iirrational Documentation

Release 0.9.0rc6

Lee McCuller

Mar 04, 2018

Contents

Release v0.9.0rc6. (*Installation*)

IIRrational is an function-fitting library for signal processing and system identification. Fitting both poles and zeros for a rational function is a nonlinear optimization problem and typical formulations require an initial guess for convergence.

This library uses a two stage approach, where the second stage is a typical gradient descent optimization, but the first is a linear technique with fast and reliable convergence, but requires gratuitous over-fitting.

CHAPTER 1

Features and Conveniences

- No initial guess required
- Automatic order reduction (using heuristics)
- No poles or zeros with frequencies above the last data-point.
- guaranteed stable poles (cannot currently be circumvented)
- uses SNR weight vector
- generates fisher matrix / covariance matrix of parameterizations
- propagates errorbars to the fit
- Z-domain for implementable filters
- matlab support through the msurrogate package (TODO link)
- Nonlinear optimization toolkit for rational functions
- direct output of second-order-sections (biquadratic filters)

```
import iirrational.v1
import iirrational.plots
from iirrational.testing import iirrational_data

dataset = iirrational_data('simple2')
fit = iirrational.v1.data2filter(
    data = dataset.data,
    F_Hz = dataset.F_Hz,
    SNR  = dataset.SNR,
    F_nyquist_Hz = 16384,
)
ax = iirrational.plots.plot_fitter_flag(fit.fitter)
```

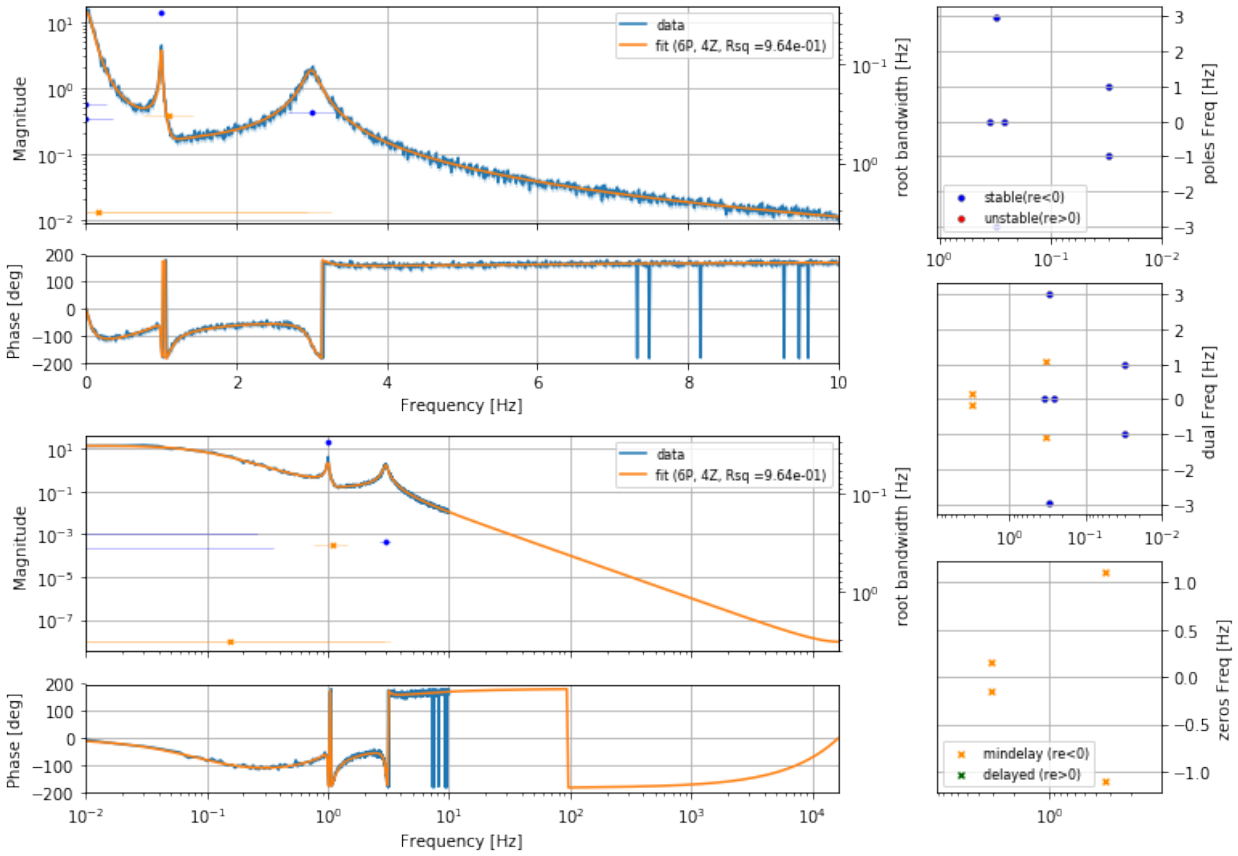


Fig. 1.1: Easy Low order example with well-behaved data. Fits in 4s.

CHAPTER 2

Contents

2.1 Installation

2.1.1 Installing IIRrational

Distribute & Pip

The recommended method of installation is through pip:

```
$ pip install iirrational
```

or if you are not administrator, you may do a –user install:

```
$ pip install iirrational --user
```

2.1.2 Download the Source

You can also install IIRrational from source. The latest release (0.9.0rc6) is available from GitHub.

- git
- tarball
- zip

Once you have a copy of the source, you can embed it in your Python package, or install it into your site-packages.

```
$ python setup.py install
```

To download the full source history from Git, see Source Control.

to adjust the source and run from the code directory, install via:

```
$ python setup.py develop
```

or better yet, use pip for this and get the dependencies as well:

```
$ pip install -e ./
```

alternatively, add to PYTHONPATH (be sure to also install or add the dependencies):

```
$ export PYTHONPATH="/path/to/iirrational:$PYTHONPATH"
```

Warning: Installing using setup.py develop will not expose the package to Matlab. Use the PYTHONPATH install instead.

Staying Updated

The latest version of IIRrational will available at:

- PyPi: http://pypi.python.org/pypi/iirrational/
- GitHub: http://github.com/mccullerlp/iirrational/

When a new version is available, upgrading using:

```
$ pip install iirrational --upgrade
```

Note: This can upgrade dependencies such as numpy and scipy as well!

2.2 Quickstart

Install IIRrational through pip as a first step. Be sure to run pip from the environment you will be using IIRrational from.

2.2.1 First Fit

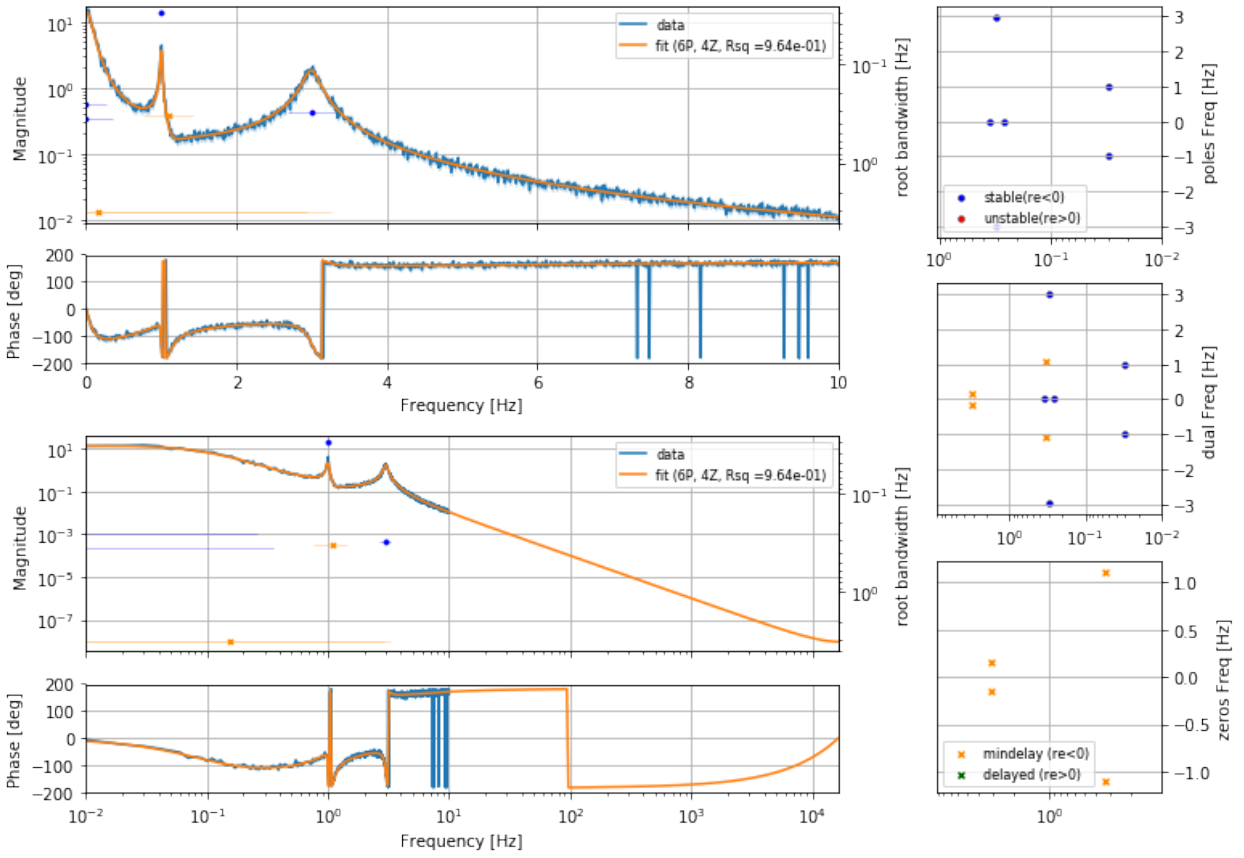
As seen in the introduction page, the minimal code to fit and see it is:

```
import iirrational.v1
import iirrational.plots
from iirrational.testing import iirrational_data

dataset = iirrational_data('simple2')
fit = iirrational.v1.data2filter(
    data = dataset.data,
    F_Hz = dataset.F_Hz,
    SNR  = dataset.SNR,
    F_nyquist_Hz = 16384,
)

#see the ZPK generated (Z-domain with the provided Nyquist frequency)
print(fit.fitter.ZPK)
```

```
#plot the output
ax = iirrational.plots.plot_fitter_flag(fit.fitter)
```



This example shows a few notable aspects of the library.

1. The main fitter code is in a "v1" submodule. This is to provide some versioning between the methods and heuristics used. Hopefully the library becomes monotonically better, but without a formal problem statement, that is impossible to define.

   For this reason, the composite all-in-one fitting methods will be versioned. For versions greater than the current library version, the API and heuristics will not be stable, for version submodules equal or below the library version, it should remain stable (this may require migrating common code into the submodule as common code changes).

2. The return object is composite, with a "fitter" attribute holding the current best fit, the data, SNR and everything in the plot.

3. A Nyquist frequency is required. This should be one half the sample frequency. Currently S-domain output is not supported (will be in the future). The ZPK output

4. Plotting functions are provided to conviently assess the quality of fit. The "flag" plots show top-left a zoom to the data, linear if the data is linear spaced, and log otherwise.

   The lower left is a zoom out to the Nyquist frequency to see if the rolloff is well-matched. Since this is Z-domain the phase will always reture to 0 or 180 at the Nyquist (thanks Cauchy). The right line are Pole, PZ, Zero plots in the S-domain (really the Z-domain zoomed in near 1 and conformally remapped..). These are also overlayed in

the loglog plots using the right y-axis. These plots are log-scaled in the bandwidth (S root real) and right-plane roots are mirrored and recolored.

5. The library includes some example data generators through the testing submodule

Example Data

Example data generators are useful for the test-suite and for this documentation. Most of them are randomly generated filters multiplied by noise consistent with a constant SNR. These data are very well behaved compared to real data in having constant, well-estimated SNR. If the library is run from its development folder, an additional module `iirrational_test_data` is available with contributed real data test cases. To reduce size these are not distributed through pip/setuptools.

You can see the available cases at

- internal https://github.com/mccullerlp/iirrational/blob/master/iirrational/testing/__init__.py
- contrbuted https://github.com/mccullerlp/iirrational/blob/master/iirrational_test_data/__init__.py

these are accessed through the functions `iirrational.testing.iirrational_data()` and `iirrational_test_data.iirrational_data_dev()` using a keyword set name. The random sets also take a `set_num` argument and `instance_num`. For the random sets, the set_num changes the last data point (10Hz to 100Hz) and also linear vs. log distribution. The instance number seeds the random noise added, so that the fitting is deterministic for a given instance.

The *`data2filter()`* function can take as its first positional argument a dictionary with the other keywords. This allows the sets to be fit and tested using only:

```
import iirrational.v1
import iirrational.plots
from iirrational.testing import iirrational_data

fit = iirrational.v1.data2filter(
  iirrational_data('simple2'),
)
ax = iirrational.plots.plot_fitter_flag(fit.fitter)
```

2.2.2 Fitting

`iirrational.v1.`**`data2filter`**`()`

Fits frequency-response data using 2-stage process

Parameters

- **`argB`** (*`Mapping`*) – optional positional argument with dict-like full of keyword arguments. Explicit arguments supercede the ones provided by this argument.
- **`data`** (*`ndarray`*) – Complex array containing transfer-function to fit.
- **`F_Hz`** (*`ndarray`*) – Real array with frequency points of each data point.
- **`SNR`** (*`ndarray or float`*) – Array of weights for the data. May be a single value. Zeros allowed, NaN's and infinities will break the fit.
- **`F_nyquist_Hz`** (*`float`*) – Value to use for Nyquist frequency of Z-domain representation.

