RF Toolbox™

User's Guide

MATLAB®

R2023a

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

RF Toolbox™ User's Guide

Revision History

June 2004	Online only	New for Version 1.0 (Release 14)
August 2004	Online only	Revised for Version 1.0.1 (Release 14+)
March 2005	Online only	Revised for Version 1.1 (Release 14SP2)
September 2005	Online only	Revised for Version 1.2 (Release 14SP3)
March 2006	Online only	Revised for Version 1.3 (Release 2006a)
September 2006	Online only	Revised for Version 2.0 (Release 2006b)
March 2007	Online only	Revised for Version 2.1 (Release 2007a)
September 2007	Online only	Revised for Version 2.2 (Release 2007b)
March 2008	Online only	Revised for Version 2.3 (Release 2008a)
October 2008	Online only	Revised for Version 2.4 (Release 2008b)
March 2009	Online only	Revised for Version 2.5 (Release 2009a)
September 2009	Online only	Revised for Version 2.6 (Release 2009b)
March 2010	Online only	Revised for Version 2.7 (Release 2010a)
September 2010	Online only	Revised for Version 2.8 (Release 2010b)
April 2011	Online only	Revised for Version 2.8.1 (Release 2011a)
September 2011	Online only	Revised for Version 2.9 (Release 2011b)
March 2012	Online only	Revised for Version 2.10 (Release 2012a)
September 2012	Online only	Revised for Version 2.11 (Release 2012b)
March 2013	Online only	Revised for Version 2.12 (Release 2013a)
September 2013	Online only	Revised for Version 2.13 (Release 2013b)
March 2014	Online only	Revised for Version 2.14 (Release 2014a)
October 2014	Online only	Revised for Version 2.15 (Release 2014b)
March 2015	Online only	Revised for Version 2.16 (Release 2015a)
September 2015	Online only	Revised for Version 2.17 (Release 2015b)
March 2016	Online only	Revised for Version 3.0 (Release 2016a)
September 2016	Online only	Revised for Version 3.1 (Release 2016b)
March 2017	Online only	Revised for Version 3.2 (Release 2017a)
September 2017	Online only	Revised for Version 3.3 (Release 2017b)
March 2018	Online only	Revised for Version 3.4 (Release 2018a)
September 2018	Online only	Revised for Version 3.5 (Release 2018b)
March 2019	Online only	Revised for Version 3.6 (Release 2019a)
September 2019	Online only	Revised for Version 3.7 (Release 2019b)
March 2020	Online only	Revised for Version 3.8 (Release 2020a)
September 2020	Online only	Revised for Version 4.0 (Release 2020b)
March 2021	Online only	Revised for Version 4.1 (Release 2021a)
September 2021	Online only	Revised for Version 4.2 (Release 2021b)
March 2022	Online only	Revised for Version 4.3 (Release 2022a)
September 2022	Online only	Revised for Version 4.4 (Release 2022b)
March 2023	Online only	Revised for Version 4.5 (Release 2023a)

Contents

1 RF Objects

2 Model an RF Component

3 Export Verilog-A Models

4 AMP File Format

5 How Tos, Definitions, Algorithms

6 RF Toolbox Examples

RF Objects

RF Data Objects

In this section...

Overview

RF Toolbox software uses data (`rfdata`) objects to store:

- Component data created from files or from information that you specify in the MATLAB® workspace.
- Analyzed data from a frequency-domain simulation of a circuit object.

You can perform basic tasks, such as plotting and network parameter conversion, on the data stored in these objects. However, data objects are primarily used to store data for use by other RF objects.

Types of Data

The toolbox uses RF data objects to store one or more of the following types of data:

- Network parameters
- Spot noise
- Noise figure
- Third-order intercept point (IP3)
- Power out versus power in

Available Data Objects

The following table lists the available `rfdata` object constructors and describes the data the corresponding objects represent. For more information on a particular object, follow the link in the table to the reference page for that object.

Constructor	Description
`rfdata.data`	Data object containing network parameter data
`rfdata.ip3`	Data object containing IP3 information
`rfdata.mixerspur`	Data object containing mixer spur information from an intermodulation table
`rfdata.network`	Data object containing network parameter information
`rfdata.nf`	Data object containing noise figure information
`rfdata.noise`	Data object containing noise information
`rfdata.power`	Data object containing power and phase information

Data Object Methods

The following table lists the methods of the data objects, the types of objects on which each can act, and the purpose of each method.

Method	Types of Objects	Purpose
`extract`	`rfdata.data`, `rfdata.network`	Extract specified network parameters from a circuit or data object and return the result in an array
`read`	`rfdata.data`	Read RF data parameters from a file to a new or existing data object.
`write`	`rfdata.data`	Write RF data from a data object to a file.

See Also

More About

- “RF Analysis”
- “RF Circuit Objects” on page 1-4
- “RF Model Objects” on page 1-8
- “RF Network Parameter Objects” on page 1-10

RF Circuit Objects

In this section...

Overview of RF Circuit Objects

RF Toolbox software uses circuit (`rfckt`) objects to represent the following components:

- Circuit components such as amplifiers, transmission lines, and ladder filters
- RLC network components
- Networks of RF components

The toolbox represents each type of component and network with a different object. You use these objects to analyze components and networks in the frequency domain.

Components Versus Networks

You define component behavior using network parameters and physical properties.

To specify an individual RF component:

1. Construct a circuit object to represent the component.
2. Specify or import component data.

You define network behavior by specifying the components that make up the network. These components can be either individual components (such as amplifiers and transmission lines) or other networks.

To specify an RF network:

1. Build circuit objects to represent the network components.
2. Construct a circuit object to represent the network.

   Note This object defines how to connect the network components. However, the network is empty until you specify the components that it contains.

3. Specify, as the `Ckts` property of the object that represents the network, a list of components that make up the network.

These procedures are illustrated by example in "Model Cascaded Network".

Available Components and Networks

To create circuit objects that represent components, you use constructors whose names describe the components. To create circuit objects that represent networks, you use constructors whose names describe how the components are connected together.

The following table lists the available `rfckt` object constructors and describes the components or networks the corresponding objects represent. For more information on a particular object, follow the link in the table to the reference page for that object.

Constructor	Description
`rfckt.amplifier`	Amplifier, described by an `rfdata` object
`rfckt.cascade`	Cascaded network, described by the list of components and networks that comprise it
`rfckt.coaxial`	Coaxial transmission line, described by dimensions and electrical characteristics
`rfckt.cpw`	Coplanar waveguide transmission line, described by dimensions and electrical characteristics
`rfckt.datafile`	General circuit, described by a data file
`rfckt.delay`	Delay line, described by loss and delay
`rfckt.hybrid`	Hybrid connected network, described by the list of components and networks that comprise it
`rfckt.hybridg`	Inverse hybrid connected network, described by the list of components and networks that comprise it
`rfckt.lcbandpasspi`	LC bandpass pi network, described by LC values
`rfckt.lcbandpasstee`	LC bandpass tee network, described by LC values
`rfckt.lcbandstoppi`	LC bandstop pi network, described by LC values
`rfckt.lcbandstoptee`	LC bandstop tee network, described by LC values
`rfckt.lchighpasspi`	LC highpass pi network, described by LC values
`rfckt.lchighpasstee`	LC highpass tee network, described by LC values
`rfckt.lclowpasspi`	LC lowpass pi network, described by LC values
`rfckt.lclowpasstee`	LC lowpass tee network, described by LC values
`rfckt.microstrip`	Microstrip transmission line, described by dimensions and electrical characteristics
`rfckt.mixer`	Mixer, described by an `rfdata` object
`rfckt.parallel`	Parallel connected network, described by the list of components and networks that comprise it
`rfckt.parallelplate`	Parallel-plate transmission line, described by dimensions and electrical characteristics
`rfckt.passive`	Passive component, described by network parameters
`rfckt.rlcgline`	RLCG transmission line, described by RLCG values
`rfckt.series`	Series connected network, described by the list of components and networks that comprise it

Constructor	Description
rfckt.seriesrlc	Series RLC network, described by RLC values
rfckt.shuntrlc	Shunt RLC network, described by RLC values
rfckt.twowire	Two-wire transmission line, described by dimensions and electrical characteristics
rfckt.txline	General transmission line, described by dimensions and electrical characteristics

Circuit Object Methods

The following table lists the methods of the circuit objects, the types of objects on which each can act, and the purpose of each method.

Method	Types of Objects	Purpose
analyze	All circuit objects	Analyze a circuit object in the frequency domain.
calculate	All circuit objects	Calculate specified parameters for a circuit object.
copy	All circuit objects	Copy a circuit or data object.
extract	All circuit objects	Extract specified network parameters from a circuit or data object, and return the result in an array.
getdata	All circuit objects	Get data object containing analyzed result of a specified circuit object.
getz0	rfckt.txline, rfckt.rlcgline, rfckt.twowire, rfckt.parallelplate, rfckt.coaxial, rfdata.microstrip, rfckt.cpw	Get characteristic impedance of a transmission line.
listformat	All circuit objects	List valid formats for a specified circuit object parameter.
listparam	All circuit objects	List valid parameters for a specified circuit object.
loglog	All circuit objects	Plot specified circuit object parameters using a log-log scale.
plot	All circuit objects	Plot the specified circuit object parameters on an X-Y plane.
plotyy	All circuit objects	Plot the specified object parameters with y-axes on both the left and right sides.
polar	All circuit objects	Plot the specified circuit object parameters on polar coordinates.

Method	Types of Objects	Purpose
`read`	`rfckt.datafile`, `rfckt.passive`, `rfckt.amplifier`, `rfckt.mixer`	Read RF data from a file to a new or existing circuit object.
`restore`	`rfckt.datafile`, `rfckt.passive`, `rfckt.amplifier`, `rfckt.mixer`	Restore data to original frequencies of `NetworkData` for plotting.
`semilogx`	All circuit objects	Plot the specified circuit object parameters using a log scale for the X-axis
`semilogy`	All circuit objects	Plot the specified circuit object parameters using a log scale for the Y-axis
`smith`	All circuit objects	Plot the specified circuit object parameters on a Smith chart.
`write`	All circuit objects	Write RF data from a circuit object to a file.
`smithplot`	All circuit objects	Plot measurement data on Smith chart

See Also

More About

- “RF Model Objects” on page 1-8
- “RF Analysis”
- “RF Data Objects” on page 1-2
- “RF Network Parameter Objects” on page 1-10

RF Model Objects

In this section...

Overview of RF Model Objects

RF Toolbox software uses model (`rfmodel`) objects to represent components and measured data mathematically for computing information such as time-domain response. Each type of model object uses a different mathematical model to represent the component.

RF model objects provide a high-level component representation for use after you perform detailed analysis using RF circuit objects. Use RF model objects to:

- Compute time-domain figures of merit for RF components
- Export Verilog-A models of RF components

Available Model Objects

The following table lists the available `rfmodel` object constructors and describes the model the corresponding objects use. For more information on a particular object, follow the link in the table to the reference page for that object.

Constructor	Description
`rfmodel.rational`	Rational function model

Model Object Methods

The following table lists the methods of the model objects, the types of objects on which each can act, and the purpose of each method.

Method	Types of Objects	Purpose
`freqresp`	All model objects	Compute the frequency response of a model object.
`timeresp`	All model objects	Compute the time response of a model object.
`write`	All model objects	Write data from a model object to a file.

See Also

More About

- “RF Analysis”
- “RF Data Objects” on page 1-2
- “RF Circuit Objects” on page 1-4

- “RF Network Parameter Objects” on page 1-10

RF Network Parameter Objects

In this section...

Overview of Network Parameter Objects

RF Toolbox software offers network parameter objects for:

- Importing network parameter data from a Touchstone file.
- Converting network parameters.
- Analyzing network parameter data.

Unlike circuit, model, and data objects, you can use existing RF Toolbox functions to operate directly on network parameter objects.

Available Network Parameter Objects

The following table lists the available network parameter objects and the functions that are used to construct them. For more information on a particular object, follow the link in the table to the reference page for that functions.

Network Parameter Object Type	Network Parameter Object Function
ABCD Parameter object	`abcdparameters`
Hybrid-g parameter object	`gparameters`
Hybrid parameter object	`hparameters`
S-parameter object	`sparameters`
Y-parameter object	`yparameters`
Z-parameter object	`zparameters`

Network Parameter Object Functions

The following table lists the functions that accept network parameter objects as inputs, the types of objects on which each can act, and the purpose of each function.

Function	Types of Objects	Purpose
`abcdparameters`	All network parameter objects	Convert any network parameters to ABCD parameters
`gparameters`	All network parameter objects	Convert any network parameters to hybrid-g parameters

Function	Types of Objects	Purpose
`hparameters`	All network parameter objects	Convert any network parameters to hybrid parameters
`sparameters`	All network parameter objects	Convert any network parameters to S-parameters
`yparameters`	All network parameter objects	Convert any network parameters to Y-parameters
`zparameters`	All network parameter objects	Convert any network parameters to Z-parameters
`cascadesparams`	S-parameter objects	Cascade S-parameters
`deembedsparams`	S-parameter objects	De-embed S-parameters
`gammain`	S-parameter objects	Calculate input reflection coefficient
`gammaml`	S-parameter objects	Calculate load reflection coefficient
`gammams`	S-parameter objects	Calculate source reflection coefficient
`gammaout`	S-parameter objects	Calculate output reflection coefficient
`ispassive`	S-parameter objects	Check S-parameter data passivity
`makepassive`	S-parameter objects	Make S-parameter data passive
`newref`	S-parameter objects	Change reference impedance
`powergain`	S-parameter objects	Calculate power gain
`rfplot`	S-parameter objects	Plot network parameters
`rfinterp1`	All network parameter objects	Interpolate network parameters at new frequencies
`rfparam`	All network parameter objects	Extract vector of network parameters
`s2tf`	S-parameter objects	Create transfer function from S-parameters
`stabilityk`	S-parameter objects	Calculate stability factor *K* of 2-port network
`stabilitymu`	S-parameter objects	Calculate stability factor *μ* of 2-port network
`smith`	All network parameter objects	Plot network parameter data on a Smith® Chart
`smithplot`	All network parameter objects	Plot measurement data on Smith chart

See Also

More About

- “RF Data Objects” on page 1-2
- “RF Circuit Objects” on page 1-4
- “RF Model Objects” on page 1-8
- “S-Parameter Notation”

Model an RF Component

Create RF Objects

In this section...

Construct a New Object

You can create any `rfdata`, `rfckt` or `rfmodel` object by calling the object constructor. You can create an `rfmodel` object by fitting a rational function to passive component data.

This section contains the following topics:

Call the Object Constructor

To create a new RF object with default property values, you call the object constructor without any arguments:

```
h = objecttype.objectname
```

where:

- `h` is the handle to the new object.
- *`objecttype`* is the object type (`rfdata`, `rfckt`, or `rfmodel`).
- *`objectname`* is the object name.

For example, to create an RLCG transmission line object, type:

```
h = rfckt.rlcgline
```

because the RLCG transmission line object is a circuit (`rfckt`) object named `rlcgline`.

The following code illustrates how to call the object constructor to create a microstrip transmission line object with default property values. The output `t1` is the handle of the newly created transmission line object.

```
t1 = rfckt.microstrip
```

RF Toolbox software lists the properties of the transmission line you created along with the associated default property values.

```
t1 =
           Name: 'Microstrip Transmission Line'
          nPort: 2
 AnalyzedResult: []
     LineLength: 0.0100
       StubMode: 'NotAStub'
    Termination: 'NotApplicable'
          Width: 6.0000e-004
         Height: 6.3500e-004
```

```
    Thickness: 5.0000e-006
     EpsilonR: 9.8000
    SigmaCond: Inf
  LossTangent: 0
```

The reference page describes these properties in detail, `rfckt.microstrip`.

Fit a Rational Function to Passive Component Data

You can create a model object by fitting a rational function to passive component data. You use this approach to create a model object that represents one of the following using a rational function:

- A circuit object that you created and analyzed.
- Data that you imported from a file.

For more information, see "Fit Model Object to Circuit Object Data" on page 2-25.

Copy an Existing Object

You can create a new object with the same property values as an existing object by using the `copy` function to copy the existing object. This function is useful if you have an object that is similar to one you want to create.

For example,

```
t2 = copy(t1);
```

creates a new object, `t2`, which has the same property values as the microstrip transmission line object, `t1`.

You can later change specific property values for this copy. For information on modifying object properties, see "Specify or Import Component Data" on page 2-4.

Note The syntax `t2 = t1` copies only the object handle and does not create a new object.

See Also

More About

- "Process File Data for Analysis" on page 2-14

Specify or Import Component Data

In this section...

RF Object Properties

Object properties specify the behavior of an object. You can specify object properties, or you can import them from a data file. To learn about properties that are specific to a particular type of circuit, data, or model object, see the reference page for that type of object.

Note The “RF Circuit Objects” on page 1-4, “RF Data Objects” on page 1-2,“RF Model Objects” on page 1-8 sections list the available types of objects and provide links to their reference pages.

Set Property Values

You can specify object property values when you construct an object or you can modify the property values of an existing object.

This section contains the following topics:

- “Specify Property Values at Construction” on page 2-4
- “Change Property Values of an Existing Object” on page 2-5

Specify Property Values at Construction

To set a property when you construct an object, include a comma-separated list of one or more property/value pairs in the argument list of the object construction command. A property/value pair consists of the arguments `'`*`PropertyName`*`'`,*`PropertyValue`*, where:

- *`PropertyName`* is a character vector specifying the property name. The name is case-insensitive. In addition, you need only type enough letters to uniquely identify the property name. For example, `'st'` is sufficient to refer to the `StubMode` property.

  Note You must use single quotation marks around the property name.

- *`PropertyValue`* is the value to assign to the property.

Include as many property names in the argument list as there are properties you want to set. Any property values that you do not set retain their default values. The circuit and data object reference pages list the valid values as well as the default value for each property.

This section contains examples of how to perform the following tasks:

- “Construct Components with Specified Properties” on page 2-5
- “Construct Networks of Specified Components” on page 2-5

Construct Components with Specified Properties

The following code creates a coaxial transmission line circuit object to represent a coaxial transmission line that is 0.05 meters long. Notice that the toolbox lists the available properties and their values.

```
t1 = rfckt.coaxial('LineLength',0.05)

t1 =

              Name: 'Coaxial Transmission Line'
             nPort: 2
    AnalyzedResult: []
        LineLength: 0.0500
          StubMode: 'NotAStub'
       Termination: 'NotApplicable'
       OuterRadius: 0.0026
       InnerRadius: 7.2500e-004
               MuR: 1
          EpsilonR: 2.3000
         LossTangent: 0
         SigmaCond: Inf
```

Construct Networks of Specified Components

To combine a set of RF components and existing networks to form an RF network, you create a network object with the `Ckts` property set to an array containing the handles of all the circuit objects in the network.

Suppose you have the following RF components:

```
t1 = rfckt.coaxial('LineLength',0.05);
a1 = rfckt.amplifier;
t2 = rfckt.coaxial('LineLength',0.1);
```

The following code creates a cascaded network of these components:

```
casc_network = rfckt.cascade('Ckts',{t1,a1,t2});
```

Change Property Values of an Existing Object

There are two ways to change the properties of an existing object:

- Using the `set` command
- Using structure-like assignments called dot notation

This section discusses the first option. For details on the second option, see “Reference Properties Directly Using Dot Notation” on page 2-11.

To modify the properties of an existing object, use the `set` command with one or more property/value pairs in the argument list. The general syntax of the command is

```
set(h,Property1',value1,'Property2',value2,...)
```

where

- `h` is the handle of the object.
- *'Property1',value1,'Property2',value2,...* is the list of property/value pairs.

For example, the following code creates a default coaxial transmission line object and changes it to a series stub with open termination.

```
t1 = rfckt.coaxial;
set(t1,'StubMode','series','Termination','open')
```

Note You can use the `set` command without specifying any property/value pairs to display a list of all properties you can set for a specific object. This example lists the properties you can set for the coaxial transmission line `t1`:

```
set(t1)

ans =
     LineLength: {}
       StubMode: {}
    Termination: {}
    OuterRadius: {}
    InnerRadius: {}
            MuR: {}
       EpsilonR: {}
      LossTangent: {}
      SigmaCond: {}
```

Import Property Values from Data Files

RF Toolbox software lets you import industry-standard data files, MathWorks® AMP files, and Agilent® P2D and S2D files into specific objects. This import capability lets you simulate the behavior of measured components.

You can import the following file formats:

- Industry-standard file formats — Touchstone SNP, YNP, ZNP, HNP, and GNP formats specify the network parameters and noise information for measured and simulated data.

  For more information on Touchstone files, see `https://ibis.org/connector/touchstone_spec11.pdf`.
- Agilent P2D file format — Specifies amplifier and mixer large-signal, power-dependent network parameters, noise data, and intermodulation tables for several operating conditions, such as temperature and bias values.

  The P2D file format lets you import system-level verification models of amplifiers and mixers.
- Agilent S2D file format — Specifies amplifier and mixer network parameters with gain compression, power-dependent S_{21} parameters, noise data, and intermodulation tables for several operating conditions.

  The S2D file format lets you import system-level verification models of amplifiers and mixers.
- MathWorks amplifier (AMP) file format — Specifies amplifier network parameters, output power versus input power, noise data and third-order intercept point.

For more information about `.amp` files, see “AMP File Data Sections” on page 4-2.

This section contains the following topics:

- “Objects Used to Import Data from a File” on page 2-7
- “How to Import Data Files” on page 2-7

Objects Used to Import Data from a File

One data object and three circuit objects accept data from a file. The following table lists the objects and any corresponding data format each supports.

Object	Description	Supported Format(s)
`rfdata.data`	Data object containing network parameter data, noise figure, and third-order intercept point	Touchstone, AMP, P2D, S2D
`rfckt.amplifier`	Amplifier	Touchstone, AMP, P2D, S2D
`rfckt.mixer`	Mixer	Touchstone, AMP, P2D, S2D
`rfckt.passive`	Generic passive component	Touchstone

How to Import Data Files

To import file data into a circuit or data object at construction, use a `read` command of the form:

```
obj = read(obj_type,'filename');
```

where

- `obj` is the handle of the circuit or data object.
- *`obj_type`* is the type of object in which to store the data, from the list of objects that accept file data shown in “Objects Used to Import Data from a File” on page 2-7.
- *`filename`* is the name of the file that contains the data.

For example,

```
ckt_obj=read(rfckt.amplifier, 'default.amp');
```

imports data from the file `default.amp` into an `rfckt.amplifier` object.

You can also import file data into an existing circuit object. The following commands are equivalent to the previous command:

```
ckt_obj=rfckt.amplifier;
read(ckt_obj, 'default.amp');
```

Note When you import component data from a `.p2d` or `.s2d` file, properties are defined for several operating conditions. You must select an operating condition to specify the object behavior, as described in “Specify Operating Conditions” on page 2-12.

Use Data Objects to Specify Circuit Properties

To specify a circuit object property using a data object, use the `set` command with the name of the data object as the value in the property/value pair.

For example, suppose you have the following `rfckt.amplifier` and `rfdata.nf` objects:

```
amp = rfckt.amplifier
f = 2.0e9;
nf = 13.3244;
nfdata = rfdata.nf('Freq',f,'Data',nf)
```

The following command uses the `rfdata.nf` data object to specify the `rfckt.amplifier` `NoiseData` property:

```
set(amp,'NoiseData',nfdata)
```

Set Circuit Object Properties Using Data Objects

In this example, you create a circuit object. Then, you create three data objects and use them to update the properties of the circuit object.

1. **Create an amplifier object.** This circuit object, `rfckt.amplifier`, has a network parameter, noise data, and nonlinear data properties. These properties control the frequency response of the amplifier, which is stored in the `AnalyzedResult` property. By default, all amplifier properties contain values from the `default.amp` file. The `NetworkData` property is an `rfdata.network` object that contains 50-ohm S-parameters. The `NoiseData` property is an `rfdata.noise` object that contains frequency-dependent spot noise data. The `NonlinearData` property is an `rfdata.power` object that contains output power and phase information.

   ```
   amp = rfckt.amplifier
   ```

   The toolbox displays the following output:

   ```
   amp =

                Name: 'Amplifier'
               nPort: 2
      AnalyzedResult: [1x1 rfdata.data]
            IntpType: 'Linear'
         NetworkData: [1x1 rfdata.network]
           NoiseData: [1x1 rfdata.noise]
       NonlinearData: [1x1 rfdata.power]
   ```

2. **Create a data object that stores network data.** Type the following set of commands at the MATLAB prompt to create an `rfdata.network` object that stores the 2-port Y-parameters at 2.08 GHz, 2.10 GHz, and 2.15 GHz. Later in this example, you use this data object to update the `NetworkData` property of the `rfckt.amplifier` object.

   ```
   f = [2.08 2.10 2.15]*1.0e9;
   y(:,:,1) = [-.0090-.0104i, .0013+.0018i; ...
                          -.2947+.2961i, .0252+.0075i];
   y(:,:,2) = [-.0086-.0047i, .0014+.0019i; ...
                          -.3047+.3083i, .0251+.0086i];
   y(:,:,3) = [-.0051+.0130i, .0017+.0020i; ...
                          -.3335+.3861i, .0282+.0110i];
   ```

```
netdata = rfdata.network('Type','Y_PARAMETERS',...
                         'Freq',f,'Data',y)
```

The toolbox displays the following output:

```
netdata =

    Name: 'Network parameters'
    Type: 'Y_PARAMETERS'
    Freq: [3x1 double]
    Data: [2x2x3 double]
      Z0: 50
```

3 **Create a data object that stores noise figure values.** Type the following set of commands at the MATLAB prompt to create a `rfdata.nf` object that contains noise figure values, in dB, at seven different frequencies. Later in this example, you use this data object to update the `NoiseData` property of the `rfckt.amplifier` object.

```
f = [1.93 2.06 2.08 2.10 2.15 2.30 2.40]*1.0e9;
nf=[12.4521 13.2466 13.6853 14.0612 13.4111 12.9499 13.3244];

nfdata = rfdata.nf('Freq',f,'Data',nf)
```

The toolbox displays the following output:

```
nfdata =

    Name: 'Noise figure'
    Freq: [7x1 double]
    Data: [7x1 double]
```

4 **Create a data object that stores output third-order intercept points.** Type the following command at the MATLAB prompt to create a `rfdata.ip3` object that contains an output third-order intercept point of 8.45 watts, at 2.1 GHz. Later in this example, you use this data object to update the `NonlinearData` property of the `rfckt.amplifier` object.

```
ip3data = rfdata.ip3('Type','OIP3','Freq',2.1e9,'Data',8.45)
```

The toolbox displays the following output:

```
ip3data =

    Name: '3rd order intercept'
    Type: 'OIP3'
    Freq: 2.1000e+009
    Data: 8.4500
```

5 **Update the properties of the amplifier object.** Type the following set of commands at the MATLAB prompt to update the `NetworkData`, `NoiseData`, and `NonlinearData` properties of the amplifier object with the data objects you created in the previous steps:

```
amp.NetworkData = netdata;
amp.NoiseData = nfdata;
amp.NonlinearData = ip3data;
```

Retrieve Property Values

You can retrieve one or more property values of an existing object using the `get` command.

This section contains the following topics:

- “Retrieve Specified Property Values” on page 2-10
- “Retrieve All Property Values” on page 2-10

Retrieve Specified Property Values

To retrieve specific property values for an object, use the `get` command with the following syntax:

```
PropertyValue = get(h,PropertyName)
```

where

- *`PropertyValue`* is the value assigned to the property.
- `h` is the handle of the object.
- *`PropertyName`* is a character vector specifying the property name.

For example, suppose you have the following coaxial transmission line:

```
h2 = rfckt.coaxial;
```

The following code retrieves the value of the inner radius and outer radius for the coaxial transmission line:

```
ir = get(h2,'InnerRadius')
or = get(h2,'OuterRadius')

ir =
  7.2500e-004

or =
    0.0026
```

Retrieve All Property Values

To display a list of properties associated with a specific object as well as their current values, use the `get` command without specifying a property name.

For example:

```
get(h2)
              Name: 'Coaxial Transmission Line'
             nPort: 2
    AnalyzedResult: []
        LineLength: 0.0100
          StubMode: 'NotAStub'
       Termination: 'NotApplicable'
       OuterRadius: 0.0026
       InnerRadius: 7.2500e-004
               MuR: 1
          EpsilonR: 2.3000
       LossTangent: 0
         SigmaCond: Inf
```

Note This list includes read-only properties that do not appear when you type `set(h2)`. For a coaxial transmission line object, the read-only properties are `Name`, `nPort`, and `AnalyzedResult`.

The `Name` and `nPort` properties are fixed by the toolbox. The `AnalyzedResult` property value is calculated and set by the toolbox when you analyze the component at specified frequencies.

Reference Properties Directly Using Dot Notation

An alternative way to query for or modify property values is by structure-like referencing. The field names for RF objects are the property names, so you can retrieve or modify property values with the structure-like syntax.

- *`PropertyValue = rfobj.PropertyName`* stores the value of the *`PropertyName`* property of the *`rfobj`* object in the *`PropertyValue`* variable. This command is equivalent to *`PropertyValue`* `= get(`*`rfobj`*`,'`*`PropertyName`*`')`.
- *`rfobj.PropertyName = PropertyValue`* sets the value of the *`PropertyName`* property to *`PropertyValue`* for the *`rfobj`* object. This command is equivalent to `set(`*`rfobj`*`,'`*`PropertyName`*`',`*`PropertyValue`*`)`.

For example, typing

```
ckt = rfckt.amplifier('IntpType','cubic');
ckt.IntpType
```

gives the value of the property `IntpType` for the circuit object `ckt`.

```
ans =
        Cubic
```

Similarly,

```
ckt.IntpType = 'linear';
```

resets the interpolation method to linear.

You do not need to type the entire field name or use uppercase characters. You only need to type the minimum number of characters sufficient to identify the property name uniquely. Thus entering the commands

```
ckt = rfckt.amplifier('IntpType','cubic');
ckt.in
```

also produces

```
ans =
        Cubic
```

See Also

"AMP File Data Sections" on page 4-2

Specify Operating Conditions

In this section...

Available Operating Conditions

Agilent P2D and S2D files contain simulation results at one or more operating conditions. Operating conditions define the independent parameter settings that are used when creating the file data. The specified conditions differ from file to file.

When you import component data from a `.p2d` or `.s2d` file, the object contains property values for several operating conditions. The available conditions depend on the data in the file. By default, RF Toolbox software defines the object behavior using the property values that correspond to the operating conditions that appear first in the file. To use other property values, you must select a different operating condition.

Set Operating Conditions

To set the operating conditions of a circuit or data object, use a `setop` command of the form:

```
setop(,'Condition1',value1,...,'ConditionN',valueN,...)
```

where

- is the handle of the circuit or data object.
- *Condition1*,*value1*,...,*ConditionN*,*valueN* are the condition/value pairs that specify the operating condition.

For example,

```
setop(myp2d, 'BiasL', 2, 'BiasU', 6.3)
```

specifies an operating condition of `BiasL = 2` and `BiasU = 6.3` for *myp2d*.

Display Available Operating Condition Values

To display a list of available operating condition values for a circuit or data object, use the `setop` method.

```
setop(obj)
```

displays the available values for all operating conditions of the object `obj`.

```
setop(obj,'Condition1')
```

displays the available values for *Condition1*.

See Also

More About

- “S-Parameter Notation”
- “AMP File Data Sections” on page 4-2
- “Determining Parameter Formats” on page 5-2

Process File Data for Analysis

In this section...

Convert Single-Ended S-Parameters to Mixed-Mode S-Parameters

After you import file data (as described in "Import Property Values from Data Files" on page 2-6), you can convert a matrix of single-ended S-parameter data to a matrix of mixed-mode S-parameters.

This section contains the following topics:

Functions for Converting S-Parameters

To convert between 4-port single-ended S-parameter data and 2-port differential-, common-, and cross-mode S-parameters, use one of these functions:

- `s2scc` — Convert 4-port, single-ended S-parameters to 2-port, common-mode S-parameters (S_{cc}).
- `s2scd` — Convert 4-port, single-ended S-parameters to 2-port, cross-mode S-parameters (S_{cd}).
- `s2sdc` — Convert 4-port, single-ended S-parameters to cross-mode S-parameters (S_{dc}).
- `s2sdd` — Convert 4-port, single-ended S-parameters to 2-port, differential-mode S-parameters (S_{dd}).

To perform the above conversions all at once, or to convert larger data sets, use one of these functions:

- `s2smm` — Convert 4N-port, single-ended S-parameters to 2N-port, mixed-mode S-parameters.
- `smm2s` — Convert 2N-port, mixed-mode S-parameters to 4N-port, single-ended S-parameters.

Conversion functions support a variety of port orderings. For more information on these functions, see the corresponding reference pages.

Convert S-Parameters

In this example, use the toolbox to import 4-port single-ended S-parameter data from a file, convert the data to 2-port differential S-parameter data, and create a new `rfckt` object to store the converted data for analysis.

At the MATLAB prompt:

1. Type this command to import data from the file `default.s4p`:

   ```
   SingleEnded4Port = read(rfdata.data,'default.s4p');
   ```

2. Type this command to convert 4-port single-ended S-parameters to 2-port mixed-mode S-parameters:

```
DifferentialSParams = s2sdd(SingleEnded4Port.S_Parameters);
```

Note The S-parameters that you specify as input to the `s2sdd` function are the ones the toolbox stores in the `S_Parameters` property of the `rfdata.data` object.

3 Type this command to create an `rfckt.passive` object that stores the 2-port differential S-parameters for simulation:

```
DifferentialCkt = rfckt.passive('NetworkData', ...
    rfdata.network('Data', DifferentialSParams, 'Freq', ...
    SingleEnded4PortData.Freq));
```

Extract M-Port S-Parameters from N-Port S-Parameters

After you import file data (as described in “Import Property Values from Data Files” on page 2-6), you can extract a set of data with a smaller number of ports by terminating one or more ports with a specified impedance.

This section contains the following topics:

Extract S-Parameters

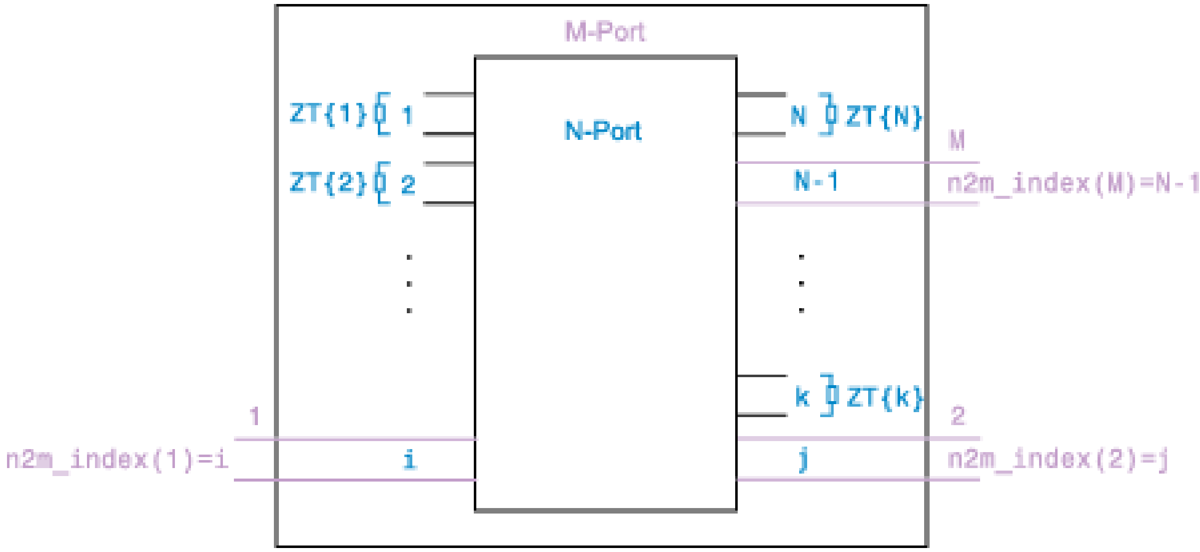
To extract M-port S-parameters from N-port S-parameters, use the `snp2smp` function with the following syntax:

```
s_params_mp = snp2smp(s_params_np, z0, n2m_index, zt)
```

where

- *`s_params_np`* is an array of *N*-port S-parameters with a reference impedance *`z0`*.
- `s_params_mp` is an array of *M*-port S-parameters.
- *`n2m_index`* is a vector of length *M* specifying how the ports of the *N*-port S-parameters map to the ports of the *M*-port S-parameters. *`n2m_index`*`(`*`i`*`)` is the index of the port from *`s_params_np`* that is converted to the *`i`*th port of `s_params_mp`.
- *`zt`* is the termination impedance of the ports.

The following figure illustrates how to specify the ports for the output data and the termination of the remaining ports.



For more details about the arguments to this function, see the `snp2smp` reference page.

Extract S-Parameters from Imported File Data

In this example, use the toolbox to import 16-port S-parameter data from a file, convert the data to 4-port S-parameter data by terminating the remaining ports, and create a new `rfckt` object to store the extracted data for analysis.

At the MATLAB prompt:

1. Type this command to import data from the file `default.s16p` into an `rfdata.data` object, `SingleEnded16PortData`:

   ```
   SingleEnded16PortData = read(rfdata.data,'default.s16p');
   ```

2. Type this command to convert 16-port S-parameters to 4-port S-parameters by using ports 1, 16, 2, and 15 as the first, second, third, and fourth ports, and terminating the remaining 12 ports with an impedance of 50 ohms:

   ```
   N2M_index = [1 16 2 15];
   FourPortSParams = snp2smp(SingleEnded16PortData.S_Parameters, ...
       SingleEnded16PortData.Z0, N2M_index, 50);
   ```

   Note The S-parameters that you specify as input to the `snp2smp` function are the ones the toolbox stores in the `S_Parameters` property of the `rfdata.data` object.

3. Type this command to create an `rfckt.passive` object that stores the 4-port S-parameters for simulation:

   ```
   FourPortChannel = rfckt.passive('NetworkData', ...
       rfdata.network('Data', FourPortSParams, 'Freq', ...
       SingleEnded16PortData.Freq));
   ```

Cascade N-Port S-Parameters

After you import file data (as described in “Import Property Values from Data Files” on page 2-6), you can cascade two or more networks of N-port S-parameters.

To cascade networks of N-port S-parameters, use the `cascadesparams` function with the following syntax:

```
s_params = cascadesparams(s1_params,s2_params,...,sn_params,nconn)
```

where

- `s_params` is an array of cascaded S-parameters.
- *`s1_params`*`,`*`s2_params`*`,...,`*`sn_params`* are arrays of input S-parameters.
- *`nconn`* is a positive scalar or a vector of size `n-1` specifying how many connections to make between the ports of the input S-parameters. `cascadesparams` connects the last port(s) of one network to the first port(s) of the next network.

For more details about the arguments to this function, see the `cascadesparams` reference page.

Import and Cascade N-Port S-Parameters

In this example, use the toolbox to import 16-port and 4-port S-parameter file data and cascade the two S-parameter networks by connecting the last three ports of the 16-port network to the first three ports of the 4-port network. Then, create a new `rfckt` object to store the resulting network for analysis.

At the MATLAB prompt:

1. Type these commands to import data from the files `default.s16p` and `default.s4p`, and create the 16- and 4-port networks of S-parameters:

   ```
   S_16Port = read(rfdata.data,'default.s16p');
   S_4Port = read(rfdata.data,'default.s4p');
   freq = [2e9 2.1e9];
   analyze(S_16Port, freq);
   analyze(S_4Port, freq);
   sparams_16p = S_16Port.S_Parameters;
   sparams_4p = S_4Port.S_Parameters;
   ```
2. Type this command to cascade 16-port S-parameters and 4-port S-parameters by connecting ports 14, 15, and 16 of the 16-port network to ports 1, 2, and 3 of the 4-port network:

   ```
   sparams_cascaded = cascadesparams(sparams_16p, sparams_4p,3)
   ```

   `cascadesparams` creates a 14-port network. Ports 1–13 are the first 13 ports of the 16-port network. Port 14 is the fourth port of the 4-port network.
3. Type this command to create an `rfckt.passive` object that stores the 14-port S-parameters for simulation:

   ```
   Ckt14 = rfckt.passive('NetworkData', ...
       rfdata.network('Data', sparams_cascaded, 'Freq', ...
       freq));
   ```

For more examples of how to use this function, see the `cascadesparams` reference page.

See Also

More About

- “S-Parameter Notation”
- “AMP File Data Sections” on page 4-2
- “Determining Parameter Formats” on page 5-2

Analyze and Plot RF Components

In this section...

Analyze Networks in Frequency Domain

RF Toolbox lets you analyze RF components and networks in the frequency domain. You use the `analyze` function to analyze a circuit object over a specified set of frequencies.

For example, to analyze a coaxial transmission line from 1 GHz to 2.9 GHz in increments of 10 MHz:

```
ckt = rfckt.coaxial;
f = [1.0e9:1e7:2.9e9];
analyze(ckt,f);
```

Note For all circuits objects except those that contain data from a file, you must perform a frequency-domain analysis with the `analyze` method before visualizing component and network data. For circuits that contain data from a file, the toolbox performs a frequency-domain analysis when you use the `read` method to import the data.

When you analyze a circuit object, the toolbox computes the circuit network parameters, noise figure values, and output third-order intercept point (OIP3) values at the specified frequencies and stores the result of the analysis in the object's `AnalyzedResult` property.

For more information, see the `analyze` function page.

Visualize Component and Network Data

The RF Toolbox lets you validate the behavior of circuit objects that represent RF components and networks by plotting the following data:

- Large- and small-signal S-parameters
- Noise figure
- Output third-order intercept point
- Power data
- Phase noise
- Voltage standing-wave ratio
- Power gain
- Group delay
- Reflection coefficients
- Stability data
- Transfer function

This table summarizes the available plots and charts, along with the functions you can use to create each one and a description of its contents.

Plot Type	Functions	Plot Contents

"Rectangular Plot" on page 2-21	`plot`
	`plotyy`
	`loglog`
	`semilogx`
	`semilogy`
		Parameters as a function of frequency or, where applicable, operating condition. The available parameters include:
		• S-parameters
		• Noise figure
		• Voltage standing-wave ratio (VSWR)
		• OIP3

"Budget Plot" on page 2-21	`plot`	Parameters as a function of frequency for each component in a cascade, where the curve for a given component represents the cumulative contribution of each RF component up to and including the parameter value of that component.
"Mixer Spur Plot" on page 2-22	`plot`	Mixer spur power as a function of frequency for an `rfckt.mixer` object or an `rfckt.cascade` object that contains a mixer.
"Polar Plots and Smith Charts" on page 2-23	`polar` `smithplot`	Polar plot: Magnitude and phase of S-parameters as a function of frequency. Smith plot: Real and imaginary parts of S-parameters as a function of frequency, used for analyzing the reflections caused by impedance mismatch.

For each plot you create, you choose a parameter to plot and, optionally, a format in which to plot that parameter. The plot format defines how the RF Toolbox displays the data on the plot. The available formats vary with the data you select to plot. The data you can plot depends on the type of plot you create.

Note You can use the `listparam` function to list the parameters of a specified circuit object that are available for plotting. You can use the `listformat` function to list the available formats for a specified circuit object parameter.

The following topics describe the available plots:

- "Rectangular Plot" on page 2-21
- "Budget Plot" on page 2-21
- "Mixer Spur Plot" on page 2-22
- "Polar Plots and Smith Charts" on page 2-23

Rectangular Plot

You can plot any parameters that are relevant to your object on a rectangular plot. You can plot parameters as a function of frequency for any object. When you import object data from a `.p2d` or `.s2d` file, you can also plot parameters as a function of any operating condition from the file that has numeric values, such as bias. In addition, when you import object data from a `.p2d` file, you can plot large-signal S-parameters as a function of input power or as a function of frequency. These parameters are denoted `LS11`, `LS12`, `LS21`, and `LS22`.

This table summarizes the methods that are available in the toolbox for creating rectangular plots and describes the uses of each one. For more information on a particular type of plot, follow the link in the table to the documentation for that method.

Method	Description
`plot`	Plot of one or more object parameters
`plotyy`	Plot of one or more object parameters with y-axes on both the left and right sides
`semilogx`	Plot of one or more object parameters using a log scale for the X-axis
`semilogy`	Plot of one or more object parameters using a log scale for the Y-axis
`loglog`	Plot of one or more object parameters using a log-log scale

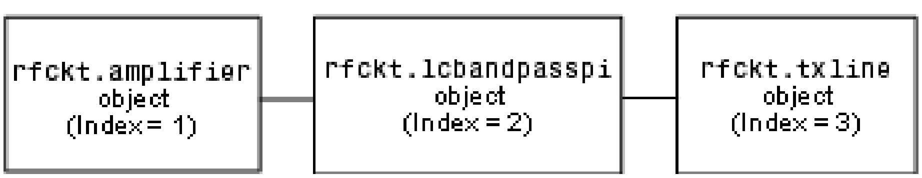
Budget Plot

You use the link budget or budget plot to understand the individual contribution of each component to a plotted parameter value in a cascaded network with multiple components. The budget plot shows one or more curves of parameter values as a function of frequency, ordered by the circuit index of the cascaded network.

Consider the following cascaded network:

```
casc = rfckt.cascade('Ckts',...
       {rfckt.amplifier,rfckt.lcbandpasspi,rfckt.txline})
```

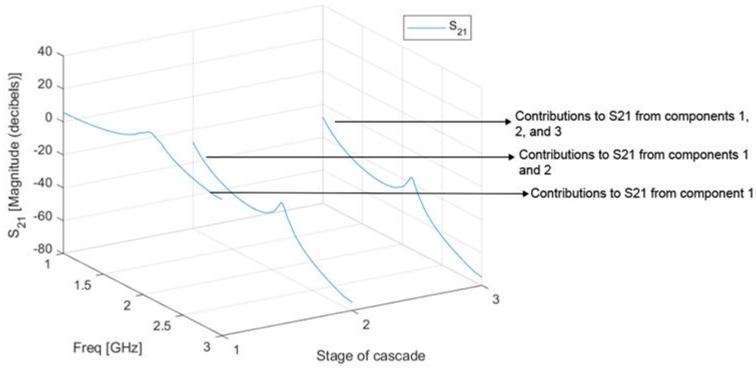
This figure shows how the circuit index is assigned to each component in the cascade, based on its sequential position in the network.



You create a 3-D budget plot for this cascade using the `plot` method with the second argument set to `'budget'`, as shown in the following command:

```
analyze(casc,linspace(1e9,3e9,100));
plot(casc,'budget','s21')
```

Note that you have to `analyze` your circuit before plotting the budget plot and by default the budget plot is a 2-D plot. If you specify the array of frequencies in the `analyze` function you can visualize the budget results in 3-D. A curve on the budget plot for each circuit index represents the contributions to the parameter value of the RF components up to that index. This figure shows the budget plot.



If you specify two or more parameters, the RF Toolbox puts the parameters in a single plot. You can only specify a single format for all the parameters.

Mixer Spur Plot

You use the mixer spur plot to understand how mixer nonlinearities affect output power at the desired mixer output frequency and at the intermodulation products that occur at the following frequencies:

$$f_{out} = N * f_{in} + M * f_{LO}$$

where

- f_{in} is the input frequency.
- f_{LO} is the local oscillator frequency.
- N and M are integers.

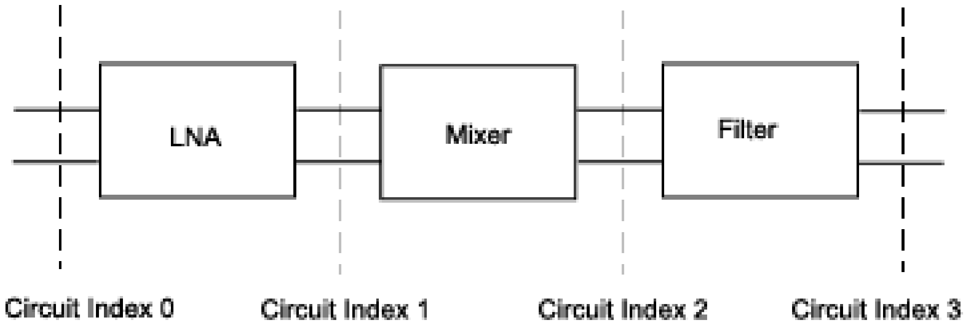
The RF toolbox calculates the output power from the mixer intermodulation table (IMT). These tables are described in detail in the “Visualize Mixer Spurs” on page 6-98 example.

The mixer spur plot shows power as a function of frequency for an `rfckt.mixer` object or an `rfckt.cascade` object that contains a mixer. By default, the plot is three-dimensional and shows a stem plot of power as a function of frequency, ordered by the circuit index of the object. You can create a two-dimensional stem plot of power as a function of frequency for a single circuit index by specifying the index in the mixer spur plot command.

Consider the following cascaded network:

```
FirstCkt = rfckt.amplifier('NetworkData', ...
    rfdata.network('Type', 'S', 'Freq', 2.1e9, ...
    'Data', [0,0;10,0]), 'NoiseData', 0, 'NonlinearData', inf);
SecondCkt = read(rfckt.mixer, 'samplespur1.s2d');
ThirdCkt = rfckt.lcbandpasstee('L', [97.21 3.66 97.21]*1e-9, ...
    'C', [1.63 43.25 1.63]*1.0e-12);
CascadedCkt = rfckt.cascade('Ckts', ...
    {FirstCkt, SecondCkt, ThirdCkt});
```

This shows how the circuit index is assigned to the components in the cascade, based on its sequential position in the network.

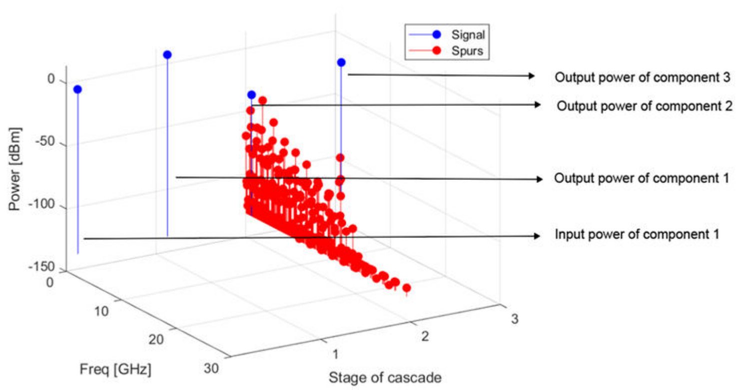


- Circuit index 0 corresponds to the cascade input.
- Circuit index 1 corresponds to the LNA output.
- Circuit index 2 corresponds to the mixer output.
- Circuit index 3 corresponds to the filter output.

You create a spur plot for this cascade using the `plot` method with the second argument set to `'mixerspur'`, as shown in the following command:

```
plot(CascadedCkt,'mixerspur')
```

Within the three dimensional plot, the stem plot for each circuit index represents the power at that circuit index. This figure shows the mixer spur plot.



For more information on mixer spur plots, see the `plot` reference page.

Polar Plots and Smith Charts

You can use the RF toolbox to generate Polar plots and Smith charts. If you specify two or more parameters, the RF toolbox puts the parameters in a single plot.

The following table describes the Polar plot and Smith charts options, as well as the available parameters.

Note `LS11`, `LS12`, `LS21`, and `LS22` are large-signal S-parameters. You can plot these parameters as a function of input power or as a function of frequency.

Plot Type	Method	Parameter
Polar plane	polar	S11, S12, S21, S22 LS11, LS12, LS21, LS22 (Objects with data from a P2D file only)
Z Smith chart	smithplot with type argument set to 'z'	S11, S22 LS11, LS22 (Objects with data from a P2D file only)
Y Smith chart	smithplot with type argument set to 'y'	S11, S22 LS11, LS22 (Objects with data from a P2D file only)
ZY Smith chart	smithplot with type argument set to 'zy'	S11, S22 LS11, LS22 (Objects with data from a P2D file only)

By default, the RF toolbox plots the parameter as a function of frequency. When you import block data from a `.p2d` or `.s2d` file, you can also plot parameters as a function of any operating condition from the file that has numeric values, such as bias.

Note The `circle` method lets you place circles on a Smith Chart to depict stability regions and display constant gain, noise figure, reflection and immittance circles. For more information about this function, see the `circle` reference page or “Designing Matching Networks for Low Noise Amplifiers” on page 6-122 example about designing matching networks.

Compute and Plot Time-Domain Specifications

The RF toolbox lets you compute and plot time-domain characteristics for RF components.

This section contains the following topics:

- “Compute Network Transfer Function” on page 2-24
- “Fit Model Object to Circuit Object Data” on page 2-25
- “Compute and Plot Time-Domain Response” on page 2-25

Compute Network Transfer Function

You use the `s2tf` function to convert 2-port S-parameters to a transfer function. The function returns a vector of transfer function values that represent the normalized voltage gain of a 2-port network.

The following code illustrates how to read a file data into a passive circuit object, extract the 2-port S-parameters from the object, and compute the transfer function of the data at the frequencies for which the data is specified. Here `z0` is the reference impedance of the S-parameters, `zs` is the source impedance, and `zl` is the load impedance. See the `s2tf` reference page for more information on how these impedances are used to define the gain.

```
PassiveCkt = rfckt.passive('File','passive.s2p')
z0=50; zs=50; zl=50;
```

```
[SParams, Freq] = extract(PassiveCkt, 'S Parameters', z0);
TransFunc = s2tf(SParams, z0, zs, zl);
```

Fit Model Object to Circuit Object Data

You use the `rationalfit` function to fit a rational function to the transfer function of a passive component. The `rationalfit` function returns an `rfmodel` object that represents the transfer function analytically.

The following code illustrates how to use the `rationalfit` function to create an `rfmodel.rational` object that contains a rational function model of the transfer function that you created in the previous example.

```
RationalFunc = rationalfit(Freq, TransFunc)
```

To find out how many poles the RF toolbox used to represent the data, look at the length of the `A` vector of the `RationalFunc` model object.

```
nPoles = length(RationalFunc.A)
```

Note The number of poles is important if you plan to use the RF model object to create a model for use in another simulator, because a large number of poles can increase simulation time. For information on how to represent a component accurately using a minimum number of poles, see "Represent Circuit Object with Model Object" on page 3-4.

Use the `freqresp` function to compute the frequency response of the fitted data. To validate the model fit, plot the transfer function of the original data and the frequency response of the fitted data.

```
Resp = freqresp(RationalFunc, Freq);
plot(Freq, 20*log10(abs(TransFunc)), 'r', ...
    Freq, 20*log10(abs(Resp)), 'b--');
ylabel('Magnitude of H(s) (decibels)');
xlabel('Frequency (Hz)');
legend('Original', 'Fitting result');
title(['Rational fitting with ', int2str(nPoles), ' poles']);
```

Compute and Plot Time-Domain Response

You use the `timeresp` function to compute the time-domain response of the transfer function that `RationalFunc` represents. This code illustrates how to create a random input signal, compute the time-domain response of `RationalFunc` to the input signal, and plot the results.

```
SampleTime=1e-11;
NumberOfSamples=4750;
OverSamplingFactor = 25;
InputTime = double((1:NumberOfSamples)')*SampleTime;
InputSignal = ...
     sign(randn(1, ceil(NumberOfSamples/OverSamplingFactor)));
InputSignal = repmat(InputSignal, [OverSamplingFactor, 1]);
InputSignal = InputSignal(:);

[tresp,t]=timeresp(RationalFunc,InputSignal,SampleTime);
plot(t*1e9,tresp);
title('Fitting Time-Domain Response', 'fonts', 12);
ylabel('Response to Random Input Signal');
xlabel('Time (ns)');
```

For more information about computing the time response of a model object, see the `timeresp` function.

See Also

More About

- “RF Analysis”
- “Export Component Data to File” on page 2-27

Export Component Data to File

In this section...

Available Export Formats

RF Toolbox software lets you export data from any `rfckt` object or from an `rfdata.data` object to industry-standard data files and MathWorks AMP files. This export capability lets you store data for use in other simulations.

Note The toolbox also lets you export data from an `rfmodel` object to a Verilog-A file. For information on how to do this, see “Export Verilog-A Model” on page 3-4.

You can export data to the following file formats:

- Industry-standard file formats — Touchstone SNP, YNP, ZNP, HNP, and GNP formats specify the network parameters and noise information for measured and simulated data.

  For more information about Touchstone files, see `https://ibis.org/connector/touchstone_spec11.pdf`.
- MathWorks amplifier (AMP) file format — Specifies amplifier network parameters, output power versus input power, noise data and third-order intercept point.

  For more information about `.amp` files, see “AMP File Data Sections” on page 4-2.

How to Export Object Data

To export data from a circuit or data object, use a `write` command of the form

```
status = write(obj,'filename');
```

where

- `status` is a return value that indicates whether the write operation was successful.
- *`obj`* is the handle of the circuit or `rfdata.data` object.
- *`filename`* is the name of the file that contains the data.

For example,

```
status = write(rfckt.amplifier, 'myamp.amp');
```

exports data from an `rfckt.amplifier` object to the file `myamp.amp`.

Export Object Data

In this example, use the toolbox to create a vector of S-parameter data, store it in an `rfdata.data` object, and export it to a Touchstone file.

At the MATLAB prompt:

1 Type the following to create a vector, `s_vec`, of S-parameter values at three frequency values:

```
s_vec(:,:,1) = ...
    [-0.724725-0.481324i, -0.685727+1.782660i; ...
      0.000000+0.000000i, -0.074122-0.321568i];
s_vec(:,:,2) = ...
    [-0.731774-0.471453i, -0.655990+1.798041i; ...
      0.001399+0.000463i, -0.076091-0.319025i];
s_vec(:,:,3) = ...
    [-0.738760-0.461585i, -0.626185+1.813092i; ...
      0.002733+0.000887i, -0.077999-0.316488i];
```

2 Type the following to create an `rfdata.data` object called `txdata` with the default property values:

```
txdata = rfdata.data;
```

3 Type the following to set the S-parameter values of `txdata` to the values you specified in `s_vec`:

```
txdata.S_Parameters = s_vec;
```

4 Type the following to set the frequency values of `txdata` to `[1e9 2e9 3e9]`:

```
txdata.Freq=1e9*[1 2 3];
```

5 Type the following to export the data in `txdata` to a Touchstone file called `test.s2p`:

```
write(txdata,'test')
```

See Also

More About

- “Export Verilog-A Model” on page 3-4

Basic Operations with RF Objects

This example shows how to read, analyze, and de-embed RF data from a Touchstone data file.

Read and Analyze RF Data from Touchstone Data File

In this example, you create an `sparameters` object by reading the S-Parameters of a 2-port passive network stored in the Touchstone format data file, `passive.s2p`.

Read S-Parameter data from a data file. Use RF Toolbox™ `sparameters` command to read the Touchstone data file, passive.s2p. This file contains 50-ohm S-Parameters at frequencies ranging from 315 kHz to 6 GHz. This operation creates an `sparameters` object, S_50, and stores data from the file in the object's properties.

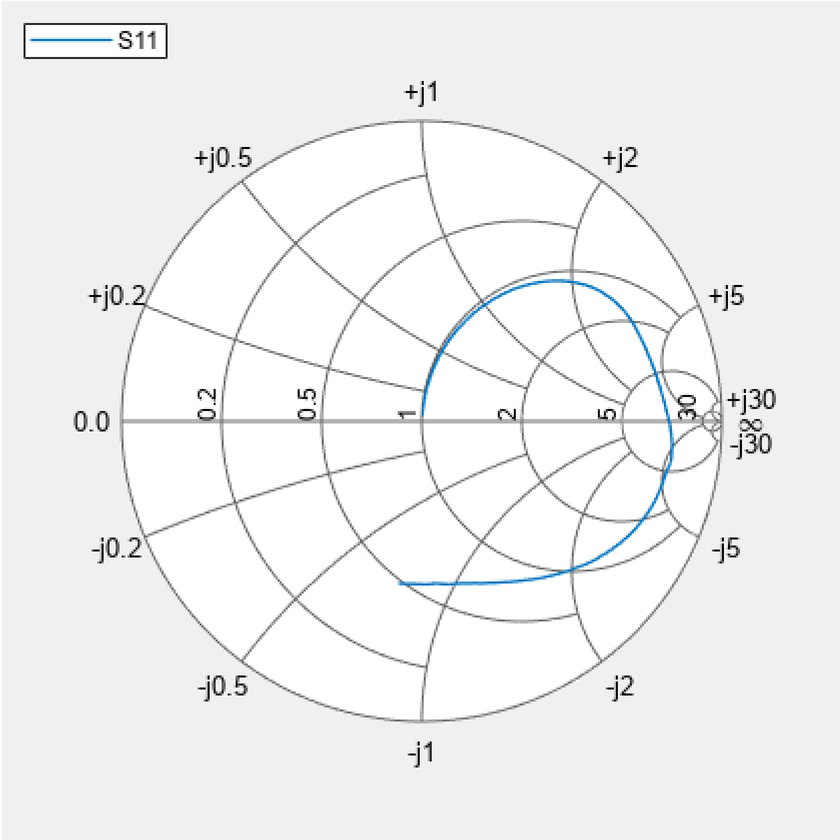
```
S_50 = sparameters('passive.s2p');
```

Use `sparameters` to convert the 50-ohm S-Parameters in the `sparameters` object, to 75-ohm S-Parameters and save them in the variable S_75. You can easily convert between parameters, for example, for Y-Parameters from the `sparameters` object use `yparameters` and save them in the variable Y.

```
Znew = 75;
S_75 = sparameters(S_50, Znew);
Y    = yparameters(S_75);
```

Plot the S11 parameters. Use the `smithplot` command to plot the 75-ohm S11 parameters on a Smith® Chart:

```
smithplot(S_75,1,1)
```



View the 75-ohm S-Parameters and Y-Parameters at 6 GHz. Type the following set of commands at the MATLAB® prompt to display the 2-port 75-ohm S-Parameter values and the 2-port Y-Parameter values at 6 GHz.

```
freq    = S_50.Frequencies;
f       = freq(end)
```

```
f = 6.0000e+09
```

```
s_6GHz  = S_75.Parameters(:,:,end)
```

```
s_6GHz = 2×2 complex

  -0.0764 - 0.5401i   0.6087 - 0.3018i
   0.6094 - 0.3020i  -0.1211 - 0.5223i
```

```
y_6GHz  = Y.Parameters(:,:,end)
```

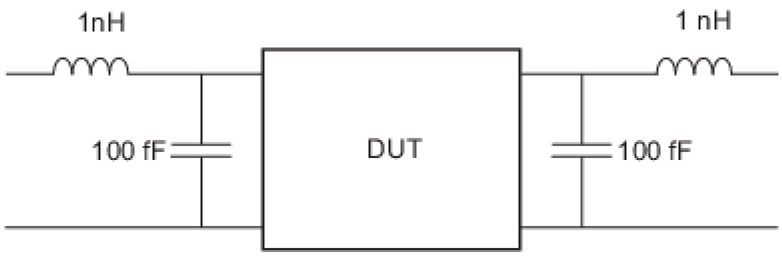
```
y_6GHz = 2×2 complex

   0.0210 + 0.0252i  -0.0215 - 0.0184i
  -0.0215 - 0.0185i   0.0224 + 0.0266i
```

For more information, see the `sparameters`, `yparameters`, `smithplot` reference pages.

De-Embed S-Parameters

The Touchstone data file samplebjt2.s2p contains S-Parameter data collected from a bipolar transistor in a test fixture. The input of the fixture has a bond wire connected to a bond pad. The output of the fixture has a bond pad connected to a bond wire.

The configuration of the bipolar transistor, which is the device under test (DUT), and the fixture is shown in the following figure.



In this example, you remove the effects of the fixture and extract the S-parameters of the DUT.

Create RF circuit objects

Create a `sparameters` object for the measured S-Parameters by reading the Touchstone data file samplebjt2.s2p. Then, create two more circuit objects, one each for the input pad and output pad.

```
measured_data = sparameters('samplebjt2.s2p');

L_left      = inductor(1e-9);
C_left      = capacitor(100e-15);
input_pad   = circuit('inputpad');
add(input_pad,[1 2],L_left)
add(input_pad,[2 0],C_left)
setports(input_pad,[1 0],[2 0])

L_right     = inductor(1e-9);
C_right     = capacitor(100e-15);
output_pad  = circuit('outputpad');
add(output_pad,[3 0],C_right)
add(output_pad,[3 4],L_right)
setports(output_pad,[3 0],[4 0])
```

Analyze the input pad and output pad circuit objects. Analyze the circuit objects at the frequencies at which the S-Parameters are measured.

```
freq                = measured_data.Frequencies;
input_pad_sparams   = sparameters(input_pad,freq);
output_pad_sparams  = sparameters(output_pad,freq);
```

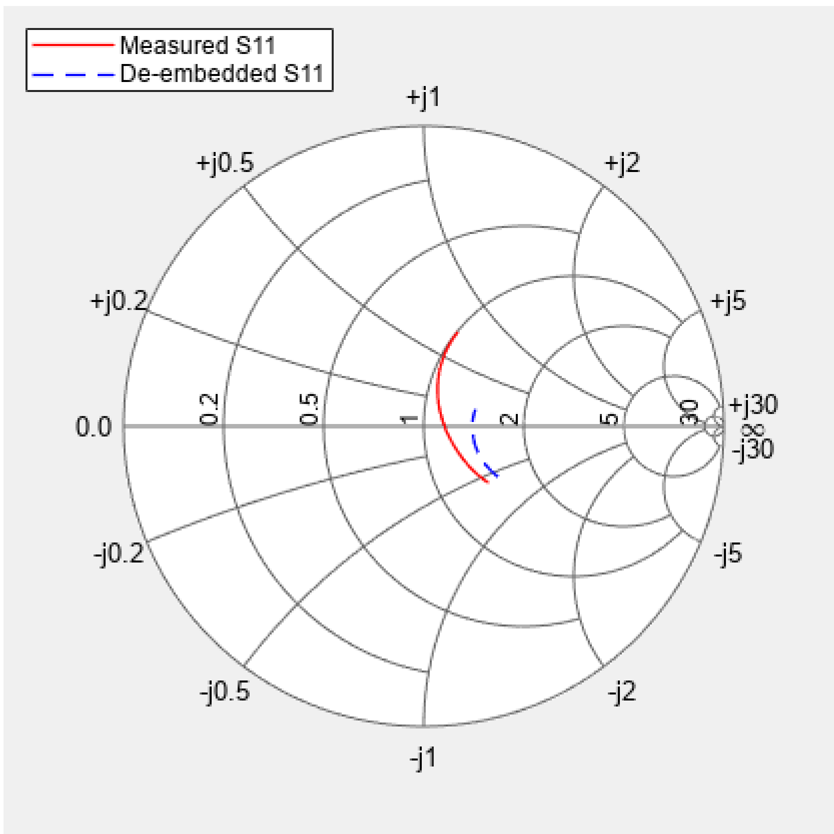
De-embed the S-parameters.

Extract the S-Parameters of the DUT from the measured S-Parameters by removing the effects of the input and output pads.

```
de_embedded_sparams = deembedsparams(measured_data,...
                      input_pad_sparams, output_pad_sparams);
```

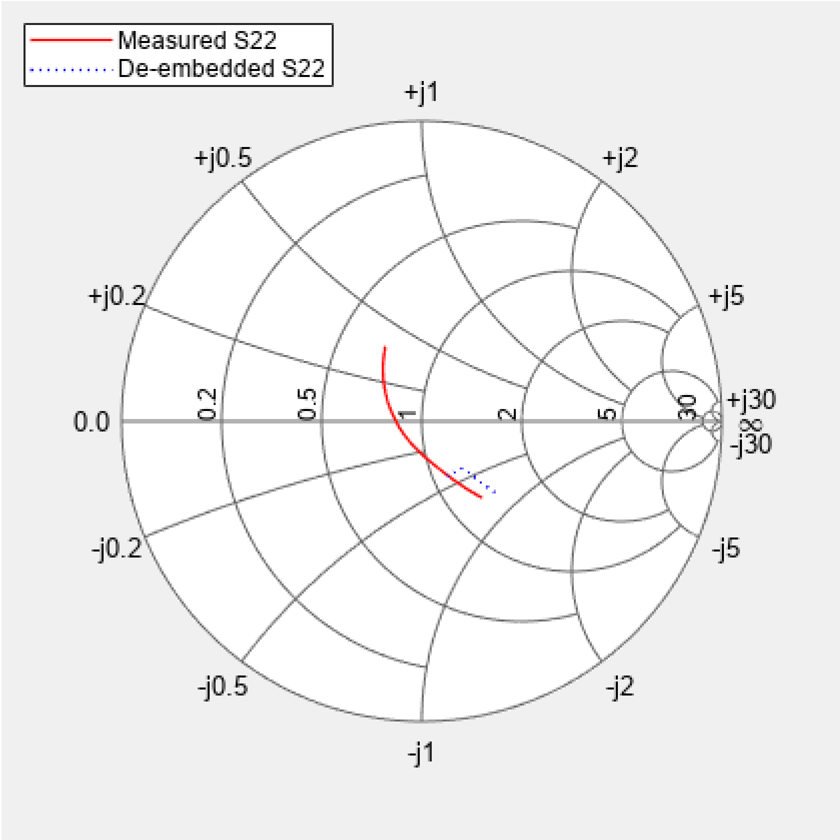
Plot the measured and de-embedded S11 parameters. Type the following set of commands at the MATLAB® prompt to plot both the measured and the de-embedded S11 parameters on a Z Smith® Chart:

```
figure;
smithplot(measured_data,1,1);
hold on
h                = smithplot(de_embedded_sparams,1,1);
h.LineStyle      = {'-';'--'};
h.ColorOrder     = [1 0 0;0 0 1];
h.LegendLabels   = {'Measured S11', 'De-embedded S11'};
```



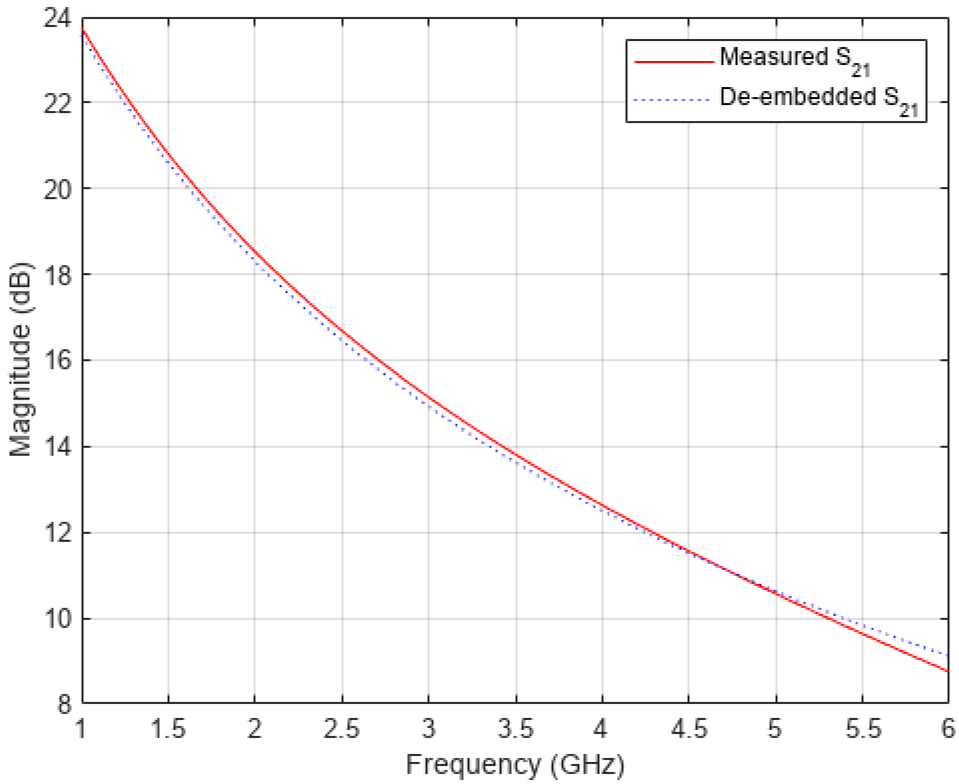
Plot the measured and de-embedded S22 parameters. Type the following set of commands at the MATLAB® prompt to plot the measured and the de-embedded S22 parameters on a Z Smith® Chart:

```
figure;
smithplot(measured_data,2,2);
hold on
h                = smithplot(de_embedded_sparams,2,2);
h.LineStyle      = {'-';':'};
h.ColorOrder     = [1 0 0;0 0 1];
h.LegendLabels   = {'Measured S22', 'De-embedded S22'};
```



Plot the measured and de-embedded S21 parameters. Type the following set of commands at the MATLAB® prompt to plot the measured and the de-embedded S21 parameters, in decibels, on an X-Y plane:

```
figure
rfplot(measured_data,2,1,'db','r');
hold on
rfplot(de_embedded_sparams,2,1,'db',':b');
legend('Measured S_{21}', 'De-embedded S_{21}');
```



See Also

More About

- “RF Data Objects” on page 1-2
- “RF Circuit Objects” on page 1-4
- “RF Model Objects” on page 1-8
- “RF Network Parameter Objects” on page 1-10

Export Verilog-A Models

Model RF Objects Using Verilog-A

In this section...

