mwavepy Manual

Alex Arsenovic

10/03/2010

Contents

1 About

mwavepy is a compilation of functions and class's for microwave/RF engineering written in python. It is useful for things such as touchstone file manipulation, calibration, data analysis, data acquisition, and plotting. **mwavepy** can be used interactively through the python interpreter, or in scripts.

mwavepy started when I began to compile all of my personal functions into a single program. I then realized that creating some abstract entities such as a n-port network, calibration instance, and virtual instruments, would make a lot of aspects of data analysis faster. It did.

Because I am not a professional developer, I can only implement what is feasible time-wise, which translates into 'things I need immediatly'. But, I have tried to structure everything with scalability in mind, so others can add onto **mwavepy.**

2 Installation

2.1 Dependencies

The requirements are basically a python environment setup to do numerical/scientific computing. If you are new to python, you should consider usingpythonxy, which provides everything you need to get started.

Here is a list of the requirements,

Necessary

- Python >= 2.6
- matplotlib
- numpy
- scipy

Recomended

- ipython (for interactive shell)

Optional

- pyvisa - for instrument control
- pythics- for VI gui interface design

2.2 Platform independent

Python has many choices for module installation, listed here are installation instructions using setuptools, distutils. All of these assume you have installed the required dependencies.

2.2.1 current version, using svn

svn will get you the most up-to-date version of the mwavepy, docs, and examples, but may have bugs. Check-out using,

```
svn checkout http://mwavepy.googlecode.com/svn/trunk/ mwavepy-read-only
```

Install mwavepy by cd'ing into the mwavepy directory, and running

```
python setup.py install
```

2.2.2 Source Package

There are also pre-made releases available. These can be installed by using setuptools[1] or distutils.

setuptools Open a terminal and type:

```
easy_install mwavepy
```

This should download and install mwavepy.

distutils Download and extract a source package from the mwavepy website. Open up terminal, cd in mwavepy's directory, and type,

```
python setup.py install
```

2.3 Windows executable

Although this may be easier than instaling from svn or a source package, the windows executable will not provide the documentation or examples. Either, way Install all python modules listed under Requirements, or install pythonxy. Then you can download and run the windows installer from the mwavepy website

2.4 Linux

I have yet to make a package for any specific distribution, so linux users will have to follow the platform independent directions. However, installing the requirements in a debian-based linux system is much easier than with windows,

```
sudo apt-get install python-numpy python-scipy python-matplotlib ipython python
```

You will probably have to go fetch pyvisa yourself, or use easy_install.

[1] http://pypi.python.org/pypi/setuptools

3 Quick Intro

This is a quick intro to get the reader comfortable working with **mwavepy**. **mwavepy**, like all of python, can be used in scripts or through the python interpreter. Fire up a python terminal (or IPython), and import the mwavepy module

```
import mwavepy as mv
```

From here all mwavepy's functions can be accessed through the variable 'mv'. In IPython you can use the autocomplete feature to inspect a module by typing mv.[hit tab], and all the classes and functions will be listed. Help for any of the functions can be accessed by typing,

```
help mv.function
```

For our first example, lets load up the data from a touchstone file (default format for VNA's, ie .s2p). Distributed with mwavepy should be a folder called examples. cd into this folder, where you will find an example touchstone file containig data for a calibrated horn antenna. **mwavepy** has a class which represents a n-port network, called Network. It can be initialized from the contents of a touchstone file like so,

```
horn = mv.Network('horn.s1p')
```

From here you can tab out the contents of the newly created Network by typing horn.[hit tab]. You can get help on the various functions as described above. Some of the plotting functions can are illustrated in chapter 4. The base storage format for a Network's data is in scattering parameters, these can be accessed by the property, 's'. Basic element-wise arithmetic can also be done on the scattering parameters, through operations on the Networks themselves. For instance if you want to form the complex division of two Networks scatering matrices,

```
horn2 =  mv.Network('horn.s1p')
horn_diff = horn/horn2
```

This can also be used to implement averaging

```
horn_average = (horn+horn2)/2
```

Other non-elementwise operations are also available, such as cascading and de-embeding two-port networks. For instance the composit network of two, two-port networks is formed using the power operator (**),

```
composit_ntwk = horn ** horn2
```