- **ZPK** (*[Z-tuple, P-tuple, float]*) – Optional! If provided, the stage-1 will be skipped and only the nonlinear fitting and order reduction will be applied
- **order_initial** (*int*) – Optional order to stage-1 fitting. If not provided the order is scaled up exponentially until residuals stabilize. This should be higher than the known system order as stage-1 warps the domain and needs extra order to compensate for systematics.
- **SNR_cutoff** (*float*) – defaults to 100. Clips SNR to prevent over-emphases on featureless parts of the data. Coherence SNR estimates can over-weight data.
- **delay_s** – If None the delay is fit out near the end. This is often degenerate with the fit order unless the data extends to high frequencies. Use 0 otherwise. Not yet implemented is to use the specified value
- **hints** (*dict or list-of-dicts*) – Hierarchical hint keywords to affect internal algorithms, verbosity, and more.

Returns FitAid collection

Fitter Output (FitAid)

The output of the fitter is a container that holds the best fit, the hint dictionary, some methods used for evaluating fits, and the annotation log. All of the internal algorithm functions interface with it for settings and to store their improvements to the fit. As a user, only the stored fits are useful. In the future some interfaces for finding hints and their default values and hierarchy will be exposed.

The additional fitters stored which have the lowest residuals seen are usable during the order-reduction algorithm for robust statistics as to how well the fitting CAN do vs. what is sacrificed by order reduction.

class v1.**FitAid**

Stores and exposes the output of fitting algorithms. Exposed attributes to use are:

fitter

A `MultiReprFilter` (MRF) Object with the current best fit. This object exposes the current residuals, errorbars and parameterization. The most immediately useful attribute of it is *ZPK* to get the Z-domain root.

fitter_lowres_avg

MRF type with the lowest average residuals seen (may not be order-reduced).

fitter_lowres_max

MRF type with the lowest max (L-infinity) residuals seen (may not be order-reduced).

fitter_lowres_med

MRF type with the lowest median residuals seen (may not be order-reduced).

fitter_loword

MRF type with the lowest order seen.

digest_write()

method to write a digest of the algorithm internals (described below).

Hints

The algorithms internally use a hierarchical hinting system to adjust thresholds and settings. The semantics for these hints are not yet well established, but some useful ones are:

log_print Specify whether to print log output. Set to False to reduce the stuff printed (which is internal algorithm diagnostics)

optimize_log Whether to log on the `MultiReprFilter.optimize()` calls. Gives the nonlinear optimizer termination conditions, be they ftol, xtol, gtol, ran past maxfevs and such.

optimize_max_nfev The number of function evaluations allowed during a nonlinear optimization call

optimize_ftol The relative tolerance in the nonlinear optimization calls. Set larger to trade speed for accuracy (the default has not be tuned)

resavg_RthreshOrdDn The *relative* threshold on the ratio of the new residual chi-sqaure from a previous residual for an order reduction. Defaults to 1.10. Increase for more aggressive reduction

resavg_RthreshOrdUp relative threshold for heuristics that increase the order. Defaults to 1 but less may make sense

resavg_RthreshOrdC relative threshold for heuristics that preserve the order.

resavg_EthreshOrdDn threshold for the Exact residual average value to accept an order reduction. Makes sense for data without noise where the residual is numeric precision or exactness-of-representation limited (designed in S, fit in Z for example)

The can be specified in a dictionary, and there is a collection of "good" sets for specific purposes. An example usage for a quiet exact fit is.

```
import iirrational.v1
import iirrational.plots
from iirrational.testing import iirrational_data

fit = iirrational.v1.data2filter(
  iirrational_data('simple0E'),
  SNR = 1,
  hints = [
      {'resavg_EthreshOrdDn': 0.1,
       'resavg_RthreshOrdC':  None,
       'resavg_RthreshOrdDn': None,
       'resavg_RthreshOrdUp': None},
      iirrational.v1.hintsets.quiet,
  ]
)
ax = iirrational.plots.plot_fitter_flag(fit.fitter)
```

Gotchas

Since this library attempts to forgo initial guesses, it must have some basis for order estimation. This means that for datasets with features that haven't been tested against, the fitter may give objectionable fits, often missing features because its initial order was too low or its reduction heuristics too aggressive.

Generally the fits are improved if `order_initial` is specified with a large value. 50-100 roots is a benchmark that seems to always get good fits, much larger and the fits slow down and become numerically less stable. The downside is that this slows down fits that don't need it, and the order reduction has to work harder, and it is not yet particularly effective.

2.2.3 Plotting

Todo: More docs of the various plotters, how to save and how to enable/disable automatic interactive plots (particularly for matlab users).

Annotated Digest

The data2filter output `FitAid` object can output a digest with plots of the internal steps of the algorithm. This is mostly useful for developing the heuristics, but useful to tech some methods. Could be used to generate logbook posts. Inspect the code behind data2filter to learn how to use it.

See the end of the *Random Filter fit with comparison* note for an example call. And the digest output can be seen in the notebook as it uses the `ipy_digest` argument.

`iirrational.v1.FitAid.`**`digest_write`**`()`
Write a digest of the fit algorithm in Markdown

Parameters

- **`folder`** – First positional with the folder to dump the plots and markdown file
- **`format`** – Currently must be the (default) "Markdown"
- **`md_name`** – Name of the markdown file within the folder. Defaults to "digest.md"
- **`plot_verbosity`** – Verbosity tags above this will not be plotted
- **`md_verbosity`** – Verbosity tags above this will not have a section
- **`clear_plots`** – Delete original plots if not over-plotted
- **`regex_plotsections`** – Only plot in sections meeting this regex (for algorithm debugging)
- **`ipy_display`** – Dump the markdown into the ipython notebook
- **`plot_format`** – Format to write the plots. Default is "png". Pdf is possible but most markdown renderers don't support it.
- **`dpi`** – DPI for plots
- **`MP_workers`** – 1 is default. If larger it uses multiprocessing. Matplotlib sometimes misbehaves and clips plots strangely when above 1.

2.2.4 Contributing Testcases

To improve the library, more examples of real-world usage are needed. A function call with nearly the same interface as *`data2filter()`* exists for this purpose. Please email, add an issue or pull request with collections of data. The Matlab format from this is relatively compact and easy to read, so please use this function to provide data. If you have larger test-suites for evential MIMO or Feedforward testing, hdf5 format is preferred but a standard layout has not yet been established.

`iirrational.v1.`**`data2testcase`**`()`
Store fit-data in a common format for inclusion into the test-suite

Parameters

- **`argB`** – mapping of arguments for packaged data
- **`data`** – complex array to fit
- **`F_Hz`** – array of frequencies for data
- **`SNR`** – float or array of SNR weights
- **`fname`** – filename to write. Should include the ".mat" format
- **`F_nyquist_Hz`** – Optional but highly recommended best nyquist frequency for problem domain. Used when a Z bestfit ZPK is provided.

- **bestfit_ZPK_z** – Optional ZPK in Z domain with a reference fit to compare against. Should be a "good" fit, rather than a failed output of data2filter.
- **bestfit_ZPK_s** – Optional ZPK in S domain with a reference fit to compare against. Should be a "good" fit.
- **F_ref_Hz** – Reference frequency for S-to-Z conversion. Defaults to 0 but this is inappropriate for high-DC-gain filters.
- **description** – A description of the dataset source, aspects to preserve or anything of relevance for testers.

2.2.5 Example Notebooks

Use the "view page source" link at the top to download the original .ipynb file to run yourself.

Noiseless Data Fitting

The default heuristics for noisy models must be modified for noiseless data. Those heuristics primarly check for relative changes to the average residuals. For noiseless data the residuals can be at numeric precision and so the checks should be done using exact thresholds rather than relative thresholds.

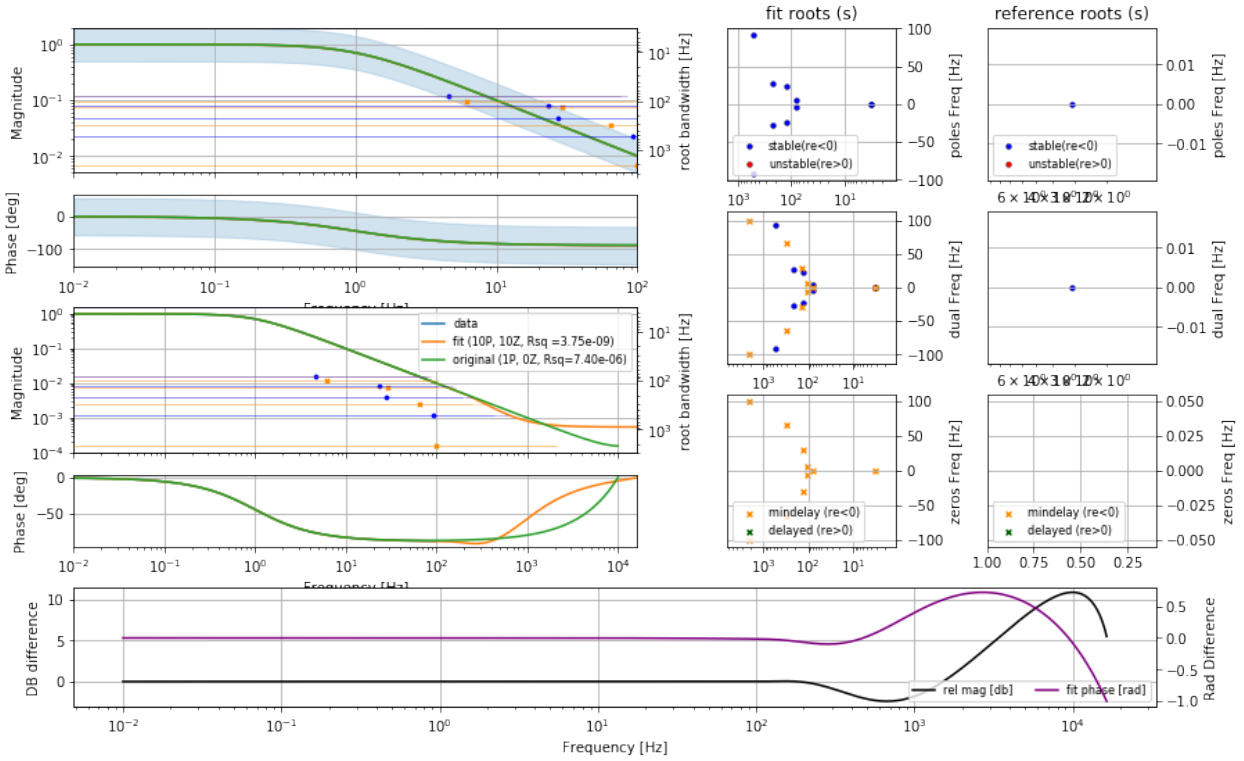
Also for exact fits the nyquist frequency must be chosen well. Exact models from the s domain will have slightly modified phase response when run at a finite nyquist frequency. The fits will choose higher order to fit out this modified phase. If the nyquist is sufficiently high and some fit error is allowed, then the order will be reduced.

- *The order reduction for this is not working well (it worked better in the past). Needs some attention.*

```
In [19]: #tdat =  iirrational_data('rand2_log100E', set_num = 1)
         tdat =  iirrational_data('simple0E', set_num = 3)
         out = v1.data2filter(
             tdat,
             SNR = 1,
             F_nyquist_Hz = 16384,
         )
         ax = plot_fitter_flag_compare(out.fitter, tdat.fitter)
```

```
(direct = 4.379e+01, Psvd= 4.379e+01, Zsvd= 4.379e+01)
LINEAR Final Residuals:  0.003905684042473498
(direct = 4.079e+01, Psvd= 4.079e+01, Zsvd= 4.079e+01)
LINEAR Final Residuals:  21461.937977934154
Using last (direct)! 20
Cleared  pole (-1.0000000000000056+0j)
Cleared  pole (-0.9337219070053355+0j)
Cleared  zero (-2.022799388440665+0j)
Cleared  zero (-0.9999999999999999+0j)
Initial Order: (Z= 18, P= 18, Z-P= 0)
TRIPLETS (rat = 1.0096714011831398, pre = 2.9715222030734957e-08, mid = 2.9695964754453158e-08, post
N:  2
[(0.9937908350719397+0.0011628065503476077j), (0.9937908350719397-0.0011628065503476077j)] zeros
[(0.9917854823999889+0.005584742537756558j), (0.9917854823999889-0.005584742537756558j)] zeros
[(0.9810664341215537+0.012280189950038723j), (0.9810664341215537-0.012280189950038723j)] zeros
[(0.871123415773924+0.01670562965952297j), (0.871123415773924-0.01670562965952297j)] zeros
[(0.9922880282193801+0.004460262849743871j), (0.9922880282193801-0.004460262849743871j)] poles
[(0.9865713702915061+0.00518825000781355j), (0.9865713702915061-0.00518825000781355j)] poles
[(0.9678518718336526+0.017061129644049967j), (0.9678518718336526-0.017061129644049967j)] poles
WEAK REMOVE:  [(0.9917854823999889+0.005584742537756558j), (0.9917854823999889-0.005584742537756558j)
```

