WHITEPAPER

EXPERIMENTING WITH OPEN-SOURCE RF ANALYSIS TOOLS:

USING SCIKIT-RF TO EVALUATE FILTER PERFORMANCE

At Knowles Precision Devices, we truly do love all things RF. While we are focused on using our expertise in ceramics and thin film manufacturing to innovate on our high-performance filters, our engineers also enjoy keeping up with the trends impacting the entire RF "ecosystem." This includes dedicating time to experimenting with the RF software modeling and analysis tools we think our customers might find useful. But we don't mean the highly sophisticated software tools many RF design engineers use to perform precise performance analysis on their RF and microwave circuits such as Sonnet, Microwave Office, and Keysight's Genesys (now known as PathWave RF Synthesis).

The tools we enjoy experimenting with are the open-source tools people outside the design team, such as product managers or component and applications engineers, may want to use to test out an idea or to independently look at performance data for a specific filter. For these engineers, heavyweight tools like the ones used by the design team are likely not the right option. Not only are the licenses for these tools fairly pricey, but there is often a steep learning curve involved.

Instead, of asking an RF design engineer with the software license and experience to help run an analysis every time you want to try something out, we have found that there are a number of free open-source tools that are great options for non-design engineers to use for experimenting with filters and plotting data. One open-source RF and microwave network analysis tool we have been working with lately that we feel does a particularly good job is scikit-rf, which is based on the Python programming language.

In this white paper, we share what we have learned about working with scikit-rf to see how "good" a filter's performance is without consulting a design engineer and using their time- and computation-intensive software. We also walk you through our suggestions for how to get started with scikit-rf and show you the details on a variety of example analyses we performed on one of our bandpass filters based on the publicly available S-parameter file for it.

Our Best Practices for Getting Started with scikit-rf

According the scikit-rf website, scikit-rf "is an open-source, BSD-licensed package for RF/Microwave engineering implemented in the Python programming language that provides a modern, object-oriented library for network analysis and calibration which is both flexible and scalable." Based on our experience using scikit-rf, there are several tools we recommend using to simplify the Python programming needed to use this tool. Before getting into some examples of how to use scikit-rf, let's first review our best practices for how to use each of the following tools to simplify performing an analysis in scikit-rf:

- Jupyter
- Matplotlib
- NumPy and SciPy

At Knowles Precision Devices, we truly do love all things RF. While we are focused on using our expertise in ceramics and thin film manufacturing to innovate on our high-performance filters, our engineers also enjoy keeping up with the trends impacting the entire RF "ecosystem."

First, it is key to understand that Python is the programming language used to control scikit-rf. Python is a high-level, object-oriented programming language that is generally easy to work with for math or science programs. While the creator of scikit-rf recommends having some familiarity with Python before using the tool, it is a pretty user-friendly programming language that is easy to write in and get immediate results. There are also tools we recommend using to make it even easier to write the Python code. We like using a Jupyter Notebook to write the Python code interactively in a local web browser. With this open-source web-based computing application, you can also easily share documents that contain live code, equations, visualizations, and narrative text.

If you prefer, you can write your code using Jupyter in an integrated development environment (IDE) rather than in a browser. While the commonly used Anaconda environment works well for this purpose, to stick with the concept of working with easy-to-acquire free tools, we are using Microsoft's free Visual Studio Code (VS Code) editor. VS Code also has some nice built-in support for Jupyter that you can learn how to setup here (just remember to install Python on your machine as well as the VS Code Python Extension). Figure 1 shows a screenshot of Jupyter running in VS Code.