Overview

Verilog-A is a language for modeling the high-level behavior of analog components and networks. Verilog-A describes components mathematically, for fast and accurate simulation.

RF Toolbox software lets you export a Verilog-A description of your circuit. You can create a Verilog-A model of any passive RF component or network and use it as a behavioral model for transient analysis in a third-party circuit simulator. This capability is useful in signal integrity engineering. For example, you can import the measured four-port S-parameters of a backplane into the toolbox, export a Verilog-A model of the backplane to a circuit simulator, and use the model to determine the performance of your driver and receiver circuitry when they are communicating across the backplane.

Behavioral Modeling Using Verilog-A

The Verilog-A language is a high-level language that uses modules to describe the structure and behavior of analog systems and their components. A *module* is a programming building block that forms an executable specification of the system.

Verilog-A uses modules to capture high-level analog behavior of components and systems. Modules describe circuit behavior in terms of

- Input and output nets characterized by predefined Verilog-A disciplines that describe the attributes of the nets.
- Equations and module parameters that define the relationship between the input and output nets mathematically.

When you create a Verilog-A model of your circuit, the toolbox writes a Verilog-A module that specifies circuit's input and output nets and the mathematical equations that describe how the circuit operates on the input to produce the output.

Supported Verilog-A Models

RF Toolbox software lets you export a Verilog-A model of an `rfmodel` object. The toolbox provides one `rfmodel` object, `rfmodel.rational`, that you can use to represent any RF component or network for export to Verilog-A.

The `rfmodel.rational` object represents components as rational functions in pole-residue form, as described in the `rfmodel.rational` reference page. This representation can include complex poles and residues, which occur in complex-conjugate pairs.

The toolbox implements each `rfmodel.rational` object as a series of Laplace Transform S-domain filters in Verilog-A using the numerator-denominator form of the Laplace transform filter:

$$H(s) = \frac{\sum_{k=0}^{M} n_k s^k}{\sum_{k=0}^{N} d_k s^k}$$

where

- M is the order of the numerator polynomial.
- M is the order of the denominator polynomial.
- n_k is the coefficient of the kth power of s in the numerator.
- d_k is the coefficient of the kth power of s in the denominator.

The number of poles in the rational function is related to the number of Laplace transform filters in the Verilog-A module. However, there is not a one-to-one correspondence between the two. The difference arises because the toolbox combines each pair of complex-conjugate poles and the corresponding residues in the rational function to form a Laplace transform numerator and denominator with real coefficients. the toolbox converts the real poles of the rational function directly to a Laplace transform filter in numerator-denominator form.

See Also

More About

- “Export Verilog-A Model” on page 3-4

Export Verilog-A Model

In this section...

Represent Circuit Object with Model Object

Before you can write a Verilog-A model of an RF circuit object, you need to create an `rfmodel.rational` object to represent the component.

There are two ways to create an RF model object:

- You can fit a rational function model to the component data using the `rationalfit` function.
- You can use the `rfmodel.rational` constructor to specify the pole-residue representation of the component directly.

This section discusses using a rational function model. For more information on using the constructor, see the `rfmodel.rational` reference page.

When you use the `rationalfit` function to create an `rfmodel.rational` object that represents an RF component, the arguments you specify affect how quickly the resulting Verilog-A model runs in a circuit simulator.

You can use the `rationalfit` function with only the two required arguments. The syntax is:

```
model_obj = rationalfit(freq,data)
```

where

- `model_obj` is a handle to the rational function model object.
- *`freq`* is a vector of frequency values that correspond to the data values.
- *`data`* is a vector that contains the data to fit.

For faster simulation, create a model object with the smallest number of poles required to accurately represent the component. To control the number of poles, use the syntax:

```
model_obj = rationalfit(freq,data,tol,weight,delayfactor)
```

where

- *`tol`* — the relative error-fitting tolerance, in decibels. Specify the largest acceptable tolerance for your application. Using tighter tolerance values may force the `rationalfit` function to add more poles to the model to achieve a better fit.
- *`weight`* — a vector that specifies the weighting of the fit at each frequency.
- *`delayfactor`* — a value that controls the amount of delay used to fit the data. Delay introduces a phase shift in the frequency domain that may require a large number of poles to fit using a rational function model. When you specify the delay factor, the `rationalfit` function represents the delay as an exponential phase shift. This phase shift allows the function to fit the data using fewer poles.

These arguments are described in detail in the `rationalfit` function reference page.

Note You can also specify the number of poles directly using the `npoles` argument. The model accuracy is not guaranteed with approach, so you should not specify `npoles` when accuracy is critical. For more information on the `npoles` argument, see the `rationalfit` reference page.

If you plan to integrate the Verilog-A module into a large design for simulation using detailed models, such as transistor-level circuit models, the simulation time consumed by a Verilog-A module may have a trivial impact on the overall simulation time. In this case, there is no reason to take the time to optimize the rational function model of the component.

For more information on the `rationalfit` function arguments, see the `rationalfit` reference page.

Write Verilog-A Module

You use the `writeva` method to create a Verilog-A module that describes the RF model object. This method writes the module to a specified file. Use the syntax:

```
status = writeva(model_obj,'obj1',{'inp','inn'},{'outp','outn'})
```

to write a Verilog-A module for the model object *`model_obj`* to the file `obj1.va`. The module has differential input nets, *`inp`* and *`inn`*, and differential output nets, *`outp`* and *`outn`*. The method returns `status`, a logical value of `true` if the operation is successful and `false` otherwise.

The `write` reference page describes the method arguments in detail.

An example of exporting a Verilog-A module appears in the RF Toolbox example, “Export Verilog-A module from Rational Function” on page 6-69.

See Also

More About

- “Model RF Objects Using Verilog-A” on page 3-2
- “Export Component Data to File” on page 2-27

AMP File Format

AMP File Data Sections

In this section...

Overview

The AMP data file describes a single nonlinear device. Its format can contain the following types of data:

- S, Y, or Z network parameters
- Noise parameters
- Noise figure data
- Power data
- IP3 data

An AMP file must contain either power data or network parameter data to be valid. To accommodate analysis at more than one frequency, the file can contain more than one section of power data. Noise data, noise figure data, and IP3 data are optional.

Note If the file contains both network parameter data and power data, RF Toolbox software checks the data for consistency. If the amplifier gain computed from the network parameters is not consistent with the gain computed from the power data, a warning appears.

Two AMP files, `samplepa1.amp` and `default.amp`, ship with the toolbox to show the AMP format. They describe a nonlinear 2-port amplifier with noise. See "Model Cascaded Network" for an example that shows how to use an AMP file.

Denoting Comments

An asterisk (`*`) or an exclamation point (`!`) precedes a comment that appears on a separate line.

A semicolon (`;`) precedes a comment that appears following data on the same line.

Data Sections

Each kind of data resides in its own section. Each section consists of a two-line header followed by lines of numeric data. Numeric values can be in any valid MATLAB format.

A new header indicates the end of the previous section. The data sections can appear in any order in the file.

Note In the data section descriptions, brackets ([]) indicate optional data or characters. All values are case insensitive.

S, Y, or Z Network Parameters

Header Line 1

The first line of the header has the format

```
Keyword [Parameter] [R[REF][=]value]
```

`Keyword` indicates the type of network parameter. Its value can be `S[PARAMETERS]`, `Y[PARAMETERS]`, or `Z[PARAMETERS]`. `Parameter` indicates the form of the data. Its value can be `MA`, `DB`, or `RI`. The default for S-parameters is `MA`. The default for Y- and Z-parameters is `RI`. `R[REF][=]value` is the reference impedance. The default reference impedance is 50 ohms.

Note `R[REF][=]value` must be a positive real scalar or vector. If `R[REF][=]value` is a vector, then the vector must be equal to the number of network parameter data points or frequency vector.

The following table explains the meaning of the allowable `Parameter` values.

Parameter	Description
MA	Data is given in (magnitude, angle) pairs with angle in degrees (default for S-parameters).
DB	Data is given in (dB-magnitude, angle) pairs with angle in degrees.
RI	Data is given in (real, imaginary) pairs (default for Y- and Z-parameters).

This example of a first line indicates that the section contains S-parameter data given in (real, imaginary) pairs, and that the reference impedance is 50 ohms.

```
S RI R 50
```

Header Line 2

The second line of the header has the format

```
Independent_variable Units
```

The data in a section is a function of the `Independent_variable`. Currently, for S-, Y-, and Z-parameters, the value of Independent_variable is always `F[REQ]`. `Units` indicates the default units of the frequency data. It can be `GHz`, `MHz`, or `KHz`. You must specify `Units`, but you can override this default on any given line of data.

This example of a second line indicates that the default units for frequency data is GHz.

```
FREQ GHZ
```

Data

The data that follows the header typically consists of nine columns.

The first column contains the frequency points where network parameters are measured. They can appear in any order. If the frequency is given in units other than those you specified as the default, you must follow the value with the appropriate units; there should be no intervening spaces. For example,

```
FREQ GHZ
1000MHZ  ...
2000MHZ  ...
3000MHZ  ...
```

Columns two though nine contain 2-port network parameters in the order N11, N21, N12, N22. Similar to the Touchstone format, each Nnn corresponds to two consecutive columns of data in the chosen form: MA, DB, or RI. The data can be in any valid MATLAB format.

This example is derived from the file `default.amp`. A comment line explains the column arrangement of the data where `re` indicates real and `im` indicates imaginary.

```
S RI R 50
FREQ GHZ
* FREQ     reS11      imS11      reS21      imS21    reS12     imS12      reS22      imS22
  1.00  -0.724725  -0.481324  -0.685727  1.782660  0.000000  0.000000  -0.074122  -0.321568
  1.01  -0.731774  -0.471453  -0.655990  1.798041  0.001399  0.000463  -0.076091  -0.319025
  1.02  -0.738760  -0.461585  -0.626185  1.813092  0.002733  0.000887  -0.077999  -0.316488
```

Noise Parameters

Header Line 1

The first line of the header has the format

```
Keyword
```

`Keyword` must be `NOI[SE]`.

Header Line 2

The second line of the header has the format

```
Variable Units
```

`Variable` must be `F[REQ]`. `Units` indicates the default units of the frequency data. It can be `GHz`, `MHz`, or `KHz`. You can override this default on any given line of data. This example of a second line indicates that frequency data is assumed to be in GHz, unless other units are specified.

```
FREQ GHz
```

Data

The data that follows the header must consist of five columns.

The first column contains the frequency points at which noise parameters were measured. The frequency points can appear in any order. If the frequency is given in units other than those you specified as the default, you must follow the value with the appropriate units; there should be no intervening spaces. For example,

```
NOI
FREQ GHZ
1000MHZ  ...
2000MHZ  ...
3        ...
4        ...
5        ...
```

Columns two through five contain, in order,

- Minimum noise figure in decibels
- Magnitude of the source reflection coefficient to realize minimum noise figure
- Phase in degrees of the source reflection coefficient
- Effective noise resistance normalized to the reference impedance of the network parameters

This example is taken from the file `default.amp`. A comment line explains the column arrangement of the data.

```
NOI RN
FREQ GHz
* Freq  Fmin(dB)  GammmaOpt(MA:Mag) GammmaOpt(MA:Ang) RN/Zo
  1.90  10.200000 1.234000         -78.400000         0.240000
  1.93  12.300000 1.235000         -68.600000         0.340000
  2.06  13.100000 1.254000         -56.700000         0.440000
  2.08  13.500000 1.534000         -52.800000         0.540000
  2.10  13.900000 1.263000         -44.400000         0.640000
```

Noise Figure Data

The AMP file format supports the use of frequency-dependent noise figure (NF) data.

Header Line 1

The first line of the header has the format

```
Keyword [Units]
```

For noise figure data, `Keyword` must be `NF`. The optional `Units` field indicates the default units of the NF data. Its value must be `dB`, i.e., data must be given in decibels.

This example of a first line indicates that the section contains NF data, which is assumed to be in decibels.

```
NF
```

Header Line 2

The second line of the header has the format

```
Variable Units
```

`Variable` must be `F[REQ]`. `Units` indicates the default units of the frequency data. It can be `GHz`, `MHz`, or `KHz`. This example of a second line indicates that frequency data is assumed to be in GHz.

```
FREQ GHz
```

Data

The data that follows the header typically consists of two columns.

The first column contains the frequency points at which the NF data are measured. Frequency points can appear in any order. For example,

```
NF
FREQ MHz
2090  ...
2180  ...
2270  ...
```

Column two contains the corresponding NF data in decibels.

This example is derived from the file `samplepa1.amp`.

```
NF dB
FREQ GHz
1.900   10.3963213
2.000   12.8797965
2.100   14.0611765
2.200   13.2556751
2.300   12.9498642
2.400   13.3244309
2.500   12.7545104
```

Note If your noise figure data consists of a single scalar value with no associated frequency, that same value is used for all frequencies. Enter the value in column 1 of the line following header line 2. You must include the second line of the header, but it is ignored.

Power Data

An AMP file describes power data as input power-dependent output power.

Header Line 1

The first line of the header has the format

```
Keyword [Units]
```

For power data, `Keyword` must be `POUT`, indicating that this section contains power data. Because output power is complex, `Units` indicates the default units of the magnitude of the output power data. It can be `dBW`, `dBm`, `mW`, or `W`. The default is `W`. You can override this default on any given line of data.

The following table explains the meaning of the allowable `Units` values.

Allowable Power Data Units

Units	Description
dBW	Decibels referenced to one watt
dBm	Decibels referenced to one milliwatt
mW	Milliwatts
W	Watts

This example of a first line indicates that the section contains output power data whose magnitude is assumed to be in decibels referenced to one milliwatt, unless other units are specified.

```
POUT dBm
```

Header Line 2

The second line of the header has the format

```
Keyword [Units] FREQ[=]value
```

`Keyword` must be `PIN`. `Units` indicates the default units of the input power data. The default is `W`. You can override this default on any given line of data. `FREQ[=]value` is the frequency point at which the power is measured. The units of the frequency point must be specified explicitly using the abbreviations `GHz`, `MHz`, `kHz`, or `Hz`.

This example of a second line indicates that the section contains input power data that is assumed to be in decibels referenced to one milliwatt, unless other units are specified. It also indicates that the power data was measured at a frequency of 2.1E+009 Hz.

```
PIN dBm FREQ=2.1E+009Hz
```

Data

The data that follows the header typically consists of three columns:

- The first column contains input power data. The data can appear in any order.
- The second column contains the corresponding output power magnitude.
- The third column contains the output phase shift in degrees.

Note RF Toolbox software does not use the phase data directly. RF Blockset™ blocks use this data in conjunction with RF Toolbox software to create the AM/PM conversion table for the Equivalent Baseband library General Amplifier and General Mixer blocks.

If all phases are zero, you can omit the third column. If all phases are zero or omitted, the toolbox assumes that the small signal phase from the network parameter section of the file (`180*angle(`S_{21}`(f))/pi`) is the phase for all power levels.

In contrast, if one or more phases in the power data section are nonzero, the toolbox interpolates and extrapolates the data to determine the phase at all power levels. The small signal phase (`180*angle(`S_{21}`(f))/pi`) from the network parameter section is ignored.

Inconsistency between the power data and network parameter sections of the file may cause incorrect results. To avoid this outcome, verify that the following criteria must is met:

- The lowest input power value for which power data exists falls in the small signal (linear) region.
- In the power table for each frequency point `f`, the power gain and phase at the lowest input power value are equal to `20*log10(abs(S21(f)))` and `180*angle(S21(f))/pi`, respectively, in the network parameter section.

If the power is given in units other than those you specified as the default, you must follow the value with the appropriate units. There should be no intervening spaces.

This example is derived from the file `default.amp`. A comment line explains the column arrangement of the data.

```
POUT dbm
PIN dBm FREQ = 2.10GHz
* Pin     Pout            Phase(degrees)
  0.0     19.28           0.0
  1.0     20.27           0.0
  2.0     21.26           0.0
```

Note The file can contain more than one section of power data, with each section corresponding to a different frequency value. When you analyze data from a file with multiple power data sections, power data is taken from the frequency point that is closest to the analysis frequency.

IP3 Data

An AMP file can include frequency-dependent, third-order input (IIP3) or output (OIP3) intercept points.

Header Line 1

The first line of the header has the format

```
Keyword [Units]
```

For IP3 data, `Keyword` can be either `IIP3` or `OIP3`, indicating that this section contains input IP3 data or output IP3 data. `Units` indicates the default units of the IP3 data. Valid values are `dBW`, `dBm`, `mW`, and `W`. The default is `W`.

This example of a first line indicates that the section contains input IP3 data which is assumed to be in decibels referenced to one milliwatt.

```
IIP3 dBm
```

Header Line 2

The second line of the header has the format

```
Variable Units
```

`Variable` must be `FREQ`. `Units` indicates the default units of the frequency data. Valid values are `GHz`, `MHz`, and `KHz`. This example of a second line indicates that frequency data is assumed to be in GHz.

```
FREQ GHz
```

Data

The data that follows the header typically consists of two columns.

The first column contains the frequency points at which the IP3 parameters are measured. Frequency points can appear in any order.

```
OIP3
FREQ GHz
2.010  ...
2.020  ...
2.030  ...
```

Column two contains the corresponding IP3 data.

This example is derived from the file `samplepa1.amp`.

```
OIP3 dBm
FREQ GHz
2.100   38.8730377
```

Note If your IP3 data consists of a single scalar value with no associated frequency, then that same value is used for all frequencies. Enter the value in column 1 of the line following header line 2. You must include the second line of the header, but the application ignores it.

Inconsistent Data Sections

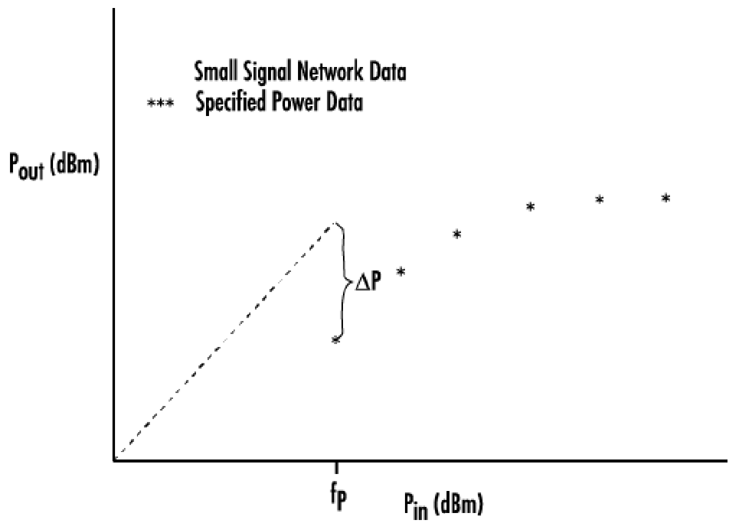
If an AMP file contains both network parameter data and power data, RF Toolbox software checks the data for consistency.

The toolbox compares the small-signal amplifier gain defined by the network parameters, S_{21}, and by the power data, $P_{out} - P_{in}$. The discrepancy between the two is computed in dBm using the following equation:

$$\Delta P = S_{21}(f_P) - P_{out}(f_P) + P_{in}(f_P)$$

wheref_P is the lowest frequency for which power data is specified.

The discrepancy is shown in the following graph.



If ΔP is more than 0.4 dB, a warning appears. Large discrepancies may indicate measurement errors that require resolution.

See Also

More About

- “S-Parameter Notation”
- “Determining Parameter Formats” on page 5-2

How Tos, Definitions, Algorithms

Determining Parameter Formats

In this section...

When you call `plotyy` without specifying the formats for the specified parameter, `plotyy` determines the formats from the primary and secondary formats.

Primary and Secondary Formats

The following table shows the primary and secondary formats for the parameters for all circuit and data objects. Use the `listparam` method to list the valid parameters for a particular object. Use the `listformat` method to list valid formats.

Parameter	Primary Format	Secondary Format
S11, S12, S21, S22	Magnitude(decibels)	Angle(Degrees)
LS11, LS12, LS21, LS22	Magnitude(decibels)	Angle(Degrees)
NF	Magnitude(decibels)	none
OIP3	dBm	W
Pout	dBm	W
Phase	Angle(Degrees)	none
AM/AM	Magnitude(decibels)	none
AM/PM	Angle(Degrees)	none
GammaIn, GammaOut	Magnitude(decibels)	Angle(Degrees)
Gt, Ga, Gp, Gmag, Gmsg	Magnitude(decibels)	none
Delta	Magnitude(decibels)	Angle(Degrees)
TF1, TF2	Magnitude(decibels)	Angle(Degrees)
GammaMS, GammaML	Magnitude(decibels)	Angle(Degrees)
VSWRIn, VSWROut	Magnitude(decibels)	none
GroupDelay	ns	none
Fmin	Magnitude(decibels)	none
GammaOPT	Magnitude(decibels)	Angle(Degrees)
K, Mu, MuPrime	none	none
RN	none	none
PhaseNoise	dBc/Hz	none
NTemp	K	none
NFactor	none	none

Determining Formats for One Parameter

When you specify only one parameter for plotting, `plotyy` creates the plot as follows:

- The predefined primary format is the format for the left Y-axis.
- The predefined secondary format is the format for the right Y-axis.

If the specified parameter does not have the predefined secondary format, `plotyy` behaves the same way as `plot`, and does not add a second y-axis to the plot.

Determining Formats for Multiple Parameters

To plot multiple parameters on two Y-axes, `plotyy` tries to find two formats from the predefined primary and secondary formats for the specified parameters. To be used in the plot, the formats must meet the following criteria:

- Each format must be a valid format for at least one parameter.
- Each parameter must be plotted at least on one Y-axis.

If cannot meet these criteria,`plotyy` it issues an error message.

The function uses the following algorithm to determine the two parameters:

1. Look up the primary and secondary formats for the specified parameters.
2. If one or more pairs of primary-secondary formats meets the preceding criteria for all parameters:

   - Select the pair that applies to the most parameters.
   - Use these formats to create the plot.

   Otherwise, proceed to the next step.
3. If no pairs of primary-secondary formats meet the criteria for all parameters, try to find one or more pairs of primary-primary format that meets the criteria. If one or more pairs of primary-primary formats meets the preceding criteria for all parameters:

   - Select the pair that applies to the most parameters.
   - Use these formats to create the plot.

   Otherwise, proceed to the next step.
4. If the preceding steps fail to produce a plot, try to find one format from the predefined primary formats. If a primary format is valid for all parameters, use this format to create the plot with the MATLAB `plot` function.
5. If all the preceding steps are not successful, issue an error message.

See Also

More About

- "S-Parameter Notation"
- "AMP File Data Sections" on page 4-2

RF and Microwave Filter Modeling

Receivers, transmitters, and frequency synthesizers use RF and microwave filters to select or reject signals with a particular band of frequency. Filtering a signal modifies its phase and magnitude components. RF receiver systems use filters such as the RF preselector filter, image rejection filter, and IF filter. You can design filters suited to your application using the `rffilter` object from RF Toolbox or the Filter blocks from RF Blockset. For example, you can design an RF preselector filter in Chebyshev configurations using an `rffilter` object or the Circuit Envelope Filter block and filter undesired frequency bands causing spurious emissions and intermodulation distortions in mixers [1].

You can also use the `rffilter` object to create a Butterworth or inverse Chebyshev filter. The `rffilter` object also supports implementing a transfer function. For more information, see “Design Data for Transfer Function Implementation”. In addition to the three configurations, you can also design an ideal filter in Simulink® environment using the Circuit Envelope Filter block. Ideal filters perfectly allow frequencies in the passband and completely reject frequencies in the stopband.

Microstrip filters play an important role in microwave applications. Almost all communication systems contain an RF front end that performs signal processing using RF and microwave filters. You can design coupled-line, hairpin, and stepped-impedance lowpass filters in the microstrip form. These filters have very low insertion loss and are easy to fabricate in compact sizes. Design these filters in a printed circuit board (PCB) using RF PCB Toolbox™ filter objects.

Design Workflows

Design RF and microwave filters using these workflows:

- “Design, Visualize and Explore Inverse Chebyshev Filter - I” on page 6-194 — This workflow shows how to determine the transfer function for a fifth-order inverse Chebyshev lowpass filter with 1 dB passband attenuation, cutoff frequency of 1 rad/sec, and a minimum attenuation of 50 dB in the stopband.
- “Frequency Response of RF Transmit/Receive Duplex Filter” (RF Blockset) — This workflow shows how to use RF Blockset Circuit Envelope blocks to simulate a transmit/receive duplex filter and calculate the frequency response curves from a broadband white-noise input.
- “Stepped Impedance Maximally Flat Lowpass Filter for Microwave Applications” (RF PCB Toolbox) — This workflow shows you how to design a stepped-impedance lowpass filter for X-band applications.
- “Model RF Filter Using Circuit Envelope” (RF Blockset) — This workflow shows how to model an RF filter using the Circuit Envelope library and compare the input and output signal amplitudes to study signal attenuation.
- “Design and Analyze HighPass Filter Using pcbComponent” (RF PCB Toolbox) — This example shows how to design and analyze a microstrip highpass filter. The filter design is based on a three-pole Chebyshev highpass prototype with 0.1 dB passband ripple and a cutoff frequency of 1.5 GHz.

References

[1] Besser, Les, and Rowan Gilmore. *Practical RF Circuit Design for Modern Wireless Systems. 1: Passive Circuits and Systems*. Boston, Mass.: Artech House, 2003.

See Also

Related Examples

- "Designing Matching Networks in RF Systems" on page 5-6

Designing Matching Networks in RF Systems

Matching networks in RF system enable maximum power transfer from the source to the load. You can design a set of circuits that match the impedance of a source to the impedance of a load at a specific center frequency using the `matchingnetwork` object or the **Matching Network Designer** app. You can also use the object and the app to visualize, and compare matching networks for the one-port loads.

Using the object and the app, you can also:

- Design two- and three-component lumped-element matching networks at desired frequencies and unloaded-Q factors.
- Provide source and load impedance as a one-port Touchstone file, scalar impedance, RF circuit object, RF network parameter object, Antenna Toolbox™ object, or as an anonymous function.
- Sort matching networks using constraints such as operating frequency range and power wave S-parameters.
- Plot power wave S-parameters [1] of the matching network on a Smith chart and Cartesian plot.
- Plot voltage standing wave ratio (VSWR) and impedance transformation plots.
- Plot magnitude, phase, real, and imaginary parts of power wave S-parameters of the matching network.
- Export selected networks as `circuit` objects or power wave S-parameters as `sparameters` objects.

Available Configuration

You can design matching networks in these network configurations:

- Pi Topology
- T Topology
- L Topology
- Three-Components

Design Workflows

You can design matching networks for RF systems using these workflows:

- “Design Matching Networks for Passive Multiport Network” on page 6-205 — This workflow shows how to design matching networks for 16-port passive networks at 39 GHz for 5G mm Wave systems. You design a matching network for each port that functions between two 1-port terminations.
- “Design Broadband Matching Networks for Antennas” on page 6-140 — This workflow shows how to design a broadband matching network between a resistive source and inductive load using optimization with direct search methods.
- “Designing Matching Networks for Low Noise Amplifiers” on page 6-122 — This workflow shows how to verify the design of input and output matching networks for a low noise amplifier (LNA) using a gain and noise figure plot.
- “Design Broadband Matching Networks for Amplifier” on page 6-149 — This workflow shows how to design broadband matching networks for a LNA.

- “Design of Quarter-Wave Transformer for Impedance Matching Applications” (RF PCB Toolbox) — This workflow shows how to design a quarter-wave transformer for impedance-matching applications using `pcbComponent`, `microstripLine`, and `traceRectangular` objects.

References

[1] Kurokawa, K. “Power Waves and the Scattering Matrix.” *IEEE Transactions on Microwave Theory and Techniques* 13, no. 2 (March 1965): 194–202. https://doi.org/10.1109/TMTT.1965.1125964.

[2] Ludwig, Reinhold, and Gene Bogdanov. *RF Circuit Design: Theory and Applications*. Upper Saddle River, NJ: Prentice-Hall, 2009.

See Also

Related Examples

- “RF and Microwave Filter Modeling” on page 5-4

RF Toolbox Examples

Superheterodyne Receiver Using RF Budget Analyzer App

This example shows how to build a superheterodyne receiver and analyze the receiver's RF budget for gain, noise figure, and IP3 using the RF Budget Analyzer app. The receiver is a part of a transmitter-receiver system described in the IEEE conference papers, [1] and [2].

Introduction

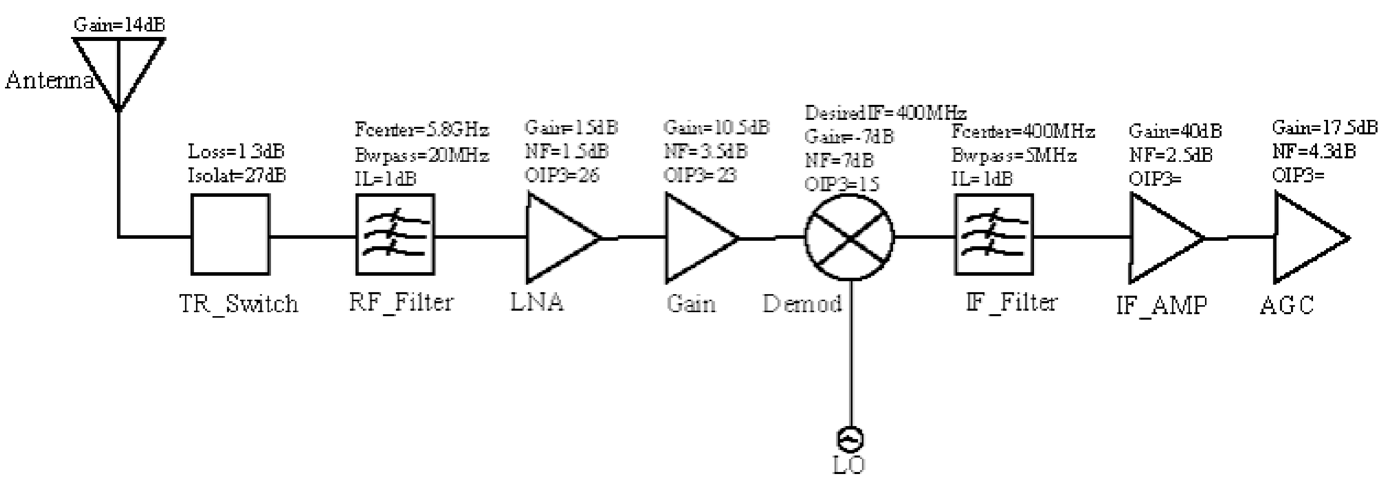
RF system designers begin the design process with a budget specification for how much gain, noise figure (NF), and nonlinearity (IP3) the entire system must satisfy. To assure the feasibility of an architecture modeled as a simple cascade of RF elements, designers calculate both the per-stage and cascade values of gain, noise figure and IP3 (third-intercept point).

Using the RF Budget Analyzer app, you can:

- Build a cascade of RF elements.
- Calculate the per-stage and cascade output power, gain, noise figure, SNR, and IP3 of the system.
- Export the per-stage and cascade values to the MATLAB™ workspace.
- Export the system design to RF Blockset for simulation.
- Export the system design to RF Blockset measurement testbench as a DUT (device under test) subsystem and verify the results obtained using the App.

System Architecture

The receiver system architecture designed using the app is:



The receiver bandwidth is between 5.825 GHz and 5.845 GHz.

Build Superheterodyne Receiver

You can build all the components of the superheterodyne receiver using MATLAB command line and view the analysis using the RF Budget Analyzer app.

The first components in the superheterodyne receiver system architecture are the **antenna** and **TR switch**. We replace the antenna block with the effective power reaching the switch.

1. The system uses the TR switch to switch between the transmitter and the receiver. The switch adds a loss of 1.3 dB to the system. Create a `TRSwitch` with a gain of -1.3 dB, and OIP3 of 37 dBm. To match the RF budget results from reference [1], the noise figure is assumed to be 2.3 dB.

```
elements(1) = rfelement('Name','TRSwitch','Gain',-1.3,'NF',2.3,'OIP3',37);
```

2. To model the RF bandpass filter use `rffilter` to design the filter. From the example “Design IF Butterworth Bandpass Filter” on page 6-183, load impedance of the filter is found to be 132.986 Ohms. But for budget calculation, each stage is terminated by 50 Ohms internally. Therefore, to achieve an insertion loss of 1 dB, the input impedance, Zin of the next element, i.e., amplifier, is set to 132.896 Ohms.

```
Fcenter = 5.8e9;
Bwpass  = 20e6;
Z       = 132.986;
elements(2) = rffilter('ResponseType','Bandpass',                         ...
    'FilterType','Butterworth','FilterOrder',6,                           ...
    'PassbandAttenuation',10*log10(2),                                    ...
    'Implementation','Transfer function',                                 ...
    'PassbandFrequency',[Fcenter-Bwpass/2 Fcenter+Bwpass/2],'Zout',50,   ...
    'Name','RF_Filter');
```

The S-parameters for this filter are not ideal and automatically inserts a loss of approximately -1dB into the system.

3. Use the `amplifier` object to model a `Low Noise Amplifier` block with a gain of 15 dB, noise figure of 1.5 dB, and OIP3 of 26 dBm.

```
elements(3) = amplifier( 'Name','LNA','Gain',15,'NF',1.5,'OIP3',26,     ...
    'Zin',Z);
```

4. Model a `Gain` block with a gain of 10.5 dB, noise figure of 3.5 dB, and OIP3 of 23 dBm.

```
elements(4) = amplifier( 'Name','Gain','Gain',10.5,'NF',3.5,'OIP3',23);
```

5. The receiver downconverts the RF frequency to an IF frequency of 400 MHz. Use the `modulator` object to create **Demodulator** block with a LO (Local Oscillator) frequency of 5.4 GHz, gain of -7 dB, noise figure of 7 dB, and OIP3 of 15 dBm.

```
elements(5) = modulator('Name','Demod','Gain',-7,'NF',7,'OIP3',15,      ...
    'LO',5.4e9, 'ConverterType','Down');
```

6. To model the RF bandpass filter use `rffilter` to design the filter.

```
Fcenter = 400e6;
Bwpass  = 5e6;
elements(6) = rffilter('ResponseType','Bandpass',                         ...
    'FilterType','Butterworth','FilterOrder',4,                           ...
    'PassbandAttenuation',10*log10(2),                                    ...
    'Implementation','Transfer function',                                 ...
    'PassbandFrequency',[Fcenter-Bwpass/2 Fcenter+Bwpass/2],'Zout',50,   ...
    'Name','IF_Filter');
```

The S-parameters for this filter are not ideal and automatically inserts a loss of approximately -1dB into the system.

7. Model an `IF Amplifier` block with a gain of 40 dB and a noise figure of 2.5 dB.

```
elements(7) = amplifier( 'Name','IFAmp','Gain',40,'NF',2.5,'Zin',Z);
```

8. As seen in the references, the receiver uses an AGC (Automatic Gain Control) block where the gain varies with the available input power level. For an input power of -80 dB, the AGC gain is at a maximum of 17.5 dB. Use an Amplifier block to model an AGC. Model an `AGC` block with a gain of 17.5 dB, noise figure of 4.3 dB, and OIP3 of 36 dBm.

```
elements(8) = amplifier('Name','AGC','Gain',17.5,'NF',4.3,'OIP3',36);
```

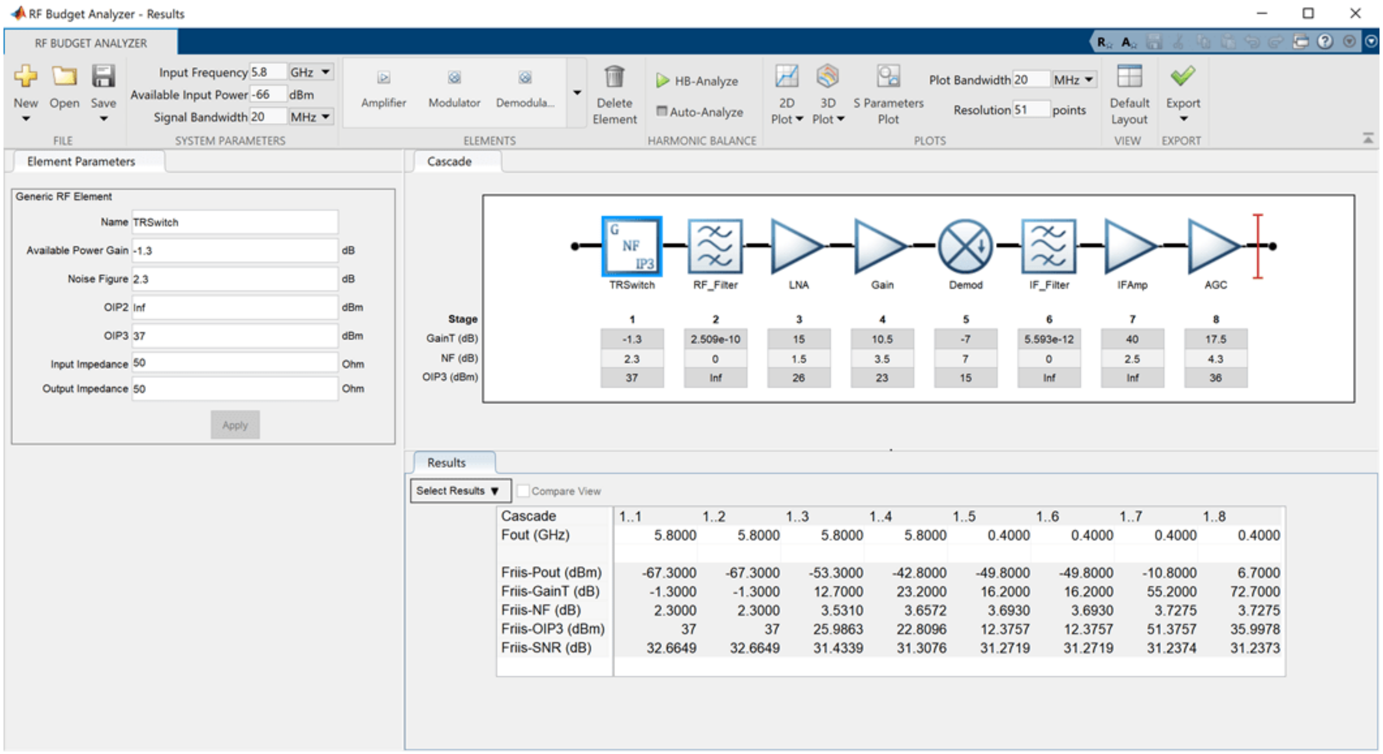
9. Calculate the rbudget of the superheterodyne receiver using the following `System Parameters`: 5.8 GHz for `Input frequency`, -80 dB for `Available input power`, and 20 MHz for `Signal bandwidth`. Replace the antenna element with the effective `Available input power` which is estimated to be -66 dB reaching the TRswitch

```
superhet = rfbudget( 'Elements',elements,'InputFrequency',5.8e9,        ...
    'AvailableInputPower',-66,'SignalBandwidth',20e6)
```

```
superhet = 
  rfbudget with properties:

               Elements: [1x8 rf.internal.rfbudget.Element]
         InputFrequency: 5.8 GHz
    AvailableInputPower: -66 dBm
        SignalBandwidth:  20 MHz
                 Solver: Friis
             AutoUpdate: true

   Analysis Results
        OutputFrequency: (GHz) [  5.8    5.8    5.8      5.8     0.4     0.4     0.4    0.4]
            OutputPower: (dBm) [-67.3  -67.3  -53.3    -42.8   -49.8   -49.8   -10.8    6.7]
         TransducerGain: (dB)  [ -1.3   -1.3   12.7     23.2    16.2    16.2    55.2   72.7]
                     NF: (dB)  [  2.3    2.3  3.531    3.657   3.693   3.693   3.728  3.728]
                   IIP2: (dBm) []
                   OIP2: (dBm) []
                   IIP3: (dBm) [ 38.3   38.3  13.29  -0.3904  -3.824  -3.824  -3.824  -36.7]
                   OIP3: (dBm) [   37     37  25.99    22.81   12.38   12.38   51.38     36]
                    SNR: (dB)  [32.66  32.66  31.43    31.31   31.27   31.27   31.24  31.24]
```

View the analysis in the **RF Budget Analyser** app by typing this command at the command line.

```
show(superhet);
```



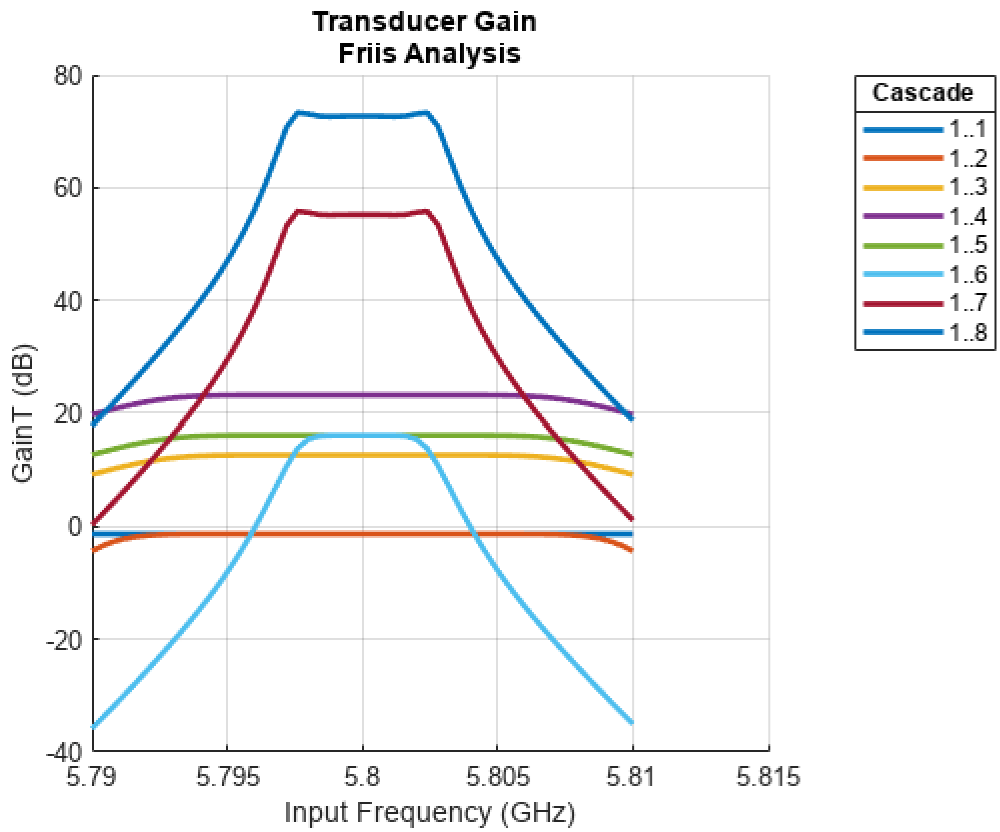
10. The app displays the cascade values such as: output frequency of the receiver, output power, gain, noise figure, OIP3, and SNR (Signal-to- Noise-Ratio).

11. The RF Budget Analyzer app saves the model in a MAT-file format.

Plot Cascade Transducer Gain and Cascade Noise Figure

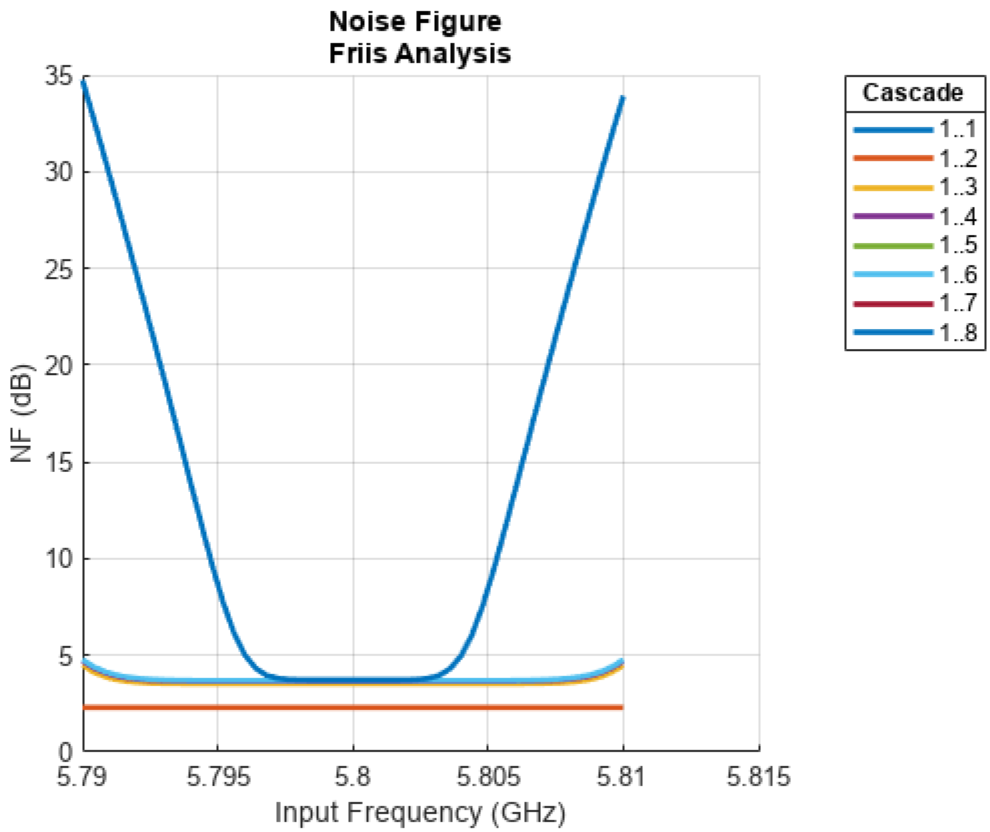
1. Plot the cascade transducer gain of the receiver using the function, `rfplot`

```
rfplot(superhet,'GainT')
view(90,0)
```



2. Plot the cascade noise figure of the receiver.

```
rfplot(superhet,'NF')
view(90,0)
```



You can also use the `Plot` button on the `RFBudgetAnalyzer` app to plot the different output values.

Export to MATLAB Script

1. You can also export the model to MATLAB script format using the **Export** button or:

```
h = exportScript(superhet);
```

The script opens automatically in a MATLAB Editor window.

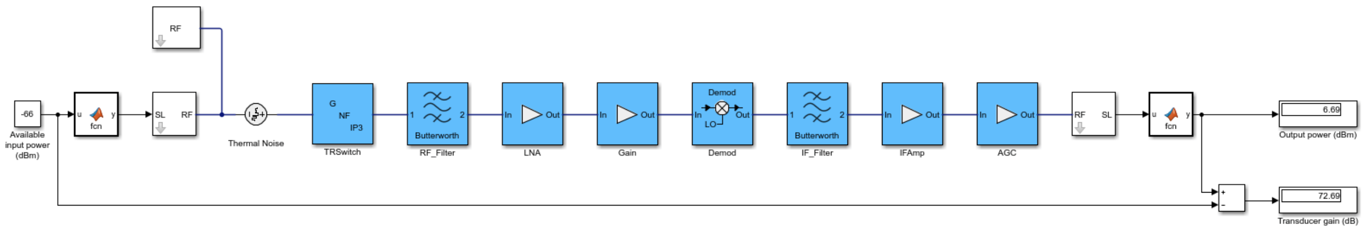
```
h.closeNoPrompt
```

Verify Output Power and Transducer Gain Using RF Blockset Simulation

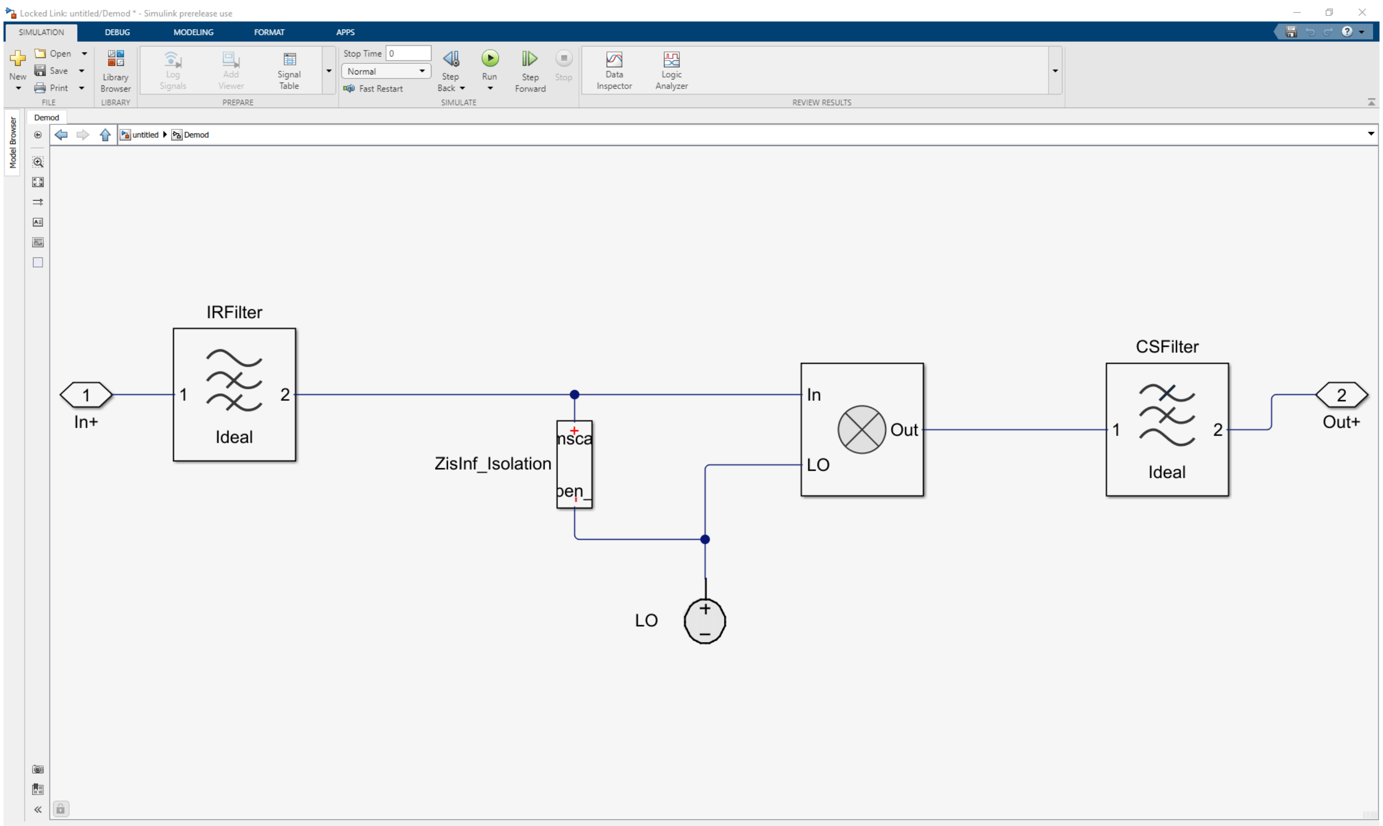
1. Use the **Export** button to export the receiver to RF Blockset or:

exportRFBlockset(superhet)

2. Run the RF Blockset model to calculate the **Output power (dBm)** and **Transducer gain (dB)** of the receiver. Note that the results match the **Pout (dBm)** and the **GainT (dB)** values of the receiver obtained using the RF Budget Analyzer app.



3. Look under the mask of the Demodulator block. This block consists of an ideal filter and a channel select filter and an LO (local oscillator) for frequency up or down conversion.



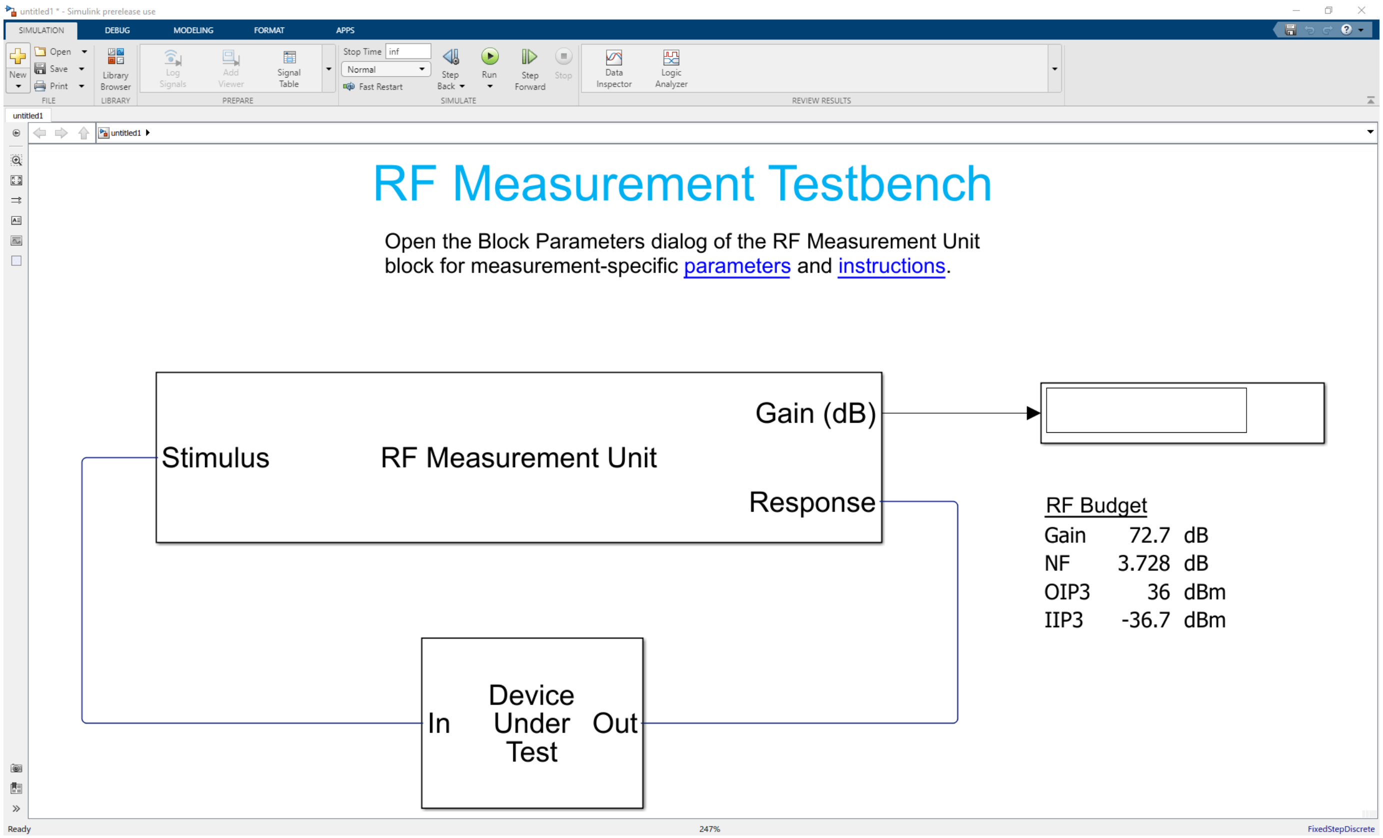
4. The stop time for the simulation is zero. To simulate time-varying results, you need to change the stop time.

Export to RF Blockset Testbench

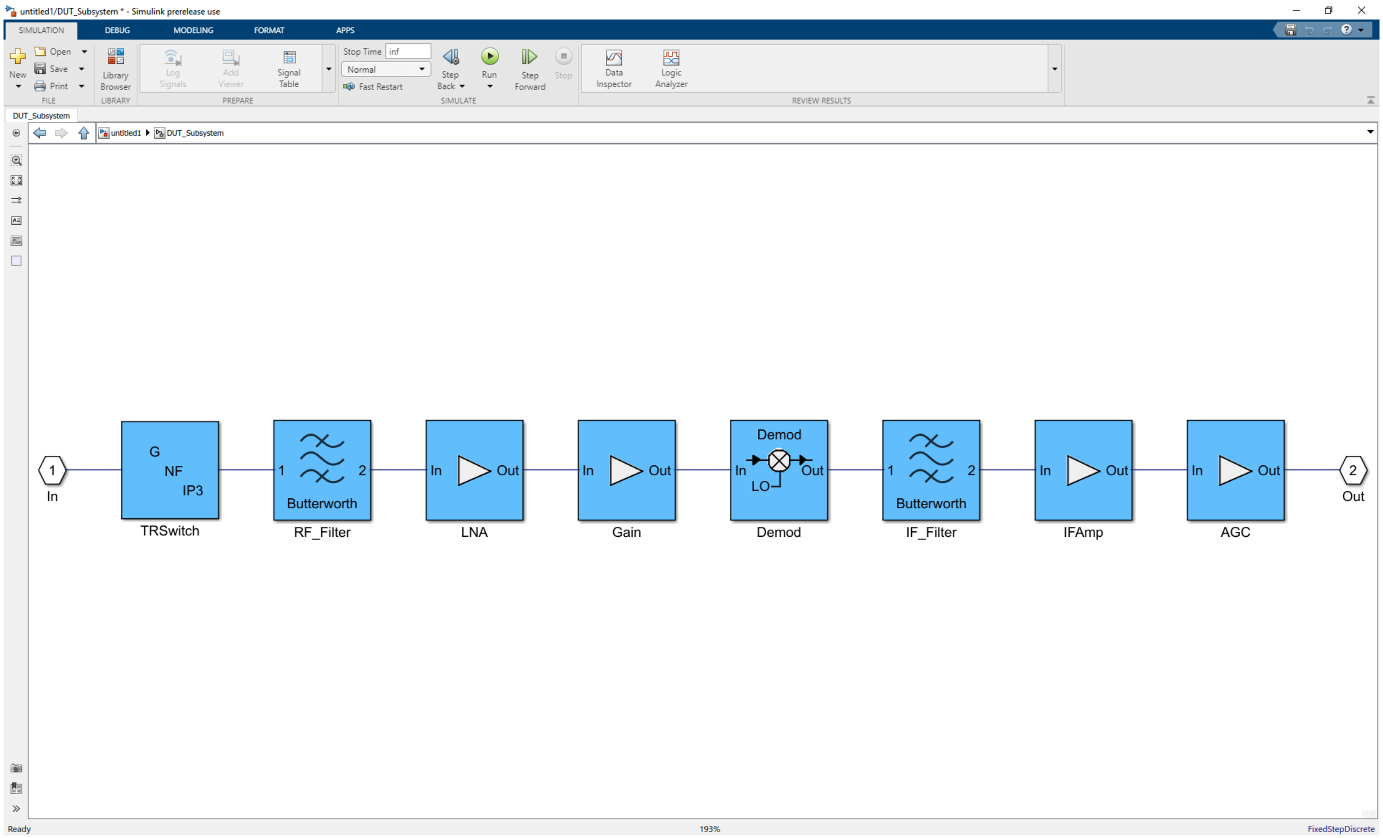
1. Use the **Export** button to export the receiver to RF Blockset measurement testbench or:

exportTestbench(superhet);

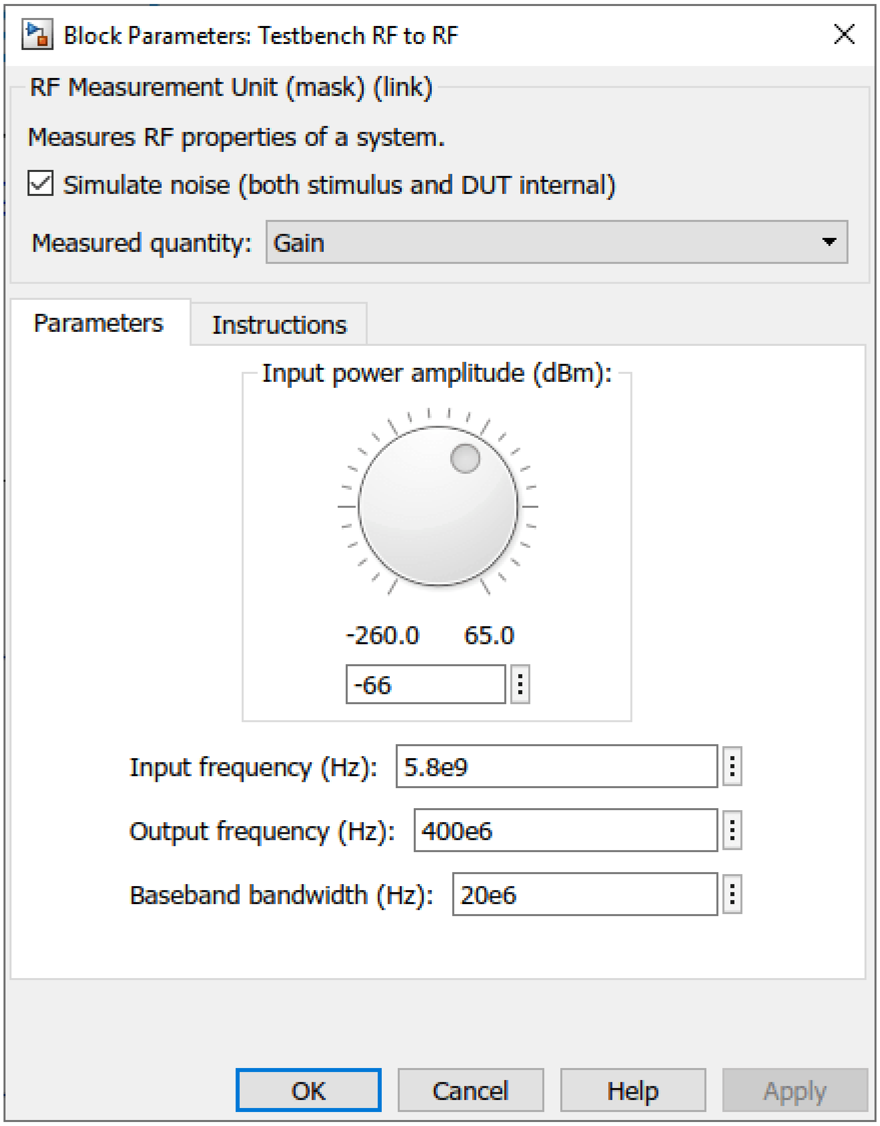
2. The RF Blockset testbench consists of two subsystems, `RF Measurement Unit` and `Device Under Test`.



3. The `Device Under Test` subsystem block contains the superheterodyne receiver you exported from the **RF Budget Analyzer** app. Double-click on the DUT subsystem block to look inside.



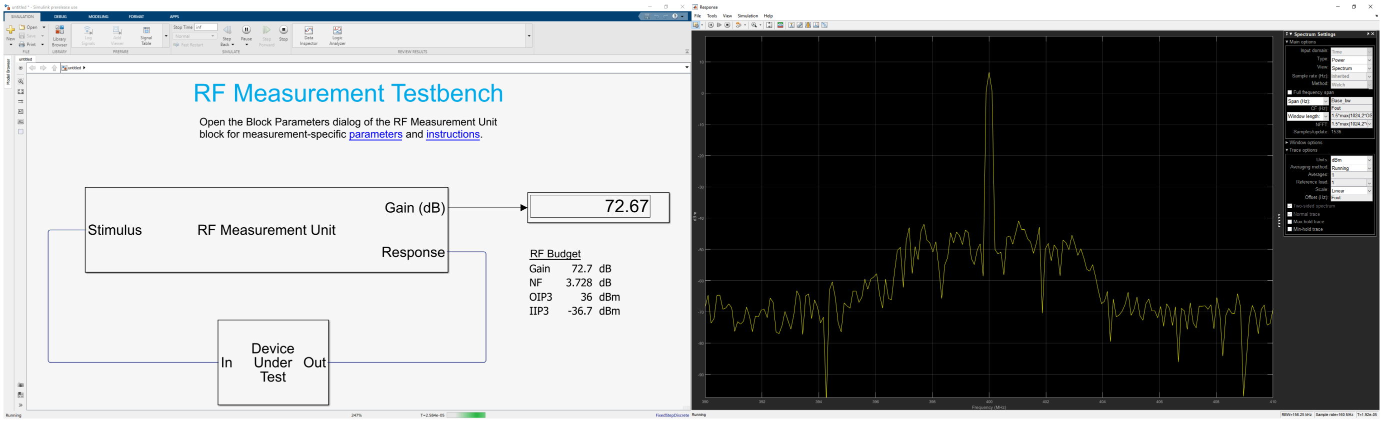
4. Double-click on the `RF Measurement Unit` subsystem block to see the system parameters. By default, RF Blockset testbench verifies gain.



Verify Gain, Noise Figure, and IP3 Using RF Blockset Testbench

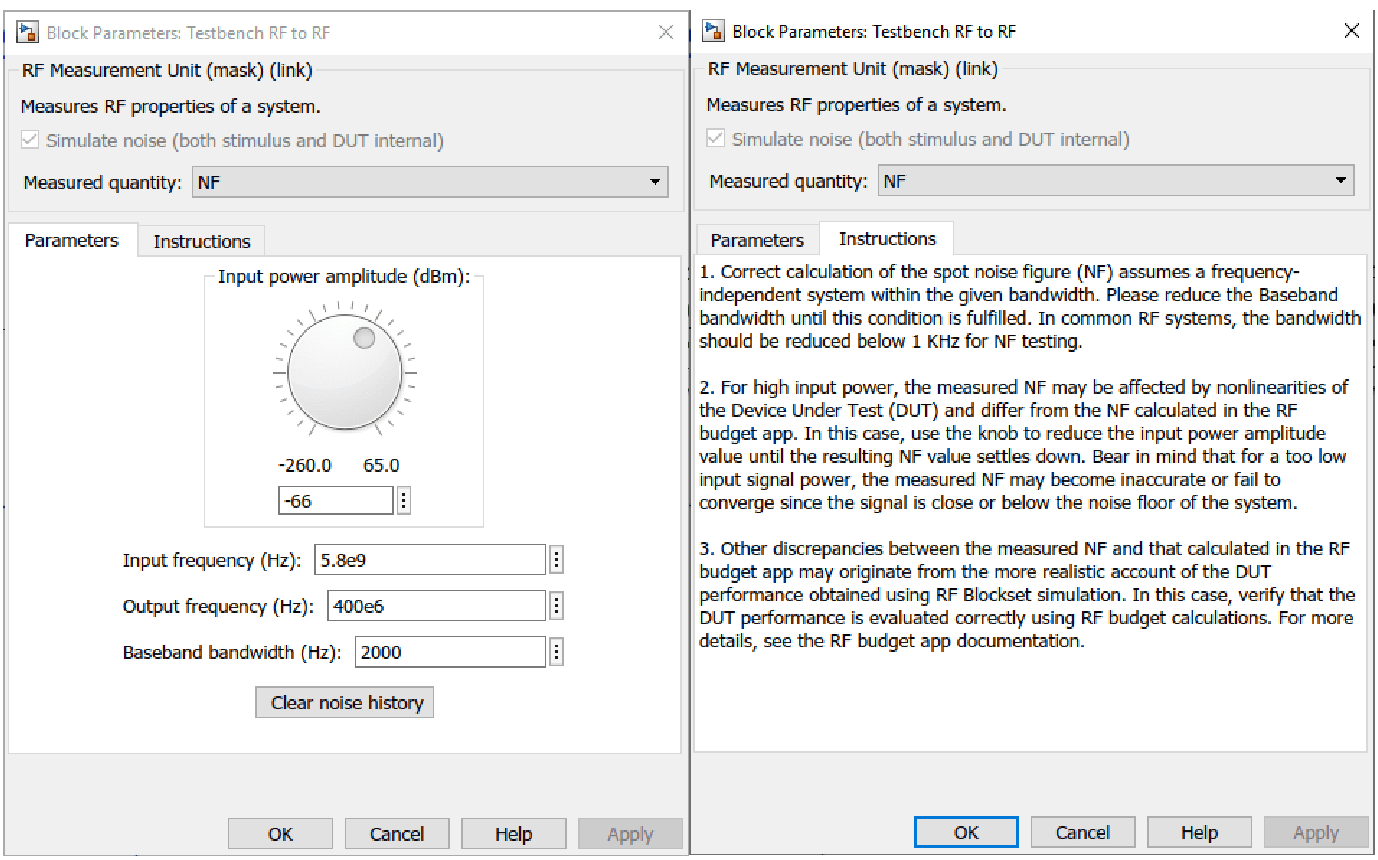
You can verify the gain, noise figure, and IP3 measurements using the RF Blockset testbench.

1. By default, the model verifies the gain measurement of the device under test. Run the model to check the gain value. The simulated gain value matches the cascade transducer gain value from the app. The scope shows an output power of approximately 6.7 dB at 400 MHz that matches the output power value in the **RF Budget Analyzer** app.