De-embeding can be accomplished by using the floor division (//) operator

```
horn2 = composit_nwtk //horn
```

When you are done with a network you can save it back to a touchstone,

```
horn2.write_to_touchstone('horn2')
```

4 Basic Usage

4.1 Basic Plotting

```
import mwavepy as mv
import pylab


# create a Network type from a touchstone file of a horn antenna
horn = mv.Network('horn.s2p')

# plot magnitude of S11
pylab.figure(1)
pylab.title('Return Loss (Mag)')
horn.plot_s_db(m=0,n=0) # m,n are S-Matrix indecies
```

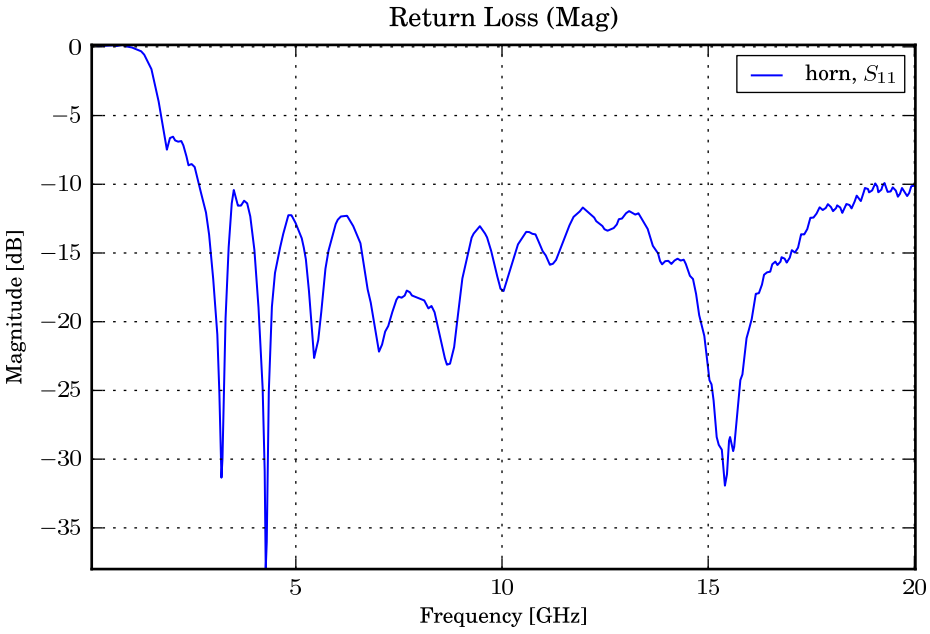

Figure 4.1:

```

# plot phase of S11
pylab.figure(2)
pylab.title('Return Loss (Phase)')
# all keyword arguments are passed to matplotlib.plot command
horn.plot_s_deg(0,0, label='Broadband Horn Antenna', color='r', linewidth=2)
```