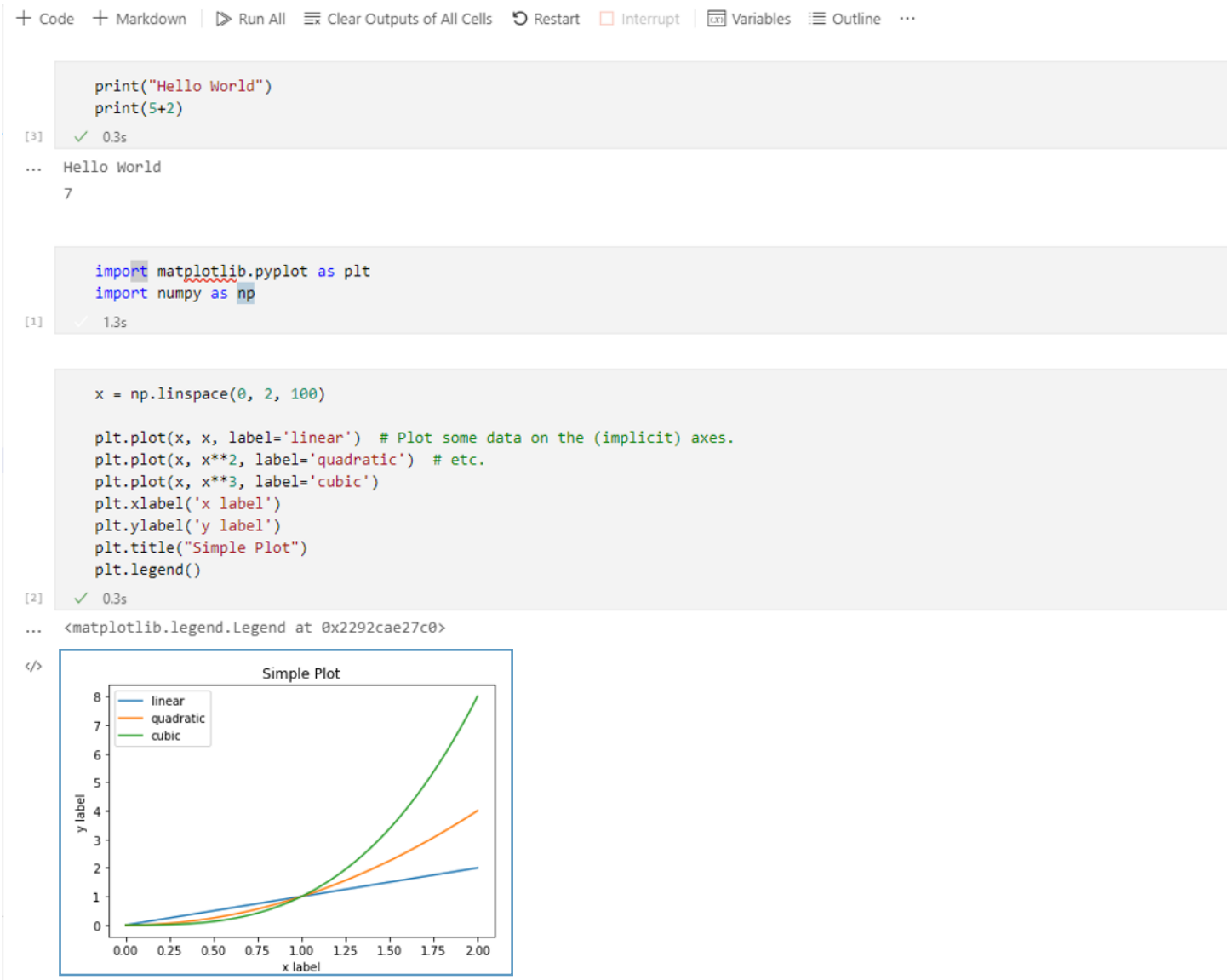

Figure 1 A screenshot of Python code programmed in Jupyter running in a VS Code editor.

To further simplify programming and perform complex functions that are not inherently easy to do in Python, we also recommend using add-on tools such as Matplotlib, NumPy, and SciPy. Matplotlib is a highly flexible plotting library that allows you to create high-quality, interactive plots. NumPy and SciPy are Python libraries that allow you to easily access many of the mathematical functions needed to perform various analyses in scikit-rf. Both Matplotlib and NumPy are used in the example code shown in Figure1.

Evaluating Filter Performance in scikit-rf Using S-Parameter Files

In scikit-rf, the object you will work with the most is called a `Network`, or an N-port microwave object. A `Network` can be created from scratch or built from an imported S-parameter file. An S-parameter file, which is what we use in our example `Networks` in this paper, refers to the scattering matrix of a microwave `Network`. The scattering matrix is a mathematical construct that quantifies how RF energy propagates through a linear multi-port `Network`. We like to think of S-parameters as the Swiss Army knife of RF data as it can tell you quite a bit about the performance of a filter.

For the examples in this paper, we are using the S-parameter file for one of our catalog filters, the B095MB1S, which is a 9.5 GHz surface mount bandpass filter. If you would like to try out these tools by following along with these examples, you can download the S-parameter file for this filter by visiting the bandpass filter section of our website and searching by part number, B095MB1S, in the search box just above the spec table as shown in Figure 2. The search will pull up the filter and will include two links, one for the datasheet and one to download the design files, which will include the S-parameter files.

Show 10 entries Search: B095MB1S

Part Number	Fc (GHz)	FL	FH	IL (@Fc, 25°C)	L x W x H, inches (mm)	Datasheet	Design Files
B095MB1S	9.5	8.9	10	1.75	0.400 (10.16) x 0.150 (3.81) x 0.098 (2.489)	B095MB1S	B095MB1S

Showing 1 to 1 of 1 entries (filtered from 85 total entries) Previous 1 Next

Figure 2 The results of searching for this bandpass filter to obtain the design files.