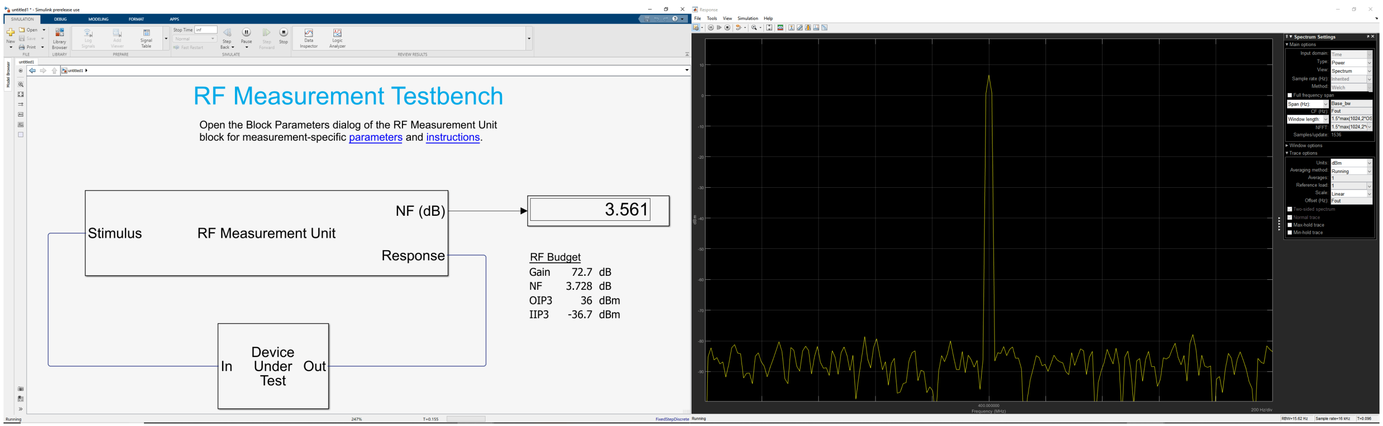


2. The RF Blockset testbench calculates the spot noise figure. The calculation assumes a frequency independent system within a given bandwidth. To simulate a frequency independent system and calculate the correct noise figure value, you need to reduce the broad bandwidth of 20 MHz to a narrow bandwidth.

3. First, stop all simulations. Double-click on the `RF Measurement Unit` Block. This opens the RF measurement unit parameters. In the **Measured Quantity** parameter drop down, change the parameter to **NF** (noise figure). In the **Parameters** tab, change the **Baseband bandwidth (Hz)** to 2000 Hz. Click **Apply**. To learn more about how to manipulate noise figure verification, click the **Instructions** tab.

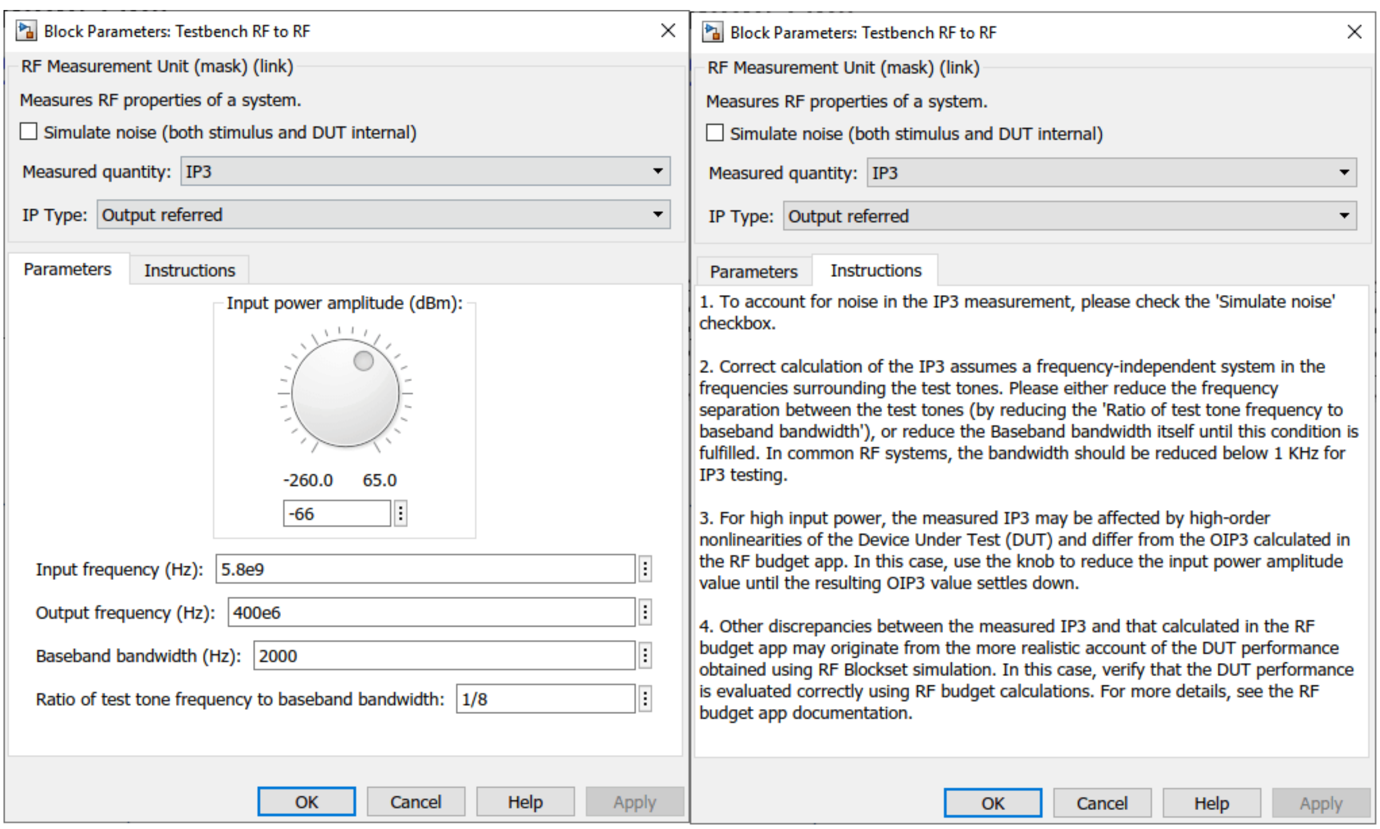


4. Run the model again to check the noise figure value. The testbench noise figure value matches the cascade noise figure value from the **RF Budget Analyzer** app.

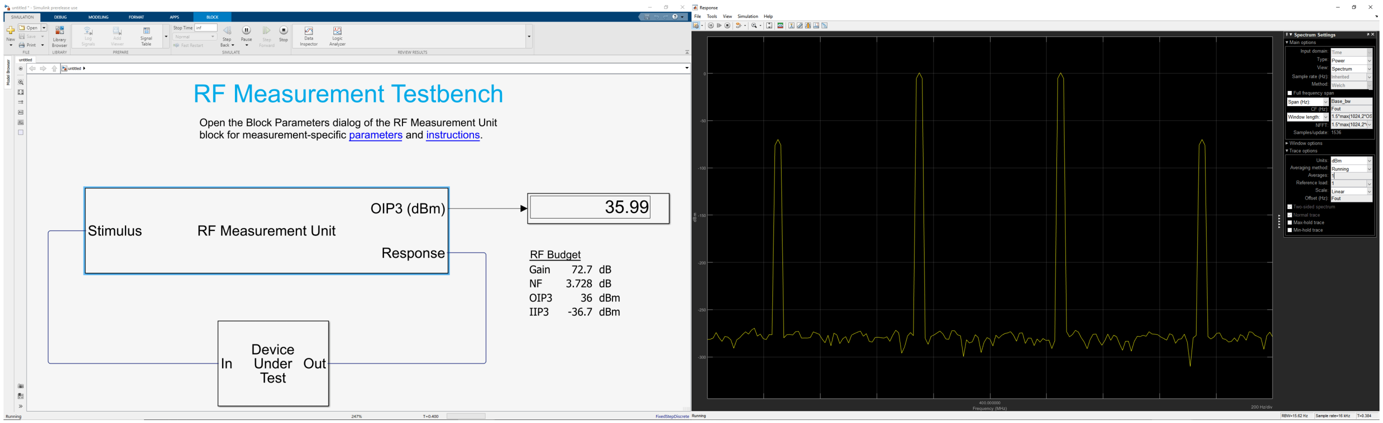


5. IP3 measurements rely on the creation and measurement of intermodulation tones that are usually small in amplitude and may be below the DUT's noise floor. For accurate IP3 measurements, clear the **Simulate noise** checkbox.

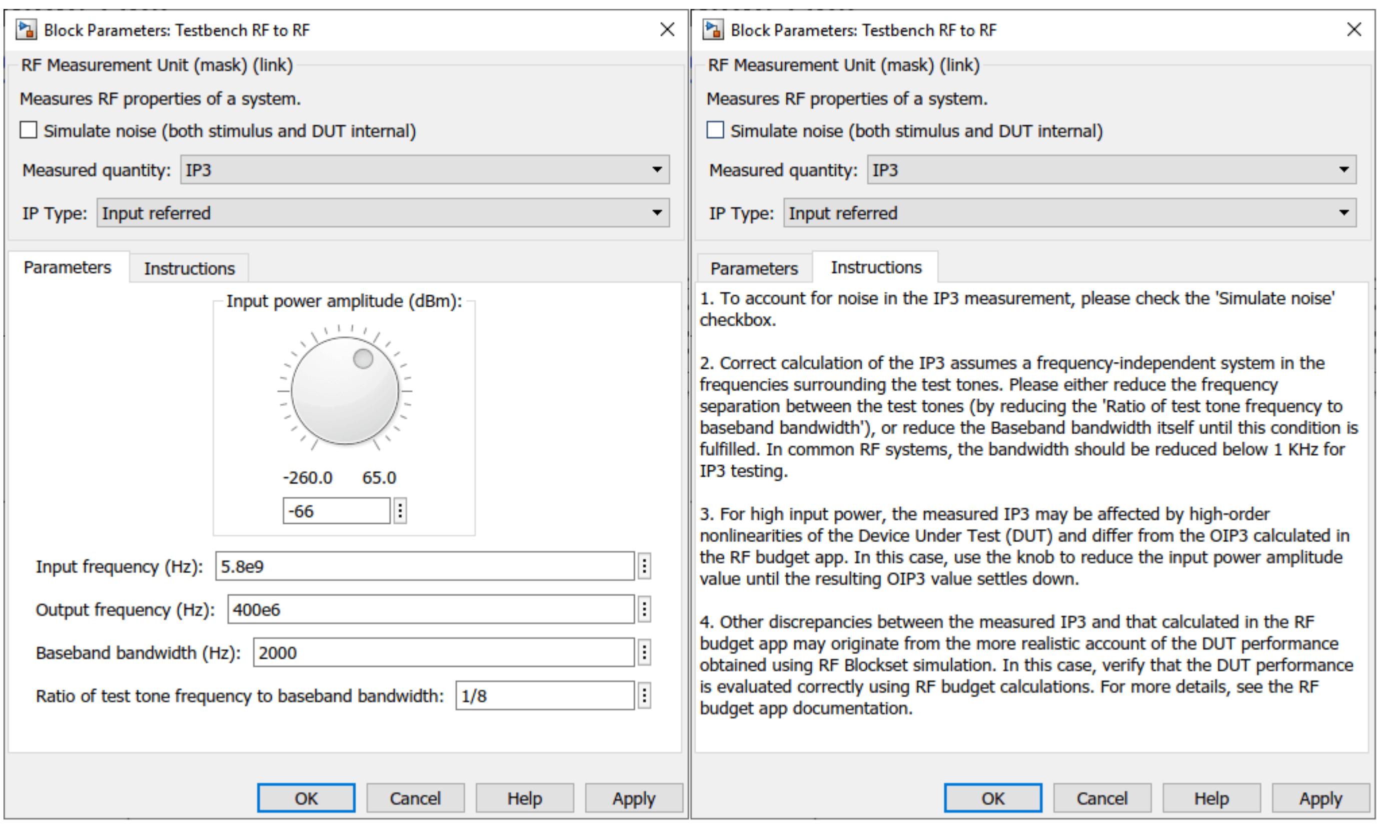
6. To verify OIP3 (output third-order intercept), stop all simulations. Open the `RF Measurement Unit` dialog box. Clear the **Simulate noise (both stimulus and DUT internal)** check box. Change the **Measured Quantity** parameter to **IP3**. Keep the **IP Type** as **Output referred**. To learn more about how to manipulate OIP3 verification, click the **Instructions** tab. Click **Apply**.



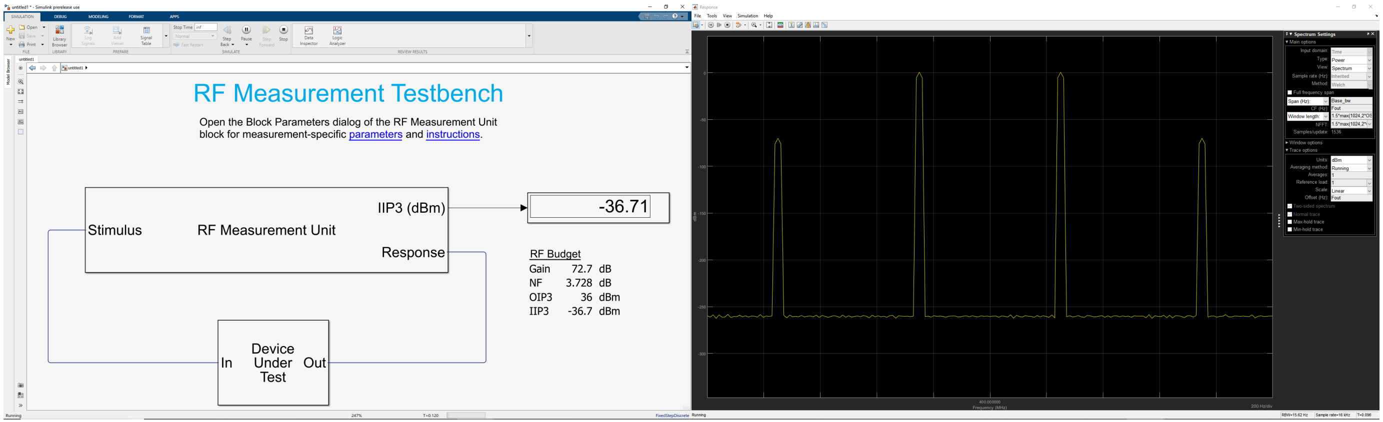
7. Run the model. The testbench OIP3 value matches the cascade OIP3 value of the app.



8. To verify IIP3 (input third-order intercept), stop all simulations. Open `RF Measurement Unit` dialog box. Clear the **Simulate noise (both stimulus and DUT internal)** check box. Change the **Measured Quantity** parameter in block parameters to **IP3**. Change the **IP Type** to **Input referred**. To learn more about how to manipulate IIP3 verification, click the **Instructions** tab. Click **Apply**.



9. Run the model again to check the IIP3 value.



References

[1] Hongbao Zhou, Bin Luo. " Design and budget analysis of RF receiver of 5.8GHz ETC reader" Published at Communication Technology (ICCT), 2010 12th IEEE International Conference, Nanjing, China, November 2010.

[2] Bin Luo, Peng Li. "Budget Analysis of RF Transceiver Used in 5.8GHz RFID Reader Based on the ETC-DSRC National Specifications of China" Published at Wireless Communications, Networking and Mobile Computing, WiCom '09. 5th International Conference, Beijing, China, September 2009.

See Also

RF Budget Analyzer

More About

- “Visualizing RF Budget Analysis over Bandwidth” on page 6-16
- “Bandpass Filter Response” on page 6-23

Visualizing RF Budget Analysis over Bandwidth

This example shows how to programmatically perform an RF budget analysis of an RF receiver system and visualize the computed budget results across the bandwidth of the input signal.

First, use `amplifier`, `modulator`, `rfelement`, and `nport` objects to specify the 2-port RF elements in a design. Then compute RF budget results by cascading the elements together into an RF system with `rfbudget`.

The `rfbudget` object enables design exploration and visualization at the MATLAB® command-line or graphically in the RF Budget Analyzer app. It also enables automatic RF Blockset™ model and measurement testbench generation.

Introduction

RF system designers typically begin a design process with budget specifications for the gain, noise figure (NF), and nonlinearity (IP3) of the entire system.

MATLAB functionality supporting RF budget analysis makes it easy to visualize gain, NF and IP3 results at multiple frequencies throughout the bandwidth of the signal. You can:

- Programmatically build an `rfbudget` object out of 2-port RF elements.
- Use the Command Line display of the `rfbudget` object to view single-frequency budget results.
- Vectorize the input frequency of the `rfbudget` object and use MATLAB plot to visualize RF budget results across the bandwidth of the input signal.

In addition, with an `rfbudget` object you can:

- Use export methods to generate MATLAB scripts, RF Blockset models, or measurement testbenches in Simulink®.
- Use `show` command to copy an `rfbudget` object into the **RF Budget Analyzer** app.

Building Elements of RF Receiver

A basic RF receiver consists of an RF filter, an RF amplifier, a demodulator, an IF filter, and an IF amplifier.

First build and parameterize each of the 2-port RF elements. Then use `rfbudget` to cascade the elements with input frequency 2.1 GHz, input power -30 dBm, and input bandwidth 45 MHz.

```
f1 = nport('RFBudget_RF.s2p','RFBandpassFilter');

a1 = amplifier('Name','RFAmplifier', ...
    'Gain',11.53, ...
    'NF',1.53, ...
    'OIP3',35);

d = modulator('Name','Demodulator', ...
    'Gain',-6, ...
    'NF',4, ...
    'OIP3',50, ...
    'LO',2.03e9, ...
    'ConverterType','Down');
```

```
f2 = nport('RFBudget_IF.s2p','IFBandpassFilter');

a2 = amplifier('Name','IFAmplifier', ...
    'Gain',30, ...
    'NF',8, ...
    'OIP3',37);

b = rfbudget('Elements',[f1 a1 d f2 a2], ...
    'InputFrequency',2.1e9, ...
    'AvailableInputPower',-30, ...
    'SignalBandwidth',45e6);
```

Visualize RF Budget Results in MATLAB

Scalar frequency results can be viewed simply by using MATLAB `disp` to see the results at the Command Line. Each column of the budget shows the results of cascading only the elements of the previous columns. Note that final column shows the RF budget results of the entire cascade.

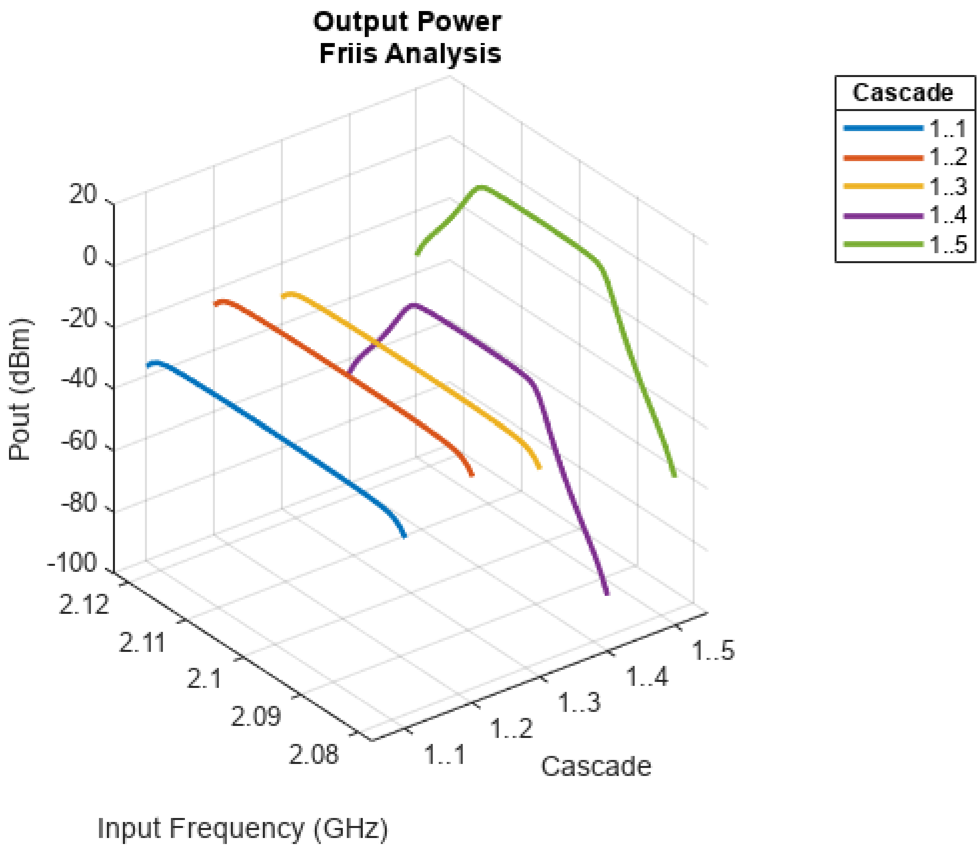
```
disp(b)
```

```
  rfbudget with properties:

               Elements: [1x5 rf.internal.rfbudget.Element]
         InputFrequency: 2.1 GHz
    AvailableInputPower: -30 dBm
        SignalBandwidth:  45 MHz
                 Solver: Friis
             AutoUpdate: true

   Analysis Results
        OutputFrequency: (GHz) [   2.1    2.1   0.07    0.07   0.07]
            OutputPower: (dBm) [-31.53    -20    -26  -27.15  2.847]
         TransducerGain: (dB)  [-1.534  9.996  3.996   2.847  32.85]
                     NF: (dB)  [ 1.533  3.064  3.377   3.611  7.036]
                   IIP2: (dBm) []
                   OIP2: (dBm) []
                   IIP3: (dBm) [   Inf     25  24.97   24.97  4.116]
                   OIP3: (dBm) [   Inf     35  28.97   27.82  36.96]
                    SNR: (dB)  [ 65.91  64.38  64.07   63.83  60.41]
```

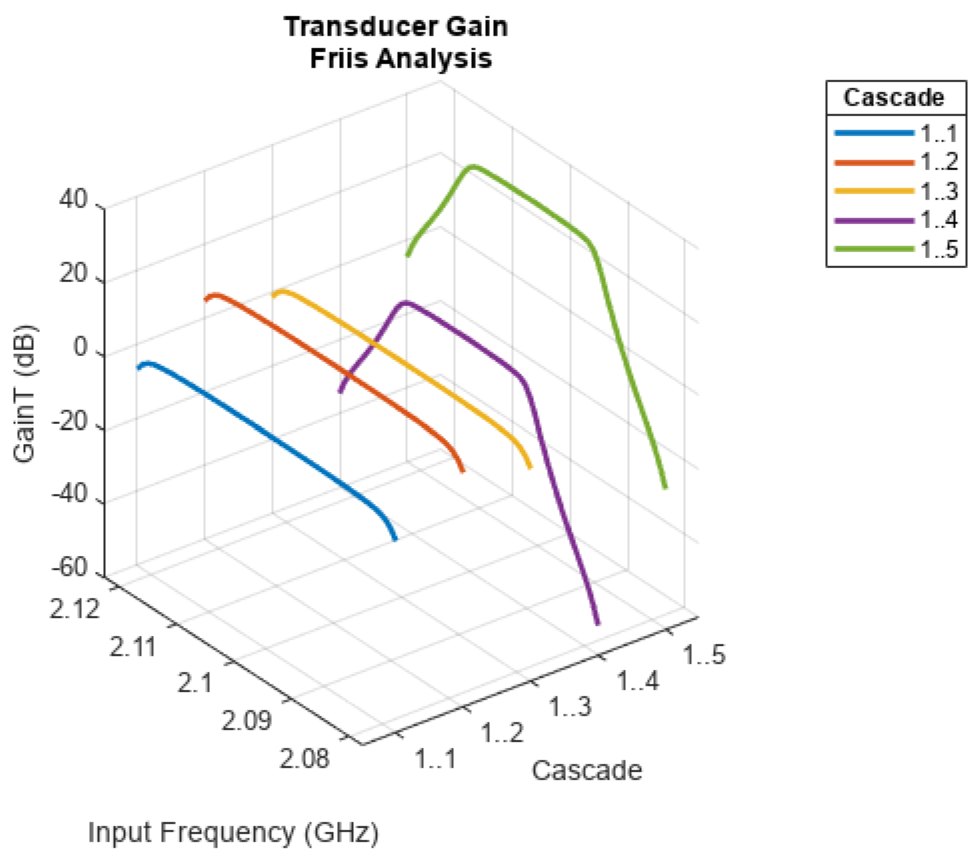
Plot RF Budget Results Versus Input Frequency

Use the budget's `rfplot` function to produce report-ready plots of cumulative RF budget results versus a range of cascade input frequencies. Cumulative (i.e. terminated sub-cascade) results are automatically computed to show the variation of the RF budget result through the entire design. Use the data cursor of the figure window to interactively explore values at different frequencies at different stages.

```
rfplot(b,'Pout')
```



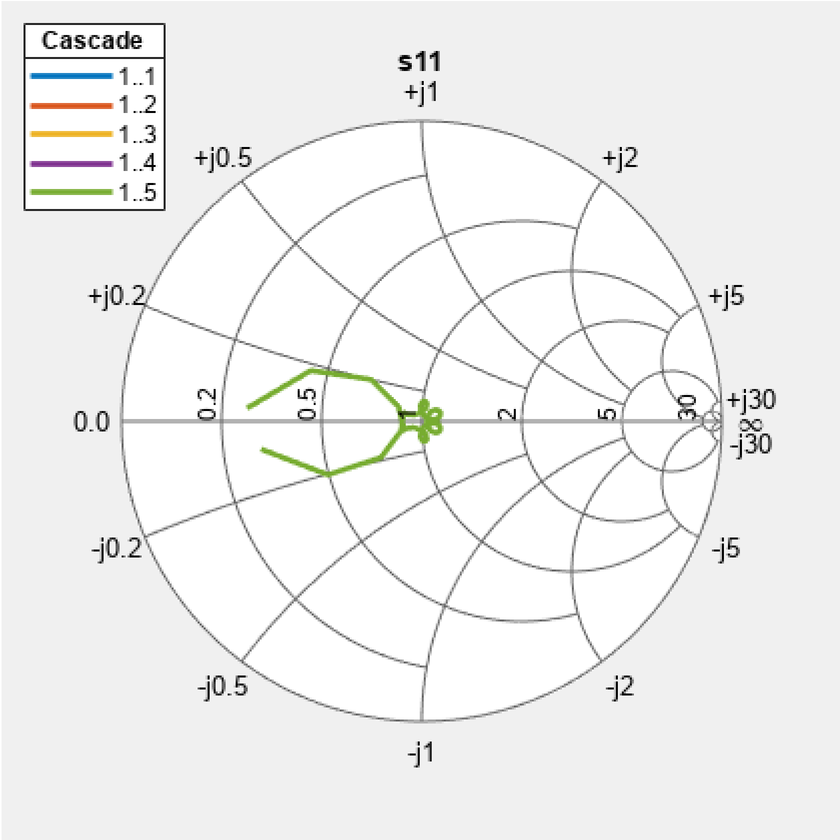

```
rfplot(b,'GainT')
```



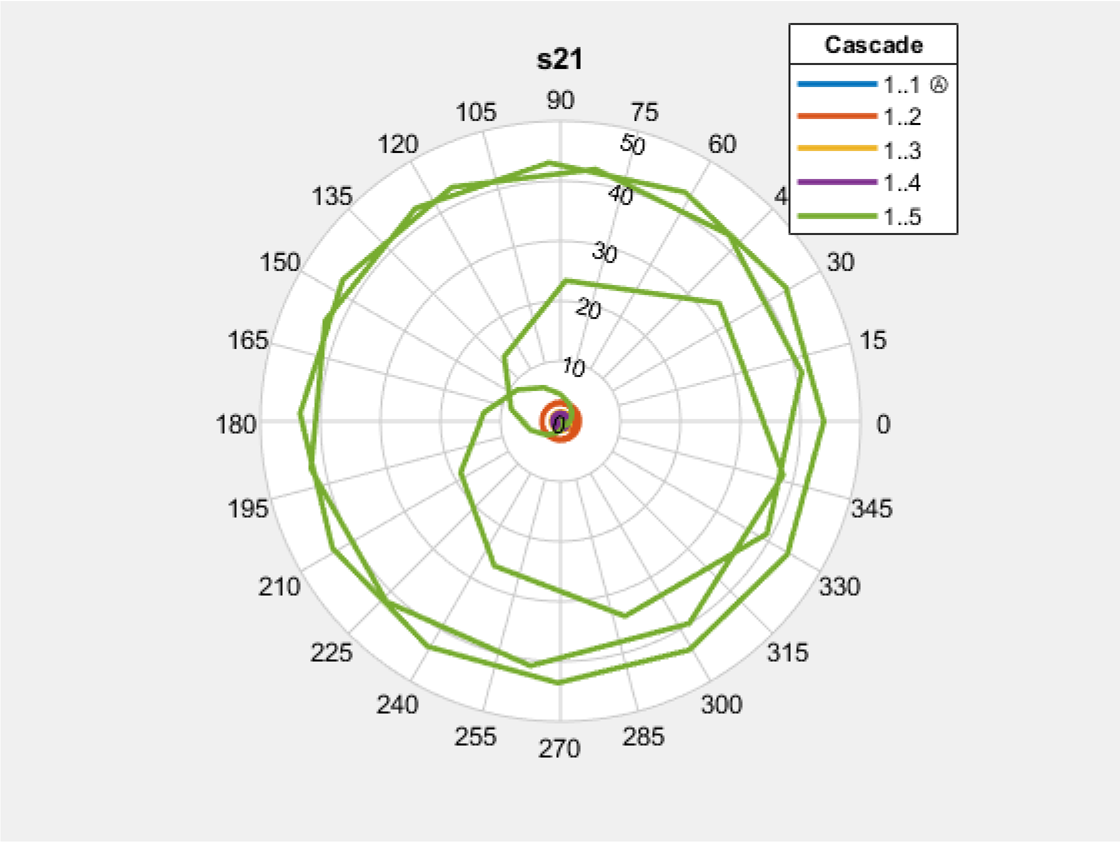
Plot RF Budget Network Parameter Results Versus Input Frequency

Use the RF budget `smithplot/polar` function to produce plots of cumulative RF budget sparameter results versus a range of cascade input frequencies. Use `smithplot` function to view reflection coefficients and polar to view reflection and transmission coefficients.

```
smithplot(b,1,1)
```


Cascade
1..1
1..2
1..3
1..4
1..5
s11
+j1
+j0.5
+j2
+j0.2
+j5
0.0
0.2
0.5
1
2
5
30
+j30
∞
-j30
-j0.2
-j5
-j0.5
-j2
-j1

```
polar(b,2,1)
```



Easily Export to RF Blockset and Simulink

The `rfbudget` object has other useful MATLAB methods:

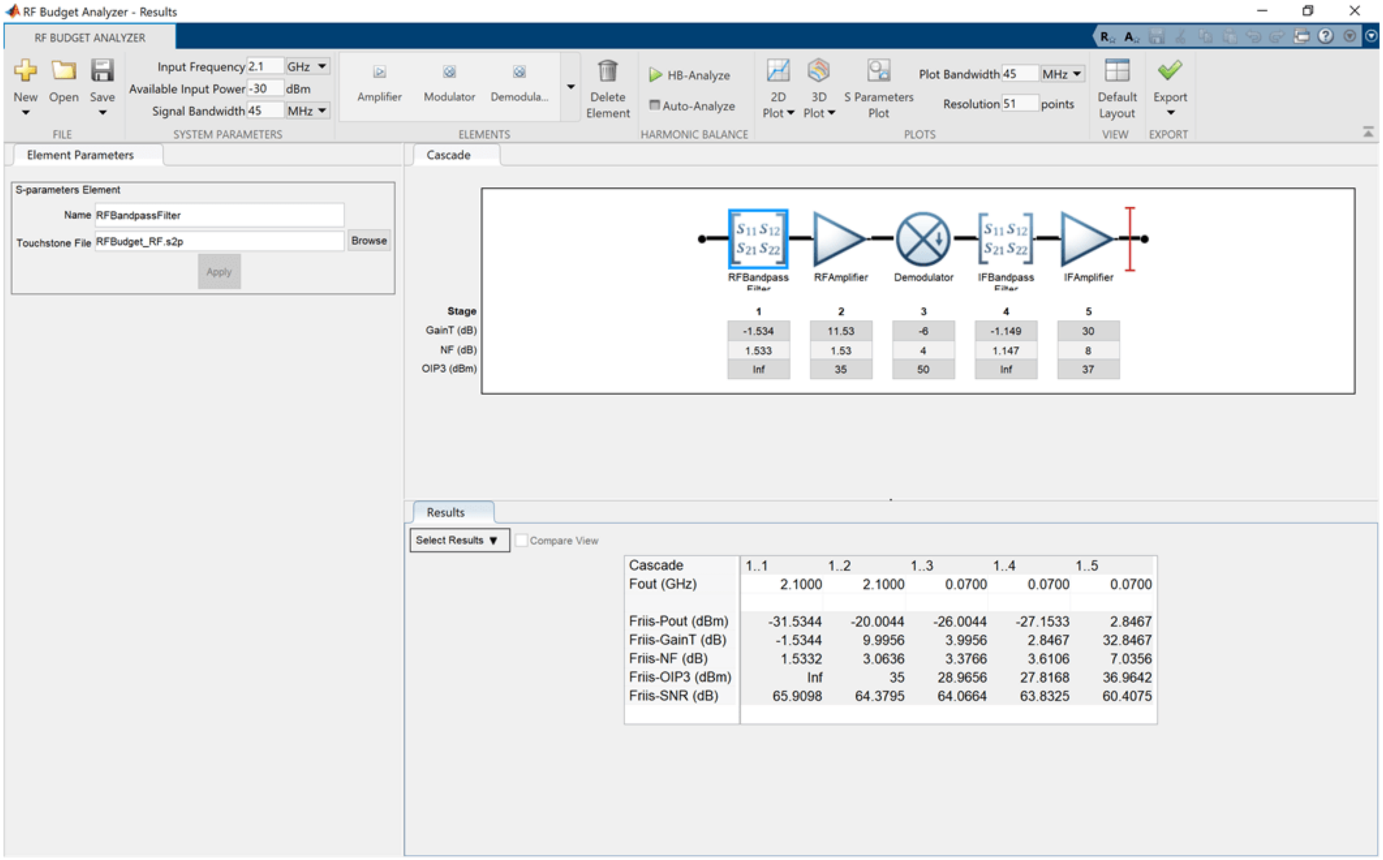
- `exportScript` - generate a MATLAB script that builds the current design
- `exportRFBlockset` - generate an RF Blockset model for simulation
- `exportTestbench` - generate a Simulink measurement testbench

Visualize RF Budget Results in App

Use the show command to copy a single-frequency `rfbudget` object into the **RF Budget Analyzer** app. The **Plot**, **Smith**, and **Polar** button in the app, with its pull-down options, calls `rfplot`, `smithplot`, and `polar` respectively.

In the app, the **Expor**t button copies the current design to an `rfbudget` object in the MATLAB workspace. All of the other export methods of the RF budget object are available through the pulldown options of the Export button.

```
show(b)
```



Automatically Create Reports From MATLAB Files

If you have written a `'myfile.m'` script that builds your design and visualizes it with `rfplot` commands, try the `publish('myfile.m')` function at the command line (or click the **Publish** button in the MATLAB editor). This automatically generates all figures and produces a report for your colleagues, saved as an html file.

To save your design, first undock using the commands shown below and then use the Figure Toolbar to pulldown the File Menu and save using **File** -> **Save As** and select the Save as type to png or pdf. To redock the figure window into the app you can click the Dock affordance on the upper right corner of the figure window.

```
h = findall(0,'type','figure','name','untitled');
set(h,'WindowStyle','normal')
set(h,'MenuBar','figure')
set(h,'ToolBar','auto')
```

See Also

More About

- "Superheterodyne Receiver Using RF Budget Analyzer App" on page 6-2
- "Bandpass Filter Response" on page 6-23

Bandpass Filter Response

This example shows how to compute the time-domain response of a simple bandpass filter. The eight steps involved in computing the time-domain response of a simple bandpass filter are,

1. Use the classic image parameter design to assign inductance and capacitance values to the bandpass filter.
2. Use the `circuit`, `capacitor`, and `inductor` objects with the `add` function to programmatically construct a Butterworth circuit.
3. Use the `setports` function to define the circuit as a 2-port network.
4. Use the `sparameters` function to extract the S-parameters of the 2-port network over a wide frequency range.
5. Use the `s2tf` function to compute the voltage transfer function from the input to the output.
6. Use the `rational` object to generate rational fits that capture the ideal RC circuit to a very high degree of accuracy.
7. Use the `randn` function to create noise in order to create a noisy input voltage waveform.
8. Use the `timeresp function` to compute the transient response to a noisy input voltage waveform.

Design Bandpass Filter Using Image Parameters

The image parameter design is a framework for analytically computing the values of the series and parallel components in the passive filters. For more information on Image Parameters, see "Complete Wireless Design" by Cotter W. Sayre, McGraw-Hill 2008 p. 331.

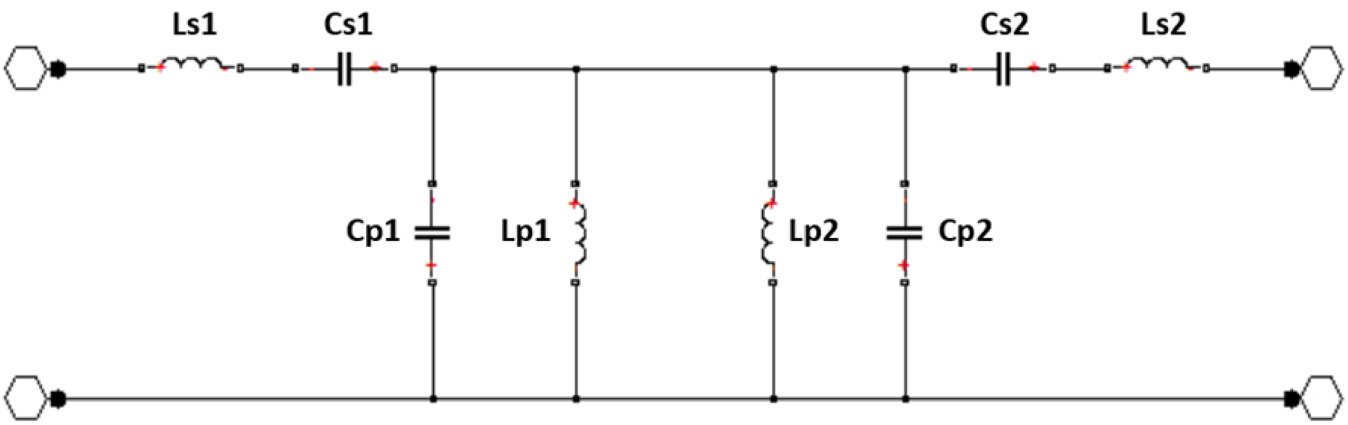


Figure 1: A Butterworth bandpass filter built out of two half-sections.

Generate the component values for a bandpass filter with a lower 3 dB cutoff frequency of 2.4 GHz and an upper 3 dB cutoff frequency of 2.5 GHz.

```
Ro = 50;
f1C = 2400e6;
f2C = 2500e6;
Ls = (Ro / (pi*(f2C - f1C)))/2;        % Ls1 and Ls2
Cs = 2*(f2C - f1C)/(4*pi*Ro*f2C*f1C);  % Cs1 and Cs2
```

```
Lp = 2*Ro*(f2C - f1C)/(4*pi*f2C*f1C);   % Lp1 and Lp2
Cp = (1/(pi*Ro*(f2C - f1C)))/2;         % Cp1 and Cp2
```

Programmatically Construct Circuit

Before building the circuit using the `inductor` and `capacitor` objects, nodes in the circuit are numbered. This is shown in figure 1.

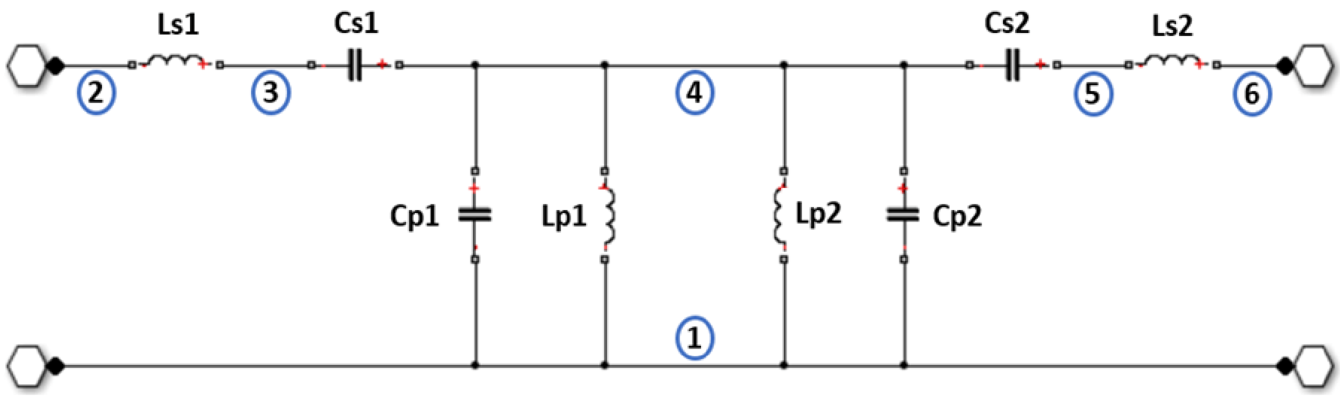


Figure 2: Node numbers added to the Butterworth bandpass filter.

Create a `circuit` object and populate it with the `inductor` and the `capacitor` objects using the `add` function.

```
ckt = circuit('butterworthBPF');

add(ckt,[3 2],inductor(Ls));      % Ls1
add(ckt,[4 3],capacitor(Cs));     % Cs1
add(ckt,[5 4],capacitor(Cs));     % Cs2
add(ckt,[6 5],inductor(Ls));      % Ls2

add(ckt,[4 1],capacitor(Cp));     % Cp1
add(ckt,[4 1],inductor(Lp));      % Lp1
add(ckt,[4 1],inductor(Lp));      % Lp2
add(ckt,[4 1],capacitor(Cp));     % Cp2
```

Extract S-Parameters From 2-Port Network

To extract S-parameters from the circuit object, first use the `setports` function to define the circuit as a 2-port network.

```
freq = linspace(2e9,3e9,101);
```

Use the `sparameters` function to extract the S-parameters at the frequencies of interest.

```
setports(ckt,[2 1],[6 1])
S = sparameters(ckt,freq);
```

Fit Transfer Function of Circuit to Rational Function

Use the `s2tf` function to generate a transfer function from the S-parameter object.

```
tfS = s2tf(S);
```

Use the `rational` object to fit the transfer function data to a rational function.

```
fit = rational(freq,tfS);
```

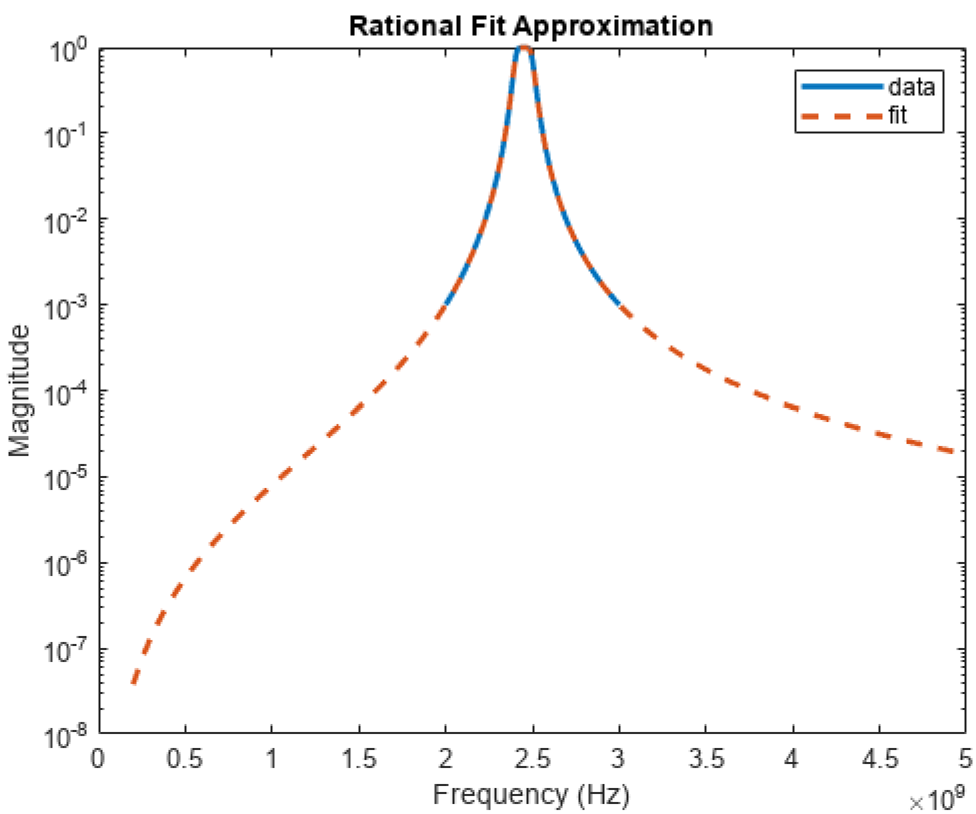
Verify Rational Fit Approximation

Use the `freqresp` function to verify that the rational fit approximation has reasonable behavior outside both sides of the fitted frequency range.

```
widerFreqs = linspace(2e8,5e9,1001);
resp = freqresp(fit,widerFreqs);
```

Plot to visualize rational fit approximation. The rational fit behaves well outside the fitted frequency range.

```
figure
semilogy(freq,abs(tfS),widerFreqs,abs(resp),'--','LineWidth',2)
xlabel('Frequency (Hz)');
ylabel('Magnitude');
legend('data','fit');
title('Rational Fit Approximation');
```



Construct Input Signal to Test Bandpass Filter

To test the bandpass filter, designed by the Image Parameter technique, a sinusoidal signal at 2.45 GHz is recovered from the noisy input signal. The noise input signal is generated by the inclusion of zero-mean random noise and a blocker at 2.35 GHz to the input signal.

Construct a input and a noisy input signal with 8192 samples.

```
fCenter = 2.45e9;
fBlocker = 2.35e9;
period = 1/fCenter;
sampleTime = period/16;
signalLen = 8192;
t = (0:signalLen-1)'*sampleTime; % 256 periods
input = sin(2*pi*fCenter*t);     % Clean input signal
rng('default')
noise = randn(size(t)) + sin(2*pi*fBlocker*t);
noisyInput = input + noise;      % Noisy input signal
```

Compute Transient Response to Input Signal

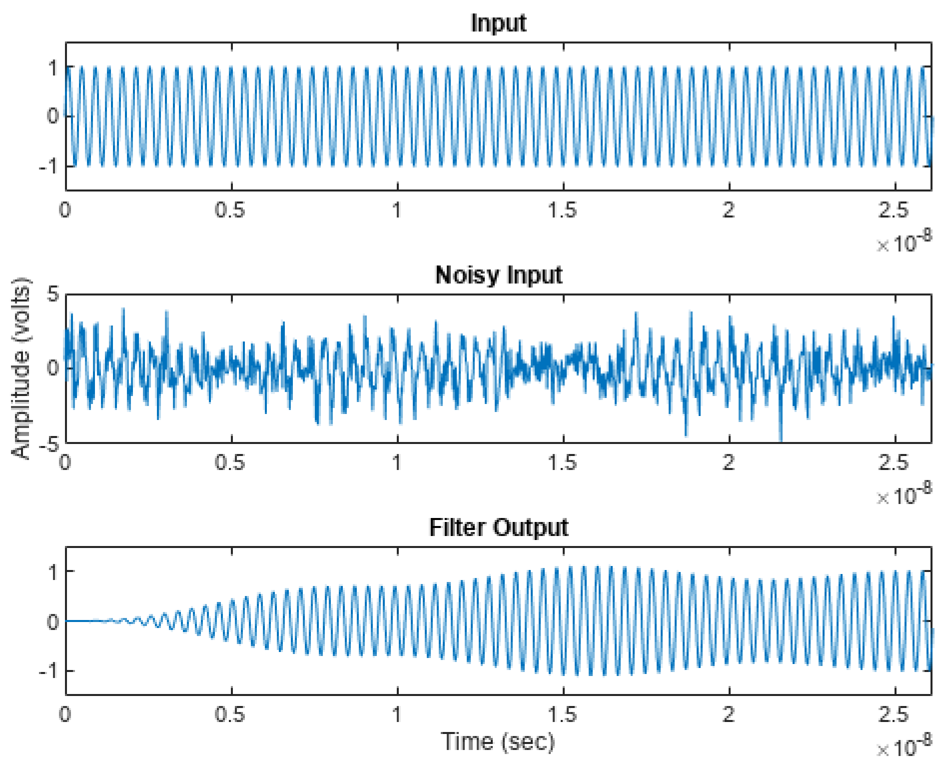
Use the `timeresp` function to compute the analytic solutions of the state-space.

```
output = timeresp(fit,noisyInput,sampleTime);
```

View Input Signal and Filter Response in Time Domain

Plot the input signal, noisy input signal, and the band pass filter output in a figure window.

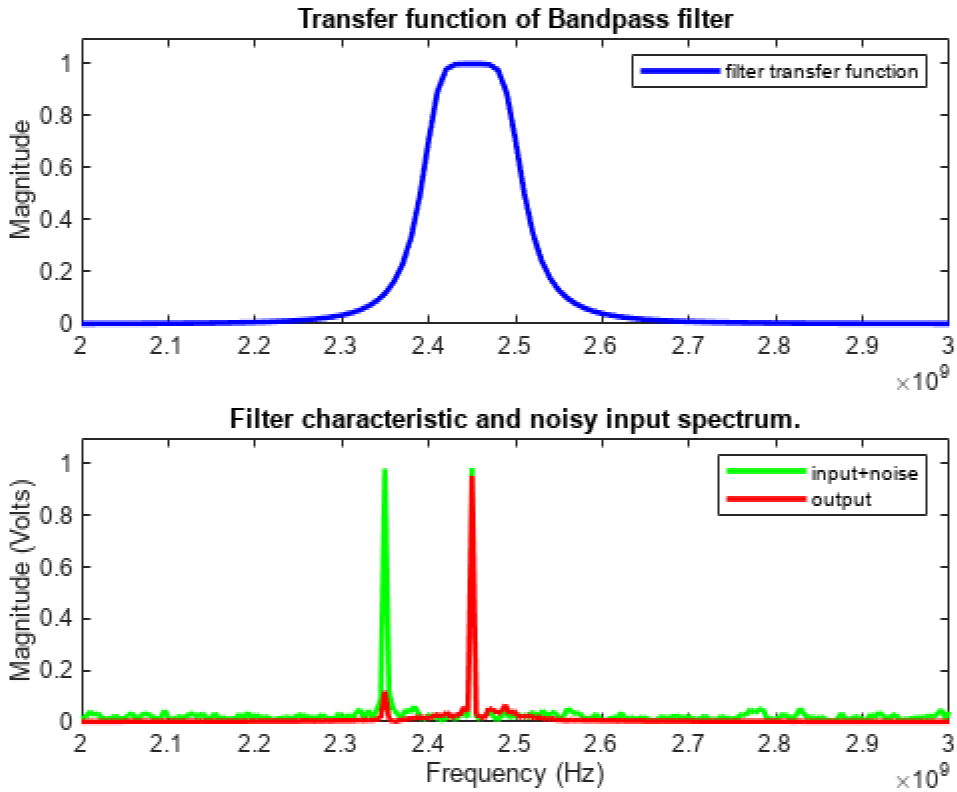
```
xmax = t(end)/8;
figure
subplot(3,1,1)
plot(t,input)
axis([0 xmax -1.5 1.5])
title('Input')

subplot(3,1,2)
plot(t,noisyInput)
axis([0 xmax floor(min(noisyInput)) ceil(max(noisyInput))]);
title('Noisy Input');
ylabel('Amplitude (volts)');

subplot(3,1,3)
plot(t,output)
axis([0 xmax -1.5 1.5]);
title('Filter Output');
xlabel('Time (sec)');
```



View Input Signal and Filter Response in Frequency Domain

Overlaying the noisy input and the filter response in the frequency domain explains why the filtering operation is successful. Both the blocker signal at 2.35 GHz and much of the noise are significantly attenuated.

```
NFFT = 2^nextpow2(signalLen); % Next power of 2 from length of y
Y = fft(noisyInput,NFFT)/signalLen;
samplingFreq = 1/sampleTime;
f = samplingFreq/2*linspace(0,1,NFFT/2+1)';
O = fft(output,NFFT)/signalLen;

figure
subplot(2,1,1)
plot(freq,abs(tfS),'b','LineWidth',2)
axis([freq(1) freq(end) 0 1.1]);
legend('filter transfer function');
title('Transfer function of Bandpass filter');
ylabel('Magnitude');

subplot(2,1,2)
plot(f,2*abs(Y(1:NFFT/2+1)),'g',f,2*abs(O(1:NFFT/2+1)),'r','LineWidth',2)
axis([freq(1) freq(end) 0 1.1]);
legend('input+noise','output');
title('Filter characteristic and noisy input spectrum.');
xlabel('Frequency (Hz)');
ylabel('Magnitude (Volts)');
```



To compute and display this bandpass filter response using RFCKT objects, see “Bandpass Filter Response Using RFCKT Objects” on page 6-35.

See Also

More About

- “Design IF Butterworth Bandpass Filter” on page 6-183
- “Superheterodyne Receiver Using RF Budget Analyzer App” on page 6-2
- “Visualizing RF Budget Analysis over Bandwidth” on page 6-16

MOS Interconnect and Crosstalk

This example shows how to build and simulate an RC tree circuit using the RF Toolbox™.

In "Asymptotic Waveform Evaluation for Timing Analysis" (IEEE Transactions on Computer-Aided Design, Vol., 9, No. 4, April 1990), Pillage and Rohrer presented and simulated an RC tree circuit that models signal integrity and crosstalk in low- to mid-frequency MOS circuit interconnect. This example confirms their simulations using RF Toolbox software.

Their circuit, reproduced in the following figure, consists of 11 resistors and 12 capacitors. In the paper, Pillage and Rohrer:

- Apply a ramp voltage input
- Compute transient responses
- Plot the output voltages across two different capacitors, `C7` and `C12`.

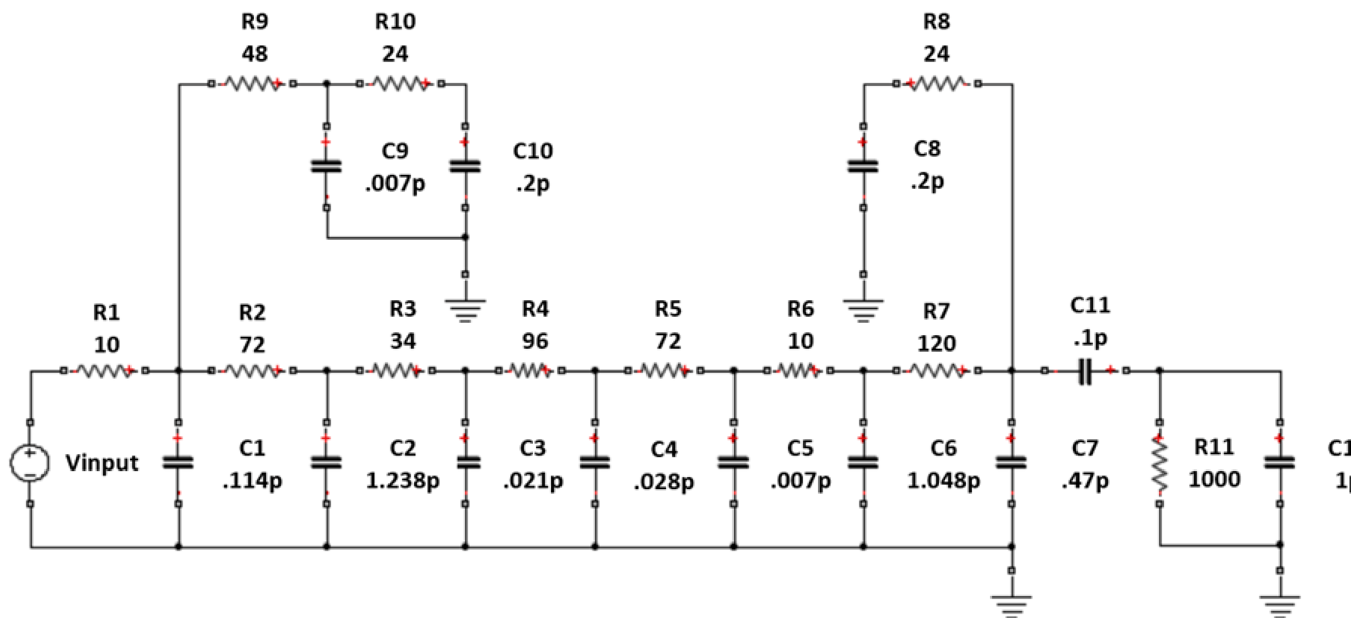


Figure 1: An RC tree model of MOS interconnect with crosstalk.

With RF Toolbox, you can programmatically construct this circuit in MATLAB and perform signal integrity simulations.

This example shows:

1. How to use `circuit`, `resistor`, and `capacitor` with the `add` function to programmatically construct the circuit.
2. How to use `clone`, `setports`, and `sparameters` objects to calculate S-parameters for each desired output over a wide frequency range.
3. How to use `s2tf` with `Zsource = 0` and `Zload = Inf` to compute the voltage transfer function from input to each desired output.

4 How to use `rationalfit` function to produce rational-function approximations that capture the ideal RC-circuit behavior to a very high degree of accuracy.

5 How to use `timeresp` function to compute the transient response to the input voltage waveform.

Insert Node Numbers Into Circuit Diagram

Before building the circuit using `resistor` and `capacitor` objects, we must number the nodes of the circuit shown in figure 1.

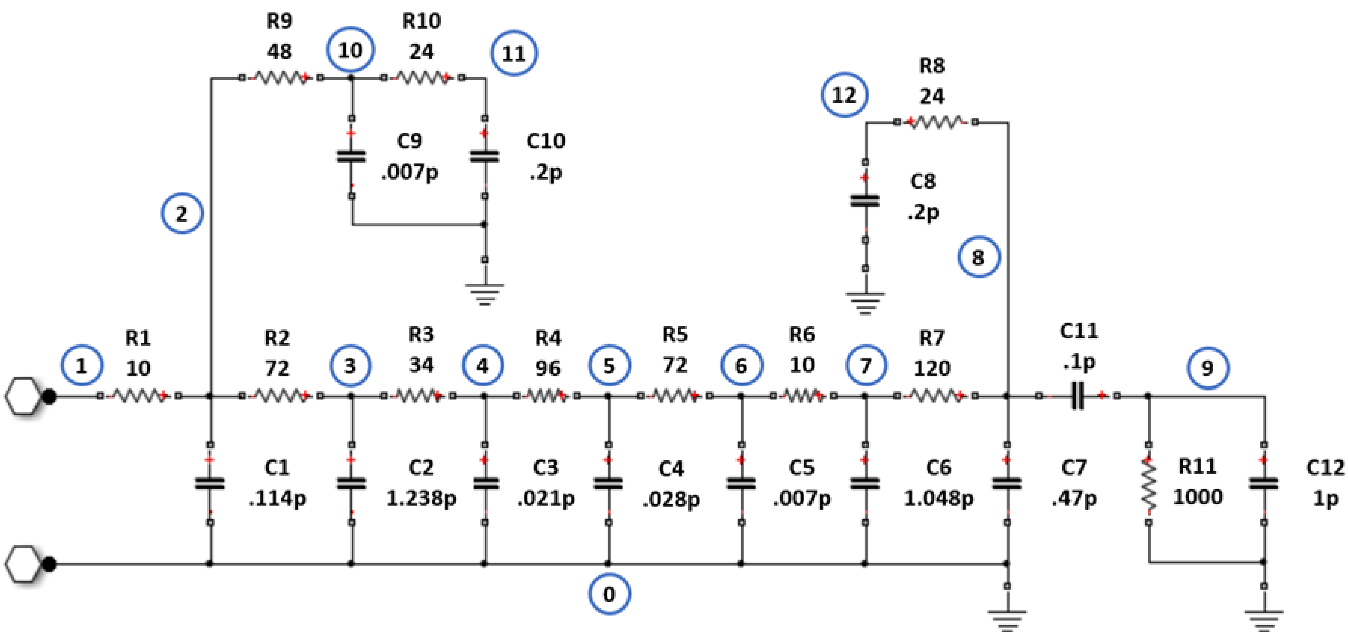


Figure 2: The circuit drawn with node numbers

Programmatically Construct Circuit

Create a `circuit` and use the `add` function to populate the circuit with named `resistor` and `capacitor` objects.

```
ckt = circuit('crosstalk');

add(ckt,[2 1],resistor(10,'R1'))
add(ckt,[2 0],capacitor(0.114e-12,'C1'))
add(ckt,[3 2],resistor(72,'R2'))
add(ckt,[3 0],capacitor(1.238e-12,'C2'))
add(ckt,[4 3],resistor(34,'R3'))
add(ckt,[4 0],capacitor(0.021e-12,'C3'))
add(ckt,[5 4],resistor(96,'R4'))
add(ckt,[5 0],capacitor(0.028e-12,'C4'))
add(ckt,[6 5],resistor(72,'R5'))
add(ckt,[6 0],capacitor(0.007e-12,'C5'))
add(ckt,[7 6],resistor(10,'R6'))
add(ckt,[7 0],capacitor(1.048e-12,'C6'))
add(ckt,[8 7],resistor(120,'R7'))
add(ckt,[8 0],capacitor(0.47e-12,'C7'))
```

```
add(ckt,[12 8],resistor(24,'R8'))
add(ckt,[12 0],capacitor(0.2e-12,'C8'))

add(ckt,[10 2],resistor(48,'R9'))
add(ckt,[10 0],capacitor(0.007e-12,'C9'))
add(ckt,[11 10],resistor(24,'R10'))
add(ckt,[11 0],capacitor(0.2e-12,'C10'))

add(ckt,[9 8],capacitor(0.1e-12,'C11'))
add(ckt,[9 0],resistor(1000,'R11'))
add(ckt,[9 0],capacitor(1e-12,'C12'))
```

Simulation Setup

The input signal used by Pillage and Rohrer is a voltage ramp from 0 to 5 volts with a rise time of one nanosecond and a duration of ten nanoseconds. The following MATLAB code models this signal with 1000 timepoints with a `sampleTime` of 0.01 nanoseconds.

The following MATLAB code also uses the `logspace` function to generate a vector of 101 logarithmically spaced analysis frequencies between 1 Hz and 100 GHz. Specifying a wide set of frequency points improves simulation accuracy.

```
sampleTime = 1e-11;
t = (0:1000)'*sampleTime;
input = [(0:100)'*(5/100); (101:1000)'*0+5];
freq = logspace(0,11,101)';
```

Calculate S-Parameters For Each 2-Port Network

To calculate the response across both the `C7` and `C12` capacitors, two separate S-parameter calculations must be made: first, assuming the `C7` capacitor represents the output port, and second, assuming the `C12` capacitor represents the output port. To calculate the S-parameters for each setup:

1. Copy the original circuit `ckt` using the `clone` function.
2. Define the input and output ports of the network using the `setports` function.
3. Calculate the S-parameters using the `sparameters` object.

Calculate S-parameters with `C7` capacitor represents the output port.

```
cktC7 = clone(ckt);
setports(cktC7,[1 0],[8 0])
S_C7 = sparameters(cktC7,freq);
```

Calculate S-parameters with `C12` capacitor represents the output port.

```
cktC12 = clone(ckt);
setports(cktC12,[1 0],[9 0])
S_C12 = sparameters(cktC12,freq);
```

Simulate Each 2-Port Network

To simulate each network:

1. The `s2tf` function, with `option = 2`, computes the gain from the source voltage to the output voltage. It allows arbitrary source and load impedances, in this case `Zsource = 0` and `Zload = Inf`. The resulting transfer functions `tfC7` and `tfC12` are frequency-dependent data vectors that can be fit with rational-function approximation.

2. The `rationalfit` function generates high-accuracy rational-function approximations. The resulting approximations match the networks to machine accuracy.
3. The `timeresp` function computes the analytic solution to the state-space equations defined by a rational-function approximation. This methodology is fast enough to enable one to push a million bits through a channel.

Simulate `cktC7` circuit.

```
tfC7 = s2tf(S_C7,0,Inf,2);
fitC7 = rationalfit(freq,tfC7);
outputC7 = timeresp(fitC7,input,sampleTime);
```

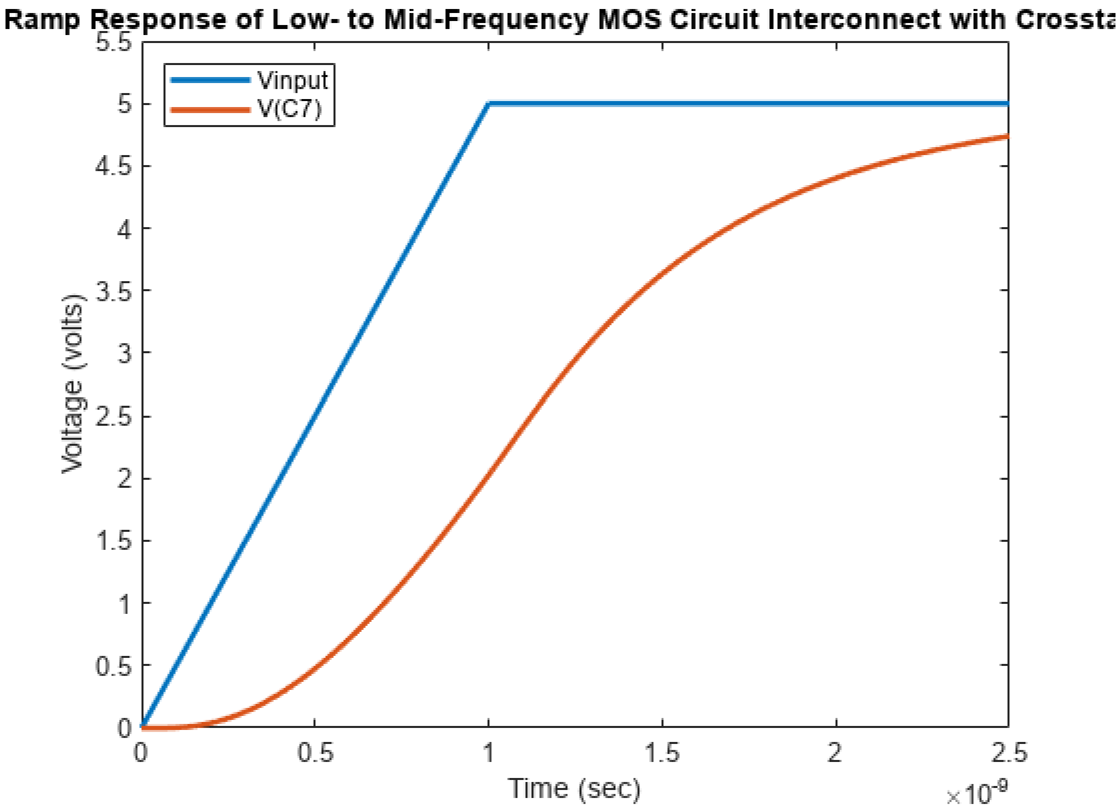
Simulate `cktC12` circuit.

```
tfC12 = s2tf(S_C12,0,Inf,2);
fitC12 = rationalfit(freq,tfC12);
outputC12 = timeresp(fitC12,input,sampleTime);
```

Plot Transient Responses

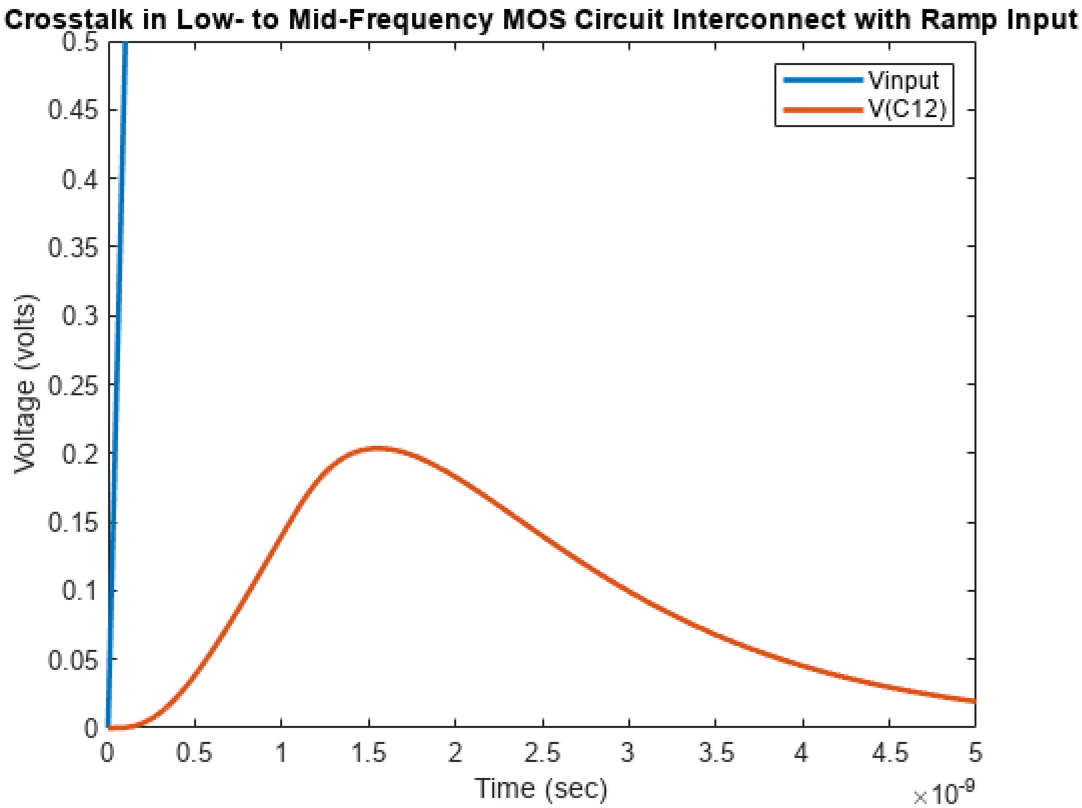
The outputs match Figures 23 and 24 of the Pillage and Rohrer paper. Plot ramp response of low- to mid-frequency MOS circuit interconnect with crosstalk.

```
figure
plot(t,input,t,outputC7,'LineWidth',2)
axis([0 2.5e-9 0 5.5]);
title('Ramp Response of Low- to Mid-Frequency MOS Circuit Interconnect with Crosstalk');
xlabel('Time (sec)');
ylabel('Voltage (volts)');
legend('Vinput','V(C7)','Location','NorthWest');
```



Plot crosstalk in low- to mid-frequency MOS circuit interconnect with ramp input.

```
figure
plot(t,input,t,outputC12,'LineWidth',2)
axis([0 5e-9 0 .5])
title('Crosstalk in Low- to Mid-Frequency MOS Circuit Interconnect with Ramp Input')
xlabel('Time (sec)')
ylabel('Voltage (volts)')
legend('Vinput','V(C12)','Location','NorthEast')
```



Verify Rational Fit Outside Fit Range

Though not shown in this example, you can also use the `freqresp` function to check the behavior of `rationalfit` function well outside the specified frequency range. The fit outside the specified range can sometimes cause surprising behavior, especially if frequency data near 0 Hz (DC) is not provided.

