q. The poles and zeros are extracted as the roots of the characteristic polynomial and the peak to peak error between the z domain response and the s domain response is returned.

The values in the body of the routine will need to be edited for different frequencies, other parameters, and different types of filters. Start and stop give a range of frequencies over which to optimize. The ranges are the set of ranges for the array of variables (x) passed by the optimizer (*brute()*). I use these to multiply the filter parameters so (.8, 1.2) is an ~ +- 20% range while the variable Ns is the fineness of the search. There is lots of room for experimentation with the values that are left up to the user.

The optimizer returns the optimized values for the x array which are then used to compute the optimized poles and zeros and the coefficients for the IIR filter which are printed out at the end.

This example is for a 5dB, 10kHz peaking filter with a Q of .5 and a sampling frequency of 48kHz. I chose to optimize only to 12kHz because increasing this rapidly degrades the fit due to the frequency warping. In fact it is easy to get no better a fit than by simply using the suggested "tweek" to the Q mentioned in the main article.

Running the script generates the following output which consists of the poles and zeros of the prototype followed by the optimum z domain poles and zeros with the computed IIR filter coefficients.