SCIPY.CPP — Using AI to Port Python's SCIPY.SIGNAL Filter-Related Functions to C++ for Use in Real Time

Julius Smith
CCRMA, Stanford University

Audio Developer Conference Online (ADCxGather-24)

November 1, 2024

ADCx GATHER

SCIPY.CPP

USING AI TO PORT PYTHON'S SCIPY.SIGNAL FILTER-RELATED FUNCTIONS TO C++ FOR USE IN REAL TIME

JULIUS SMITH

Driving Problem: Real-Time Filter Design in an Audio Plugin

(Red-Bordered Buttons Added to **Plugin GUI Magic**'s Equalizer Example)

Methods for Arbitrary Filter Design

- Frequency Sampling

  1. *Draw* or *Load* Your Desired Magnitude Frequency Response
  2. Make it *Minimum Phase* (so the filter will be *causal*)
  3. Inverse-FFT gives the Desired Impulse Response (IR)
  4. “Window” the IR to the Affordable FIR length (smoothing the Frequency Response)
  5. Use *Convolution* to implement the FIR filter (typical for Amp Cabinets and such)

Methods for Arbitrary Filter Design, Continued

- Equation-Error Filter Design: Minimize $\|\hat{A}(\omega)H(\omega) - \hat{B}(\omega)\|$
  - E.g., `invfreqz` in MATLAB and Octave
  - We need C++ for an Audio Plugin!
    (or some easily embedded filter-design language)
  - AI Chatbots translate *well known languages* to C++ very well
  - They also write good starting *unit tests*
  - Speed Bumps:
    - MATLAB is proprietary (and no longer even precisely documented)
    - Octave is GPL (but contributing authors could be asked for permission)
    - Python is mostly BSD, but has no `invfreqz` yet in `scipy.signal`
  - **Plan:** Implement `invfreqz` from scratch in Python and translate to C++
  - **Method:** Paste the algorithm description[1] into Claude 3.5 Sonnet and debug
  - **This actually worked!**

[1] https://ccrma.stanford.edu/~jos/filters/FFT_Based_Equation_Error_Method.html

Claude 3.5 Sonnet Converts LaTeX Description of `invfreqz` to Python

1. *Prompt 1:*

   Following is a LaTeX description of a fast equation-error algorithm. Please write a Python implementation.
   <LaTeX source of algorithm description>

2. *Prompt 2:*

   Write a separate test program in Python which uses `` `scipy.freqz` `` to generate three different test examples of progressing complexity. That way, the original and estimated filter coefficients can be compared. A good source of example starting filters would be `` `scipy.signal.butter` `` and `` `scipy.signal.cheby1` `` etc.

3. This was the starting test program for the one in my `scipy` fork:
   `https://github.com/josmithiii/scipy/blob/jos/scipy/signal/test_invfreqz_jos.py`

Claude 3.5 Sonnet Converts a *Paper Abstract* to Working Python

Prompt: Write a Python function that designs a *spectral tilt filter* as described in this paper abstract:[2]

> We derive closed-form expressions for the poles and zeros of approximate fractional integrator/differentiator filters, which correspond to spectral roll-off filters having any desired log-log slope to a controllable degree of accuracy over any bandwidth. The filters can be described as a **uniform exponential distribution of poles along the negative-real axis of the s plane, with zeros interleaving them. Arbitrary spectral slopes are obtained by sliding the array of zeros relative to the array of poles, where each array maintains periodic spacing on a log scale.** The nature of the slope approximation is close to Chebyshev optimal in the interior of the pole-zero array, approaching conjectured Chebyshev optimality over all frequencies in the limit as the order approaches infinity. Practical designs can arbitrarily approach the equal-ripple approximation by enlarging the pole-zero array band beyond the desired frequency band. The spectral roll-off slope can be robustly modulated in real time by varying only the zeros controlled by one slope parameter. Software implementations are provided in matlab and Faust.

[2] `https://ccrma.stanford.edu/~jos/spectilt/spectilt.pdf`

Why Python?