To perform this check for the rational-function approximation in this example, uncomment and run the following MATLAB code.

```
% widerFreqs = logspace(0,12,1001);
% respC7 = freqresp(fitC7,widerFreqs);
% figure
% loglog(freq,abs(tfC7),'+',widerFreqs,abs(respC7))
% respC12 = freqresp(fitC12,widerFreqs);
% figure
% loglog(freq,abs(tfC12),'+',widerFreqs,abs(respC12))
```

For example on how to build and simulate this RC tree circuit using RFCKT objects, see “MOS Interconnect and Crosstalk Using RFCKT Objects” on page 6-41.

See Also

More About

- “MOS Interconnect and Crosstalk Using RFCKT Objects” on page 6-41

Bandpass Filter Response Using RFCKT Objects

This example shows how to compute the time-domain response of a simple bandpass filter:

1. Choose inductance and capacitance values using the classic image parameter design method.
2. Use `rfckt.seriesrlc`, `rfckt.shuntrlc`, and `rfckt.cascade` objects to programmatically construct a Butterworth circuit as a 2-port network.
3. Use `analyze` to extract the S-parameters of the 2-port network over a wide frequency range.
4. Use `s2tf` function to compute the voltage transfer function from the input to the output.
5. Use `rationalfit` function to generate rational fits that capture the ideal RC circuit to a very high degree of accuracy.
6. Create a noisy input voltage waveform.
7. Use `timeresp` function to compute the transient response to a noisy input voltage waveform.

Design Bandpass Filter by Image Parameters

The image parameter design method is a framework for analytically computing the values of the series and parallel components in passive filters. For more information on this method, see "Complete Wireless Design" by Cotter W. Sayre, McGraw-Hill 2008 p. 331.

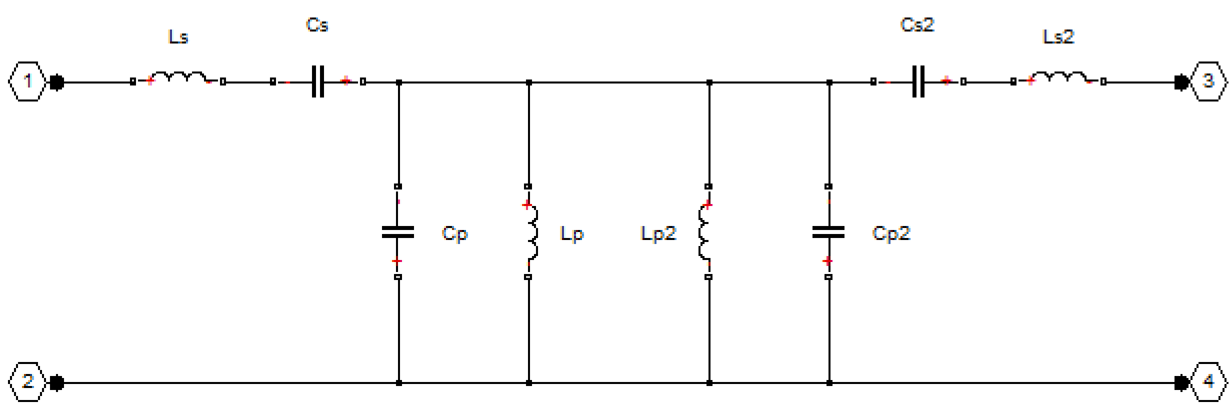


Figure 1: A Butterworth bandpass filter built out of two half-sections.

The following MATLAB® code generates component values for a bandpass filter with a lower 3-dB cutoff frequency of 2.4 GHz and an upper 3 dB cutoff frequency of 2.5 GHz.

```
Ro = 50;
f1C = 2400e6;
f2C = 2500e6;

Ls = (Ro / (pi*(f2C - f1C)))/2;
Cs = 2*(f2C - f1C)/(4*pi*Ro*f2C*f1C);

Lp = 2*Ro*(f2C - f1C)/(4*pi*f2C*f1C);
Cp = (1/(pi*Ro*(f2C - f1C)))/2;
```

Programmatically Construct Circuit as 2-Port Network

The L and C building blocks are formed by selecting appropriate values with the `rfckt.shuntrlc` object shown in Figure 2 or the `rfckt.seriesrlc` object shown in Figure 3. The building blocks are then connected together with `rfckt.cascade` as shown in Figure 4.

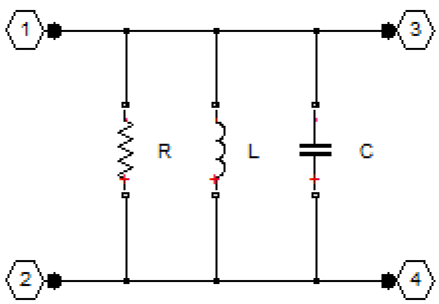


Figure 2: The 2-port network created by the `rfckt.shuntrlc` object

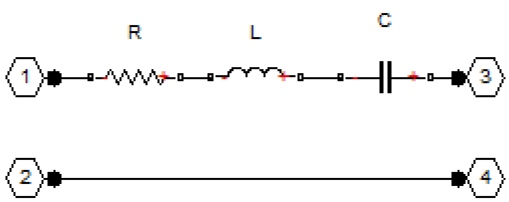


Figure 3: The 2-port network created by the `rfckt.seriesrlc` object

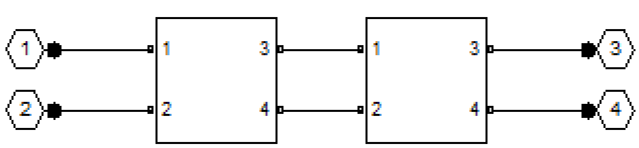


Figure 4: Connecting 2-port networks with the `rfckt.cascade` object

```
Seg1 = rfckt.seriesrlc('L',Ls,'C',Cs);
Seg2 = rfckt.shuntrlc('L',Lp,'C',Cp);
Seg3 = rfckt.shuntrlc('L',Lp,'C',Cp);
Seg4 = rfckt.seriesrlc('L',Ls,'C',Cs);

cktBPF = rfckt.cascade('Ckts',{Seg1,Seg2,Seg3,Seg4});
```

Extract S-Parameters From 2-Port Network

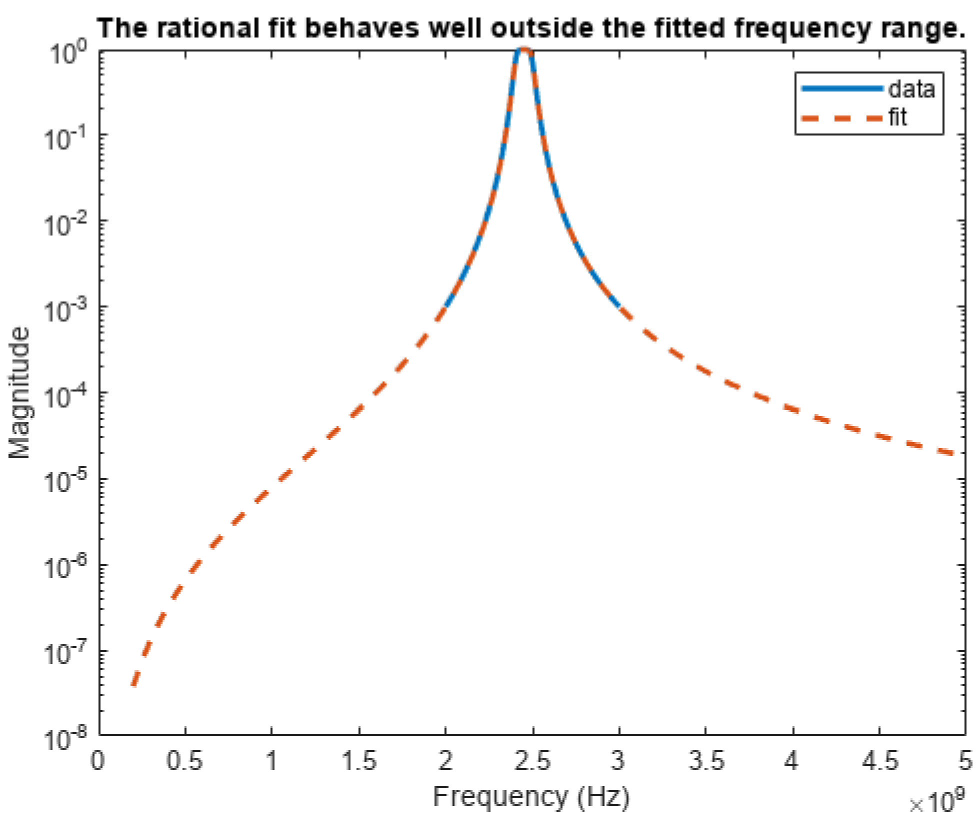
The `analyze` function extracts the S-parameters from a circuit over a specified vector of frequencies. This example provides a set of frequencies that spans the passband of the filter and analyzes with the default 50-Ohm reference, source impedance, and load impedances. Next, the `s2tf` function computes the voltage transfer function across the S-parameter model of the circuit. Finally, we generate a high-accuracy rational approximation using the `rationalfit` function. The resulting approximation matches the network to machine accuracy.

```
freq = linspace(2e9,3e9,101);
analyze(cktBPF,freq);
sparams = cktBPF.AnalyzedResult.S_Parameters;
tf = s2tf(sparams);
fit = rationalfit(freq,tf);
```

Verify that Rational Fit Tends to Zero

Use the `freqresp` function to verify that the rational fit approximation has reasonable behavior outside both sides of the fitted frequency range.

```
widerFreqs = linspace(2e8,5e9,1001);
resp = freqresp(fit,widerFreqs);

figure
semilogy(freq,abs(tf),widerFreqs,abs(resp),'--','LineWidth',2)
xlabel('Frequency (Hz)')
ylabel('Magnitude')
legend('data','fit')
title('The rational fit behaves well outside the fitted frequency range.')
```



Construct Input Signal to Test Band Pass Filter

This bandpass filter should be able to recover a sinusoidal signal at 2.45 GHz that is made noisy by the inclusion of zero-mean random noise and a blocker at 2.35 GHz. The following MATLAB code constructs such a signal from 4096 samples.

```
fCenter = 2.45e9;
fBlocker = 2.35e9;
period = 1/fCenter;
sampleTime = period/16;
signalLen = 8192;
t = (0:signalLen-1)'*sampleTime; % 256 periods

input = sin(2*pi*fCenter*t);     % Clean input signal
rng('default')
noise = randn(size(t)) + sin(2*pi*fBlocker*t);
noisyInput = input + noise;      % Noisy input signal
```

Compute Transient Response to Input Signal

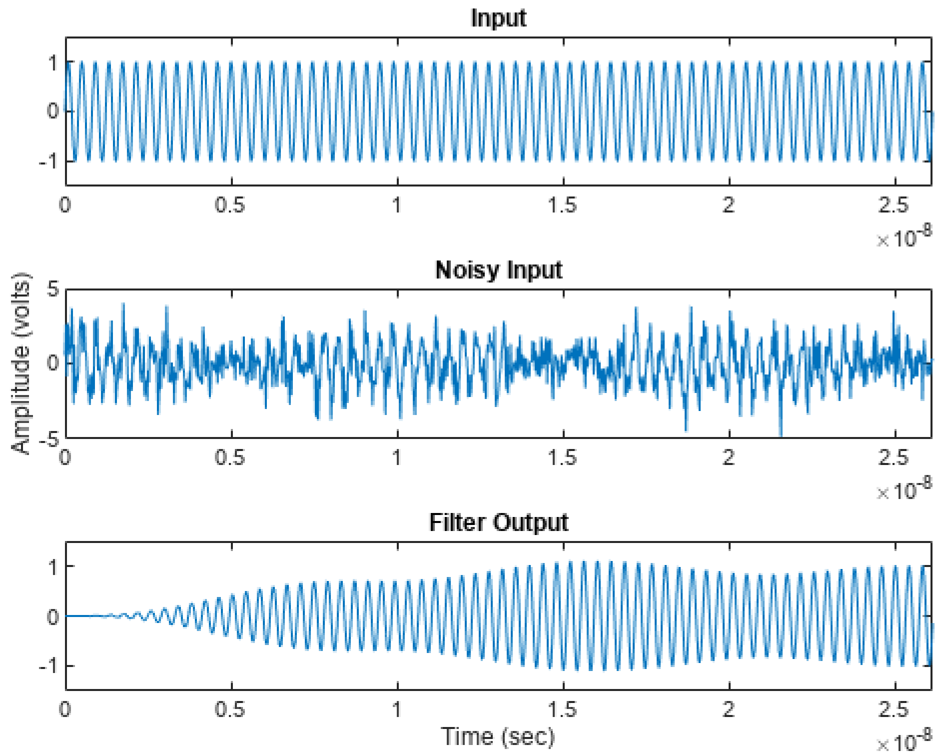
The `timeresp` function computes the analytic solution to the state-space equations defined by the rational fit and the input signal.

```
output = timeresp(fit,noisyInput,sampleTime);
```

View Input Signal and Filter Response in Time Domain

Plot the input signal, noisy input signal, and the band pass filter output in a figure window.

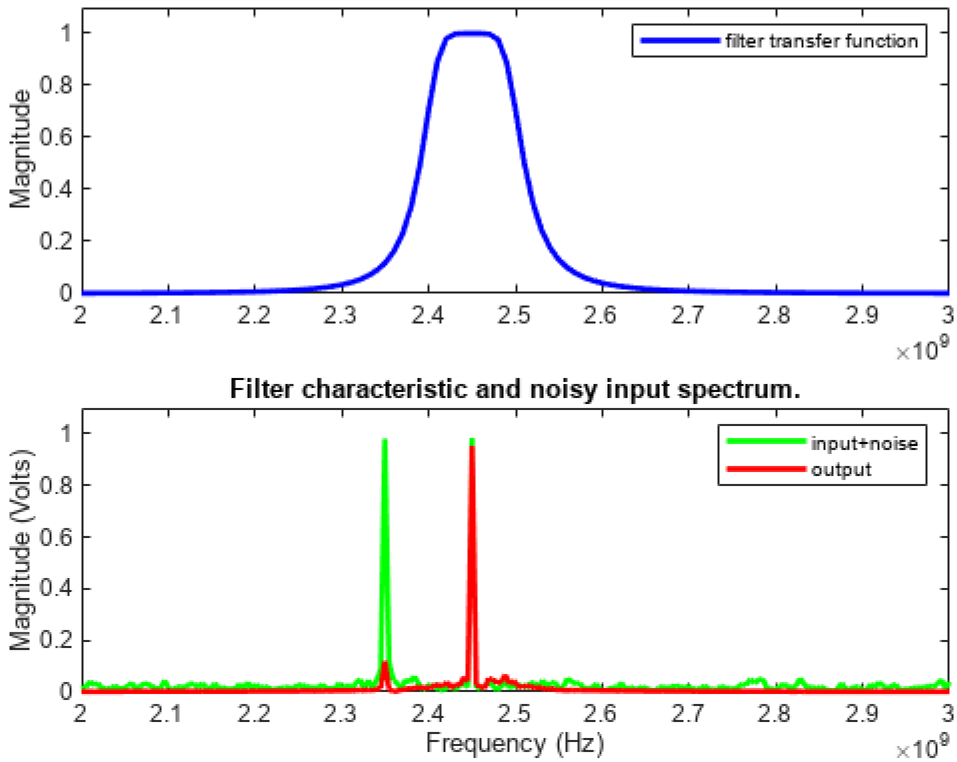
```
xmax = t(end)/8;
figure
subplot(3,1,1)
plot(t,input)
axis([0 xmax -1.5 1.5])
title('Input')

subplot(3,1,2)
plot(t,noisyInput)
axis([0 xmax floor(min(noisyInput)) ceil(max(noisyInput))])
title('Noisy Input')
ylabel('Amplitude (volts)')

subplot(3,1,3)
plot(t,output)
axis([0 xmax -1.5 1.5])
title('Filter Output')
xlabel('Time (sec)')
```



View Input Signal and Filter Response in Frequency Domain

Overlaying the noisy input and the filter response in the frequency domain explains why the filtering operation is successful. Both the blocker signal at 2.35 GHz and much of the noise is significantly attenuated.

```
NFFT = 2^nextpow2(signalLen); % Next power of 2 from length of y
Y = fft(noisyInput,NFFT)/signalLen;
samplingFreq = 1/sampleTime;
f = samplingFreq/2*linspace(0,1,NFFT/2+1)';
O = fft(output,NFFT)/signalLen;

figure
subplot(2,1,1)
plot(freq,abs(tf),'b','LineWidth',2)
axis([freq(1) freq(end) 0 1.1])
legend('filter transfer function')
ylabel('Magnitude')

subplot(2,1,2)
plot(f,2*abs(Y(1:NFFT/2+1)),'g',f,2*abs(O(1:NFFT/2+1)),'r','LineWidth',2)
axis([freq(1) freq(end) 0 1.1])
legend('input+noise','output')
title('Filter characteristic and noisy input spectrum.')
xlabel('Frequency (Hz)')
ylabel('Magnitude (Volts)')
```



See Also

More About

- "Bandpass Filter Response" on page 6-23
- "Operations with RF Circuit Objects" on page 6-173

MOS Interconnect and Crosstalk Using RFCKT Objects

This example shows how to build and simulate an RC tree circuit using the RF Toolbox™.

In "Asymptotic Waveform Evaluation for Timing Analysis" (IEEE Transactions on Computer-Aided Design, Vol., 9, No. 4, April 1990), Pillage and Rohrer presented and simulated an RC tree circuit that models signal integrity and crosstalk in low- to mid-frequency MOS circuit interconnect. This example confirms their simulations using RF Toolbox software.

Their circuit, reproduced in the following figure, consists of 11 resistors and 12 capacitors. In the paper, Pillage and Rohrer:

- Apply a ramp voltage input
- Compute transient responses
- Plot the output voltages across two different capacitors, `C7` and `C12`.

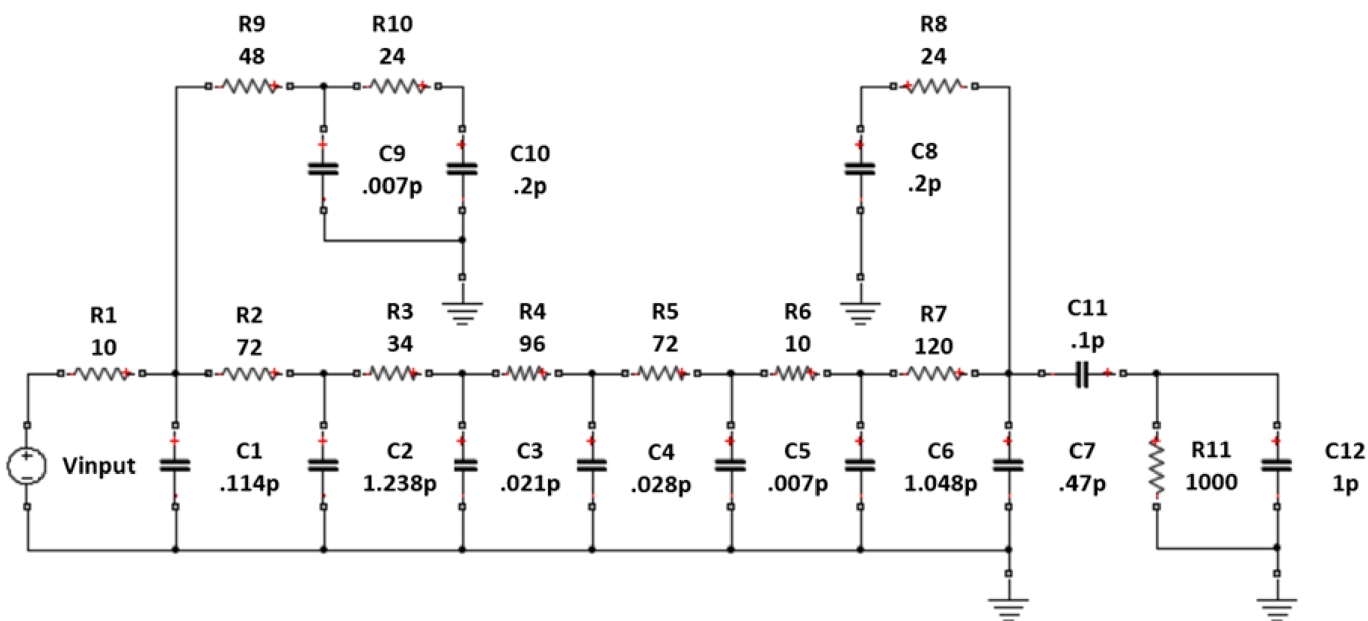


Figure 1: RC tree model of MOS interconnect with crosstalk.

With RF Toolbox software, you can programmatically construct this circuit in MATLAB and perform signal integrity simulations.

This example shows:

1. How to use `rfckt.seriesrlc`, `rfckt.shuntrlc`, `rfckt.series`, and `rfckt.cascade` object to programmatically construct the circuit as two different networks, depending on the desired output.
2. How to use `analyze` function to extract the S-parameters for each 2-port network over a wide frequency range.
3. How to use `s2tf` function with `Zsource = 0` and `Zload = Inf` to compute the voltage transfer function from input to each desired output.

4 How to use `rationalfit` function to produce rational-function approximations that capture the ideal RC-circuit behavior to a very high degree of accuracy.

5 How to use `timeresp` function to compute the transient response to the input voltage waveform.

Redraw Circuit as Distinct 2-Port Networks

To duplicate both output plots, RF Toolbox calculates the output voltage across C7 and C12. To that end, the circuit must be expressed as two distinct 2-port networks, each with the appropriate capacitor at the output. Figure 2 shows the 2-port configuration for computing the voltage across C7. Figure 3 shows the configuration for C12. Both 2-port networks retain the original circuit topology, and share much of the same structure.

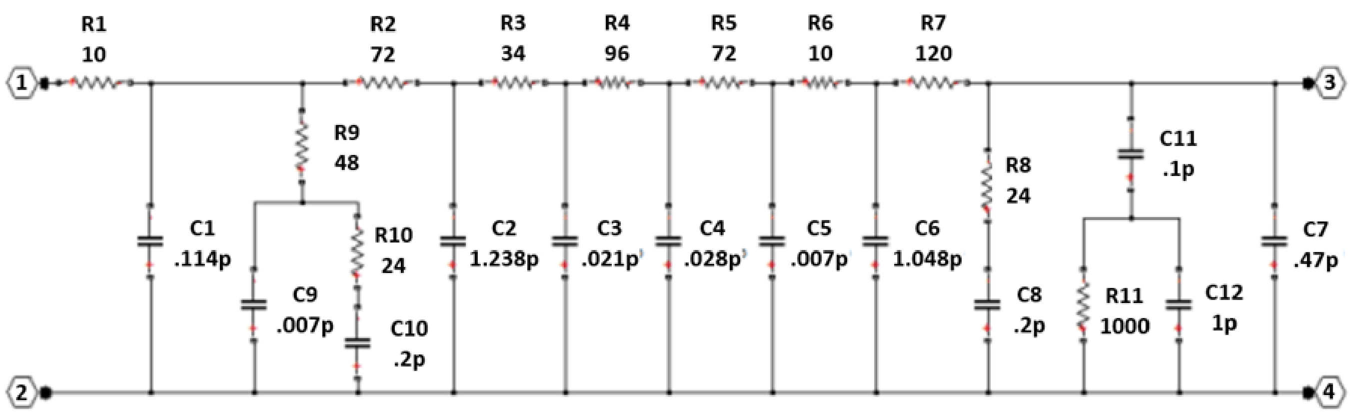


Figure 2: The circuit drawn as a 2-port network with output across C7.

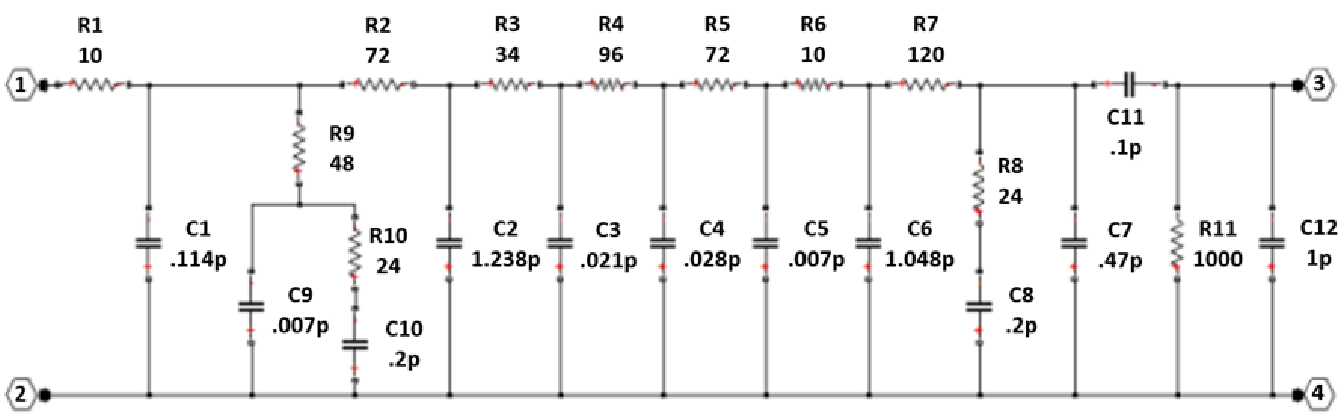


Figure 3: The circuit drawn as a 2-port network with output across C12.

Using RLC Building Blocks

All of the building blocks are formed by selecting appropriate values with the `rfckt.shuntrlc` object shown in Figure 4 or the `rfckt.seriesrlc` object shown in Figure 5. The 2-port building blocks are then connected using `rfckt.cascade` object as shown in Figure 6 or `rfckt.series` object as shown in Figure 7.

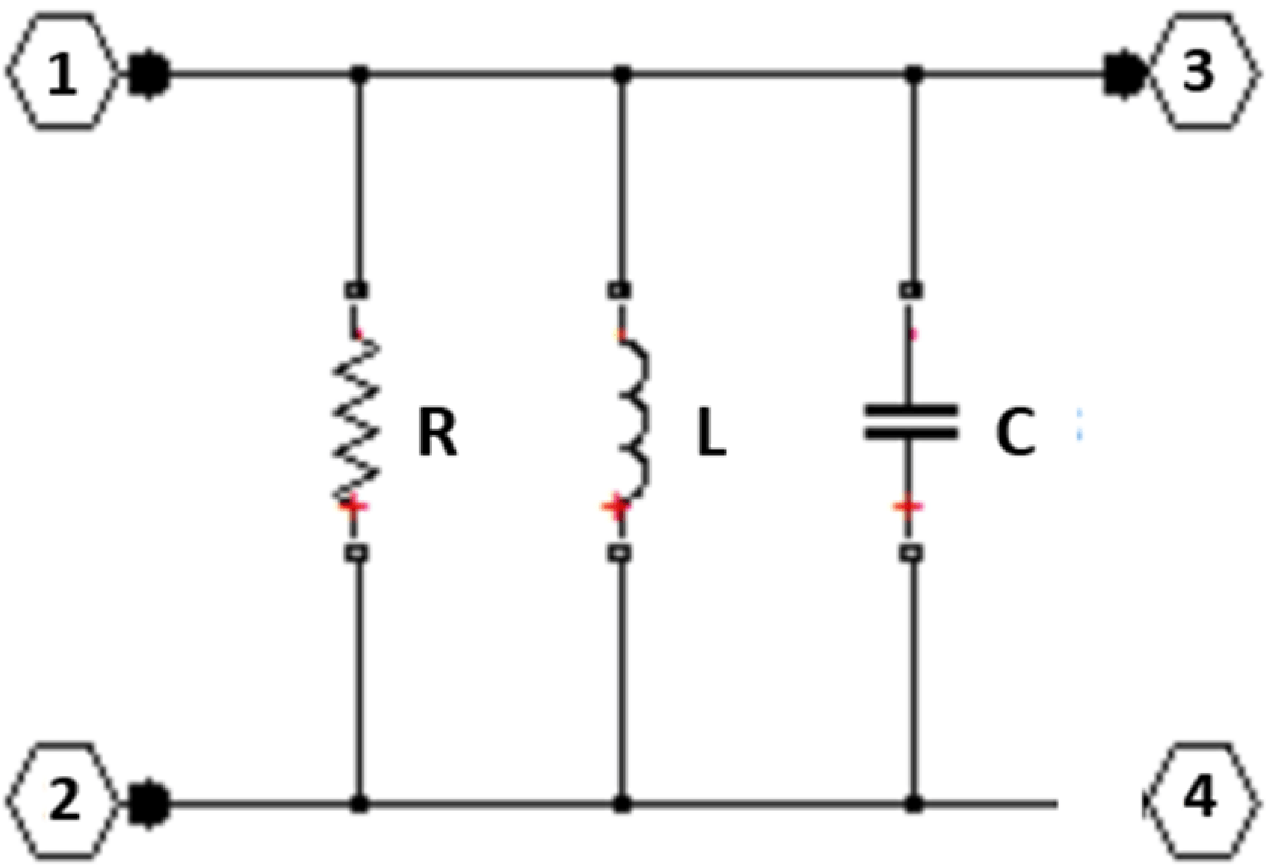


Figure 4: The 2-port network created using the `rfckt.shuntrlc` object.

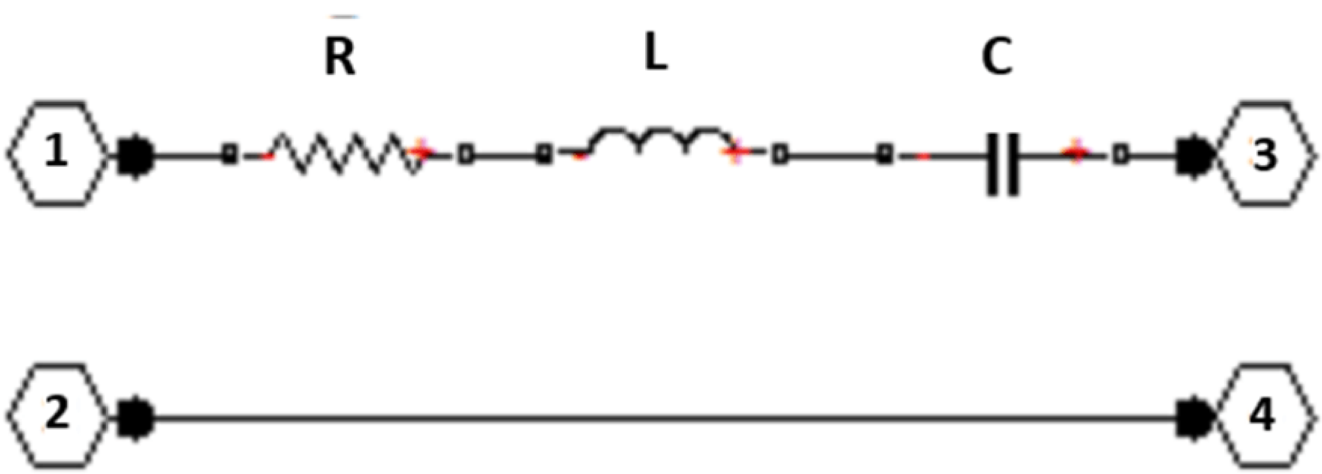


Figure 5: The 2-port network created using the `rfckt.seriesrlc` object.

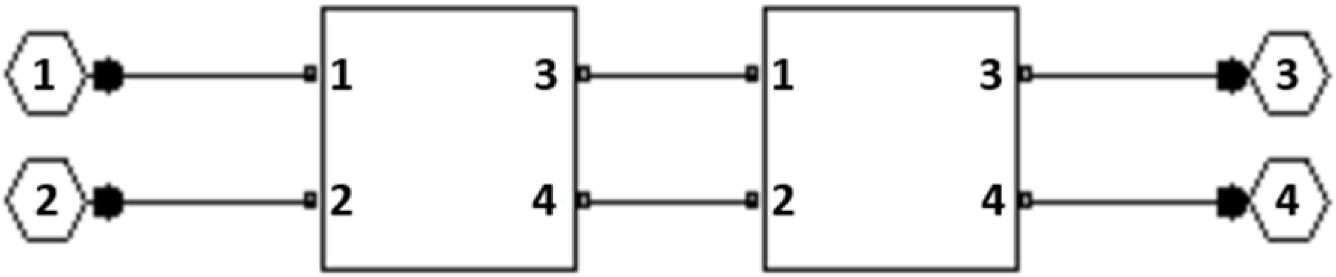


Figure 6: Connect 2-port networks with the `rfckt.cascade` object.

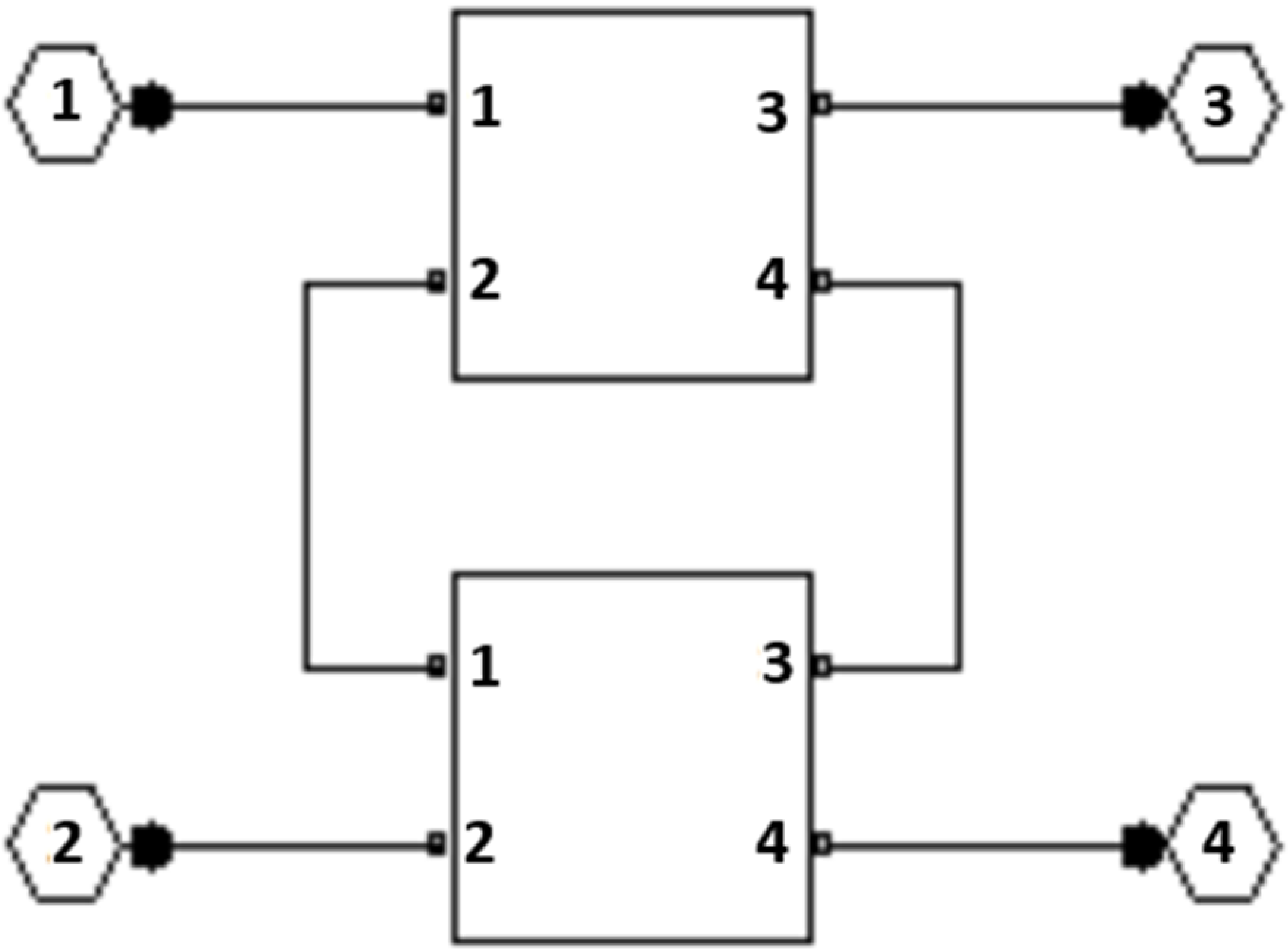


Figure 7: Connect 2-port networks with the `rfckt.series` object.

Shared Pieces of 2-Port Networks

The following MATLAB code constructs the portion of the network shared between the two variants.

```
R1 = rfckt.seriesrlc('R',10);
C1 = rfckt.shuntrlc('C',0.114e-12);
R9 = rfckt.shuntrlc('R',48);
C9 = rfckt.shuntrlc('C',0.007e-12);
R10 = rfckt.shuntrlc('R',24);
C10 = rfckt.shuntrlc('C',0.2e-12);
R10C10 = rfckt.series('Ckts',{R10,C10});
C9R10C10 = rfckt.cascade('Ckts',{C9,R10C10});
R9C9R10C10 = rfckt.series('Ckts',{R9,C9R10C10});
R2 = rfckt.seriesrlc('R',72);
C2 = rfckt.shuntrlc('C',1.238e-12);
R3 = rfckt.seriesrlc('R',34);
C3 = rfckt.shuntrlc('C',0.021e-12);
R4 = rfckt.seriesrlc('R',96);
C4 = rfckt.shuntrlc('C',0.028e-12);
R5 = rfckt.seriesrlc('R',72);
C5 = rfckt.shuntrlc('C',0.007e-12);
R6 = rfckt.seriesrlc('R',10);
C6 = rfckt.shuntrlc('C',1.048e-12);
R7 = rfckt.seriesrlc('R',120);
R8 = rfckt.shuntrlc('R',24);
C8 = rfckt.shuntrlc('C',0.2e-12);
R8C8 = rfckt.series('Ckts',{R8,C8});
sharedckt = rfckt.cascade('Ckts', ...
    {R1,C1,R9C9R10C10,R2,C2,R3,C3,R4,C4,R5,C5,R6,C6,R7,R8C8});

% Additional shared building blocks used in both 2-port networks.
C7 = rfckt.shuntrlc('C',0.47e-12);
R11C12 = rfckt.shuntrlc('R',1000,'C',1e-12);
```

Construct Each 2-Port Network

Figure 2 shows that constructing a 2-port network with an output port across `C7` requires creating `C11` using `rfckt.shuntrlc` object, then combining `C11` with `R11` and `C12` using `rfckt.series` object, and finally combining `C11R11C12` with the rest of the network and `C7` using `rfckt.cascade` object.

Similarly, Figure 3 shows that constructing a 2-port network with an output port across `C12` requires creating another version of `C11` (`C11b`) using `rfckt.seriesrlc` object and combining all the parts together using `rfckt.cascade` object.

Construct shunt RLC circuit.

```
C11 = rfckt.shuntrlc('C',0.1e-12);
C11R11C12 = rfckt.series('Ckts',{C11,R11C12});
cktC7 = rfckt.cascade('Ckts',{sharedckt,C11R11C12,C7});
```

Construct series RLC circuit.

```
C11b = rfckt.seriesrlc('C',0.1e-12);
cktC12 = rfckt.cascade('Ckts',{sharedckt,C7,C11b,R11C12});
```

Simulation Setup

The input signal used by Pillage and Rohrer is a voltage ramp from 0 to 5 volts with a rise time of one nanosecond and a duration of ten nanoseconds. The following MATLAB code models this signal with 1000 timepoints with a `sampleTime` of 0.01 nanoseconds.

The following MATLAB code also uses the logspace function to generate a vector of 101 logarithmically spaced analysis frequencies between 1 Hz and 100 GHz. Specifying a wide set of frequency points improves simulation accuracy.

```
sampleTime = 1e-11;
t = (0:1000)'*sampleTime;
input = [(0:100)'*(5/100); (101:1000)'*0+5];
freq = logspace(0,11,101)';
```

Simulate Each 2-Port Network

To simulate each network:

1. The analyze function extracts S-parameters over the specified frequency range.
2. The s2tf function, with option = 2, computes the gain from the source voltage to the output voltage. It allows arbitrary source and load impedances, in this case Zsource = 0 and Zload = Inf. The resulting transfer functions tfC7 and tfC12 are frequency-dependent data vectors that can be fit with rational-function approximation.
3. The rationalfit function generates high-accuracy rational-function approximations. The resulting approximations match the networks to machine accuracy.
4. The timeresp function computes the analytic solution to the state-space equations defined by a rational-function approximation. This methodology is fast enough to enable one to push a million bits through a channel.

Simulate cktC7 circuit.

```
analyze(cktC7,freq);
sparamsC7 = cktC7.AnalyzedResult.S_Parameters;
tfC7 = s2tf(sparamsC7,50,0,Inf,2);
fitC7 = rationalfit(freq,tfC7);
outputC7 = timeresp(fitC7,input,sampleTime);
```

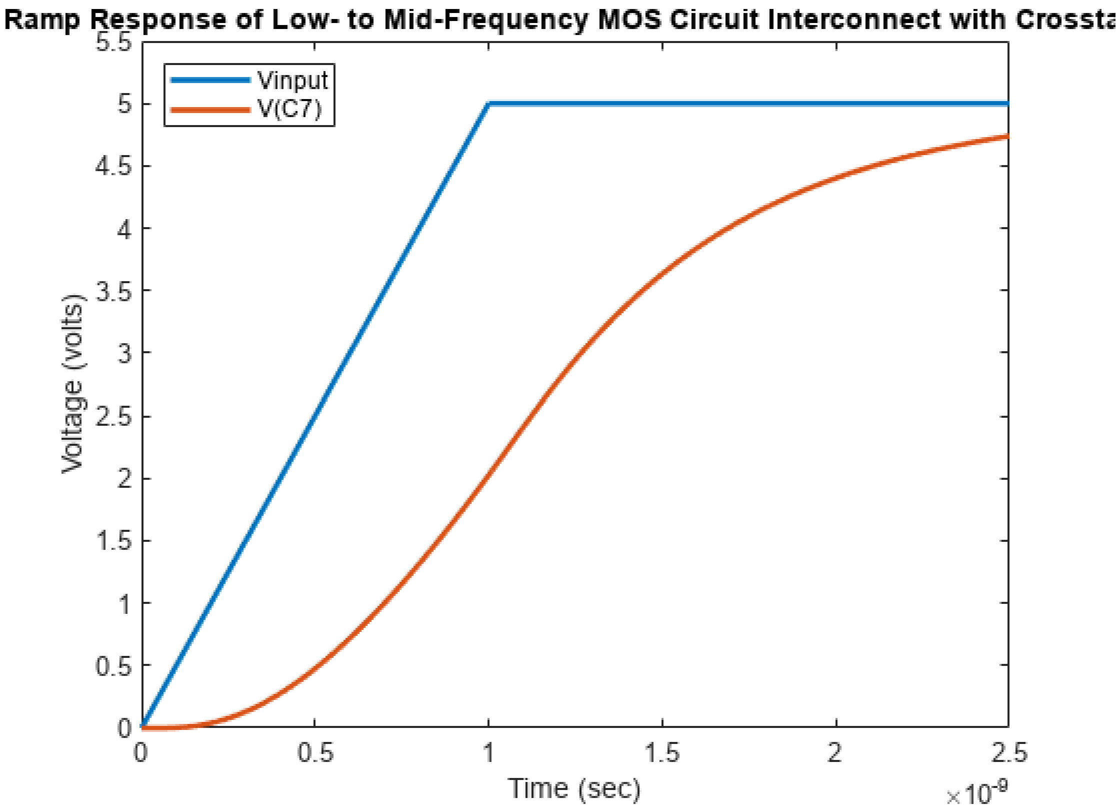
Simulate cktC12 circuit.

```
analyze(cktC12,freq);
sparamsC12 = cktC12.AnalyzedResult.S_Parameters;
tfC12 = s2tf(sparamsC12,50,0,Inf,2);
fitC12 = rationalfit(freq,tfC12);
outputC12 = timeresp(fitC12,input,sampleTime);
```

Plot Transient Responses

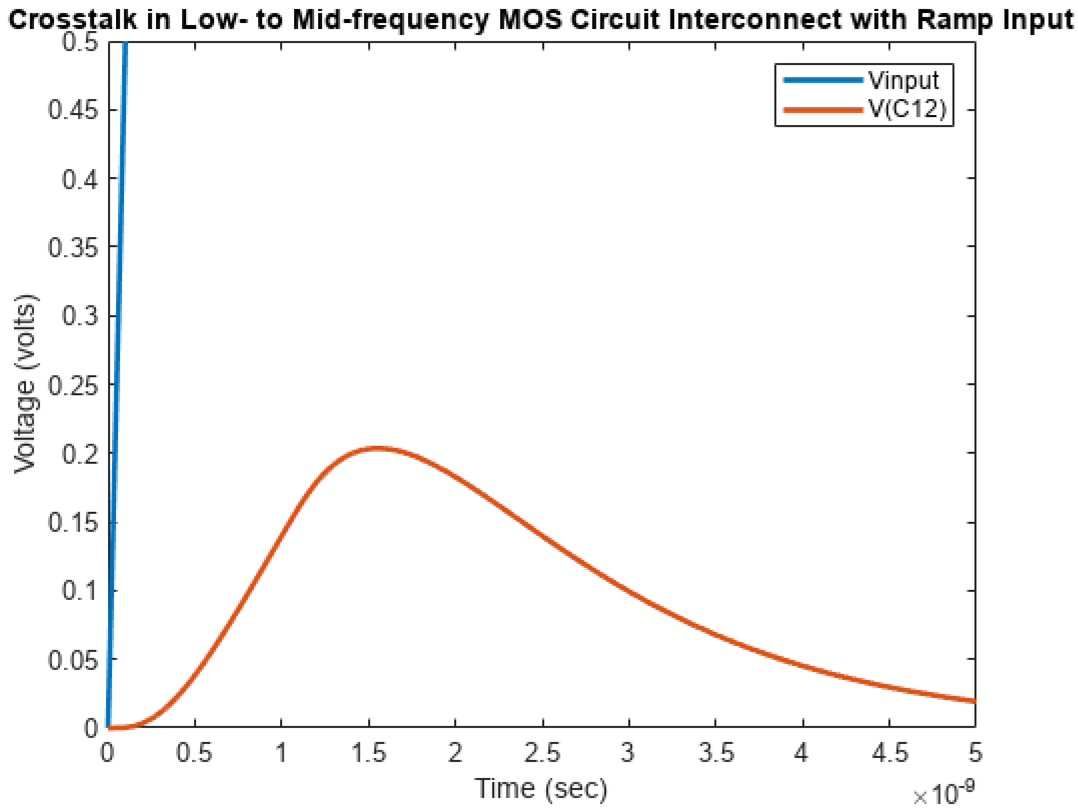
The outputs match Figures 23 and 24 of the Pillage and Rohrer paper. Plot the ramp response of low- to mid-frequency MOS circuit interconnect with crosstalk.

```
figure
plot(t,input,t,outputC7,'LineWidth',2)
axis([0 2.5e-9 0 5.5]);
title('Ramp Response of Low- to Mid-Frequency MOS Circuit Interconnect with Crosstalk');
xlabel('Time (sec)');
ylabel('Voltage (volts)');
legend('Vinput','V(C7)','Location','NorthWest');
```



Plot the crosstalk in low- to mid-frequency MOS circuit interconnect with ramp input.

```
figure
plot(t,input,t,outputC12,'LineWidth',2)
axis([0 5e-9 0 .5]);
title('Crosstalk in Low- to Mid-frequency MOS Circuit Interconnect with Ramp Input');
xlabel('Time (sec)');
ylabel('Voltage (volts)');
legend('Vinput','V(C12)','Location','NorthEast');
```



Verify Rational Fit Outside Fit Range

Though not shown in this example, you can also use the `freqresp` function to check the behavior of `rationalfit` function well outside the specified frequency range. The fit outside the specified range can sometimes cause surprising behavior, especially if frequency data near 0 Hz (DC) was not provided.

To perform this check for the rational-function approximation in this example, uncomment and run the following MATLAB code.

```
% widerFreqs = logspace(0,12,1001);
% respC7 = freqresp(fitC7,widerFreqs);
% figure
% loglog(freqs,abs(tfC7),'+',widerFreqs,abs(respC7))
% respC12 = freqresp(fitC12,widerFreqs);
% figure
% loglog(freqs,abs(tfC12),'+',widerFreqs,abs(respC12))
```

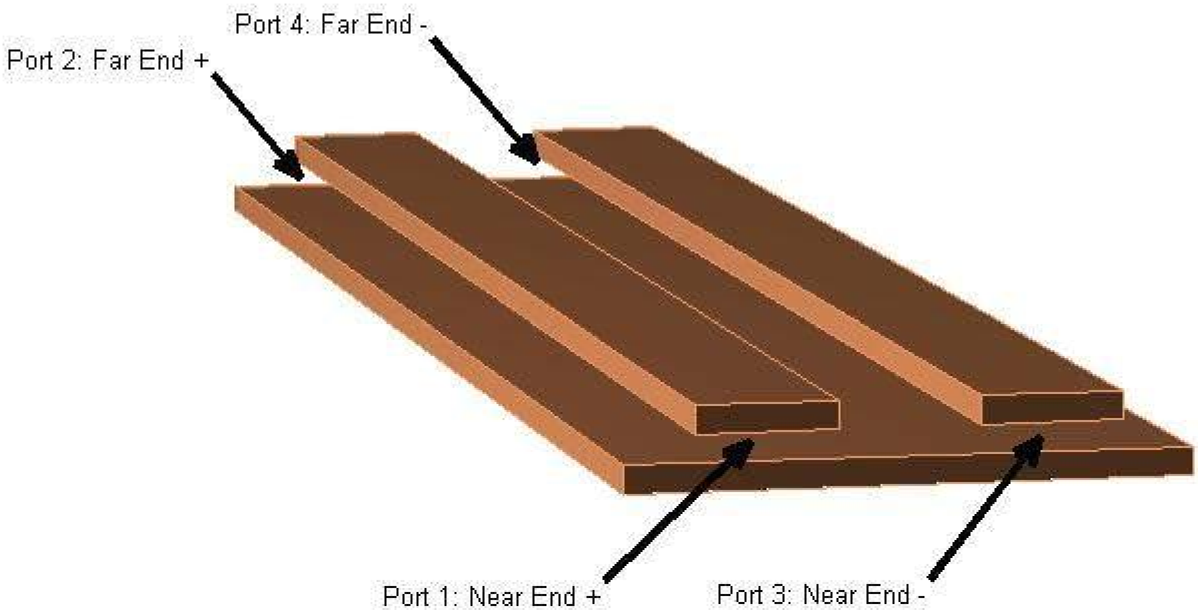
See Also

More About

- “MOS Interconnect and Crosstalk” on page 6-29

Model and Analyze High-Speed Backplanes

This five-part tutorial shows you how to model and analyze high-speed backplanes using `sparameters` and `rationalfit` functions. High-speed backplanes, as shown in the diagram, are vital in enabling data transfer between multiple boards connected to a larger system. In this example, you model a high-speed backplane using S-parameters and fit the S-parameter data for analysis.



Rational Fitting

RF Toolbox™ uses the `rationalfit` function to fit data defined in the frequency domain with an equivalent Laplace transfer function. Using rational function fitting you can create simple models for the required accuracy, model order reduction, zero phase on extrapolation to DC, and causal modeling system.

This type of modeling is useful to signal integrity engineers, whose goal is to reliably connect high-speed semiconductor devices with multi-Gbps serial data streams across backplanes and printed circuit boards.

Compared to traditional techniques such as linear interpolation, rational function fitting provides more insight into the physical characteristics of a high-speed backplane. It enables you to trade off between complexity and accuracy using model order reduction. For a given accuracy, rational functions are less complex than other types of models such as FIR filters generated by IFFT techniques. In addition, rational function models inherently constrain the phase to be zero on extrapolation to DC. Methods that do not provide as much insight into physical characteristics require elaborate constraint algorithms to force the extrapolated phase to zero at DC.

Design Workflow

To model, analyze, and export high-speed backplanes, follow these five steps.

1. Import S-parameters and port reduction: Use RF Toolbox to import N-port S-parameters representing high-speed backplane channels to model the channels and the crosstalk between channels. This example imports 16-port S-parameters that represent a 16-port high-speed backplane and reduces them to 4-port S-parameters for analysis. For more information, see “Use S-parameters with Port Reduction” on page 6-51.
2. Fit S-parameters: Fit 4-port S-parameters from the step 1 using the `rataionalfit` function to create a simpler high-speed backplane model for accuracy, model order reduction, and zero phase on extrapolation to DC. For more information, see “Fit S-Parameters with Rational Function” on page 6-55.
3. Analyze your high-speed backplane: Calculate the time-domain reflectometry and time-domain transmission of a differential high-speed backplane channel for further analysis. For more information, see “Compute Differential TDR and TDT” on page 6-62.
4. Build your model in Simulink: Build your model in Simulink® to simulate your high-speed backplane and study the impact of the differential backplane on the random input signal. For more information, see “Build Simulink Model from Rational Function” on page 6-65.
5. Export your model to Verilog-A: Export your high-speed backplane model in S-parameters to a Verilog-A module that models the high-level behavior of a high-speed backplane. For more information, see “Export Verilog-A module from Rational Function” on page 6-69.

See Also
`rationalfit` | `rational` | `rfmodel.rational`

Use S-parameters with Port Reduction

This example shows how to use RF Toolbox™ to import N-port S-parameters representing high-speed backplane channels, and converts 16-port S-parameters to 4-port S-parameters to model the channels and the crosstalk between the channels.

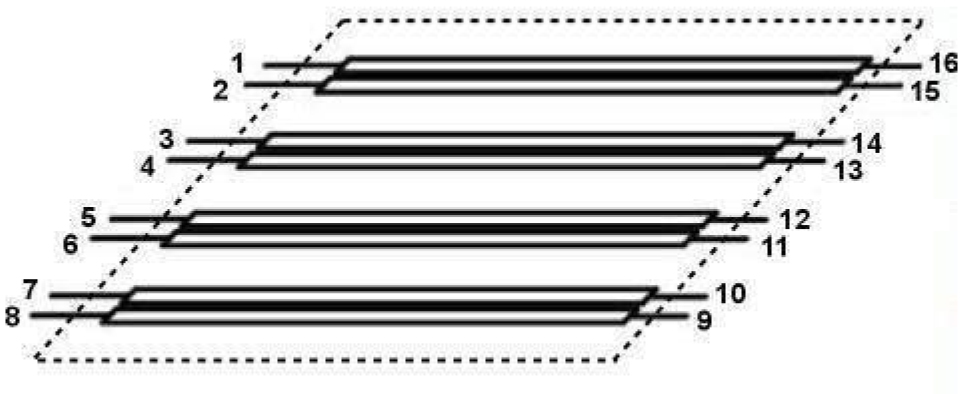


Figure 1: 16-Port differential backplane

Read Single-Ended 16-Port S-Parameters

Read a Touchstone® data file into an `sparameters` object. The data in this file are the 50-ohm S-parameters of a 16-port differential backplane designed for a 2-Gbps high-speed signal, shown in Figure 1, measured at 1496 frequencies ranging from 50 MHz to 15 GHz.

```
filename = 'default.s16p';
backplane = sparameters(filename)
```

```
backplane = 
  sparameters: S-parameters object

       NumPorts: 16
    Frequencies: [1496x1 double]
     Parameters: [16x16x1496 double]
      Impedance: 50

  rfparam(obj,i,j) returns S-parameter Sij
```

```
freq = backplane.Frequencies;
```

Convert 16-Port S-Parameters to 4-Port S-Parameters to Model Differential Channel

Use the `snp2smp` function to convert 16-port S-parameters to 4-port S-parameters that represent the first differential channel. The port index of this differential channel, `N2M`, specifies how the ports of the 16-port S-parameters map to the ports of the 4-port S-parameters, is `[1 16 2 15]`. (The port indices of the second, third and fourth channels are `[3 14 4 13]`, `[5 12 6 11]` and `[7 10 8 9]`, respectively). The other 12 ports, `[3 4 5 6 7 8 9 10 11 12 13 14]`, are terminated with the characteristic `Impedance` specified by the `sparameters` object. Then, create an `sparameters` object with 4-port S-parameters for the first differential channel.

```
                 (Port 1)         (Port 16)
        Port 1  > ----->|         |<----- <   Port 2
```

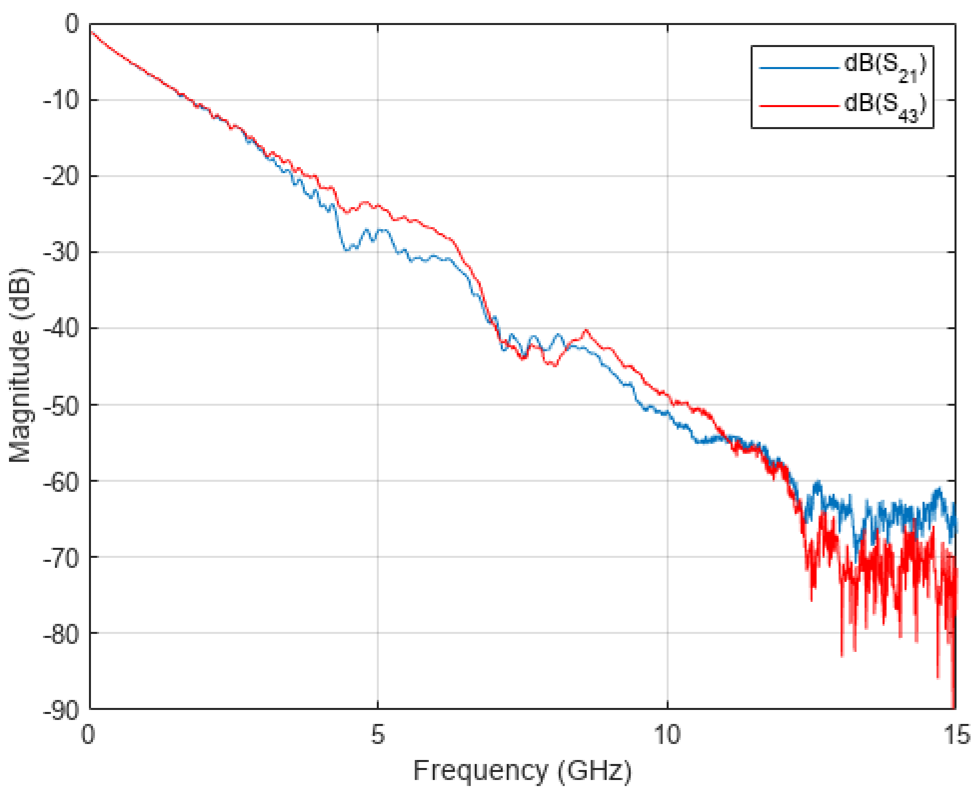
```
                  |    DUT    |
Port 3  > ----->|           |<----- <   Port 4
         (Port 2)          (Port 15)
```

```
n2m = [1 16 2 15];
z0 = backplane.Impedance;
first4portdata = snp2smp(backplane.Parameters,z0,n2m,z0);
first4portsparams = sparameters(first4portdata,freq,z0)
```

```
first4portsparams = 
  sparameters: S-parameters object

       NumPorts: 4
    Frequencies: [1496x1 double]
     Parameters: [4x4x1496 double]
      Impedance: 50

  rfparam(obj,i,j) returns S-parameter Sij
```

Plot S21 and S43 of the first differential channel.

```
figure
rfplot(first4portsparams,2,1)
hold on
rfplot(first4portsparams,4,3,'-r')
```




```
% % If you want to write the 4-port S-parameters of the differential
% % channel into a |.s4p| file, then uncomment the line below.
```

```
%
% rfwrite(first4portsparams,'firstchannel.s4p')
```

Convert 16-Port S-Parameters to 4-Port S-Parameters to Model Crosstalk Between Two Differential Channels

Use the `snp2smp` function to convert 16-port S-parameters to 4-port S-parameters that represent the crosstalk between port `[3 4]` and port `[16 15]`. As shown in Figure 1, these ports are on different channels. The other 12 ports, `[1 2 5 6 7 8 9 10 11 12 13 14]`, are terminated with the characteristic `Impedance` specified by the `sparameters` object. Then, create an `sparameters` object with 4-port S-parameters for the crosstalk.

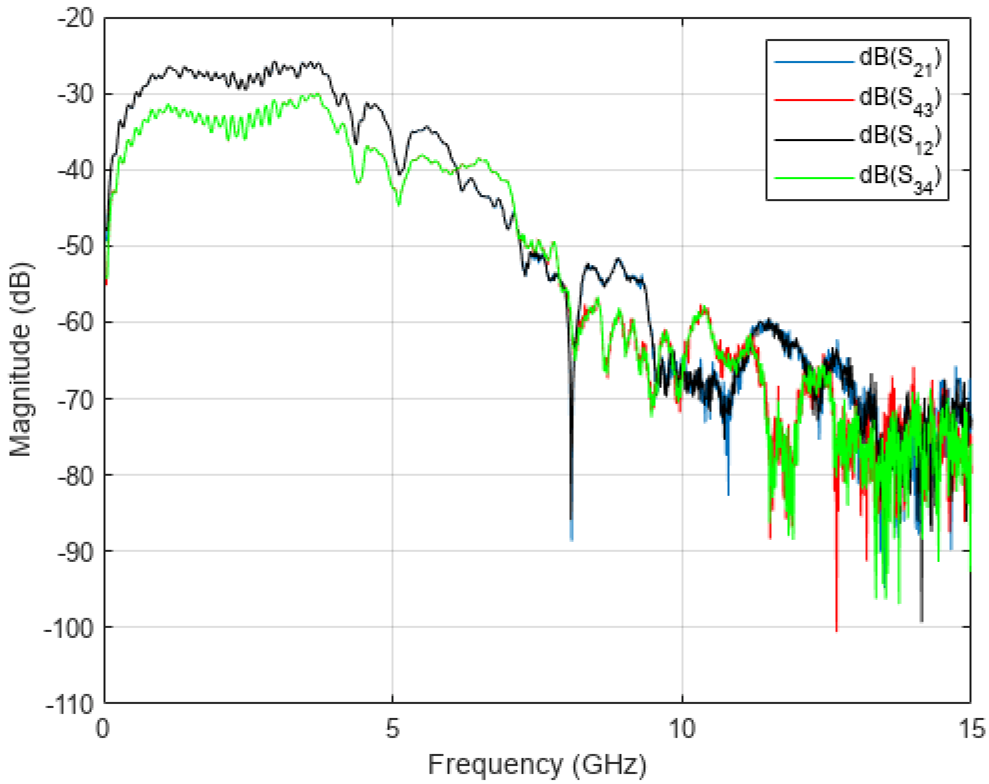
```
                 (Port 3)         (Port 16)
        Port 1  > ----->|         |<----- <   Port 2
                        |   DUT   |
        Port 3  > ----->|         |<----- <   Port 4
                 (Port 4)         (Port 15)
```

```
n2m = [3 16 4 15];
crosstalk4portdata = snp2smp(backplane.Parameters,z0,n2m,z0);
crosstalk4portsparams = sparameters(crosstalk4portdata,freq,z0)
```

```
crosstalk4portsparams = 
  sparameters: S-parameters object

       NumPorts: 4
    Frequencies: [1496x1 double]
     Parameters: [4x4x1496 double]
      Impedance: 50

  rfparam(obj,i,j) returns S-parameter Sij
```

Plot `S21`, `S43`, `S12` and `S34` to show the crosstalk between these two channels.

```
figure
rfplot(crosstalk4portsparams,2,1)
hold on
rfplot(crosstalk4portsparams,4,3,'-r')
rfplot(crosstalk4portsparams,1,2,'-k')
rfplot(crosstalk4portsparams,3,4,'-g')
```




```
% % If you want to write the 4-port S-parameters of the crosstalk into an
% % .s4p file, then uncomment the line below.
%
% rfwrite(crosstalk4portsparams,'crosstalk.s4p')
```

See Also

More About

- “Fit S-Parameters with Rational Function” on page 6-55
- “Compute Differential TDR and TDT” on page 6-62
- “Export Verilog-A module from Rational Function” on page 6-69

Fit S-Parameters with Rational Function

This example shows how to use RF Toolbox™ to model a differential high-speed backplane channel using rational functions. This type of model is useful to signal integrity engineers, whose goal is to reliably connect high-speed semiconductor devices with, for example, multi-Gbps serial data streams across backplanes and printed circuit boards.

Compared to traditional techniques such as linear interpolation, rational function fitting provides more insight into the physical characteristics of a high-speed backplane. It provides a means, called model order reduction, of making a trade-off between complexity and accuracy. For a given accuracy, rational functions are less complex than other types of models such as FIR filters generated by IFFT techniques. In addition, rational function models inherently constrain the phase to be zero on extrapolation to DC. Less physically-based methods require elaborate constraint algorithms in order to force the extrapolated phase to zero at DC.

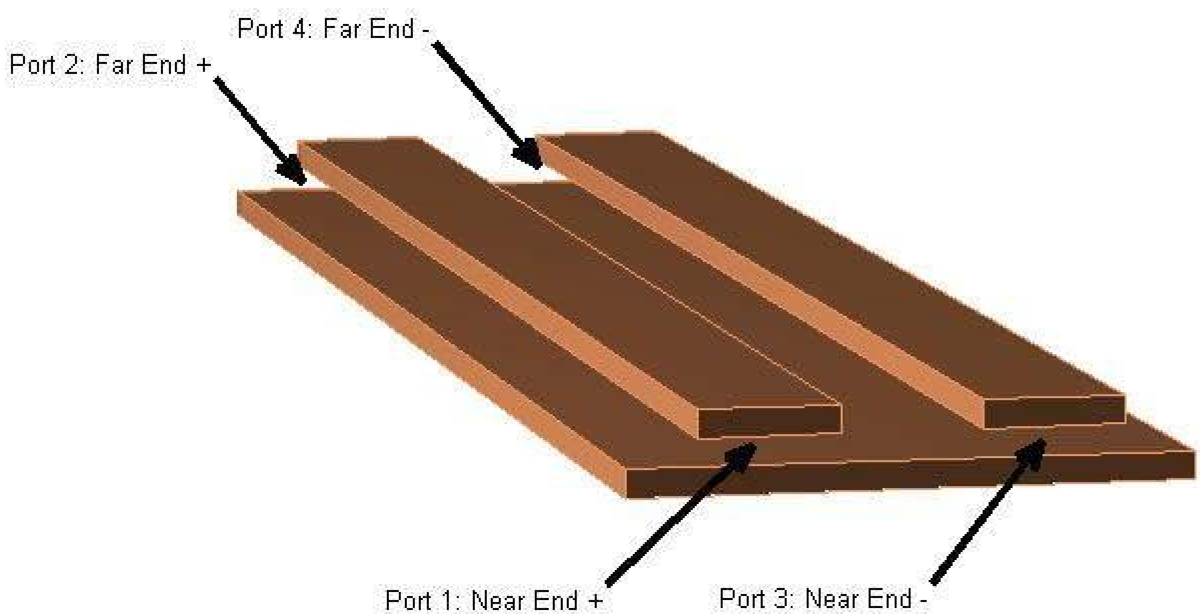


Figure 1: A differential high-speed backplane channel

Read Single-Ended 4-Port S-Parameters and Convert Them to Differential 2-Port S-Parameters

Read a Touchstone® data file, `default.s4p`, into an `sparameters` object. The parameters in this data file are the 50-ohm S-parameters of the single-ended 4-port passive circuit shown in Figure 1, given at 1496 frequencies ranging from 50 MHz to 15 GHz. Then, get the single-ended 4-port S-parameters and use the matrix conversion function `s2sdd` to convert them to differential 2-port S-parameters. Finally, plot the differential `S11` parameter on a Smith chart.

```
filename = 'default.s4p';
backplane = sparameters(filename);
data = backplane.Parameters;
freq = backplane.Frequencies;
z0 = backplane.Impedance;
```

Convert to 2-port differential S-parameters.

