scikit-rf: An open source Python package for microwave network creation, analysis and calibration

Alexander Arsenovic, Julien Hillairet, Jackson Anderson, Henrik Forsten, Vincent Riess, Michael Eller, Noah Sauber, Robert Weikle, William Barnhart, Franz Forstmayr

scikit-rf: An Open Source Python Package for Microwave Network Creation, Analysis and Calibration

Alexander Arsenovic, Julien Hillairet, Jackson Anderson, Henrik Forsten, Vincent Rieß, Michael Eller, Noah Sauber, Robert Weikle, William Barnhart, Franz Forstmayr, and GitHub Contributors[*]

1 Context and Motivations

With the rapid proliferation of telecommunication and radio-frequency (RF) applications, so too has demand grown for tools to design and characterize these devices. scikit-rf is a Python package designed to make RF/Microwave engineering both robust and approachable. The package provides a modern, object-oriented library for RF network analysis, circuit building, calibration, and simulation. Besides offering standard microwave network operations, such as reading/writing Touchstone files (`.sNp` files), connecting or de-embedding N-port networks, frequency/port slicing, concatenation or interpolations, it is also capable of advanced operations such as Vector Network Analyzer (VNA) calibrations, time-gating, vector fitting, interpolating between an individual set of networks, deriving network statistical properties and support of Virtual Instruments for direct communication to VNAs. The package also allows straightforward plotting of rectangular plots (dB, magnitude, phase, group delay, etc.), Smith Charts or automated uncertainty bounds.

The project was created in 2009 and has been continually developed since. The package is distributed under the Berkeley Software Distribution (BSD) licence and is actively developed by more than 50 volunteers on GitHub. The project has users in several universities and research institutes around the world as well as corporate users from large companies including Keysight, Rohde & Schwarz, National Instruments, Nvidia,

[*] https://github.com/scikit-rf/scikit-rf/graphs/contributors

and 3M. As of 2021, the package has been downloaded more than 220,000 times since its creation and has been used in over thirty publications.

As the package is developed in Python, it makes it naturally compatible with the rich set of modern scientific Python libraries. Results can be shared and directly reproduced by other researchers using tools such as Binder or Google Colab. Being a Python library, it is also compatible with the robust testing and code coverage frameworks developed for the language, with reproducible modelling approaches using online virtualization services. Being open-source, users of scikit-rf are able to see exactly what the source code is doing and, if a feature does not exist, are able to freely contribute it to extend the library for their work and for others. And, of course, it is free.

2 Basic Usage of scikit-rf

Detailed installation instructions can be found in the scikit-rf documentation (https://scikit-rf.readthedocs.io/en/latest/). For those familiar with Python, it is no different from installing any other package, being distributed through both pip and Conda. After installation, the package can be imported, which is assumed for all the following examples:

```
import skrf
```

The following sections outline some fundamental features of scikit-rf, and include code snippets to provide a starting point for new users.

2.1 Reading and Plotting Networks

The central object in the scikit-rf package is a N-port microwave `Network` object. A `Network` can be created in various ways, for example by reading a Touchstone file named `measured_data.s2p`:

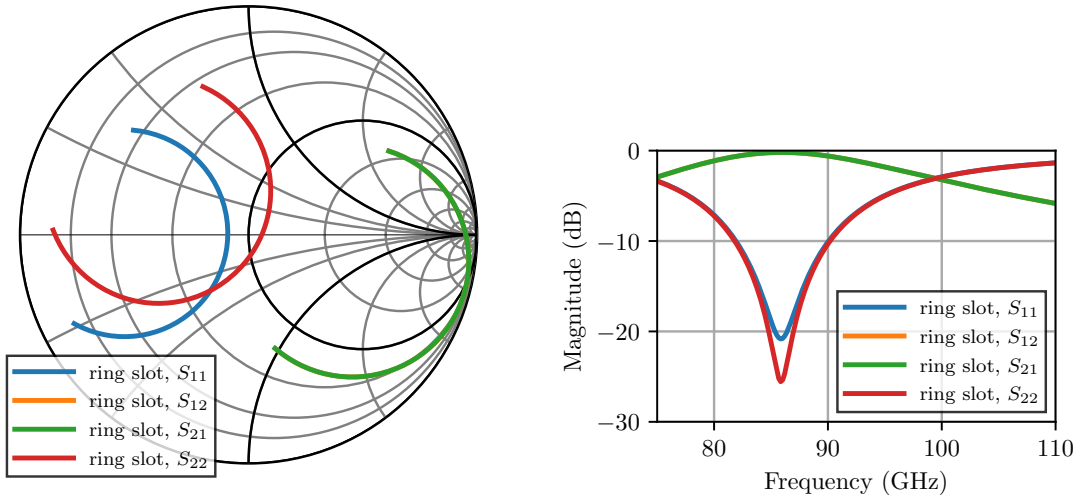
```
example_network = skrf.Network('measured_data.s2p')
```

In order for the reader to reproduce the given examples, a test-case is provided in the package and can be imported via:

```
>>> from skrf.data import ring_slot
>>> ring_slot
2-Port Network: 'ring slot',  75.0-110.0 GHz, 201 pts, z0=[50.+0.j 50.+0.j]
```

The basic attributes of a microwave network are provided by the following properties of the `Network` class:

- `s`: Scattering parameter matrix.
- `z0`: Port impedance matrix.



