Bilinear Transform Computation Software

Raza

June 25, 2025

1 Introduction

Bilinear Transform Computation Software, is a tool developed for designing and analyzing digital filters derived from analog prototypes using the bilinear transform. The software supports filter design, frequency response analysis, time-domain simulation, stability verification, and inverse bilinear transformation. It implements mathematical models and algorithms to facilitate the conversion between analog and digital domains, compute filter characteristics, and simulate filter behavior.

2 Functioning Logic

The software operates by transforming analog filter transfer functions from the s-domain to digital filter transfer functions in the z-domain using the bilinear transform. It supports multiple filter types (Butterworth, Chebyshev I/II, Elliptic, Bessel) and provides tools for analyzing frequency response, time-domain response, and stability.

- Filter Design: Generates analog filter transfer functions based on user-specified parameters (e.g., filter type, order, cutoff frequency).
- Bilinear Transform: Converts analog transfer functions to digital using the bilinear transform.
- Frequency Pre-Warping: Adjusts frequencies to compensate for warping effects.
- Stability Analysis: Verifies digital filter stability by analyzing pole locations.
- Time-Domain Simulation: Computes impulse, step, and custom input responses.
- Inverse Bilinear Transform: Converts digital filters back to analog for analysis.

3 Mathematical Models and Equations

The software relies on signal processing and control theory principles, with the following key models and equations.

3.1 Transfer Function Representation

A transfer function represents a linear time-invariant system as the ratio of output to input in the frequency domain. For an analog filter, it is defined as:

$$\mathrm{H}(s) = \frac{N(s)}{D(s)} = \frac{\sum_{k=0}^{M} b_k s^k}{\sum_{k=0}^{N} a_k s^k},$$

where $N(s)$ and $D(s)$ are polynomials in the Laplace variable s, with coefficients b_k (numerator) and a_k (denominator), and M and N are their respective degrees. For a digital filter, it is:

$$\mathrm{H}(z) = \frac{N(z)}{D(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}},$$

where z is the z-transform variable. The software uses the `SymbolicTransferFunction` class to encapsulate numerator and denominator coefficients and the domain variable (s or z).

3.2 Bilinear Transform

The bilinear transform maps the s-domain to the z-domain using:

$$s = \frac{2}{T} \cdot \frac{z-1}{z+1},$$

where T is the sampling period. This transform preserves stability and approximates the frequency response by mapping the analog frequency axis ($\mathrm{j}\omega$) to the unit circle in the z-plane. The resulting digital transfer function is computed by substituting s into the analog $\mathrm{H}(s)$, yielding:

$$\mathrm{H}(z) = \mathrm{H}(s)\Big|_{s=\frac{2}{T}\cdot\frac{z-1}{z+1}}.$$

The `BilinearTransform` class performs this substitution by expanding the numerator and denominator polynomials, computing coefficients for $(z-1)^k$ and $(z+1)^k$ using binomial expansions.

3.3 Frequency Pre-Warping

The bilinear transform warps the frequency axis according to:

$$\omega_a = \frac{2}{T} \tan\left(\frac{\omega_d T}{2}\right),$$

where ω_a is the analog frequency and ω_d is the desired digital frequency. The `PreWarpingCapability` class computes the pre-warped frequency to ensure the digital filter's critical frequencies (e.g., cutoff) align with the analog prototype. The inverse mapping is:

$$\omega_d = \frac{2}{T} \arctan\left(\frac{\omega_a T}{2}\right).$$

3.4 Inverse Bilinear Transform

The inverse bilinear transform maps a digital filter back to the analog domain using:

$$z = \frac{2+sT}{2-sT}.$$

This is implemented in `InvBilinearTransform`, which substitutes z into $\mathrm{H}(z)$ to obtain $\mathrm{H}(s)$. The resulting analog poles and zeros are computed for visualization.

3.5 Stability Analysis

A digital filter is stable if all poles of $\mathrm{H}(z)$ lie inside the unit circle ($|z| < 1$). The `StabilityVerification` class computes poles by finding the roots of the denominator polynomial $D(z)$ and checks their magnitudes. The filter is stable if:

$$\max_i |p_i| < 1,$$

where p_i are the poles.

3.6 Time-Domain Response

The time-domain response is computed using the difference equation for a digital filter:

$$y[n] = \frac{1}{a_0}\left(\sum_{k=0}^{M} b_k x[n-k] - \sum_{k=1}^{N} a_k y[n-k]\right),$$

where $x[n]$ is the input sequence, $y[n]$ is the output sequence, and a_0 is the leading denominator coefficient. The `TimeDomainSimulation` class implements this for:

- Impulse Response: $x[0] = 1$, $x[n] = 0$ for $n \neq 0$.
- Step Response: $x[n] = 1$ for all $n \geq 0$.
- Custom Input Response: User-specified sequence $x[n]$.