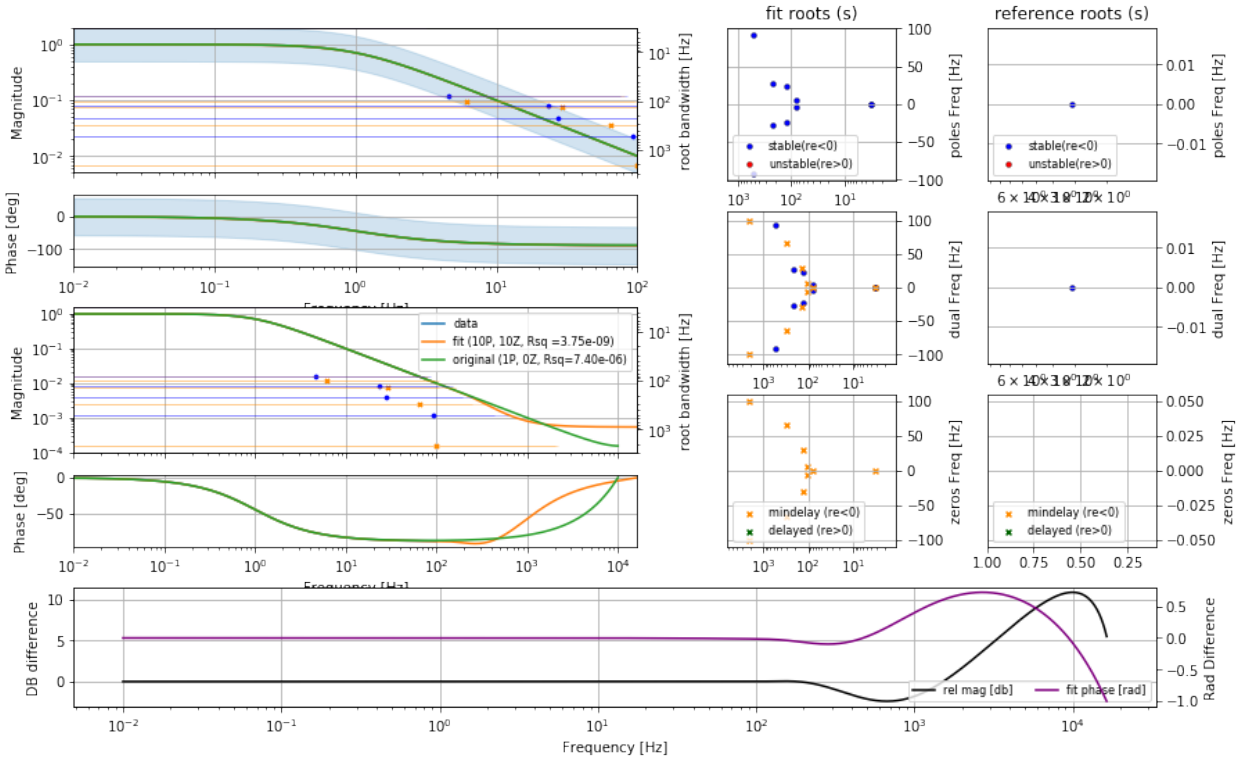
```
RATIO for WEAK:  7.761149887267759
FINAL RESIDUALS 3.746302208870283e-09
```




```
In [20]: out = v1.data2filter(
             tdat,
             SNR = 1,
             F_nyquist_Hz = 16384,
             hints = [
                 v1.hintsets.exact_data,
             ],
         )
         ax = plot_fitter_flag_compare(out.fitter, tdat.fitter)
```

```
(direct = 4.379e+01, Psvd= 4.379e+01, Zsvd= 4.379e+01)
LINEAR Final Residuals:  0.003905684042473498
(direct = 4.079e+01, Psvd= 4.079e+01, Zsvd= 4.079e+01)
LINEAR Final Residuals:  21461.937977934154
Using last (direct)! 20
Cleared  pole (-1.0000000000000056+0j)
Cleared  pole (-0.9337219070053355+0j)
Cleared  zero (-2.022799388440665+0j)
Cleared  zero (-0.9999999999999999+0j)
Initial Order: (Z= 18, P= 18, Z-P= 0)
TRIPLETS (rat = 1.0096714011831398, pre = 2.9715222030734957e-08, mid = 2.9695964754453158e-08, post
N:  2
[(0.9937908350719397+0.0011628065503476077j), (0.9937908350719397-0.0011628065503476077j)] zeros
[(0.9917854823999889+0.005584742537756558j), (0.9917854823999889-0.005584742537756558j)] zeros
[(0.9810664341215537+0.012280189950038723j), (0.9810664341215537-0.012280189950038723j)] zeros
[(0.871123415773924+0.01670562965952297j), (0.871123415773924-0.01670562965952297j)] zeros
[(0.9922880282193801+0.004460262849743871j), (0.9922880282193801-0.004460262849743871j)] poles
[(0.9865713702915061+0.00518825000781355j), (0.9865713702915061-0.00518825000781355j)] poles
[(0.9678518718336526+0.017061129644049967j), (0.9678518718336526-0.017061129644049967j)] poles
WEAK REMOVE:  [(0.9917854823999889+0.005584742537756558j), (0.9917854823999889-0.005584742537756558j)
```

```
RATIO for WEAK:  7.761149887267759
FINAL RESIDUALS 3.746302208870283e-09
```

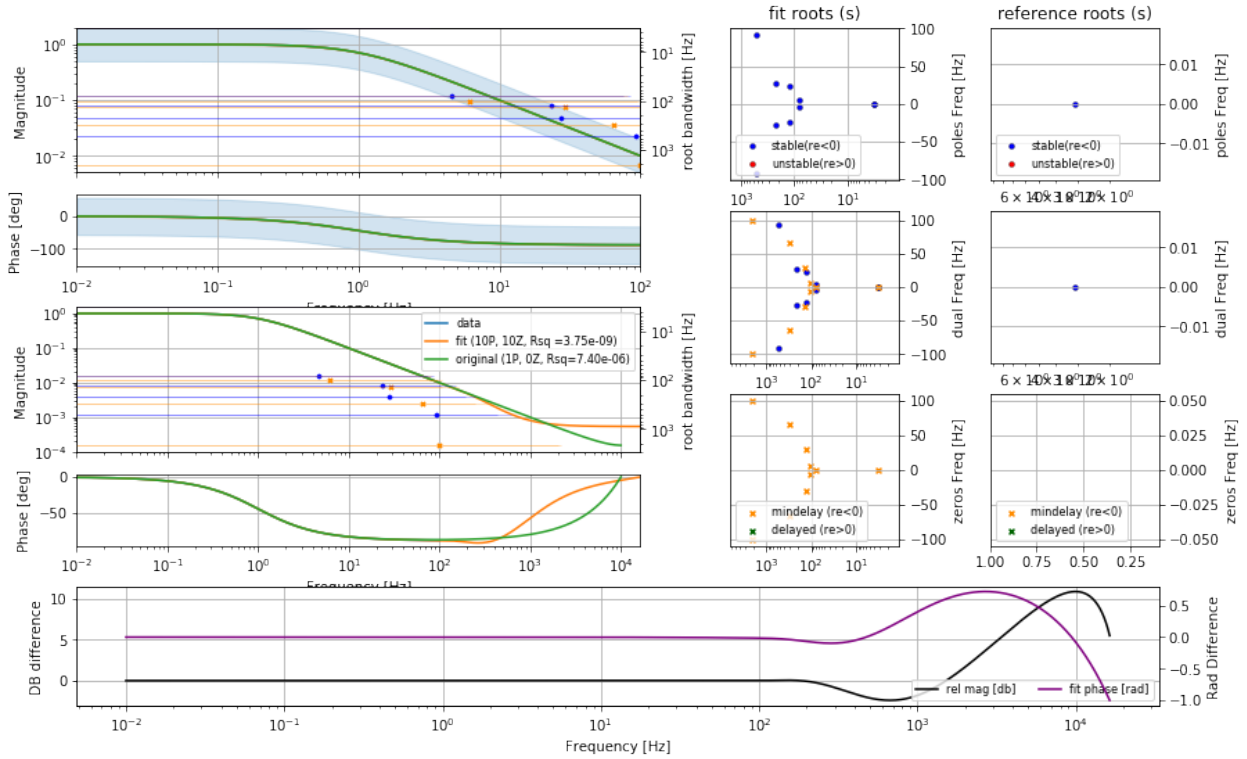


- Inspect the hintsets to get an idea how to run for different applications. The "hints" argument to data2filter can take a list of sets which it will overlay.

```
In [23]: v1.hintsets.exact_data
```

```
Out[23]: {'resavg_EthreshOrdDn': 0.01,
          'resavg_RthreshOrdC': None,
          'resavg_RthreshOrdDn': None,
          'resavg_RthreshOrdUp': None}
```

```
In [25]: out = v1.data2filter(
             tdat,
             SNR = 1,
             F_nyquist_Hz = 16384,
             hints = [
                 {'resavg_EthreshOrdDn': 1,
                  'resavg_RthreshOrdC': None,
                  'resavg_RthreshOrdDn': None,
                  'resavg_RthreshOrdUp': None},
                 v1.hintsets.quiet,
             ]
         )
         ax = plot_fitter_flag_compare(out.fitter, tdat.fitter)
```




```
In [1]: from __future__ import division
        from iirrational.utilities.ipynb_lazy import *
```

```
Populating the interactive namespace from numpy and matplotlib
```

Random Filter fit with comparison

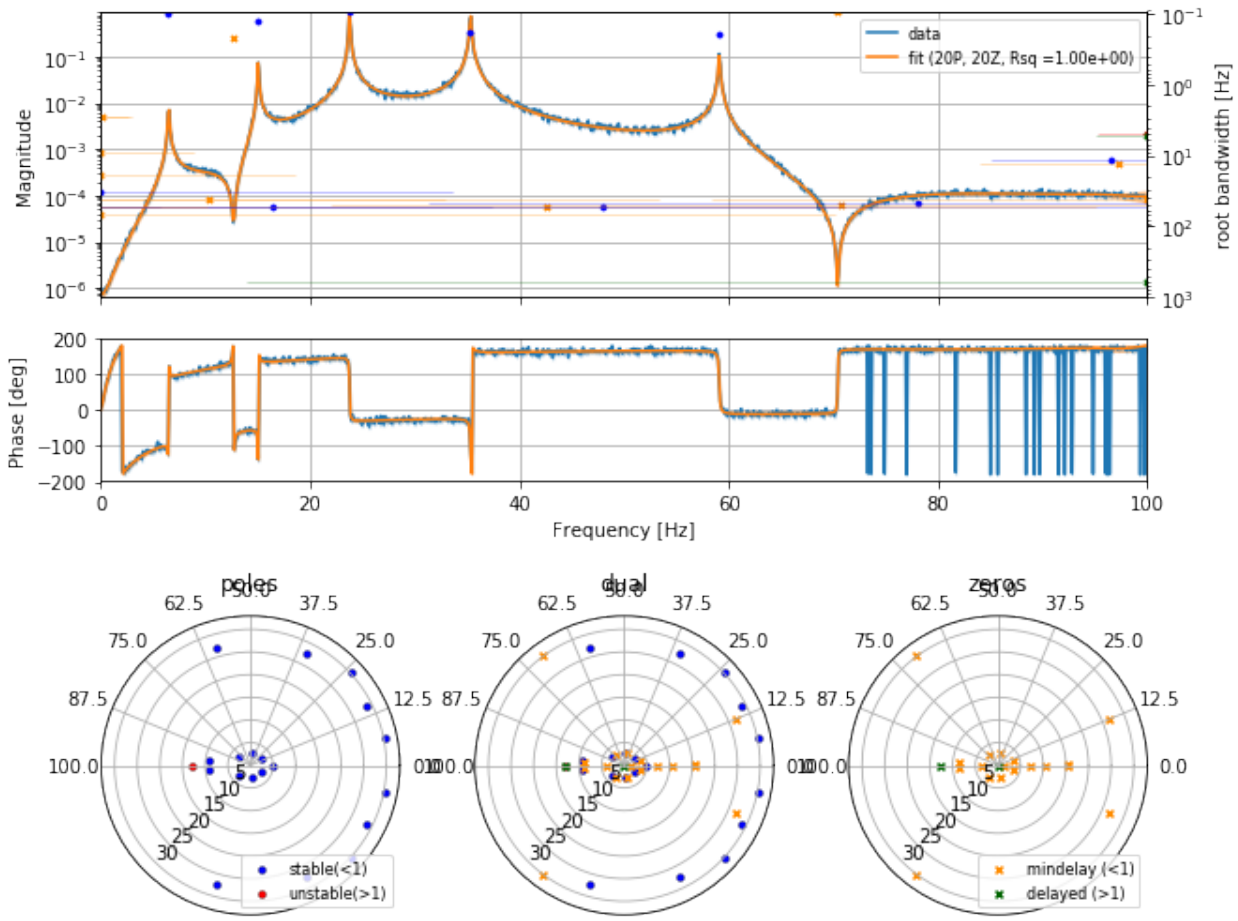
Show stage-1 Rational Disc Fit

Here is the preliminary fit on a reduced-nyquist disc

The poles and zeros are shown on the full disc. Note that unstable poles are allowed *on the real line* as these are removed by analytic surgery during the nyquist shift

```
In [6]: dat = iirrational_data('rand10_lin1k', set_num = 5)
        out = v1.rational_disc_fit(
            dat,
        )
        ax = plot_fitter_flag(out)
```

```
(direct = 4.164e+00, Psvd= 4.164e+00, Zsvd= 4.164e+00)
LINEAR Final Residuals:  3.95445037567
(direct = 3.930e+00, Psvd= 3.930e+00, Zsvd= 3.930e+00)
LINEAR Final Residuals:  3.93098246203
Using last (reduced)! 20
```

