Calibration With Three Receivers

April 15, 2015

```
In [30]: from IPython.display import *
         %nbtoc
```

1 Intro

It has long been known that full error correction is possible given a VNA with only three recievers and no internal coaxial switch. However, since no modern VNA employs such an architecture, the software required to make fully corrected measurements is not available on today's modern VNA's.

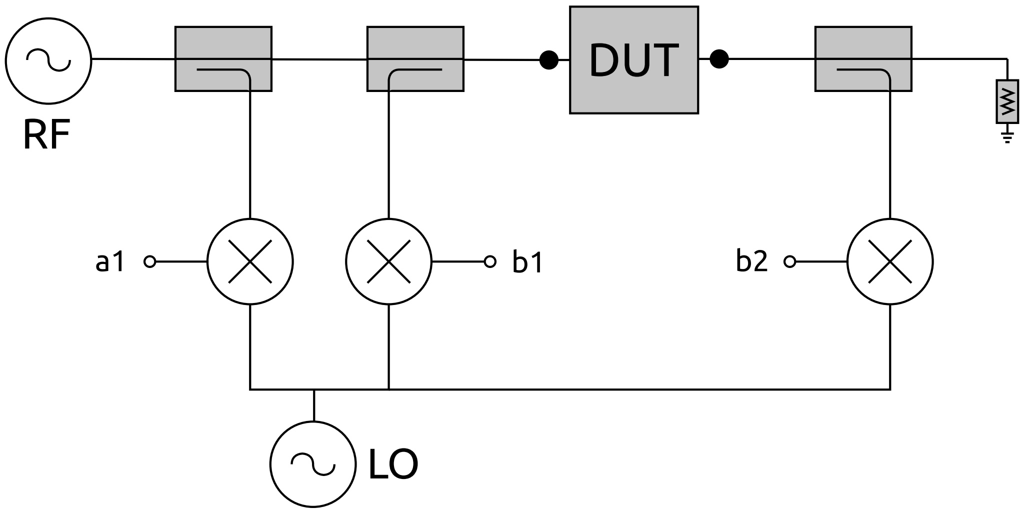
Recently, the application of Frequency Extender units containing only three receivers has become more common. Thus, there is a need for full error correction capability on systems with three receivers and no internal coaxial switch. This document describes how to use scikit-rf to fully correct two-port measurements made on such a system.

2 Theory

A circuit model for a switch-less three receiver system is shown below.

```
In [31]: SVG('pics/vnaBlockDiagramForwardRotated.svg')
```

```
Out[31]:
```



To fully correct an arbitrary two-port, the device must be measured in two orientations, call these forward and reverse. Because there is no switch present, this requires the operator to physically flip the device, and save the pair of measurements. In on-wafer scenarios, one could fabricate two identical devices, one in each orientation. In either case, a pair of measurements are required for each DUT before correction can occur.

While in reality the device is being flipped, one can imaging that the device is static, and the entire VNA circuitry is flipped. This interpretation lends itself to implementation, as the existing 12-term correction can be re-used by simply copying the forward error coefficients into the corresponding reverse error coefficients. This is what `scikit-rf` does internally.

3 Worked Example

This example demonstrates how to create a TwoPortOnePath and EnhancedResponse calibration from measurements taken on a Agilent PNAX with a set of VDI WR-10 TXRX-RX Frequency Extender heads. Comparisons between the two algorithms are made by correcting an asymmetric DUT.

3.1 Read in the Data

The measurements of the calibration standards and DUT's were downloaded from the VNA by saving touchstone files of the raw s-parameter data to disk. This currently reside in the folder `./data/`

```
In [32]: ls data/
```

```
attenuator (forward).s2p      short.s2p
attenuator (reverse).s2p      thru.s2p
load.s2p                      wr15 shim and swg (forward).s2p
quarter wave delay short.s2p  wr15 shim and swg (reverse).s2p
```

These files can be read by scikit-rf into `Networks` with the following.

```
In [33]: import skrf as rf
         raw = rf.read_all_networks('data/')
         # list the raw measurments
         raw.keys()
```

```
Out[33]: ['load',
          'attenuator (reverse)',
          'short',
          'attenuator (forward)',
          'wr15 shim and swg (reverse)',
          'wr15 shim and swg (forward)',
          'thru',
          'quarter wave delay short']
```

