`filter_designer` – An Interactive Tool for Designing Digital Filters

Danny Harvey
Boulder Real Time Technologies, Inc.

Antelope User Group Meeting
ARSO, Slovenian Environment Agency, Ljubljana
2018 May

Digital Filtering in Antelope

- All digital filtering in Antelope utilizes **time-domain** convolution and recursive methodologies.
- Digital time-domain filtering offers significant advantages over FFT based frequency-domain filtering.
  1. Can operate on infinite time series in a continuous fashion.
  2. Minimal edge effects that can be confined within finite time windows.
  3. Much more computationally efficient.
  4. Simplicity of implementation.

Filtering Basics

- Filtering is defined as the convolution of two functions (from https://en.wikipedia.org/wiki/Convolution).

The convolution of *f(t)* and *g(t)* is written *(f∗g)(t)*,

$$(f * g)(t) \overset{\mathrm{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t-\tau)\, d\tau$$
$$= \int_{-\infty}^{\infty} f(t-\tau) g(\tau)\, d\tau.$$

- Filtering is fundamental to seismic data processing.

Filtering Basics

Another fundamental representation of data in seismology is its frequency-domain spectrum, as computed using the Fourier transform or its related Laplace transform.

$$\begin{aligned}\hat{f}(\omega) &= \mathcal{F}\{f(t)\} \\ &= \mathcal{L}\{f(t)\}\Big|_{s=i\omega} = F(s)\Big|_{s=i\omega} \\ &= \int_{-\infty}^{\infty} e^{-i\omega t} f(t)dt\end{aligned}$$

If $\mathcal{F}$ denotes the Fourier transform operator, then $\mathcal{F}\{f\}$ and $\mathcal{F}\{g\}$ are the Fourier transforms of $f(t)$ and $g(t)$ respectively. Then

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$$

Where $\cdot$ denotes point-wise multiplication.

By applying the inverse Fourier transform, we can write:

$$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}$$

Filtering Basics

- Implementation of Fourier transforms is done with the Discrete Fourier Transform (DFT) using a clever digital algorithm known as the Fast Fourier Transform (FFT).
- All DFTs, regardless of how they are implemented, are necessarily computed over finite time windows, usually no more than thousands of time samples, which causes them to be subject to an artifact known as "wraparound".
- FFT computational efficiency of order 5 * N * log2(N) vs brute force direct time-domain convolution computation efficiency of order 2 * N * N.
- However, most convolutions involve one function (the filter impulse response) with a reduced and constant value of N.

Antelope Filtering

- All filtering of time sampled waveforms in Antelope are done in the time domain and do not involve the computations of signal spectra using FFTs.
- All Antelope digital time domain filters can be applied to arbitrary time series and can be applied to continuous time series of indefinite length.
- There are no inherent time windowing parameters needed by the Antelope filters as there would be if filtering were done in the frequency domain. No “wraparound” effects.
- The Antelope time domain filters are very computationally efficient compared to frequency domain methods

BRTT

Kinemetrics

Antelope Filtering

- All Antelope time domain filters are implemented with the **`wffil(3)`** library which provides general purpose interfaces to various time domain waveform filter methods.
- Specific filtering groups are defined in **`wffilbrtt(3)`**, which includes Butterworth, generalized S-domain polynomials, differentiator/integrator, Wood-Anderson instrument response, generalized FIR filters, and **`wffilave(3)`**, which provides a variety of averaging filters.

Antelope Filtering

- In digital filtering, continuous signals of time, $s(t)$, are represented as discrete time series,
  $x[n] = s(nT)$, where n is the integer sample index ranging over minus infinity to plus infinity and T is the constant time sampling increment.
- The Antelope filtering library provides straightforward digital convolution filtering, also known as Finite Impulse Response or FIR filtering. If $f[k]|_{k=0}^{M-1}$ are the M FIR filter coefficients, the the output, $y[n]$, of the filtered input signal, $x[n]$, will be:

$$y[n] = \sum_{k=0}^{M-1} x[n-k-l] \cdot f[k]$$

Where l is a sample offset that defines the 0 time lag of the FIR filter.

- FIR filtering in Antelope is used primarily to simulate datalogger anti-alias filtering and primarily to support data resampling.

Recursive Digital Filtering

- Most filtering in Antelope for the purpose of data processing, such as the filters used in **`orbdetect`**, for example, is done using recursive digital filters, also known as Infinite Impulse Response, or IIR, filters.
- A new application, **`filter_designer`**, is available in the 5.8 release of Antelope. This app provides for the design and visualization of Antelope IIR filters.

Design of Recursive Digital Filters

- Basic approach:
  1. Design the filter as if it were an analog filter in the s domain (Laplace transform domain with $s = i\omega$).
  2. Convert the s domain representation of the filter transfer function to the z domain representation (Z transform domain with $z = e^{sT}$).
  3. Construct a "realizable" and hopefully stable recursion relation using the z domain transfer function.
  4. Filter the time sampled data using the recursion relation.

BRTT

Kinemetrics

Recursive Digital Filter Recursion Relation

```
for (i=2; i<nsamp; i++) {
     yout[i] = xin[i]*G + xin[i-1]*NM1
                        + xin[i-2]*NM2
                        - yout[i-1]*DM1
                        - yout[i-2]*DM2;
}
```

- This is the `c` code for filtering an input sampled function, `xin[i]`, to produce a filtered output function, `yout[i]`.
- The `G`, `NM1`, `NM2`, `DM1`, `DM2` values are all floating point constants. The recursion comes about due to the feedback of the two previous output values into the forward computation of the present output value.
- This implements a single filter stage that can represent any second order differential operator.
- This replaces the two forward FFTs, followed by a multiplication of two complex spectral values over the entire frequency range, followed by a reverse FFT required when using frequency domain filtering.
- This is by its definition a realizable relation. Its filter characteristics and stability are determined by the constant coefficients. The job of designing a digital recursion filter is to determine the constant coefficients based upon desired characteristics.

BRTT

Kinemetrics

Z-transforms

$$X(z) = \mathcal{Z}\{x[n]\} = \sum_{n=-\infty}^{\infty} x[n] z^{-n}$$

$$\mathcal{Z}\{x[n+k]\} = z^k \mathcal{Z}\{x[n]\} = z^k X(z)$$

$$z = e^{sT} = e^{i\omega t}$$

- Note that a z^k operator applied to a time series is equivalent to time shifting the time series by k index values.
- Also note that an s^k Laplace operator applied to a continuous time function is equivalent to differentiating the continuous time series k times, or integrating the continuous time series $-k$ times if $k < 0$.
- These properties provide logical connections between digital recursion relations as representations of z domain transfer functions and linear differential equations as representations of s domain transfer functions.

Recursion Equations to Z-transforms

$$y[n] = \sum_{k=0}^{K} x[n-k] \cdot N_k - \sum_{l=1}^{L} y[n-l] \cdot D_l$$

- If $K = 2$, $L = 2$, $N_0 = \mathtt{G}$, $N_1 = \mathtt{NM1}$, $N_2 = \mathtt{NM2}$, $D_1 = \mathtt{DM1}$, $D_2 = \mathtt{DM2}$, this relation is identical to the `c` code relation shown earlier.

$$Y(z) = \sum_{k=0}^{K} X(z) \cdot z^{-k} \cdot N_k - \sum_{l=1}^{L} Y(z) \cdot z^{-l} \cdot D_l$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{K} z^{-k} \cdot N_k}{1 + \sum_{l=1}^{L} z^{-l} \cdot D_l}$$

- In other words, we can easily derive a software code recursion formula to implement a Z-transform transfer function of a ratio of rational polynomials in *z*

BRTT

Kinematrics

Differential Equations to S-transforms

$$\ddot{y}(t) + 2\zeta\omega_0\dot{y}(t) + \omega_0^2 y(t) = x(t)$$

- Simple harmonic linear oscillator

Since $\mathcal{L}\{\ddot{y}(t)\} = s^2 Y(s)$ and $\mathcal{L}\{\dot{y}(t)\} = \mathrm{s}\, Y(s)$

$$s^2 Y(s) + s2\zeta\omega_0 Y(s) + \omega_0^2 Y(s) = X(s)$$

$$H(s) = \frac{Y(s)}{X(s)} = \frac{1}{s^2 + s2\zeta\omega_0 + \omega_0^2}$$

- In other words, we can easily derive an S-transform transfer function of a ratio of rational polynomials in *s* from a differential equation.