Once you have the S-parameter file stored on your computer in a location where it is easy to access from within your Jupyter setup, it is best to rename the file with a name that is easy to recognize. For our example, we put the file in a folder called 'xband2' and named the file 'B095MB1S.s2p.' Since Jupyter can see this folder, we can make a `Network` and review the `Network` properties. Below is the Python code we input in Jupyter to call this S-parameter file and create this example `Network` as well as the results.

Code:

```
import skrf as rf

testNtwk = rf.Network('xband2\B095MB1S.s2p')

testNtwk
```

Results:

```
2-Port Network: 'B095MB1S', 2000.0-30000.0 MHz, 1601 pts, z0=[50.+0.j 50.+0.j]
```

Note that in this example, the `testNtwk` is a `Network` object representing a two-port `Network`. The summary information in the results tells us the frequency range of this data set, the number of data points, and the impedance of the `Network`. The `Network` class also comes with convenient built-in methods for plotting and manipulating data. In this example, we can quickly plot the log-magnitude in decibels for the filter's frequency range for four standard S-parameters by calling the `plot_s_db` method (results shown in Figure 3).

Code:

```
testNtwk.plot_s_db()
```

Results:

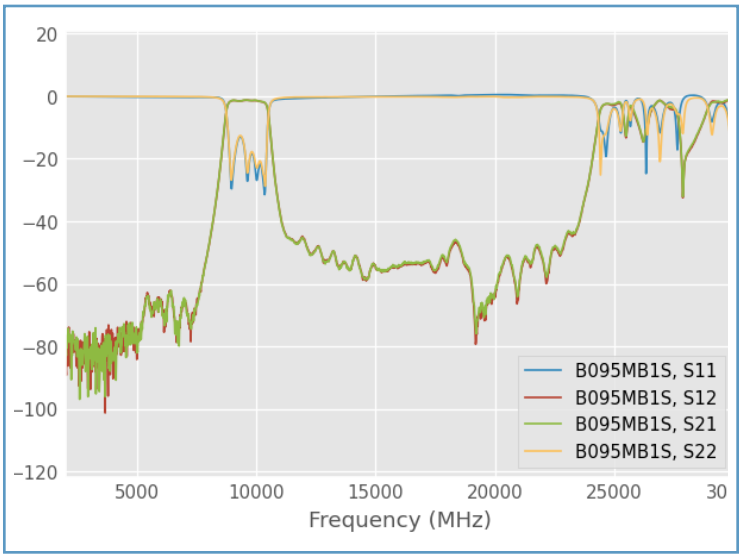

Figure 3 The resulting log-magnitude plot created for this bandpass filter.

Refining Analysis Parameters in scikit-rf

In this last example, we demonstrated how to pull an S-parameter file into scikit-rf to create a plot that looks at the filter's full frequency range. Now, let's look at how we can measure the performance of this filter in a narrower frequency range. This next example script will perform the following functions:

- Import the necessary packages
- Create a `Network` from a specified S-parameter file in a specified folder
- Zoom in on a particular frequency range of interest – 7.5 – 12GHz
- Plot the values for all four S-parameters
- Add a title
- Save the plot in the specified format, using the title as the filename

The resulting plot is shown in Figure 4.

Code:

```
import skrf as rf
import os
%matplotlib inline
from pylab import *
rf.stylely()


# assign directory
directory = 'xband2'

# pick a filename
filename = 'B095MB1S.s2p'

#assign a zoom range
bandStr = '7.5-12ghz'

f = os.path.join(directory, filename)

ntwk = rf.Network(f)
ntwk = ntwk[bandStr]
ntwk.frequency.unit = 'ghz'

ntwk.plot_s_db()
title(filename)

rf.plotting.save_all_figs('.', format=['png','pdf'])
```

Open-source tools such as scikit-rf can be extremely powerful for providing a fairly quick, and free, method for analyzing data and sharing results with your team.