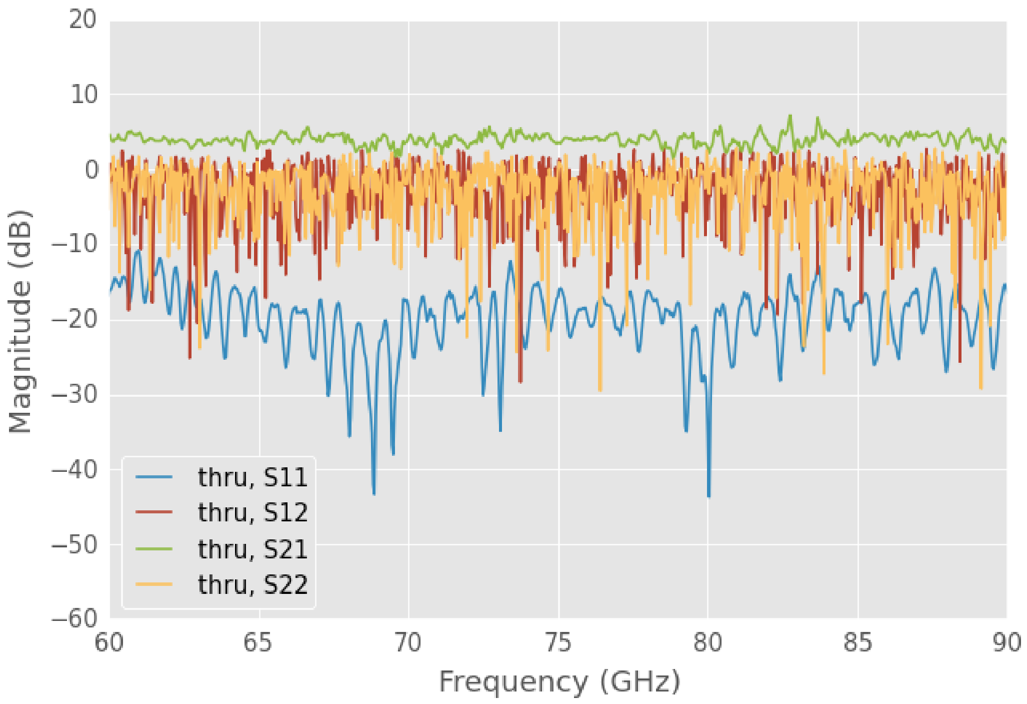
Each `Network` can be accessed through the dictionary `raw`.

```
In [34]: thru = raw['thru']
         thru
```

```
Out[34]: 2-Port Network: 'thru',  60-90 GHz, 721 pts, z0=[ 50.+0.j  50.+0.j]
```

If we look at the *raw* measurement of the flush thru, it can be seen that only S_{11} and S_{21} contain meaningful data. The other s-parameters are noise.

```
In [35]: thru.plot_s_db()
```



3.2 Create Calibration

In the code that follows a TwoPortOnePath calibration is created from corresponding measured and ideal responses of the calibration standards. The measured networks are read from disk, while their corresponding ideal responses are generated using scikit-rf. More information about using scikit-rf to do offline calibrations can be found here.

```
In [36]: from skrf.calibration import TwoPortOnePath
         from skrf.media import RectangularWaveguide
         from skrf import two_port_reflect as tpr
         from skrf import  mil

         # pull frequency information from measurements
         frequency = raw['short'].frequency

         # the media object
         wg = RectangularWaveguide(frequency=frequency, a=120*mil, z0=50)

         # list of 'ideal' responses of the calibration standards
         ideals = [wg.short(nports=2),
                   tpr(wg.delay_short( 90,'deg'), wg.match()),
                   wg.match(nports=2),
                   wg.thru()]

         # corresponding measurments to the 'ideals'
         measured = [raw['short'],
                     raw['quarter wave delay short'],
                     raw['load'],
                     raw['thru']]

         # the Calibration object
         cal = TwoPortOnePath(measured = measured, ideals = ideals )
```

3.3 Apply Correction

There are two types of correction possible with a 3-receiver system.