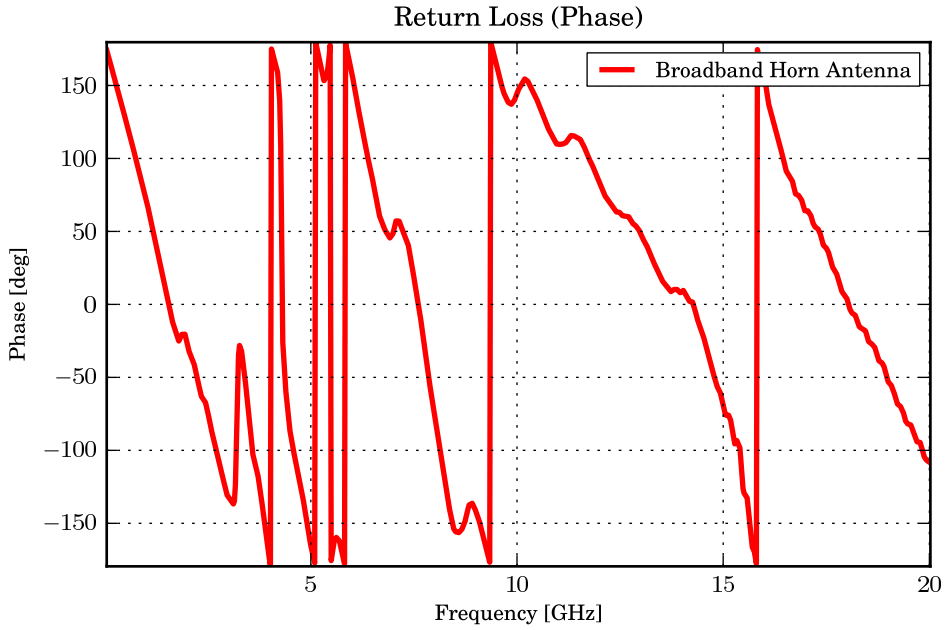

Figure 4.2:

```
# plot unwrapped phase of S11
pylab.figure(3)
pylab.title('Return Loss (Unwrapped Phase)')
horn.plot_s_deg_unwrapped(0,0)
```

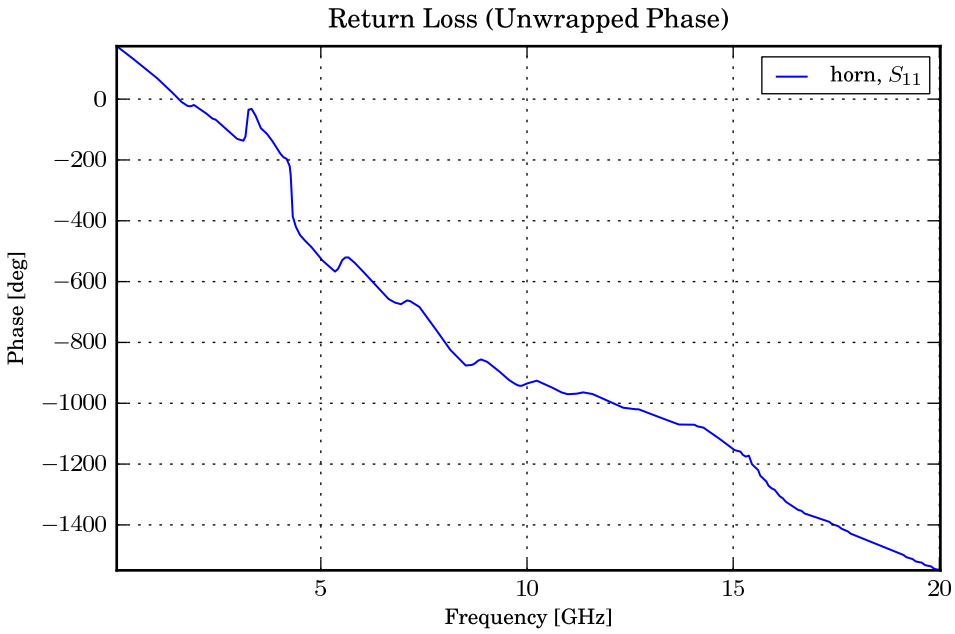

Figure 4.3:

```
# plot complex S11 on smith chart
pylab.figure(5)
horn.plot_s_smith(0,0, show_legend=False)
pylab.title('Return Loss, Smith')
```

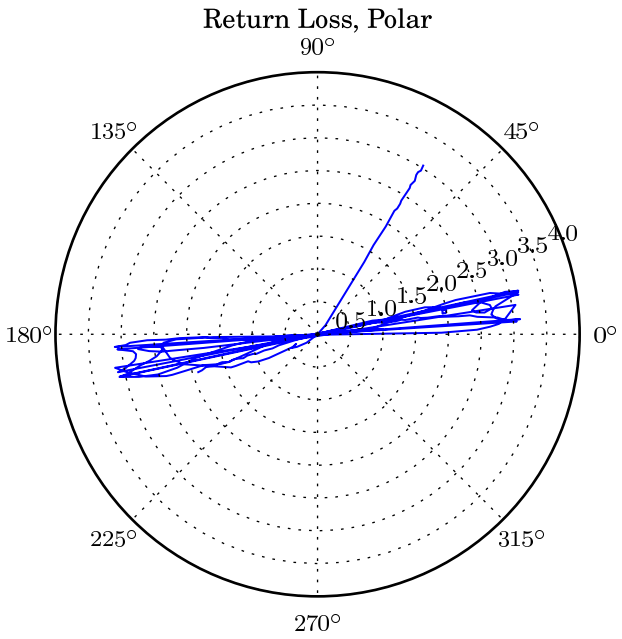

Figure 4.4:

```
# plot complex S11 on smith chart
pylab.figure(5)
horn.plot_s_smith(0,0, show_legend=False)
pylab.title('Return Loss, Smith')
```