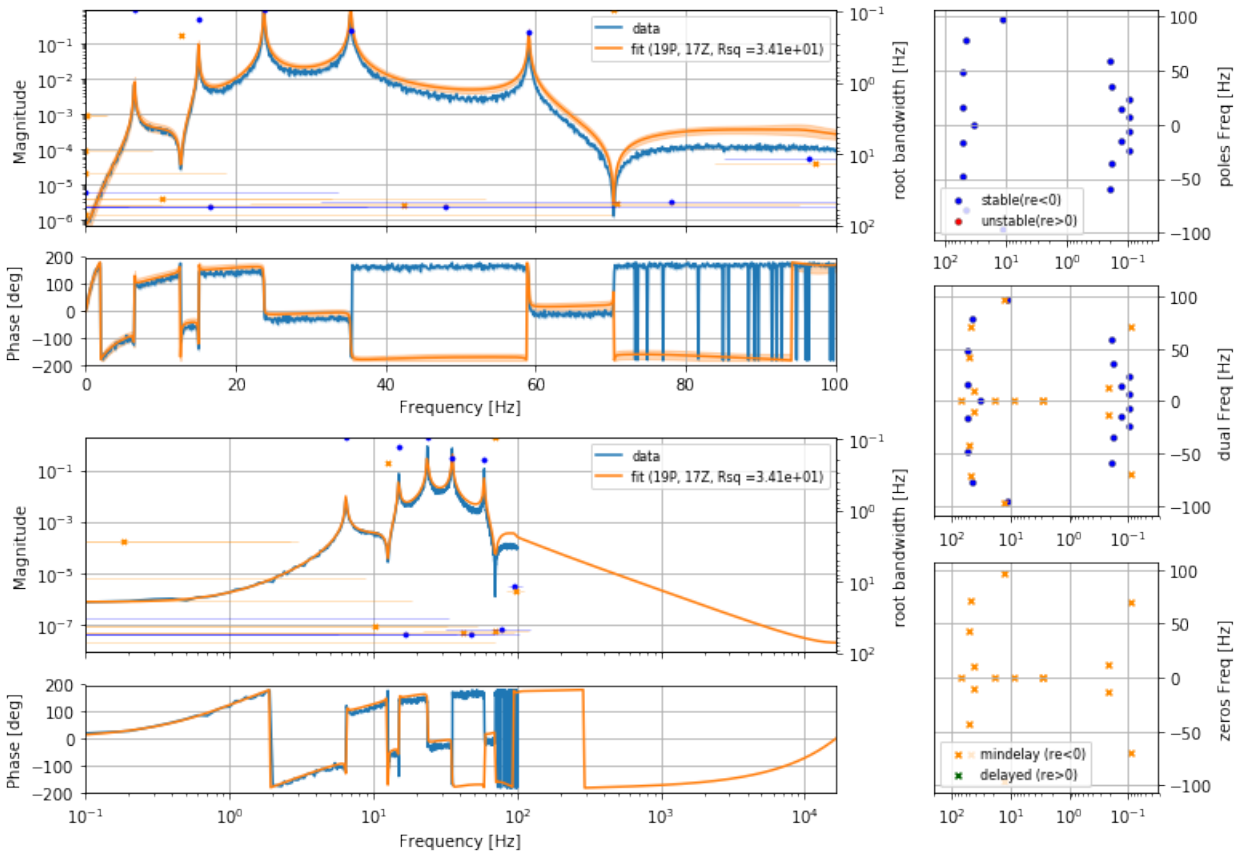


With nyquist shift

The data detunes from being a great fit

```
In [7]: dat = iirrational_data('rand10_lin1k', set_num = 5)
        out = v1.rational_disc_fit(
            dat,
            nyquist_final_Hz = 16384,
        )
        ax = plot_fitter_flag(out)
```

```
(direct = 4.164e+00, Psvd= 4.164e+00, Zsvd= 4.164e+00)
LINEAR Final Residuals:  3.95445037567
(direct = 3.930e+00, Psvd= 3.930e+00, Zsvd= 3.930e+00)
LINEAR Final Residuals:  3.93098246203
Using last (reduced)! 20
Cleared  pole (-1.04899302868+0j)
Cleared  zero (-7.22335881797+0j)
Cleared  zero (-1.05198172682+0j)
Cleared  zero (-0.584295509961+0j)
```



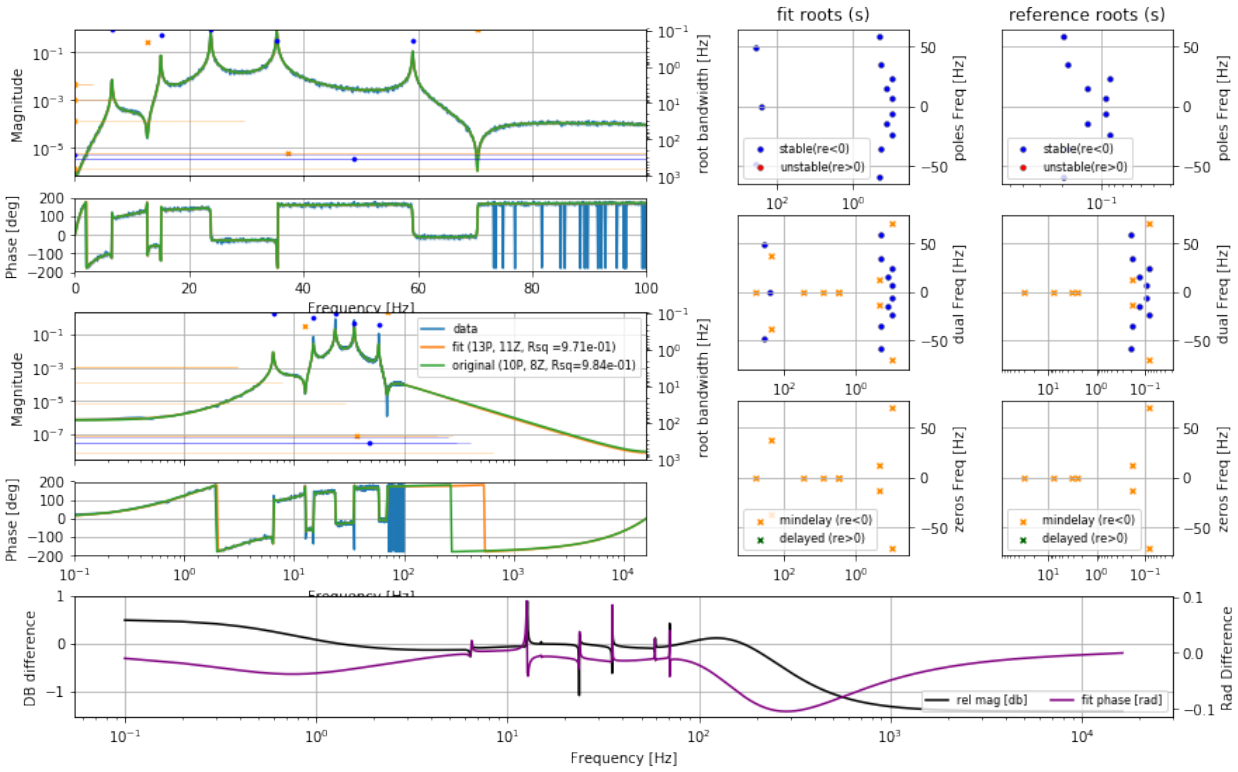
In [8]: %%time

```
        out = v1.data2filter(
            dat,
            delay_s = None,
        )
```

```
(direct = 4.164e+00, Psvd= 4.164e+00, Zsvd= 4.164e+00)
LINEAR Final Residuals:  3.95445037567
(direct = 3.930e+00, Psvd= 3.930e+00, Zsvd= 3.930e+00)
LINEAR Final Residuals:  3.93098246203
Using last (reduced)! 20
Cleared  pole (-1.04899302868+0j)
Cleared  zero (-7.22335881797+0j)
Cleared  zero (-1.05198172682+0j)
Cleared  zero (-0.584295509961+0j)
Initial Order: (Z= 17, P= 19, Z-P= -2)
TRIPLETS (rat = 1.0010150038962191, pre = 0.9734729303474599, mid = 0.9734729249578383, post = 0.973
N:  2
RATIO:  10.9002733839
fit NOT improved from pair at 13.8397945578
RATIO:  18.3818734556
fit NOT improved from pair at 9.56094083217
RATIO:  10.7017576506
fit NOT improved from pair at 18.2046398213
RATIO:  1.03476207996
fit NOT improved from pair at 64.7360498735
[0.96102176342334822] zeros
[(0.98751653189313626+0.0050603760550500913j), (0.98751653189313626-0.0050603760550500913j)] zeros
```

```
[0.98726211674090025] poles
[(0.97815059543903571+3.9610796052692903e-05j), (0.97815059543903571-3.9610796052692903e-05j)] poles
WEAK REMOVE:  [(0.98751653189313626+0.0050603760550500913j), (0.98751653189313626-0.0050603760550500
RATIO for WEAK:  1.00330207037
FINAL RESIDUALS 0.970649728969
CPU times: user 16.3 s, sys: 34 s, total: 50.2 s
Wall time: 15.8 s
```

In [10]: `ax = plot_fitter_flag_compare(out.fitter, dat.fitter)`



In [12]:
```
out.digest_write(
            folder = 'random',
            clear_plots = True,
            ipy_display = True,
            MP_workers = 1,
        )
```

```
REMOVING:  random/plot-main
REMOVING:  random/plot-1
PLOTTING:  random/plot-1-1
PLOTTING:  random/plot-1-2
PLOTTING:  random/plot-1-3
REMOVING:  random/plot-1-4
PLOTTING:  random/plot-1-5
PLOTTING:  random/plot-1-6
PLOTTING:  random/plot-1-7
PLOTTING:  random/plot-2
PLOTTING:  random/plot-3
REMOVING:  random/plot-4
PLOTTING:  random/plot-4-1
PLOTTING:  random/plot-4-2
REMOVING:  random/plot-5
REMOVING:  random/plot-6
```

```
REMOVING:   random/plot-7
REMOVING:   random/plot-8
REMOVING:   random/plot-9
PLOTTING:   random/plot-10
```

fit_sequence version 1

```
v1.fit_sequence
```

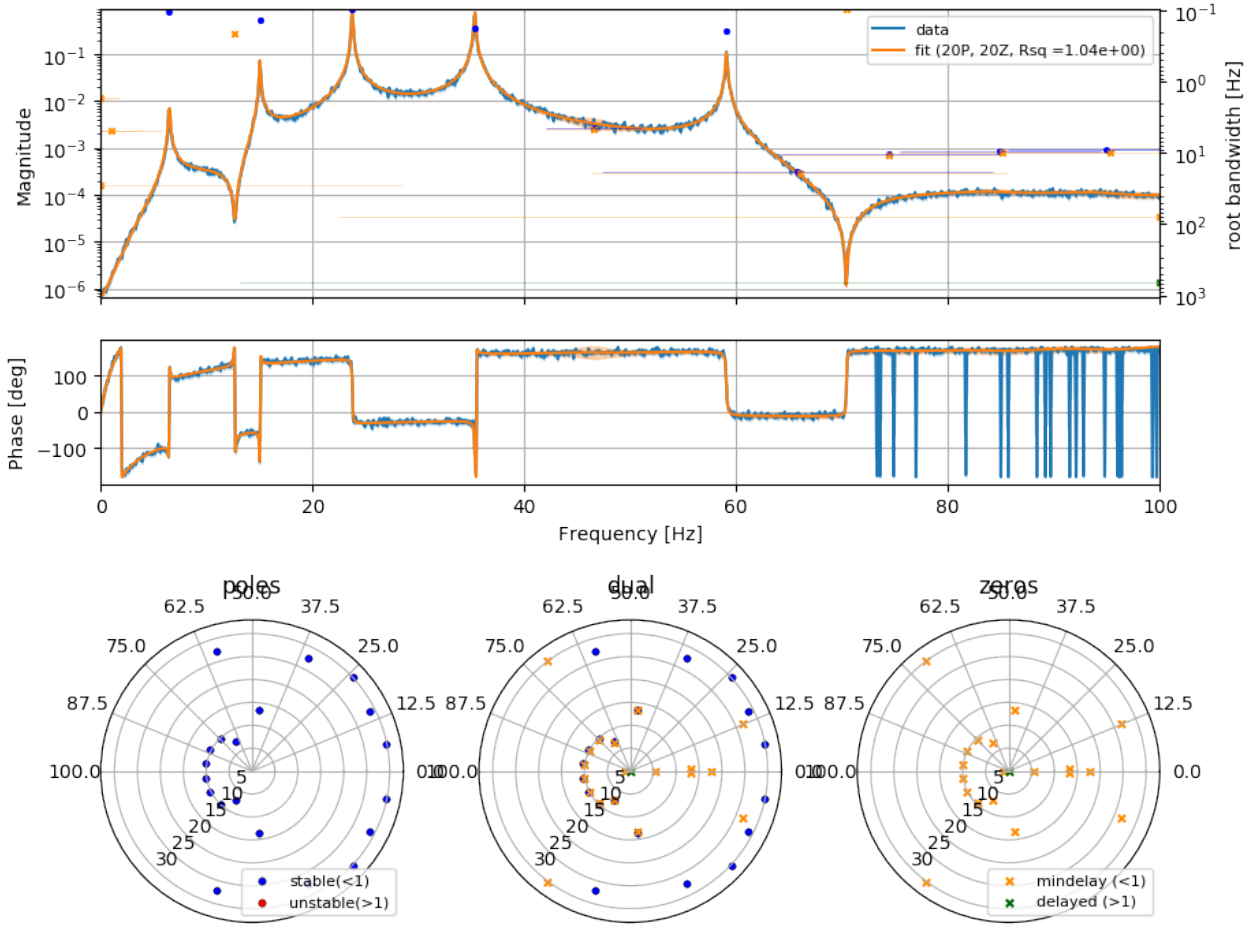
Version 1 smart fitter in iirrational library. Uses SVD method with high order over-fitting, then switches to nonlinear fits with heuristics to remove poles and zeros down to a reasonable system order.