(a) (b)

Figure 1: Scattering parameters of the `ring_slot` plotted on a Smith chart (a) and in logarithmic magnitude (b).

- `frequency`: Frequency object.

S-parameters are represented by a $(n_{bf} \times N \times N)$ NumPy array [1] (https://numpy.org/), where n_{bf} is the number of frequency points and N is the number of ports of the network. For example, to inspect the 2×2 scattering parameters for the first frequency element:

```
>>> ring_slot.s[0]  # or ring_slot.s[0,:]
array([[-0.50372318+0.4578448j  ,  0.6134571 +0.36678139j],
[ 0.6134571 +0.36678139j, -0.19958433+0.6483347j ]])
```

The `Network` object has numerous other properties and methods, which can found in the documentation. If you are using IPython[2], the Jupyter Notebook[3] or any advanced Python editor, then these properties and methods can be 'tabbed' out on the command line:

```
>>> ring_slot.s<TAB>
ring_slot.s ring_slot.s_arcl
ring_slot.s11 ring_slot.s_arcl_unwrap ...
```

scikit-rf uses the *matplotlib* package [4] to generate various different types plots for the attributes of a `Network`. For example, plotting the ring slot's scattering parameters on the Smith chart can be done in one line; with the output shown in Fig. 1(a):

```
>>> ring_slot.plot_s_smith(lw=2)
```

Frequency ranges of `Network`'s can also be selected by using human-readable strings. For instance, the

following line will plot the logarithmic magnitude of $S_{1,1}$ in decibels for the frequency range of $80-90$ GHz, expressed in "natural" language. The output is shown in Fig. 1(b):

```
>>> ring_slot['80-90ghz'].plot_s_db(m=0, n=0)
```

Other parameters are accessible, such as impedance (Z), admittance (Y), transfer (T) or chain (ABCD) parameters using `ring_slot.z`, `ring_slot.y`, `ring_slot.t` or `ring_slot.a`, respectively. Several other features related to network processing can be found in the scikit-rf documentation. The next sections illustrate a few of them.

2.2 Network Operations

Element-wise mathematical operations on the scattering parameter matrices are accessible through overloaded operators. Hence, `Network`s can be added, subtracted, multiplied along frequency and port axes. For example, to plot the complex difference between a `short` and a `delayshort`:

```
>>> from skrf.data import wr2p2_short as short
>>> from skrf.data import wr2p2_delayshort as delayshort
>>> difference = (short - delayshort)
>>> difference.plot_s_mag(label='Mag of difference')
```

Another common application is calculating the phase difference between two networks using the division operator (`/`):

```
>>> (delayshort/short).plot_s_deg(label='Detrended Phase')
```

2.3 Cascading and De-Embedding

scikit-rf supports the connection of arbitrary ports of N-port networks. Cascading and de-embedding 2-port networks with scikit-rf can also be done though the Python power operator (`**`). For example, as shown in Fig. 2, cascaded connection of two individual 2-port networks `line` and `short`, can simply be calculated with:

```
>>> short = skrf.data.wr2p2_short
>>> line = skrf.data.wr2p2_line
>>> delayshort = line ** short
```

De-embedding can be accomplished by cascading the inverse of a network, as shown in Fig. 3. This is implemented in scikit-rf through the property `Network.inv` property. To de-embed the short from `delayshort`:

```
>>> short_2 = line.inv ** delayshort
>>> short_2 == short
True
```

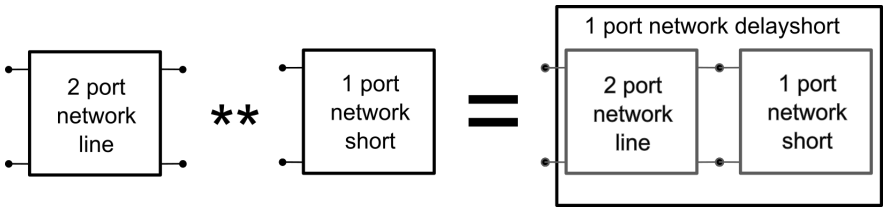


Figure 2: Cascading two 2-Port `Networks` in scikit-rf works with the `**` operator.