Strategy

- We are almost there.

1. Start by specifying a filter as a set of differential operators on the data. Typically, the filter transfer functions are designed in the frequency domain to accentuate certain frequencies in the data. This part is facilitated by **`filter_designer`**.
2. Convert the differential equations into S-domain transfer functions that are ratios of rational polynomials in *s*. This part is done automatically by the Antelope libraries.
3. Derive Z-domain transfer functions that are ratios of rational polynomials in *z* from the S-domain transfer functions. We need a conformal S to Z domain mapping function to do this.
4. Derive digital recursion relations from the Z-domain transfer functions. This part is done automatically by the Antelope libraries.
5. We now have simple formulas we can apply to the data to implement the filters.

S to Z Mapping

$$z = e^{sT}$$

$$s = \frac{1}{T}\ln(z)$$

- The exact mapping is not helpful since we cannot derive Z-domain transfer functions from S-domain transfer functions as ratios of rational polynomials in *z*
- Fortunately, there is a well known and well used approximation that allows the derivation of Z-domain transfer functions from S-domain transfer functions that produces ratios of rational polynomials in *z*. This is known as the bilinear S to Z transform.

$$s \approx \frac{2}{T}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right)$$

S to Z Mapping – Frequency Warping

- Note that by using the S to Z bilinear transform, any arbitrary S-domain transfer function represented as a ratio of rational polynomials in *s* will produce a Z-domain transfer function as a ratio of rational polynomials in *z* (try the harmonic oscillator as an example to convince yourself).
- The main artifact of the approximation is that it effectively warps the frequency axis so that the S-domain frequency range of minus infinity to plus infinity gets warped into a Z-domain frequency range of minus the Nyquist frequency to plus the Nyquist frequency.

$$\omega_a = \frac{2}{T}\tan\left(\frac{T}{2}\omega_d\right)$$
$$\omega_d = \frac{2}{T}\tan^{-1}\left(\frac{T}{2}\omega_a\right)$$

Where ω_a is the analog, or S-domain, frequency and ω_d is the digital, or Z-domain frequency.

S to Z Mapping – Frequency Warping

- A good description of the effects of the S to Z bilinear transform can be found in https://en.wikipedia.org/wiki/Bilinear_transform:

  "The discrete-time filter behaves at frequency ω_d the same way that the continuous-time filter behaves at frequency $^2/_T \tan(^T/_2\, \omega_d)$. Specifically, the gain and phase shift that the discrete-time filter has at frequency ω_d is the same gain and phase shift that the continuous-time filter has at frequency $^2/_T \tan(^T/_2\, \omega_d)$. This means that every feature, every "bump" that is visible in the frequency response of the continuous-time filter is also visible in the discrete-time filter, but at a different frequency."

BRTT

Kinematrics

S to Z Mapping – Frequency Pre-warping

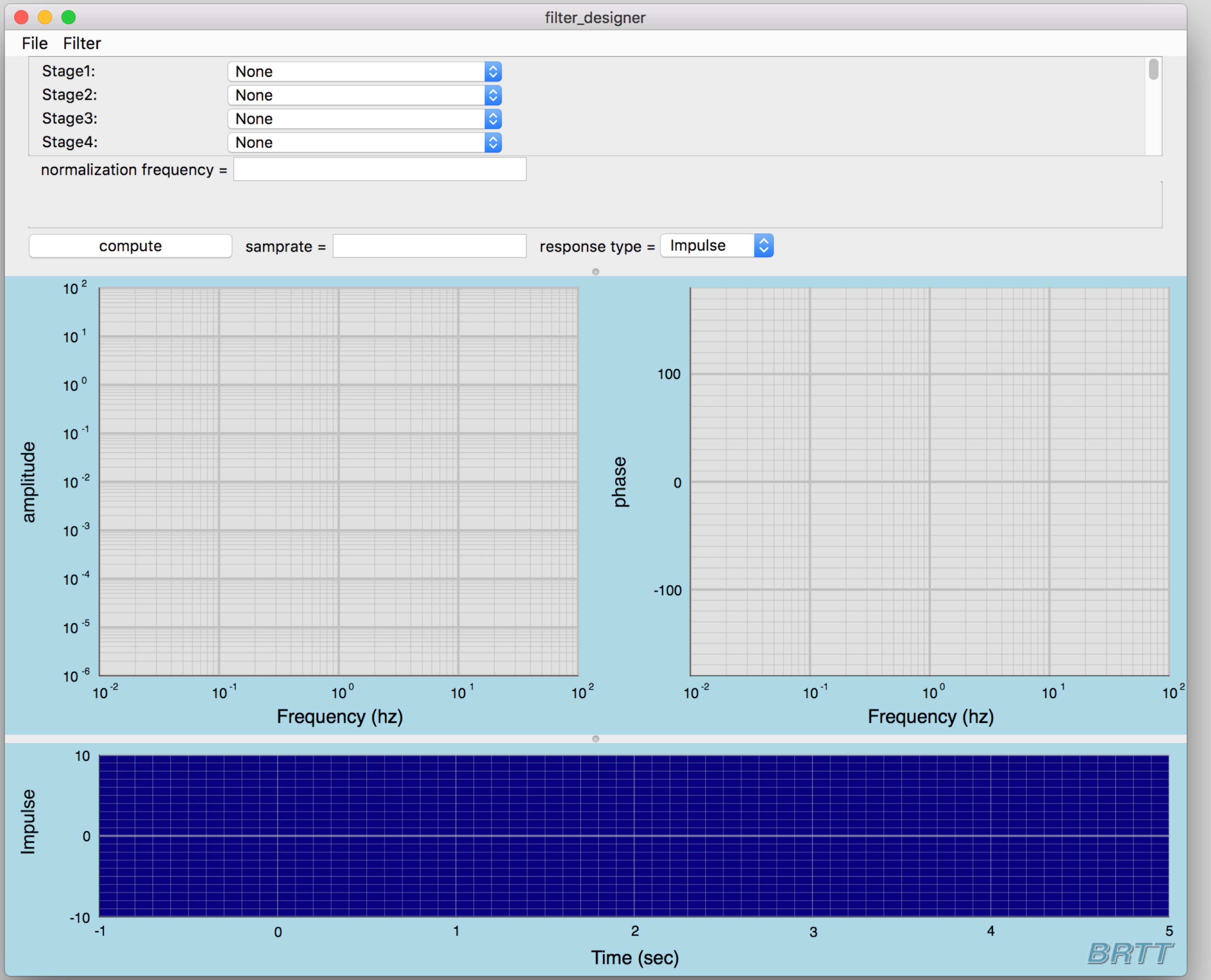
- In order to counteract the warping effects of the S to Z bilinear transform, pre-warping of critical frequency parameters in the filters, such as cutoff frequencies and resonance frequencies, are pre-warped using the warping relations so that the digital implementation of the filter will produce response spectra results that preserve the intent of the filter design.
- The S to Z bilinear transform and the pre-warping are done automatically by the Antelope libraries.
- Watch out for response stages that have infinite response at infinite frequency. Infinite response at zero frequency is ok.

BRTT

Kinemetrics

filter_designer

- Python script using the new Antelope **pythonbqplot(3Y)** python graphics libraries