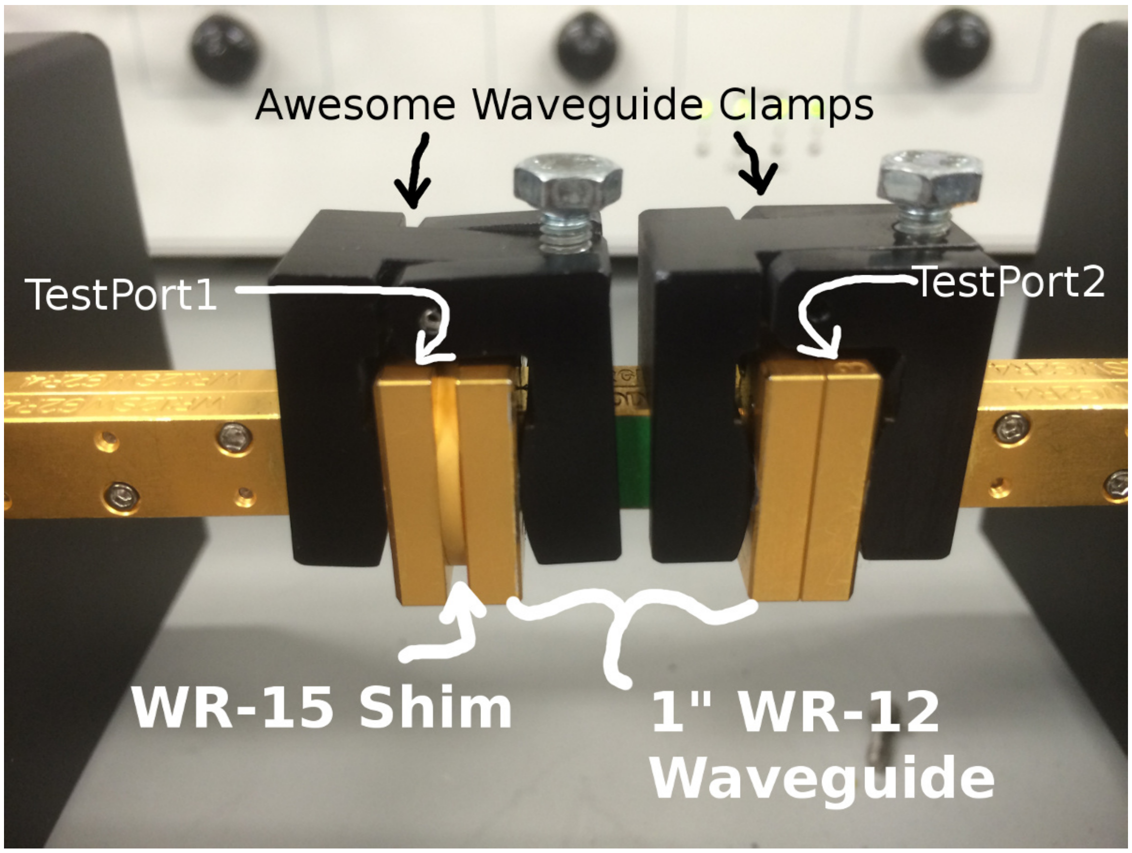
1. Full (TwoPortOnePath)
2. Partial (EnhancedResponse)

`scikit-rf` uses the same `Calibration` object for both, but employs different correction algorithms depending on the `type` of the DUT.

The DUT this example is a WR-15 shim cascaded with a WR-12 1" straight waveguide, as shown in the picture below. Measurements of this DUT are corrected with both *full* and *partial* correction and the results are compared below.

```
In [37]: Image('pics/asymmetic DUT.jpg', width='75%')
```

```
Out[37]:
```