Results:

```
./B095MB1S.s2p.png
./B095MB1S.s2p.pdf
```

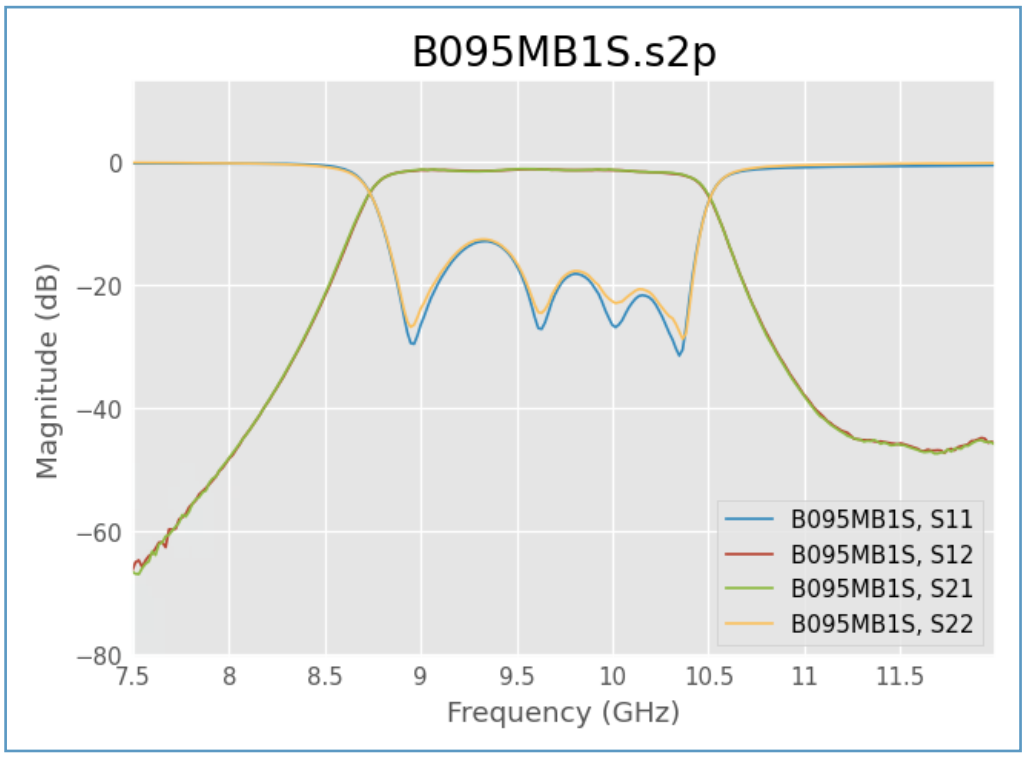

Figure 4 The resulting plot for the frequency range of interest for this bandpass filter.

Now, let's say we are interested in the filter's performance at a particular frequency instead. We can use the same approach to pull this data as we did above, except this time we input a single frequency and then ask for the dB values of each of the complex S-parameters.

Code:

```
for x in ntwk['9.5GHz'].s:

    print(' S11: {result:.2f}db'.format(result=rf.mathFunctions.complex_2_db(x[0,
0])))
    print(' S21: {result:.2f}db'.format(result=rf.mathFunctions.complex_2_db(x[1,
0])))
    print(' S11: {result:.2f}db'.format(result=rf.mathFunctions.complex_2_db(x[0,
1])))
    print(' S22: {result:.2f}db'.format(result=rf.mathFunctions.complex_2_db(x[1,
1])))
```

Results:

```
S11: -17.32db
 S21: -1.18db
 S11: -1.27db
 S22: -16.82db
```

We can also ask scikit-rf to just show the results for a single S-parameter, such as S21, for a certain frequency range. Note, the S-parameters are indexed, with the index starting at 0, so S21 is m=1 and n=0 (results shown in Figure 5).

Code:

```
ntwk.plot_s_db(m=1, n=0)
```

Results:

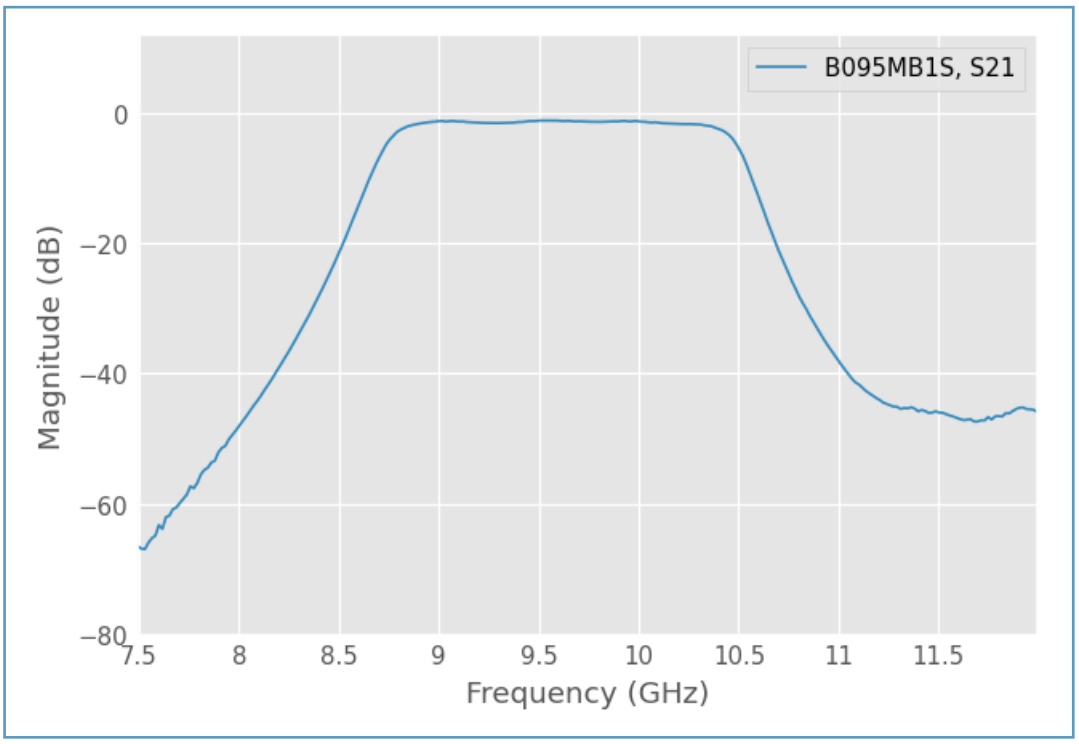

Figure 5 A plot showing the results for a single S-parameter across a defined frequency.

Another use case for scikit-rf is to run an experiment that offers a zoomed-in look at information shown in a filter's datasheet. For this filter, the datasheet shows the measured performance across a wide range of frequencies as shown in Figure 6.

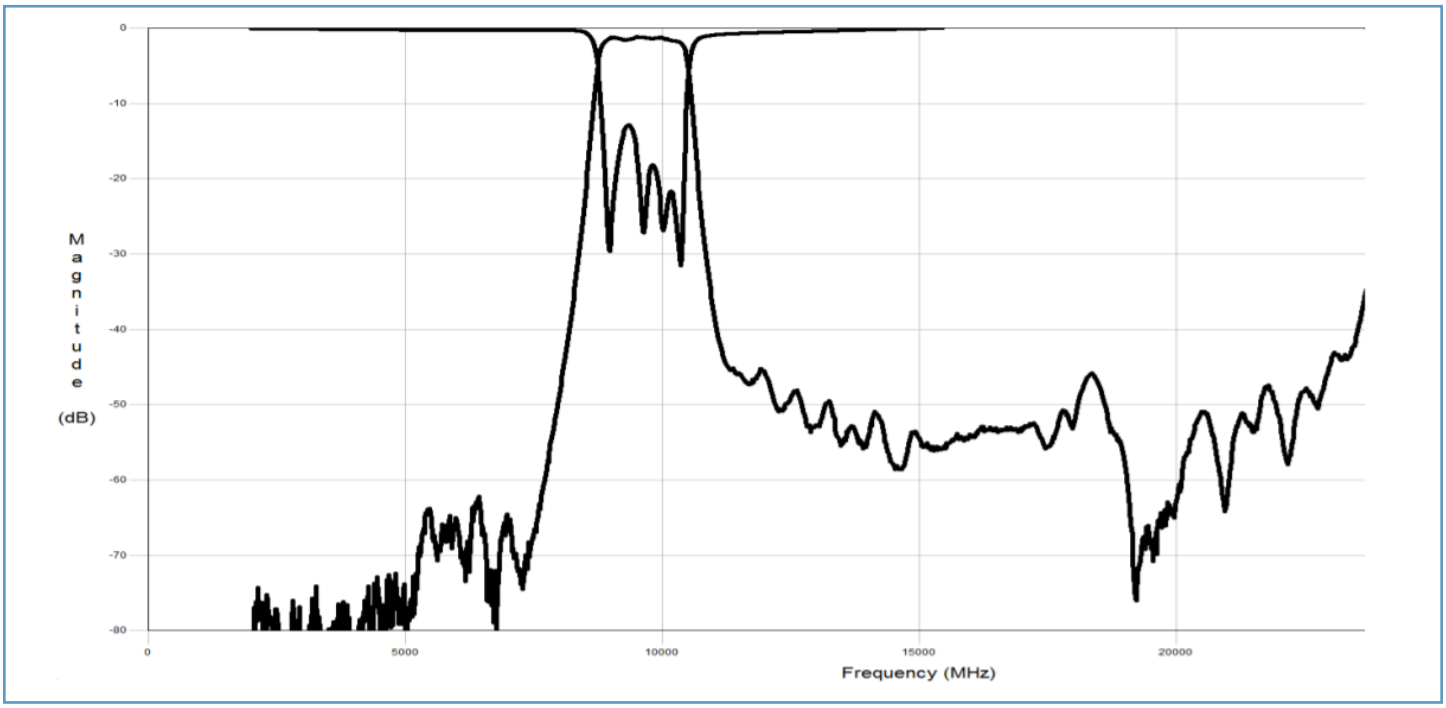

Figure 6 The measured performance for this bandpass filter shown in the filter's datasheet.