filter_designer

File Filter

Stage1: None

Stage2: None

Stage3: None

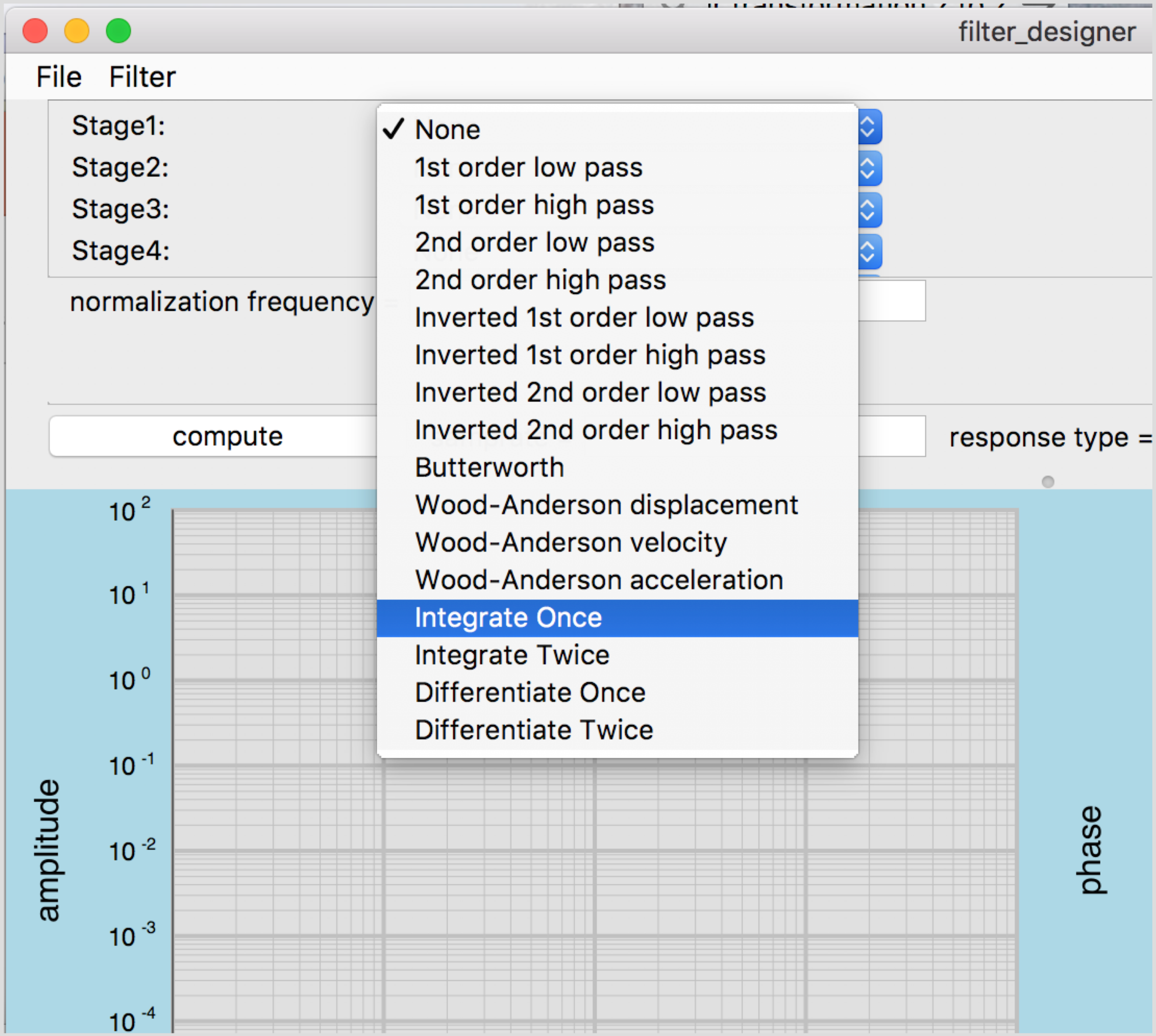
Stage4: None

normalization frequency =

compute

samprate =

response type = Impulse



filter_designer

File Filter

Stage1:
Stage2:
Stage3:
Stage4:

normalization frequency =

compute

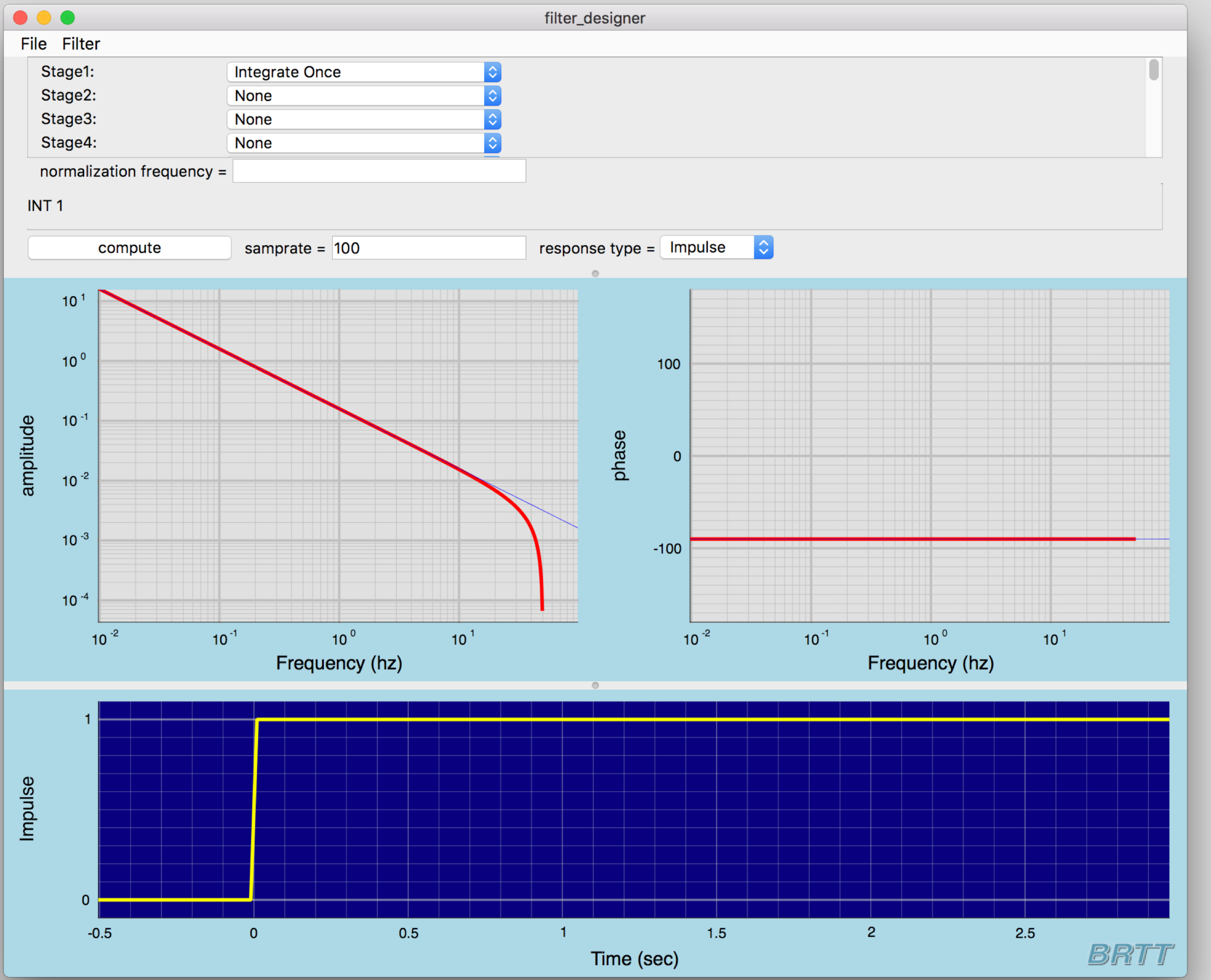
response type =

- ✓ None
- 1st order low pass
- 1st order high pass
- 2nd order low pass
- 2nd order high pass
- Inverted 1st order low pass
- Inverted 1st order high pass
- Inverted 2nd order low pass
- Inverted 2nd order high pass
- Butterworth
- Wood-Anderson displacement
- Wood-Anderson velocity
- Wood-Anderson acceleration
- Integrate Once
- Integrate Twice
- Differentiate Once
- Differentiate Twice