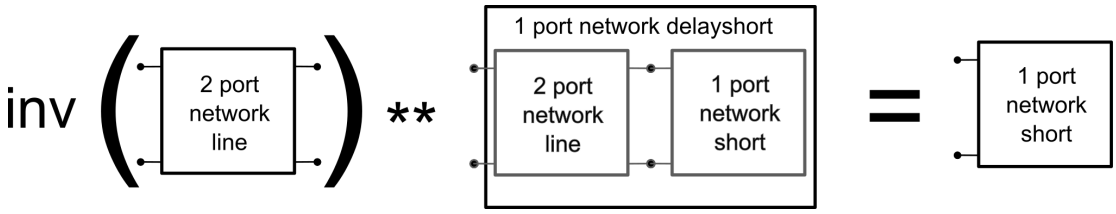


Figure 3: Illustration of de-embedding using the inverse and cascade operators in scikit-rf.

As shown in the previous example, Python comparison operators such as `==` also work with `Network` objects.

2.4 Statistical Analysis

The `NetworkSet` class is useful for analysing multiple `.sNp` files initialized as `Network` objects or read from a directory. `NetworkSet`'s implement the same plotting routines as regular `Network`'s, which is convenient to plot a group of frequency responses. In addition, the `NetworkSet` class enables statistical analyses, such as calculating and plotting the uncertainty with the method `plot_uncertainty_bounds_s_db`. This is demonstrated in the following example, with the outputs shown in Fig. 4:

```
from skrf.data import ro_1, ro_2, ro_3
ntwk_list = skrf.NetworkSet([ro_1, ro_2, ro_3])
ntwk_list.plot_s_db()
ntwk_list.plot_uncertainty_bounds_s_db(label='ro mean with uncertainty, S11')
```

The standard deviations of data in a `NetworkSet` can be applied to a variety of attributes. The method `uncertainty_ntwk_triplet` can be utilized for gain output, similar to the previous example. For that case, the arguments `s_mag` and `3` are used to specify that the uncertainty bounds should be for the magnitude of the scattering parameters within the range of 3 standard deviations. The `Network` objects generated by this function can be saved in various formats to be reused later:

```
ntwk_mean, ntwk_lb, ntwk_ub = ntwk_list.uncertainty_ntwk_triplet('s_mag', 3)
```

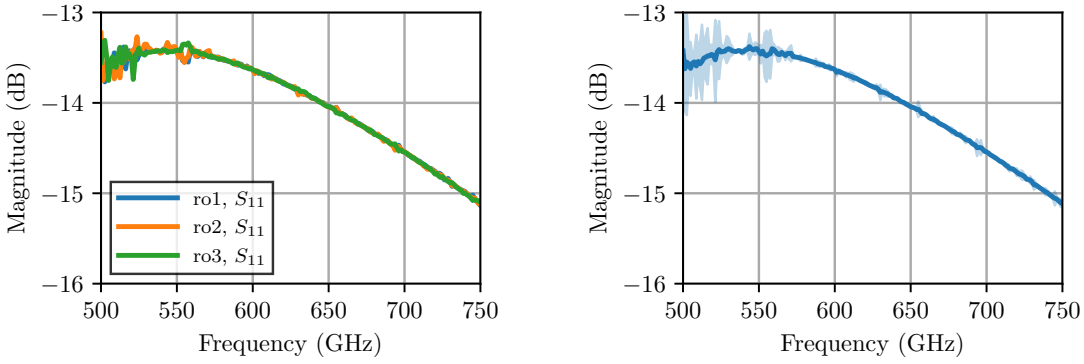


Figure 4: Logarithmic magnitudes of S_{11} of the `ro` example networks. (a) Individual frequency responses. (b) Calculated average and uncertainty bound (3σ).

2.5 Interpolation and Concatenation

A common need is to change the number of frequency points of a `Network`, for instance, but the previous operators and cascading functions require the networks to have matching frequencies:

```
>>> from skrf.data import wr2p2_line1 as line1
# next line fails due to different frequencies
>>> line1 + line
# next line works
>>> line.interpolate_from_f(line1.frequency) + line1
```

A related application is the need to combine networks which cover different frequency ranges. Two `Networks` can be concatenated (stitched) using `stitch()`, which concatenates the `Networks` along their frequency axis. For example, to combine a WR-2.2 `Network` with a WR-1.5 `Network`:

```
>>> from skrf.data import wr2p2_line, wr1p5_line
>>> big_line = skrf.stitch(wr2p2_line, wr1p5_line)
```

2.6 Port Impedance Re-Normalization

Scattering parameters are defined for a given reference impedance Z_0. It can be necessary to re-normalize these parameters to a different reference impedance. This example demonstrates how to use scikit-rf to re-normalize the scattering parameters of a `Network` to different port impedances. Although trivial, this example creates a matched load in 50 Ω and then re-normalizes to a 25 Ω environment, producing a reflection coefficient of 1/3. In case of complex reference impedances, scikit-rf supports both power-waves and pseudo-waves scattering parameter definitions [5].

```
>>> match_at_50 = skrf.wr10.match()
>>> match_at_50
1-Port Network: '',  75.0-110.0 GHz, 1001 pts, z0=[50.+0.j]
>>> match_at_50.s[0]  # S-parameter for the first frequency point
array([[0.+0.j]])
>>> match_at_50.renormalize(25)
>>> match_at_50
1-Port Network: '',  75.0-110.0 GHz, 1001 pts, z0=[25.+0.j]
>>> match_at_50.s[0]
array([[0.33333333+0.j]])
```

3 Calibration

It is possible with scikit-rf to calibrate a device under test (DUT), assuming that an acceptable set of standards have been measured and a corresponding set of ideal responses is known. This may be referred to as *offline* calibration, because it is not occurring on-board the VNA itself. One benefit of this technique is that it provides maximal flexibility for non-conventional calibrations, and preserves all raw data. Self-calibration algorithms, such as Thru-Reflect-Line (TRL)[6], do not require predefined ideal responses.