1 initial

1.1 initial_direct

```
fit_poles, fit_zeros
```

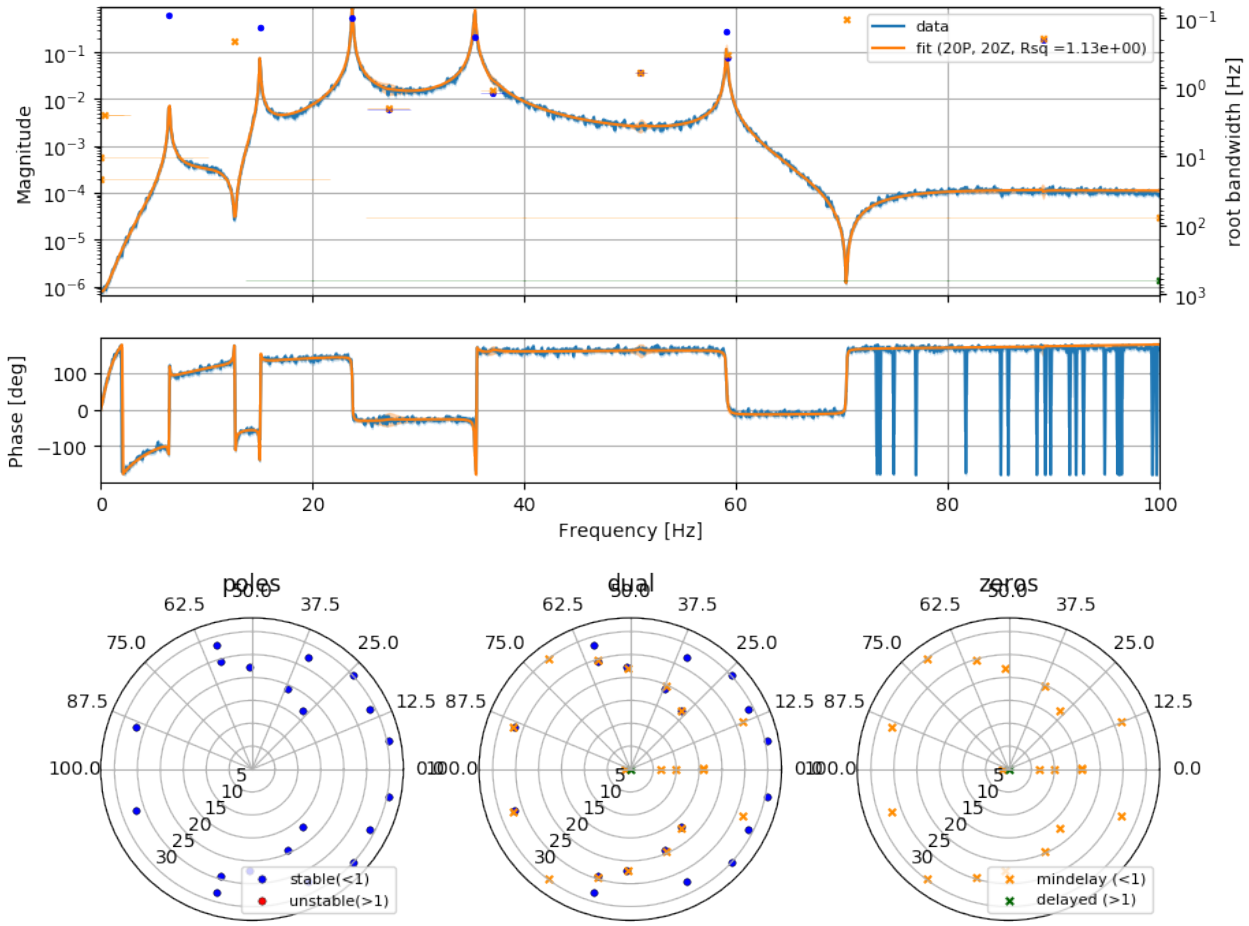
initial guess without SVD technique



1.2 initial_poles

```
fit_poles_mod_zeros
```

Performs the SVD for a rough initial guess



1.3 initial_zeros

```
fit_poles_mod_zeros
```

Performs the SVD for a rough initial guess

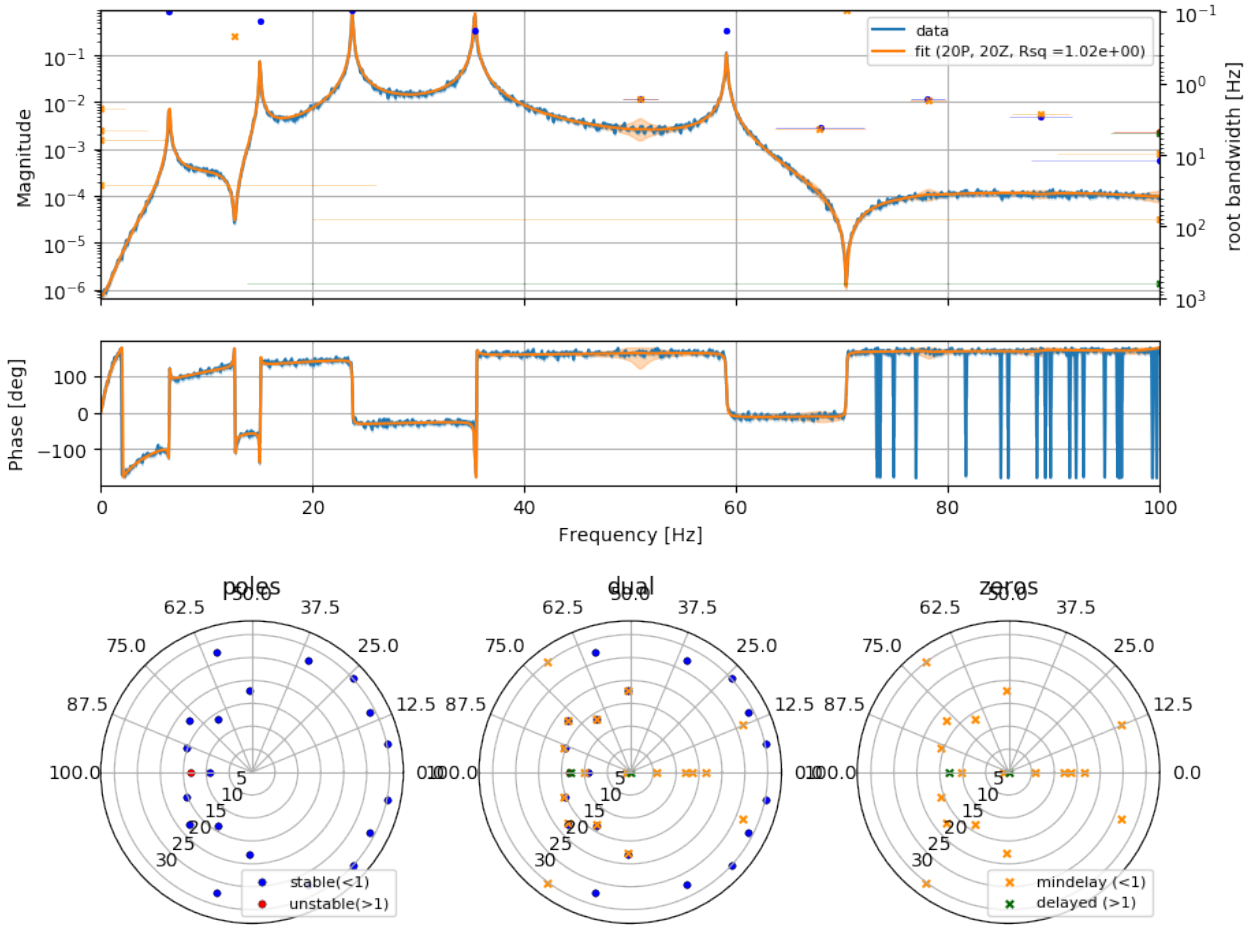
1.4 choose zeros

```
if
```

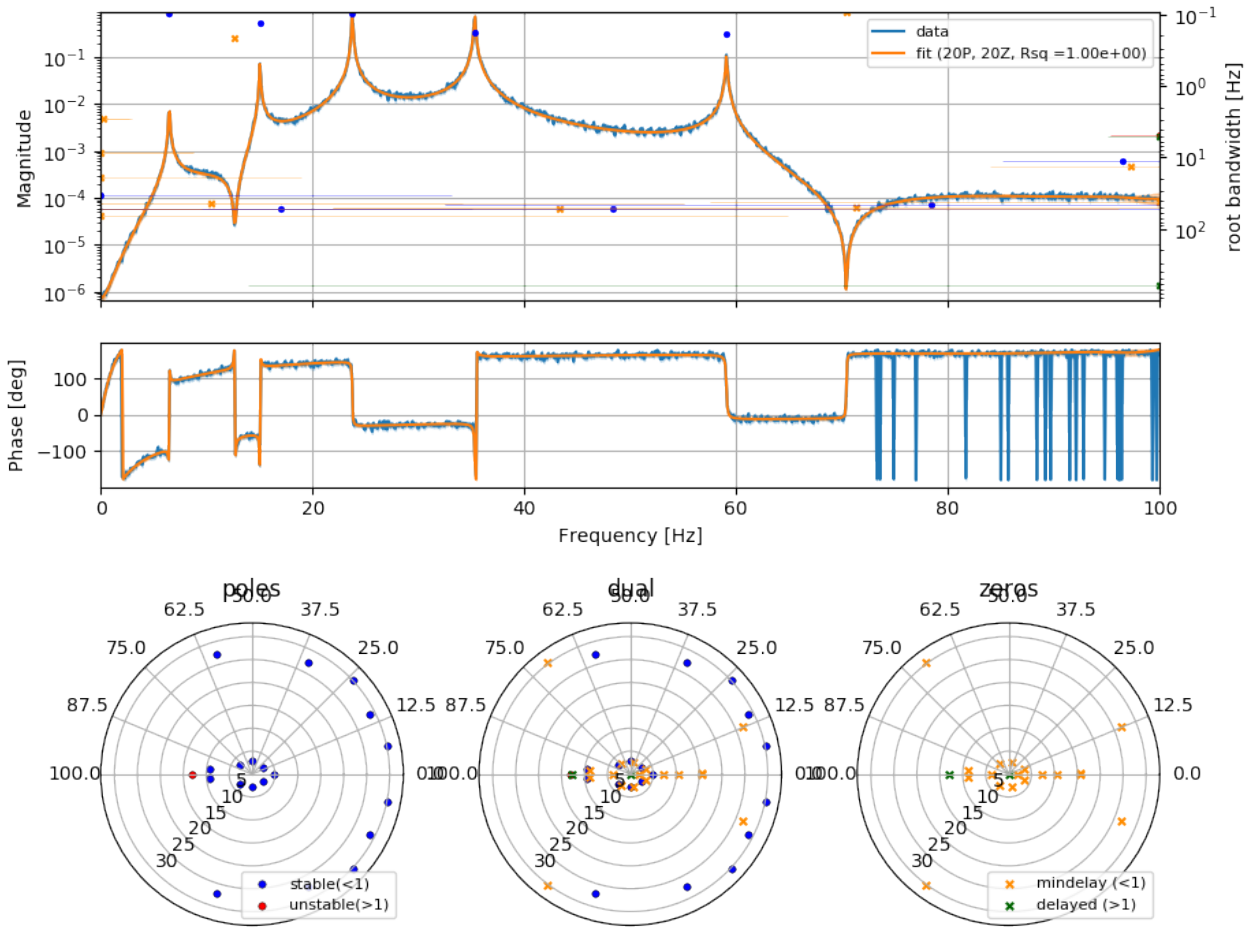
Chose the zeros SVD fitter as it had the smaller residual of 4.04e+00 vs. 4.46e+00 for the poles

1.5 seq_iter_3

```
RationalDiscFilter.fit_poles, RationalDiscFilter.fit_zeros
```


data
fit (20P, 20Z, Rsq =1.02e+00)
Magnitude
root bandwidth [Hz]
Phase [deg]
Frequency [Hz]
poles
dual
zeros
stable(<1)
unstable(>1)
mindelay (<1)
delayed (>1)

First iterations, enforcing stabilized poles residual of 3.95e+00



1.6 seq_iter_4

```
RationalDiscFilter.fit_poles, RationalDiscFilter.fit_zeros
```