Filter Stages

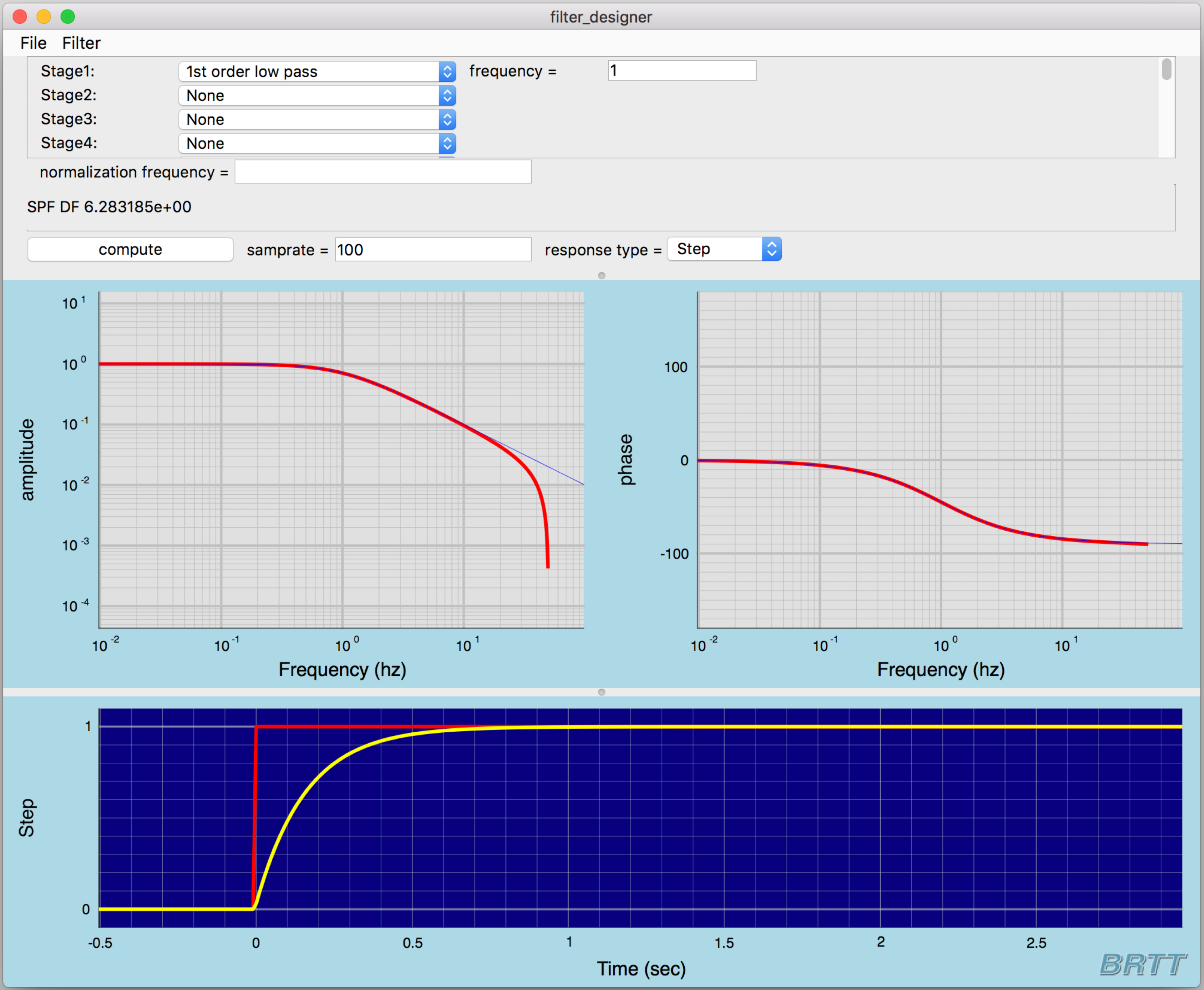
type	S-domain transfer function	Antelope filter string	Description
1st order low pass	$\frac{C}{s+C}$	`DF C`	first order denominator polynomial suitable as a zero frequency normalized first order low-pass filter
1st order high pass	$\frac{s}{s+C}$	`DFDIF1 C`	first order denominator polynomial with a single differentiation suitable as an infinite frequency normalized first order high-pass filter
2nd order low pass	$\frac{C}{s^2+Bs+C}$	`DS B C`	second order denominator polynomial suitable as a zero frequency normalized second order low-pass filter
2nd order high pass	$\frac{s^2}{s^2+Bs+C}$	`DSDIF2 B C`	second order denominator polynomial with a double differentiation suitable as an infinite frequency normalized second order high-pass filter
…			

- Filter stages are defined in **`wffilbrtt(3)`**



Stage1: Integrate Once
Stage2: None
Stage3: None
Stage4: None
normalization frequency =
INT 1
samprate = 100
response type = Impulse

- The red line is the digital Z-domain response. The thin blue line is the analog response. Note the effects of the frequency warping.