Several calibration routines are available in scikit-rf, for single port, two-ports or multi-ports networks. Traditional 12-terms [7] or 16-terms [8] error models are available, as well as 8-terms models [9], Short-Open-Load-Thru (SOLT) [10], overdetermined one-port [11], Unknown-Thru [12], Short-Delay-Delay-Load (SDDL) [13], and some special algorithms developed by our contributors. In some cases, it may be necessary or desirable to use a one-port network analyser to determine the full set of scattering parameters of a two-port device. This technique is called one-port two-tier calibration [14] and is also implemented in scikit-rf. A complete list can be found on the scikit-rf website and only a single one-port is given as an example in this section.

3.1 One-Port Example

A calibration in scikit-rf is generated using the `Calibration` object. In general, `Calibration` objects require two arguments: a list of measured `Network`'s and a list of ideal `Network`'s. The following example assumes that a set of measured and ideal network standards are stored in separate directories named `ideals` and `measured`, for example a conventional short-open-load (SOL) calibration kit. These are used to create a one-port `Calibration` and to subsequently correct a measured DUT:

```
from skrf.calibration import OnePort
# reads all Touchstone files located in the specified directory
```

```
my_ideals = skrf.load_all_touchstones_in_dir('ideals/')
my_measured = skrf.load_all_touchstones_in_dir('measured/')
# create a Calibration instance
cal = skrf.OnePort(\
    ideals = [my_ideals[k] for k in ['short','open','load']],
    measured = [my_measured[k] for k in ['short','open','load']],
)
# run calibration algorithm
cal.run()
# apply it to a DUT
dut = skrf.Network('my_dut.s1p')
dut_caled = cal.apply_cal(dut)
```

3.2 Multi-Line TRL Calibration

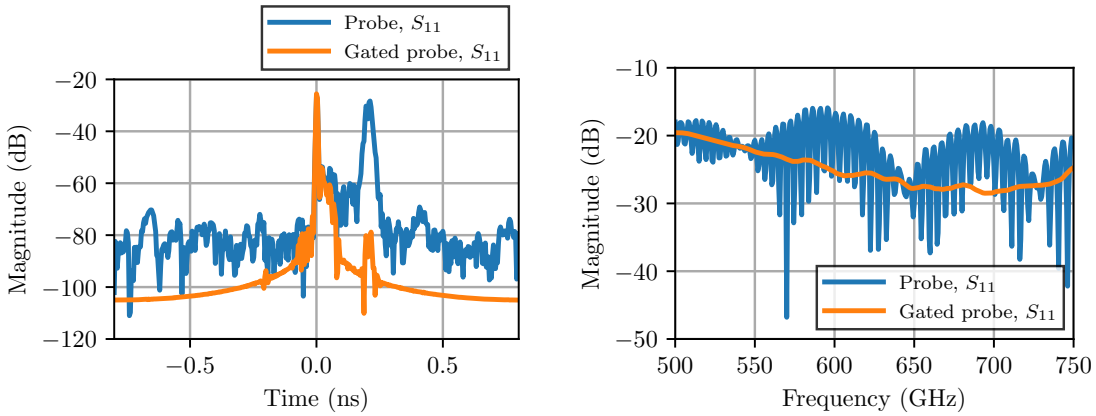
scikit-rf can also be used for wideband Thru-Reflect-Line (TRL) calibration using multiple lines [15]. The necessary calibration standards in TRL calibration involve at least two transmission lines of different lengths and one or more reflective loads. Exact responses of the transmission lines and reflective loads does not need to be known and will be solved during the calibration. Ordinary SOLT or LRRM [16] calibration usually moves the reference plane after the calibration to the SMA connector or probe tip used to contact the calibration standards. TRL calibration can be used to move the reference planes to the transmission lines on the substrate being measured if every measured standard includes a similar launch to the transmission lines. This makes TRL calibration very useful when accurate short, open, load and thru standards cannot be manufactured, or when the measurement reference plane needs to be on the substrate being measured.

4 Time-Domain and Gating

Time-gating is a processing technique which is commonly used to pinpoint a response of interest in the presence of multiple reflections in order to isolate their effects [17, 18]. This is commonly done on-board a VNA. However, if instead the time-gating occurs offline on a computer, the user can keep the raw measurements separate from the processed results. This is important so that the processing algorithm can be altered in the future without remeasuring the data.