3.3.1 Full Correction (TwoPortOnePath)

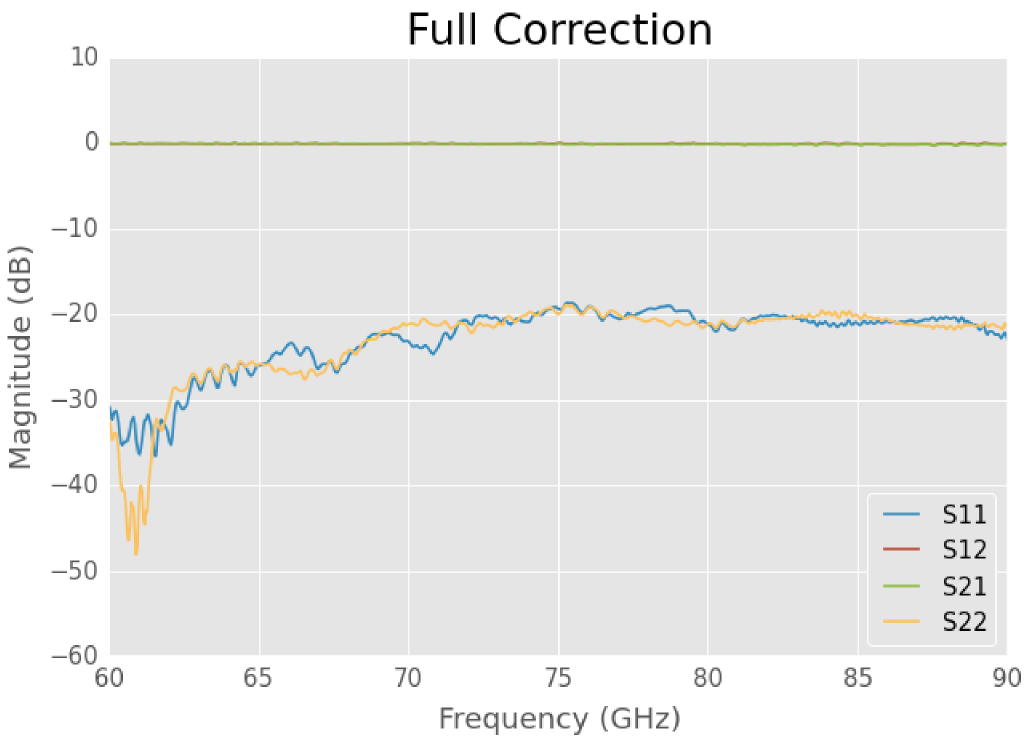
Full correction is achieved by measuring each device in both orientations, **forward** and **reverse**. Meaning the DUT was physically removed, flipped, and re-inserted. The resulting pair of measurements are then passed to the `apply_cal()` function as a tuple. This returns a single corrected response.

```
In [38]: dutf = raw['wr15 shim and swg (forward)']
         dutr = raw['wr15 shim and swg (reverse)']

         corrected_full = cal.apply_cal((dutf, dutr)) # note argument is an ordered tuple

         corrected_full.name = None
         corrected_full.plot_s_db()
         title('Full Correction')
```

```
Out[38]: <matplotlib.text.Text at 0x7f81bde1ef90>
```



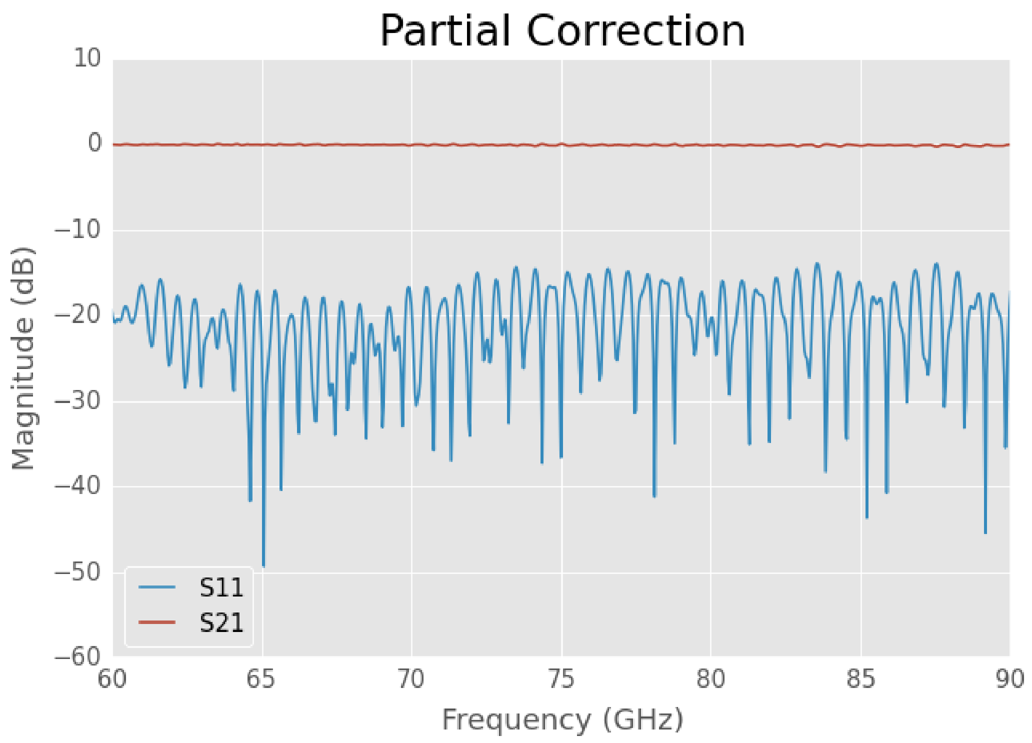
3.3.2 Partial Correction (Enhanced Response)