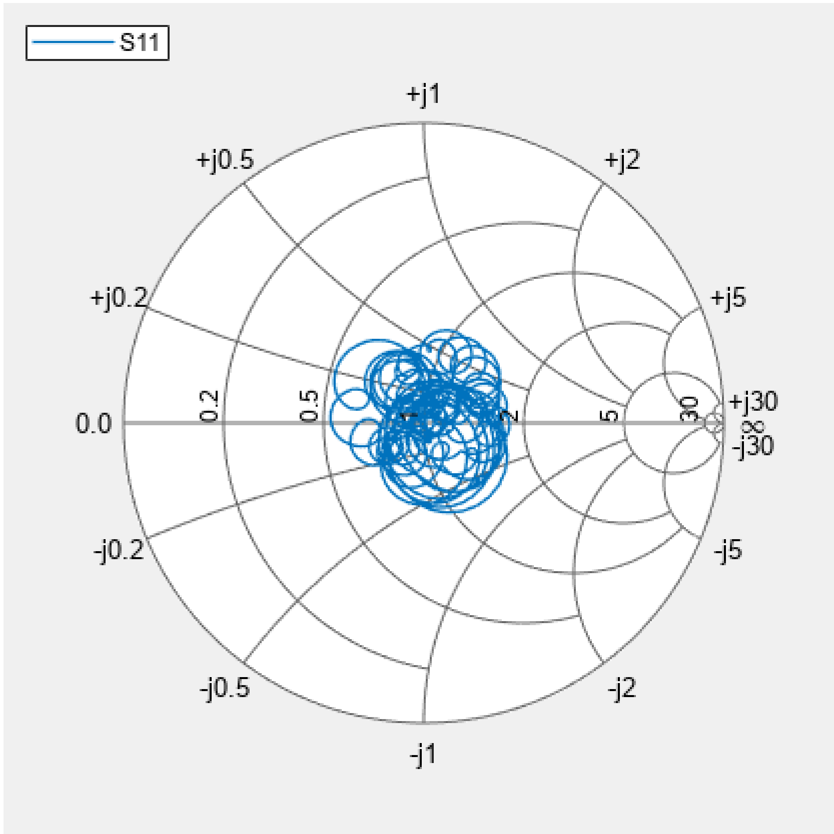
```
diffdata = s2sdd(data);
diffz0 = 2*z0;
```

By default, s2sdd expects ports 1 & 3 to be inputs and ports 2 & 4 to be outputs. However if your data has ports 1 & 2 as inputs and ports 3 & 4 as outputs, then use 2 as the second input argument to `s2sdd` function to specify this alternate port arrangement. For example, `diffdata = s2sdd(data,2);`

```
diffsparams = sparameters(diffdata,freq,diffz0)
```

```
diffsparams = 
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [1496x1 double]
     Parameters: [2x2x1496 double]
      Impedance: 100

  rfparam(obj,i,j) returns S-parameter Sij
```

```
figure
smithplot(diffsparams,1,1)
```



Compute Transfer Function and Its Rational Function Object Representation

First, use the `s2tf` function to compute the differential transfer function. Then, use the `rationalfit` function to compute the analytical form of the transfer function and store it in an `rfmodel.rational` object. The `rationalfit` function fits a rational function object to the specified data over the specified frequencies. The run time depends on the computer, the fitting tolerance, the number of data points, etc.

```
difftransfunc = s2tf(diffdata,diffz0,diffz0,diffz0);
delayfactor = 0.98; % Delay factor. Leave at the default of zero if your
                    % data does not have a well-defined principle delay
rationalfunc = rationalfit(freq,difftransfunc,'DelayFactor',delayfactor)
```

```
rationalfunc = 
   rfmodel.rational with properties:

        A: [31x1 double]
        C: [31x1 double]
        D: 0
    Delay: 6.5521e-09
     Name: 'Rational Function'
```

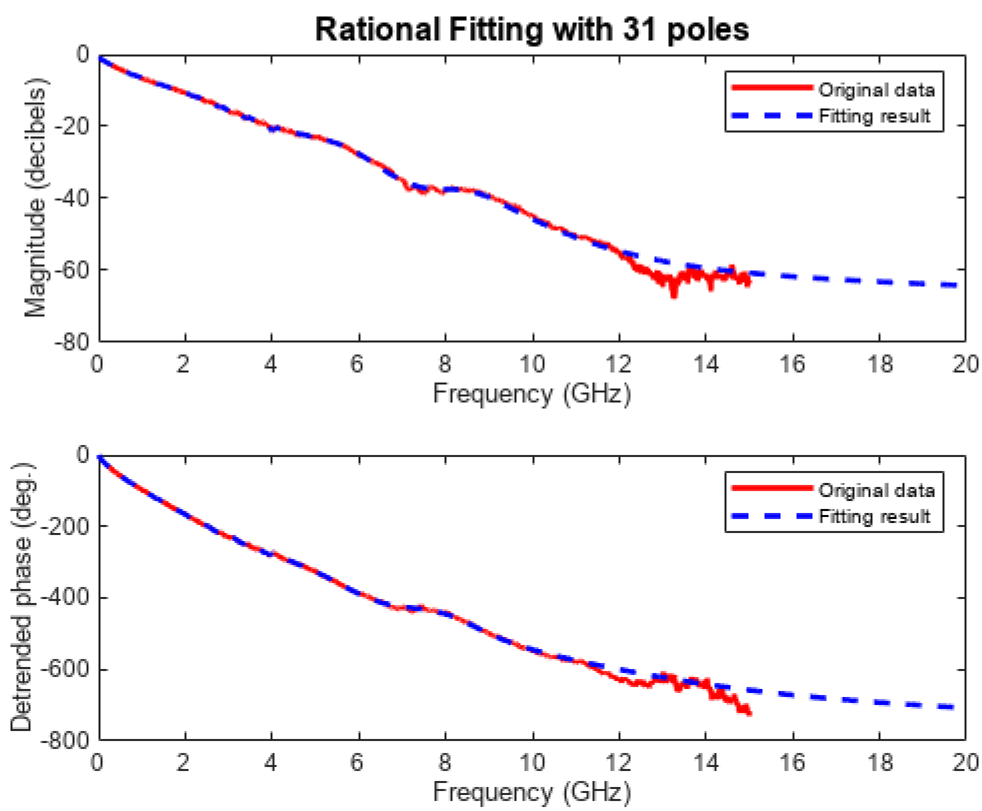
```
npoles = length(rationalfunc.A);
fprintf('The derived rational function contains %d poles.\n',npoles);
```

```
The derived rational function contains 31 poles.
```

Validate Differential-Mode Frequency Response

Use the `freqresp` method of the `rfmodel.rational` object to get the frequency response of the rational function object. Then, create a plot to compare the frequency response of the rational function object and that of the original data. Note that detrended phase (i.e. phase after the principle delay is removed) is plotted in both cases.

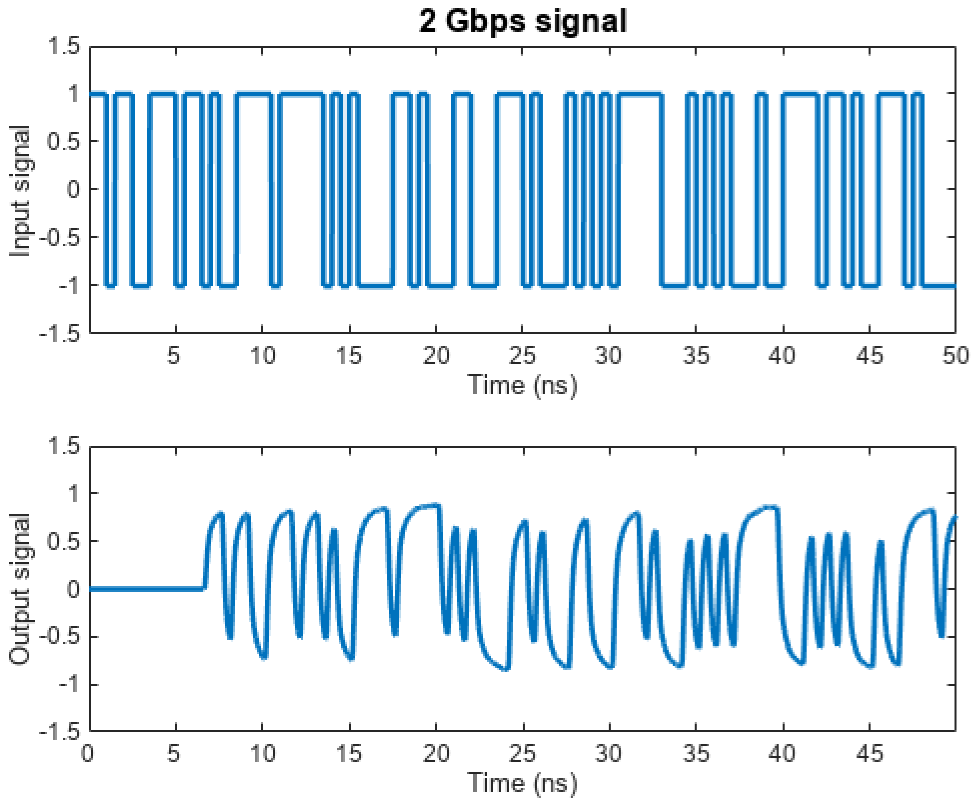
```
freqsforresp = linspace(0,20e9,2000)';
resp = freqresp(rationalfunc,freqsforresp);

figure
subplot(2,1,1)
plot(freq*1.e-9,20*log10(abs(difftransfunc)),'r',freqsforresp*1.e-9, ...
                20*log10(abs(resp)),'b--','LineWidth',2)
title(sprintf('Rational Fitting with %d poles',npoles),'FontSize',12)
ylabel('Magnitude (decibels)')
xlabel('Frequency (GHz)')
legend('Original data','Fitting result')

subplot(2,1,2)
origangle = unwrap(angle(difftransfunc))*180/pi+360*freq*rationalfunc.Delay;
plotangle = unwrap(angle(resp))*180/pi+360*freqsforresp*rationalfunc.Delay;
plot(freq*1.e-9,origangle,'r',freqsforresp*1.e-9,plotangle,'b--', ...
    'LineWidth',2)
ylabel('Detrended phase (deg.)')
xlabel('Frequency (GHz)')
legend('Original data','Fitting result')
```



Calculate and Plot Differential Input and Output Signals of High-Speed Backplane

Generate a random 2 Gbps pulse signal. Then, use the `timeresp` method of the `rfmodel.rational` object to compute the response of the rational function object to the random pulse. Finally, plot the input and output signals of the rational function model that represents the differential circuit.

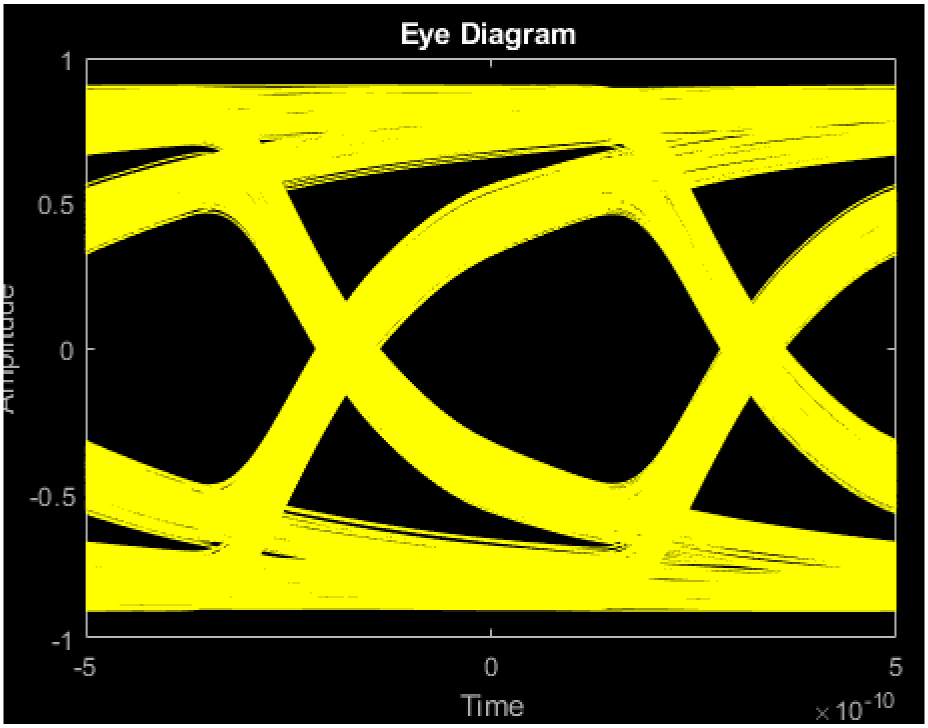
```
datarate = 2*1e9;                % Data rate: 2 Gbps
samplespersymb = 100;
pulsewidth = 1/datarate;
ts = pulsewidth/samplespersymb;
numsamples = 2^17;
numplotpoints = 10000;

t_in = double((1:numsamples)')*ts;
input = sign(randn(1,ceil(numsamples/samplespersymb)));
input = repmat(input,[samplespersymb, 1]);
input = input(:);
[output,t_out] = timeresp(rationalfunc,input,ts);

figure
subplot(2,1,1)
plot(t_in(1:numplotpoints)*1e9,input(1:numplotpoints),'LineWidth',2)
title([num2str(datarate*1e-9),' Gbps signal'],'FontSize',12)
ylabel('Input signal')
xlabel('Time (ns)')
axis([-inf,inf,-1.5,1.5])

subplot(2,1,2)
plot(t_out(1:numplotpoints)*1e9,output(1:numplotpoints),'LineWidth',2)
ylabel('Output signal')
xlabel('Time (ns)')
axis([-inf,inf,-1.5,1.5])
```



Plot Eye Diagram of 2 Gbps Output Signal

Estimate and remove the delay from the output signal and create an eye diagram by using Communications Toolbox™ functions.

```
if ~isempty(which('eyediagram'))
    ignoreBits = 1500;
    eyediagram(output(ignoreBits:end),2*samplespersymb,2/datarate)
end
```



See Also

More About

- "Use S-parameters with Port Reduction" on page 6-51
- "Compute Differential TDR and TDT" on page 6-62
- "Build Simulink Model from Rational Function" on page 6-65

Compute Differential TDR and TDT

This example shows how to use RF Toolbox™ functions to calculate the TDR (Time-Domain Reflectometry) and TDT (Time-Domain Transmission) of a differential high-speed backplane channel.

Read Single-Ended 4-Port S-Parameters and Convert Them to Differential 2-Port S-Parameters

Read a Touchstone® data file, `default.s4p`, into an `sparameters` object. The parameters in this data file are the 50-ohm S-parameters of a single-ended 4-port passive circuit, measured at 1496 frequencies ranging from 50 MHz to 15 GHz. Then, get the single-ended 4-port S-parameters from the data object, and use the matrix conversion function `s2sdd` to convert them to differential 2-port S-parameters.

```
filename = 'default.s4p';
backplane = sparameters(filename);
data = backplane.Parameters;
freq = backplane.Frequencies;
z0 = backplane.Impedance;
```

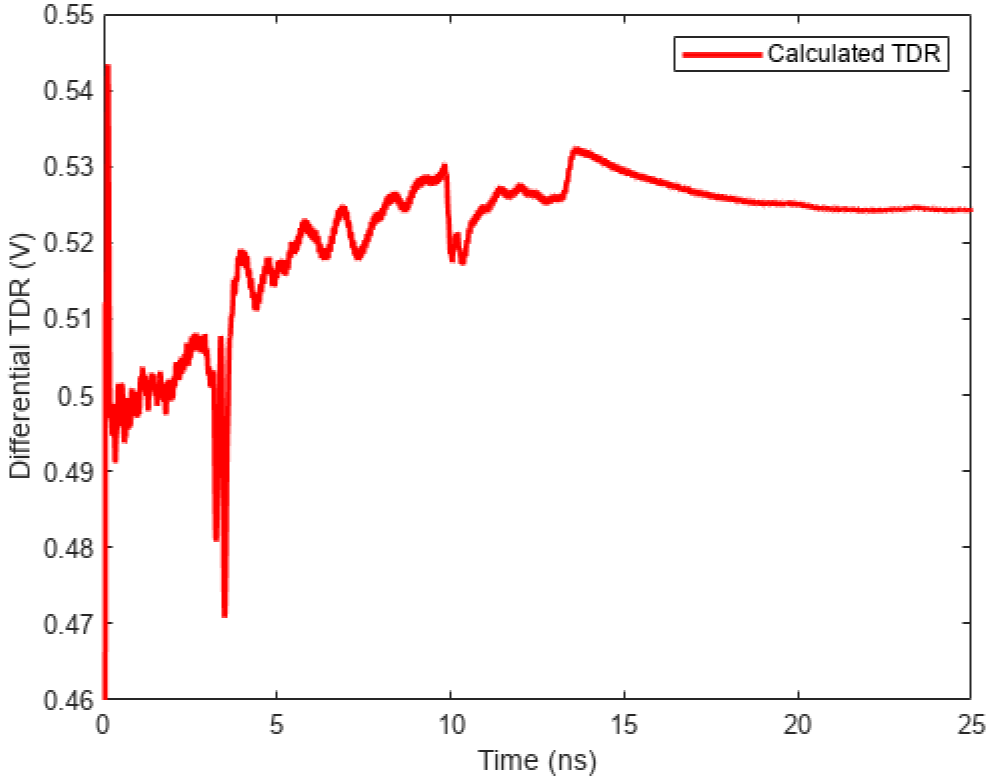
Convert to 2-port differential S-parameters.

```
diffdata = s2sdd(data);
diffsparams = sparameters(diffdata,freq,2*z0);
```

Calculate and Plot the Differential Time-Domain Reflectometry

TDR is the reflected voltage signal for a step input. First, extract the differential `S11` data using the `rfparam` function, and convert the `S11` data to TDR voltage transfer function data [1]. Next, create a rational function of that data using the `rationalfit` function, then compute the TDR using the `stepresp` function of the `rfmodel.rational` object. Lastly, plot the calculated TDR.

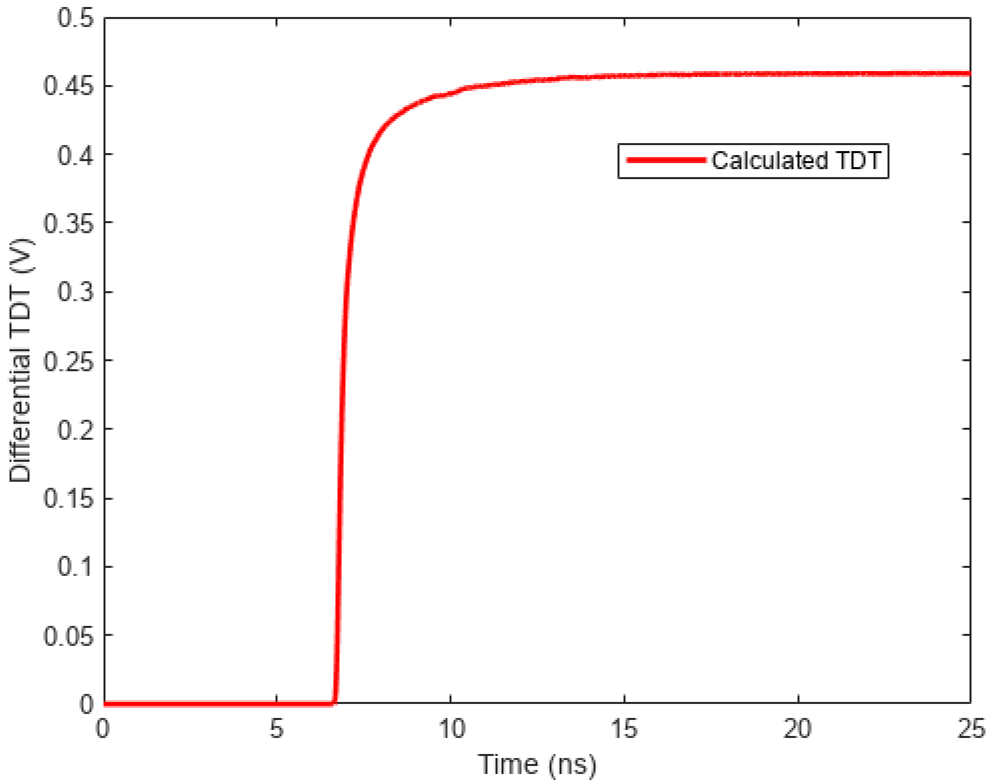
```
s11 = rfparam(diffsparams,1,1);
Vin = 1;
tdrfreqdata = Vin*(s11+1)/2;
tdrfit = rationalfit(freq,tdrfreqdata,'NPoles',350);
Ts = 5e-12;
N = 5000; % number of samples
Trise = 5e-11; % Define a step signal
[Vtdr,tdrT] = stepresp(tdrfit,Ts,N,Trise);
figure
plot(tdrT*1e9,Vtdr,'r','LineWidth',2)
ylabel('Differential TDR (V)')
xlabel('Time (ns)')
legend('Calculated TDR')
ylim([0.46 0.55])
```



Calculate and Plot the Differential Time-Domain Transmission

TDT is the transmitted voltage signal for a step input. Use the `rationalfit` function to get the rational function object of the TDT voltage frequency data, then use the `stepresp` function to compute TDT. Lastly, plot the calculated TDT.

```
delayfactor = 0.98; % Delay factor. Set delay factor to zero if your
                    % data does not have a well-defined delay
s21 = rfparam(diffsparams,2,1);
tdtfreqdata = Vin*s21/2;
tdtfit = rationalfit(freq,tdtfreqdata,'DelayFactor',delayfactor);
Ts = 5e-12;
N = 5000; % number of samples
Trise = 5e-11;
[tdt,tdtT] = stepresp(tdtfit,Ts,N,Trise);
figure
plot(tdtT(1:N)*1e9,tdt(1:N),'r','LineWidth',2)
ylabel('Differential TDT (V)')
xlabel('Time (ns)')
legend('Calculated TDT','Location','best')
```



References

[1] A. S. Ali, R. Mittra. "Time-Domain Reflectometry using Scattering Parameters and a De-Embedding Application" Technical Report, Electromagnetic Communication Laboratory Report No. 86-4, May 1986.

See Also

More About

- “Use S-parameters with Port Reduction” on page 6-51
- “Fit S-Parameters with Rational Function” on page 6-55
- “Build Simulink Model from Rational Function” on page 6-65

Build Simulink Model from Rational Function

This example shows how to use Simulink® to simulate a differential high-speed backplane channel. The example first reads a Touchstone® data file that contains single-ended 4-port S-parameters for a differential high-speed backplane and converts them to 2-port differential S-parameters. It computes the transfer function of the differential circuit and uses the `rationalfit` function to fit a closed-form rational function to the circuit's transfer function. Then, the example converts the poles and residues of the rational function object into the numerators and denominators of the Laplace Transform S-Domain transfer functions that it uses to build the Simulink model of the rational function object.

To run this example, you must have Simulink installed.

Read Single-Ended 4-Port S-Parameters and Convert Them to Differential 2-Port S-Parameters

Read a Touchstone data file, `default.s4p`, into an `sparameters` object. The parameters in this data file are the 50-ohm S-parameters of a single-ended 4-port passive circuit, measured at 1496 frequencies ranging from 50 MHz to 15 GHz. Then, get the single-ended 4-port S-parameters from the data object, and use the matrix conversion function `s2sdd` to convert them to differential 2-port S-parameters.

```
filename = 'default.s4p';
backplane = sparameters(filename);
data = backplane.Parameters;
freq = backplane.Frequencies;
z0 = backplane.Impedance;
```

Convert to 2-port differential S-parameters. This operation pairs together odd-numbered ports first, followed by the even-numbered ports. If a different configuration has been used to measure the single ended S-parameters, you can specify a different second argument in the s2sdd command. For example, option "2" will allow you to pair the input and output ports in ascending order. Alternatively, you can use the command snp2smp to change the port order.

```
diffdata = s2sdd(data,1);
diffz0 = 2*z0;
```

Compute Transfer Function and Its Rational Function Representation

First, use the `s2tf` function to compute the differential transfer function. Then, use the `rationalfit` function to compute the closed form of the transfer function and store it in an `rfmodel.rational` object. The `rationalfit` function fits a rational function object to the specified data over the specified frequencies.

```
difftf = s2tf(diffdata,diffz0,diffz0,diffz0);
fittol = -30;           % Rational fitting tolerance in dB
delayfactor = 0.9;      % Delay factor
rationalfunc = rationalfit(freq,difftf,fittol,'DelayFactor', delayfactor)
npoles = length(rationalfunc.A);
fprintf('The derived rational function contains %d poles.\n', npoles);
```

```
rationalfunc = 

   rfmodel.rational with properties:
```

```
        A: [20x1 double]
        C: [20x1 double]
        D: 0
    Delay: 6.0172e-09
     Name: 'Rational Function'

The derived rational function contains 20 poles.
```

Get Numerator and Denominator of Laplace Transform S-Domain Transfer Functions

This example uses Laplace Transform S-Domain transfer functions to represent the backplane in the Simulink model. Convert the poles and corresponding residues of the rational function object into numerator and denominator form for use in the Laplace Transform transfer function blocks. Each transfer function block represents either one real pole and the corresponding real residue, or a pair of complex conjugate poles and residues, so the transfer function block always has real coefficients. For this example, the rational function object contains 2 real poles/residues and 6 pairs of complex poles/residues, so the Simulink model contains 8 transfer function blocks.

```
A = rationalfunc.A;
C = rationalfunc.C;
den = cell(size(A));
num = cell(size(A));
k = 1;                          % Index of poles and residues
n = 0;                          % Index of numerators and denominators
while k <= npoles
    if isreal(A(k))             % Real poles
        n = n + 1;
        num{n} = C(k);
        den{n} = [1, -A(k)];
        k = k + 1;
    else                        % Complex poles
        n = n + 1;
        real_a = real(A(k));
        imag_a = imag(A(k));
        real_c = real(C(k));
        imag_c = imag(C(k));
        num{n} = [2*real_c, -2*(real_a*real_c+imag_a*imag_c)];
        den{n} = [1, -2*real_a, real_a^2+imag_a^2];
        k = k + 2;
    end
end
den = den(1:n);
num = num(1:n);
```

Build Simulink Model of Backplane

Build a Simulink model of the backplane using the Laplace Transform transfer functions. Then, connect a random source to the input of the backplane and a scope to its input and output.

```
modelname = fliplr(strtok(fliplr(tempname), filesep));
simulink_rfmodel_build_rational_system_helper(modelname , numel(num))
simulink_rfmodel_add_source_sink_helper(modelname)
```

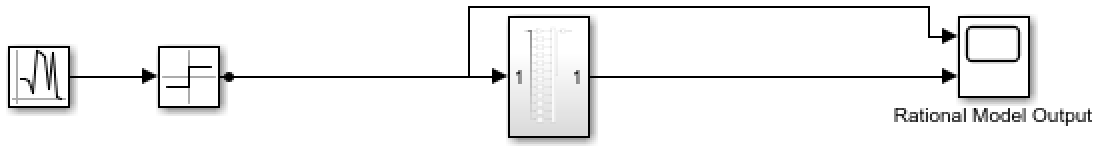
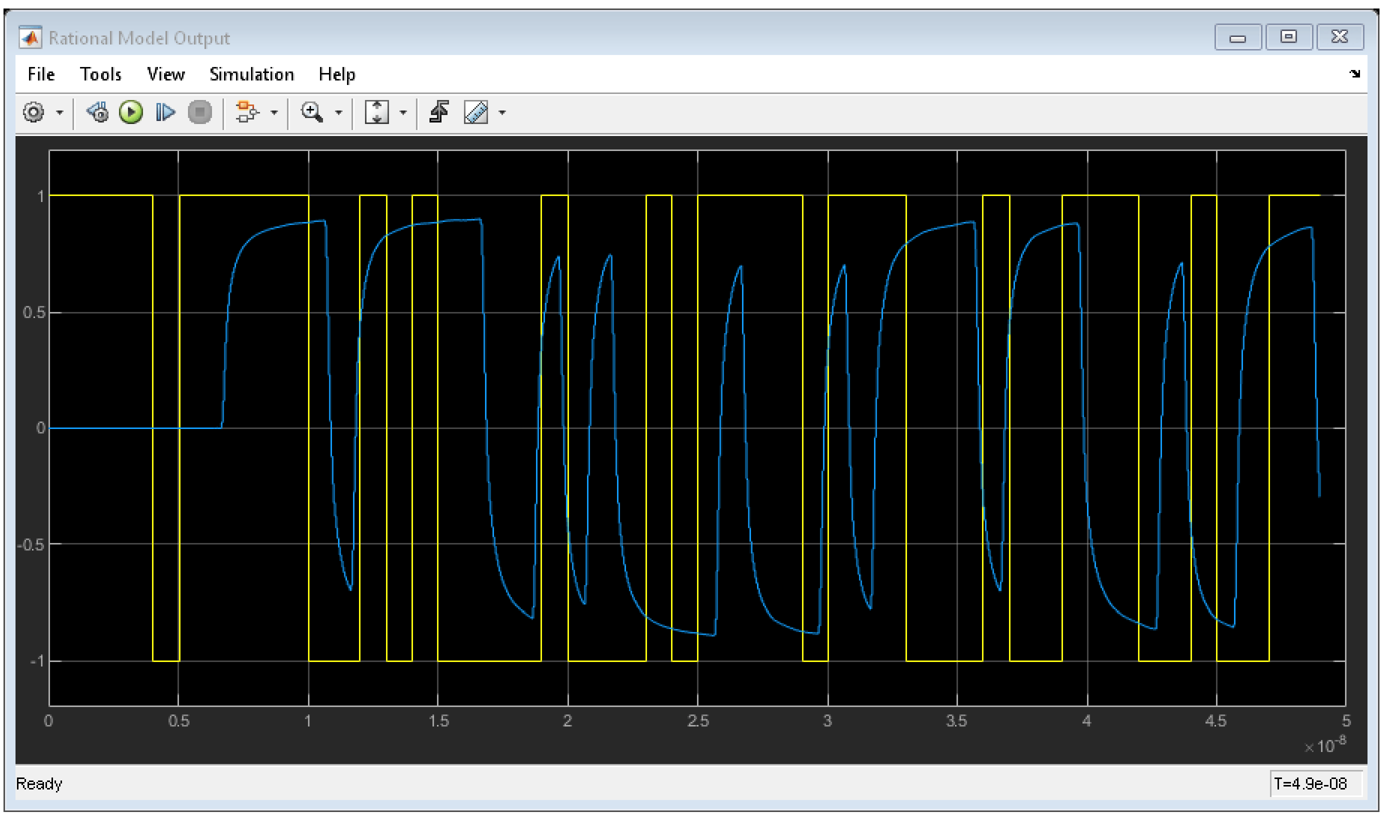


Figure 1. Simulink model for a rational function

Simulate Simulink Model of Rational Function

When you simulate the model, the Scope shows the impact of the differential backplane on the random input signal.

```
set_param([modelname,'/Rational Model Output'], 'Open', 'on')
sc = get_param([modelname,'/Rational Model Output'],'ScopeConfiguration');
sc.Position = [200, 216, 901, 442];
sim(modelname);
```



Close Model

```
close_system(modelname, 0)
```

See Also

More About

- “Export Verilog-A module from Rational Function” on page 6-69
- “Use S-parameters with Port Reduction” on page 6-51

Export Verilog-A module from Rational Function

This example shows how to use RF Toolbox™ functions to generate a Verilog-A module that models the high-level behavior of a high-speed backplane. First, it reads the single-ended 4-port S-parameters for a differential high-speed backplane and converts them to 2-port differential S-parameters. Then, it computes the transfer function of the differential circuit and fits a rational function to the transfer function. Next, the example exports a Verilog-A module that describes the model. Finally, it plots the unit step response of the generated Verilog-A module in a third-party circuit simulation tool.

Use Rational Function Object to Describe High-Level Behavior of High-Speed Backplane

Read a Touchstone® data file, `default.s4p`, into an `sparameters` object. The parameters in this data file are the 50-ohm S-parameters of a single-ended 4-port passive circuit, measured at 1496 frequencies ranging from 50 MHz to 15 GHz. Then, extract the single-ended 4-port S-parameters from the data stored in the `Parameters` property of the `sparameters` object, use the `s2sdd` function to convert them to differential 2-port S-parameters, and use the `s2tf` function to compute the transfer function of the differential circuit. Then, use the `rationalfit` function to generate an `rfmodel.rational` object that describes the high-level behavior of this high-speed backplane. The `rfmodel.rational` object is a rational function object that expresses the circuit's transfer function in closed form using poles, residues, and other parameters, as described in the `rationalfit` reference page.

```
filename = 'default.s4p';
backplane = sparameters(filename);
data = backplane.Parameters;
freq = backplane.Frequencies;
z0 = backplane.Impedance;
```

Convert to 2-port differential S-parameters.

```
diffdata = s2sdd(data);
diffz0 = 2*z0;
difftf = s2tf(diffdata,diffz0,diffz0,diffz0);
```

Fit the differential transfer function into a rational function.

```
fittol = -30;           % Rational fitting tolerance in dB
delayfactor = 0.9;      % Delay factor
rationalfunc = rationalfit(freq,difftf,fittol,'DelayFactor',delayfactor)
```

```
rationalfunc = 
   rfmodel.rational with properties:

        A: [20x1 double]
        C: [20x1 double]
        D: 0
    Delay: 6.0172e-09
     Name: 'Rational Function'
```

Export Rational Function Object as Verilog-A Module

Use the `writeva` method of the `rfmodel.rational` object to export the rational function object as a Verilog-A module, called `samplepassive1`, that describes the rational model. The input and output nets of samplepassive1 are called line_in and line_out. The predefined Verilog-A discipline, electrical, describes the attributes of these nets. The format of numeric values, such as the Laplace transform numerator and denominator coefficients, is %12.10e. The electrical discipline is defined in the file disciplines.vams, which is included in the beginning of the samplepassive1.va file.

```
workingdir = tempname;
mkdir(workingdir)
writeva(rationalfunc, fullfile(workingdir,'samplepassive1'), ...
    'line_in', 'line_out', 'electrical', '%12.10e', 'disciplines.vams');

type(fullfile(workingdir,'samplepassive1.va'));
```

```
// Module: samplepassive1

// Generated by MATLAB(R) 9.14 and the RF Toolbox 4.5.

// Generated on: 03-Mar-2023 22:52:41

`include "disciplines.vams"

module samplepassive1(line_in, line_out);
 electrical line_in, line_out;
 electrical node1;

 real nn1[0:1], nn2[0:1], nn3[0:1], nn4[0:1], nn5[0:1], nn6[0:1], nn7[0:1], nn8[0:1], nn9[0:1], n
 real dd1[0:2], dd2[0:2], dd3[0:2], dd4[0:2], dd5[0:2], dd6[0:2], dd7[0:2], dd8[0:2], dd9[0:2], d

 analog begin

   @(initial_step) begin
     nn1[0] = -3.8392614832e+18;
     nn1[1] = 5.2046393014e+07;
     dd1[0] = 2.8312609831e+21;
     dd1[1] = 3.5124823781e+09;
     dd1[2] = 1.0000000000e+00;
     nn2[0] = -2.0838483814e+19;
     nn2[1] = 5.3487174017e+08;
     dd2[0] = 1.8020362314e+21;
     dd2[1] = 7.8266367089e+09;
     dd2[2] = 1.0000000000e+00;
     nn3[0] = 1.7726270794e+19;
     nn3[1] = 2.5185716022e+09;
     dd3[0] = 1.2157471895e+21;
     dd3[1] = 8.1132784895e+09;
     dd3[2] = 1.0000000000e+00;
     nn4[0] = 2.3112282793e+20;
     nn4[1] = 9.2690544437e+08;
     dd4[0] = 7.9582429152e+20;
     dd4[1] = 1.1379108659e+10;
     dd4[2] = 1.0000000000e+00;
     nn5[0] = 8.9321469721e+19;
     nn5[1] = -1.4945928109e+10;
     dd5[0] = 4.1473706594e+20;
     dd5[1] = 1.1346735824e+10;
     dd5[2] = 1.0000000000e+00;
     nn6[0] = -3.5180951909e+20;
     nn6[1] = -1.9895507212e+10;
```

```
    dd6[0] = 1.9080843811e+20;
    dd6[1] = 1.0434555792e+10;
    dd6[2] = 1.0000000000e+00;
    nn7[0] = -1.0593240107e+20;
    nn7[1] = 1.9248932577e+10;
    dd7[0] = 6.1152960549e+19;
    dd7[1] = 1.0001203231e+10;
    dd7[2] = 1.0000000000e+00;
    nn8[0] = 5.4441539403e+16;
    nn8[1] = -9.7818749687e+06;
    dd8[0] = 4.3821946493e+19;
    dd8[1] = 6.6700188623e+08;
    dd8[2] = 1.0000000000e+00;
    nn9[0] = 2.2556903052e+16;
    nn9[1] = 7.9711163023e+06;
    dd9[0] = 2.1228807651e+19;
    dd9[1] = 4.9531801417e+08;
    dd9[2] = 1.0000000000e+00;
    nn10[0] = 1.1592988960e+10;
    dd10[0] = 3.0829914556e+09;
    dd10[1] = 1.0000000000e+00;
    nn11[0] = 1.2852839051e+08;
    dd11[0] = 5.9779845807e+08;
    dd11[1] = 1.0000000000e+00;
  end

  V(node1) <+ laplace_nd(V(line_in), nn1, dd1);
  V(node1) <+ laplace_nd(V(line_in), nn2, dd2);
  V(node1) <+ laplace_nd(V(line_in), nn3, dd3);
  V(node1) <+ laplace_nd(V(line_in), nn4, dd4);
  V(node1) <+ laplace_nd(V(line_in), nn5, dd5);
  V(node1) <+ laplace_nd(V(line_in), nn6, dd6);
  V(node1) <+ laplace_nd(V(line_in), nn7, dd7);
  V(node1) <+ laplace_nd(V(line_in), nn8, dd8);
  V(node1) <+ laplace_nd(V(line_in), nn9, dd9);
  V(node1) <+ laplace_nd(V(line_in), nn10, dd10);
  V(node1) <+ laplace_nd(V(line_in), nn11, dd11);
  V(line_out) <+ absdelay(V(node1), 6.0171901584e-09);
 end
endmodule
```

Plot Unit Step Response of Generated Verilog-A Module

Many third-party circuit simulation tools support the Verilog-A standard. These tools simulate standalone components defined by Verilog-A modules and circuits that contain these components. The following figure shows the unit step response of the samplepassive1 module. The figure was generated with a third-party circuit simulation tool.

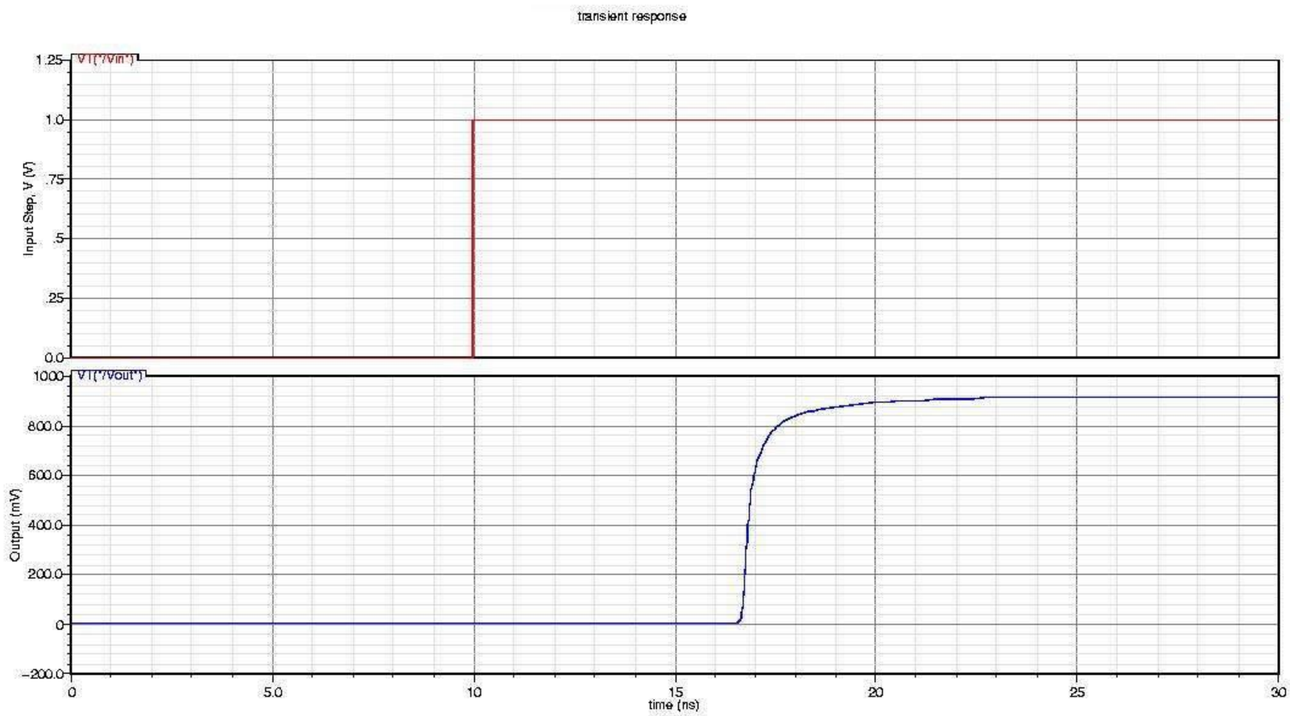


Figure 1: The unit step response.

```
delete(fullfile(workingdir,'samplepassive1.va'));
rmdir(workingdir)
```

See Also

More About

- "Build Simulink Model from Rational Function" on page 6-65
- "Use S-parameters with Port Reduction" on page 6-51

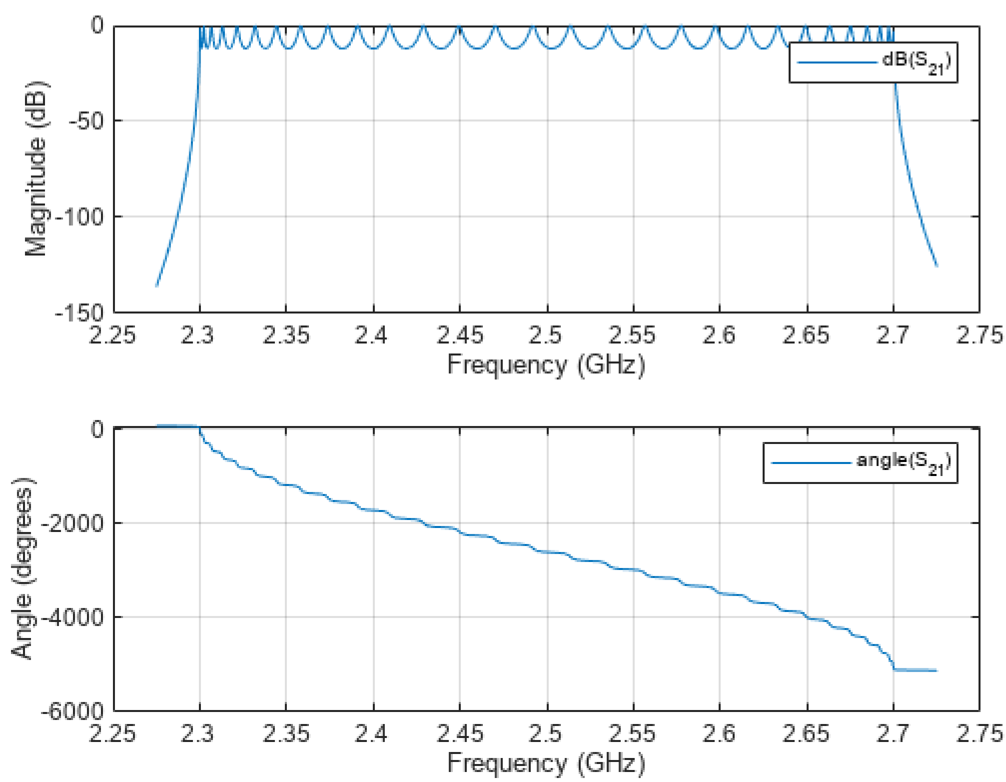
Using 'NPoles' Parameter With rationalfit

This example shows how to use the 'NPoles' parameter to improve the quality of the output of `rationalfit`. By default, the `rationalfit` function uses 48 or fewer poles to find the rational function that best matches the data. If 48 poles is not enough, it may be advantageous to change the range of the number of poles used by `rationalfit`.

First, read in the bandpass filter data contained in the file `npoles_bandpass_example.s2p`, and plot the `S21` data. Next, use the `rationalfit` function to fit a rational function to the `S21` data, with the 'NPoles' parameter set to its default value, and visually compare the results to the original data. Lastly, use `rationalfit` again, this time specifying a larger number of poles, and see if the result improves.

Read and Visualize Data

```
S = sparameters('npoles_bandpass_example.s2p');
figure
subplot(2,1,1)
rfplot(S,2,1,'db')
subplot(2,1,2)
rfplot(S,2,1,'angle')
```



Analyze Output of rationalfit When Using Default Value for 'NPoles'

Use the `rfparam` function to extract the `S21` values, and then call `rationalfit`.

```
s21 = rfparam(S,2,1);
datafreq = S.Frequencies;
defaultfit = rationalfit(datafreq,s21);
```

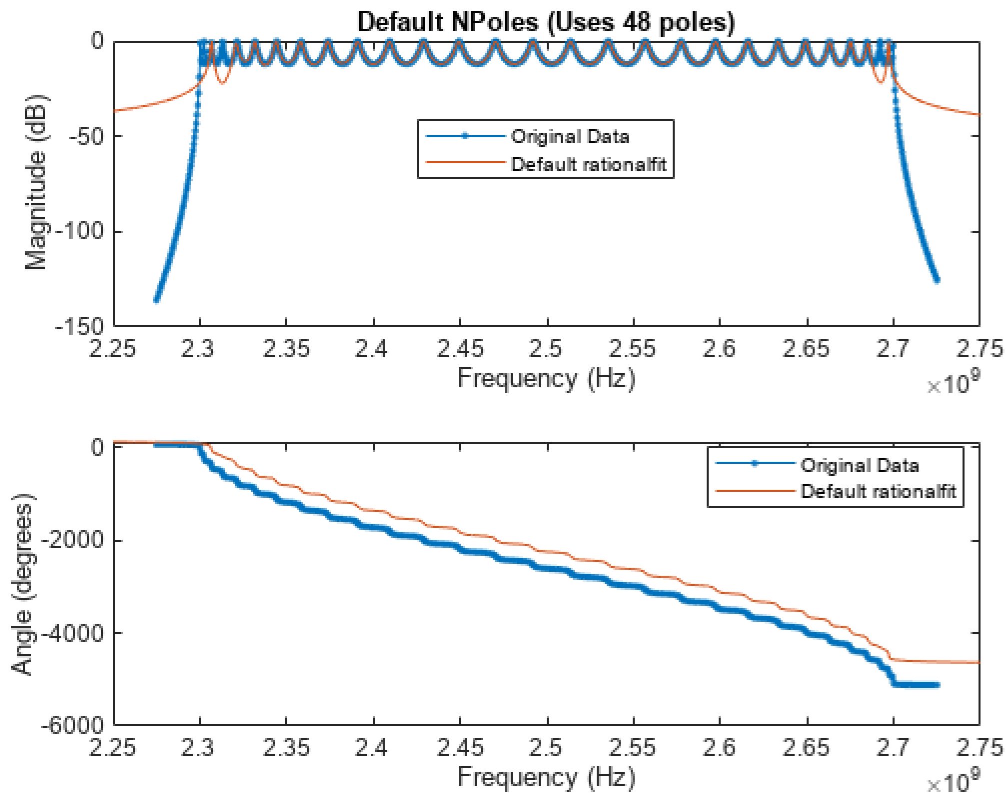
```
Warning: Achieved only -13.0 dB accuracy with 48 poles, not -40.0 dB.  Consider specifying a larg
```

Use the `freqresp` function to calculate the response of the output of `rationalfit`.

```
respfreq = 2.25e9:2e5:2.75e9;
defaultresp = freqresp(defaultfit,respfreq);
```

Compare the original data against the frequency response of the default rational function calculated by `rationalfit`.

```
subplot(2,1,1)
plot(datafreq,20*log10(abs(s21)),'.-')
hold on
plot(respfreq,20*log10(abs(defaultresp)))
hold off
xlabel('Frequency (Hz)')
ylabel('Magnitude (dB)')
defaultnpoles = numel(defaultfit.A);
defaultstr = ['Default NPoles (Uses ',num2str(defaultnpoles),' poles)'];
title(defaultstr)
legend('Original Data','Default rationalfit','Location','best')
subplot(2,1,2)
plot(datafreq,unwrap(angle(s21))*180/pi,'.-')
hold on
plot(respfreq,unwrap(angle(defaultresp))*180/pi)
hold off
xlabel('Frequency (Hz)')
ylabel('Angle (degrees)')
legend('Original Data','Default rationalfit','Location','best')
```



Analyzing how well the output of `rationalfit` matches the original data, it appears that while the default values of `rationalfit` do a reasonably good job in the center of the bandpass region, the fit is poor on the edges of the bandpass region. It is possible that using a more complex rational function will achieve a better fit.

Analyze Output of rationalfit When Using Custom Value for 'NPoles'

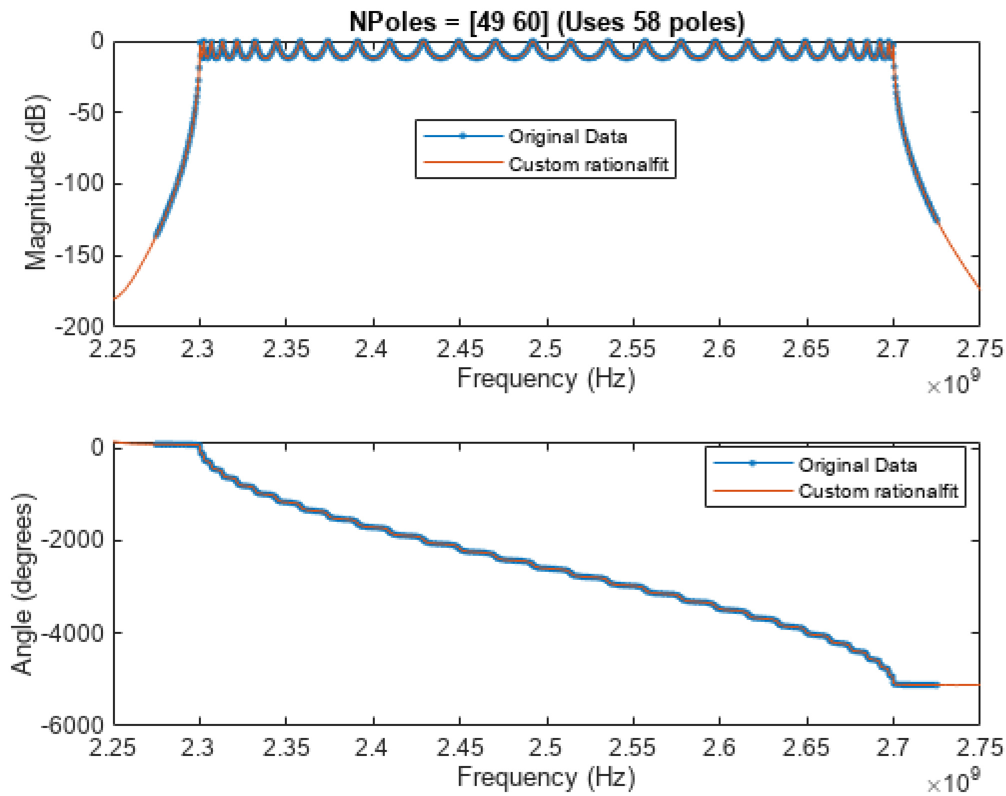
Fit the original `S21` data, but this time, instruct `rationalfit` to use between 49 and 60 poles using the 'NPoles' parameter.

```
customfit = rationalfit(datafreq,s21,'NPoles',[49 60]);
customresp = freqresp(customfit,respfreq);
```

Compare the original data against the frequency response of the custom rational function calculated by `rationalfit`.

```
figure
subplot(2,1,1)
plot(datafreq,20*log10(abs(s21)),'.-')
hold on
plot(respfreq,20*log10(abs(customresp)))
hold off
xlabel('Frequency (Hz)')
ylabel('Magnitude (dB)')
customnpoles = numel(customfit.A);
customstr = ['NPoles = [49 60] (Uses ',num2str(customnpoles),' poles)'];
title(customstr)
```

```
legend('Original Data','Custom rationalfit','Location','best')
subplot(2,1,2)
plot(datafreq,unwrap(angle(s21))*180/pi,'.-')
hold on
plot(respfreq,unwrap(angle(customresp))*180/pi)
hold off
xlabel('Frequency (Hz)')
ylabel('Angle (degrees)')
legend('Original Data','Custom rationalfit','Location','best')
```



The fit using a larger number of poles is clearly more precise.

See Also

More About

- “Using 'Weight' Parameter With rationalfit” on page 6-77
- “Using 'DelayFactor' Parameter With rationalfit” on page 6-83
- “Using Rational Object to Fit S-Parameters” on page 6-215

Using 'Weight' Parameter With rationalfit

This example shows how to use the `'Weight'` parameter to improve the quality of the output of `rationalfit`.

By default, the `rationalfit` function minimizes the absolute error between the data and the rational function, treating all data points equally. If you want to emphasize some of the data points more than the others, use the `'Weight'` parameter.

For example, If the magnitude of the input data has a large dynamic range, it is often useful to be more concerned with the relative error at each data point, rather than the absolute error at each data point, so that the data points with relatively smaller magnitudes are fit accurately. The common way to do this is to set the 'Weight' parameter to `1./abs(data)`.

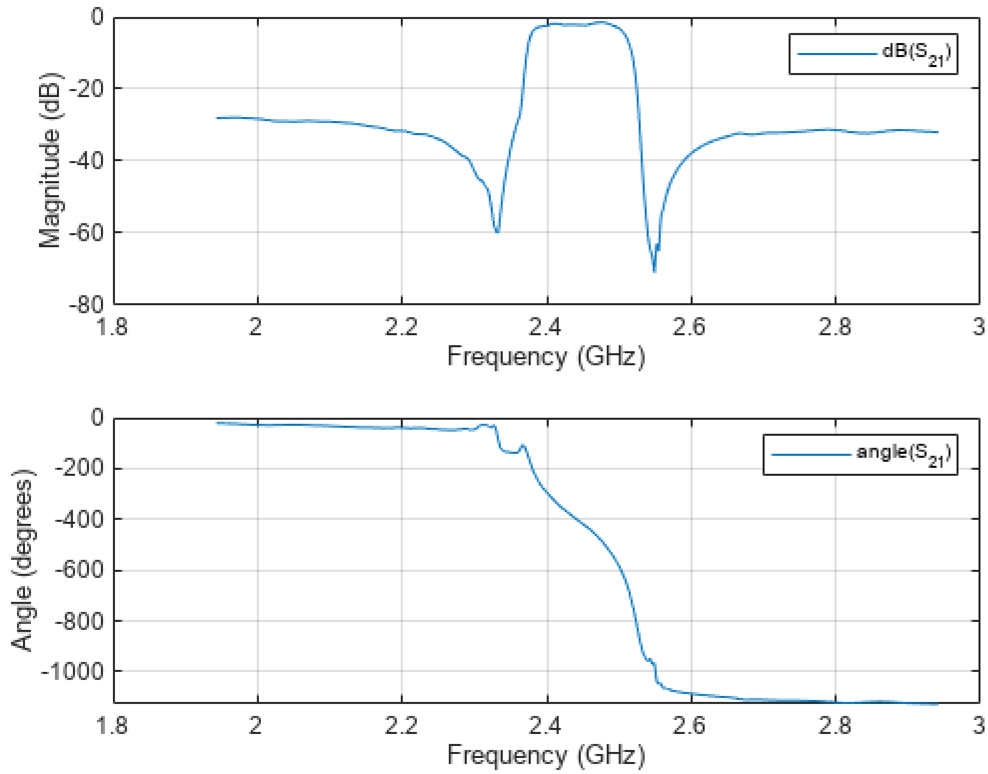
To put the example above in practice, follow the steps below.

- Read in the SAW filter data contained in the file `sawfilter.s2p`, and plot the `S21` data.
- Use the `rationalfit` function to fit a rational function to the `S21` data, with the `'Weight'` parameter set to its default value, and visually compare the results to the original data.
- Use `rationalfit` again, this time specifying the `'Weight'` parameter to be `1./abs(S21)`, and see if the result improves.

Read and Visualize Data

Read and visualize SAW filter S-parameters data.

```
S = sparameters('sawfilter.s2p');
figure
subplot(2,1,1)
rfplot(S,2,1,'db')
subplot(2,1,2)
rfplot(S,2,1,'angle')
```



Analyze Output of `rationalfit` When Using Default Value for `'Weight'`

Use the `rfparam` function to extract the `S21` values, and then call `rationalfit`.

```
s21 = rfparam(S,2,1);
datafreq = S.Frequencies;
defaultfit = rationalfit(datafreq,s21);
```

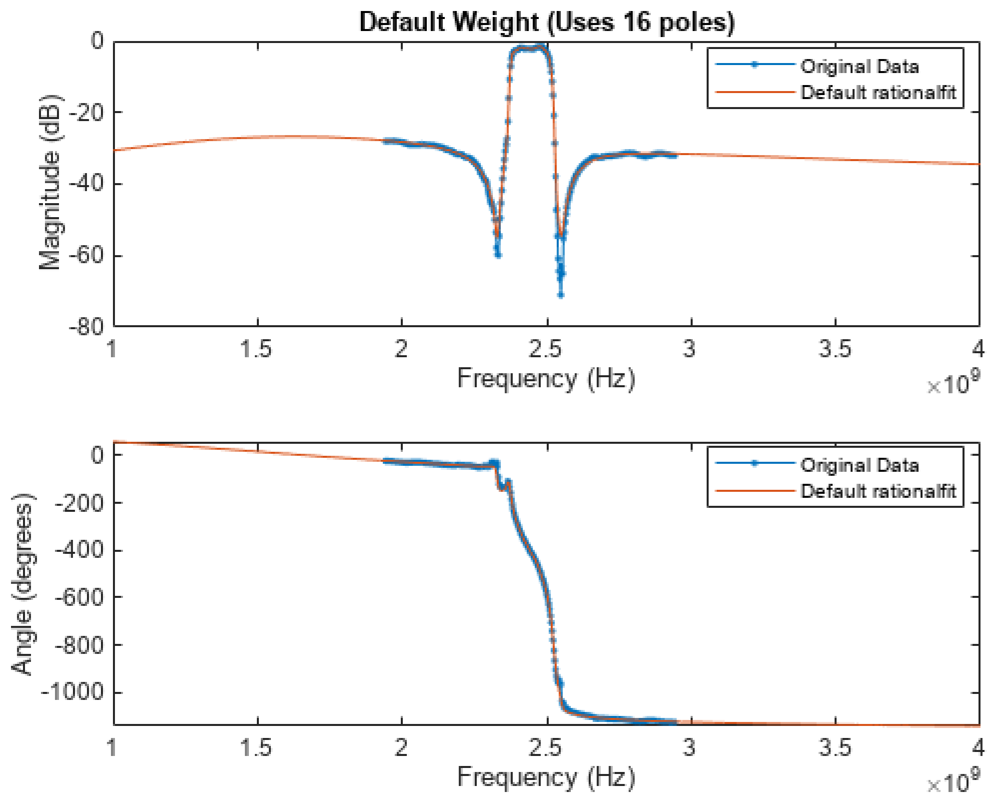
Use the `freqresp` function to calculate the response of the output of `rationalfit`.

```
respfreq = 1e9:1.5e6:4e9;
defaultresp = freqresp(defaultfit,respfreq);
```

Compare the original data against the frequency response of the default rational function calculated by `rationalfit`.

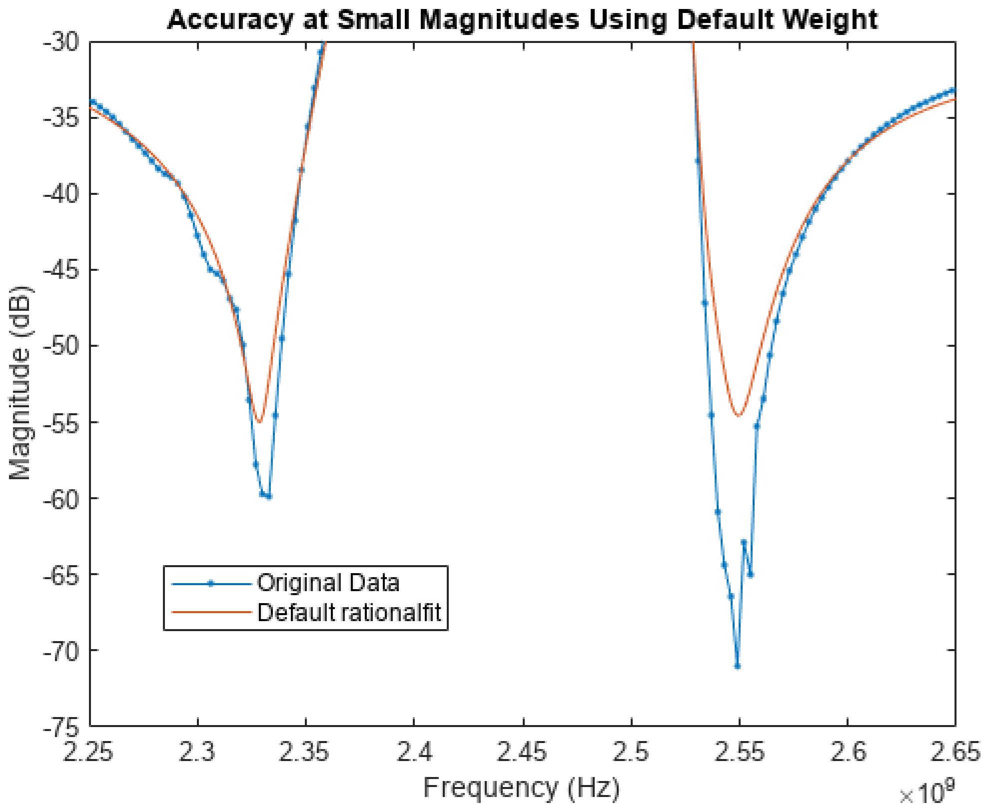
```
subplot(2,1,1)
plot(datafreq,20*log10(abs(s21)),'.-')
hold on
plot(respfreq,20*log10(abs(defaultresp)))
hold off
xlabel('Frequency (Hz)')
ylabel('Magnitude (dB)')
defaultnpoles = numel(defaultfit.A);
defaultstr = ['Default Weight (Uses ',num2str(defaultnpoles),' poles)'];
title(defaultstr)
legend('Original Data','Default rationalfit','Location','best')
subplot(2,1,2)
```

```
plot(datafreq,unwrap(angle(s21))*180/pi,'.-')
hold on
plot(respfreq,unwrap(angle(defaultresp))*180/pi)
hold off
xlabel('Frequency (Hz)')
ylabel('Angle (degrees)')
legend('Original Data','Default rationalfit','Location','best')
```



While the output of `rationalfit` is not awful, it does not match the regions in the data that are very small in magnitude.

```
figure
plot(datafreq,20*log10(abs(s21)),'.-')
hold on
plot(respfreq,20*log10(abs(defaultresp)))
hold off
axis([2.25e9 2.65e9 -75 -30])
xlabel('Frequency (Hz)')
ylabel('Magnitude (dB)')
title('Accuracy at Small Magnitudes Using Default Weight')
legend('Original Data','Default rationalfit','Location','best')
```



Using the 'Weight' parameter to make that data relatively more important can help the accuracy of the fit.

Analyze Output of `rationalfit` When Using Custom Value for `'Weight'`

By using a 'Weight' of `1./abs(s21)`, `rationalfit` minimizes the relative error of the system, instead of the absolute error of the system.

```
customfit = rationalfit(datafreq,s21,'Weight',1./abs(s21));
```

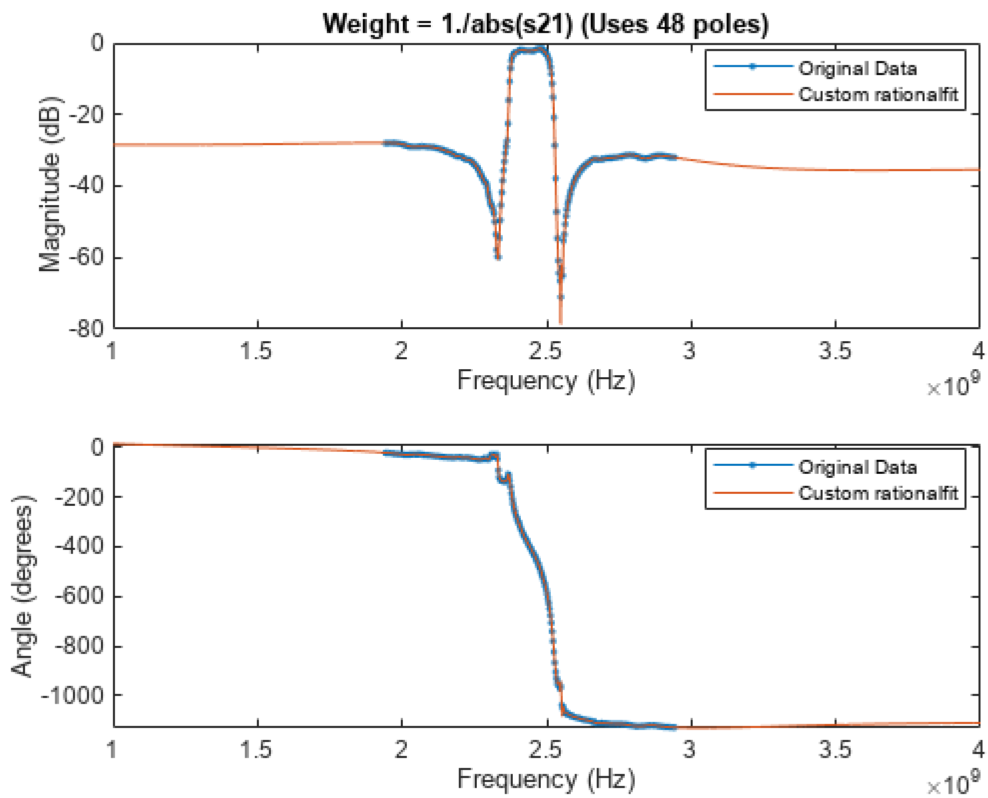
```
Warning: Achieved only -39.7 dB accuracy with 48 poles, not -40.0 dB.  Consider specifying a larg
```

```
customresp = freqresp(customfit,respfreq);
```

Compare the original data against the frequency response of the custom rational function calculated by `rationalfit`.

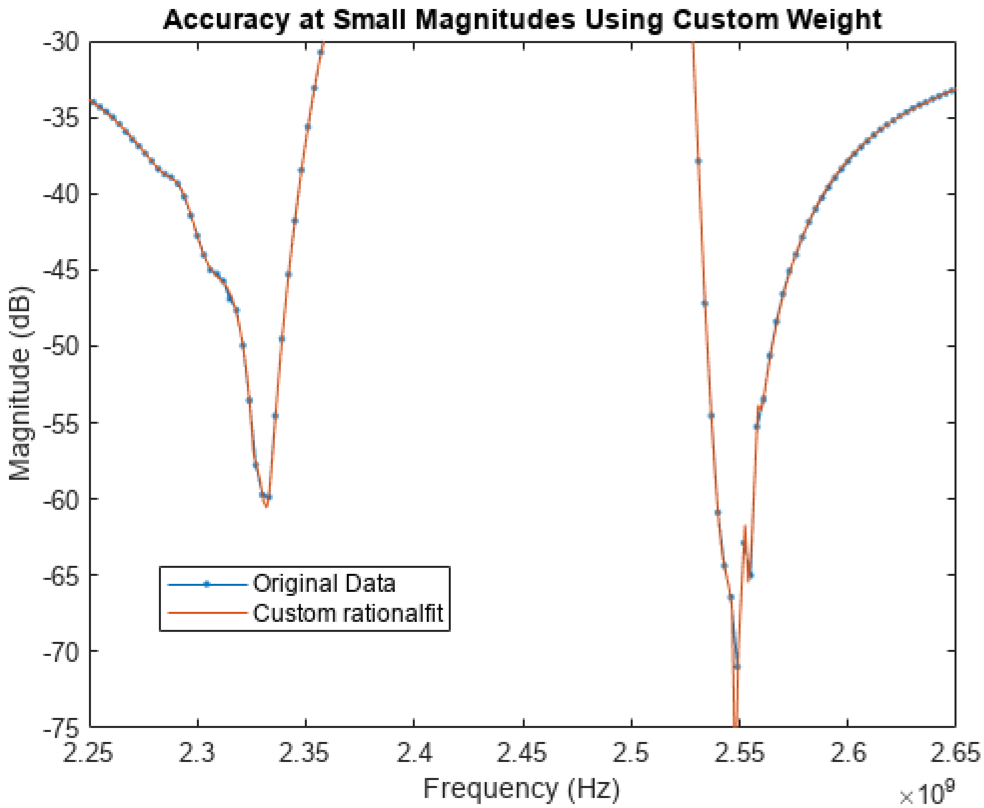
```
figure
subplot(2,1,1)
plot(datafreq,20*log10(abs(s21)),'.-')
hold on
plot(respfreq,20*log10(abs(customresp)))
hold off
xlabel('Frequency (Hz)')
ylabel('Magnitude (dB)')
customnpoles = numel(customfit.A);
customstr = ['Weight = 1./abs(s21) (Uses ',num2str(customnpoles),' poles)'];
title(customstr)
```

```
legend('Original Data','Custom rationalfit','Location','best')
subplot(2,1,2)
plot(datafreq,unwrap(angle(s21))*180/pi,'.-')
hold on
plot(respfreq,unwrap(angle(customresp))*180/pi)
hold off
xlabel('Frequency (Hz)')
ylabel('Angle (degrees)')
legend('Original Data','Custom rationalfit','Location','best')
```



The plot shows that the custom `'Weight'` parameter created a better fit for the data points with smaller magnitudes.

```
figure
plot(datafreq,20*log10(abs(s21)),'.-')
hold on
plot(respfreq,20*log10(abs(customresp)))
hold off
axis([2.25e9 2.65e9 -75 -30])
xlabel('Frequency (Hz)')
ylabel('Magnitude (dB)')
title('Accuracy at Small Magnitudes Using Custom Weight')
legend('Original Data','Custom rationalfit','Location','best')
```



See Also

More About

- “Using 'DelayFactor' Parameter With rationalfit” on page 6-83
- “Using Rational Object to Fit S-Parameters” on page 6-215
- “Using Rational Object to Fit S-Parameters” on page 6-215

Using 'DelayFactor' Parameter With rationalfit

This example shows how to use the 'DelayFactor' parameter to improve the quality of the output of `rationalfit`.

The `rationalfit` function selects a rational function that matches frequency domain data. If that data contains a significant "time delay", which would present itself as a phase shift in the frequency domain, then it might be very difficult to fit using a reasonable number of poles.

In these cases, when the input data contains a large negative slope (i.e. data with a large enough time delay), we can ask `rationalfit` to first remove some of the delay from the data, and then find a rational function that best fits the remaining "undelayed" data. The `rationalfit` function accounts for the removed delay by storing it within the 'Delay' parameter of the output. By default, `rationalfit` does not remove any delay from the data.

First, create differential transfer function data from 4-port backplane S-parameters. Next, attempt to fit the data using the default settings of the `rationalfit` function. Lastly, use the 'DelayFactor' parameter to improve the accuracy of the output of `rationalfit`.