In the following example, the time-gating functions of scikit-rf are used to filter out the effects of an undesired reflection. This can be done by using the method `Network.time_gate`, and provide it an appropriate centre and span (in nanoseconds):

```
probe_s11 = skrf.Network('./probe.s2p').s11
```



(a) (b)

Figure 5: Comparison of the original network response S_{11} with its gated version. (a) In the time-domain. (b) In the frequency-domain.

```
probe_s11.name = 'Probe'
probe_s11_gated = probe_s11.time_gate(center=0, span=0.2)
probe_s11_gated.name = 'Gated probe'
# time-domain plot:
s11.plot_s_db_time()
s11_gated.plot_s_db_time()
# frequency-domain plot:
s11.plot_s_db()
s11_gated.plot_s_db()
```

To see the effects of the gate, both the original and gated response are compared. As shown in Fig. 5, the original response shows an interference pattern in the frequency-domain due to this undesirable reflection at 0.2 ns. After gating, the response is cleaned.

5 Circuit Building

scikit-rf enables the building of a circuit with arbitrary topology, consisting of an arbitrary number of N-port `Network`'s connected together. Similar to an electronic circuit simulator, the circuit must have one or more ports connected to the circuit. With the `Circuit` object, the combined responses of the M-port network can be calculated (and thus its network parameters: S, Z, etc.), where M is the number of ports defined. Moreover, the `Circuit` object also allows calculating the scattering matrix S of the entire circuit, that is the "internal" scattering matrices for the various intersections in the circuit. The calculation algorithm is based on [19].

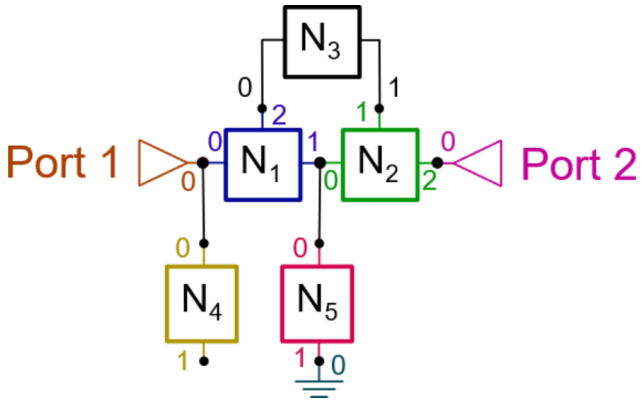


Figure 6: Example of a `Circuit` made of various kinds of N-port `Network`'s.

Fig. 6 illustrates a network with 2 ports, `Network` elements N_i and intersections. In order to define such a circuit, the connection list needs to be defined, which describes how the networks are connected:

```
connections = [
[(network1, network1_port_nb), (network2, network2_port_nb), (network2, network2_port_nb),
    ...],
...
]
```

The connection list to construct the `Circuit` illustrated in Fig. 6 could be:

```
>>> connections = [
[(port1, 0), (network1, 0), (network4, 0)],
[(network1, 1), (network2, 0), (network5, 0)],
[(network1, 2), (network3, 0)],
[(network2, 1), (network3, 1)],
[(network2, 2), (port2, 0)],
[(network5, 1), (ground1, 0)]
]
```

In this example, the `Circuit` elements `port1`, `port2`, `ground1` and all the `network1` to `network5` are assumed to be scikit-rf `Network` objects with the same `Frequency` attribute. The individual networks can have different (real) reference impedances and mismatches are taken into account. Note that port 1 of `network4` is left open, so is not described in the connection list. Once the connection list is defined, the `Circuit` is built with:

```
resulting_circuit = skrf.Circuit(connections)
```

The resulting 2-ports `Network` is obtained with `Circuit.network`:

```
resulting_network = resulting_circuit.network
```

6 Vector Fitting

Microwave circuit design requires simulations in the time or frequency-domain with accurate models of all involved circuit elements. For passive structures, electromagnetic field simulations or measurements can provide accurate network models in the form of sampled frequency responses, but these cannot directly be used in circuit simulators such as SPICE. To translate the sampled frequency responses of a N-port network into a model for circuit simulations, scikit-rf provides an implementation of the well-known vector fitting algorithm to fit the samples in the frequency-domain [20]. The rational basis functions used for the fit enable the subsequent generation of linear equivalent circuits based on resistors, capacitors, inductors, and controlled current and voltage sources to be used in most types of circuit simulators [21].

To summarize the vector fitting approach, the vector $\mathbf{H}(\mathrm{s})$ represents the stack of fitting functions for the $N \cdot N$ individual network responses defined in the Laplace domain with $\mathrm{s} = \sigma + \mathrm{j}\omega$:

$$\mathbf{H}(\mathrm{s}) = \mathbf{d} + \mathrm{s}\,\mathbf{e} + \sum_{k=1}^{K} \frac{\mathbf{z}_k}{\mathrm{s} - p_k}. \tag{1}$$

In (1), $\mathbf{H}(\mathrm{s})$ includes a series of K rational fractions with a common set of poles p_k for all the responses of the network, but with individual zero vectors $\mathbf{z}_k$, as well as individual constant and proportional vectors $\mathbf{d}$ and $\mathbf{e}$, respectively.