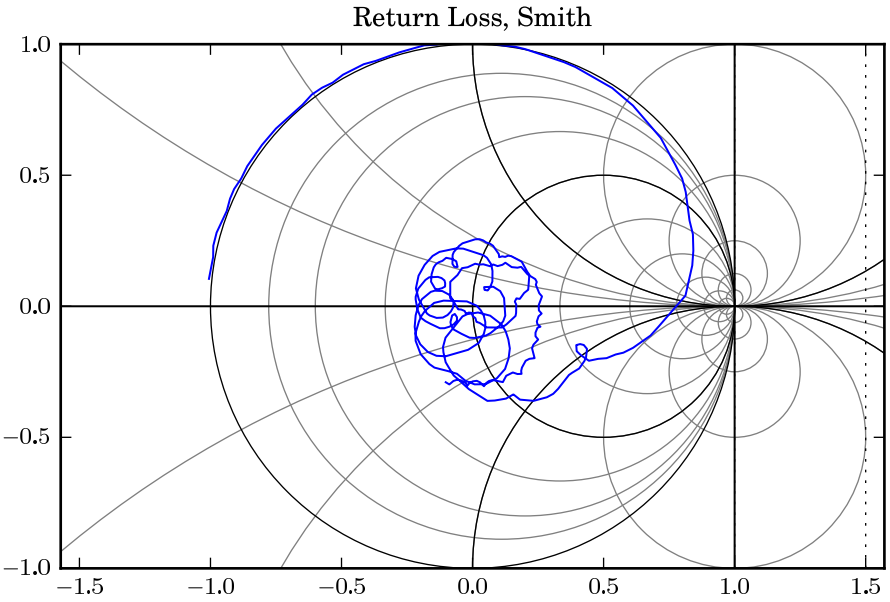

Figure 4.5:

```
# uncomment to save all figures,
#mvy.save_all_figs('.', format = ['png','eps'])

# show the plots
pylab.show()
```

4.2 One-port Calibration

```
import mwavepy as mv


dir = '.'
raw = mv.load_all_touchstones(dir,f_unit = 'ghz')
myfreq = raw[raw.keys()[0]].frequency


cal_std = mv.Calibration(  \
    measured = [\
        raw['Cal3_L1'],\
        raw['Cal3_L2'],\
        raw['Cal3_L3'],\
        raw['Cal3_L4'],\
        raw['Cal3_L5'],\
        ],
    ideals =[\
        raw['short'],\
        raw['delayshort1'],\
        raw['delayshort2'],\
        raw['delayshort3'],\
```

```
        raw['delayshort4'],\
        ],\
    name='Standard Cal',\
    type='one port',\
    frequency= myfreq,\
    is_reciprocal=True,\
    )
```