filter_designer

File Filter

Stage1: 1st order low pass frequency = 1

Stage2: None

Stage3: None

Stage4: None

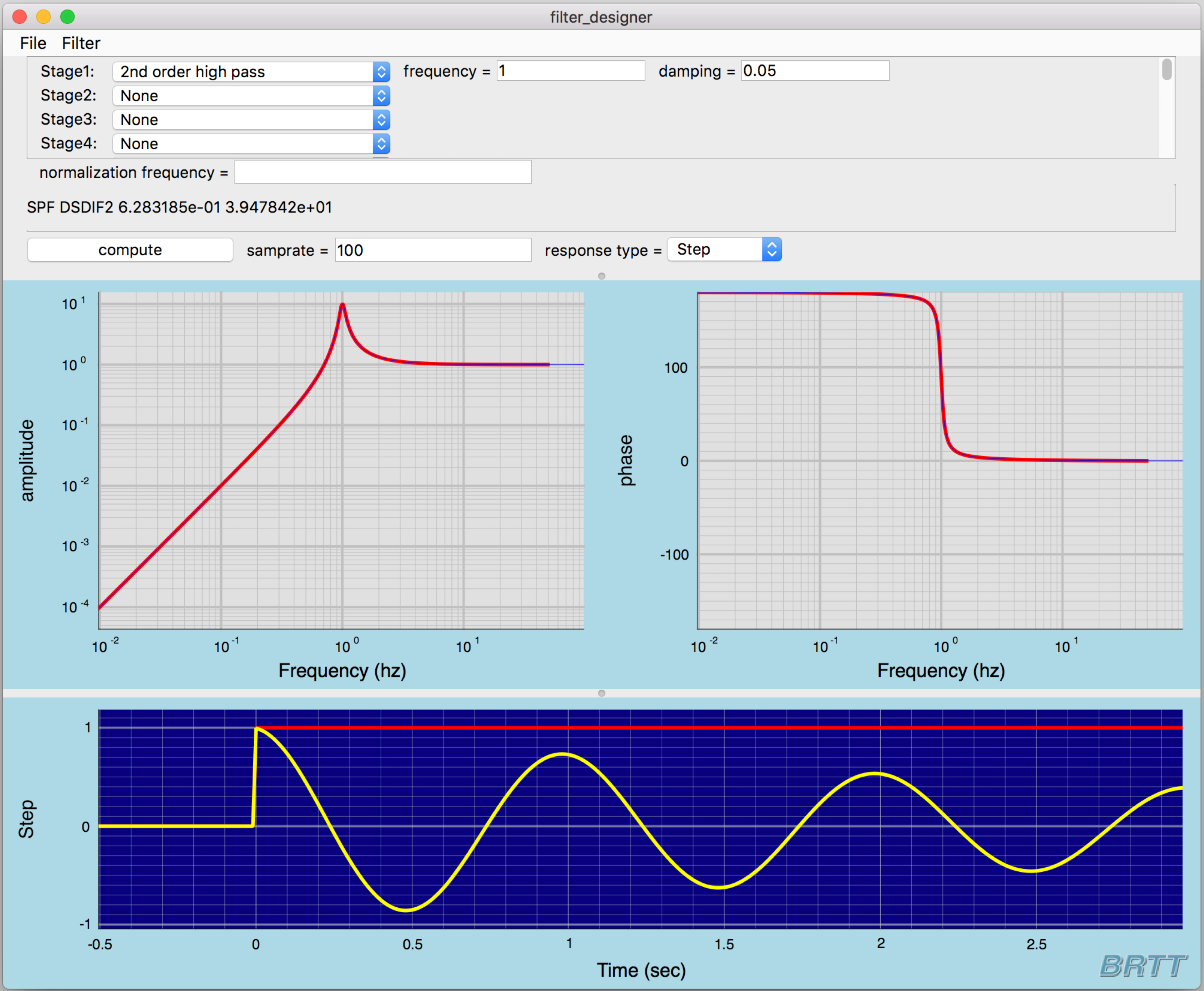
normalization frequency =

SPF DF 6.283185e+00

compute samprate = 100 response type = Step

BRTT

Kinemetrics



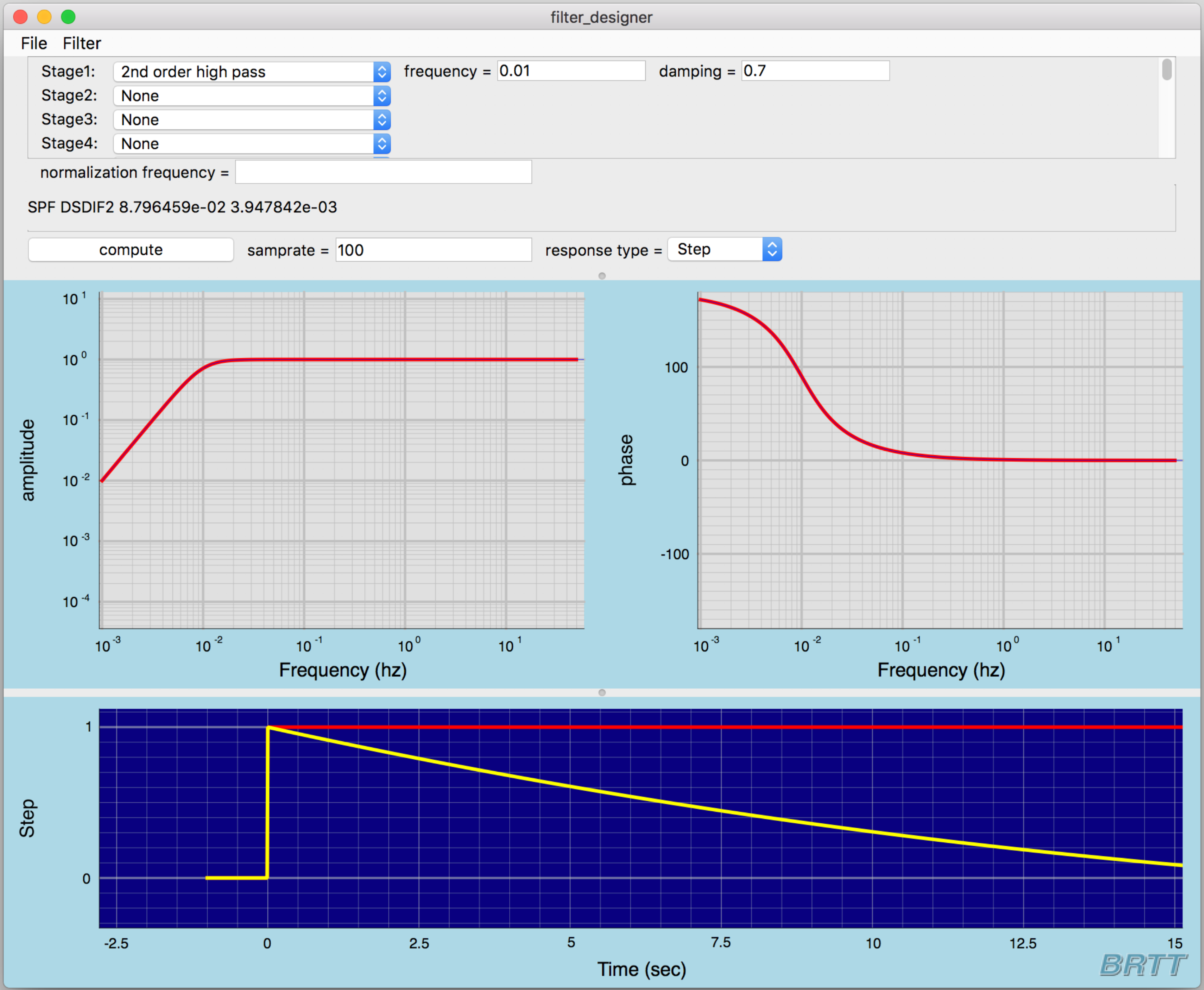
Stage1: 2nd order high pass frequency = 1 damping = 0.05
Stage2: None
Stage3: None
Stage4: None
normalization frequency =

```
SPF DSDIF2 6.283185e-01 3.947842e+01
```

samprate = 100 response type = Step

BRTT

Kinemetrics

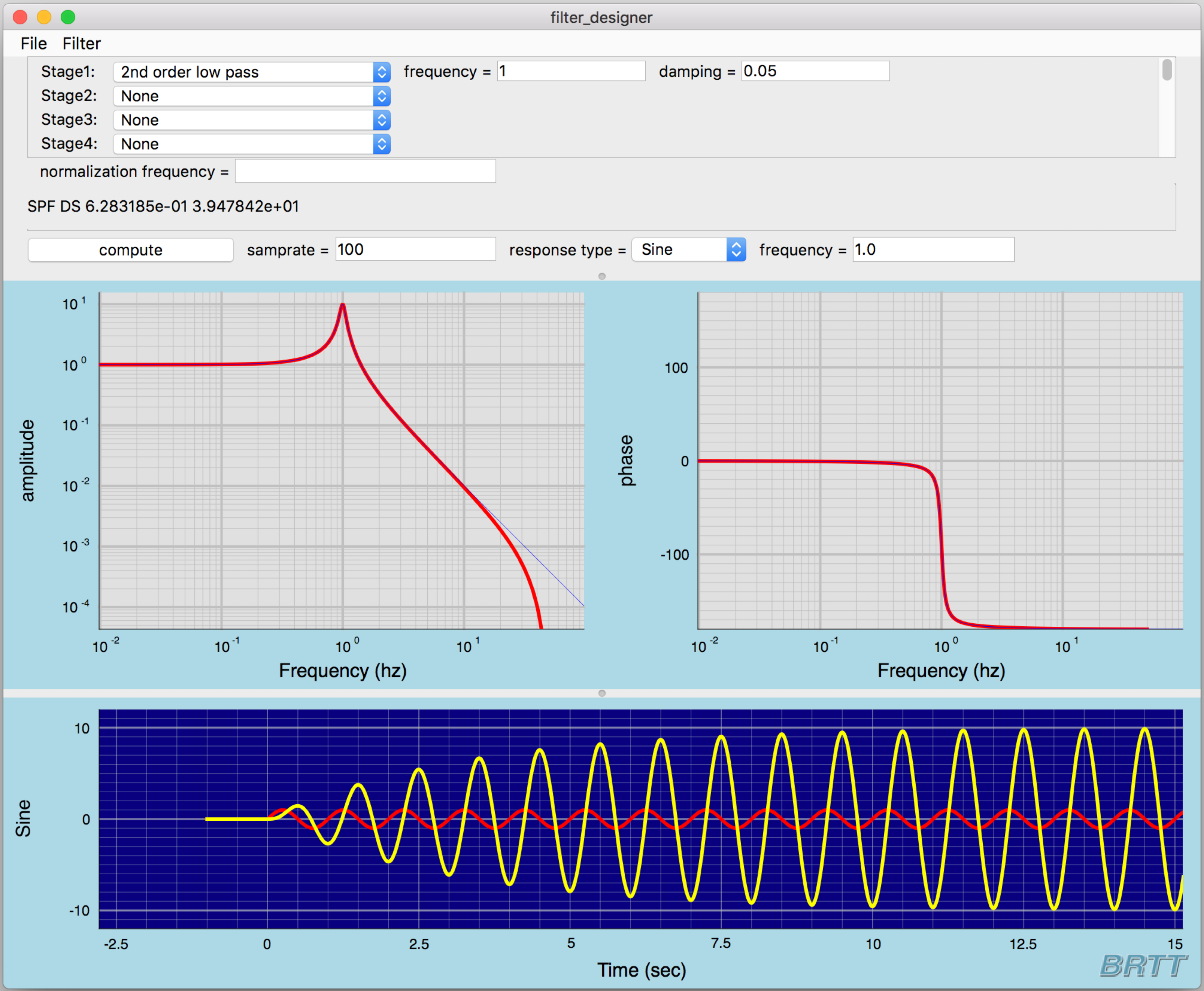

filter_designer

Stage1: 2nd order high pass frequency = 0.01 damping = 0.7
Stage2: None
Stage3: None
Stage4: None
normalization frequency =

```
SPF DSDIF2 8.796459e-02 3.947842e-03
```

samprate = 100 response type = Step

- Basic seismometer response (note the filter string).