For example, the scattering parameters of a 2-port can be subjected to vector fitting with K poles. The objective is therefore to match the original network samples at all sampling frequencies $\omega_m \in [\omega_1, \omega_2, \cdots, \omega_M]$:

$$\begin{pmatrix} S_{11}(\omega_m) \\ S_{12}(\omega_m) \\ S_{21}(\omega_m) \\ S_{22}(\omega_m) \end{pmatrix} \overset{!}{=} \begin{pmatrix} d_{11} + \mathrm{j}\omega_m e_{11} + \sum_{k=1}^{K} \frac{z_{k,11}}{\mathrm{j}\omega_m - p_k} \\ d_{12} + \mathrm{j}\omega_m e_{12} + \sum_{k=1}^{K} \frac{z_{k,12}}{\mathrm{j}\omega_m - p_k} \\ d_{21} + \mathrm{j}\omega_m e_{21} + \sum_{k=1}^{K} \frac{z_{k,21}}{\mathrm{j}\omega_m - p_k} \\ d_{22} + \mathrm{j}\omega_m e_{22} + \sum_{k=1}^{K} \frac{z_{k,22}}{\mathrm{j}\omega_m - p_k} \end{pmatrix}. \tag{2}$$

The fitting process involves running an iterative least-squares algorithm [20], which is implemented in scikit-rf including the speed improvements proposed in [22] and [23].

In scikit-rf, the class `VectorFitting` is instanced with the `Network` to be fitted. A subsequent call of the `vector_fit` routine starts the fitting process with the number of real and complex-conjugate poles defined in the function arguments. Once the fitting is finished, the function `write_spice_subcircuit_s` can be called to generate a SPICE subcircuit file based on the fitted poles $\mathbf{p}$, zeros $\mathbf{z}$, constants $\mathbf{d}$, and proportional coefficients $\mathbf{e}$ of the scattering parameter responses:

```
nw = skrf.Network('example.s4p')
vf = skrf.VectorFitting(nw)
```