If you pass a single measurement to the `TwoPortOnePath.apply_cal()` function, then it will use partial correction. This correction is known as `EnhancedResponse`. Depending on the measurment application, this type of correction may be *good enough*, and perhaps the only choice.

```
In [39]: corrected_partial = cal.apply_cal(dutf) # note we pass a single network

         corrected_partial.name = None
         corrected_partial.plot_s_db(0,0)
         corrected_partial.plot_s_db(1,0)
         title('Partial Correction')
```

```
Out[39]: <matplotlib.text.Text at 0x7f81c46bb810>
```



Note the *partially* corrected measurement only has a partially filled s-matrix, while the fully correct measurement has a full s-matrix

```
In [40]: corrected_partial['75ghz'].s

Out[40]: array([[[ 0.01398785+0.00470094j,  0.00000000+0.j        ],
                 [ 0.22933932-0.96883735j,  0.00000000+0.j        ]]])

In [41]: corrected_full['75ghz'].s

Out[41]: array([[[ 0.09106062-0.05667315j,  0.21885438-0.96923775j],
                 [ 0.22778341-0.95953481j,  0.05839562+0.08057083j]]])
```

3.4 Direct Comparison

Below are direct comparisons of the DUT shown above corrected with *full* and *partial* algorithms. It shows that the partial calibration produces a large ripple on the reflect measurements, and slightly larger ripple on the transmissive measurments.

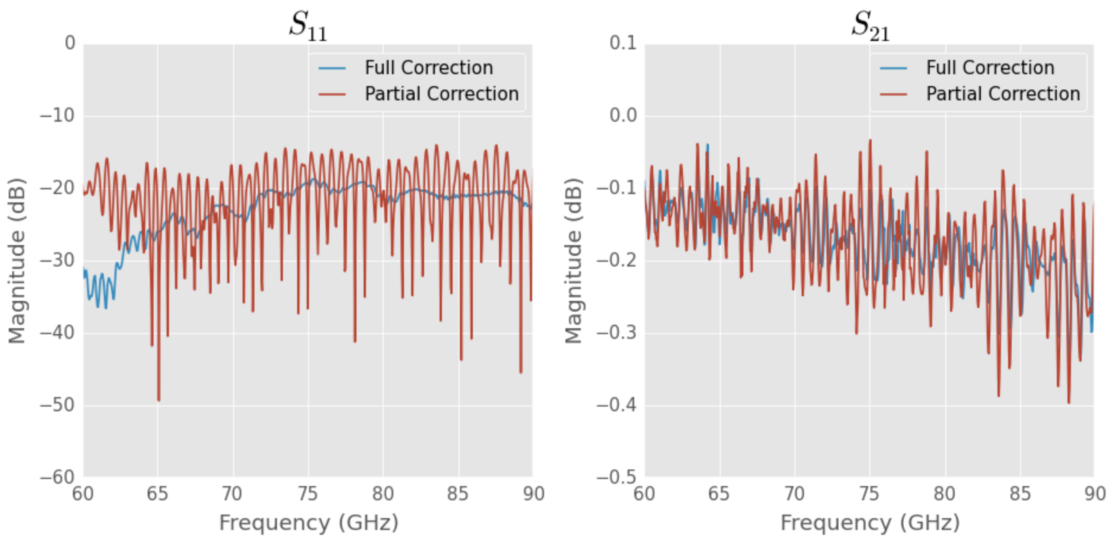
```
In [42]: f, ax = subplots(1,2, figsize=(8,4))

         ax[0].set_title ('$S_{11}$')
         ax[1].set_title ('$S_{21}$')

         corrected_full.plot_s_db(0,0, label='Full Correction', ax = ax[0])
         corrected_full.plot_s_db(1,0, label='Full Correction', ax = ax[1])
```

```
corrected_partial.plot_s_db(0,0, label='Partial Correction',ax=ax[0])
corrected_partial.plot_s_db(1,0, label='Partial Correction',ax=ax[1])


tight_layout()
```



3.5 Formating

```
In [43]: from IPython.core.display import HTML


         def css_styling():
             styles = open("../styles/plotly.css", "r").read()
             return HTML(styles)
         css_styling()

Out[43]: <IPython.core.display.HTML at 0x7f81c4d4d990>
```