3.7 Frequency Response

The frequency response is computed by evaluating $\mathrm{H}(z)$ on the unit circle ($z = e^{\mathrm{j}\omega}$):

$$\mathrm{H}(e^{\mathrm{j}\omega}) = \frac{\sum_{k=0}^{M} b_k e^{-\mathrm{j}\omega k}}{\sum_{k=0}^{N} a_k e^{-\mathrm{j}\omega k}}.$$

The magnitude (in dB) and phase are:

$$|\mathrm{H}(e^{\mathrm{j}\omega})|_{\mathrm{dB}} = 20\log_{10}\left|\mathrm{H}(e^{\mathrm{j}\omega})\right|, \quad \arg(\mathrm{H}(e^{\mathrm{j}\omega})) = \tan^{-1}\left(\frac{\mathrm{Im}(\mathrm{H}(e^{\mathrm{j}\omega}))}{\mathrm{Re}(\mathrm{H}(e^{\mathrm{j}\omega}))}\right).$$

The group delay is:

$$\tau_g(\omega) = -\frac{d}{d\omega}\arg(\mathrm{H}(e^{\mathrm{j}\omega})).$$

These are computed in `FrequencyResponse` and `FrequencyResponsePanel`.

4 Algorithms Used

The software employs several algorithms to implement the mathematical models.

4.1 Polynomial Root Finding (Laguerre's Method)

The `StabilityVerification` class uses Laguerre's method to find roots of numerator and denominator polynomials for poles and zeros. For a polynomial $p(z) = \sum_{k=0}^{n} a_k z^k$, the algorithm:

1. Initializes a random complex guess z_0.
2. Iteratively updates the guess using:

   $$z_{k+1} = z_k - \frac{np(z_k)}{G \pm \sqrt{(n-1)(nH - G^2)}},$$

   where $G = \frac{p'(z_k)}{p(z_k)}$, $H = G^2 - \frac{p''(z_k)}{p(z_k)}$, and n is the polynomial degree. The larger denominator is chosen to ensure convergence.
3. Deflates the polynomial after finding a root by synthetic division:

   $$p(z) = (z - r)q(z) + R,$$

   where r is the found root and $q(z)$ is the deflated polynomial.
4. Repeats until all roots are found or numerical issues arise.

The method handles complex arithmetic and checks for convergence using a small threshold ($\varepsilon = 10^{-10}$).

4.2 Bilinear Transform Implementation

The `BilinearTransform` and `DirectBilinearMappingEngine` classes compute the digital transfer function by:

1. Substituting $s = \frac{2}{T} \cdot \frac{z-1}{z+1}$ into $\mathrm{H}(s)$.
2. Expanding the resulting expression:

$$\mathrm{H}(z) = \frac{\sum_{k=0}^{M} b_k \left(\frac{2}{T} \cdot \frac{z-1}{z+1}\right)^k}{\sum_{k=0}^{N} a_k \left(\frac{2}{T} \cdot \frac{z-1}{z+1}\right)^k}.$$

3. Using binomial expansion for $(z-1)^k$ and $(z+1)^k$:

$$(z \pm 1)^k = \sum_{m=0}^{k} \binom{k}{m} z^m (\pm 1)^{k-m}.$$

4. Collecting like terms to form the numerator and denominator polynomials in z.

4.3 Filter Design (ADFilterMapping)

The `ADFilterMapping` class designs analog filters and applies the bilinear transform:

- Butterworth: Places poles on a circle at $s_k = e^{j\omega\pi(2k+n+1)/(2n)}$, where n is the filter order.
- Chebyshev I: Scales Butterworth poles by $\varepsilon = \sqrt{10^{R_p/10} - 1}$ for passband ripple R_p.
- Chebyshev II: Places zeros and scales poles inversely for stopband attenuation.
- Elliptic: Approximates pole placement with ripple parameters (simplified in the codebase).
- Bessel: Uses Bessel polynomial roots for linear phase approximation.

The resulting analog $\mathrm{H}(s)$ is transformed to $\mathrm{H}(z)$ using the bilinear transform.