- The test `__main__` block can conveniently use `numpy`, `scipy`, and `matplotlib` functions for test displays and subsequent interactive development
- Chatbots:
  - are trained on a *lot* of Python, and it's a relatively simple language,
  - are *not yet good* at signal processing (even simple polynomial algebra), and
  - tend to fall apart on low-level signal-processing details
- I influence them to work in terms of *well documented high-level APIs* such as functions in `scipy.signal` rather than writing C++ from scratch
- Translation from Python to C++ has been mostly smooth
- `Eigen3` gets used a lot

invfreqz.py Today

`invfreqz.py` is working now in the jos `scipy` fork at

https://github.com/josmithiii/scipy/blob/jos/scipy/signal/test_invfreqz_jos.py

(pull-request in preparation)

Features:

- New `min_phase` option for creating minimum phase desired frequency response
- New `stabilize` option for reflecting unstable poles into the unit circle
- New `method` argument for selecting other methods besides equation-error:
  - Equation-error method (default)
  - Steiglitz-McBride (original iterative method)
  - Prony's method (least-squares numerator)
  - Padé-Prony method (impulse-response-matching numerator)
  - Maybe: "Recursive Gauss-Newton iterations" [Hessian(n) $\approx \sum_n \nabla_n \nabla_n^T$)]
  - Maybe: *Neural map* from desired frequency response to starting poles and zeros
- All but Steiglitz-McBride are passing their unit tests
- It remains to decide what to finally do and integrate the proposed final version into `_filter_design.py` for a `scipy` pull request

`scipy.cpp` **Today**

Since Claude uses `scipy.signal` functions in its generated Python, we need those translated to C++ as well. Translated so far by Claude (most were fast):

- `tf2zpk` - convert transfer function to zero-pole-gain (ZPK) representation
- `zpk2tf` - inverse of tf2zpk
- `tf2sos` - convert transfer function to second-order-sections (sos)
- `sos2tf` - inverse of tf2sos
- `zpk2sos` - zero-pole-gain (ZPK) directly to SOS
- `roots` - compute the roots of a polynomial (uses Eigen3)
- `bilinear` - convert analog IIR filter to digital using bilinear transform
- `bilinear_zpk` - bilinear transform for zeros, poles, and gain
- `lp2lp_zpk` - lowpass to lowpass frequency scaling for analog zeros, poles, and gain
- Unit Tests for all (`Catch2`) — *This is very important — Claude can write most of them*
- Status:
  - Working through what's needed now in `_filter_design.py` and its dependencies
  - A complete `scipy.signal.cpp` would nice to complete from there
  - Other `scipy` subirectories, such as `fft` and `linalg`, are in much better shape

Summary

- Translating Python to C++ for real-time use is greatly facilitated by Chatbots
- Claude 3.5 Sonnet has been the clear winner
- They all struggle with sample-level signal processing, and polynomial algebra
- Several `scipy.signal._filter_design` functions are done and tested
- In general, Python is a good intermediate language for new C++ DSP functions
  - Pushes chatbots away from sample-level code
  - Facilitates visual test plots using `matplotlib` etc.
  - Encourages simpler C++ using Eigen3 etc.
- `invfreqz` is now available in Python on GitHub
- `scipy.signal.cpp` seems about half done
- These overheads (including all *links*) are available on the JOS Home Page (as well as the ADC website)