First iterations, enforcing stabilized poles residual of 3.95e+00

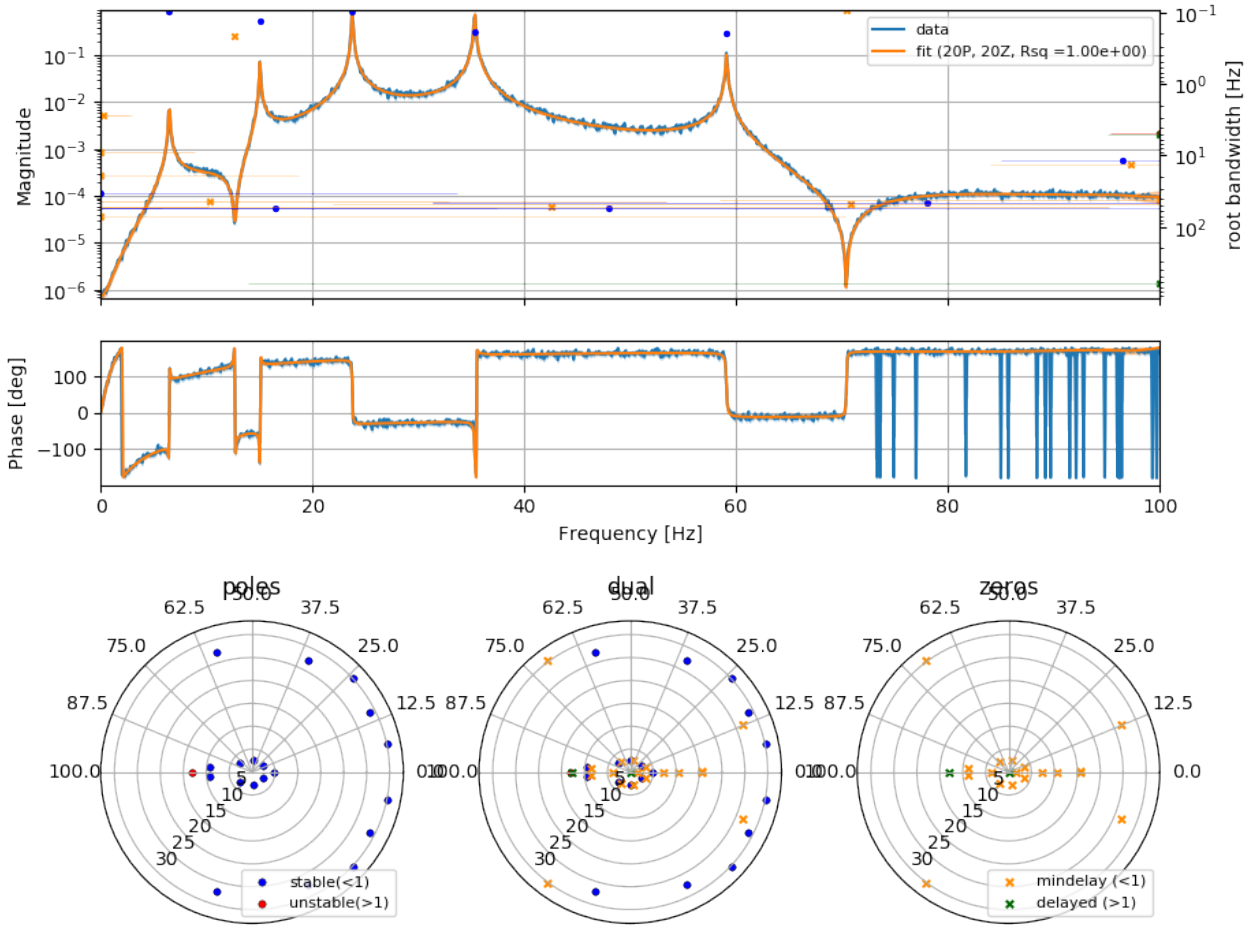
1.7 Final

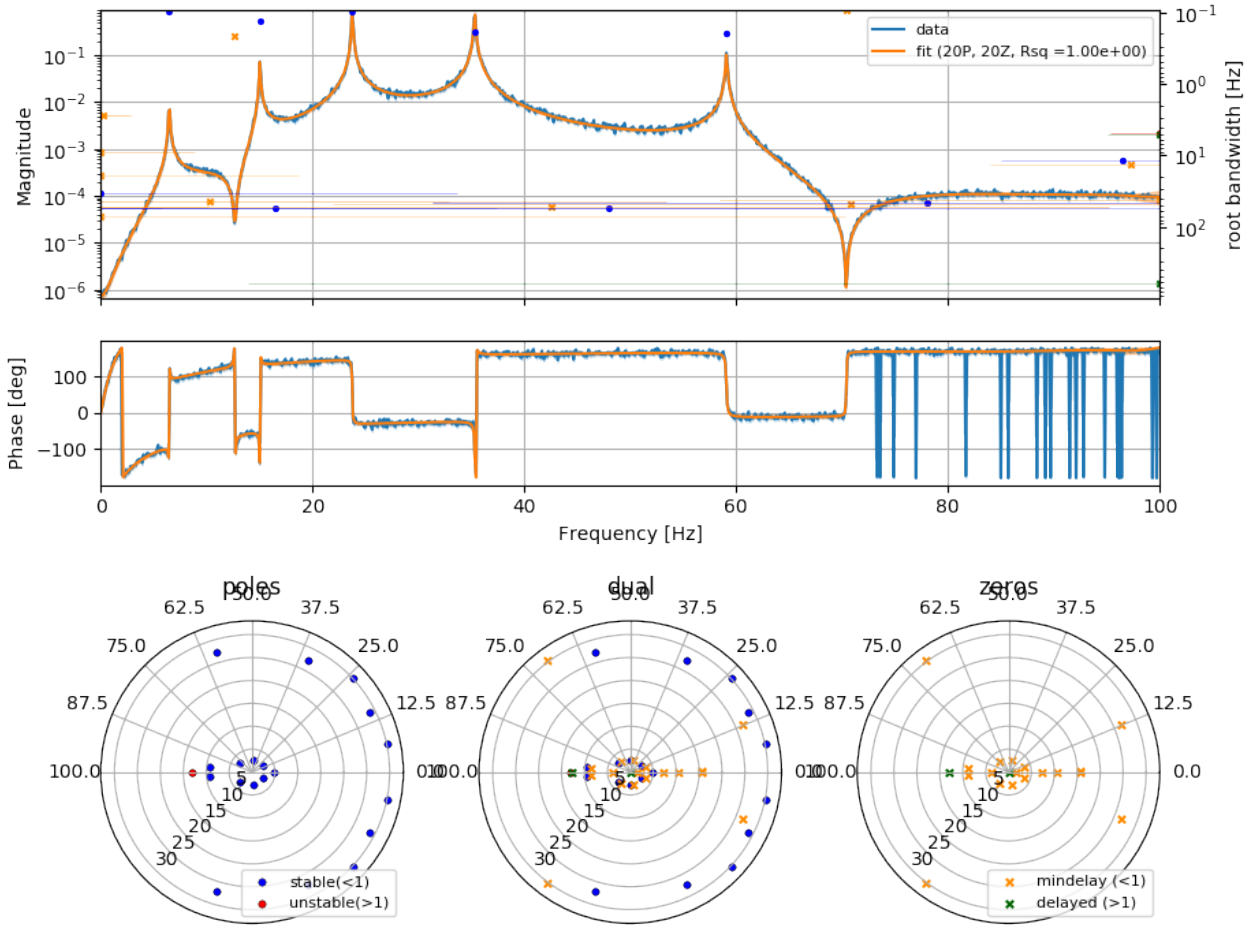
```
SVD_method
```

create initial guess of fit for data, followed by several iterative fits (see reference ???). * It is a linear method, finding global optimum (nonlocal). This makes it get stuck if systematics are bad. To prevent this, it requires gratuitous overfitting to reliably get good fits. * It requires a nyquist frequency that is very low, near the last data point. This can cause artifacts due to phasing discontinuity near the nyquist. * The provided nyquist frequency is shifted up at the end, removing the real poles/zeros that are typically due to phasing discontinuity

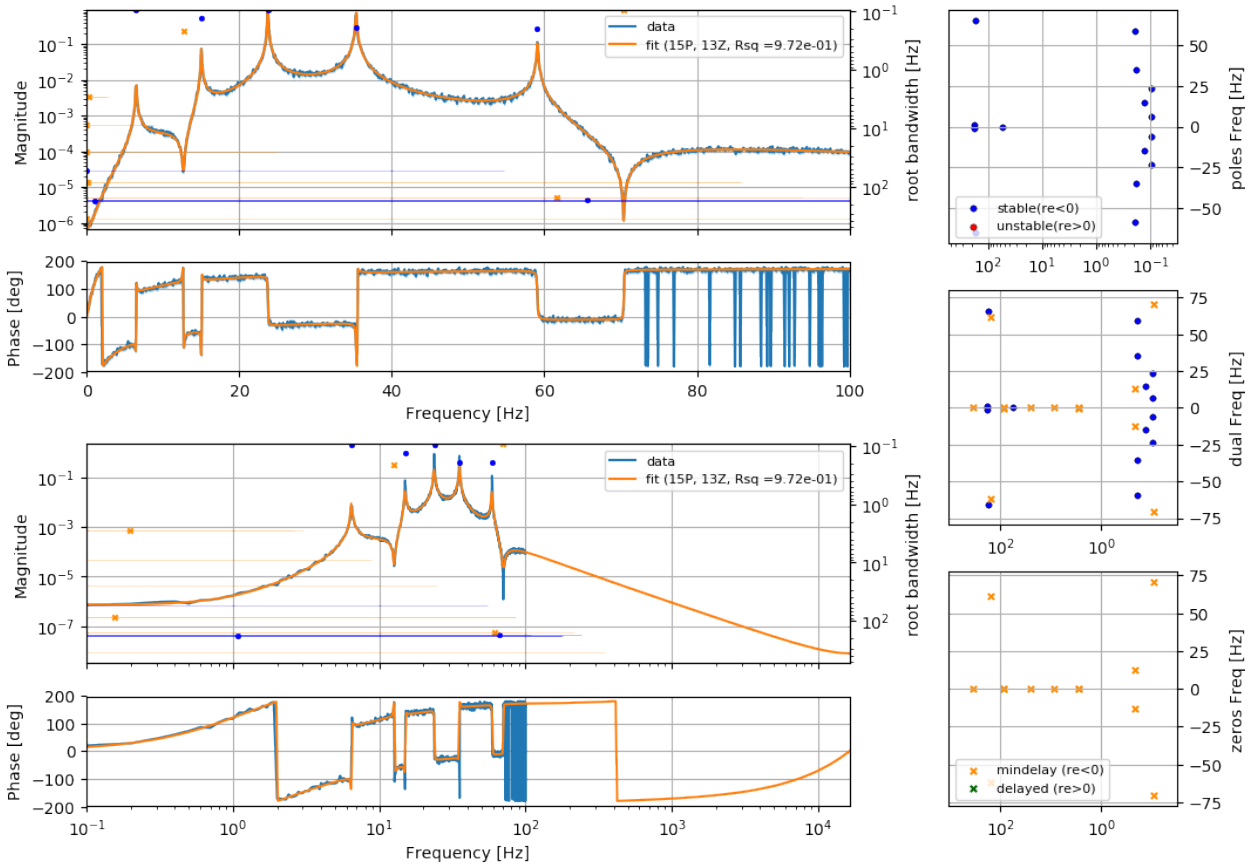
2 nonlinear pre-reduce and optimize

```
MultiReprFilter.optimize
```


data
fit (20P, 20Z, Rsq =1.00e+00)
Magnitude
root bandwidth [Hz]
Phase [deg]
Frequency [Hz]
poles
dual
zeros
stable(<1)
unstable(>1)
mindelay (<1)
delayed (>1)


data
fit (20P, 20Z, Rsq =1.00e+00)
Magnitude
root bandwidth [Hz]
Phase [deg]
Frequency [Hz]
poles
dual
zeros
stable(<1)
unstable(>1)
mindelay (<1)
delayed (>1)

initial conservative order reduction (for speed), followed by a nonlinear optimization.



3 optimize nonlinear after bandwidth limiting

`MultiReprFilter.optimize`

ths limited in the nonlinear representation to half of the local average the frequency spacing. Nonlinear optimization then applied.