5 Advanced Usage

6 Architecture

6.1 Module Layout and Inheritance

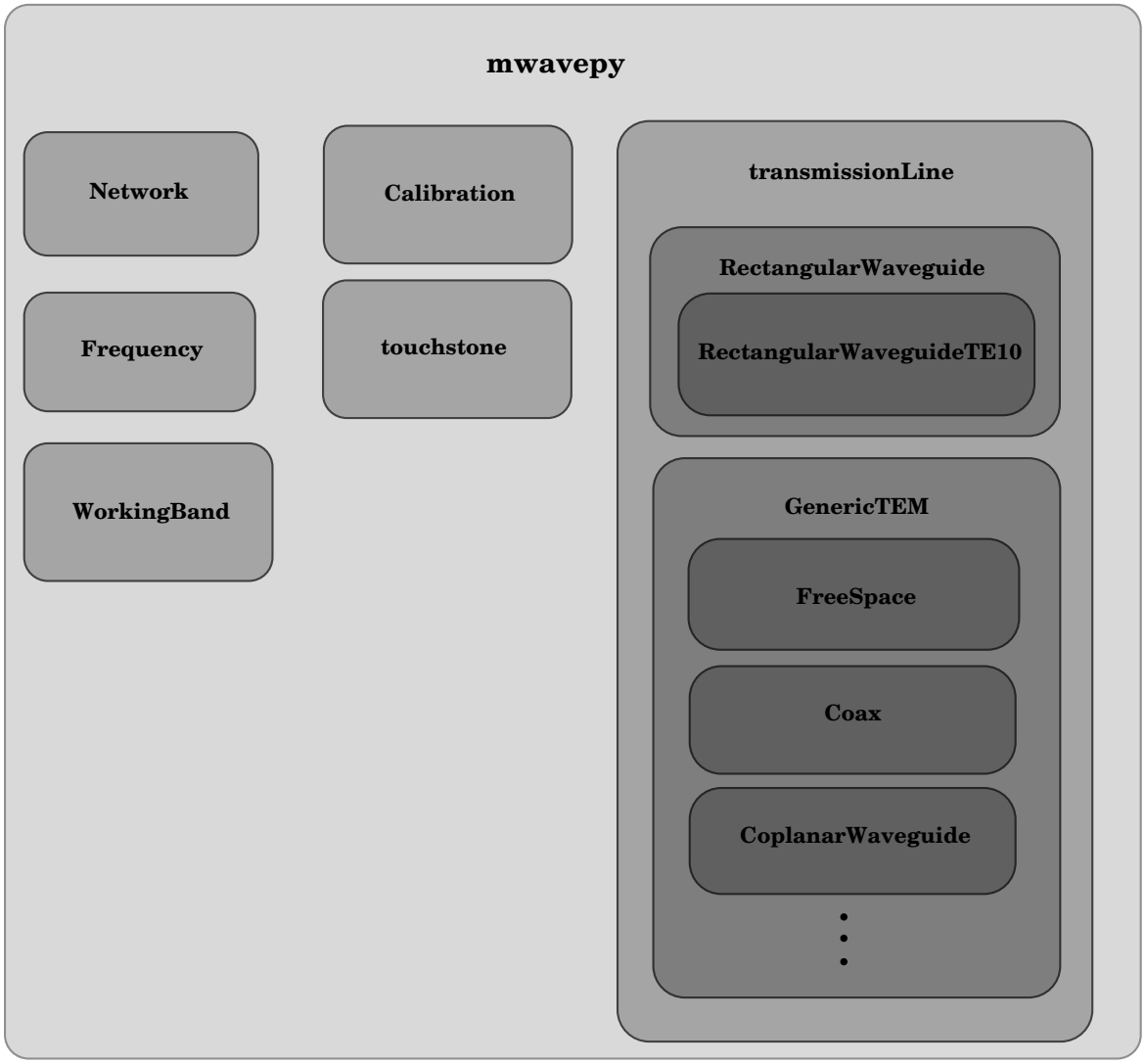

Figure 6.1: Module Layout and inheritance

6.2 Individual Class Architectures

```
import mwavepy as mv
mv.Network('Interface 1/caled/short.s1p').plot_s_deg
```

6.2.1 Frequency

The frequency object was created to make storing and manipulating frequency information easier and more rigid. A major convenience this class provides is the acounting of the frequency vector's unit. Other objects, such as Network, and Calibration require a frequency vector to be meaningful. This vector is commonly referenced when a plot is generated, which one generally doesnt was in units of Hz. If the Frequency object did not exist other objects which require frequency information would have to implement the unit and multiplier bagage.

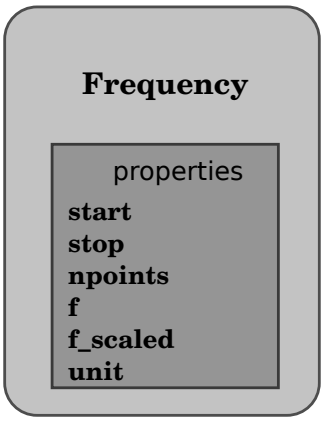

Figure 6.2: Frequency class architecture

Example:

```
freq = mv.Frequency(start = 80,stop=120, npoints = 201, unit='ghz')
```

