How to implement a digital biquadratic notch filter (in software)

by [Brandon Bickerstaff](https://www.linkedin.com/in/brandonbickerstaff/)

# Background

As presented in the [complimentary white paper](https://github.com/branbick/oscar/blob/main/biquad_notch_filter/doc/damping_ratios.pdf), the continuous-time transfer function of a biquadratic—or “biquad”—notch filter is …

Where Laplace variable, damping ratio (—), center/notch frequency (rad/s), and the subscripts and designate “numerator” and “denominator,” respectively. is a design parameter, and the aforementioned white paper derives an algorithm for calculating and . That’s great, and can be practically utilized with the “help” of third-party software such as [MathWorks’ Control System Toolbox](https://www.mathworks.com/products/control.html). But what if the goal is to implement a notch filter in our *own* software—software running on an embedded system, for example? If that’s the case, then a *digital* filter is needed, and its coefficients and corresponding difference equation must be determined.

# Derivation

## Bilinear transform

The [-transform](https://lpsa.swarthmore.edu/ZXform/FwdZXform/FwdZXform.html) is the discrete-time counterpart of the [Laplace transform](https://lpsa.swarthmore.edu/LaplaceXform/FwdLaplace/LaplaceXform.html), where ...

And sampling period (s), a design parameter. The first-order, [Maclaurin-series](https://mathworld.wolfram.com/MaclaurinSeries.html) approximation of is . Thus, …

The latter result is known as the bilinear transform: replacing with allows a transfer function to be converted from continuous time to discrete time. Unfortunately, that transformation causes a “warping” of the frequencies—because, after all, it’s based on an approximation (notably, a nonlinear one). For example, upon bilinearly transforming , a Bode plot of the consequent, *digital* filter would show that —which was specified for the original, *analog* filter—shifted. (Try it for yourself!) Obviously, that’s problematic, as the frequency the filter was designed to “notch out” won’t properly be attenuated. Thankfully, that predicament can be remedied via frequency prewarping.

## Frequency prewarping

Let …

* be an arbitrary (continuous-time) transfer function
* be the bilinearly transformed (discrete-time) equivalent of
* be the defining (analog) frequency of —e.g., the center frequency of a notch filter
* be the warped (digital) frequency corresponding to

The objective is to determine the relationship between and that makes the frequency responses of evaluated at and evaluated at —recalling the previous definition of —equal, i.e., …

Employing the bilinear transform yields …

Where the left-hand side equals …

(Refer to [Wikipedia](https://en.wikipedia.org/wiki/Bilinear_transform#Frequency_warping) for the redacted “mathematical magic.”) Therefore, …

Clearly, : the former does indeed get “warped” (into the latter) upon application of the bilinear transform. But, in practice, those two frequencies *should* be identical: the defining frequency that the design of the initial transfer function is established on (in the domain) *should* be accurately represented in the domain. So, what if the bilinear transform could be multiplied by a constant, , that *pre*warps the analog frequencies such that, upon digitization of , 1) , and 2) the frequency-response equality—i.e., —is maintained? If that were possible (hint: it is), then, similar to above, …

Where the left-hand side equals …

Therefore, …

Consequently, the bilinear transform *with frequency prewarping* is given by …

That is, replacing with allows a transfer function to be converted from continuous time to discrete time *while avoiding warping of the defining frequency, (rad/s)*.

**Bonus resource:** Check out [this video by Brian Douglas](https://youtu.be/NRbGPgcLhU0) for an incredible explanation of this topic.

## Filter coefficients

Now, apply the bilinear transform *with frequency prewarping* to , noting that is the defining frequency of a notch filter:

Canceling out and multiplying the numerator and denominator by yields …

Collecting like terms yields …

For the sake of simplicity, because the numerator and denominator have the same form, let …

Which is merely the numerator/denominator of with / replaced by . Applying the [half-angle](https://en.wikipedia.org/wiki/List_of_trigonometric_identities#Half-angle_formulae) and [power-reduction](https://en.wikipedia.org/wiki/List_of_trigonometric_identities#Power-reduction_formulae) formulas for tangent (recalling that ) yields …

Multiplying both sides by yields …

Dividing both sides by and applying the Pythagorean identity yields …

Dividing both sides by yields …

Such that …

Reconstructing as …

Yields …

Rewriting the above in standard form yields …

Where …

Finally, let …

Such that the filter coefficients (below) are normalized by for the sake of implementation:

## Difference equation

Where and output and input, respectively, of . Thus, …

Applying the time-shifting property of the -transform yields …

Where and *discrete-time* output and input, respectively, of the *digital* biquad notch filter, and sample number such that …

* current output
* previous output
* second-to-last output
* current input
* previous input
* second-to-last input

Ultimately, the following function can be implemented in software to “notch” a discrete-time signal:

[Note that the above is the [direct-form-I realization](https://www.dsprelated.com/freebooks/filters/Direct_Form_I.html) of .]

# Summary

The (software) implementation of a digital biquad notch filter is as follows:

Where …

And …