4 order_reduce 1

4.1 duals_ID_1

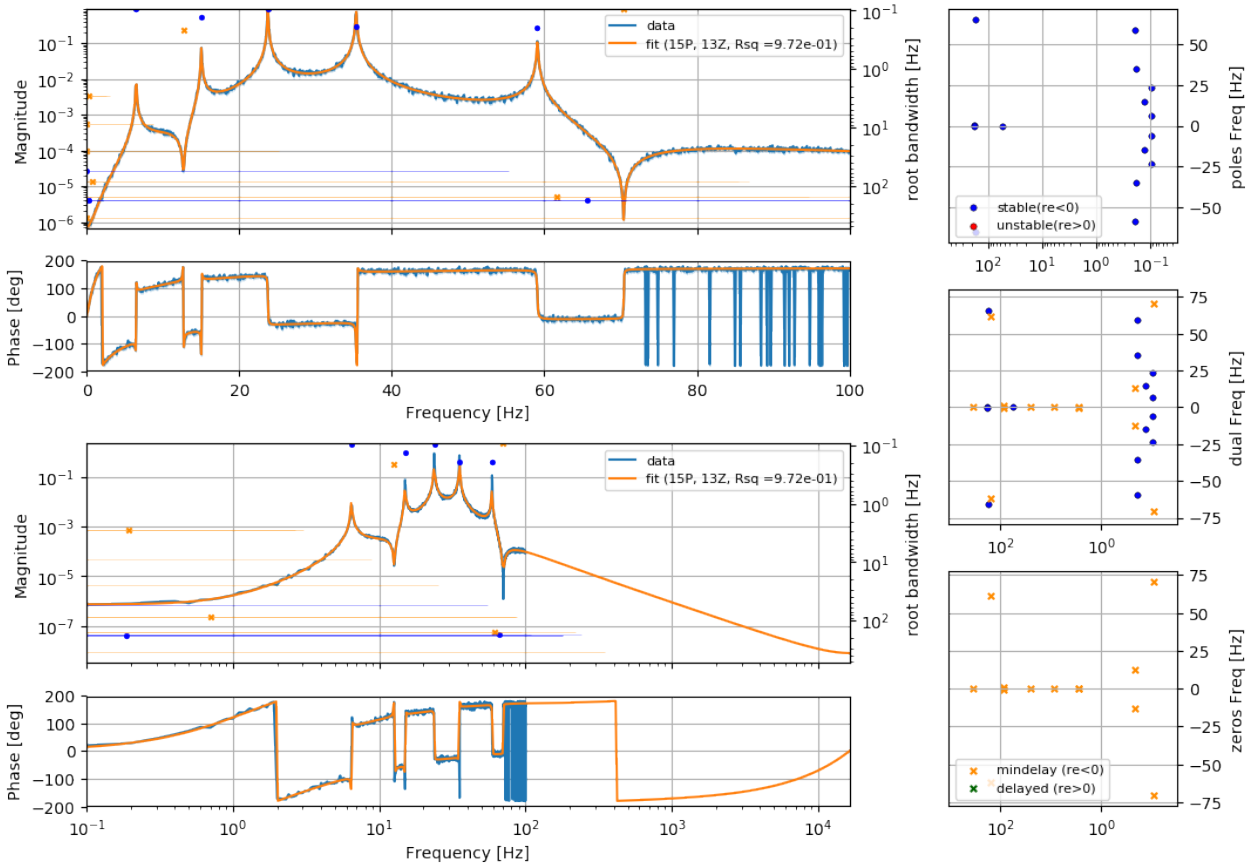
`MultiReprFilter.optimize`

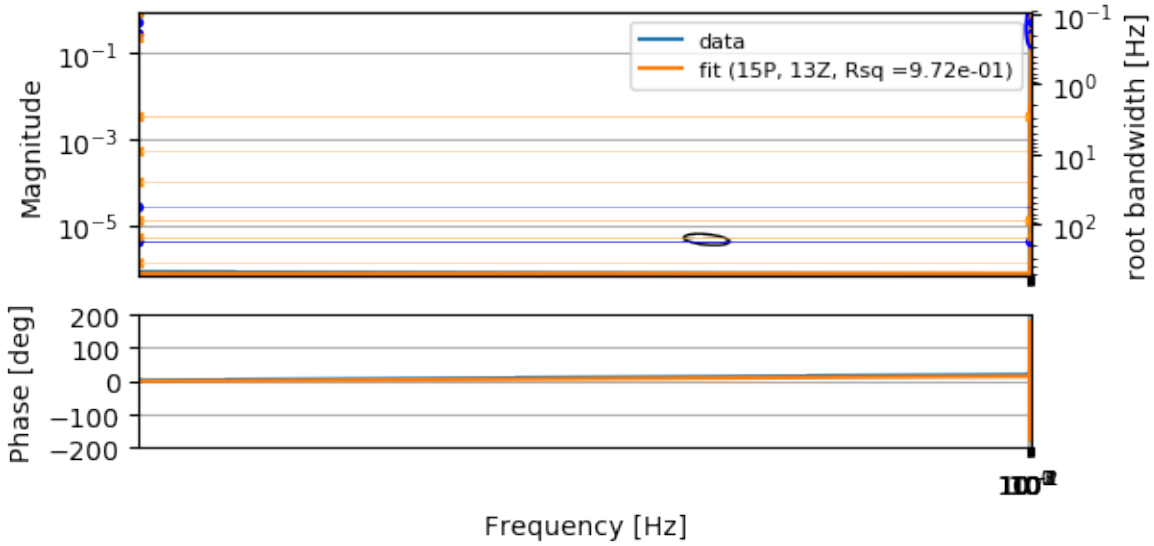
ID Pairs for order reduction

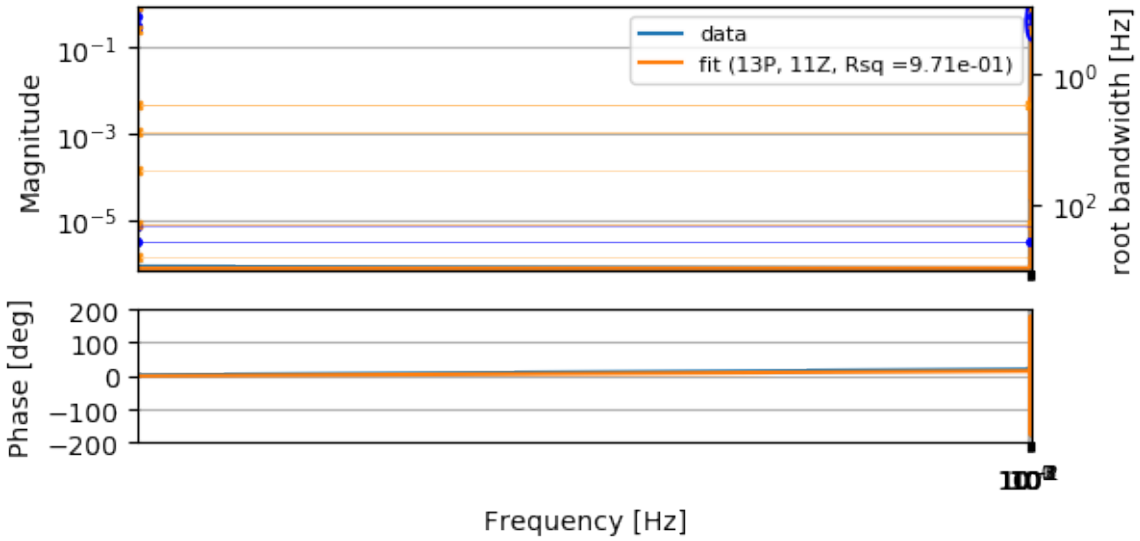
4.2 duals_ID_2

`MultiReprFilter.optimize`

ID Pairs for order reduction


data
fit (15P, 13Z, Rsq =9.72e-01)
Magnitude
root bandwidth [Hz]
Phase [deg]
Frequency [Hz]
stable(re<0)
unstable(re>0)
poles Freq [Hz]
dual Freq [Hz]
mindelay (re<0)
delayed (re>0)
zeros Freq [Hz]


data
fit (15P, 13Z, Rsq =9.72e-01)
Magnitude
root bandwidth [Hz]
Phase [deg]
Frequency [Hz]



5 remove_negroots

6 remove_weakroots

7 remove_negroots

8 remove_negroots

9 flip mindelay

10 final

`MultiReprFilter.optimize`

ing nonlinear parameterizations, TODO: describe as used

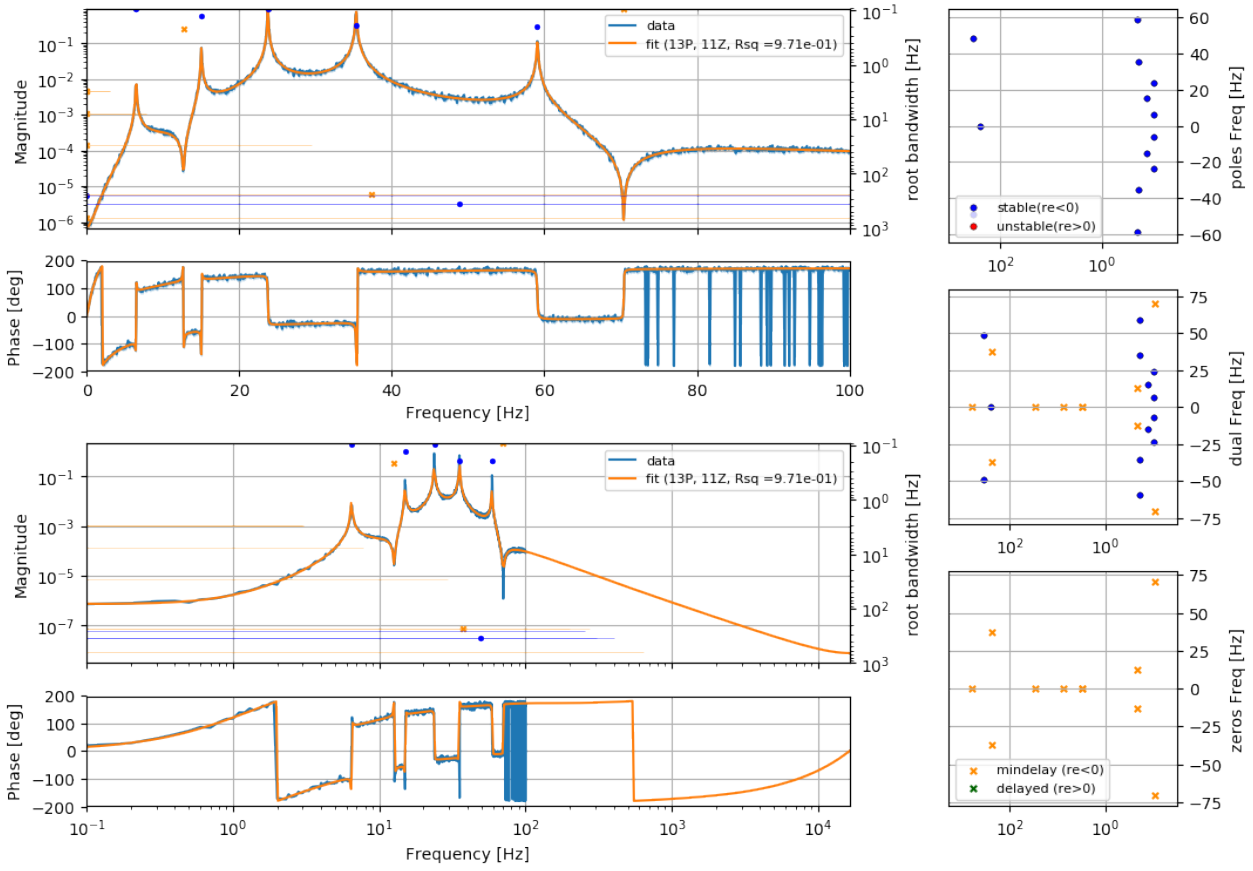
Export Testcase Example

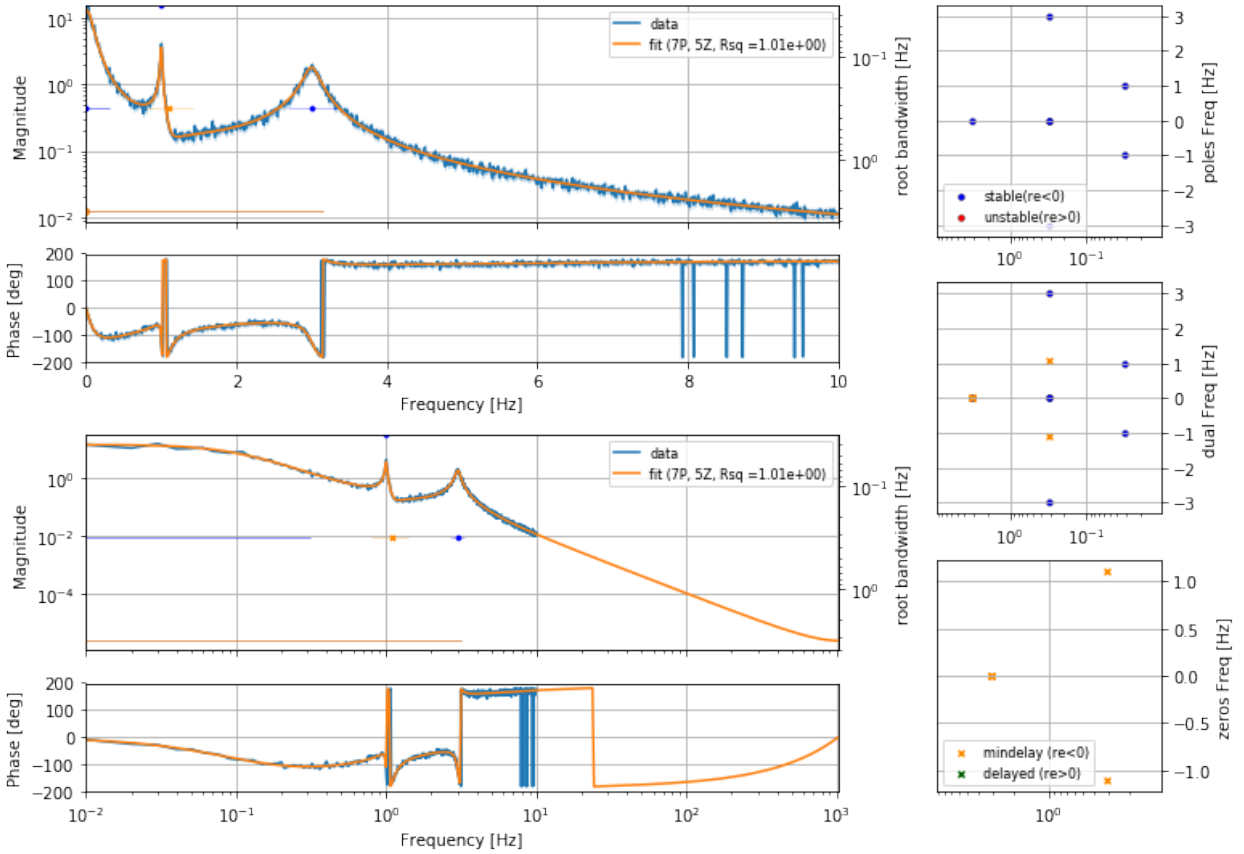
Here an existing randomly generated testcase is exported

```
In [1]: from __future__ import division
        from iirrational.utilities.ipynb_lazy import *

Populating the interactive namespace from numpy and matplotlib

In [2]: dat = iirrational_data('simple2', instance_num = 2)
        axB = plot_fitter_flag(dat.fitter)
```


data
fit (13P, 11Z, Rsq =9.71e-01)
Magnitude
root bandwidth [Hz]
Phase [deg]
Frequency [Hz]
stable(re<0)
unstable(re>0)
poles Freq [Hz]
dual Freq [Hz]
mindelay (re<0)
delayed (re>0)
zeros Freq [Hz]




```
In [3]: iirrational.v1.data2testcase(
            fname = 'simple2.mat',
            data = dat.data,
            F_Hz = dat.F_Hz,
            SNR = dat.SNR,
            bestfit_ZPK_z = dat.bestfit_ZPK_z,
            F_nyquist_Hz = dat.F_nyquist_Hz,
            description = 'test',
        )

In [4]: import scipy.io
        d = scipy.io.loadmat('simple2.mat')

In [5]: d['bestfit_ZPK_z']

Out[5]: array([[ array([[ 0.99693204+0.j        ,  0.99693204+0.j        ,
                  0.99693204+0.j        ,  0.99968751+0.00337372j,
                  0.99968751-0.00337372j]]),
                array([[ 0.99969320+0.j        ,  0.99969320+0.j        ,
                  0.99693204+0.j        ,  0.99965086+0.00920093j,
                  0.99996461+0.00306786j,  0.99965086-0.00920093j,
                  0.99996461-0.00306786j]]),
                array([[  9.41201413e-06]])]], dtype=object)
```

2.3 Quickstart for Matlab

Although this is a python library, it may be used from Matlab via the builtin (since 2014) python interfaces

Unfortunately, IIRrational cannot be directly run inside of the Matlab python interpreter. Matlab uses its own library ecosystem, including a number of critical numeric libraries shared by Scipy. At least on linux these are not ABI compatible and python will fail to import many scipy modules due to symbol lookup errors.

A workaround is provided using the msurrogate library, which allows IIRrational to be run as a separate process, while appearing native to Matlab. This not only fixes scipy, but also allows easy multithreaded access and extends the native Matlab python array wrapping conveniently to and from Numpy types.

2.3.1 Installation Steps

First *Install IIRrational through pip* and msurrogate will come with it automatically as a dependency. Now you only need to:

1) check that python is working from Matlab

See the Matlab documentation.

On Linux (Fedora 25) with 2016a, this appears to *just work*, with Matlab finding and using the system python27. If redirected to system python35 matlab fails to find the python libraries. Either it does not support such a new python3 (likely for 2016a as the matlab.engine python module does not support python35). Please submit issues/pull requests to debug this on mac/windows particularly using non-native python distributions such as Anaconda.

2) Add IIRrational to matlabpath

The matlab code to use is stored within the python module (both for IIRrational and msurrogate). To access it, just put it into the path using

```
addpath(char(py.iirrational.matlabpath()));
```

This asks for the path through the python function call matlabpath of the iirrational module. The Matlab *char* function converts from a python string. If this throws an error, then either

- Python is not found by Matlab
- IIRrational is not installed, or **not installed into the python environment found by Matlab.** Check which python Matlab found. If you are running from an install from Anaconda, Homebrew, macports et al, then Matlab may not be using the non-system python!

If this is working, then you may either add this code to all projects wishing to use the IIRrational library, or get the paths returned from this function and add them manually to a permanent matlabrc or to MATLABPATH.

2.3.2 Two ways to start

The separate python process hosting IIRrational communicates with Matlab through sockets. The addresses for this communication are held in a cookie file provided by the python host process. This process must be started with the location to write the cookie file, and then Matlab must be given the location of this file. Using sockets, the python process may even reside on a different computer than Matlab. Since Matlab's python can start this process itself, first try the direct method.