filter_designer

File Filter

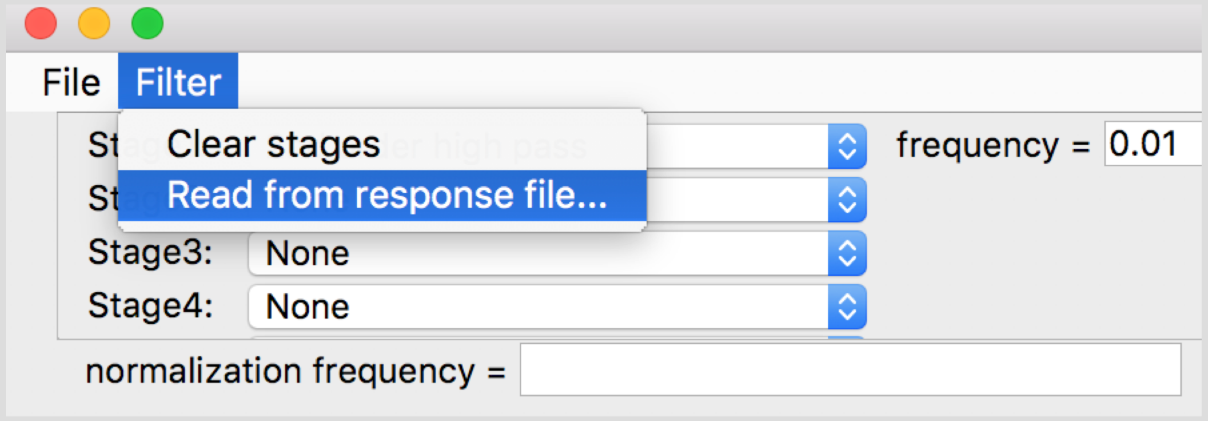
Stage1: 2nd order low pass frequency = 1 damping = 0.05
Stage2: None
Stage3: None
Stage4: None
normalization frequency =

```
SPF DS 6.283185e-01 3.947842e+01
```

compute samprate = 100 response type = Sine frequency = 1.0

- Strong motion response function.

- Grab real instrument responses

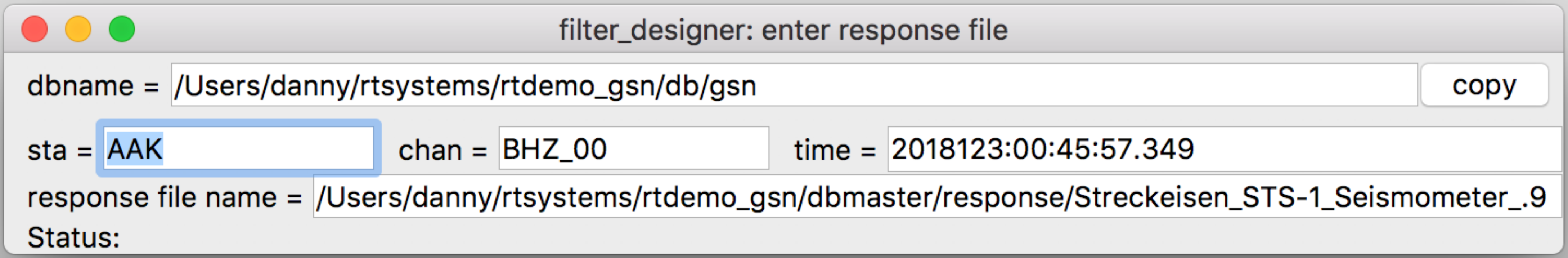


frequency = 0.01

Stage3: None

Stage4: None

normalization frequency =


filter_designer: enter response file

dbname = /Users/danny/rtsystems/rtdemo_gsn/db/gsn

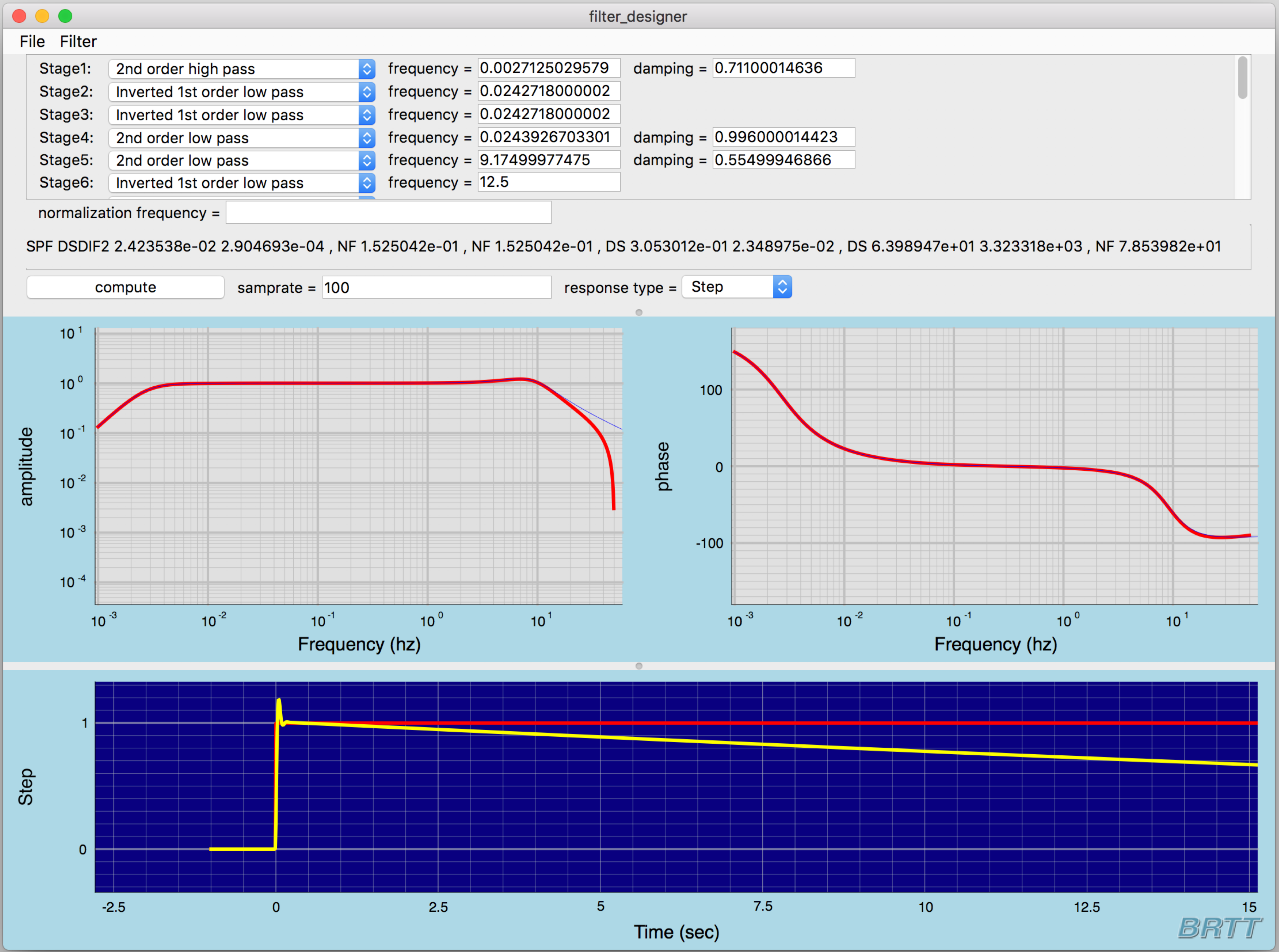
sta = AAK   chan = BHZ_00   time = 2018123:00:45:57.349

response file name = /Users/danny/rtsystems/rtdemo_gsn/dbmaster/response/Streckeisen_STS-1_Seismometer_.9

Status:

BRTT

Kinemetrics



filter_designer

File Filter

Stage1: 2nd order high pass frequency = 0.0027125029579 damping = 0.71100014636
Stage2: Inverted 1st order low pass frequency = 0.0242718000002
Stage3: Inverted 1st order low pass frequency = 0.0242718000002
Stage4: 2nd order low pass frequency = 0.0243926703301 damping = 0.996000014423
Stage5: 2nd order low pass frequency = 9.17499977475 damping = 0.55499946866
Stage6: Inverted 1st order low pass frequency = 12.5