Since we can see that there are several spikes and dips around 9.5 to 10 GHz, we can run a test to get a closer look at how our model performs for insertion loss and return loss in this narrow frequency range (results show in Figure 7).

Code:

```
ntwkPass = ntwk['9-10.1GHz']
ntwkPass.plot_s_db()
```

Results:

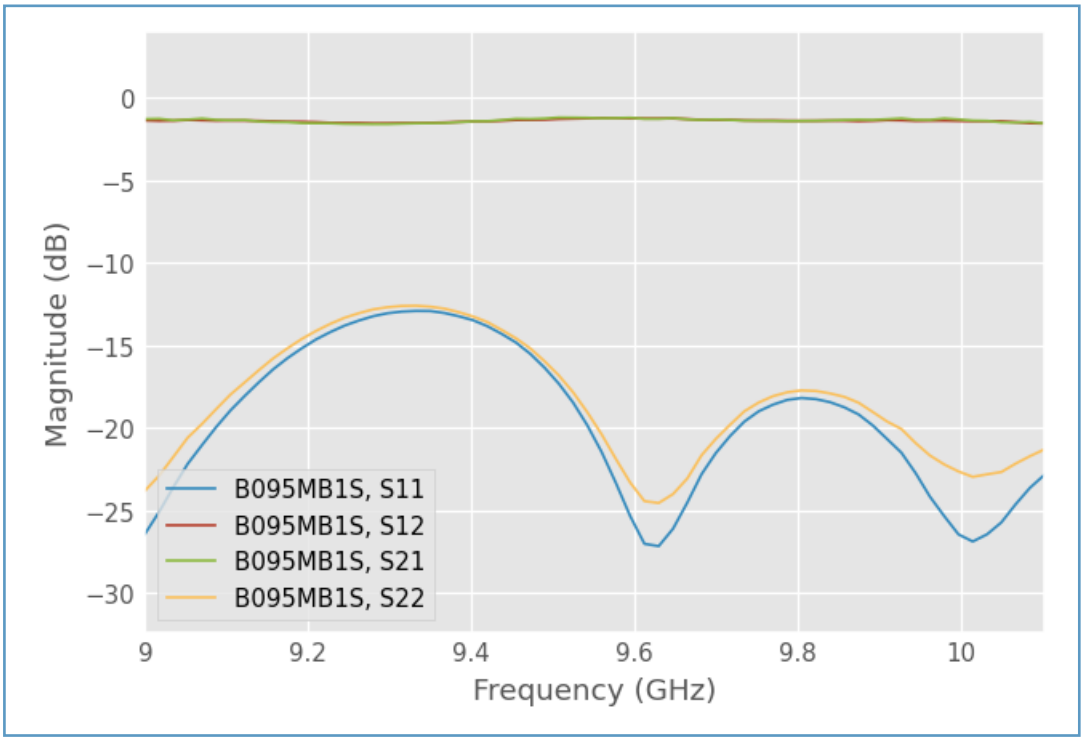

Figure 7 This plot shows the results of this analysis performed on an extremely narrow frequency range.

Performing More Complex Analysis in scikit-rf: A Cascading Filter Example

Now, let's get a little more complex with an analysis and use scikit-rf to look at what would happen if we used two bandpass filters in series, which is known as cascading, to improve rejection. As mentioned above, the `Network` class function in scikit-rf comes with convenient methods to manipulate the model in different ways. Above, we used the `plot_s_db method`. In this next series of experiments we will use the `cascade()` method, which can be conveniently written with the ** operator to look at the effects of placing two filters in series. Here is an example of doing this for the S21 parameter (plotted results shown in Figure 8).

Code:

```
ntwk_cascade = ntwk ** ntwk
ntwk_cascade.plot_s_db(m=1, n=0)
```

Results:

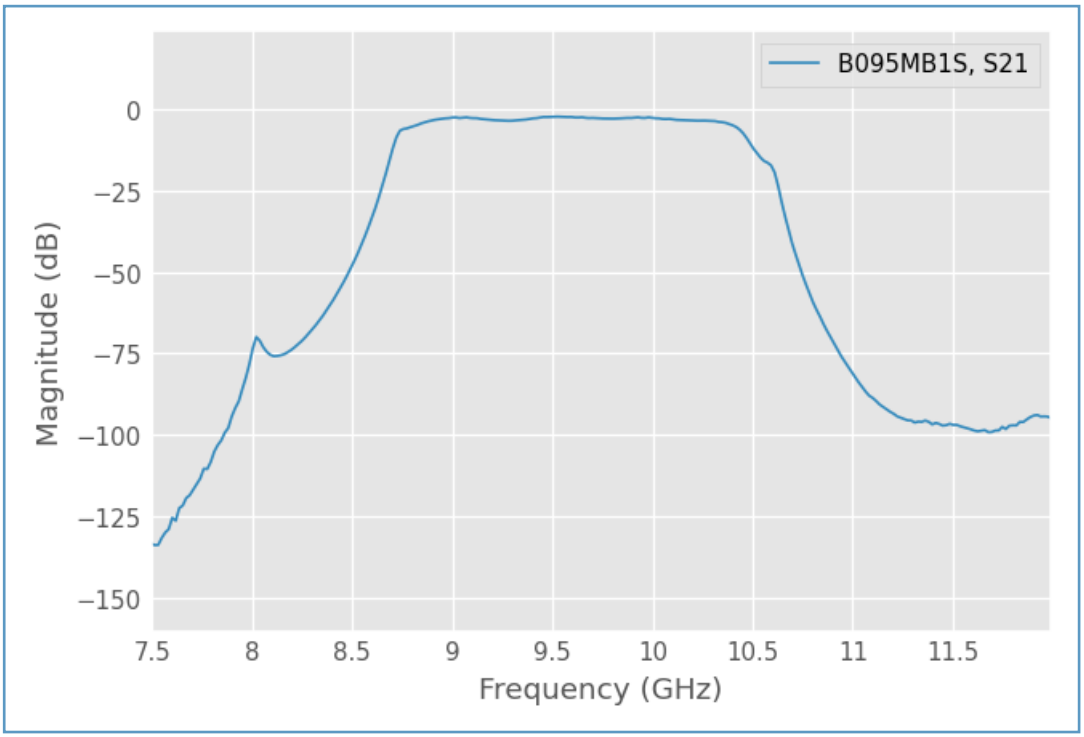

Figure 8 A plot showing the results of using the two bandpass filters in sequence.

We can then ask scikit-rf to show us the S21 values at 9.5GHz and 8.5GHz for comparison with the single filter. The single filter and the cascaded filter's responses are plotted below in Figure 9.

Code:

```
for x in ntwk_cascade['9.5GHz'].s:
    print(' S21: {result:.2f}db'.format(result=rf.mathFunctions.complex_2_db(x[1,
0])))
```

Results:

```
S21: -2.23db
```

Code:

```
for x in ntwk['8.5GHz'].s:
    print(' S21: {result:.2f}db'.format(result=rf.mathFunctions.complex_2_db(x[1,
0])))
```

Results:

```
S21: -22.03db
```

Code:

```
for x in ntwk_cascade['8.5GHz'].s:
    print(' S21: {result:.2f}db'.format(result=rf.mathFunctions.complex_2_db(x[1,
0])))
```

Results:

```
S21: -48.68db
```

Code:

```
ntwk.plot_s_db(m=1, n=0)
ntwk_cascade.plot_s_db(m=1, n=0, label='Scikit-RF cascade')
```

Results:

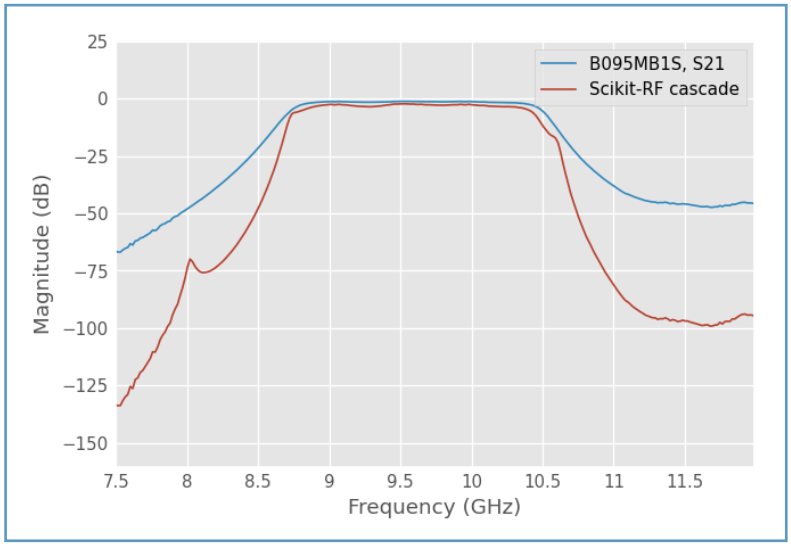

Figure 9 Comparing an analysis of a single filter and a separate analysis of cascaded filters.

In this quick analysis, you can see that the rejection at 8.5GHz drops from 22dB to 48dB, and insertion loss increases from 1.2dB to 2.2dB.

Checking the Results of Our Open-Source Tool Experiments Using Genesys

To show how a free open-source tool such as scikit-rf compares with a sophisticated software program when performing these more complex analyses, we ran this same analysis on two filters cascaded using Genesys. Figure 10 shows the results from both tools on the same plot.