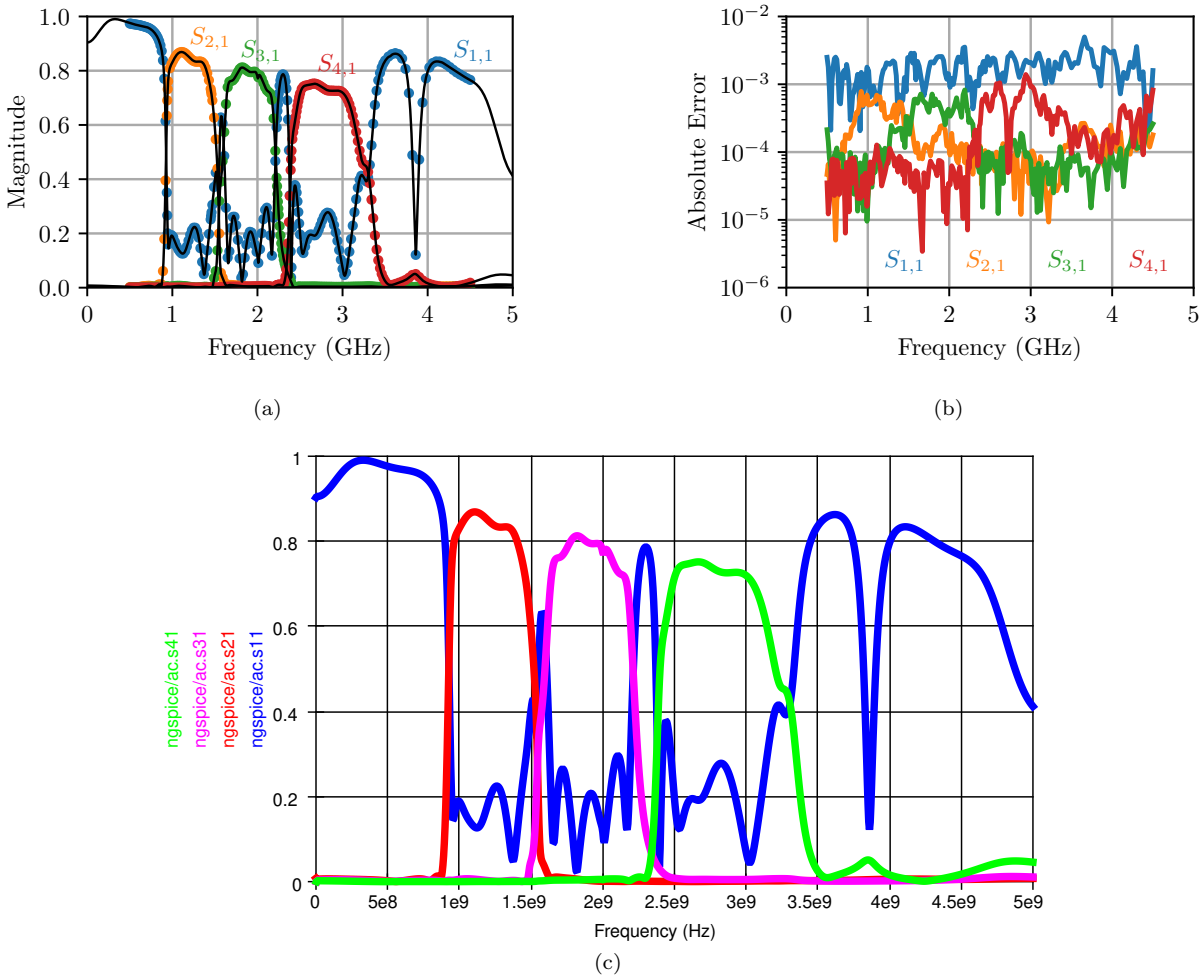


Figure 7: Selected scattering parameter responses of a vector fitted noisy 4-port example network. (a) Magnitudes in linear scale with markers for the samples and solid lines for the fit. (b) Absolute error magnitudes of the respective fits: absolute error $= |S_{\mathrm{fit},i,1} - S_{\mathrm{sample},i,1}|$. (c) Simulation of the scattering parameters with *ngspice* using the exported SPICE subcircuit.

```
vf.vector_fit(n_poles_real=2, n_poles_cmplx=32)
vf.write_spice_subcircuit_s('example.sp')
```

In Fig. 7(a) and 7(b), four of the 16 vector fitted scattering parameter responses of the example network are plotted and compared to the original network samples, showing an absolute error of less than 0.01. SPICE simulations using the exported subcircuit file in *ngspice* [24] achieve a similar accuracy. The simulated scattering parameter obtained from an AC simulation are shown in Fig. 7(c).

The quality of the fit strongly depends on the choice of real and complex-conjugate starting poles, which should suit the number of resonances in the responses. Measurement noise in the sampled responses, as included in this 4-port example, further contributes to the deviation of the fit with the smooth basis functions

used by the vector fitting method. The translation of the fitting parameters into the SPICE equivalent subcircuit is straightforward, and the accuracy is only limited by rounding errors. However, the accuracy of the simulation results also depends on the tolerance settings of the simulator.

7 Conclusion

scikit-rf is an open-source Python package produced for RF/Microwave engineering. The package provides a modern, object-oriented library for RF network analysis, circuit building and calibration. Below is a current non-exhaustive feature list of scikit-rf (as of version 0.18).

- Microwave network operations
  - Read/Write touchstone (.sNp) files
  - S/Z/Y/ABCD/T - parameter conversions
  - Arithmetic operations on scattering parameters
  - Mixed mode and single ended port conversion
  - Cascade/De-embed Networks
  - Frequency and port slicing and concatenation.
  - Assembly of multiple-port networks
  - Vector fitting (S, Z, Y parameters) and export of equivalent SPICE subcircuits (based on S parameters only)
- Sets of Networks:
  - Statistical properties of a set of Network
  - Interpolation between a set of Network (frequency or parametric)
  - Methods to sort and visualize set behaviour
- Plotting abilities:
  - Rectangular plots (dB, magnitude, phase, group delay)
  - Smith chart
  - Automated uncertainty bounds
- Calibration Routines:
  - One-Port: SOL, Least Squares, SDDL

  - Two-Port: TRL, Multiline TRL, SOLT, Unknown Thru, 8/16-Term
  - Partial : Enhanced Response, One-Port Two-Path
- Virtual Instruments (completeness varies by model)
  - VNAs: PNA, PNAX, ZVA, HP8510, HP8720
  - SA: HP8500
  - Others: ESP300
- Transmission Line Physics:
  - Distributed Circuit, Coaxial/Coplanar/Rectangular/Circular Waveguides, Free-space,
  - Transmission line voltage, current, power and losses calculations
  - Complex characteristic impedance support

In addition to these rich features, there are also GUIs for time gating and sNp file viewing, with source code available at https://github.com/scikit-rf/dash-apps and functional demos hosted by Plotly at https://dash-gallery.plotly.host/Portal/.

8 Contributing

scikit-rf is a free and open-source project. For those seeking help or reporting bugs, there is a public mailing list and GitHub issues tracker which can be found on the scikit-rf website (https://scikit-rf.org). If you would to participate in development, first join GitHub (https://github.com) and then take a look at the scikit-rf development pages. For companies or individuals who need private support and development services, feel free to contact 810 Labs LLC (http://810lab.com/).

Acknowledgment

scikit-rf would not be possible without the feedback, bug fixes, and new features contributed by its user community. Visit https://github.com/scikit-rf/scikit-rf/graphs/contributors to see a full list of contributors.