Create Transfer Function

Read in the 4-port backplane S-parameter data from 'default.s4p'.

```
S = sparameters('default.s4p');
fourportdata = S.Parameters;
freq = S.Frequencies;
fourportZ0 = S.Impedance;
```

Convert 4-port single ended S-parameters into 2-port differential S-parameters

```
diffdata = s2sdd(fourportdata);
diffZ0 = 2*fourportZ0;
```

Create a transfer function from the differential 2-port data

```
tfdata = s2tf(diffdata,diffZ0,diffZ0,diffZ0);
```

Analyze Output of rationalfit When Using Default Value for 'DelayFactor'

Use the `freqresp` function to calculate the response of the output of `rationalfit`.

```
defaultfit = rationalfit(freq,tfdata)
```

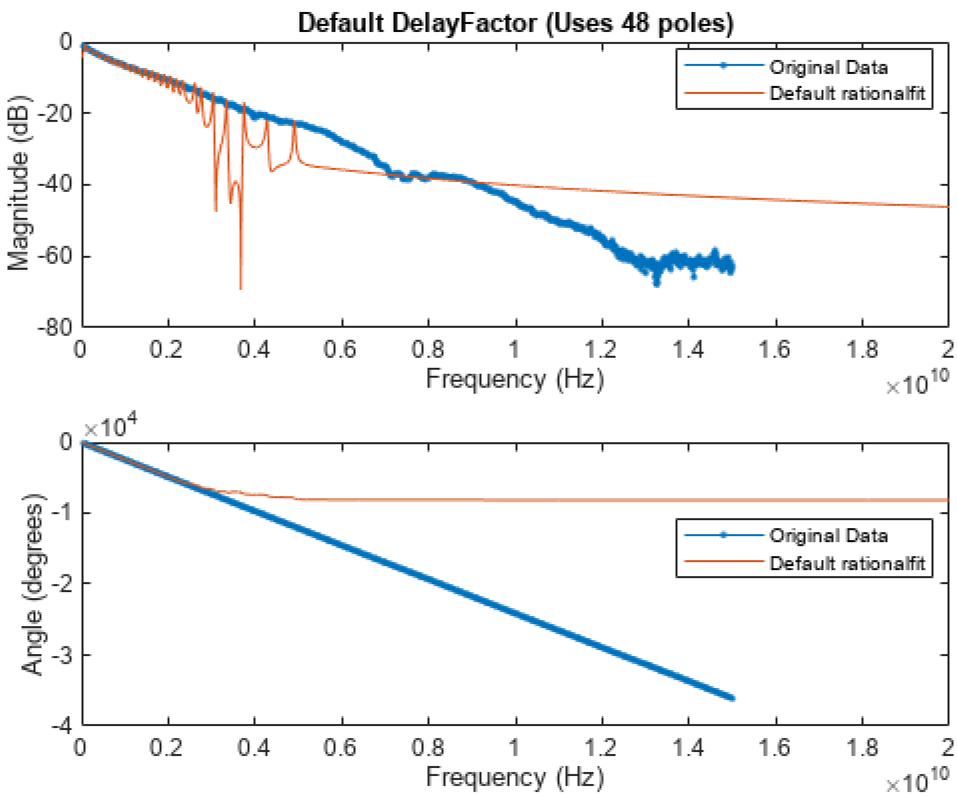
```
Warning: Achieved only -10.2 dB accuracy with 48 poles, not -40.0 dB.  Consider specifying a larg

defaultfit =
   rfmodel.rational with properties:

        A: [48x1 double]
        C: [48x1 double]
        D: 0
    Delay: 0
     Name: 'Rational Function'
```

```
respfreq = 0:4e6:20e9;
defaultresp = freqresp(defaultfit,respfreq);
```

Note that the 'Delay' parameter is zero (no delay removed from the data).

Plot the original data vs. the default output of `rationalfit`.

```
figure
subplot(2,1,1)
tfdataDB = 20*log10(abs(tfdata));
plot(freq,tfdataDB,'.-')
hold on
plot(respfreq,20*log10(abs(defaultresp)))
hold off
xlabel('Frequency (Hz)')
ylabel('Magnitude (dB)')
defaultnpoles = numel(defaultfit.A);
defstr = ['Default DelayFactor (Uses ',num2str(defaultnpoles),' poles)'];
title(defstr)
legend('Original Data','Default rationalfit','Location','best')
subplot(2,1,2)
tfdataphase = 180*unwrap(angle(tfdata))/pi;
plot(freq,tfdataphase,'.-')
hold on
plot(respfreq,180*unwrap(angle(defaultresp))/pi)
hold off
xlabel('Frequency (Hz)')
ylabel('Angle (degrees)')
legend('Original Data','Default rationalfit','Location','best')
```



Note that the results when using the default settings of `rationalfit` are poor. Because the phase of the original data has a very large negative slope, it may be possible to improve the accuracy of the rational function by using the 'DelayFactor' parameter.

Analyze Output of rationalfit When Using Custom Value for 'DelayFactor'

'DelayFactor' must be set to a value between 0 and 1. Choosing which value is an exercise in trial and error. For some data sets (those whose phase has an overall upward slope), changing the value of 'DelayFactor' will have no effect on the outcome.

Holding all other possible parameters of `rationalfit` constant, 0.98 is found to create a good fit.

```
customfit = rationalfit(freq,tfdata,'DelayFactor',0.98)
```

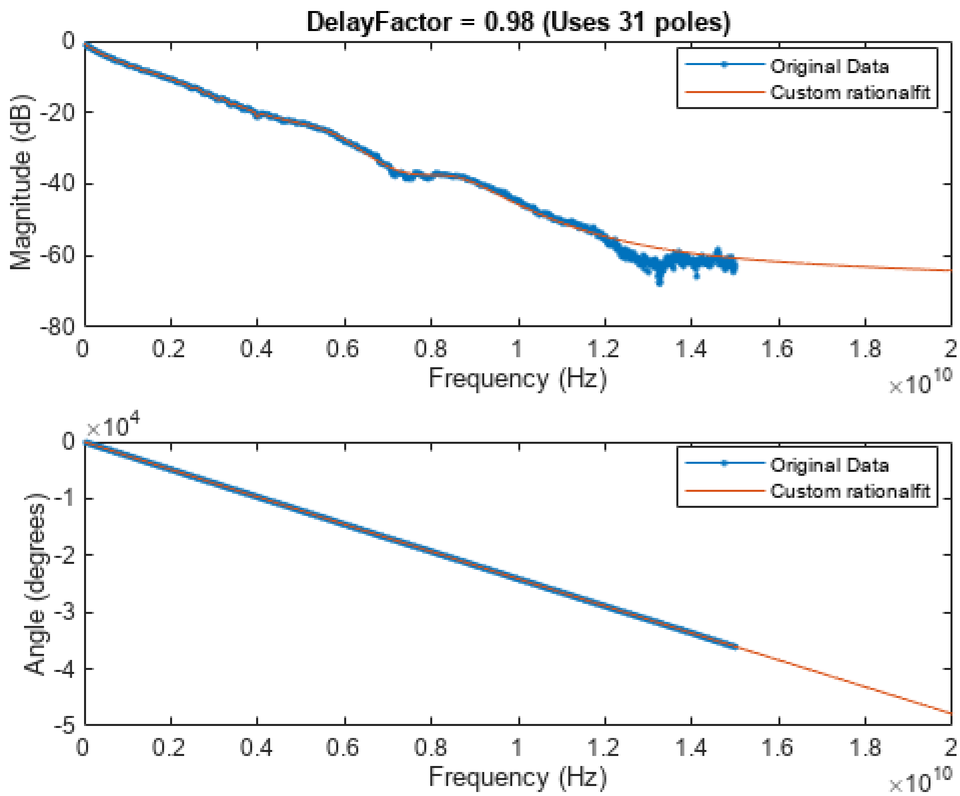
```
customfit =
   rfmodel.rational with properties:

        A: [31x1 double]
        C: [31x1 double]
        D: 0
    Delay: 6.5521e-09
     Name: 'Rational Function'
```

```
customresp = freqresp(customfit,respfreq);
```

Note that the 'Delay' parameter is not zero (`rationalfit` removed some delay from the data).

Plot the original data vs. the custom output of `rationalfit`.

```
subplot(2,1,1)
plot(freq,tfdataDB,'.-')
hold on
plot(respfreq,20*log10(abs(customresp)))
hold off
xlabel('Frequency (Hz)')
ylabel('Magnitude (dB)')
customnpoles = numel(customfit.A);
customstr = ['DelayFactor = 0.98 (Uses ',num2str(customnpoles),' poles)'];
title(customstr)
legend('Original Data','Custom rationalfit','Location','best')
subplot(2,1,2)
plot(freq,tfdataphase,'.-')
hold on
plot(respfreq,180*unwrap(angle(customresp))/pi)
hold off
xlabel('Frequency (Hz)')
ylabel('Angle (degrees)')
legend('Original Data','Custom rationalfit','Location','best')
```



The rational function created by using a custom value for 'DelayFactor' is much more accurate, and uses fewer poles.

See Also

More About

- “Using Rational Object to Fit S-Parameters” on page 6-215
- “Using 'NPoles' Parameter With rationalfit” on page 6-73
- “Using 'Weight' Parameter With rationalfit” on page 6-77

Data Analysis on S-Parameters of RF Data Files

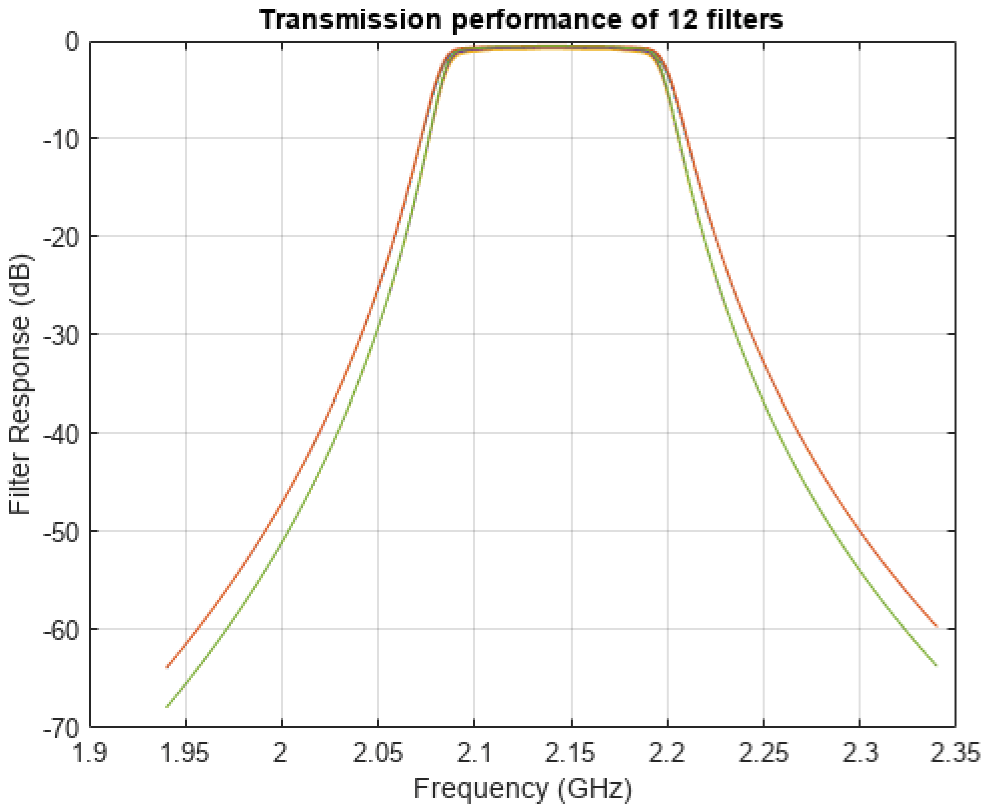
This example shows how to perform statistical analysis on a set of S-parameter data files using magnitude, mean, and standard deviation (STD).

First, read twelve S-parameter files, where these files represent the twelve similar RF filters into the MATLAB® workspace and plot them. Next, plot and analyze the passband response of these filters to ensure they meet statistical norms.

Read S-Parameters from Filter Data Files

Use built-in RF Toolbox™ functions for reading a set of S-Parameter data files. For each filter plot the S21 dB values. The names of the files are AWS_Filter_1.s2p through AWS_Filter_12.s2p. These files represent 12 passband filters with similar specifications.

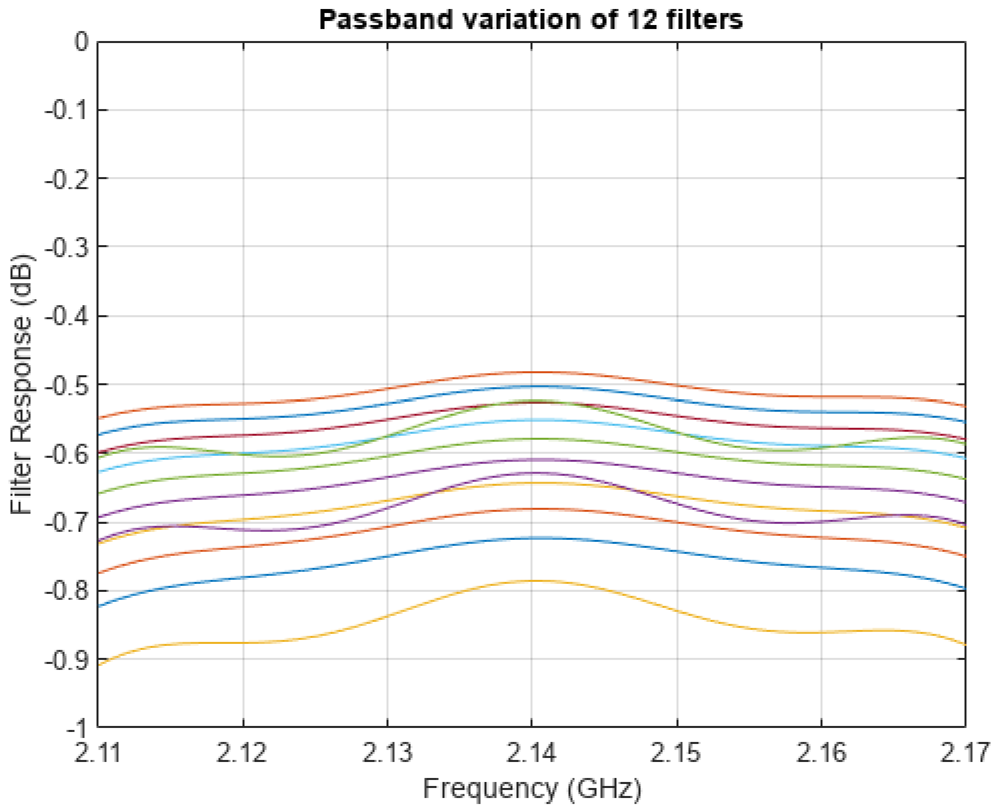
```
numfiles = 12;
filename = "AWS_Filter_"+(1:numfiles)+".s2p";    % Construct filenames
S = sparameters(filename(1));                    % Read file #1 for initial set-up
freq = S.Frequencies;                            % Frequency values are the same for all files
numfreq = numel(freq);                           % Number of frequency points
s21_data = zeros(numfreq,numfiles);              % Preallocate for speed
s21_groupdelay = zeros(numfreq,numfiles);        % Preallocate for speed

% Read Touchstone files
for n = 1:numfiles
    S = sparameters(filename(n));
    s21 = rfparam(S,2,1);
    s21_data(:,n) = s21;
    s21_groupdelay(:,n) = groupdelay(S,freq,2,1);
end
s21_db = 20*log10(abs(s21_data));

figure
plot(freq/1e9,s21_db)
xlabel('Frequency (GHz)')
ylabel('Filter Response (dB)')
title('Transmission performance of 12 filters')
axis on
grid on
```



Filter Passband Visualization

In this section, find, store, and plot the S21 data from the AWS downlink band (2.11 - 2.17 GHz).

```
idx = (freq >= 2.11e9) & (freq <= 2.17e9);
s21_pass_data = s21_data(idx,:);
s21_pass_db = s21_db(idx,:);
freq_pass_ghz = freq(idx)/1e9; % Normalize to GHz

plot(freq_pass_ghz,s21_pass_db)
xlabel('Frequency (GHz)')
ylabel('Filter Response (dB)')
title('Passband variation of 12 filters')
axis([min(freq_pass_ghz) max(freq_pass_ghz) -1 0])
grid on
```



Basic Statistical Analysis of S21 Data

To determine whether the data follows a normal distribution and if there is an outlier, perform statistical analysis on the magnitude and group delay of all passband S21 data sets.

```
abs_S21_pass_freq = abs(s21_pass_data);
```

Calculate the mean and the STD of the magnitude of the entire passband S21 data set.

```
mean_abs_S21 = mean(abs_S21_pass_freq,'all')
```

```
mean_abs_S21 = 0.9289
```

```
std_abs_S21 = std(abs_S21_pass_freq(:))
```

```
std_abs_S21 = 0.0104
```

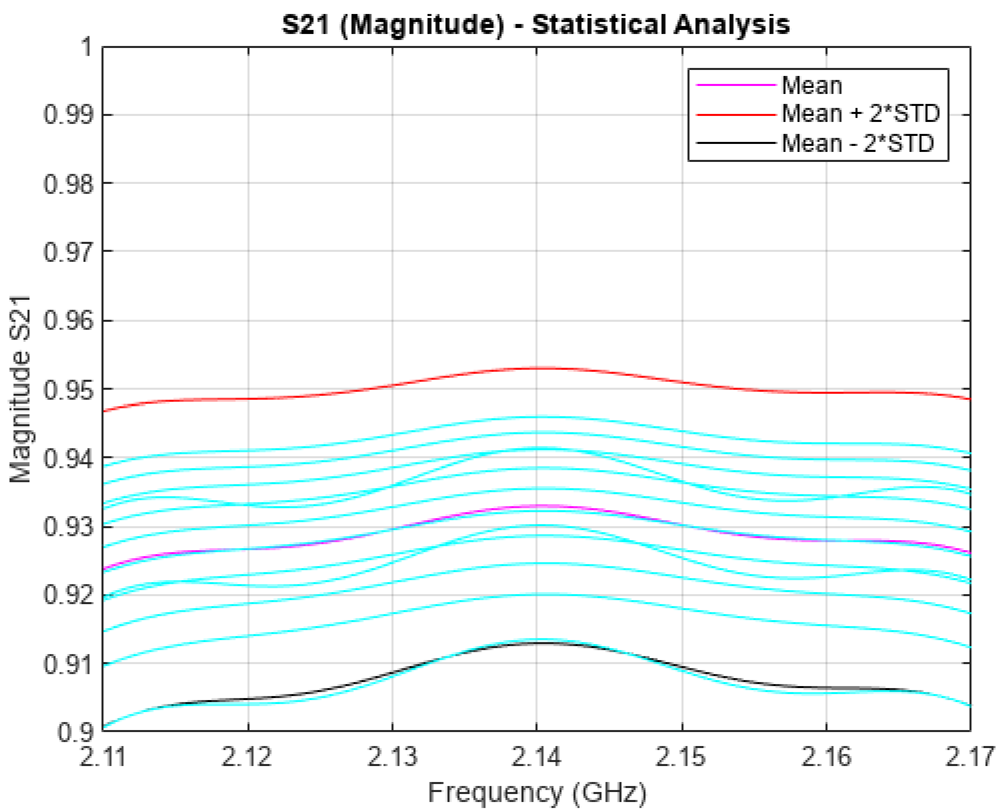
Calculate the mean and STD of the passband magnitude response at each frequency point. This determines if the data follows a normal distribution.

```
mean_abs_S21_freq = mean(abs_S21_pass_freq,2);
std_abs_S21_freq = std(abs_S21_pass_freq,0,2);
```

Plot all the raw passband magnitude data as a function of frequency, as well as the upper and lower limits defined by the basic statistical analysis.

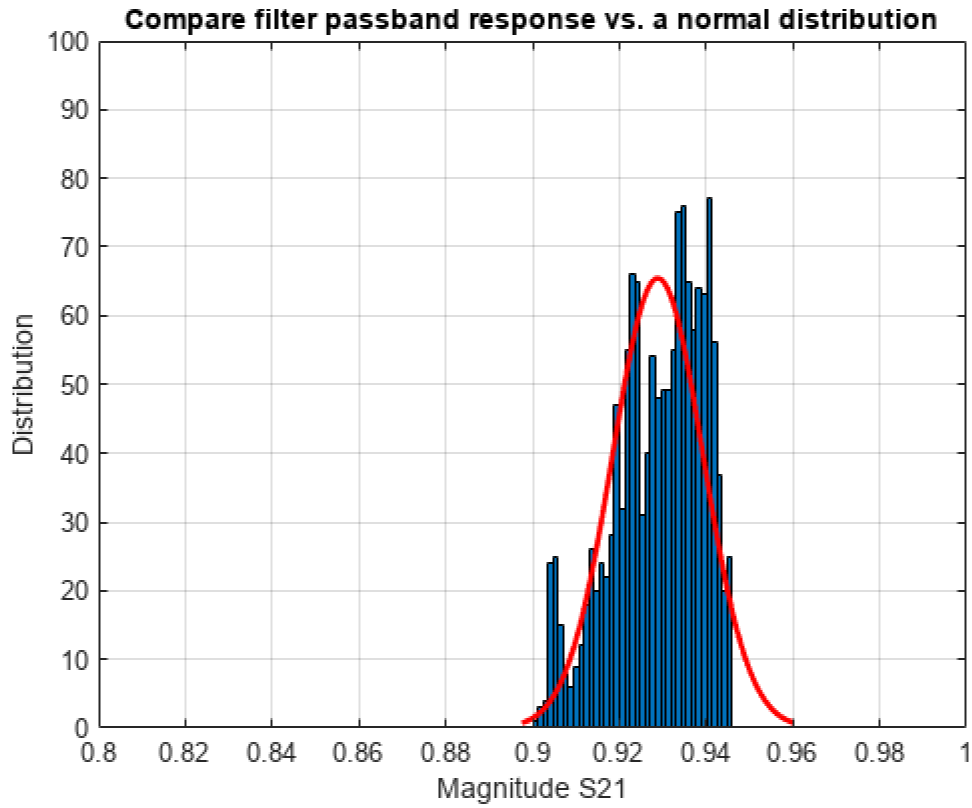
```
plot(freq_pass_ghz,mean_abs_S21_freq,'m')
hold on
```

```
plot(freq_pass_ghz,mean_abs_S21_freq + 2*std_abs_S21_freq,'r')
plot(freq_pass_ghz,mean_abs_S21_freq - 2*std_abs_S21_freq,'k')
legend('Mean','Mean + 2*STD','Mean - 2*STD')
plot(freq_pass_ghz,abs_S21_pass_freq,'c','HandleVisibility','off')
grid on
axis([min(freq_pass_ghz) max(freq_pass_ghz) 0.9 1])
ylabel('Magnitude S21')
xlabel('Frequency (GHz)')
title('S21 (Magnitude) - Statistical Analysis')
hold off
```



Plot a histogram for the passband magnitude data. This determines if the upper and lower limits of the data follow a normal distribution.

```
histfit(abs_S21_pass_freq(:))
grid on
axis([0.8 1 0 100])
xlabel('Magnitude S21')
ylabel('Distribution')
title('Compare filter passband response vs. a normal distribution')
```



Get the groupdelay of the passband S21 data. Use inner 60% of the bandwith for statistical analysis of the groupdelay and normalize it to 10 ns.

```
idx_gpd = (freq >= 2.13e9) & (freq <= 2.15e9);
freq_pass_ghz_gpd = freq(idx_gpd)/1e9; % Normalize to GHz
s21_groupdelay_pass_data = s21_groupdelay(idx_gpd,:)/10e-9; % Normalize to 10 ns
```

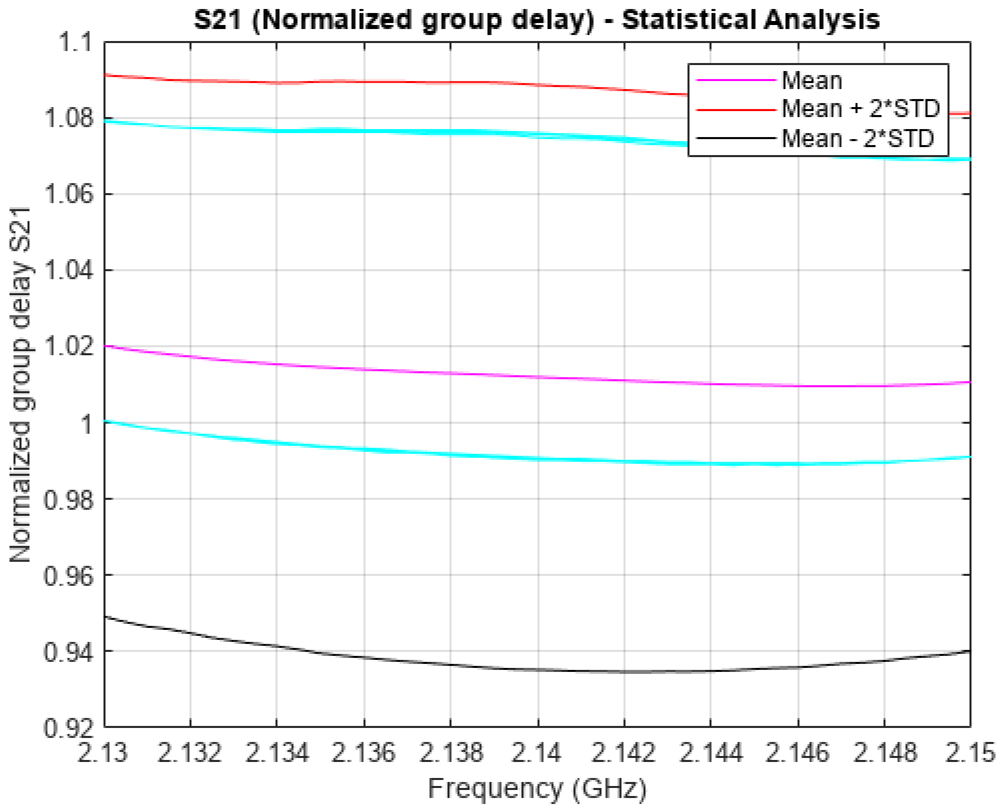
Calculate the per-frequency mean and standard deviation of the normalized group delay response. All the data is collected into a single vector for alter analysis.

```
mean_grpdelay_S21 = mean(s21_groupdelay_pass_data,2);
std_grpdelay_S21 = std(s21_groupdelay_pass_data,0,2);
all_grpdelay_data = reshape(s21_groupdelay_pass_data.',numel(s21_groupdelay_pass_data),1);
```

Plot all the normalized passband groupdelay data as a function of frequency, including the upper and lower limits defined by the basic statistical analysis.

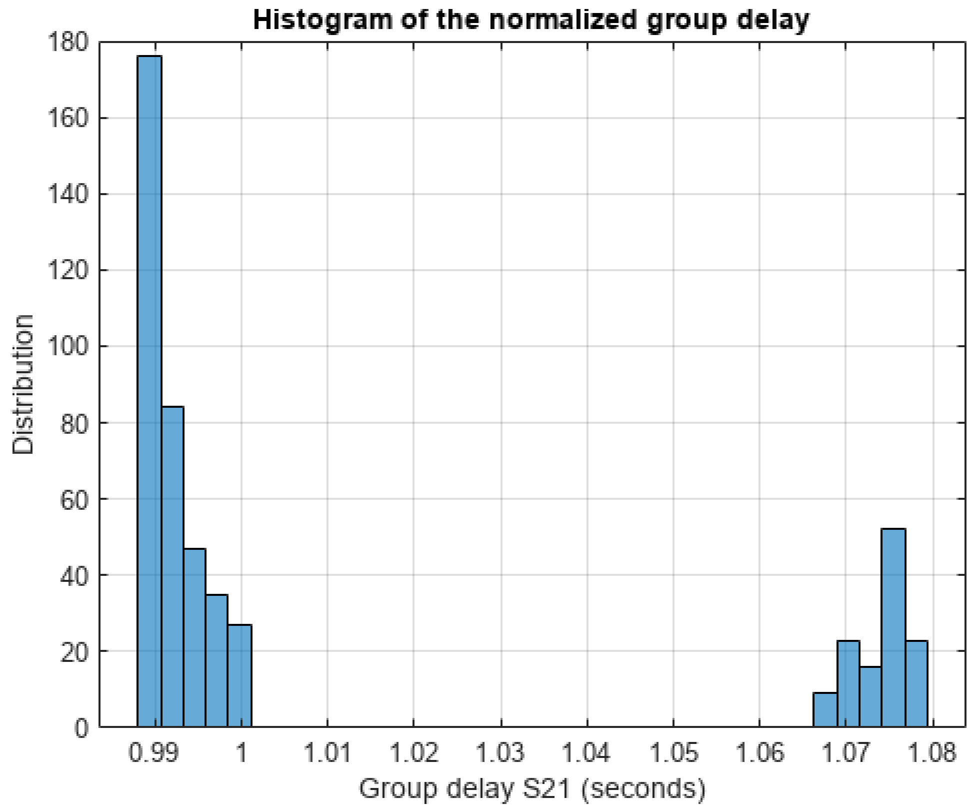
```
plot(freq_pass_ghz_gpd,mean_grpdelay_S21,'m')
hold on
plot(freq_pass_ghz_gpd,mean_grpdelay_S21 + 2*std_grpdelay_S21,'r')
plot(freq_pass_ghz_gpd,mean_grpdelay_S21 - 2*std_grpdelay_S21,'k')
legend('Mean','Mean + 2*STD','Mean - 2*STD')
plot(freq_pass_ghz_gpd,s21_groupdelay_pass_data,'c','HandleVisibility','off')
grid on
xlim([min(freq_pass_ghz_gpd) max(freq_pass_ghz_gpd)])
ylabel('Normalized group delay S21')
xlabel('Frequency (GHz)')
```

```
title('S21 (Normalized group delay) - Statistical Analysis')
hold off
```



Plot a histogram for the normalized passband group delay data. This determines if the upper and lower limits of the data follow a uniform distribution.

```
histogram(all_grpdelay_data,35)
grid on
xlabel('Group delay S21 (seconds)')
ylabel('Distribution')
title('Histogram of the normalized group delay')
```



See Also

More About

- “Bandpass Filter Response Using RFCKT Objects” on page 6-35
- “Bandpass Filter Response” on page 6-23

Write S2P Touchstone Files

This example shows how to write the data in a `circuit` object created in the MATLAB® workspace into an industry-standard data file, Touchstone®. You can use these files in third-party tools.

To write a touchstone file, in this example an RLGC transmission line object is created and analyzed in the frequency domain. This analyzed results are written into a Touchstone file and the data is compared with the original result.

Create RF Circuit Object to Represent an RLCG Transmission Line

Create a `txlineRLCGLine` object to represent a RLCG transmission line. This example uses Name-Value pairs to implement the parameters in the RLCG transmission line shown in figure 1 [1].

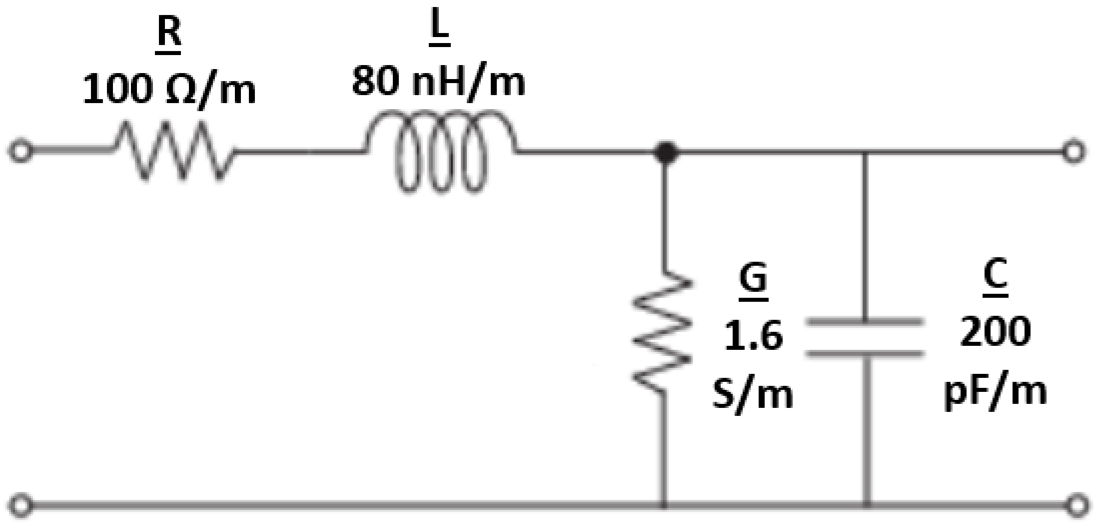


Figure 1: RLCG transmission line.

```
ckt1 = txlineRLCGLine('R',100,'L',80e-9,'C',200e-12,'G',1.6);
```

Clone Circuit Object

Use the `clone` function to make a copy of the transmission line object.

```
ckt2 = clone(ckt1)
```

```
ckt2 = 
  txlineRLCGLine: RLCGLine element

         Name: 'RLCGLine'
    Frequency: 1.0000e+09
            R: 100
            L: 8.0000e-08
            C: 2.0000e-10
            G: 1.6000
     IntpType: 'Linear'
   LineLength: 0.0100
```

```
    Termination: 'NotApplicable'
       StubMode: 'NotAStub'
       NumPorts: 2
      Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Cascade Two Circuit Objects

Use the `circuit` object to cascade the two transmission lines.

```
ckt = circuit([ckt1,ckt2]);
```

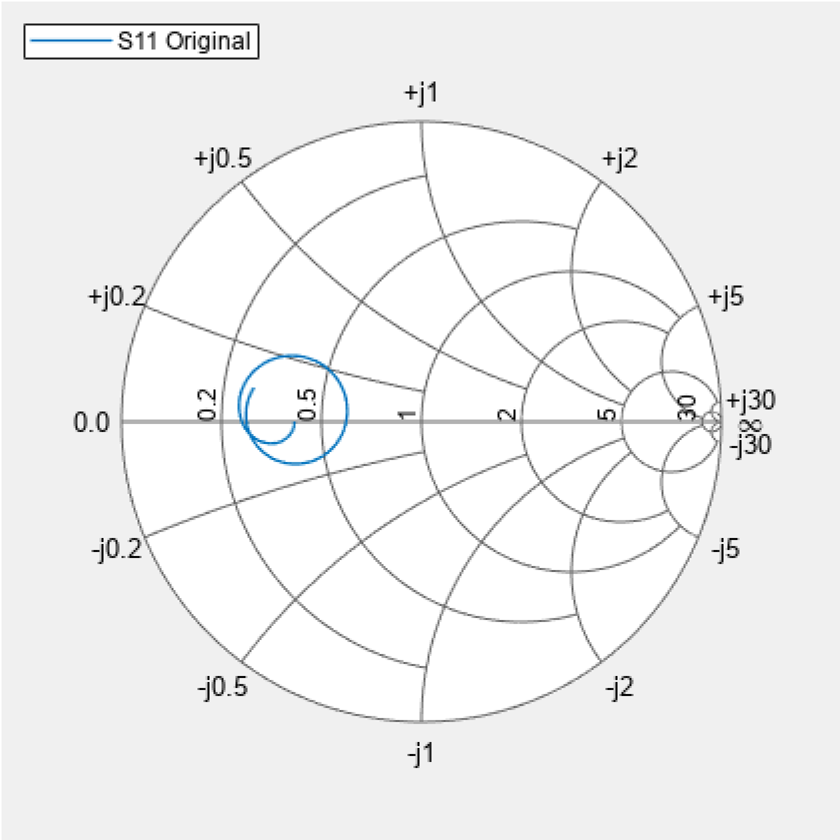
Analyze and Plot S-Parameter Data

Use the `sparameters` object to analyze the cascadeed transmission line in the frequency domain.

```
freq = linspace(0,10e9);
ckt_sparameters = sparameters(ckt,freq);
```

Use the `smithplot` method to plot the object's `S11` on a Smith chart®.

```
figure
smithplot(ckt_sparameters,[1,1],'LegendLabels','S11 Original')
```



Write Data to S2P File

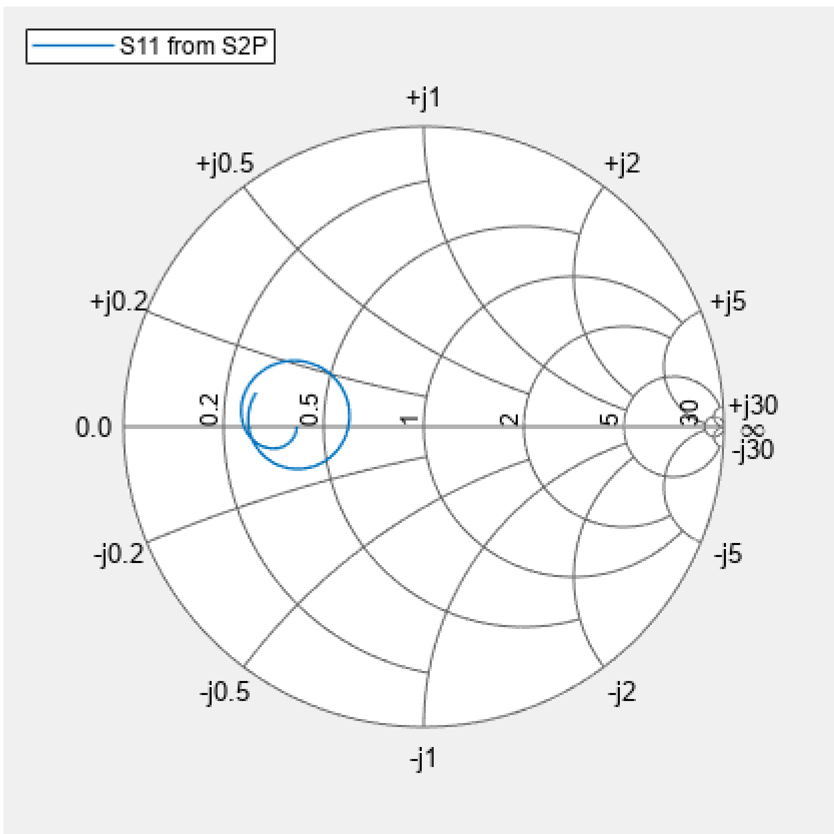
Use the `rfwrite` function to write the data to a file.

```
workingdir = tempname;
mkdir(workingdir);
filename = fullfile(workingdir,'myrlcg.s2p');
if exist(filename,'file')
    delete(filename)
end
rfwrite(ckt_sparameters,filename);
```

Compare Data

Read the data from the file `myrlcg.s2p` into a new `sparameters` object and plot input reflection coefficient, `S11` on a Smith chart. Visually compare the 'S11 original' and 'S11 from S2P' to confirm that the data matches.

```
compare_ckt = sparameters(filename);
figure
smithplot(compare_ckt,[1,1],'LegendLabels','S11 from S2P')
```



[1] M. Steer, "Transmission Lines," in *Microwave and RF Design: Transmission Lines*. vol. 2, 3rd ed. Raleigh, North Carolina, US: North Carolina State University, 2019, ch. 2, sec. 2, pp.58.

See Also

More About

- “Visualize Mixer Spurs” on page 6-98

- “Finding Free IF Bandwidths” on page 6-104

Visualize Mixer Spurs

This example shows how to create an `rfckt.mixer` object and plot the mixer spurs of that object.

Mixers are non-linear devices used in RF systems. They are typically used to convert signals from one frequency to another. In addition to the desired output frequency, mixers also produce intermodulation products (also called mixer spurs), which are unwanted side effects of their nonlinearity. The output of the mixer occurs at the frequencies:

$$F_{out}(N, M) = |NF_{in} + MF_{LO}|$$

where:

- F_{in} is the input frequency.
- F_{LO} is the local oscillator (LO) frequency.
- N is a nonnegative integer.
- M is an integer.

Only one of these output frequencies is the desired tone. For example, in a downconversion mixer (i.e. $F_{in} = F_{RF}$) with a low-side LO (i.e. $F_{RF} > F_{LO}$), the case $N = 1$, $M = -1$ represents the desired output tone. That is:

$$F_{out}(1, -1) = F_{IF} = |NF_{in} + MF_{LO}| = F_{RF} - F_{LO}$$

All other combinations of N and M represent the spurious intermodulation products.

Intermodulation tables (IMTs) are often used in system-level modeling of mixers. This example first examines the IMT of a mixer. Then the example reads an `.s2d` format file containing an IMT, and plots the output power at each output frequency, including the desired signal and the unwanted spurs. The example also creates a cascaded circuit which contains a mixer with IMT followed by a filter, whose purpose is to mitigate the spurs, and plots the output power before and after mitigation.

For more information on IMTs, see the OpenIF example “Finding Free IF Bandwidths” on page 6-104.

Create Mixer Object from Data File

Create an `rfckt.mixer` object to represent the downconverting mixer that is specified in the file, `samplespur1.s2d`. The mixer is characterized by S-parameters, spot noise and IMT. These data are stored in the `NetworkData`, `NoiseData` and `MixerSpurData` properties of the `rfckt` object, respectively.

```
Mixer = rfckt.mixer('FLO', 1.7e9);      % Flo = 1.7GHz
read(Mixer,'samplespur1.s2d');
disp(Mixer)
```

```
  rfckt.mixer with properties:

     MixerSpurData: [1x1 rfdata.mixerspur]
         MixerType: 'Downconverter'
               FLO: 1.7000e+09
        FreqOffset: []
   PhaseNoiseLevel: []
         NoiseData: [1x1 rfdata.noise]
```

```
      NonlinearData: Inf
           IntpType: 'Linear'
        NetworkData: [1x1 rfdata.network]
              nPort: 2
     AnalyzedResult: [1x1 rfdata.data]
               Name: 'Mixer'
```

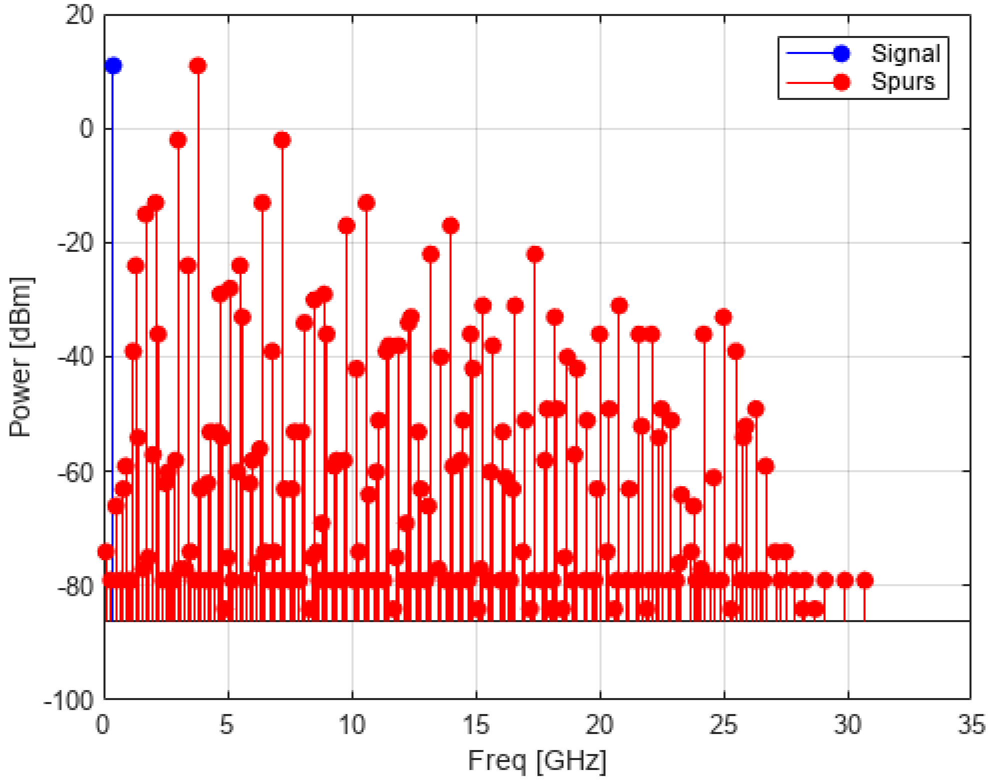
```
IMT = Mixer.MixerSpurData.data
```

```
IMT = 16×16

    99    26    35    39    50    41    53    49    51    42    62    51    60    47    77    50
    24     0    35    13    40    24    45    28    49    33    53    42    60    47    63    99
    73    73    74    70    71    64    69    64    69    62    74    62    72    60    99    99
    67    64    69    50    77    47    74    44    74    47    75    44    70    99    99    99
    86    90    86    88    88    85    86    85    90    85    85    85    99    99    99    99
    90    80    90    71    90    68    90    65    88    65    85    99    99    99    99    99
    90    90    90    90    90    90    90    90    90    90    99    99    99    99    99    99
    90    90    90    90    90    87    90    90    90    99    99    99    99    99    99    99
    99    95    99    95    99    95    99    95    99    99    99    99    99    99    99    99
    90    95    90    90    90    99    90    99    99    99    99    99    99    99    99    99
      ⋮
```

Plot Mixer Output Signal and Spurs

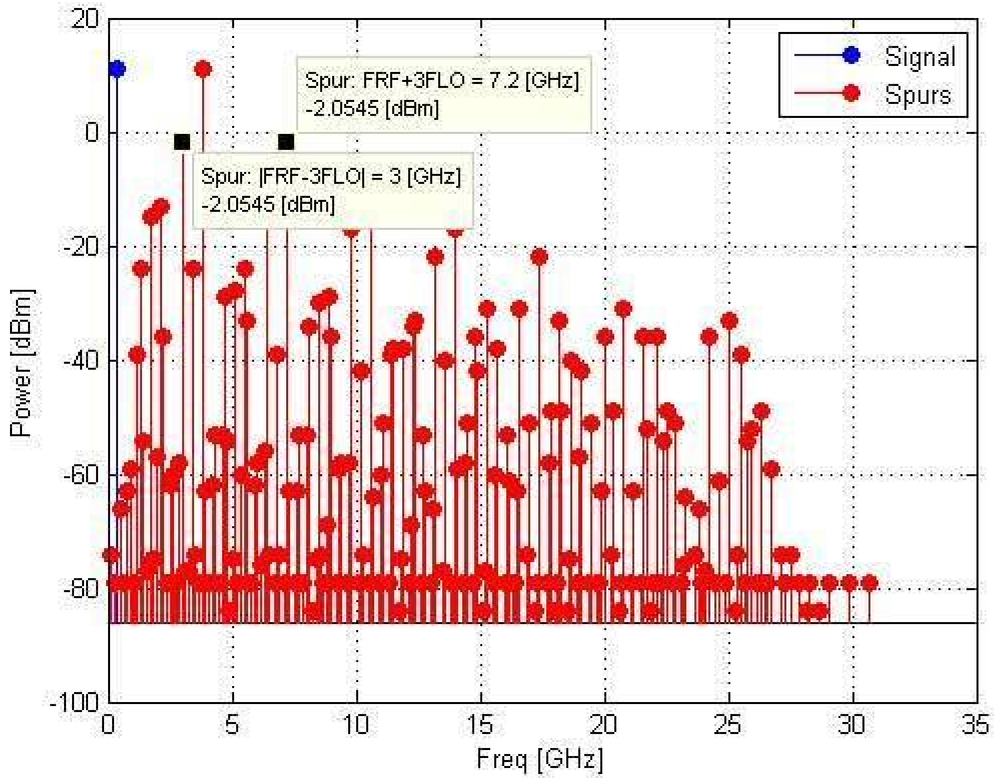
Use the `plot` method of the `rfckt` object to plot the power of the desired output signal and the spurs. The second input argument must be the string `'MIXERSPUR'`. The third input argument must be the index of the circuit for which to plot output power data. The `rfckt.mixer` object only contains one circuit (the mixer), so index 0 corresponds to the mixer input and index 1 corresponds to the mixer output.

```
CktIndex = 1;       % Plot the output only
Pin = -10;          % Input power is -10dBm
Fin = 2.1e9;        % Input frequency is 2.1GHz
figure
plot(Mixer,'MIXERSPUR',CktIndex,Pin,Fin);
```



Use Data Cursor

Run the cursor over the plot to get the frequency and power level of each signal and spur.



Create Cascaded Circuit

Create an amplifier object for LNA, mixer, and LC Bandpass Tee objects. Then build the cascade shown in the following figure:

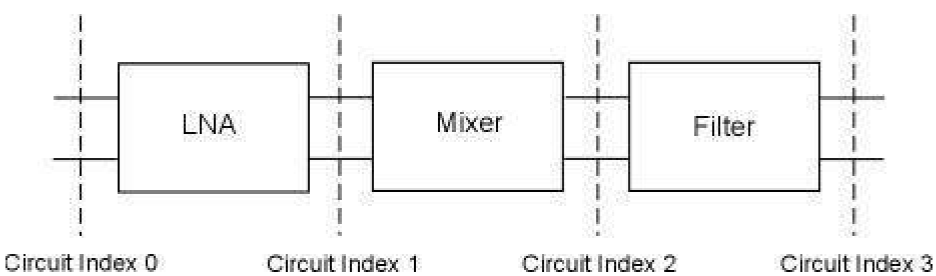


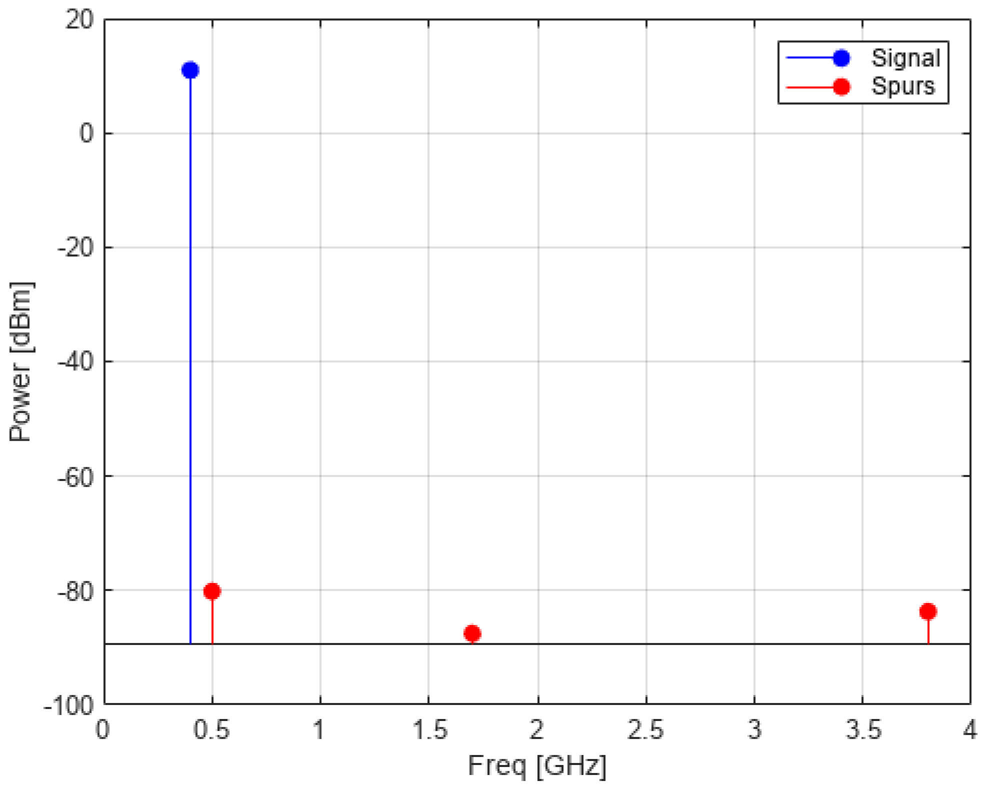
Figure 1: Cascaded Circuit

```
FirstCkt = rfckt.amplifier('NetworkData', ...
    rfdata.network('Type','S','Freq',2.1e9,'Data',[0,0;10,0]), ...
    'NoiseData',0,'NonlinearData',Inf);          % 20dB LNA
SecondCkt = copy(Mixer);                        % Mixer with IMT table
ThirdCkt = rfckt.lcbandpasstee('L',[97.21 3.66 97.21]*1.0e-9, ...
    'C',[1.63 43.25 1.63]*1.0e-12);             % LC Bandpass filter
CascadedCkt = rfckt.cascade('Ckts',{FirstCkt,SecondCkt,ThirdCkt});
```

Plot Output Signal and Spurs of LC filter in Cascade

Use the `plot` method of the `rfckt` object to plot the power of the desired output signal and the spurs. The third input argument is 3, which directs the toolbox to plot the power at the output of the third component of the cascade (the LC filter).

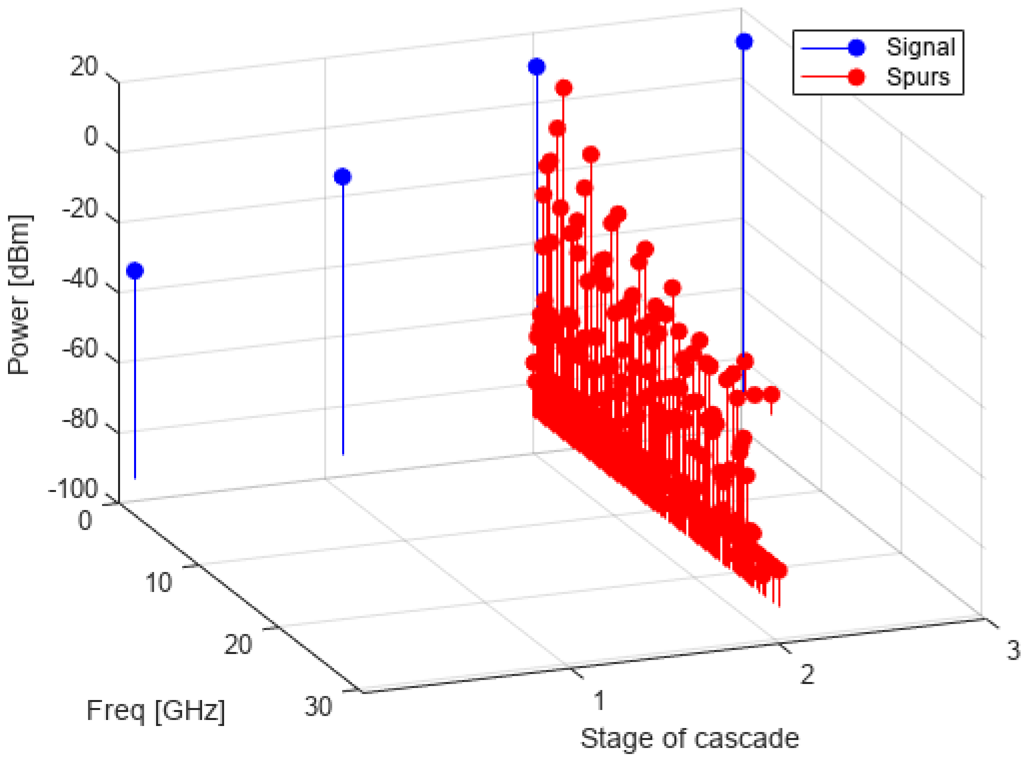
```
CktIndex = 3;       % Plot the output signal and spurs of the LC filter,
                    % which is the 3rd circuit in the cascade
Pin = -30;          % Input power is -30dBm
Fin = 2.1e9;        % Input frequency is 2.1GHz
plot(CascadedCkt,'MIXERSPUR',CktIndex,Pin,Fin)
```



Plot Cascade Signal and Spurs in 3D

Use the `plot` method of the `rfckt` object with a third input argument of `'all'` to plot the input power and the output power after each circuit component in the cascade. Circuit index `0` corresponds to the input of the cascade. Circuit index `1` corresponds to the output of the LNA. Circuit index `2` corresponds to the output of the mixer, which was shown in the previous plot. Circuit index `3` corresponds to the output of the LC Bandpass Tee filter.

```
CktIndex = 'all';   % Plot the input signal, the output signal, and the
                    % spurs of the three circuits in the cascade: FirstCkt,
                    % SecondCkt and ThirdCkt
Pin = -30;          % Input power is -30dBm
Fin = 2.1e9;        % Input frequency is 2.1GHz
plot(CascadedCkt,'MIXERSPUR',CktIndex,Pin,Fin)
view([68.5 26])
```



See Also

More About

- “Finding Free IF Bandwidths” on page 6-104
- “Write S2P Touchstone Files” on page 6-94

Finding Free IF Bandwidths

This example shows how to select an Intermediate Frequency (IF) that is free from any intermodulation distortion. First, you create an `OpenIF` object and specify whether you are designing a transmitter or receiver. Second, you use the `addMixer` function to define the properties of each mixer as well as the specific Radio Frequency (RF) it interacts with. Lastly, you view the results using the functions `report` and `show`.

Background Knowledge (Mixer Spurs)

When converting from RF to IF (receiver) or from IF to RF (transmitter), a mixer is used. Unfortunately, mixers are nonlinear and their outputs contain energy at unwanted frequencies (we call these unwanted outputs "spurs"). The `OpenIF` tool helps you to select an IF which avoids having these spurious mixer outputs interfere with the mixer output. The output of the mixer is characterized by the following equation:

$$F_{out}(N, M) = |NF_{in} + MF_{LO}|$$

where:

- F_{in} is the input frequency.
- F_{LO} is the local oscillator (LO) frequency.
- N is a nonnegative integer.
- M is an integer.

Only one of these output frequencies is the desired tone. For example, in a downconversion mixer (i.e. $F_{in} = F_{RF}$) with a low-side LO (i.e. $F_{RF} > F_{LO}$), the case $N = 1$, $M = -1$ represents the desired output tone. That is:

$$F_{out}(1, -1) = F_{IF} = |NF_{in} + MF_{LO}| = F_{RF} - F_{LO}$$

All other combinations of N and M represent the spurious intermodulation products. To characterize these intermodulation products, an Intermodulation Table (IMT) is used.

Background Knowledge (Intermodulation Tables)

The IMT provides information on the amount of power generated at each intermodulation product frequency. For accurate mixer spurs analysis results, the IMT should be built from simulated or measured data at the desired input signal and local oscillator frequency and power conditions. Extrapolation to other conditions will lead to inaccuracies.

Here is the IMT of a downconverting mixer with a low side LO, measured at $F_{in} = F_{RF} = 2.1$ GHz, $P_{in} = P_{RF} = -10$ dBm, $F_{LO} = 1.7$ GHz, and $P_{LO} = 7$ dBm.

```
     ! Element (N,M) gives power of |N*Fin+M*Flo| in dBc
     ! Top indices give M =
     ! Left-hand indices give N =
      %0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
 0%   99 26 35 39 50 41 53 49 51 42 62 51 60 47 77 50
 1%   24 0  35 13 40 24 45 28 49 33 53 42 60 47 63
 2%   73 73 74 70 71 64 69 64 69 62 74 62 72 60
 3%   67 64 69 50 77 47 74 44 74 47 75 44 70
```

```
 4%   86 90 86 88 88 85 86 85 90 85 85 85
 5%   90 80 90 71 90 68 90 65 88 65 85
 6%   90 90 90 90 90 90 90 90 90 90
 7%   90 90 90 90 90 87 90 90 90
 8%   99 95 99 95 99 95 99 95
 9%   90 95 90 90 90 99 90
10%   99 99 99 99 99 99
11%   90 99 90 95 90
12%   99 99 99 99
13%   90 99 90
14%   99 99
15%   99
```

Notice that it is a convention in industry-standard IMTs to assume symmetry, namely:

$$P_{out}(N, M) = P_{out}(N, -M)$$

and RF Toolbox™ software follows this convention.

If the measurement reveals that in fact the mixer is asymmetric, i.e.:

$$P_{out}(N, M) \neq P_{out}(N, -M)$$

there is no way of accommodating this information in an industry-standard IMT. In this situation, the most common convention is to build an approximate model by placing the value:

$$\max(P_{out}(N, M), P_{out}(N, -M))$$

at position N, M.

Thus industry-standard IMTs in general and RF Toolbox in particular will over-estimate the power of one spur in each pair of asymmetric spurs.

In the IMT, a `0` always appears in the table at the position $N = 1$, $M = 1$, which represents both the desired signal and its symmetric image pair. All other entries are specified in dBc below the power of the mixer output at the desired frequency. (In the unlikely case of a spur being above the power of the desired, it will appear as a negative number, the magnitude of which is the spur power in dBc above the desired.)

For example, in the IMT above, at row $N = 1$, column $M = 3$, the IMT value is `13`. RF Toolbox will place a pair of symmetric IM products at:

$$F_{out}(1, 3) = F_{in} + 3F_{LO}$$

$$F_{out}(1, -3) = |F_{in} - 3F_{LO}|$$

each with a power level of -13 dBc. The absolute power of a spur in dBm is calculated by subtracting the IMT dBc value from the output power (also in dBm) of the desired tone.

By convention, the special value of `99` means the tone at that index is negligible.

For more information on intermodulation tables, see [1] on page 6-112.

Design Requirements

Find a spur-free IF for a receiver. The receiver must be able to downconvert from three separate RF bands to the same (shared) IF. To find an IF center frequency that is spur-free for all three RF bands, your requirements must specify the RF Center Frequency, the RF Bandwidth, and the IF Bandwidth that goes with that particular RF:

```
% RF band 1
RFCF1 = 2400e6; % 2.4 GHz
RFBW1 = 200e6;  % 200 MHz
IFBW1 = 20e6;   %  20 MHz

% RF band 2
RFCF2 = 3700e6; % 3.7 GHz
RFBW2 = 250e6;  % 250 MHz
IFBW2 = 20e6;   %  20 MHz

% RF band 3
RFCF3 = 5400e6; % 5.4 GHz
RFBW3 = 250e6;  % 250 MHz
IFBW3 = 50e6;   %  50 MHz
```

Next we must have an IMT measured for each RF band. Assume you have tested and measured the mixers you plan to use with the following results:

```
IMT1 = [99  0 21 17 26;
        11  0 29 29 63;
        60 48 70 86 41;
        90 89 74 68 87;
        99 99 95 99 99];

IMT2 = [99  1  9 12 15;
        20  0 26 31 48;
        55 70 51 70 53;
        85 90 60 70 94;
        96 95 94 93 92];

IMT3 = [99  2 11 15 16;
        27  0 16 41 55;
        25 61 66 65 47;
        92 83 66 77 88;
        97 94 91 92 99];
```

Find Spur-Free frequencies using the OpenIF object

Create the object using the `OpenIF` function. Specify you are designing a receiver by setting the 'IFLocation' property to 'MixerOutput'.

```
h = OpenIF('IFLocation', 'MixerOutput');
```

Use the `addMixer` method to input the information for each RF band. Here low-side injection is assumed for each mixer, but high-side injection could be tried later.

```
addMixer(h,IMT1, RFCF1, RFBW1, 'low', IFBW1);
addMixer(h,IMT2, RFCF2, RFBW2, 'low', IFBW2);
addMixer(h,IMT3, RFCF3, RFBW3, 'low', IFBW3);
```

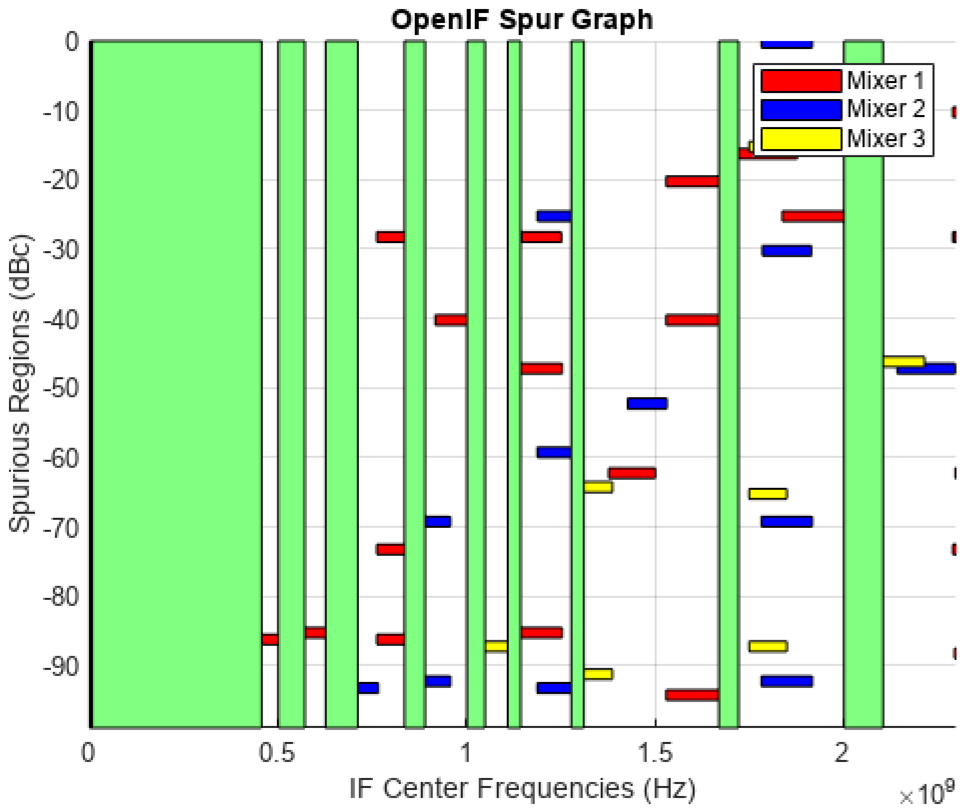
View the results textually using the `report` method.