Summary of Resources Online

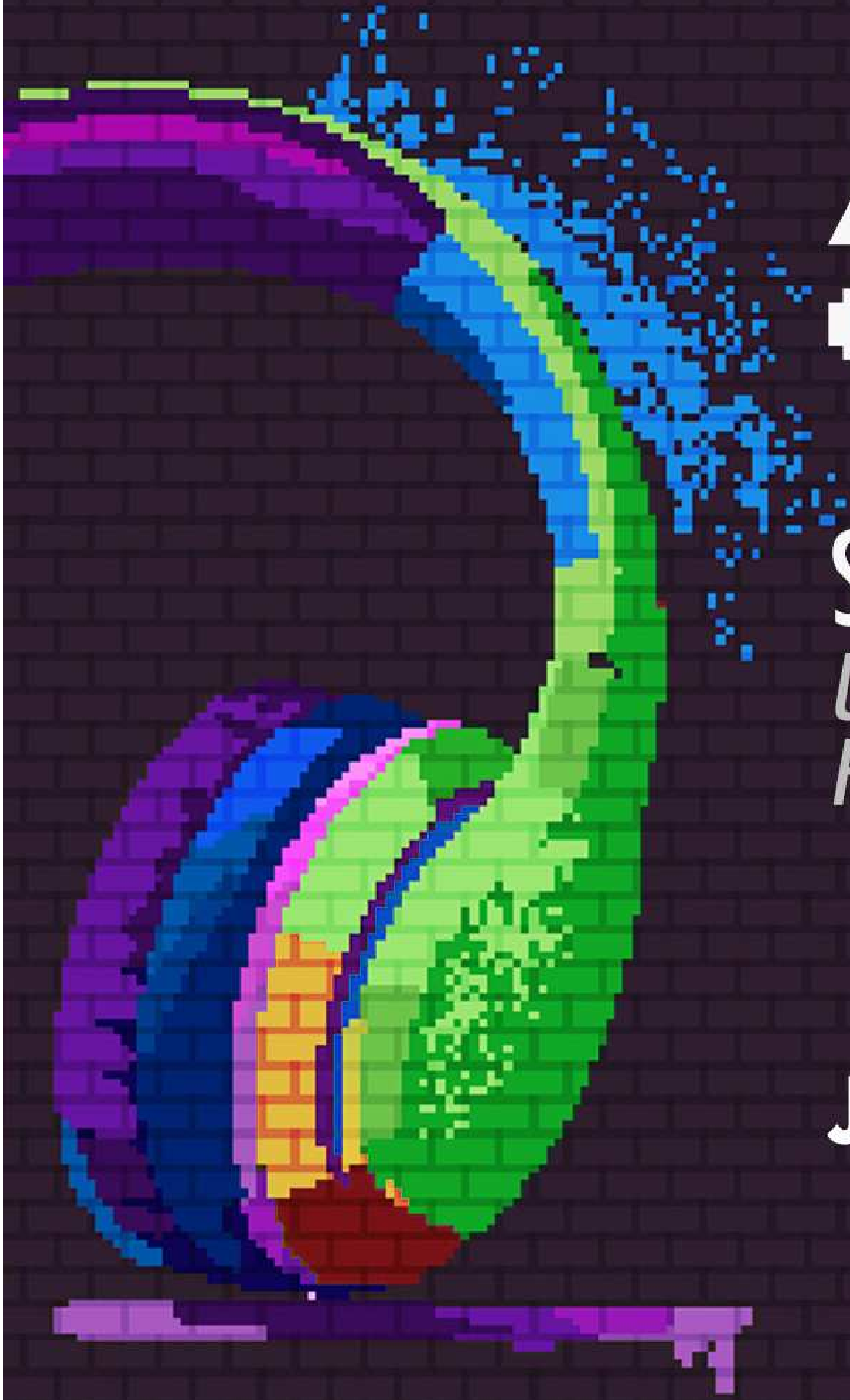
- JOS Home Page (Videos, Overheads):
  `https://ccrma.stanford.edu/~jos/`

- Equation-Error Minimization for Filter Design:
  `https://ccrma.stanford.edu/~jos/filters/FFT_Based_Equation_Error_Method.html`

- `invfreqz` for Python in JOS *scipy* fork:
  `https://github.com/josmithiii/scipy/blob/jos/scipy/signal/test_invfreqz_jos.py`

- Spectral Tilt Filters:
  `https://ccrma.stanford.edu/~jos/spectilt/spectilt.pdf`

- *Take 1* of this talk (correlated but different):
  `https://ccrma.stanford.edu/~jos/mp4/ADCxGather-take1-2024-10-26-20-E1.mp4`



ADCx GATHER

SCIPY.CPP

USING AI TO PORT PYTHON'S SCIPY.SIGNAL FILTER-RELATED FUNCTIONS TO C++ FOR USE IN REAL TIME

JULIUS SMITH