4.4 Time-Domain Simulation

The `TimeDomainSimulation` class implements the difference equation algorithm:

1. Initializes input sequence $x[n]$ (impulse, step, or user-defined).
2. Computes output $y[n]$ iteratively using:

$$y[n] = \frac{1}{a_0} \left(\sum_{k=0}^{M} b_k x[n-k] - \sum_{k=1}^{N} a_k y[n-k] \right).$$

3. Stores $y[n]$ for $n = 0$ to $N_{\text{samples}} - 1$ (default 50 samples).

4.5 Frequency Response Computation

The `FrequencyResponse` and `FrequencyResponsePanel` classes compute:

1. Evaluate $\mathrm{H}(e^{\mathrm{j}\omega})$ at discrete frequencies $\omega \in [0, \pi]$.
2. Compute magnitude:
$$|\mathrm{H}(e^{\mathrm{j}\omega})| = \sqrt{\left|\frac{\sum b_k e^{-\mathrm{j}\omega k}}{\sum a_k e^{-\mathrm{j}\omega k}}\right|^2}.$$
3. Compute phase:
$$\arg(\mathrm{H}(e^{\mathrm{j}\omega})) = \tan^{-1}\left(\frac{\sum b_k \sin(\omega k)}{\sum b_k \cos(\omega k)}\right) - \tan^{-1}\left(\frac{\sum a_k \sin(\omega k)}{\sum a_k \cos(\omega k)}\right).$$
4. Compute group delay using numerical differentiation of the phase.

4.6 Inverse Bilinear Transform

The `InvBilinearTransform` class:

1. Substitutes $z = \frac{2+sT}{2-sT}$ into $\mathrm{H}(z)$.
2. Simplifies the resulting rational function to obtain $\mathrm{H}(s)$.
3. Computes poles and zeros of $\mathrm{H}(s)$ using Laguerre's method.

5 Simulation Logic

The simulation logic integrates the mathematical models and algorithms to provide dynamic analysis of digital filters.

5.1 Time-Domain Simulation

The `TimeDomainSimulation` class simulates the filter's response to:

- Impulse Input: Simulates the filter's natural response to a unit impulse.
- Step Input: Simulates the response to a sustained unit input.
- Custom Input: Allows arbitrary input sequences, enabling flexible testing.

The difference equation is computed iteratively, with results stored for visualization. The simulation uses 50 samples by default, with each sample computed using the filter coefficients.

5.2 Frequency-Domain Simulation

The frequency response is simulated by evaluating $\mathrm{H}(e^{\mathrm{j}\omega})$ over a range of normalized frequencies ($\omega \in [0, \pi]$). The software computes:

- Magnitude response in dB for gain analysis.
- Phase response for phase distortion analysis.
- Group delay for phase linearity assessment.

The `FrequencyResponse` class provides detailed plots, while `FrequencyResponsePanel` offers a simplified magnitude plot.

5.3 Stability Simulation

The `StabilityVerification` class simulates stability by:

1. Computing poles of $\mathrm{H}(z)$ using Laguerre's method.
2. Checking if all pole magnitudes satisfy $|p_i| < 1$.
3. Providing pole and zero locations for visualization in `StabilityFeedbackWindow`.

The Nyquist plot evaluates the stability by plotting $\mathrm{H}(e^{j\omega})$ in the complex plane and checking encirclements of the critical point (-1, 0).

5.4 Pole-Zero Analysis

Poles and zeros are computed for both digital and analog transfer functions using `PolynomialRootFinder` and `StabilityVerification`. The results are visualized in:

- Pole-Zero Plot: Shows poles (x) and zeros (o) relative to the unit circle (z-domain) or imaginary axis (s-domain).
- Nyquist Plot: Displays the frequency response trajectory in the complex plane.

6 Something to read

Ich fühlte ein Licht des Glücks, nachdem ich die Entwicklung der Software abgeschlossen hatte. Es war keine leichte Aufgabe, die Software von Grund auf neu zu schreiben und die notwendigen mathematischen und physikalischen Modelle, Berechnungen und Algorithmen zu integrieren. Während der Entwicklung war ich ziemlich gestresst, hatte Kopfschmerzen und überanstrengte Augen, aber trotzdem habe ich es geschafft. Diese Software könnte Fehler aufweisen und verbesserungsbedürftig sein, daher ist sie möglicherweise nicht 100% perfekt. Aber wissen Sie was? Kein von Menschenhand geschaffenes Ding ist vollkommen perfekt.

I felt a sense of happiness after I finished developing the software. It wasn't an easy work to write the software from scratch and integration of necessary mathematics & physics models, calculations, and algorithms. While developing this software, I felt quite stressed alongside headache and strained eyes, despite this I managed to develop the software. This software might have flaws and may need further enhancements, so this software may not be 100% perfect. But guess what? No man-made thing is completely perfect.