```
report(h);
```

```
Intermediate Frequency (IF) Planner
IF Location: MixerOutput

-- MIXER 1 --
RF Center Frequency: 2.4 GHz
RF Bandwidth: 200 MHz
IF Bandwidth: 20 MHz
MixerType: low
Intermodulation Table:   99   0  21  17  26
                         11   0  29  29  63
                         60  48  70  86  41
                         90  89  74  68  87
                         99  99  95  99  99

-- MIXER 2 --
RF Center Frequency: 3.7 GHz
RF Bandwidth: 250 MHz
IF Bandwidth: 20 MHz
MixerType: low
Intermodulation Table:   99   1   9  12  15
                         20   0  26  31  48
                         55  70  51  70  53
                         85  90  60  70  94
                         96  95  94  93  92

-- MIXER 3 --
RF Center Frequency: 5.4 GHz
RF Bandwidth: 250 MHz
IF Bandwidth: 50 MHz
MixerType: low
Intermodulation Table:   99   2  11  15  16
                         27   0  16  41  55
                         25  61  66  65  47
                         92  83  66  77  88
                         97  94  91  92  99

Spur-Free Zones:
   2.00 -    2.50 MHz
   2.50 -    3.33 MHz
   3.33 -    6.25 MHz
   6.25 -    8.33 MHz
   8.33 -  458.00 MHz
 502.00 -  572.50 MHz
 627.50 -  713.00 MHz
 836.67 -  891.25 MHz
   1.00 -    1.05 GHz
   1.11 -    1.15 GHz
   1.28 -    1.31 GHz
   1.67 -    1.72 GHz
   2.00 -    2.10 GHz
```

View the results graphically using the `show` method.

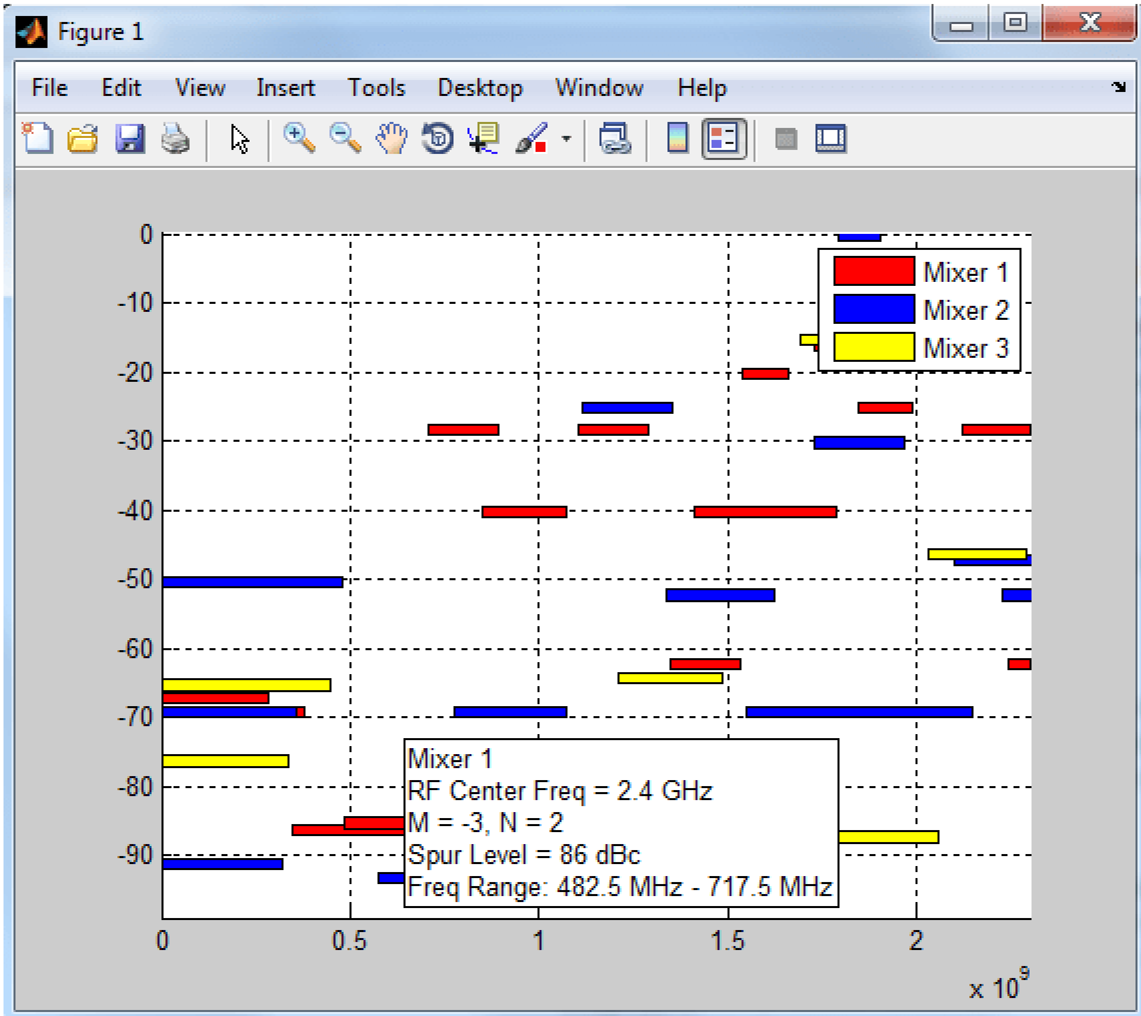
```
figure;
show(h);
```



Interpreting the Graphical Results

The figure created by the `show` method displays all relevant spurious frequency ranges as colored horizontal rectangles. If there any spur-free zones (there may not be) it will be displayed as vertical green rectangle.

In this example, as we can see in the figure, there are no spur-free zones. The legend in the upper right-hand corner tells us which color each Mixer is associated with. If we wish more detailed information about a spurious region, we can click on one of the rectangles:

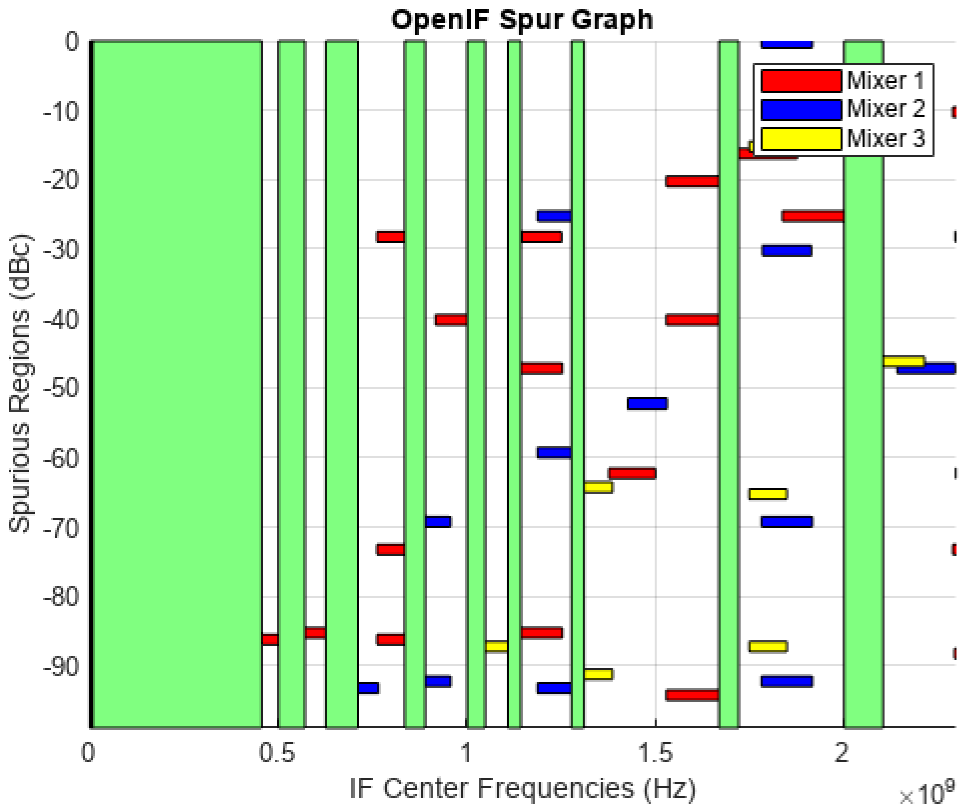


If we wish to find a spur-free zone, we will have to adjust some of the parameters of the setup.

Adjusting a Mixer Property to find Spur-Free Zones

In the current setup, there are no spur-free zones available. We will need to adjust some of the setup parameters in order to find a spur-free zone. The values laid out in the design requirements (RF Bandwidth, RF Center Frequency, and IF Bandwidth) cannot be changed. However, some parameters (such as altering low- or high-side injection) are design decisions. We can see if changing the first mixer to high-side injection will open up a spur-free zone:

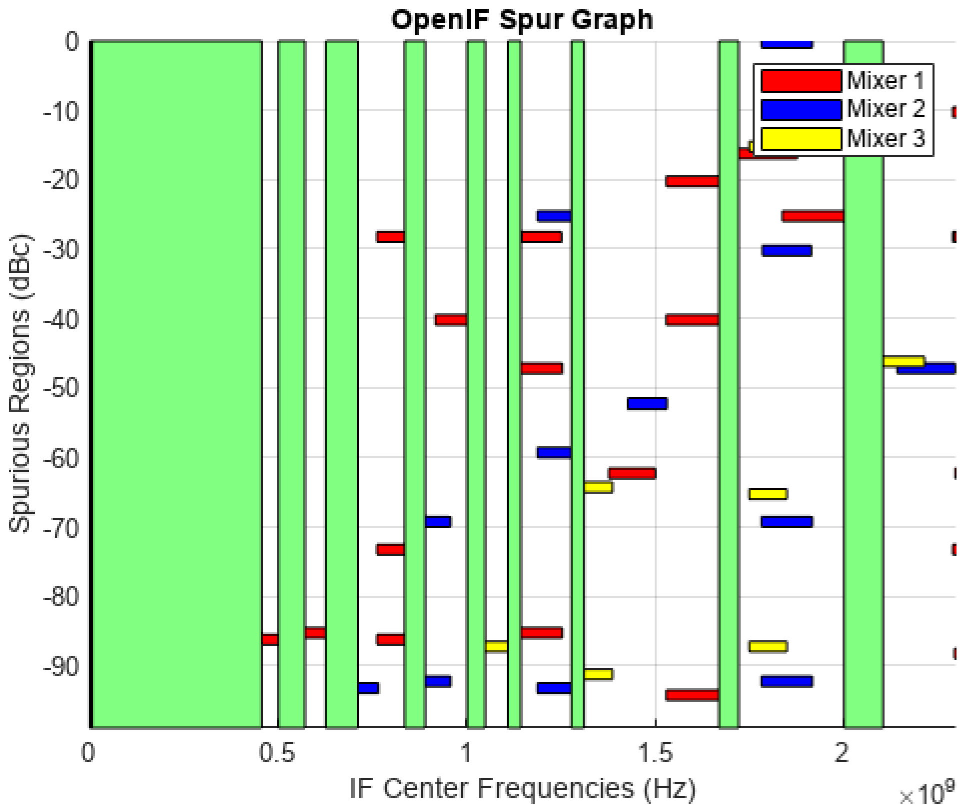
```
h.Mixers(1).MixingType = 'high';
figure;
show(h);
```



Adjusting the SpurFloor to find Spur-Free Zones

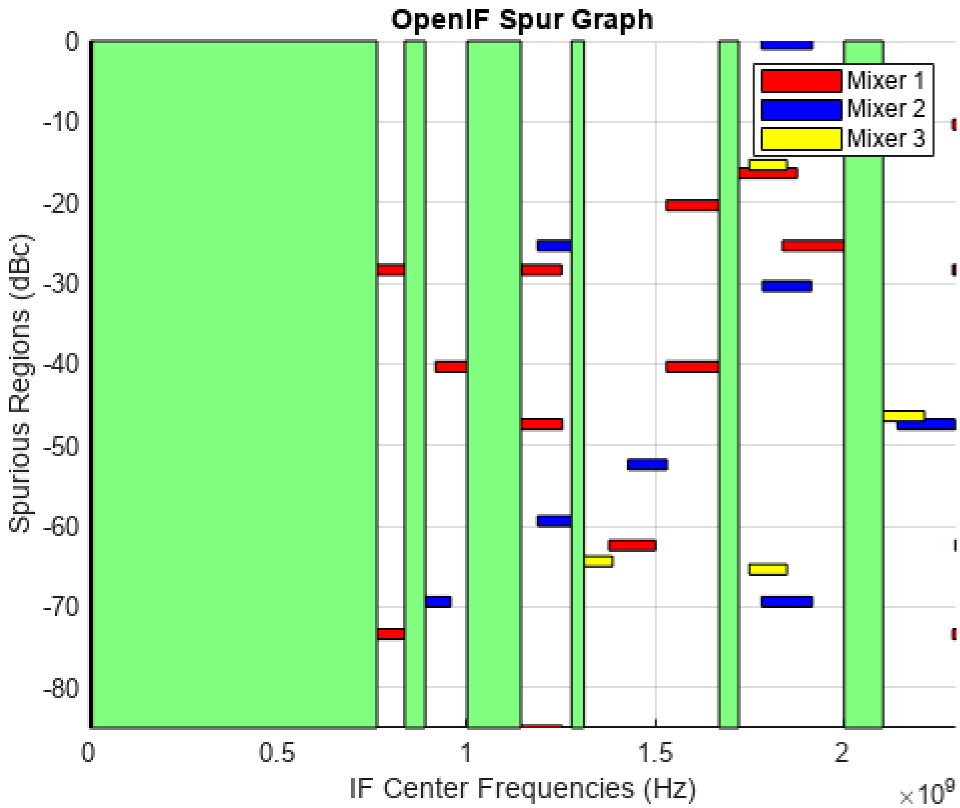
If we wish to use low-side injection in all of the mixers, we must find acceptable spur-free zones by adjusting other parameters. Here we reset the OpenIF object to all low-side injection, and re-plot the results:

```
h.Mixers(1).MixingType = 'low';
figure;
show(h);
```



We notice there is a section around 500 MHz where there is a opening all the way down to roughly -85 dBc. We can find that zone by adjusting the `SpurFloor` property:

```
h.SpurFloor = 85;
show(h);
```



References

[1] Daniel Faria, Lawrence Dunleavy, and Terje Svensen. "The Use of Intermodulation Tables for Mixer Simulations," Microwave Journal, Vol. 45, No. 4, December 2002, p. 60.

See Also

More About

- “Visualize Mixer Spurs” on page 6-98
- “Write S2P Touchstone Files” on page 6-94

De-Embedding S-Parameters

This example shows you how to extract the S-parameters of a Device Under Test (DUT). First, read a Touchstone® file into a `sparameters` object, second, calculate the S-parameters for the left and right pads, third, de-embed the S-parameters using the `deembedsparams` function and finally display the results.

This example uses the S-parameter data in the file `samplebjt2.s2p` that was collected from a bipolar transistor in a fixture with a bond wire (series inductance of 1 nH) connected to a bond pad (shunt capacitance of 100 fF) on the input, and a bond pad (shunt capacitance of 100 fF) connected to a bond wire (series inductance of 1 nH) on the output, see Figure 1.

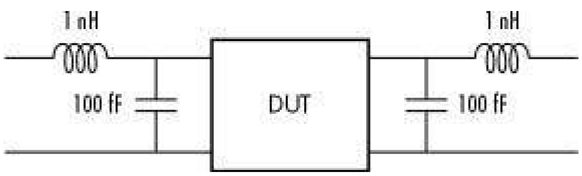


Figure 1: Device under test (DUT) and the test fixture.

This example will also show how to remove the effects of the fixture in order to extract the S-parameters of the DUT.

Read Measured S-Parameters

Create a `sparameters` object for the measured S-parameters, by reading the Touchstone® data file, `samplebjt2.s2p`.

```
S_measuredBJT = sparameters('samplebjt2.s2p');
freq = S_measuredBJT.Frequencies;
```

Calculate S-Parameters for Left Pad

Create a two port `circuit` object representing the left pad, containing a series `inductor` and a shunt `capacitor`. Then calculate the S-parameters using the frequencies from `samplebjt2.s2p`.

```
leftpad = circuit('left');
add(leftpad,[1 2],inductor(1e-9));
add(leftpad,[2 3],capacitor(100e-15));
setports(leftpad,[1 3],[2 3]);
S_leftpad = sparameters(leftpad,freq);
```

Calculate S-Parameters for Right Pad

Create a two port `circuit` object representing the right pad, containing a series `inductor` and shunt `capacitor`. Then, calculate the S-parameters using the frequencies from `samplebjt2.s2p`.

```
rightpad = circuit('right');
add(rightpad,[1 3],capacitor(100e-15));
add(rightpad,[1 2],inductor(1e-9));
setports(rightpad,[1 3],[2 3]);
S_rightpad = sparameters(rightpad,freq);
```

De-Embed S-Parameters

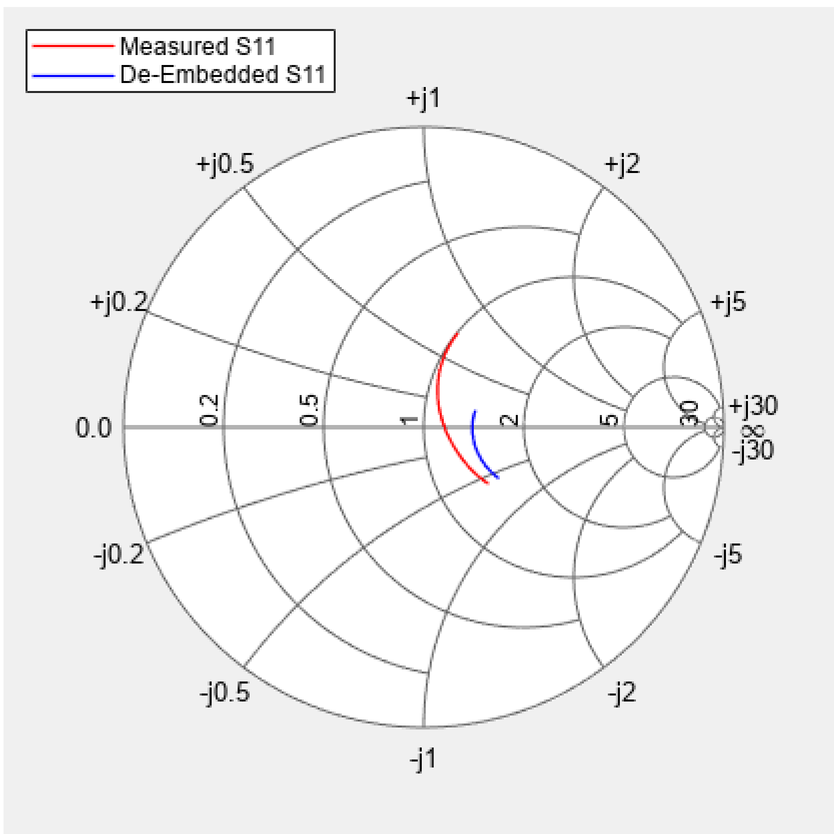
De-embed the S-parameters of the DUT from the measured S-parameters by removing the effects of input and output pads (`deembedsparams`).

```
S_DUT = deembedsparams(S_measuredBJT,S_leftpad,S_rightpad);
```

Plot Measured and De-Embedded S11 Parameters on Z Smith® Chart

Use the `smithplot` function to plot the measured and de-embedded S11 parameters.

```
figure
hs = smithplot(S_measuredBJT,1,1);
hold on;
smithplot(S_DUT,1,1)
hs.ColorOrder = [1 0 0; 0 0 1];
hs.LegendLabels = {'Measured S11','De-Embedded S11'};
```

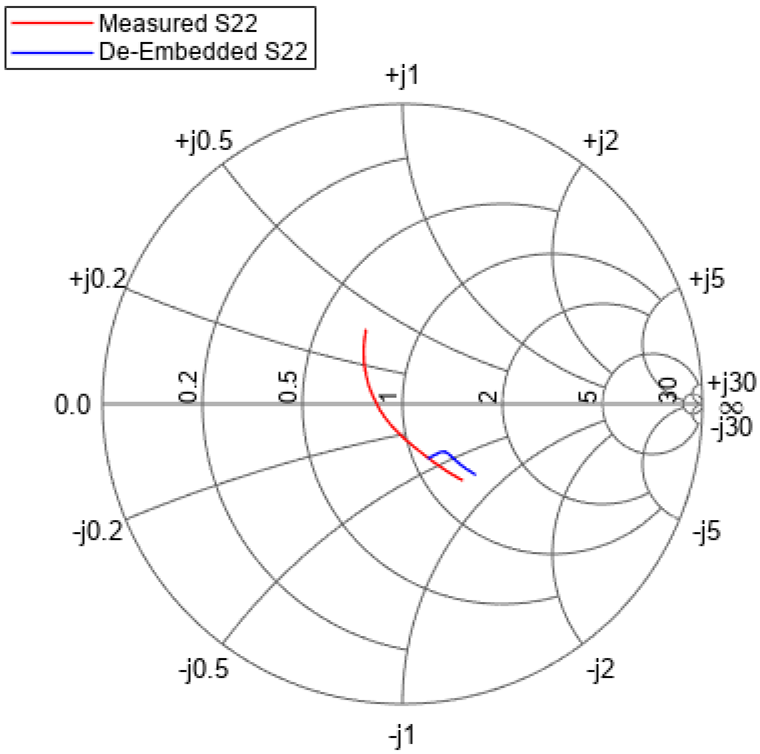


Plot Measured and De-Embedded S22 Parameters on Z Smith Chart

Use the `smithplot` function to plot the measured and de-embedded S22 parameters.

```
figure
hold off;
smithplot(S_measuredBJT,2,2)
hold on;
smithplot(S_DUT,2,2)
hs = smithplot('gco');
```

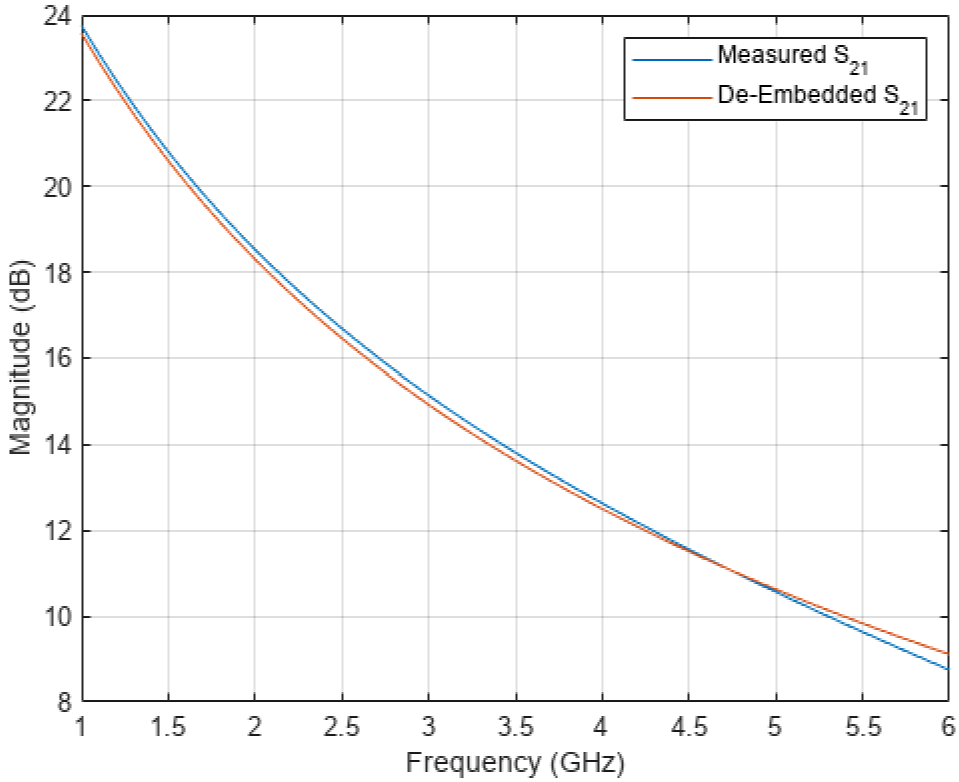
```
hs.ColorOrder = [1 0 0; 0 0 1];
hs.LegendLabels = {'Measured S22','De-Embedded S22'};
```



Plot Measured and De-Embedded S21 Parameters in Decibels

Use the `rfplot` function to plot the measured and de-embedded S21 parameters.

```
figure
hold off;
h1 = rfplot(S_measuredBJT,2,1);
hold on;
h2 = rfplot(S_DUT,2,1);
legend([h1,h2],{'Measured S_{21}','De-Embedded S_{21}'});
```



See Also

More About

- “Extract S-Parameters from Circuit” on page 6-237
- “Extract S-Parameters from Mutual Inductor” on page 6-241
- “Bisect S-Parameters of Cascaded Probes” on page 6-117

Bisect S-Parameters of Cascaded Probes

This example shows how to separate the S-parameters of two identical, passive, symmetric probes connected in a cascade.

Introduction

Consider a DUT (device under test) connected to two probes. In order to de-embed the S-parameters of DUT, you need to know the S-parameters of each individual probe. For accurate S-parameters of the two probes, the calibration is done in the lab using SOLT (short, open, load, and thru) or TRL (thru, reflect, line) measurements. However, if you assume the probes are identical and symmetric, then you can approximate S-parameters quickly using the procedure sketched here.

The file `connectedprobes.s2p` contains the S-parameter data when the probes are connected directly to each other.

ABCD-parameters

This example uses ABCD-parameters to bisect measured S-parameter data into the data for each individual probe.

When you cascade two networks, you can calculate the ABCD-parameters of the combined network by matrix multiplying the ABCD-parameters of the two individual networks.

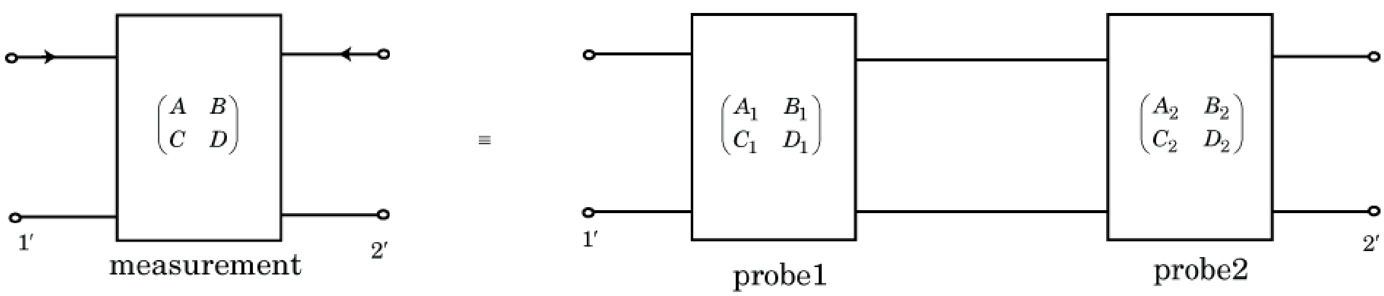


Figure 1: Main network and network with two symmetric probes connected in a cascade

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} A_1 & B_1 \\ C_1 & D_1 \end{pmatrix}\begin{pmatrix} A_2 & B_2 \\ C_2 & D_2 \end{pmatrix}$$

If, $\begin{pmatrix} A_1 & B_1 \\ C_1 & D_1 \end{pmatrix} = \begin{pmatrix} A_2 & B_2 \\ C_2 & D_2 \end{pmatrix}$, then, $\begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} A_1 & B_1 \\ C_1 & D_1 \end{pmatrix}^2$

From the above equation, you can find the ABCD-parameters of the two individual probes by taking the matrix square root of the ABCD-parameters of main network.

Since both probes are identical, you can calculate the S-parameters of either one of the probes.

Extract Required S-Parameter Data from Given Touchstone file

Create an `sparameters` object from the Touchstone® data file `connectedprobes.s2p`.

```
filename = 'connectedprobes.s2p';
S = sparameters(filename);
```

```
numports = S.NumPorts;
freq = S.Frequencies;
numfreq = numel(freq);
z0 = S.Impedance;
```

Calculate S-Parameter Data of Individual Probe

Create a zero matrix to store the ABCD-parameter data of the probe.

```
abcd_probe_data = zeros(numports,numports,numfreq);
```

To calculate S-Parameters of the probe, you need to know the S-parameters at every frequency it operates. Convert the S-parameters extracted from **connectedprobes.s2p** to ABCD-parameters. Then calculate the matrix square root of ABCD-parameters using `sqrtm` function to get the ABCD-parameters of the probe. Convert these ABCD-parameters of the probe to S-parameters.

```
ABCD = abcdparameters(S);
for n = 1:numfreq
    abcd_meas = ABCD.Parameters(:,:,n);
    abcd_probe_data(:,:,n) = sqrtm(abcd_meas);
end
ABCD_probe = abcdparameters(abcd_probe_data,freq);
```

Create an S-parameter object from the calculated S-parameter data of the probe.

```
S_probe = sparameters(ABCD_probe,z0);
```

Compare Calculated S-Parameters with Expected S-Parameters

For this example, **connectedprobes.s2p** gives the S-Parameter data of this network.

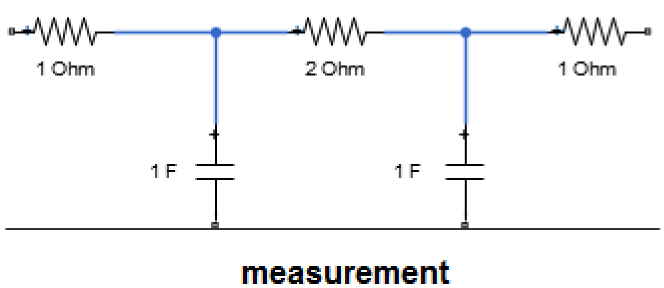


Figure 2: Network derived from connectedprobes.s2p

Split the above network into two identical networks, **probe1** and **probe2**. The S-parameters of these probes represent the expected result.

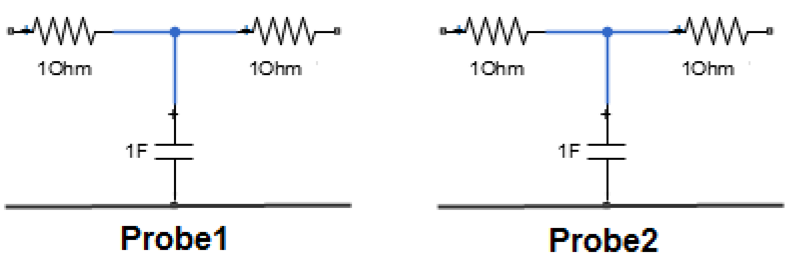


Figure 3: Identical Networks

Create **probe1** using `circuit`, `resistor`, and `capacitor` objects from the RF Toolbox.

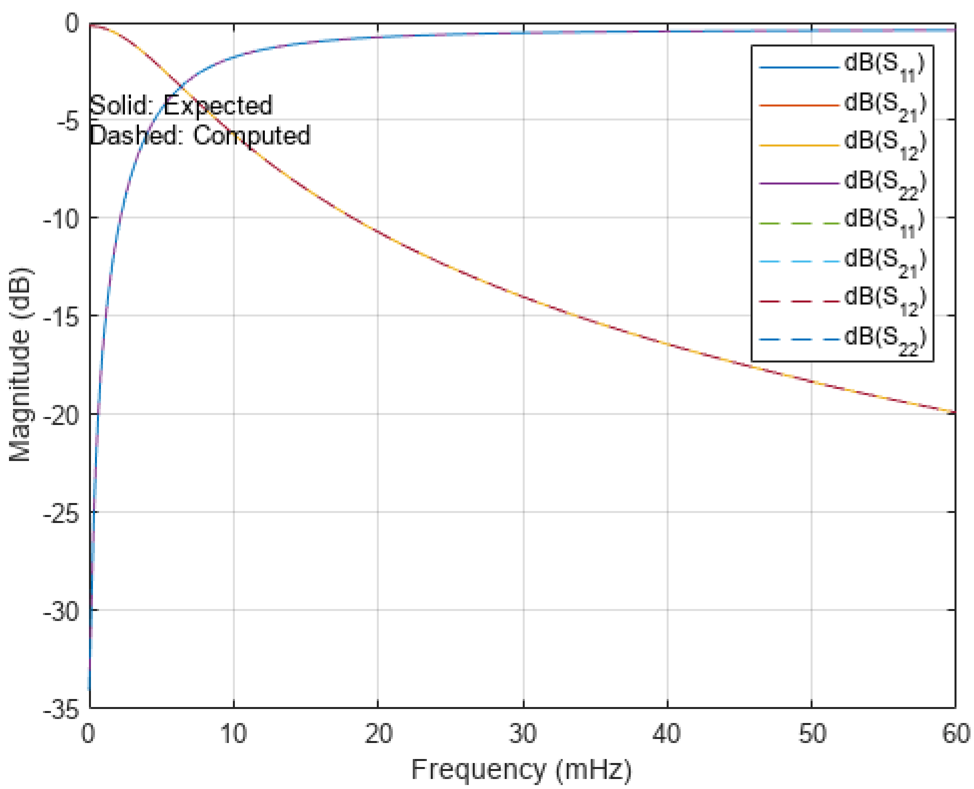
```
R1 = 1;
C1 = 1;
R2 = 1;
ckt = circuit('probe1');
add(ckt,[1 2],resistor(R1))
add(ckt,[2 4],capacitor(C1))
add(ckt,[2 3],resistor(R2))
```

Calculate the expected S-parameters of probe 1.

```
setports(ckt,[1 4],[3 4])
S_exp = sparameters(ckt,freq,z0);
```

Plot and compare the expected S-parameters from **probe1** and those calculated using ABCD-parameters and compare.

```
rfplot(S_exp)
hold on
rfplot(S_probe,'--')
hold off
text(0.02,-5,{'Solid: Expected','Dashed: Computed'})
```



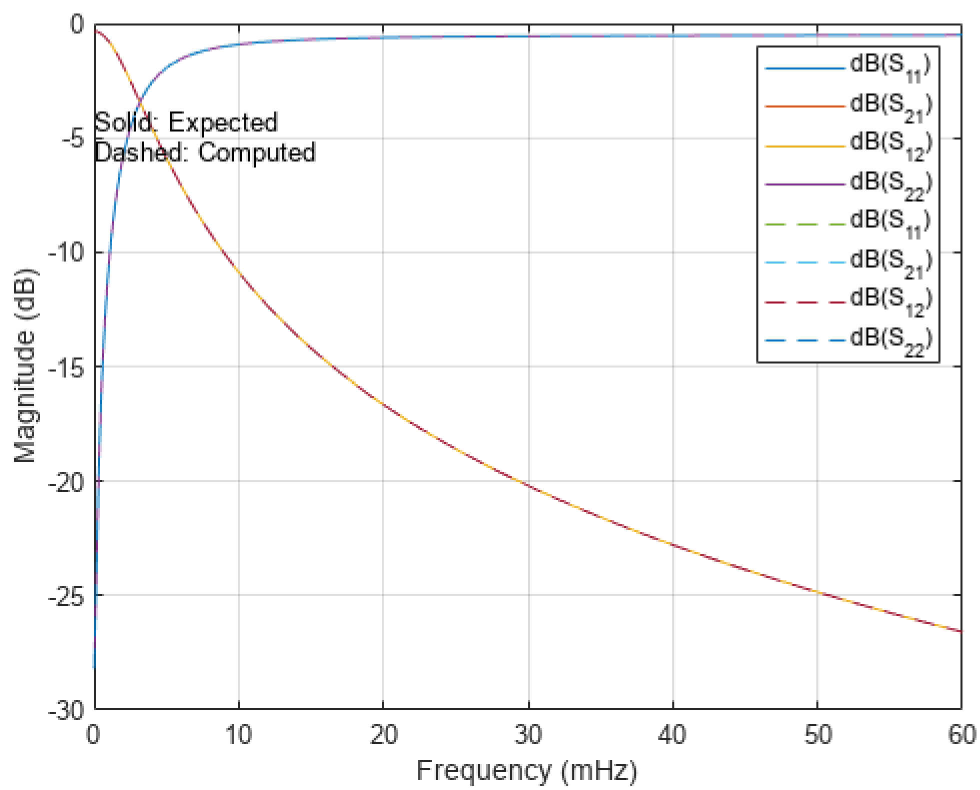
Compare Cascaded S-Parameters of Probe1 with S-Parameters of Combined Network

Cascade S-parameters of **probe1** with itself using `cascadesparams` function and create an S-parameter object with cascaded S-parameters.

```
S_combined = cascadesparams(S_probe,S_probe);
```

Plot and compare S-parameters from **connectedprobes.s2p** and those calculated from combined probe1.

```
figure
rfplot(S)
hold on
rfplot(S_combined,'--')
hold off
text(0.02,-5,{'Solid: Expected','Dashed: Computed'})
```



Limitations

The procedure shown here cannot replace traditional calibration. We include it as an example of using RF Toolbox™ and MATLAB™ to manipulate network parameters mathematically.

There are some limitations to using this procedure.

- There is no guaranteed solution. Some matrices do not have a square root.

- The solution may not be unique. Often, there are two or more viable matrix square roots.

See Also

More About

- “De-Embedding S-Parameters” on page 6-113
- “Extract S-Parameters from Circuit” on page 6-237
- “Extract S-Parameters from Mutual Inductor” on page 6-241

Designing Matching Networks for Low Noise Amplifiers

This example shows how to verify the design of input and output matching networks for a Low Noise Amplifier (LNA) using gain and noise figure plot.

In wireless communications, receivers need to be able to detect and amplify incoming low-power signals without adding much noise. Therefore, an LNA is often used as the first stage of these receivers. To design an LNA, this example uses the available gain design technique, which involves selecting an appropriate matching network that provides a suitable compromise between gain and noise.

In this example, to design matching networks for an LNA, the `rfckt.amplifier` object and the `analyze` method are used to examine the transducer power gains, the available power gain, and the maximum available power gain. The method `circle` is used to determine optimal source reflection coiefficent, GammaS and the function `fzero` is used in amplifier stabilization.

LNA Design Specifications

The LNA design specifications are as follows:

- Frequency range: 5.10 - 5.30 GHz
- Noise Figure <= 2.2 dB
- Transducer Gain > 11 dB
- Operating between 50-ohm terminations

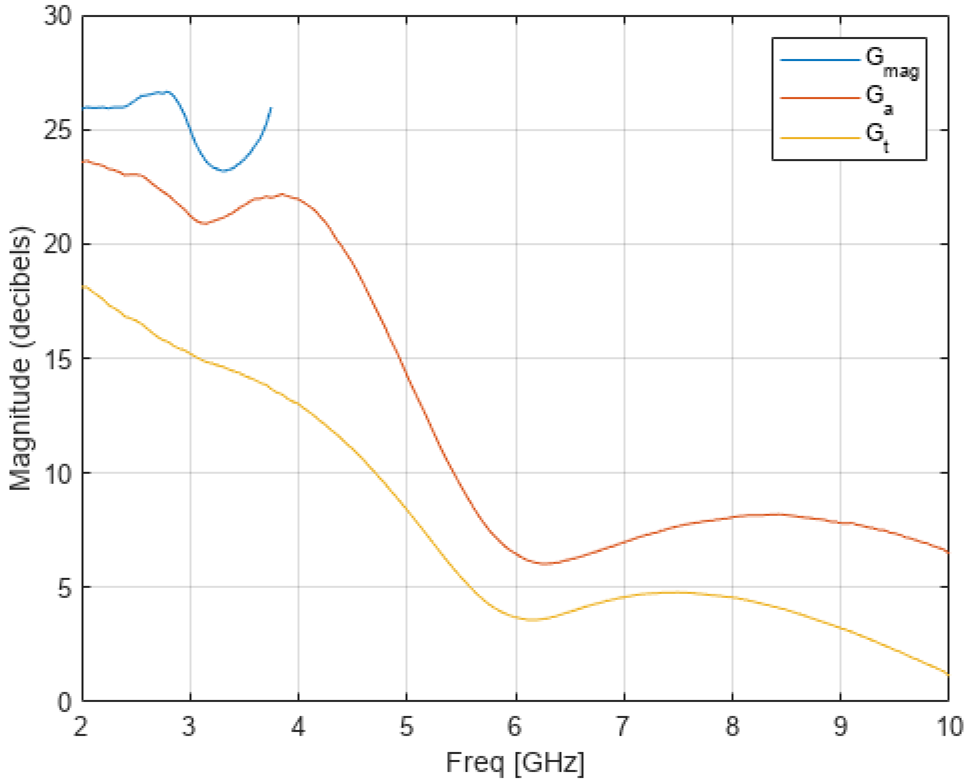
Create `rfckt.amplifier` Object and Examine Amplifier Power Gains and Noise Figure

Create an `rfckt.amplifier` object to represent the amplifier that is specified in the file, 'samplelna1.s2p'. Analyze the amplifier using the `analyze` function the amplifier in the frequency range from 2 - 10 GHz.

```
unmatched_amp = read(rfckt.amplifier, 'samplelna1.s2p');
analyze(unmatched_amp, 2e9:50e6:10e9);
```

Plot the transducer power gain (`Gt`), the available power gain (`Ga`) and the maximum available power gain (`Gmag`).

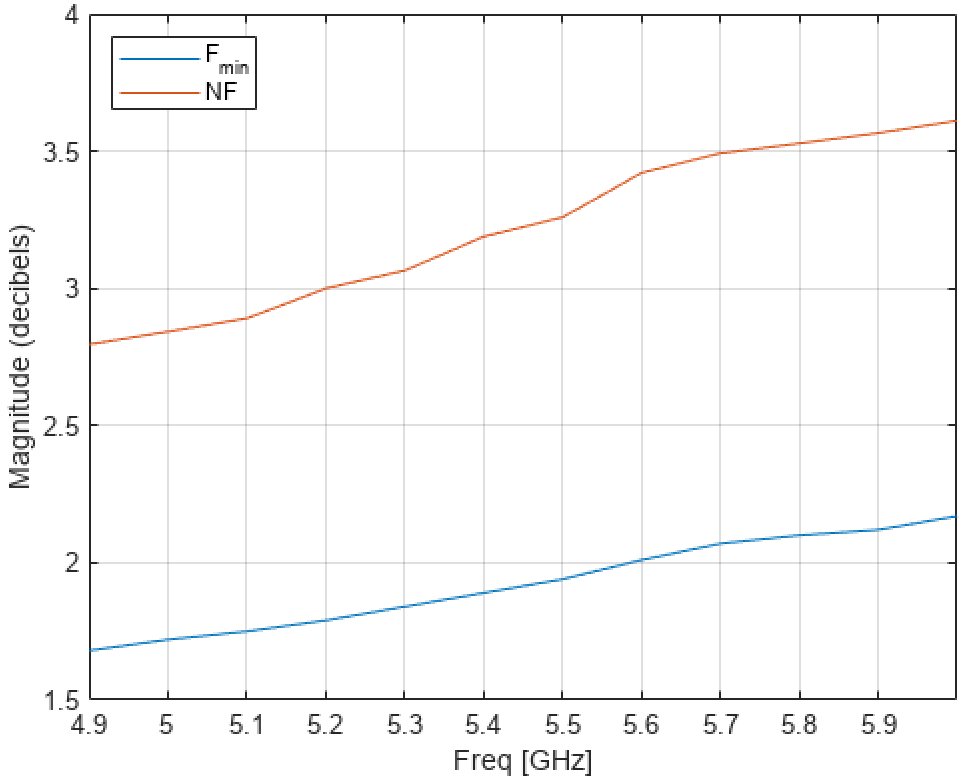
```
figure
plot(unmatched_amp,'Gmag','Ga','Gt','dB')
```



Examine the power gains at 5.2 GHz in order to design the input and output matching networks 5.2 GHz. Without the input and output matching networks, the transducer power gain at 5.2 GHz is about 7.2 dB. This is below the gain requirement of 11 dB in the design specifications and less than the available power gain. This amplifier is also potentially unstable at 5.2 GHz, since the maximum available gain does not exist at 5.2 GHz.

Plot the measured minimum noise figure (`Fmin`) and the noise figure (`NF`) calculated when there is no input matching network. Specify an *x*-axis range of 4.9 GHz to 6 GHz, where the minimum noise figure is measured.

```
plot(unmatched_amp,'Fmin','NF','dB')
axis([4.9 6 1.5 4])
legend('Location','NorthWest')
```

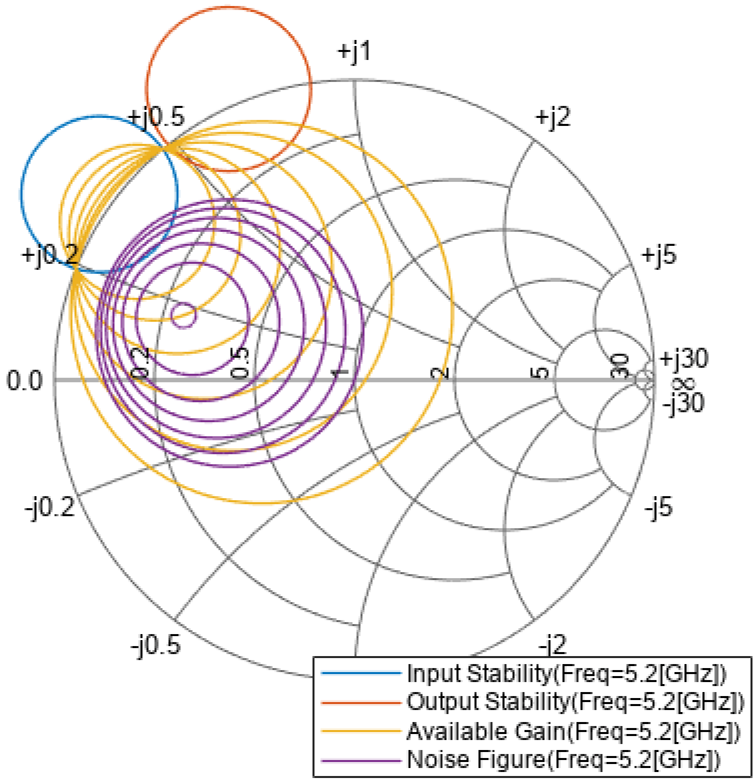


In the absence of an input matching network, the noise figure is between 5.10 - 5.30 GHz which is above the noise figure requirement of 2.2 dB in the specification.

Plot Gain, Noise Figure, and Stability Circles

Both the available gain and the noise figure are functions of the source reflection coefficient, GammaS. To select an appropriate GammaS that provides a suitable compromise between gain and noise, use the `circle` method of the `rfckt.amplifier` object to place the constant available gain and the constant noise figure circles on the Smith chart. As mentioned earlier, the amplifier is potentially unstable at 5.2 GHz. Therefore, the following `circle` command also places the input and output stability circles on the Smith chart.

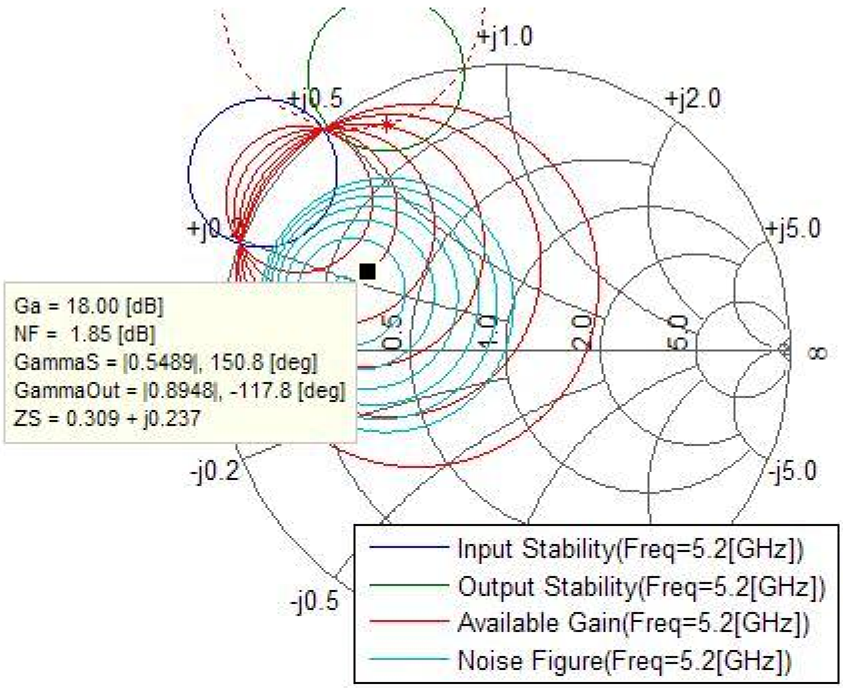
```
fc = 5.2e9;
hsm = smithplot;
circle(unmatched_amp,fc,'Stab','In','Stab','Out','Ga',10:2:20, ...
    'NF',1.8:0.2:3,hsm);
legend('Location','SouthEast')
```



Enable the data cursor and click on the constant available gain circle. The data tip displays the following data:

- Available power gain (`Ga`)
- Noise figure (`NF`)
- Source reflection coefficient (`GammaS`)
- Output reflection coefficient (`GammaOut`)
- Normalized source impedance (`ZS`)

`Ga`, `NF`, `GammaOut` and `ZS` are all functions of the source reflection coefficient, `GammaS`. `GammaS` is the complex number that corresponds to the location of the data cursor. A star ('*') and a circle-in-dashed-line will also appear on the Smith chart. The star represents the matching load reflection coefficient (`GammaL`) that is the complex conjugate of `GammaOut`. The gain is maximized when `GammaL` is the complex conjugate of `GammaOut`. The circle-in-dashed-line represents the trajectory of the matching `GammaL` when the data cursor moves on a constant available gain or noise figure circle.



Because both the `S11` and `S22` parameters of the amplifier are less than unity in magnitude, both the input and output stable region contain the center of the Smith chart. In order to make the amplifier stable, `GammaS` must be in the input stable region and the matching `GammaL` must be in the output stable region. The output stable region is shaded in the above figure. However, when a `GammaS` that gives a suitable compromise between gain and noise is found, the matching `GammaL` always falls outside the output stable region. This makes amplifier stabilization necessary.

Amplifier Stabilization

One way to stabilize an amplifier is to cascade a shunt resistor at the output of the amplifier. However, this approach will also reduce gain and add noise. At the end of the example, you will notice that the overall gain and noise still met the requirement.

To find the maximum shunt resistor value that makes the amplifier unconditionally stable, use the `fzero` function to find the resistor value that makes stability `MU` equal to 1. The `fzero` function always tries to achieve a value of zero for the objective function, so the objective function should return `MU-1`.

```
type('lna_match_stabilization_helper.m')
```

```
function mu_minus_1 = lna_match_stabilization_helper(propval, fc, ckt, element, propname)
%LNA_MATCH_STABILIZATION_HELPER Return Stability MU-1.
%   MU_MINUS_1 = LNA_MATCH_STABILIZATION_HELPER(PROPVALUE, FC, CKT,
%   ELEMENT, PROPNAME) returns stability parameter MU-1 of a circuit, CKT
%   when the property called PROPNAME of an element, ELEMENT is set to
%   PROPVAL.
%
```

```
%   LNA_MATCH_STABILIZATION_HELPER is a helper function of RF
%   Toolbox demo: Designing Matching Networks (Part 1: Networks with an LNA
%   and Lumped Elements).

%   Copyright 2007-2008 The MathWorks, Inc.

set(element, propname, propval)
analyze(ckt, fc);
mu_minus_1 = stabilitymu(ckt.AnalyzedResult.S_Parameters) - 1;
```

Compute the parameters for objective function and pass the objective function to `fzero` to get the maximum shunt resistor value.

```
stab_amp = rfckt.cascade('ckts', {unmatched_amp, rfckt.shuntrlc});
R1 = fzero(@(R1) lna_match_stabilization_helper(R1,fc,stab_amp,stab_amp.Ckts{2},'R'),[1 1e5])
```

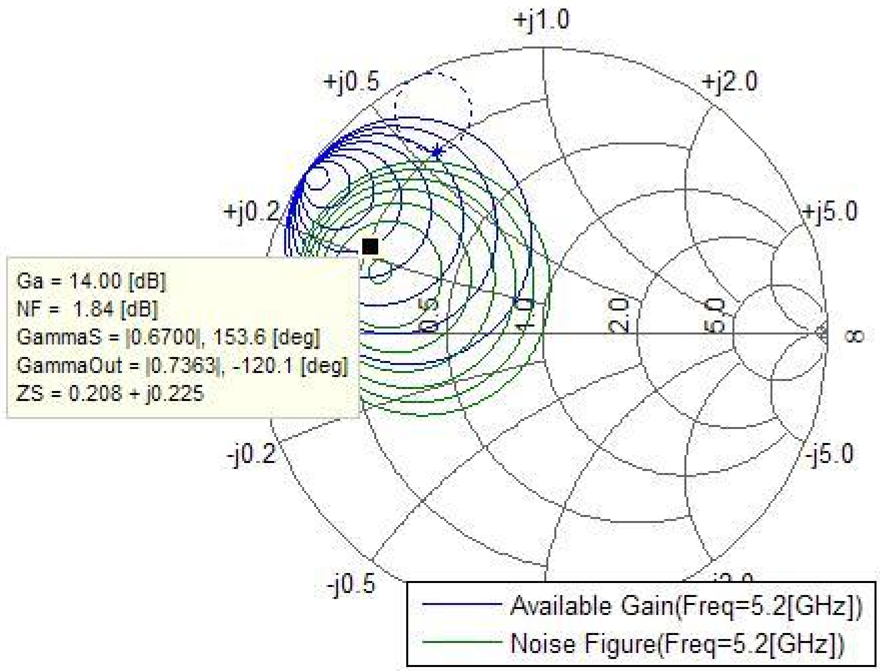
```
R1 = 118.6213
```

Find GammaS and GammaL

Cascade a 118-ohm resistor at the output of the amplifier and analyze the cascaded network. Place the new constant available gain and the constant noise figure circles on the Smith chart.

```
shunt_r = rfckt.shuntrlc('R',118);
stab_amp = rfckt.cascade('ckts',{unmatched_amp,shunt_r});
analyze(stab_amp,fc);
hsm = smithplot;
circle(stab_amp,fc,'Ga',10:17,'NF',1.80:0.2:3,hsm)
legend('Location','SouthEast')
```

Use the data cursor to locate a `GammaS`. You can find that there is a suitable compromise between gain and noise.



The example is designined to select a `GammaS` that gives a gain of 14 dB and noise figure of 1.84 dB. Compute the matching `GammaL`, which is the complex conjugate of `GammaOut` on the data tip.

```
GammaS = 0.67*exp(1j*153.6*pi/180)
```

```
GammaS = -0.6001 + 0.2979i
```

Compute the normalized source impedance.

```
Zs = gamma2z(GammaS,1)
```

```
Zs = 0.2080 + 0.2249i
```

Compute the matching `GammaL` that is equal to the complex conjugate of `GammaOut`.

```
GammaL = 0.7363*exp(1j*120.1*pi/180)
```

```
GammaL = -0.3693 + 0.6370i
```

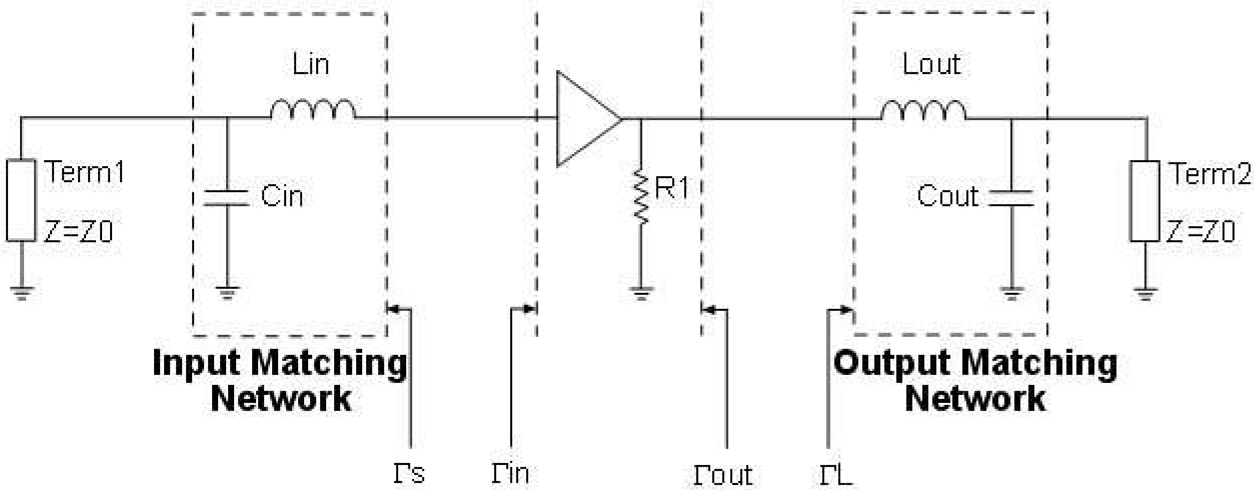
Compute the normalized load impedance.

```
Zl = gamma2z(GammaL,1)
```

```
Zl = 0.2008 + 0.5586i
```

Design Input Matching Network Using GammaS

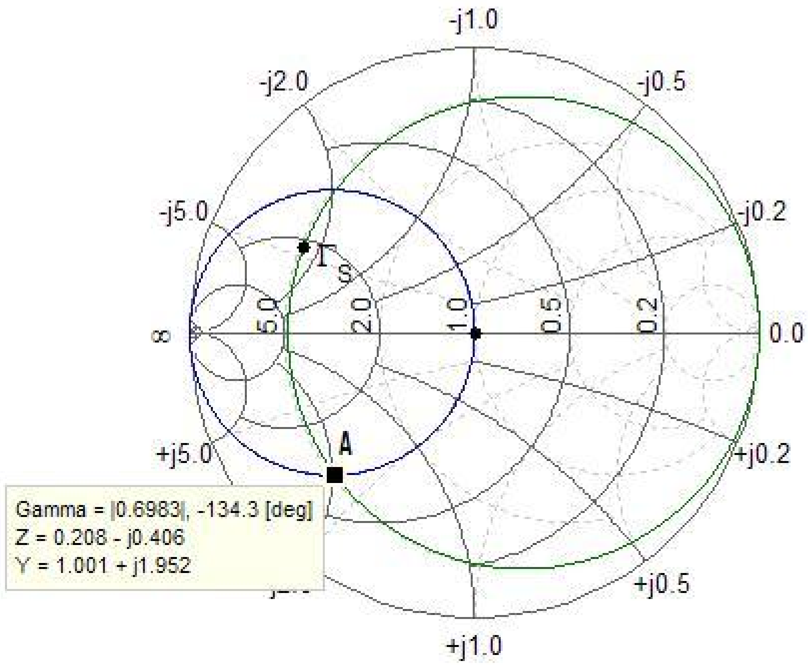
In this example, the lumped LC elements are used to build the input and output matching networks as follows:



The input matching network consists of one shunt capacitor, Cin, and one series inductor, Lin. Use the Smith chart and the data cursor to find component values. To do this, start by plotting the constant conductance circle that crosses the center of the Smith chart and the constant resistance circle that crosses `GammaS`.

```
hsm = smithplot;
circle(stab_amp,fc,'G',1,'R',real(Zs),hsm);
hsm.GridType = 'YZ';
hold all
plot(GammaS,'k.','MarkerSize',16)
text(real(GammaS)+0.05,imag(GammaS)-0.05,'\Gamma_{S}','FontSize', 12, ...
    'FontUnits','normalized')
plot(0,0,'k.','MarkerSize',16)
hold off
```

Then, find the intersection points of the constant conductance and the constant resistance circle. Based on the circuit diagram above, the intersection point in the lower half of the Smith chart should be used. Mark it as point A.




```
GammaA = 0.6983*exp(1j*(-134.3)*pi/180);
Za = gamma2z(GammaA,1);
Ya = 1/Za;
```

Determine the value of `Cin` from the difference in susceptance from the center of the Smith chart to point A. Namely,

$$2\pi f_c C_{in} = \mathrm{Im}\left(\frac{Y_a}{50}\right)$$

where 50 is the reference impedance.

```
Cin = imag(Ya)/50/2/pi/fc
```

```
Cin = 1.1945e-12
```

Determine the value of `Lin` from the difference in reactance from point A to `GammaS`. Namely,

$$2\pi f_c L_{in} = 50(\mathrm{Im}(Z_s) - \mathrm{Im}(Z_a))$$

```
Lin = (imag(Zs) - imag(Za))*50/2/pi/fc
```

```
Lin = 9.6522e-10
```

Design Output Matching Network Using GammaL

Use the approach described in the previous section on designing the input matching network to design the output matching network and get the values of `Cout` and `Lout`.

```
GammaB = 0.7055*exp(1j*(-134.9)*pi/180);
Zb = gamma2z(GammaB, 1);
Yb = 1/Zb;
Cout = imag(Yb)/50/2/pi/fc
```

```
Cout = 1.2194e-12
```

```
Lout = (imag(Zl) - imag(Zb))*50/2/pi/fc
```

```
Lout = 1.4682e-09
```

Verify Design

Create the input and output matching networks. Cascade the input matching network, the amplifier, the shunt resistor and the output matching network to build the LNA.

```
input_match = rfckt.cascade('Ckts', ...
    {rfckt.shuntrlc('C',Cin),rfckt.seriesrlc('L',Lin)});
output_match = rfckt.cascade('Ckts', ...
    {rfckt.seriesrlc('L',Lout),rfckt.shuntrlc('C',Cout)});
LNA = rfckt.cascade('ckts', ...
    {input_match,unmatched_amp,shunt_r,output_match});
```

Analyze the LNA around the design frequency range and plot the available and transducer power gain. The available and transducer power gain at 5.2 GHz are both 14 dB as the design intended. The transducer power gain is above 11 dB in the design frequency range, which meets the requirement in the specification.

```
analyze(LNA,5.05e9:10e6:5.35e9);
plot(LNA,'Ga','Gt','dB');
```

Plot the noise figure around the design frequency range.

```
plot(LNA,'NF','dB')
```

The noise figure is below 2.2 dB in the design frequency range, which also meets the requirement in the specification. The noise figure of the LNA at 5.2 GHz is about 0.1 dB above that of the amplifier (1.84 dB), which demonstrates added noise by the shunt resistor.

The available gain design method is often used in LNA matching. In the second part of the example -- “Designing Matching Networks (Part 2: Single Stub Transmission Lines)” on page 6-132, a simultaneous conjugate matching example is presented.

See Also

More About

- “Designing Matching Networks (Part 2: Single Stub Transmission Lines)” on page 6-132
- “Design Broadband Matching Networks for Antennas” on page 6-140

Designing Matching Networks (Part 2: Single Stub Transmission Lines)

This example shows how to use the RF Toolbox to determine the input and output matching networks that maximize power delivered to a 50-Ohm load and system. Designing input and output matching networks is an important part of amplifier design. This example first calculates the reflection factors for simultaneous conjugate match and then determines the placement of a shunt stub in each matching network at a specified frequency. Finally, the example cascades the matching networks with the amplifier and plots the results.

Create an `rfckt.amplifier` Object

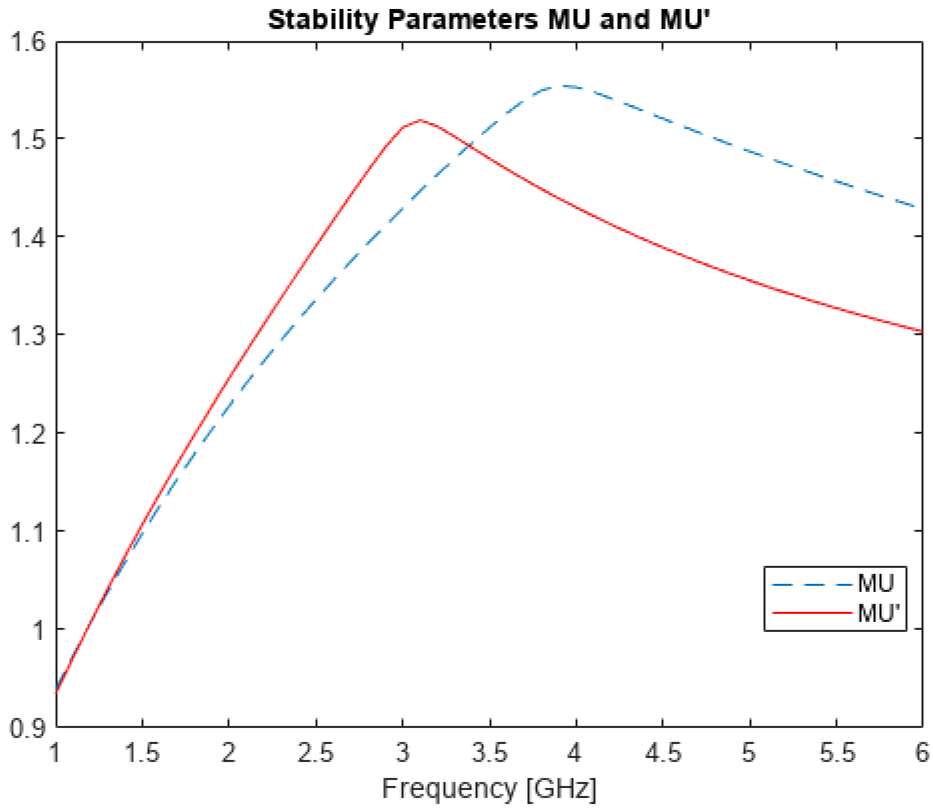
Create an `rfckt.amplifier` object to represent the amplifier described by the measured frequency-dependent S-parameter data in the file `samplebjt2.s2p`. Then, `extract` the frequency-dependent S-parameter data from the `rfckt.amplifier` object.

```
amp = read(rfckt.amplifier,'samplebjt2.s2p');
[sparams,AllFreq] = extract(amp.AnalyzedResult,'S_Parameters');
```

Check for Amplifier Stability

Before proceeding with the design, determine the measured frequencies at which the amplifier is unconditionally stable. Use the `stabilitymu` function to calculate `mu` and `muprime` at each frequency. Then, check that the returned values for `mu` are greater than one. This criteria is a necessary and sufficient condition for unconditional stability. If the amplifier is not unconditionally stable, print out the corresponding frequency value.

```
[mu,muprime] = stabilitymu(sparams);
figure
plot(AllFreq/1e9,mu,'--',AllFreq/1e9,muprime,'r')
legend('MU',"MU'",'Location','Best')
title("Stability Parameters MU and MU'")
xlabel('Frequency [GHz]')
```




```
disp('Measured Frequencies where the amplifier is not unconditionally stable:')
```

```
Measured Frequencies where the amplifier is not unconditionally stable:
```

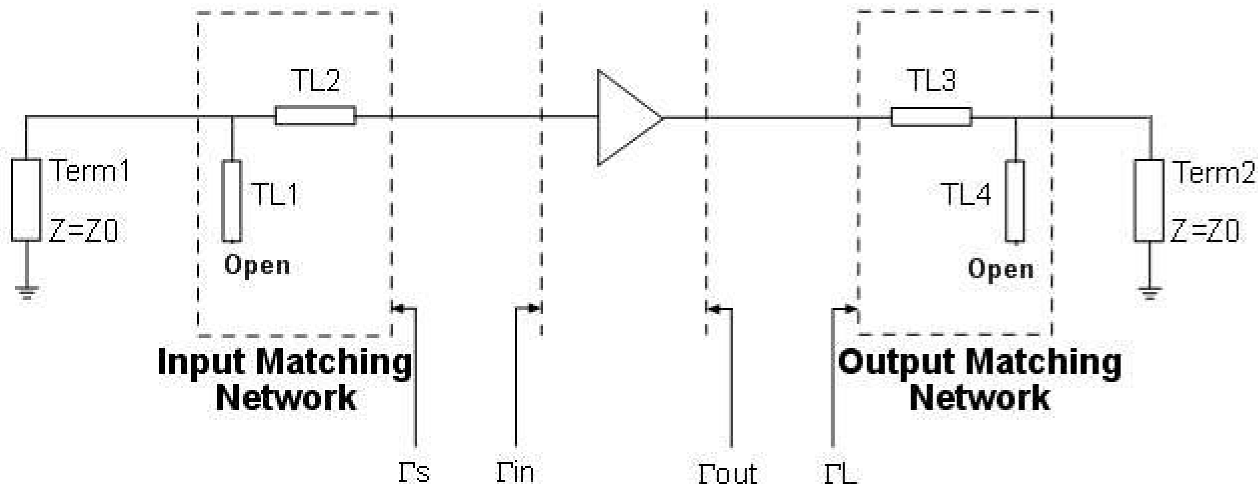
```
fprintf('\tFrequency = %.1e\n',AllFreq(mu<=1))
```

```
    Frequency = 1.0e+09
    Frequency = 1.1e+09
```

For this example, the amplifier is unconditionally stable at all measured frequencies except 1.0 GHz and 1.1 GHz.

Determine the Source and Load Matching Networks for a Simultaneous Conjugate Match

Begin designing the input and output matching networks by transforming the reflection coefficients for simultaneous conjugate match at the amplifier interfaces into the appropriate source and load admittance. This example uses the following lossless transmission line matching scheme:



The design parameters for this single stub matching scheme are the location of the stubs with reference to the amplifier interfaces and the stub lengths. The procedure uses the following design principles:

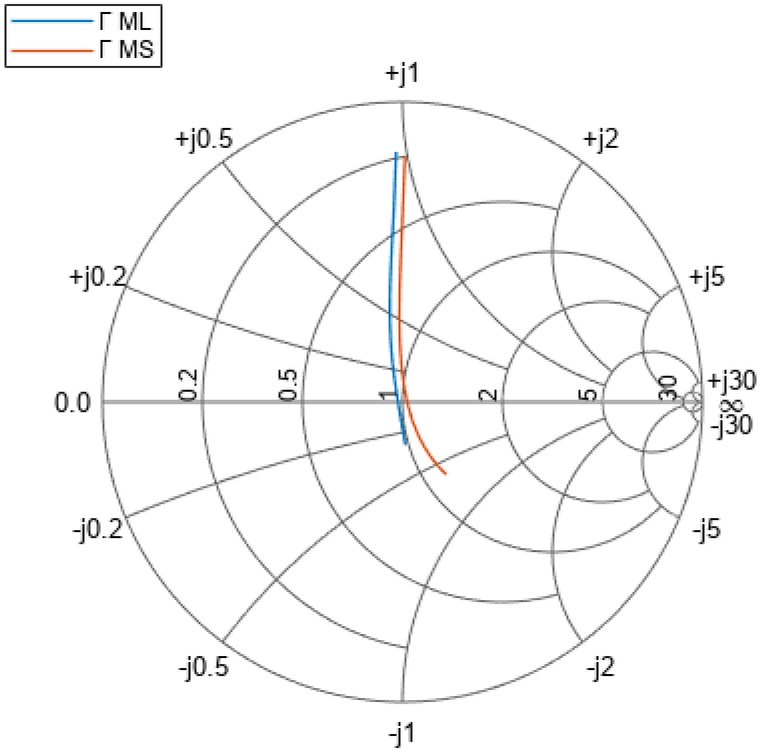
- The center of the Smith chart represents a normalized source or load immittance.
- Movement along a transmission line is equivalent to traversing a circle centered at the origin of the Smith chart with radius equal to a reflection coefficient magnitude.
- A single transmission line stub can be inserted at the point on a transmission line when its admittance (transmission line) intersects the unity conductance circle. At this location, the stub will negate the transmission line susceptance, resulting in a conductance that equals the load or source terminations.

This example uses the YZ Smith chart because it's easier to add a stub in parallel with a transmission line using this type of Smith chart.

Calculate and Plot the Complex Load and Source Reflection Coefficients

`calculate` and plot all complex load and source reflection coefficients for simultaneous conjugate match at all measured frequency data points that are unconditionally stable. These reflection coefficients are measured at the amplifier interfaces.

```
AllGammaL = calculate(amp,'GammaML','none');
AllGammaS = calculate(amp,'GammaMS','none');
hsm = smithplot([AllGammaL{:} AllGammaS{:}]);
hsm.LegendLabels = {'#Gamma ML','#Gamma MS'};
```



Determine the Load Reflection Coefficient at a Single Frequency

Find the load reflection coefficient, `GammaL`, for the output matching network at the design frequency 1.9 GHz.

```
freq = AllFreq(AllFreq == 1.9e9);
GammaL = AllGammaL{1}(AllFreq == 1.9e9)
```

```
GammaL = -0.0421 + 0.2931i
```

Draw the Constant Magnitude Circle for Load Reflection Coefficient GammaL

Draw a circle that is centered at the normalized admittance Smith chart origin and whose radius equals the magnitude of `GammaL`. A point on this circle represents the reflection coefficient at a particular location on the transmission line. The reflection coefficient for the transmission line at the amplifier interface is `GammaL`, while the center of the chart represents the normalized load admittance, `y_L`. The example uses the `circle` method to draw all appropriate circles on a Smith chart.

```
hsm = smithplot;
circle(amp,freq,'Gamma',abs(GammaL),hsm);
hsm.GridType = 'yz';
hold all
plot(0,0,'k.','MarkerSize',16)
plot(GammaL,'k.','MarkerSize',16)
txtstr = sprintf('\\Gamma_{L}\\fontsize{8}\\bf=\\mid%s\\mid%s^\\circ', ...
    num2str(abs(GammaL),4),num2str((angle(GammaL)*180/pi),4));
```

```
text(real(GammaL),imag(GammaL)+.1,txtstr,'FontSize',10, ...
    'FontUnits','normalized');
plot(0,0,'r',0,0,'k.','LineWidth',2,'MarkerSize',16);
text(0.05,0,'y_L','FontSize',12,'FontUnits','normalized')
```

Draw the Unity Constant Conductance Circle and Find Intersection Points

To determine the stub wavelength (susceptance) and its location with respect to the amplifier load matching interface, plot the normalized unity conductance circle and the constant magnitude circle and figure out where the two circles intersect. Find the points of intersection interactively using the data cursor or analytically using the helper function, `imped_match_find_circle_intersections_helper`. This example uses the helper function. The circles intersect at two points. The example uses the third-quadrant point, which is labeled "A". The unity conductance circle is centered at (-.5,0) with radius .5. The constant magnitude circle is centered at (0,0) with radius equal to the magnitude of `GammaL`.

```
circle(amp,freq,'G',1,hsm);
hsm.ColorOrder(2,:) = [1 0 0];
[~,pt2] = imped_match_find_circle_intersections_helper([0 0], ...
    abs(GammaL),[-.5 0],.5);
GammaMagA = sqrt(pt2(1)^2 + pt2(2)^2);
GammaAngA = atan2(pt2(2),pt2(1));
ax = hsm.Parent.CurrentAxes;
hold (ax,"on");
plot(ax, pt2(1),pt2(2),'k.','MarkerSize',16);
txtstr = sprintf('A=\\mid%s\\mid%s^\\circ',num2str(GammaMagA,4), ...
    num2str(GammaAngA*180/pi,4));
text(ax, pt2(1),pt2(2)-.07,txtstr,'FontSize',8,'FontUnits','normalized', ...
    'FontWeight','Bold')
container = hsm.Parent;
annotation(container,'textbox','VerticalAlignment','middle',...
    'String',{'Unity','Conductance','Circle'},...
    'HorizontalAlignment','center','FontSize',8,...
    'EdgeColor',[0.04314 0.5176 0.7804],...
    'BackgroundColor',[1 1 1],'Position',[0.1403 0.1608 0.1472 0.1396])
annotation(container,'arrow',[0.2786 0.3286],[0.2778 0.3310])
annotation(container,'textbox','VerticalAlignment','middle',...
    'String',{'Constant','Magnitude','Circle'},...
    'HorizontalAlignment','center','FontSize',8,...
    'EdgeColor',[0.04314 0.5176 0.7804],...
    'BackgroundColor',[1 1 1],'Position',[0.8107 0.3355 0.1286 0.1454])
annotation(container,'arrow',[0.8179 0.5761],[0.4301 0.4887]);
```

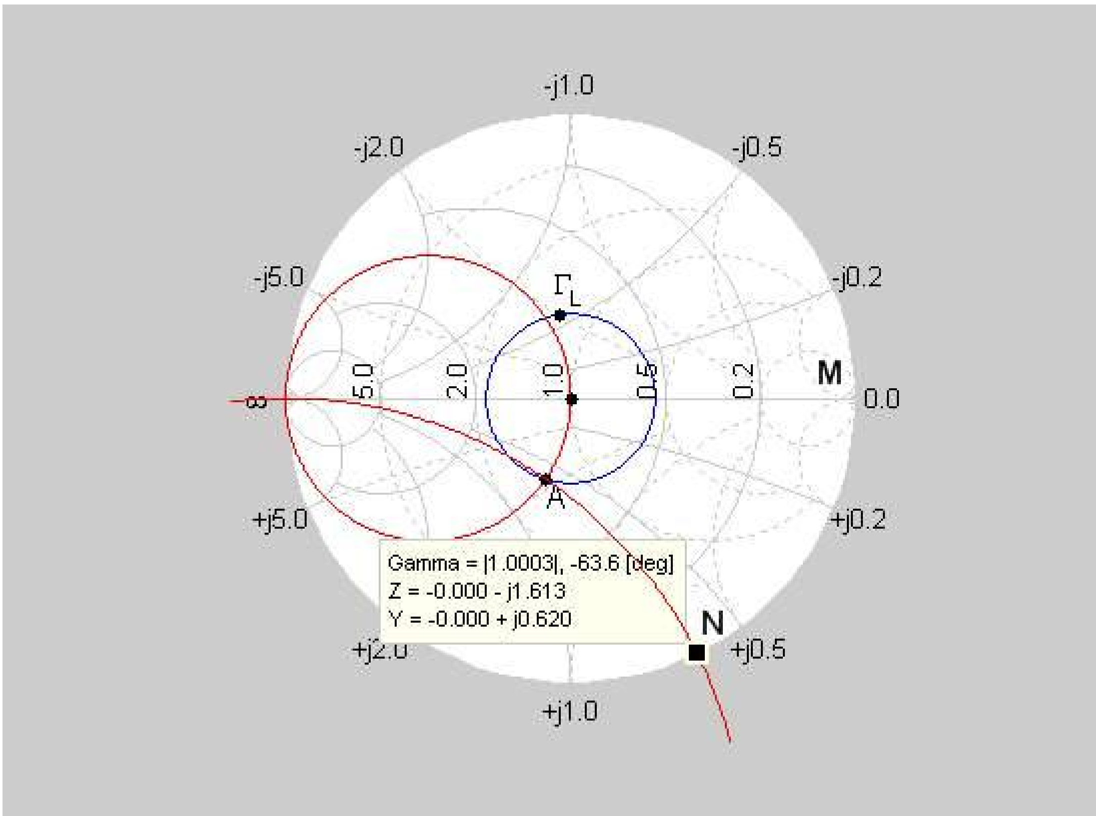
Calculate the Stub Location and the Stub Length for the Output Matching Network

The open-circuit stub location in wavelengths from the amplifier load interface is a function of the clockwise angular difference between point "A" and `GammaL`. When point "A" appears in the third quadrant and `GammaL` falls in the second quadrant, the stub position in wavelengths is calculated as follows:

```
StubPositionOut = ((2*pi + GammaAngA) - angle(GammaL))/(4*pi)
```

```
StubPositionOut = 0.2147
```

The stub value is the amount of susceptance that is required to move the normalized load admittance (the center of the Smith chart) to point "A" on the constant magnitude circle. An open stub transmission line can be used to supply this value of susceptance. Its wavelength is defined by the amount of angular rotation from the open-circuit admittance point on the Smith chart (point "M" on the following figure) to the required susceptance point "N" on the outer edge of the chart. Point "N" is where a constant susceptance circle with a value equal to the susceptance of point "A" intersects the unit circle. In addition, the `StubLengthOut` formula used below requires "N" to fall in the third or fourth quadrant.