6.2.2 Network

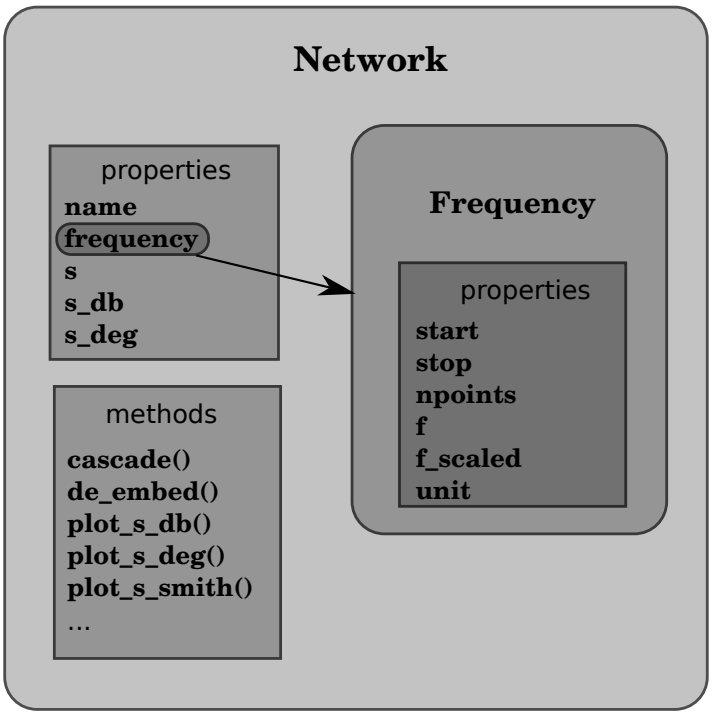

Figure 6.3: Network class architecture

6.2.3 touchstone

The standard file format used to store data retrieved from Vector Network Analyzers (VNAs) is the touchstone file format. This file contains all relevent data of a measured network such as frequency info, network parameters (s, y,z, etc), and port impedance.