Direct Subprocess Method

```
addpath(char(py.iirrational.matlabpath()));

%Without arguments, a subprocess is started
iir = iirrational.surrogate();

%positional arguments MUST be within a cell array due to
%the calling convention of the python-wrapping done by msurrogate
%(they are also discouraged for readability in favor of more verbose keyword␣
↪arguments)
dat = iir.testing.iirrational_data({'simple1'});

%Create native keyword arguments
kw = struct();
kw.data = dat.data;
kw.F_Hz = dat.F_Hz;
kw.F_nyquist_Hz = dat.F_nyquist_Hz;

%also add SNR as a keyword arguement
fit_out = iir.v1.data2filter(kw, 'SNR', dat.SNR);

%Show the ZPK output to use in other systems
disp(fit_out.fitter.ZPK)

%Write a plot
axB = iir.plots.plot_fitter_flag({fit_out.fitter}, 'fname', 'test_plot.pdf');
```

This method is convenient as Matlab will manage the lifetime of the subprocess. Furthermore, the stdout (console out) of the python process will be piped into Matlab (although often with a delay).

Separate Process Method

Use this method if the subprocess method is not working, you would like to manage the lifetime of the python process, or you are running python remotely. You can actually use the system python with Matlab and a newer python environment from the subprocess (perhaps use a virtualenv to get newest scipy/numpy/matplotlib).

Check out the start options with

```
python -m iirrational.matlab -h
```

and minimally start it using

```
python -m iirrational.matlab -c workspace/path/iirrational.cookie
```

now inside of matlab

```
addpath(char(py.iirrational.matlabpath()));

%with arguments, a cookie filename to connect to is assumed
iir = iirrational.surrogate('workspace/path/iirrational.cookie');
...
```

If the process is created on a separate machine, the `--public` option should be given, along with a `--host` hostname (the library is not particularly secure since it transmits using pickle objects, but it does take some minimal steps). The cookie file must be copied to the matlab machine in this case. If the `--port` and `--secret` options are also given, then the cookie file will not change between invocations and the copy is only necessary once.

2.3.3 Usage

The return value of the Matlab *iirrational.surrogate* function is an object representing the proxy workspace. It has a similar structure to the python modules, with a *.v1* attribute providing access to the functions in the python *v1* submodule. It also has *plots* and *annotation*. Tab completion should work for the objects, so try it out to find methods to call and properites to inspect.

Calling Conventions

Function calls are done using the *()* operator from Matlab, wheras item lookup *even into arrays* is done using *{}* operators. If the array was a numpy array it will be converted back into a matlab array and the Matlab *()* indexing syntax will be used. When in doubt, check the return type. *msurrogate.PyWrap* and *msurrogate.PyroWrap* use the python syntax, and otherwise matlab syntax should be assumed.

Function calling

As alluded in the examples, the function calling syntax is idiosyncratic to conveniently accommodate keyword arguments. The general pattern isempty

```
iir.module.function({positional1, positional2, . . .}, kwarg_struct, 'kwarg1', val,
↪'kwarg2', val, . . .)
```

And actually any number of cell arrays and kwarg_structs may be used. Positional arg cell arrays are concatenated and kwarg_structs are overlayed, with later ones taking precedence. The first string argument switches the parser to assuming the rest are argname, value pairs and these take the highest precedence. There is no other way to provide positional arguments than through the cell arrays. It is easy to accidentally omit them and hopefully the error messages are helpful.

2.3.4 Gotchas

- The python subprocess has its own current working directory, so relative paths will NOT be with respect to the current Matlab path, but the python one (likely the directory where matlab was started).
- Interactive plotting requires matplotlib to be setup with an appropriate backend.
- The python workspace currently does not automatically clean up old objects, so it can eat memory if used for an extended period. Garbage collection is planned but not particularly tested
- in principle, multiple users/workspaces could connect to a single iirrational process. This is untested.
- Only python lists, tuples, dictionaries and numpy arrays are transmitted. Everything else is a proxy object into the python process. Native types like dicts will be proxies as well if they contain any proxied object.
- Proxy objects will be "unwrapped" on the python side, so function arguments can be a proxy and python will use the native object in its workspace (good for `MultiReprFilter` objects returned with *`data2filter()`*).

2.4 Development

2.4.1 Design Considerations

Coding Conventions

Testing

Uses py.test for tests.

tox

tox aids testing against multiple versions of python in clean environments. To run it:

```
$tox
```

and to run against the full test suite on the system python:

```
$tox -e full
```

The typical build and install distribution does not include the test suite data.

Helpful links

- http://tox.readthedocs.io/en/latest/example/general.html

2.4.2 Building Docs

Documentation is written in the powerful, flexible, and standard Python documentation format, reStructured Text. Documentation builds are powered by the powerful Pocoo project, Sphinx. The *API Documentation* is mostly documented inline throughout the module.

The Docs live in `iirrational/docs`. In order to build them, you will first need to install Sphinx.

```
$ pip install sphinx
$ pip install nbsphinx
```

and optionally (but very helpful)

```
$ pip install sphinx-autobuild
```

Then, to build an HTML version of the docs, run the following from the **docs** directory:

```
$ make html
```

and for the autoreload, run:

```
$ make livehtml
```

The `docs/build/html` directory will then contain an HTML representation of the documentation. When committed to github, readthedocs will automatically build the latest version and host it.

2.5 Fitting API

The library using a versioning scheme to preserve old heuristics in future versions. There will be an iirrational.vN submodule that is **larger** Than the current full version number. This one will _not_ be api stable or heuristics stable. The versions equal or below the current full version will be preserved.

2.5.1 v1 fitting interface

iirrational.v1.**data2filter**(*argB=None*, *data=None*, *F_Hz=None*, *SNR=None*, *F_nyquist_Hz=None*, *ZPK=None*, *SNR_initial=None*, *order_initial=None*, *verbosity_limit=None*, *SNR_cutoff=None*, *doc_db=None*, *delay_s=None*, *hints=None*, *alt_res=False*)

2.5.2 Fitter Objects

iirrational.MRF.**MultiReprFilter**(*_args=None*, *copy=None*, *lightweight=True*, ***kwargs*)

2.5.3 Plotting

2.5.4 Exporting

2.6 Advanced

While the library aims primarily to expose a simple fitting API, its internal toolkit is useful and versatile for advanced applications. Here it is documented. The alternative way to learn how it operates is to view the sources for the simpler API. These are within iirrational/v1/data2filter.py and its imports.

Please contribute if a new technique, parameterization, or simplification shows promise!

2.6.1 Contents

Toolkit API

Linear Rational Disc Stage

class iirrational.RDF.**RationalDiscFilter**(*_args=None*, *copy=None*, *lightweight=True*, ***kwargs*)

matched_pairs_clear(*Q_rank_cutoff=0.5*)
Match unique closest pairs, if they are within a bandwidth of 0Hz, then they are ignored

Nonlinear flexible parameterization fitting

class iirrational.MRF.**MultiReprFilter**(*_args=None*, *copy=None*, *lightweight=True*, ***kwargs*)

optimize_NM(*bootstrap=True*, *residuals_type=None*, ***kwargs*)
Uses Nelder-Mead with a simplex that is bootstrapped from the svd of the jacobian. This gives particularly good initial directions to check

Root Parameterization Codings

2.7 Ideas

2.7.1 Power Spectra

The two stages can be straghtforewardly adapted to fit without phase information to generate ZPK representations of power spectra. Very useful for later application of quadratic regulator methods for optimal control (even with SISO).

2.7.2 Order Reduction

Use Pade tables to find alternate ZPK representations during the order reduction stage. This may work more efficiently and provide a non-greedy method. May have numerical issues.

2.7.3 Wiener Filter / Feedforward

Since Weiner filter methods ultimately work as least-squares parallel SISO filters, this library could provide IIR representations and potentially incorporate error information better.

Todo: Make a timeseries interface with internal PSD/CSD estimation, error bars, and automically feed these estimates to data2filter. Potentially compare the result to the standard FIR method with some error estimate. Perhaps allow witness vs. Fit datasets.

2.7.4 MIMO

With sufficient reliability, this SISO fitter could compose multiple fits into a MIMO system. The Order reduction algorithms could be modified for common root identification to weak SISO ZPK outputs into a MIMO Z, common-P, K that could generate state-space matrices of a reasonable order.

2.7.5 Error Estimation

SNR Reconditioning

SNR estimates from coherence can be biased and wrong in a number of ways. With enough data-points, an initial fit could be used to find the residuals and some variance smoothing method could be applied to find the (somehow defined) local SNR of data points where it is apparent that the SNR is poorly estimated. The fit could then be iterated with updated SNR estimates.

MCMC

Use emcee for a stage-3 error estimation after the least squares.

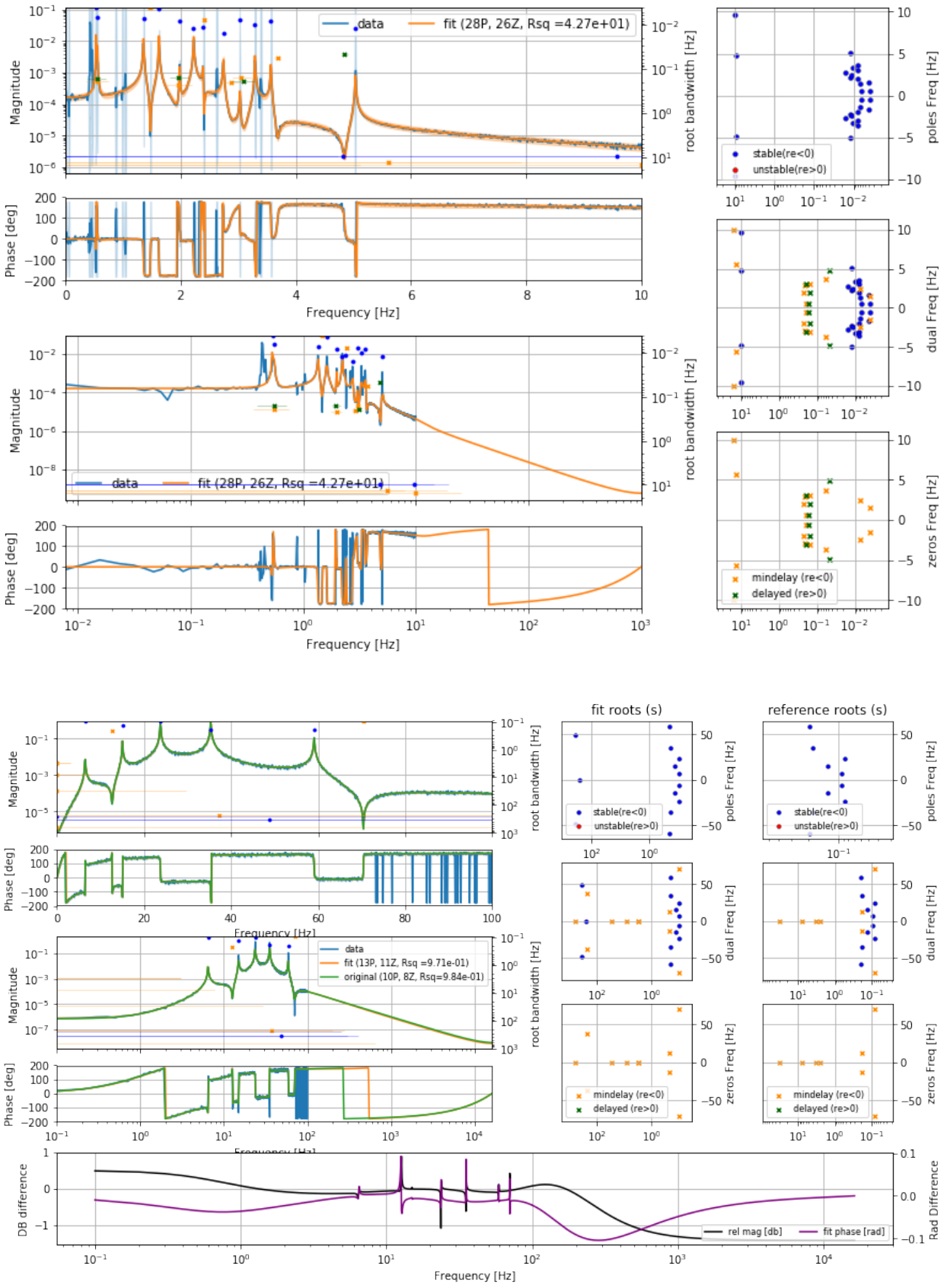
CHAPTER 3

Example Output

Each of these fits has 1000 data points. The first is an Pitch to Vertical cross term coupling of an advanced LIGO suspension. The second is a random filter with SNR of 10 at each data point. Neither fit requires user tuning and (as is typical) correctly identifies the rolloff above the data.

solves in 60s

solves in 20s


data
fit (28P, 26Z, Rsq =4.27e+01)
Magnitude
root bandwidth [Hz]
Phase [deg]
Frequency [Hz]
stable(re<0)
unstable(re>0)
poles Freq [Hz]
dual Freq [Hz]
mindelay (re<0)
delayed (re>0)
zeros Freq [Hz]
fit roots (s)
reference roots (s)
fit (13P, 11Z, Rsq =9.71e-01)
original (10P, 8Z, Rsq=9.84e-01)
DB difference
rel mag [db]
fit phase [rad]
Rad Difference

CHAPTER 4

Future Directions

The principle goal at this stage of the project is to improve reliability and speed of the fitting. To do this, I encourage using the export utility and posting the data on the github issues for integration into the test suite. Future versions aim to reliably fit all data in the test suite. Real data is essential as it exercises the heuristics in ways difficult to predict and emulate with random data.

Currently this package is only for SISO fitting. Its use for MIMO systems is limited to how automated fitting can become. Future work may extend the algorithms to intrinsically fit MIMO systems. This may be done either in the linear stage-1 or to compose many SISO functions, order-reduce them as a MIMO system and then perform nonlinear optimization on the resulting state-space formulation.

CHAPTER 5

Alternatives

- Vectfit [matlab] https://www.sintef.no/projectweb/vectfit/
- python vectfit [python] https://github.com/PhilReinhold/vectfit_python
- fdident [matlab] https://www.mathworks.com/products/connections/product_detail/product_35570.html
- TFestimate [matlab] https://github.com/Nikhil-Mukund/TFestimate/blob/master/README.md

Python Module Index

i

Index