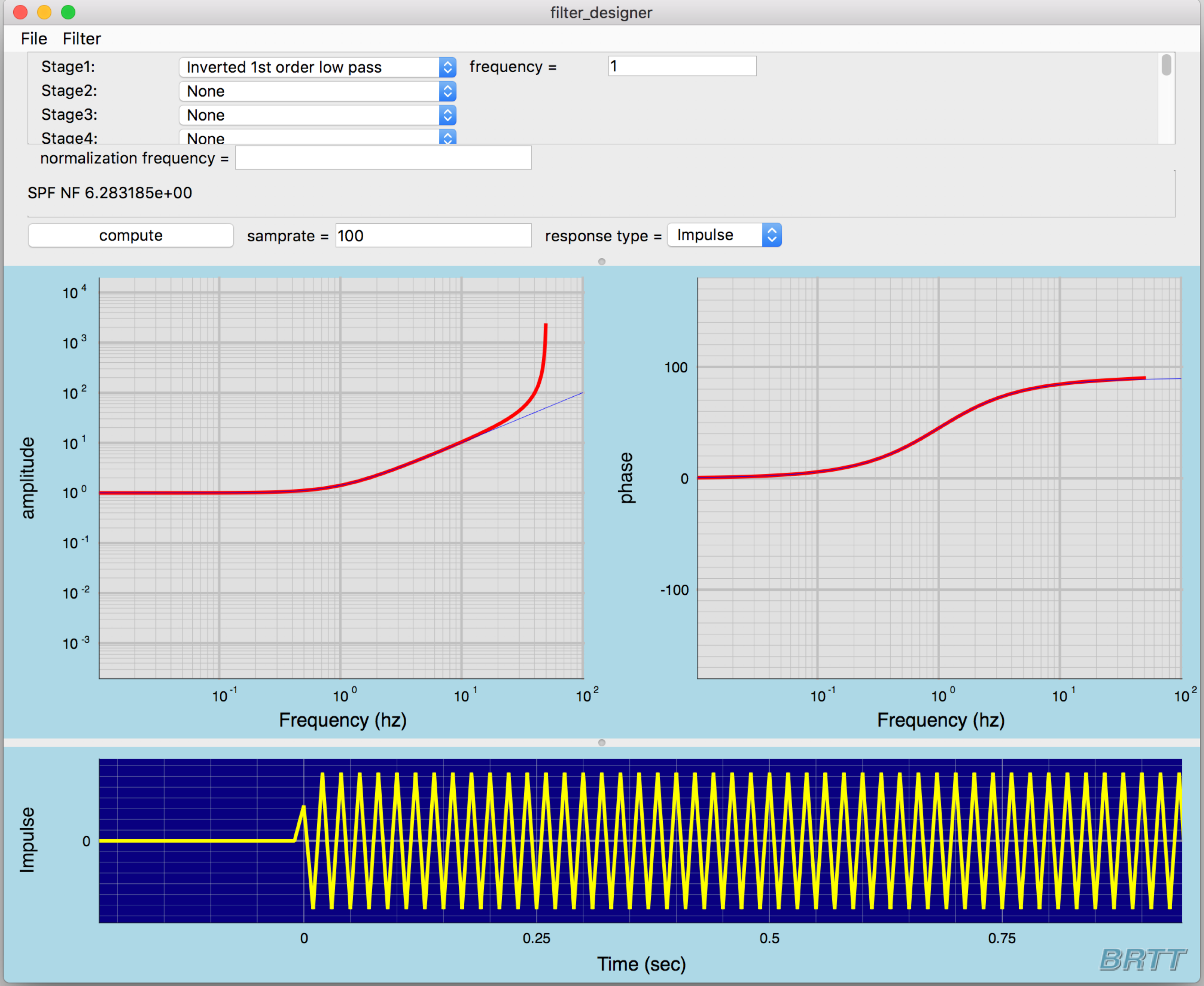
normalization frequency =

```
SPF DSDIF2 2.423538e-02 2.904693e-04 , NF 1.525042e-01 , NF 1.525042e-01 , DS 3.053012e-01 2.348975e-02 , DS 6.398947e+01 3.323318e+03 , NF 7.853982e+01
```

compute samprate = 100 response type = Step

BRTT

Kinematics


filter_designer

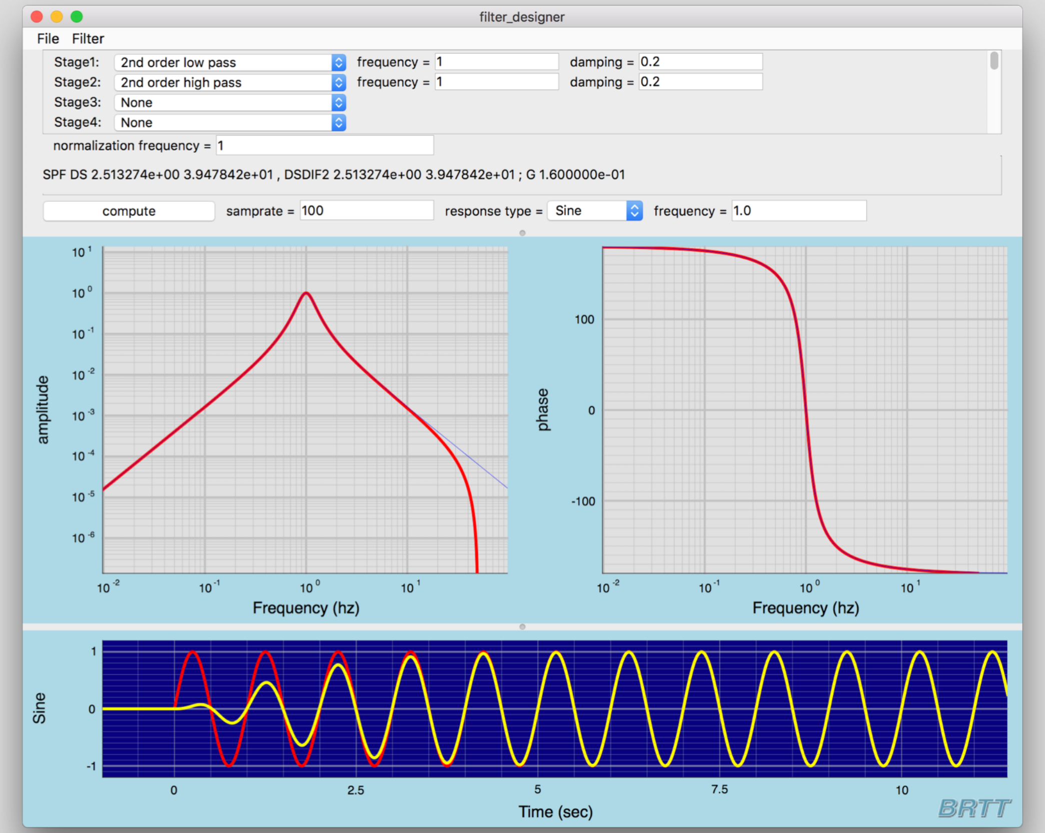
File Filter

Stage1: Inverted 1st order low pass frequency = 1
Stage2: None
Stage3: None
Stage4: None
normalization frequency =

SPF NF 6.283185e+00

compute samprate = 100 response type = Impulse

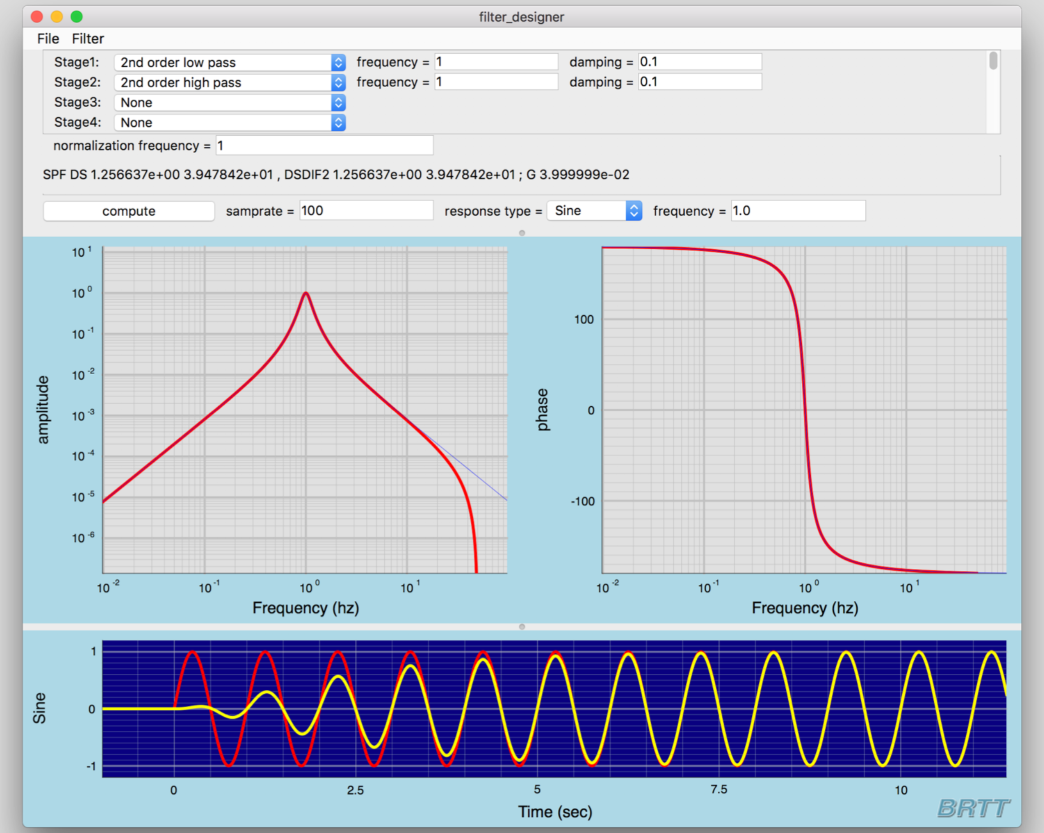
- Inverted filter stages are inherently unstable. They should only be used in combination with non-inverted filter stages.