6.2.4 WorkingBand

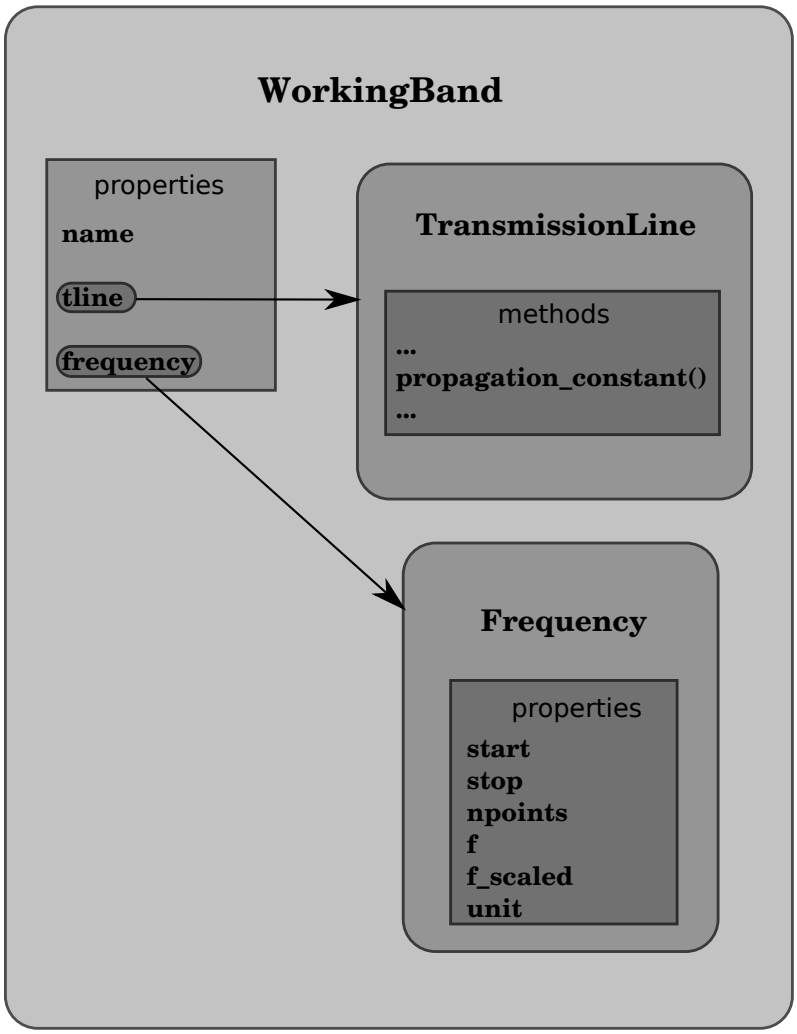

Figure 6.4: WorkingBand class architecture

6.2.5 Calibration

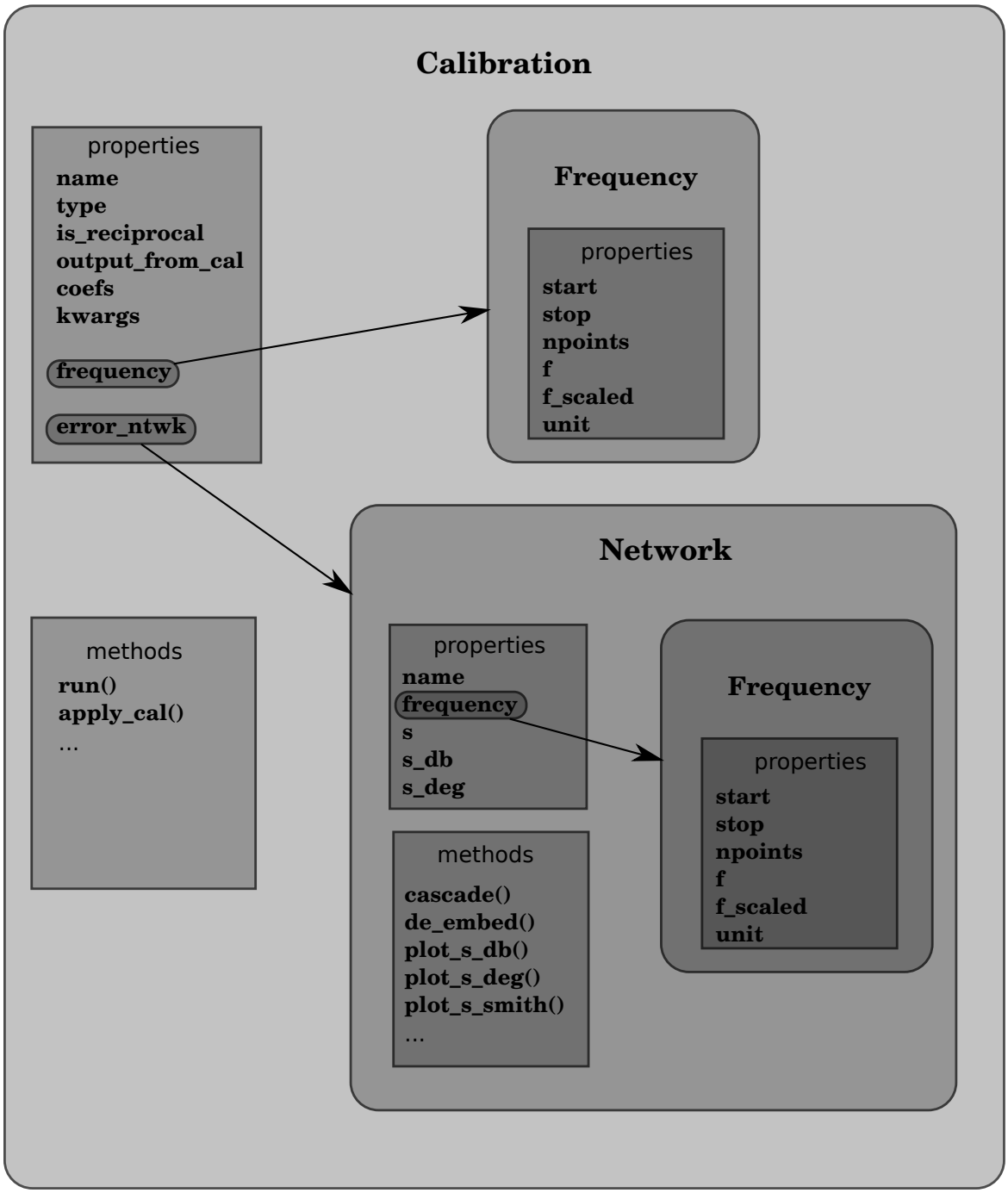

Figure 6.5: Calibration class architecture

7 Future Work