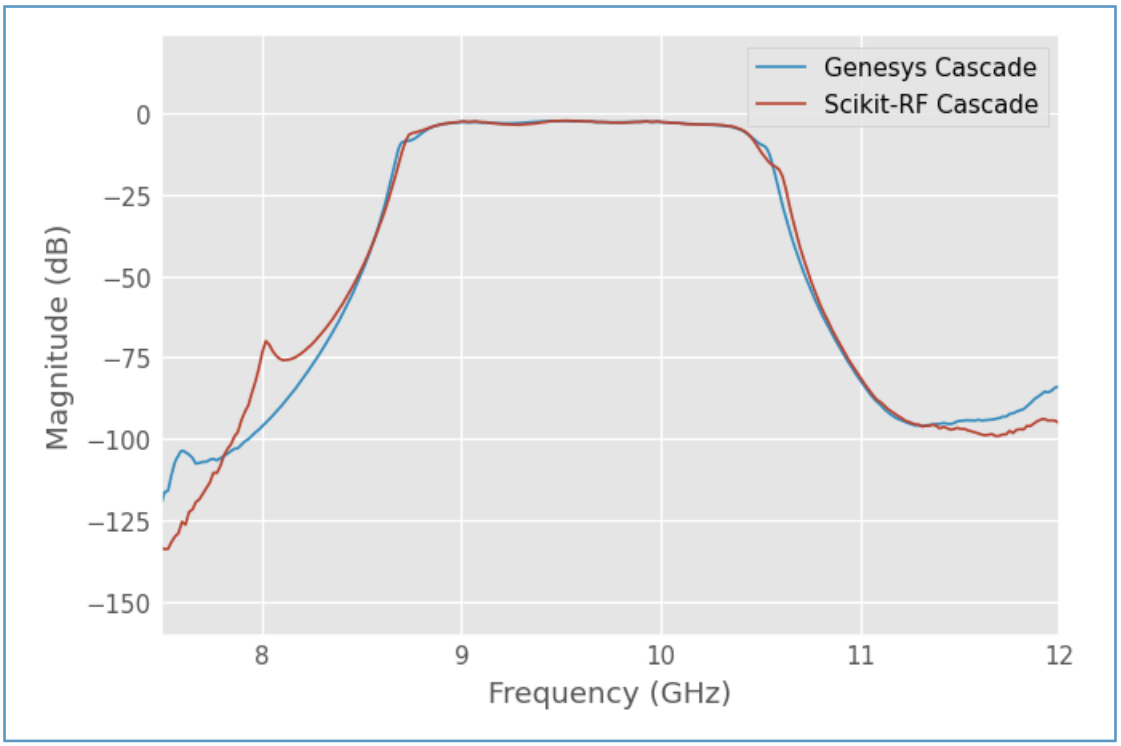

Figure 10 A side-by-side comparison of the same cascade filter analysis done in scikit-rf and Genesys.

By overlaying the Genesys results for this cascade analysis onto the plot of our scikit-rf-based analysis, we can see that overall, the free scikit-rf tool provides a pretty good approximation of the filter's performance. For example, the results between 9 and 10 GHz are nearly identical. However, there are differences, such as the location of an artifact caused by the interaction of the two filters. In Genesys, this local peak happens around 7.5GHz, whereas in the scikit-rf model we see it occur at 8GHz.

Open-Source Tools: An Excellent Option for Experimenting with Filter Performance

If you want to understand what S-parameters are telling you about a filter's performance across frequencies, or even perform some more complex analyses, you may not need access to a highly sophisticated RF modeling tool. Instead, as shown through the examples in this paper, open-source tools such as scikit-rf can be extremely powerful for providing a fairly quick, and free, method for analyzing data and sharing results with your team – and we have barely scratched the surface of what one can do with this tool! Furthermore, if you are also using a collaboration-friendly tool like Jupyter, along with various Python programming add-ons, performing plot intensive work like we have done in these examples, and sharing it with your team, becomes even easier as well.

However, as shown in Figure 10, which highlights the results from our final example where we compare the RF performance for a single filter and a cascaded filter, the more complex your parameters are, the more likely it is that your results could have some variations. And while not an issue we experienced with the examples used in this paper, it is also important to note that through other experiments we have conducted, we saw that when working with cascading data sets with differently spaced frequency data, scikit-rf does try its best to interpolate that data, but there may be some additional issues to watch out for.

So, if you have some more complex network analyses to perform and do not have access to a tool such as Genesys, or do not have the training to effectively use one of these tools, just give us a call. We love to experiment with open-source tools such as scikit-rf and we would be happy to share what we have learned with you.

Interested in learning more about what we are up to at Knowles Precision Devices? Subscribe to our blog for the latest updates and insights from our team.