Stage1: 2nd order low pass frequency = 1 damping = 0.2
Stage2: 2nd order high pass frequency = 1 damping = 0.2
Stage3: None
Stage4: None
normalization frequency = 1

```
SPF DS 2.513274e+00 3.947842e+01 , DSDIF2 2.513274e+00 3.947842e+01 ; G 1.600000e-01
```

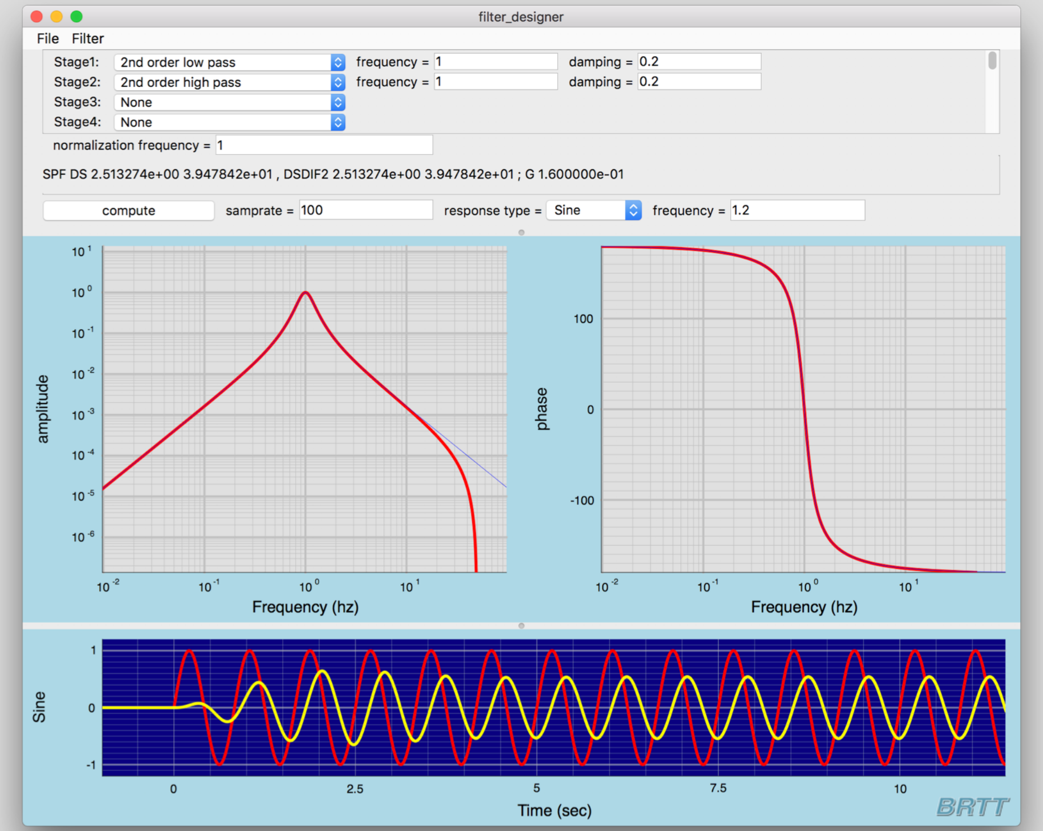
compute samprate = 100 response type = Sine frequency = 1.0



Stage1: 2nd order low pass frequency = 1 damping = 0.1
Stage2: 2nd order high pass frequency = 1 damping = 0.1
Stage3: None
Stage4: None
normalization frequency = 1

```
SPF DS 1.256637e+00 3.947842e+01 , DSDIF2 1.256637e+00 3.947842e+01 ; G 3.999999e-02
```

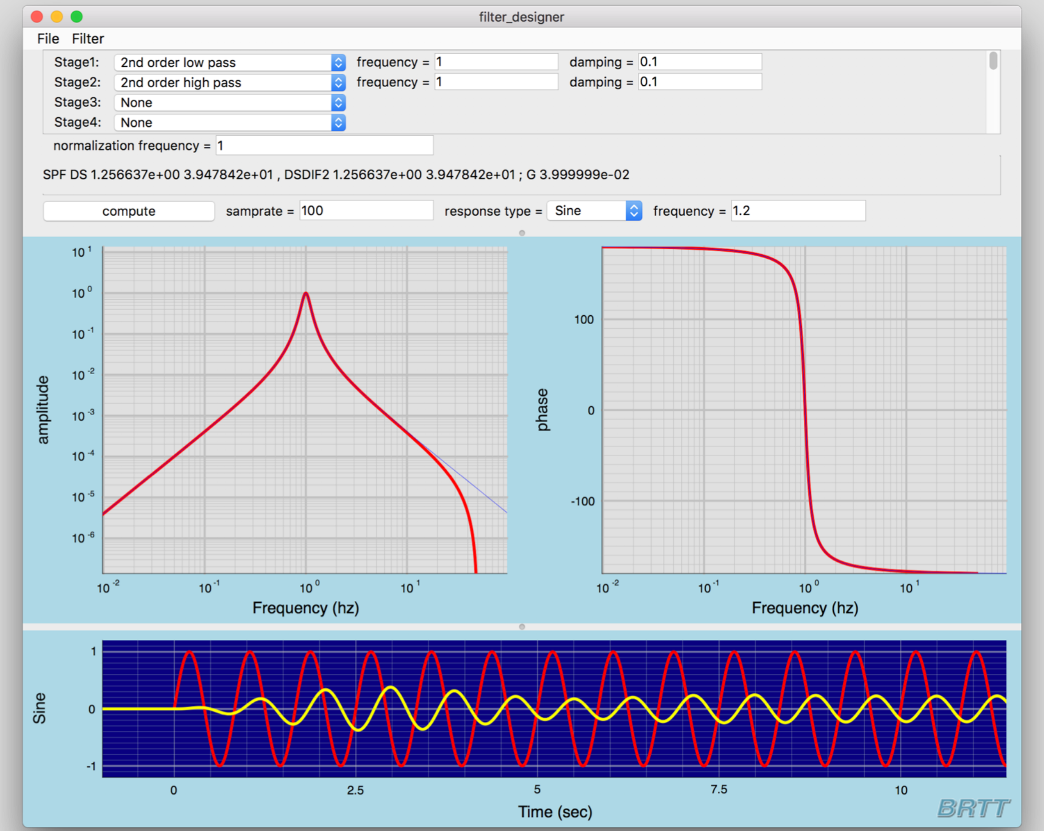
compute samprate = 100 response type = Sine frequency = 1.0



Stage1: 2nd order low pass frequency = 1 damping = 0.2
Stage2: 2nd order high pass frequency = 1 damping = 0.2
Stage3: None
Stage4: None
normalization frequency = 1

```
SPF DS 2.513274e+00 3.947842e+01 , DSDIF2 2.513274e+00 3.947842e+01 ; G 1.600000e-01
```

compute samprate = 100 response type = Sine frequency = 1.2



Stage1: 2nd order low pass frequency = 1 damping = 0.1
Stage2: 2nd order high pass frequency = 1 damping = 0.1
Stage3: None
Stage4: None
normalization frequency = 1

```
SPF DS 1.256637e+00 3.947842e+01 , DSDIF2 1.256637e+00 3.947842e+01 ; G 3.999999e-02
```

compute samprate = 100 response type = Sine frequency = 1.2