References

[1] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane,

J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array Programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[2] F. Perez and B. E. Granger, "IPython: A System for Interactive Scientific Computing," *Computing in Science Engineering*, vol. 9, no. 3, pp. 21–29, 2007.

[3] B. E. Granger and F. Pérez, "Jupyter: Thinking and Storytelling With Code and Data," *Computing in Science Engineering*, vol. 23, no. 2, pp. 7–14, 2021.

[4] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[5] D. Williams, "Traveling Waves and Power Waves: Building a Solid Foundation for Microwave Circuit Theory," *IEEE Microw. Mag.*, vol. 14, no. 7, pp. 38–45, 2013.

[6] G. Engen and C. Hoer, "Thru-Reflect-Line: An Improved Technique for Calibrating the Dual Six-Port Automatic Network Analyzer," *IEEE Trans. Microw. Theory Techn.*, vol. 27, no. 12, pp. 987–993, 1979.

[7] R. B. Marks, "Formulations of the Basic Vector Network Analyzer Error Model Including Switch-Terms," in *Proc. 50th ARFTG Conf. Dig.*, vol. 32, pp. 115–126, 1997.

[8] K. J. Silvonen, "Calibration of 16-Term Error Model," *Electronics Letters*, vol. 29, no. 17, pp. 1544–1545, 1993.

[9] R. Speciale, "A Generalization of the TSD Network-Analyzer Calibration Procedure, Covering n-Port Scattering-Parameter Measurements, Affected by Leakage Errors," *IEEE Trans. Microw. Theory Techn.*, vol. 25, no. 12, pp. 1100–1115, 1977.

[10] W. Kruppa and K. Sodomsky, "An Explicit Solution for the Scattering Parameters of a Linear Two-Port Measured with an Imperfect Test Set (Correspondence)," *IEEE Trans. Microw. Theory Techn.*, vol. 19, no. 1, pp. 122–123, 1971.

[11] R. Bauer and P. Penfield, "De-Embedding and Unterminating," *IEEE Trans. Microw. Theory Techn.*, vol. 22, no. 3, pp. 282–288, 1974.

[12] A. Ferrero and U. Pisani, "Two-Port Network Analyzer Calibration Using an Unknown 'Thru'," *IEEE Microw. Guided Wave Lett.*, vol. 2, no. 12, pp. 505–507, 1992.

[13] Z. Liu and R. Weikle, "A Reflectometer Calibration Method Resistant to Waveguide Flange Misalignment," *IEEE Trans. Microw. Theory Techn.*, vol. 54, no. 6, pp. 2447–2452, 2006.

[14] J. Ou and M. Caggiano, "Determine Two-Port S-Parameters from One-Port Measurements Using Calibration Substrate Standards," in *Proc. 2005 Electron. Compon. and Techn. (ECTC)*, pp. 1765–1768 Vol. 2, 2005.

[15] R. Marks, "A Multiline Method of Network Analyzer Calibration," *IEEE Trans. Microw. Theory Techn.*, vol. 39, no. 7, pp. 1205–1215, 1991.

[16] A. Davidson, K. Jones, and E. Strid, "LRM and LRRM Calibrations with Automatic Determination of Load Inductance," in *Proc. 36th ARFTG Conf. Digest*, vol. 18, pp. 57–63, 1990.

[17] H. M. Cronson and P. G. Mitchell, "Time-Domain Measurements of Microwave Components," *IEEE Trans. Instrum. Meas.*, vol. 22, no. 4, pp. 320–325, 1973.

[18] C. Bennett and G. Ross, "Time-Domain Electromagnetics and Its Applications," *Proc. IEEE*, vol. 66, no. 3, pp. 299–318, 1978.

[19] P. Hallbjörner, "Method for Calculating the Scattering Matrix of Arbitrary Microwave Networks Giving Both Internal and External Scattering," *Microw. Opt. Technol. Lett.*, vol. 38, no. 2, pp. 99–102, 2003.

[20] B. Gustavsen and A. Semlyen, "Rational Approximation of Frequency Domain Responses by Vector Fitting," *IEEE Trans. Power Del.*, vol. 14, no. 3, pp. 1052–1061, 1999.

[21] G. Antonini, "SPICE Equivalent Circuits of Frequency-Domain Responses," *IEEE Trans. Electromagn. Compat.*, vol. 45, no. 3, pp. 502–512, 2003.

[22] B. Gustavsen, "Improving the Pole Relocating Properties of Vector Fitting," *IEEE Trans. Power Del.*, vol. 21, no. 3, pp. 1587–1592, 2006.

[23] D. Deschrijver, M. Mrozowski, T. Dhaene, and D. De Zutter, "Macromodeling of Multiport Systems Using a Fast Implementation of the Vector Fitting Method," *IEEE Microw. Wireless Compon. Lett.*, vol. 18, no. 6, pp. 383–385, 2008.

[24] "Website of the ngspice project." http://ngspice.sourceforge.net/. [Online].