```
GammaA = GammaMagA*exp(1j*GammaAngA);
bA = imag((1 - GammaA)/(1 + GammaA));
StubLengthOut = -atan2(-2*bA/(1 + bA^2),(1 - bA^2)/(1 + bA^2))/(4*pi)
```

```
StubLengthOut = 0.0883
```

Calculate the Stub Location and the Stub Length for the Input Matching Network

In the previous sections, the example calculated the required lengths and placements, in wavelengths, for the output matching transmission network. Following the same approach, the line lengths for the input matching network are calculated:

```
GammaS = AllGammaS{1}(AllFreq == 1.9e9)
```

```
GammaS = -0.0099 + 0.2501i
```

```
[pt1,pt2] = imped_match_find_circle_intersections_helper([0 0], ...
    abs(GammaS),[-.5 0],.5);
GammaMagA = sqrt(pt2(1)^2 + pt2(2)^2);
GammaAngA = atan2(pt2(2),pt2(1));
GammaA = GammaMagA*exp(1j*GammaAngA);
bA = imag((1 - GammaA)/(1 + GammaA));
StubPositionIn = ((2*pi + GammaAngA) - angle(GammaS))/(4*pi)
```

```
StubPositionIn = 0.2267
```

```
StubLengthIn = -atan2(-2*bA/(1 + bA^2),(1 - bA^2)/(1 + bA^2))/(4*pi)
```

```
StubLengthIn = 0.0759
```

Verify the Design

To verify the design, assemble a circuit using 50-Ohm microstrip transmission lines for the matching networks. First, determine if the microstrip line is a suitable choice by analyzing the default microstrip transmission line at a design frequency of 1.9 GHz.

```
stubTL4 = rfckt.microstrip;
analyze(stubTL4,freq);
Z0 = stubTL4.Z0;
```

This characteristic impedance is close to the desired 50-Ohm impedance, so the example can proceed with the design using these microstrip lines.

To calculate the required transmission line lengths in meters for the placement of the stubs, analyze the microstrip to obtain a phase velocity value.

```
phase_vel = stubTL4.PV;
```

Use the phase velocity value, which determines the transmission line wavelength and the stub location to set the appropriate transmission line lengths for the two microstrip transmission lines, TL2 and TL3.

```
TL2 = rfckt.microstrip('LineLength',phase_vel/freq*StubPositionIn);
TL3 = rfckt.microstrip('LineLength',phase_vel/freq*StubPositionOut);
```

Use the phase velocity again to specify stub length and stub mode for each stub.

```
stubTL1 = rfckt.microstrip('LineLength',phase_vel/freq*StubLengthIn, ...
    'StubMode','shunt','Termination','open');
set(stubTL4,'LineLength',phase_vel/freq*StubLengthOut, ...
    'StubMode','shunt','Termination','open')
```

Now cascade the circuit elements and analyze the amplifier with and without the matching networks over the frequency range of 1.5 to 2.3 GHz.

```
matched_amp = rfckt.cascade('Ckts',{stubTL1,TL2,amp,TL3,stubTL4});
analyze(matched_amp,1.5e9:1e7:2.3e9);
analyze(amp,1.5e9:1e7:2.3e9);
```

To verify the simultaneous conjugate match at the input of the amplifier, plot the S11 parameters in dB for both the matched and unmatched circuits.

```
clf
plot(amp,'S11','dB')
hold all
hline = plot(matched_amp,'S11','dB');
hline.Color = 'r';
legend('S_{11} - Original Amplifier', 'S_{11} - Matched Amplifier')
legend('Location','SouthEast')
hold off
```

To verify the simultaneous conjugate match at the output of the amplifier, plot the S22 parameters in dB for both the matched and unmatched circuits.

```
plot(amp,'S22','dB')
hold all
hline = plot(matched_amp,'S22','dB');
hline.Color = 'r';
legend('S_{22} - Original Amplifier', 'S_{22} - Matched Amplifier')
legend('Location','SouthEast')
hold off
```

Finally, plot the transducer gain (`Gt`) and the maximum available gain (`Gmag`) in dB for the matched circuit.

```
hlines = plot(matched_amp,'Gt','Gmag','dB');
hlines(2).Color = 'r';
```

You can see that the transducer gain and the maximum available gain are very close to each other at 1.9 GHz.

See Also

More About

- "Designing Matching Networks for Low Noise Amplifiers" on page 6-122
- "Design Broadband Matching Networks for Antennas" on page 6-140
- "Design Broadband Matching Networks for Amplifier" on page 6-149

Design Broadband Matching Networks for Antennas

This example shows how to design a broadband matching network between a resistive source and inductive load using optimization with direct search methods.

In an RF system, a matching network circuit plays a vital role in transferring maximum power between source and the load of the system. In most RF systems, such as wireless devices, a design parameter called operation bandwidth is specified. By taking the operation bandwidth into consideration, the purpose of the matching network is further extended to provide maximum power transfer over a range of frequencies. Alternatively, you can use the L - section matching (conjugate match) approach, guarantees maximum power transfer it does so only at a single frequency.

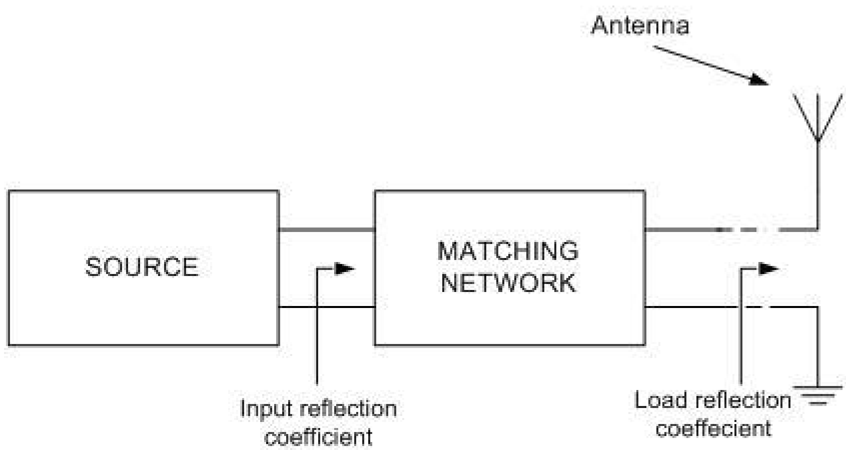


Figure 1: Impedance matching of an antenna to a source

To design a broadband matching network, first set the design parameters such as center frequency, bandwidth, and impedances of source, load and reference. Then calculate the load reflection coefficient and power gain to determine the frequency at which the matching network of the antenna must operate and once the design is complete, optimize the derived network.

Specify Frequency and Impedance

Specify the center frequency, 350 MHz, and bandwidth, 110 MHz, of match to build a matching network with a bandpass response.

```
fc = 350e6;
BW = 110e6;
```

Specify the source impedance, the reference impedance and the load resistance. In this example the load `Zl` is modeled as a series R-L circuit. Instead of calculating the load impedance, you could measure the impedance of the load.

```
Zs = 50;                                % Source impedance (ohm)
Z0 = 50;                                % Reference impedance (ohm)
Rl = 40;                                % Load resistance (ohm)
L = 12e-8;                              % Load inductance (Henry)
```

Define the number of frequency points to use for analysis and set up the frequency vector.

```
nfreq = 256;                          % Number of frequency points
fLower = fc - (BW/2);                 % Lower band edge
fUpper = fc + (BW/2);                 % Upper band edge
freq = linspace(fLower,fUpper,nfreq); % Frequency array for analysis
w = 2*pi*freq;                        % Frequency (radians/sec)
```

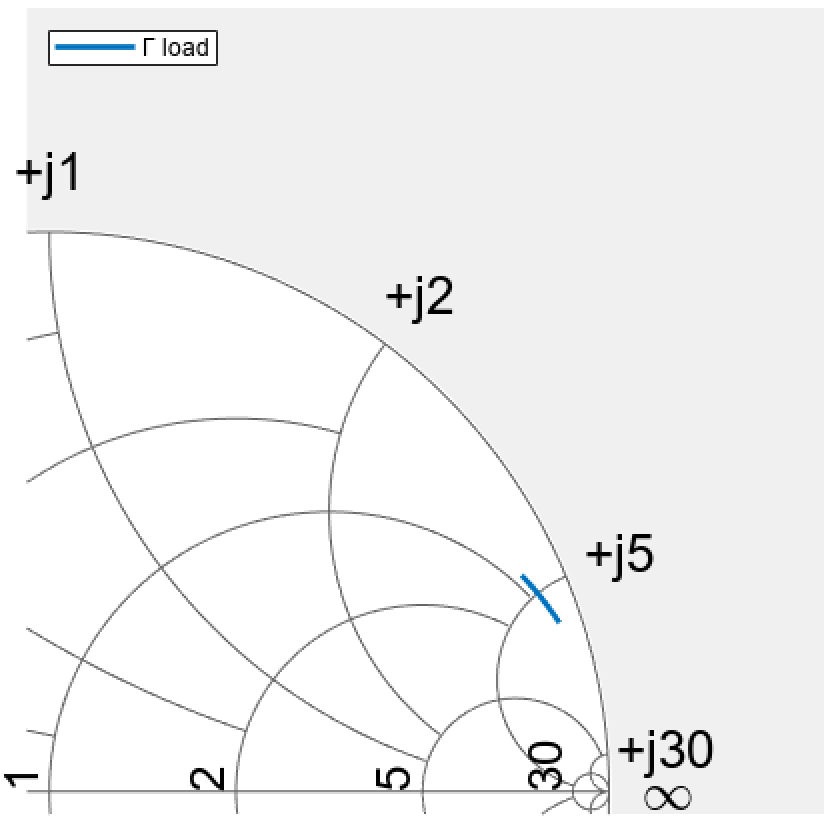
Understand Load Behavior using Reflection Coefficient and Power Gain

Use two simple expressions for calculating the load reflection coefficient and the power gain. This corresponds to directly connecting the source to the input terminals of an antenna i.e. in Figure 1 there is no matching network.

```
Xl = w*L;                               % Reactance (ohm)
Zl = Rl + 1i*Xl;                        % Load impedance (ohm)
GammaL = (Zl - Z0)./(Zl + Z0);          % Load reflection coefficient
unmatchedGt = 10*log10(1 - abs(GammaL).^2); % Power delivered to load
```

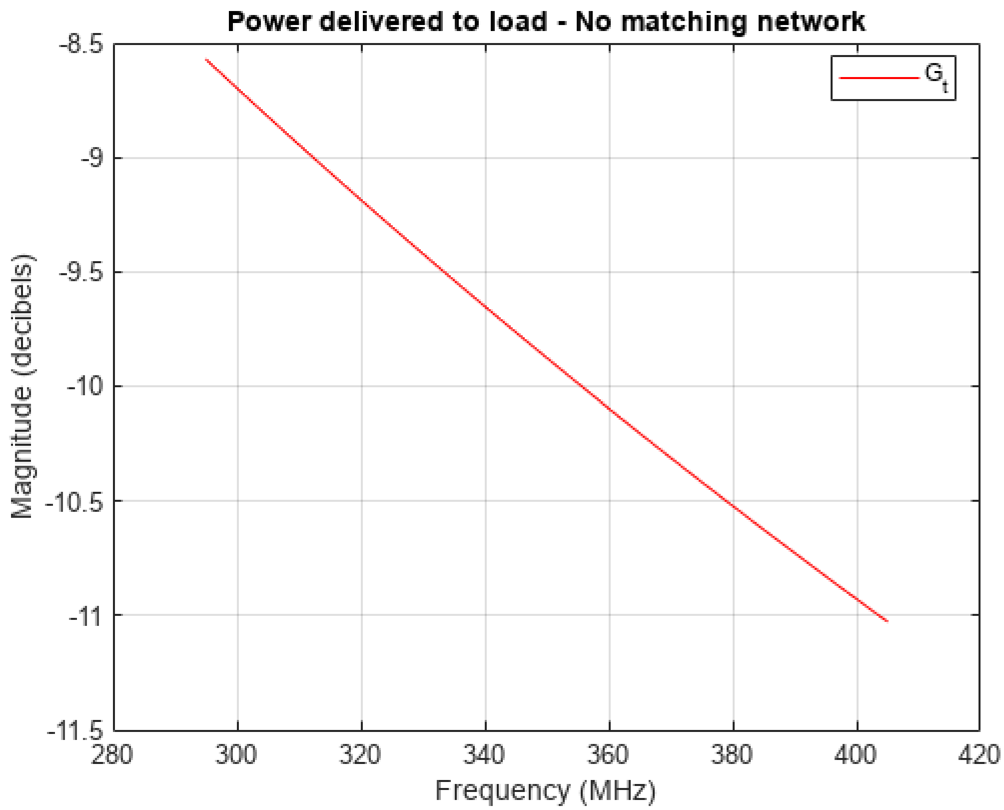
Use the `smithplot` function to plot the variation in the load reflection coefficient with frequency. An input reflection coefficient closer to center of the Smith chart denotes a better matching performance.

```
figure
smithplot(freq,GammaL,'LegendLabels','#Gamma load','LineWidth',2,...
    'View','top-right');
```



This plot shows that the load reflection coefficient is far away from this point. Therefore, there is an impedance mismatch. You can confirm this mismatch by plotting the transducer gain as a function of frequency.

```
figure
plot(freq.*1e-6,unmatchedGt,'r')
grid on;
title('Power delivered to load - No matching network');
xlabel('Frequency (MHz)');
ylabel('Magnitude (decibels)');
legend('G_t','Location','Best');
```



As the plot shows, there is approximately 10 dB power loss around the desired region of operation (295 - 405 MHz). As a result, the antenna needs a matching network that operates over a 110 MHz bandwidth centered at 350 MHz.

Design Matching Network

The matching network must operate between 295 MHz and 405 MHz, therefore you choose a bandpass topology for the matching network shown below.

Type - I: Series LC first element followed by shunt LC

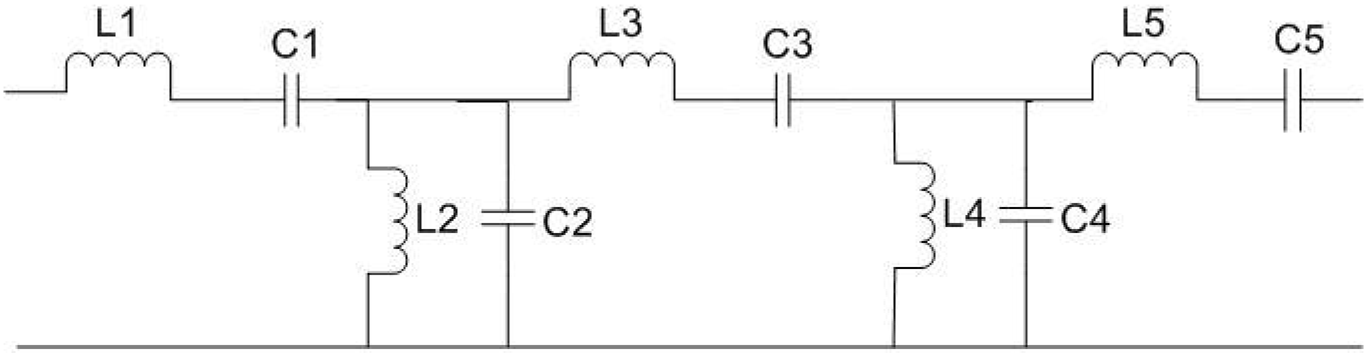


Figure 2: Matching network topology

The approach is to design an odd order 0.5 dB Chebyshev bandpass to obtain the initial design for the matching network shown in figure 2. This is a single match problem [1] on page 6-148, i.e. the source is purely resistive while load is a combination of R and L, solution is you can begin by choosing a five-element prototype network.

```
N = 5; % Order of matching network
filter = rffilter('FilterType',"Chebyshev","FilterOrder",N, ...
    "Implementation","LC Tee","ResponseType","Bandpass",...
    "PassbandFrequency",[fLower fUpper],"PassbandAttenuation",0.5);
Lvals = filter.DesignData.Inductors;
```

Use the `lcladder` object to build the bandpass tee matching network. Note that the topology demands a bandpass tee prototype that begins with a series inductor. If the topology chosen is an LC bandpass pi then you would begin with shunt C for the lowpass prototype.

```
% Create the matching network
matchingNW = lcladder(filter);

% Copy initial values for comparison
L_initial = Lvals;
```

Optimize Designed Matching Network

There are several factors to consider prior to the optimization.

- Objective function - The objective function can be built in different ways depending on the problem at hand. For this example, the objective function is shown in the file below.
- Choice of cost function - The cost function is the function we would like to minimize (maximize) to achieve near optimal performance. There could be several ways to choose the cost function. One obvious choice is the input reflection coefficient, gammaIn. In this example we have chosen to minimize the average reflection coefficient in the passband.
- Optimization variables - In this case it is a vector of values, for the specific elements to optimize in the matching network.
- Optimization method - A direct search based technique, the MATLAB® function `fminsearch`, is used in this example to perform the optimization.

- Number of iterations/function evaluations - Set the maximum number of iterations and function evaluations to perform, so as to tradeoff between speed and quality of match.

The objective function used during the optimization process by `fminsearch` is shown here.

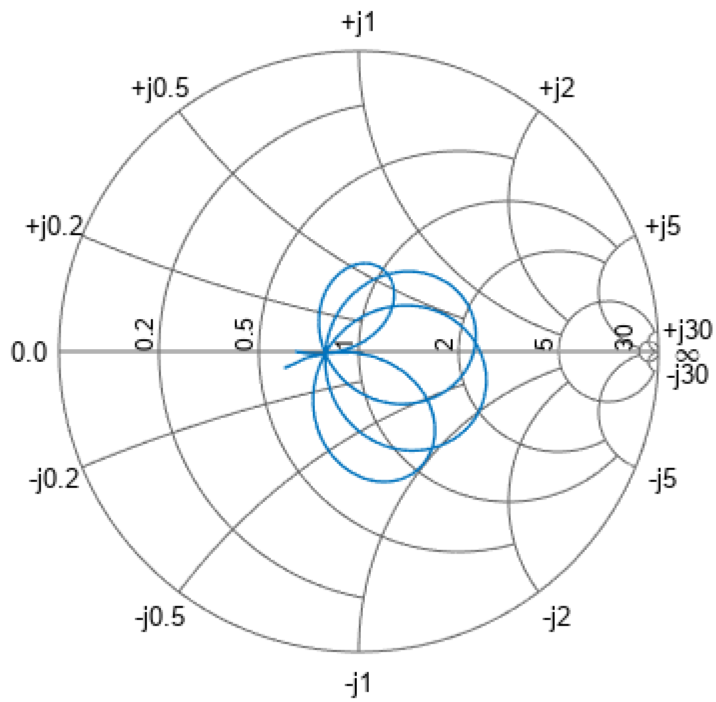
```
type('antennaMatchObjectiveFun.m')
```

```
function output = antennaMatchObjectiveFun(matchingNW,Lvalues,freq,ZL,Z0)
%ANTENNAMATCHOBJECTIVEFUN is the objective function used by the example
% Designing Broadband Matching Networks for Antennas.
%
% OUTPUT = ANTENNAMATCHOBJECTIVEFUN(MATCHINGNW,LVALUES,FREQ,Z0)
% returns the current value of the objective function stored in OUTPUT
% evaluated after updating the inductor values in the object, MATCHINGNW.
% The inductor values are stored in the variable LVALUES.
%
% ANTENNAMATCHOBJECTIVEFUN is an objective function of RF Toolbox demo:
% Designing Broadband Matching Networks (Part I: Antenna)

%   Copyright 2008-2020 The MathWorks, Inc.

% Ensure positive element values
if any(Lvalues <= 0)
    output = Inf;
    return
end

% Update the element values in the matching network
matchingNW.Inductances(1) = Lvalues(1);
matchingNW.Inductances(end) = Lvalues(end);

% Perform analysis on tuned matching network
S = sparameters(matchingNW,freq,Z0);

% Calculate input reflection coefficient 'gammaIn'
gIn = gammain(S,ZL);

% Cost function
output = mean(abs(gIn));

% Other possible choices for objective function could be : -
% output =  max(abs(gIn));
% output = -1*mean(Gt_pass);

% Animate
smithplot(freq,gIn);
drawnow
```

There are several ways to choose the cost function and some options are shown within the objective function above (in comments). The optimization variables are the first and last inductors, L1 and L5 respectively. The element values are stored in the variable `L_Optimized`.

```
niter = 125;
options = optimset('Display','iter','MaxIter',niter);  % Set options structure
L_Optimized = [Lvals(1) Lvals(end)];
L_Optimized = ...
    fminsearch(@(L_Optimized)antennaMatchObjectiveFun(matchingNW, ...
    L_Optimized,freq,Zl,Z0),L_Optimized,options);
```

```
 Iteration   Func-count         f(x)         Procedure
     0            1         0.933982
     1            3         0.933982         initial simplex
     2            5         0.920323         expand
     3            7         0.911353         expand
     4            9         0.853255         expand
     5           11         0.730444         expand
     6           13         0.526448         reflect
     7           15         0.526448         contract inside
     8           17         0.421103         reflect
     9           19         0.421103         contract inside
    10           20         0.421103         reflect
    11           22         0.421103         contract inside
    12           24         0.421103         contract inside
    13           26         0.339935         expand
    14           27         0.339935         reflect
    15           29          0.28528         reflect
    16           31          0.28528         contract inside
    17           32          0.28528         reflect
    18           34         0.283527         reflect
    19           36         0.283527         contract inside
    20           38         0.278939         contract inside
    21           40         0.278123         reflect
    22           41         0.278123         reflect
    23           43          0.27636         contract inside
    24           45         0.275782         contract inside
    25           47         0.275637         contract inside
    26           49         0.275498         reflect
    27           51         0.275282         contract inside
    28           52         0.275282         reflect
    29           54         0.275282         contract inside
```




```
    30             56          0.275282          contract inside

Optimization terminated:
 the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-04
 and F(X) satisfies the convergence criteria using OPTIONS.TolFun of 1.000000e-04
```

Update Matching Network Elements with Optimal Values

When the optimization routine stops, the optimized element values are stored in `L_Optimized`. The following code updates the input and output matching network with these values.

```
matchingNW.Inductances(1) = L_Optimized(1);     % Update the matching network inductor L1
matchingNW.Inductances(end) = L_Optimized(end); % Update the matching network inductor L5
```

Analyze and Display Optimization Results

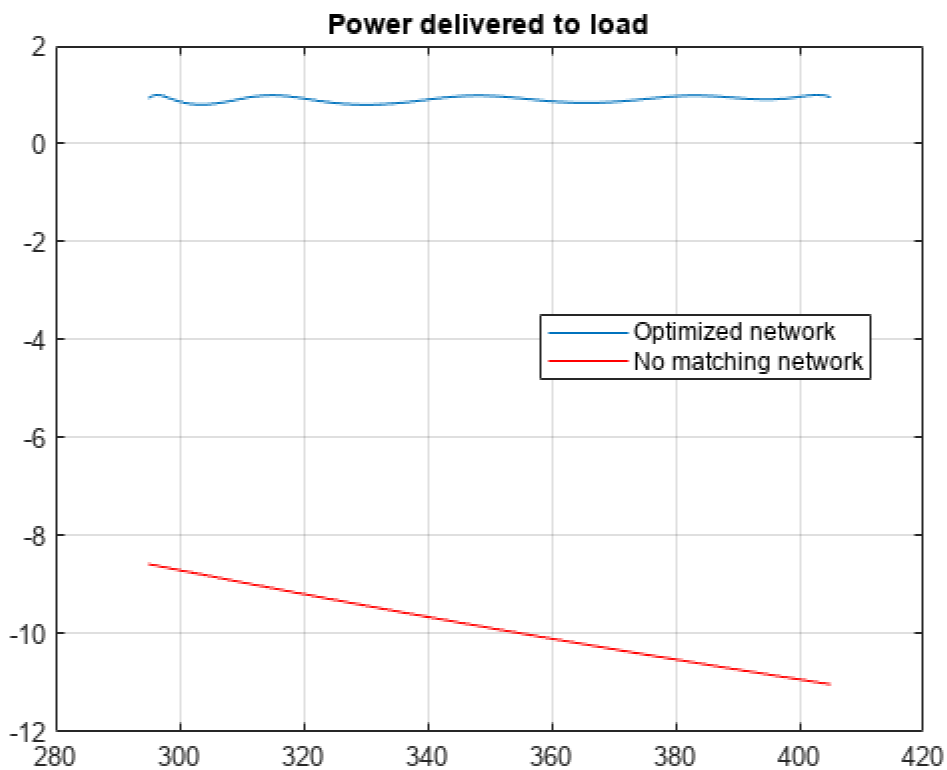
Compare and plot the input reflection coefficient of the matched and unmatched results.

```
S = sparameters(matchingNW,freq,Z0);
gIn = gammain(S,Zl);
smithplot(freq,[gIn transpose(GammaL)],'LegendLabels',...
    {'#Gamma in (Matched)','#Gamma in (Unmatched)'})
```

The optimized matching network improves the performance of the circuit. In the passband (295 - 405 MHz), the input reflection coefficient is closer to the center of the Smith chart. Plot the power delivered to load for both the matched and unmatched system.

```
matchedGt = powergain(S,Zs,Zl,'Gt');
figure;
```

```
plot(freq*1e-6,matchedGt)
hold all;
plot(freq*1e-6,unmatchedGt,'r')
grid on;
hold off;
title('Power delivered to load');
legend('Optimized network','No matching network','Location','Best');
```



The power delivered to the load is approximately 1 dB down for the optimized matching network.

Display Optimized Element Values

The following code shows the initial and optimized values for inductors L1 and L5.

```
L1_Initial = L_initial(1)
```

```
L1_Initial = 1.2340e-07
```

```
L1_Optimized = L_Optimized(1)
```

```
L1_Optimized = 1.2111e-07
```

```
L5_Initial = L_initial(end)
```

```
L5_Initial = 1.2340e-07
```

```
L5_Optimized = L_Optimized(end)
```

```
L5_Optimized = 1.7557e-09
```

There are few things to consider when setting up an optimization:

- Choosing a different objective function would change the result.
- You can use advanced direct search optimization functions such as `patternsearch` and `simulannealband` in your optimization, but you must have the Global Optimization Toolbox™ installed to access them.

References

1. Cuthbert, Thomas R. *Broadband Direct-Coupled and Matching RF Networks*. TRCPEP, 1999.
2. Ludwig, Reinhold, and Pavel Bretchko. *RF Circuit Design: Theory and Applications*. Prentice-Hall, 2000.
3. Pozar, David. *Microwave Engineering*. 2nd ed., John Wiley and Sons, 1999.

See Also

"Design Broadband Matching Networks for Amplifier" on page 6-149.

See Also

More About

- "Design Broadband Matching Networks for Amplifier" on page 6-149
- "Impedance Matching of Small Monopole Antenna" on page 6-167

Design Broadband Matching Networks for Amplifier

This example shows how to design broadband matching networks for a low noise amplifier (LNA) with ideal and real-world lumped LC elements. The real-world lumped LC elements are obtained from the Modelithics SELECT+ Library™. The LNA is designed to the target gain and noise figure specifications over a specified bandwidth. A direct-search based approach is used to arrive at the optimum element values in the input and output matching networks.

In an RF receiver front end, the LNA is commonly found immediately after the antenna or after the first bandpass filter that follows the antenna. Its position in the receiver chain ensures that it deals with weak signals that have significant noise content. As a result the LNA has to not only provide amplification to such signals but also minimize its own noise footprint on the amplified signal.

Figure 1: Impedance matching of an amplifier

Set Design Parameters

The design specifications are as follows.

- Amplifier is an LNA amplifier
- Center Frequency = 250 MHz
- Bandwidth = 100 MHz
- Transducer Gain greater than or equal to 10 dB
- Noise Figure less than or equal to 2.0 dB
- Operating between 50-Ohm terminations

Specify Design Parameters

You are building the matching network for an LNA with a bandpass response, so specify the bandwidth of match, center frequency, gain, and noise figure targets.

```
BW = 100e6;           % Bandwidth of matching network (Hz)
fc = 250e6;           % Center frequency (Hz)
Gt_target = 10;       % Transducer gain target (dB)
NFtarget = 2;         % Max noise figure target (dB)
```

Specify the source impedance, reference impedance, and the load impedance.

```
Zs = 50;              % Source impedance (Ohm)
Z0 = 50;              % Reference impedance (Ohm)
Zl = 50;              % Load impedance (Ohm)
```

Create Amplifier Object and Perform Analysis

Create an `amplifier` object from `lnadata.s2p`.

```
Mismatched_Amp = amplifier(FileName='lnadata.s2p');
```

Define the number of frequency points to use for analysis and set up the frequency vector.

```
Npts = 32;                          % No. of analysis frequency points
fLower = fc - (BW/2);               % Lower band edge
fUpper = fc + (BW/2);               % Upper band edge
freq = linspace(fLower,fUpper,Npts); % Frequency array for analysis
```

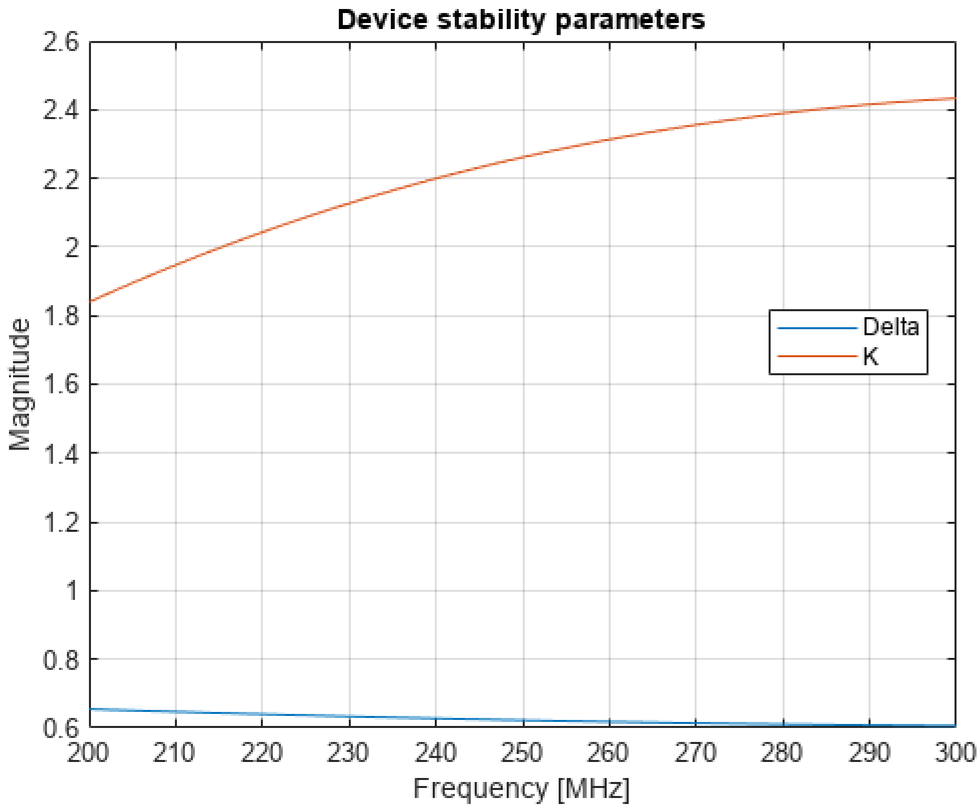
Perform frequency domain analysis on the mismatched amplifier.

```
sparam = sparameters(Mismatched_Amp,freq);
bMismatched = rfbudget(Mismatched_Amp,freq',-30,BW);
[K,~,~,Delta] = stabilityk(sparam);
Ga = powergain(sparam,'Ga');
```

Examine Stability, Power Gain, and Noise Figure

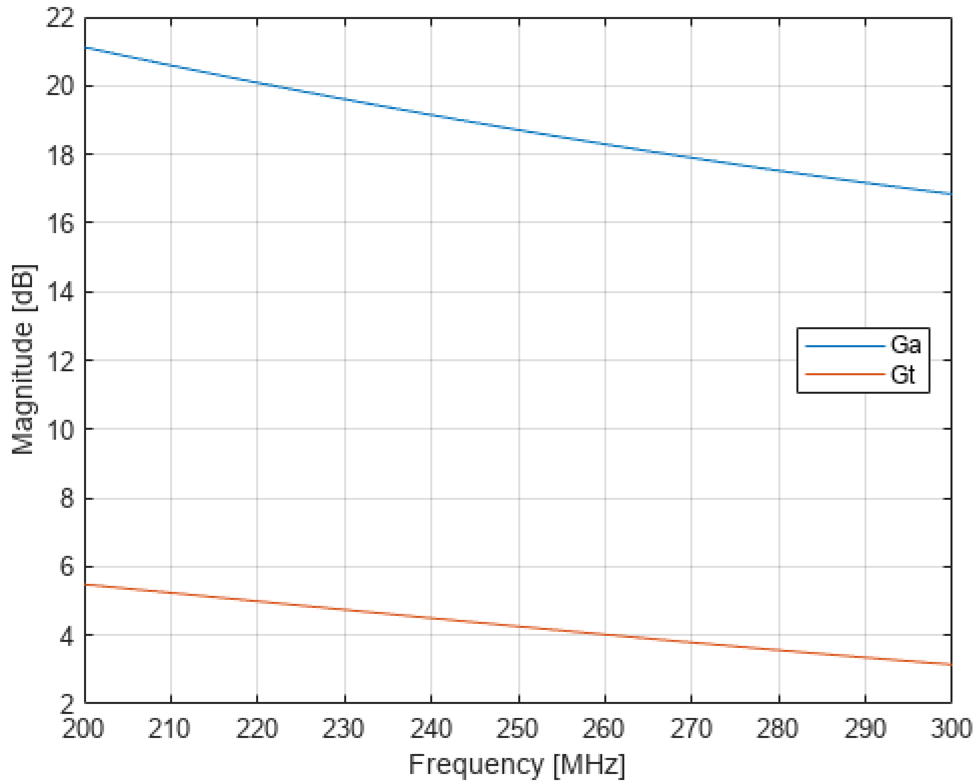
Plot the LNA stability parameters and examine stability, power gain and noise figure.

```
figure
plot(freq*1e-6,abs(Delta))
hold on
plot(freq*1e-6,K)
legend('Delta','K','Location','best')
title('Device stability parameters')
xlabel('Frequency [MHz]')
ylabel('Magnitude')
grid on
hold off
```



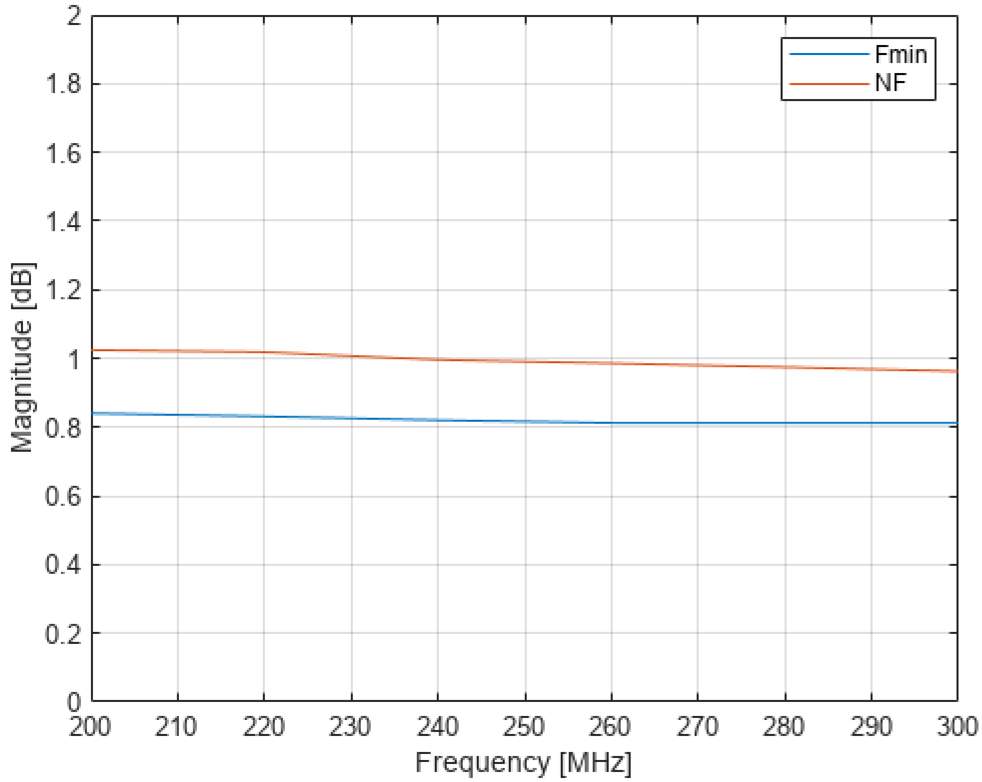
As the plot shows, $K > 1$ and $\Delta < 1$ for all frequencies in the bandwidth of interest. This means that the device is unconditionally stable. It is also important to view the power gain and noise figure behavior across the same bandwidth. Together with the stability information this data allows you to determine if the gain and noise figure targets can be met.

```
figure
plot(freq*1e-6,10*log10(Ga))
hold on
plot(freq*1e-6,bMismatched.TransducerGain)
legend('Ga','Gt','Location','east')
xlabel('Frequency [MHz]')
ylabel('Magnitude [dB]')
hold off
grid on
```



This plot, shows the power gain across the 100-MHz bandwidth. It indicates that the transducer gain varies linearly between 5.5 dB to about 3.1 dB and achieves only 4.3 dB at band center. It also suggests there is sufficient headroom between the transducer gain `Gt` and the available gain Ga to achieve our target `Gt` of 10 dB.

```
figure
plot(Mismatched_Amp.NoiseData.Frequencies*1e-6,Mismatched_Amp.NoiseData.Fmin)
hold on
plot(freq*1e-6,bMismatched.NF)
hold off
axis([200 300 0 2])
legend('Fmin','NF','Location','NorthEast')
xlabel('Frequency [MHz]')
ylabel('Magnitude [dB]')
grid on
```



This plot shows the variation of the noise figure with frequency. The unmatched amplifier clearly meets the target noise figure requirement. However this would change once the input and output matching networks are included. Most likely, the noise figure of the LNA would exceed the requirement.

Design Input and Output Matching Networks

The region of operation is between 200 — 300 MHz. Therefore, choose a bandpass topology for the matching networks which is shown here.

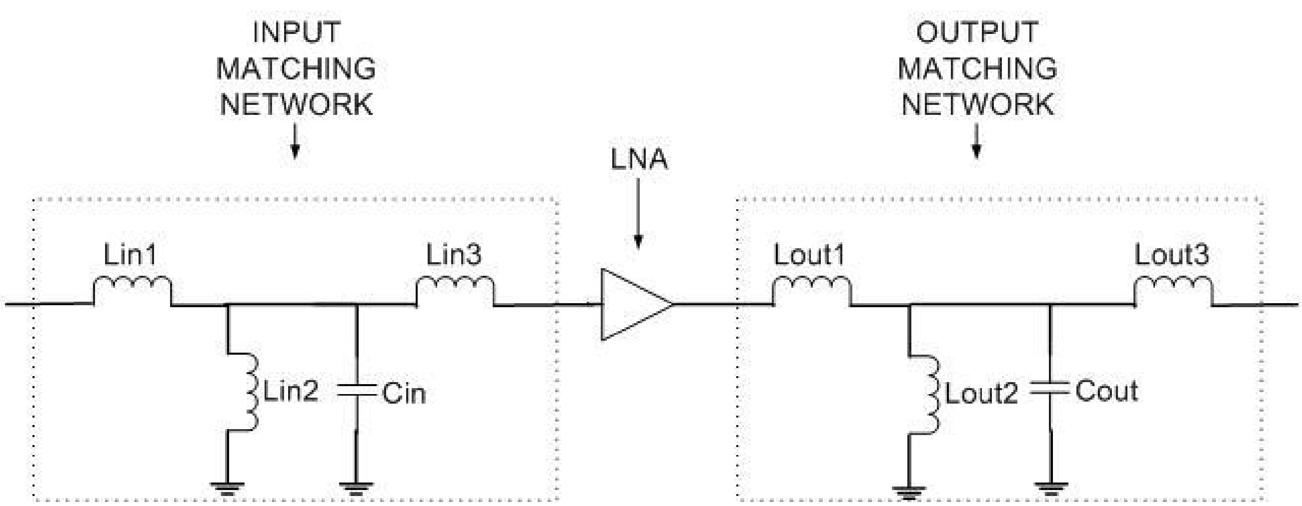


Figure 2: Matching network topology

The topology chosen, as seen in Figure 2, is a direct-coupled prototype bandpass network of parallel resonator type with top coupling [2], that is initially tuned to the geometric mean frequency with respect to the bandwidth of operation.

```
N = 3;                  % Order of input/output matching network
wU = 2*pi*fUpper;       % Upper band edge
wL = 2*pi*fLower;       % Lower band edge
w0 = sqrt(wL*wU);       % Geometric mean
```

For the initial design all the inductors are assigned the same value on the basis of the first series inductor. As mentioned in [3], choose the prototype value to be unity and use standard impedance and frequency transformations to obtain denormalized values [1]. The value for the capacitor in the parallel trap is set using this inductor value to make it resonate at the geometric mean frequency. Note that there are many ways of designing the initial matching network. This example shows one possible approach.

```
LvaluesIn = (Zs/(wU-wL))*ones(N,1);         % Series and shunt L's [H]
CvaluesIn = 1 / ( (w0^2)*LvaluesIn(2));     % Shunt C [F]
LvaluesOut = LvaluesIn;
CvaluesOut = CvaluesIn;
```

Build Matching Networks

Build each branch of the matching network and then define the circuit as a two-port network. This example uses identical input and output matching networks.

```
InputMatchingNW = circuit;
add(InputMatchingNW,[1 2],inductor(LvaluesIn(1)));
add(InputMatchingNW,[2 0],inductor(LvaluesIn(2)));
add(InputMatchingNW,[2 0],capacitor(CvaluesIn));
add(InputMatchingNW,[2 3],inductor(LvaluesIn(3)));
setports(InputMatchingNW,[1 0],[3 0]);

OutputMatchingNW = circuit;
add(OutputMatchingNW,[1 2],inductor(LvaluesOut(1)));
add(OutputMatchingNW,[2 0],inductor(LvaluesOut(2)));
add(OutputMatchingNW,[2 0],capacitor(CvaluesOut));
add(OutputMatchingNW,[2 3],inductor(LvaluesOut(3)));
setports(OutputMatchingNW,[1 0],[3 0]);
```

Optimize Input & Output Matching Network

There are several points to consider prior to the optimization.

- Objective function: The objective function can be built in different ways depending on the problem at hand. For this example, the objective function is shown in the file below.
- Choice of cost function: The cost function is the function you would like to minimize (maximize) to achieve near optimal performance. There could be several ways to choose the cost function. For this example you have two requirements to satisfy simultaneously, i.e. gain and noise figure. To create the cost function you first, find the difference, between the most current optimized network and the target value for each requirement at each frequency. The cost function is the L2-norm of the vector of gain and noise figure error values.
- Optimization variables: In this case it is a vector of values, for the specific elements to optimize in the matching network.

- Optimization method: A direct search based technique, the MATLAB® function `fminsearch`, is used in this example to perform the optimization.
- Number of iterations/function evaluations: Set the maximum no. of iterations and function evaluations to perform, so as to tradeoff between speed and quality of match.
- Tolerance value: Specify the variation in objective function value at which the optimization process should terminate.

The objective function used during the optimization process by `fminsearch` is shown here.

```
type('broadband_match_amplifier_objective_function.m')
```

```
function output = broadband_match_amplifier_objective_function(AMP,inMNW,outMNW,LC_Optim,freq,Gt_
%BROADBAND_MATCH_AMPLIFIER_OBJECTIVE_FUNCTION is the objective function.
% OUTPUT = BROADBAND_MATCH_AMPLIFIER_OBJECTIVE_FUNCTION(AMP,INMNW,OUTMNW,LC_OPTIM,FREQ,GT_TARGET
% returns the current value of the objective function stored in OUTPUT
% evaluated after updating the element values in the object, inMNW and
% outMNW. The inductor and capacitor values are stored in the variable
% LC_OPTIM.
%
% BROADBAND_MATCH_AMPLIFIER_OBJECTIVE_FUNCTION is an objective function of RF Toolbox demo:
% Designing Broadband Matching Networks (Part II: Amplifier)

%   Copyright 2023 The MathWorks, Inc.

% Ensure positive element values
if any(LC_Optim<=0)
    output = inf;
    return;
end
% Update matching network elements - The object inMNW and outMNW have
% several properties among which the cell array 'Elements' consists of all
% LC elements in the circuit.
inMNW.Elements(1).Inductance = LC_Optim(1);
inMNW.Elements(2).Inductance = LC_Optim(2);
inMNW.Elements(4).Inductance = LC_Optim(3);
inMNW.Elements(3).Capacitance = LC_Optim(4);
outMNW.Elements(1).Inductance = LC_Optim(5);
outMNW.Elements(2).Inductance = LC_Optim(6);
outMNW.Elements(4).Inductance = LC_Optim(7);
outMNW.Elements(3).Capacitance = LC_Optim(8);

% Perform analysis on tuned matching network
Npts            = length(freq);
BW              = max(freq) - min(freq);
np1             = nport(sparameters(inMNW,freq));
np2             = nport(sparameters(outMNW,freq));
b               = rfbudget([np1 clone(AMP) np2],freq',-30,BW);

% Calculate transducer power gains and noise figures of the amplifier
Gt              = b.TransducerGain(:,end);
NF_amp          = b.NF(:,end);

% Calculate target gain and noise figure error
errGt           = (Gt - Gt_target);
errNF           = (NF_amp - NF);

% Check to see if gain and noise figure target are achieved by specifying
```

```
% bounds for variation.
deltaG        = 0.40;
deltaNF       = -0.05;
errGt(abs(errGt)<=deltaG) = 0;
errNF(errNF<deltaNF) = 0;

% Cost function
err_vec       = [errGt;errNF];
output        = norm((err_vec),2);

% Animate
Gmax          = (Gt_target + deltaG).*ones(1,Npts);
Gmin          = (Gt_target - deltaG).*ones(1,Npts);
plot(freq*1e-6,Gt)
hold on
plot(freq*1e-6,NF_amp)
plot(freq.*1e-6,Gmax,'r-*')
plot(freq.*1e-6,Gmin,'r-*')
legend('G_t','NF','Gain bounds','Location','East');
xlabel('Freq [MHz]')
ylabel('Magnitude [dB]')
axis([freq(1)*1e-6 freq(end)*1e-6 0 Gt_target+2]);
hold off
drawnow;
```

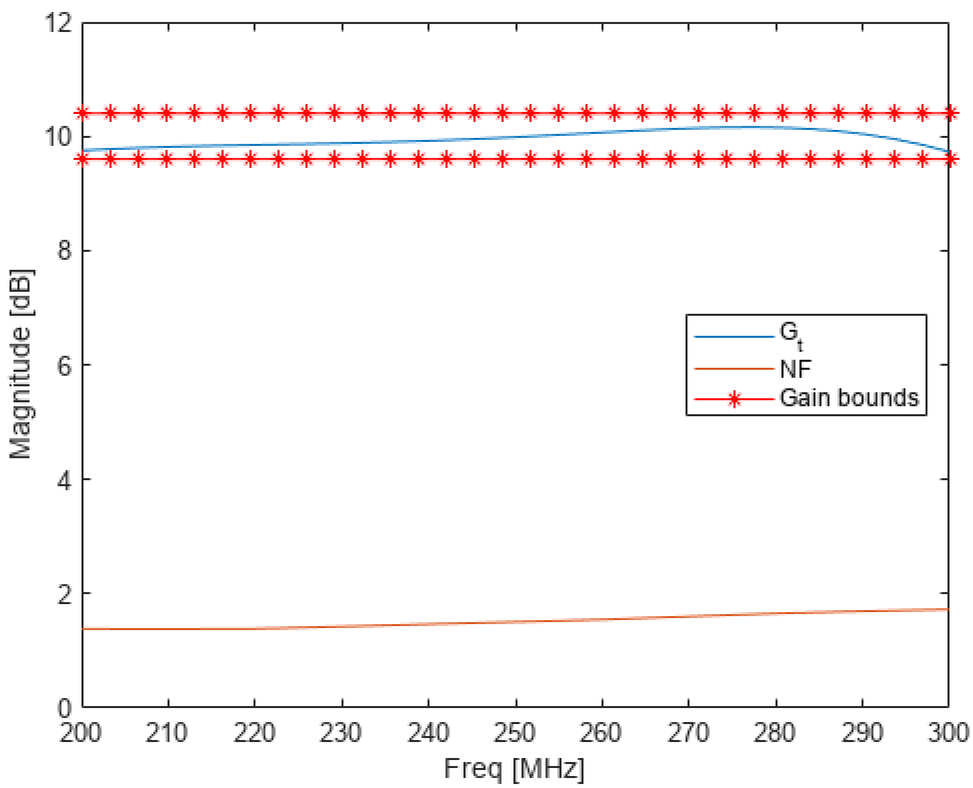
The optimization variables are all the elements (inductors and capacitors) of the input and output matching networks.

```
nIter = 125; % Max No of Iterations
options = optimset('Display','iter','TolFun',1e-2,'MaxIter',nIter);   % Set options structure
LC_Optimized = [LvaluesIn;CvaluesIn;LvaluesOut;CvaluesOut];
LC_Optimized = fminsearch(@(LC_Optimized) broadband_match_amplifier_objective_function(Mismatched
                             OutputMatchingNW,LC_Optimized,freq,Gt_target,NFtarget),LC_Optimized
```

```
 Iteration   Func-count         f(x)         Procedure
     0            1         30.4869
     1            9         28.3549         initial simplex
     2           11         25.5302         expand
     3           12         25.5302         reflect
     4           13         25.5302         reflect
     5           14         25.5302         reflect
     6           16         22.8228         expand
     7           17         22.8228         reflect
     8           19         19.0289         expand
     9           20         19.0289         reflect
    10           21         19.0289         reflect
    11           22         19.0289         reflect
    12           24         14.8785         expand
    13           25         14.8785         reflect
    14           27          10.721         expand
    15           28          10.721         reflect
    16           29          10.721         reflect
    17           31         9.84796         expand
    18           32         9.84796         reflect
    19           33         9.84796         reflect
    20           34         9.84796         reflect
    21           35         9.84796         reflect
```

```
    22          37          9.84796         contract outside
    23          39          9.84796         contract outside
    24          41          9.84796         contract inside
    25          43          9.64666         reflect
    26          45          9.64666         contract inside
    27          46          9.64666         reflect
    28          48          9.64666         contract inside
    29          49          9.64666         reflect
    30          51          9.64666         contract inside
    31          53           7.9372         expand
    32          55           7.9372         contract outside
    33          56           7.9372         reflect
    34          57           7.9372         reflect
    35          58           7.9372         reflect
    36          59           7.9372         reflect
    37          60           7.9372         reflect
    38          62          5.98211         expand
    39          63          5.98211         reflect
    40          64          5.98211         reflect
    41          65          5.98211         reflect
    42          66          5.98211         reflect
    43          68          4.31973         expand
    44          70          4.31973         contract inside
    45          71          4.31973         reflect
    46          72          4.31973         reflect
    47          73          4.31973         reflect
    48          74          4.31973         reflect
    49          75          4.31973         reflect
    50          77          2.83135         expand
    51          79          1.17624         expand
    52          80          1.17624         reflect
    53          81          1.17624         reflect
    54          82          1.17624         reflect
    55          84         0.691645         reflect
    56          85         0.691645         reflect
    57          86         0.691645         reflect
    58          88         0.691645         contract inside
    59          90         0.691645         contract outside
    60          91         0.691645         reflect
    61          93         0.691645         contract inside
    62          95         0.691645         contract inside
    63          96         0.691645         reflect
    64          97         0.691645         reflect
    65          98         0.691645         reflect
    66         100         0.691645         contract inside
    67         102         0.691645         contract outside
    68         103         0.691645         reflect
    69         105         0.691645         contract inside
    70         107         0.497434         reflect
    71         109         0.497434         contract inside
    72         111         0.497434         contract inside
    73         112         0.497434         reflect
    74         114         0.497434         contract inside
    75         116         0.497434         contract inside
    76         118         0.444957         reflect
    77         120         0.402851         expand
    78         122                0         reflect
    79         123                0         reflect
```

```
    80          125               0         contract inside
    81          127               0         contract inside
    82          128               0         reflect
    83          129               0         reflect
    84          130               0         reflect
    85          131               0         reflect
    86          132               0         reflect
    87          133               0         reflect
    88          134               0         reflect
    89          135               0         reflect
    90          137               0         contract inside
```




```
    91          139               0         contract outside

Optimization terminated:
 the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-04
 and F(X) satisfies the convergence criteria using OPTIONS.TolFun of 1.000000e-02
```

Update Matching Network and Analyze LNA

Update the input and output matching networks with optimized element values, `LC_Optimized`.

```
InputMatchingNW.Elements(1).Inductance = LC_Optimized(1);
InputMatchingNW.Elements(2).Inductance = LC_Optimized(2);
InputMatchingNW.Elements(4).Inductance = LC_Optimized(3);
InputMatchingNW.Elements(3).Capacitance = LC_Optimized(4);
OutputMatchingNW.Elements(1).Inductance = LC_Optimized(5);
OutputMatchingNW.Elements(2).Inductance = LC_Optimized(6);
```

```
OutputMatchingNW.Elements(4).Inductance = LC_Optimized(7);
OutputMatchingNW.Elements(3).Capacitance = LC_Optimized(8);
```

Analyze the S-parameters of input and output matching networks and then convert the S-parameter data into an `nport` object.

```
s1 = sparameters(InputMatchingNW,freq);
np1 = nport(s1);
s2 = sparameters(OutputMatchingNW,freq);
np2 = nport(s2);
```

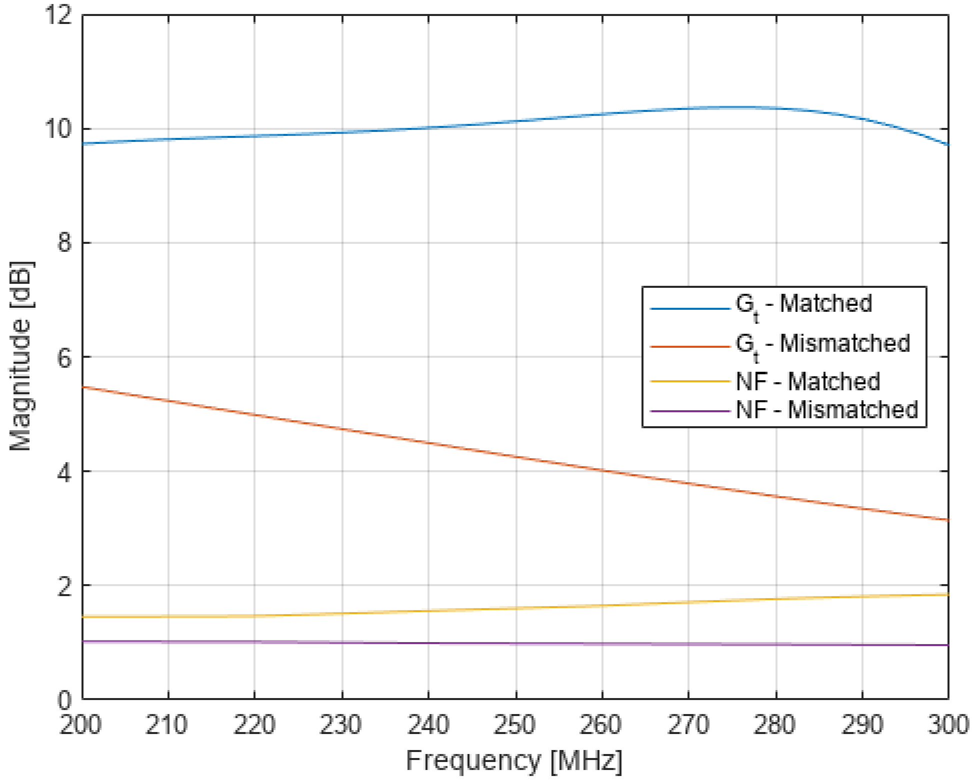
Use the `rfbudget` object to calculate the transducer gain and the noise figure across the bandwidth. Compute RF budget of the LNA network. The LNA network comprises of matching network and low noise amplifier.

```
bMatched = rfbudget([np1 clone(Mismatched_Amp) np2],freq',-30,BW);
```

Verify Design

Plot the transducer gain and the noise figure and then compare the optimization results with the mismatched network.

```
plot(freq*1e-6,bMatched.TransducerGain(:,end))
hold all
plot(freq*1e-6,bMismatched.TransducerGain(:,end))
plot(freq*1e-6,bMatched.NF(:,end))
plot(freq*1e-6,bMismatched.NF(:,end))
legend('G_t - Matched','G_t - Mismatched','NF - Matched',...
       'NF - Mismatched','Location','East')
xlabel('Frequency [MHz]')
ylabel('Magnitude [dB]')
axis([freq(1)*1e-6 freq(end)*1e-6 0 12])
grid on
hold off
```



The plot shows that the target requirement for both gain and noise figure have been met. To understand the effect of optimizing with respect to only the transducer gain, use the first choice for the cost function. The cost function involves only the gain term within the objective function.

Display Optimized Element Values

Display the optimized inductor and capacitor values for the input matching network.

```
Lin_Optimized = LC_Optimized(1:3)
```

```
Lin_Optimized = 3×1
10^-7 ×

    0.5722
    0.9272
    0.3546
```

```
Cin_Optimized = LC_Optimized(4)
```

```
Cin_Optimized = 6.8526e-12
```

Display the optimized inductor and capacitor values for the output matching network

```
Lout_Optimized = LC_Optimized(5:7)
```

```
Lout_Optimized = 3×1
10^-6 ×
```

```
    0.0517
    0.1275
    0.0581
```

```
Cout_Optimized = LC_Optimized(8)
```

```
Cout_Optimized = 5.4408e-12
```

Matching Network with Real-world Lumped Component Models

The real-world lumped components are selected from the Modelithics Select+ Library [5]. You must have the Modelithics Select+ Library license to run the following codes. Note that the Modelithics® LC elements are scaled with respect to the substrate characteristics. This example shows one possible approach using a 59 mil FR4 substrate.

Create RF Component from Modelithics SELECT+ Library

Set up the Modelithics Select+ Library by specifying the full path to the library.

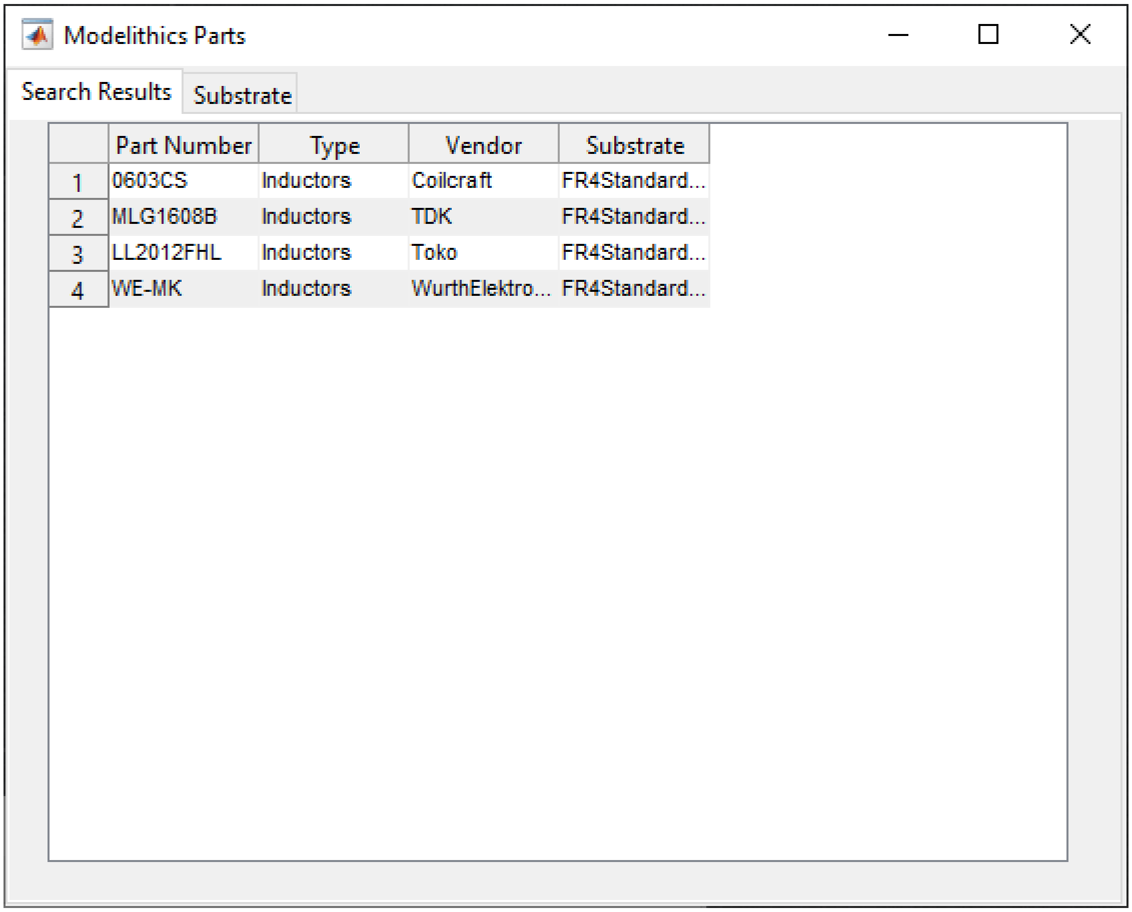
```
mdlxSetup('C:\mdlx_library\SELECT')
```

Create the Modelithics® library object.

```
mdlx = mdlxLibrary;
```

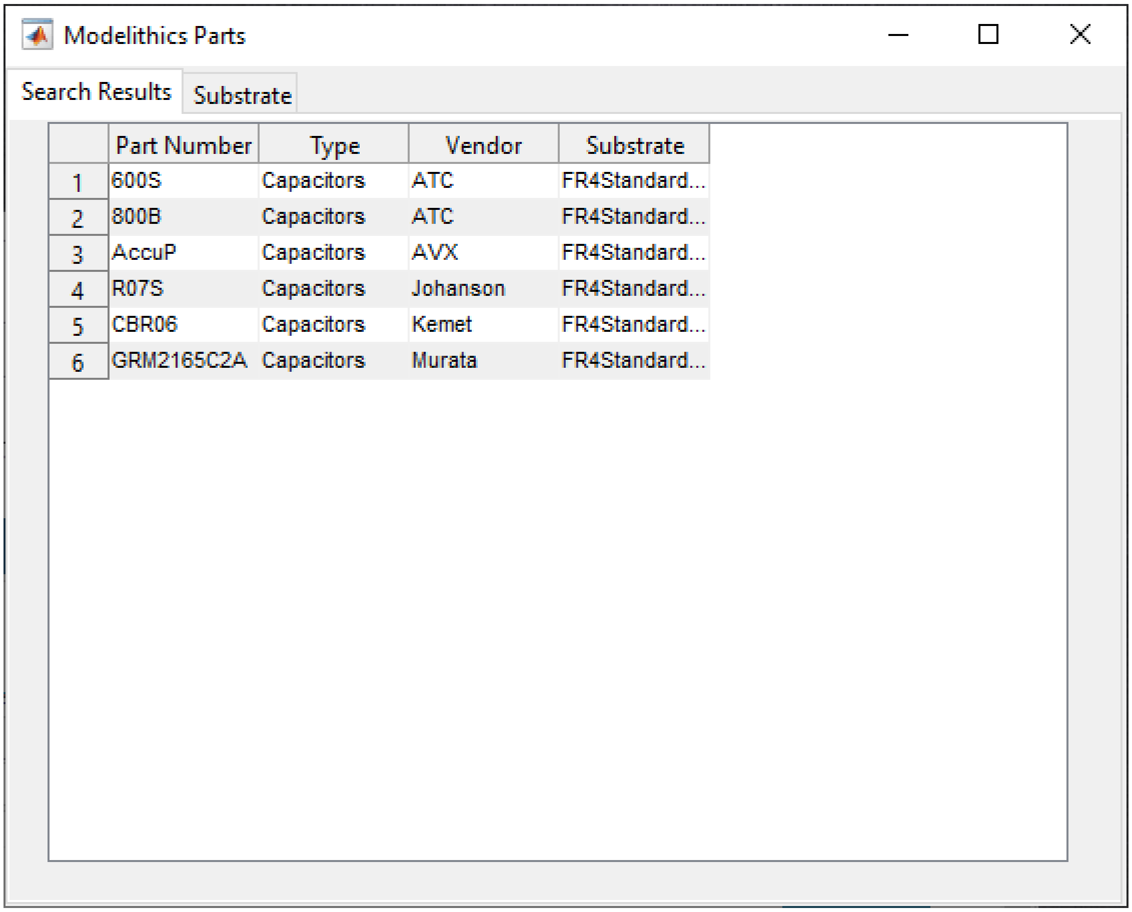
Search the library for inductors with the optimized value for the input matching network.

```
search(mdlx,'FR4Standard59mil',Type='Inductors',Value=Lin_Optimized(1))
```



Search the library for capacitors with the optimized value for the input matching network.

```
search(mdlx,'FR4Standard59mil',Type='Capacitors',Value=Cin_Optimized)
```



	Part Number	Type	Vendor	Substrate
1	600S	Capacitors	ATC	FR4Standard...
2	800B	Capacitors	ATC	FR4Standard...
3	AccuP	Capacitors	AVX	FR4Standard...
4	R07S	Capacitors	Johanson	FR4Standard...
5	CBR06	Capacitors	Kemet	FR4Standard...
6	GRM2165C2A	Capacitors	Murata	FR4Standard...

Create an array of Modelithics® LC elements for the input and output matching networks.

```
Lin1_mdlx = search(mdlx,'FR4Standard59mil',Type='Inductors',Value=Lin_Optimized(1));
Lin2_mdlx = search(mdlx,'FR4Standard59mil',Type='Inductors',Value=Lin_Optimized(2));
Lin3_mdlx = search(mdlx,'FR4Standard59mil',Type='Inductors',Value=Lin_Optimized(3));
Cin_mdlx = search(mdlx,'FR4Standard59mil',Type='Capacitors',Value=Cin_Optimized);

Lout1_mdlx = search(mdlx,'FR4Standard59mil',Type='Inductors',Value=Lout_Optimized(1));
Lout2_mdlx = search(mdlx,'FR4Standard59mil',Type='Inductors',Value=Lout_Optimized(2));
Lout3_mdlx = search(mdlx,'FR4Standard59mil',Type='Inductors',Value=Lout_Optimized(3));
Cout_mdlx = search(mdlx,'FR4Standard59mil',Type='Capacitors',Value=Cout_Optimized);
```

Create and Anlyze Matching Networks

Create the input and output matching networks with Modelithics® capacitors and inductors. Note that most of the Modelithics® lumped components have two ports. Use the `setports` function to define the circuit as a 2-port network.

```
InputMatchingNWMdlx = circuit;
add(InputMatchingNWMdlx,[1 2 0 0],Lin1_mdlx(1));
add(InputMatchingNWMdlx,[2 0 0 0],Lin2_mdlx(1));
add(InputMatchingNWMdlx,[2 0 0 0],Cin_mdlx(1));
add(InputMatchingNWMdlx,[2 3 0 0],Lin3_mdlx(1));
setports(InputMatchingNWMdlx,[1 0],[3 0]);
```

```
OutputMatchingNWMdlx = circuit;
add(OutputMatchingNWMdlx,[1 2 0 0],Lout1_mdlx(1));
add(OutputMatchingNWMdlx,[2 0 0 0],Lout2_mdlx(1));
add(OutputMatchingNWMdlx,[2 0 0 0],Cout_mdlx(1));
add(OutputMatchingNWMdlx,[2 3 0 0],Lout3_mdlx(1));
setports(OutputMatchingNWMdlx,[1 0],[3 0]);
```

Analyze the S-parameters of input and output matching networks and then convert the S-parameter data into an `nport` object.

```
sm1 = sparameters(InputMatchingNWMdlx,freq);
npm1 = nport(sm1);

sm2 = sparameters(OutputMatchingNWMdlx,freq);
npm2 = nport(sm2);
```

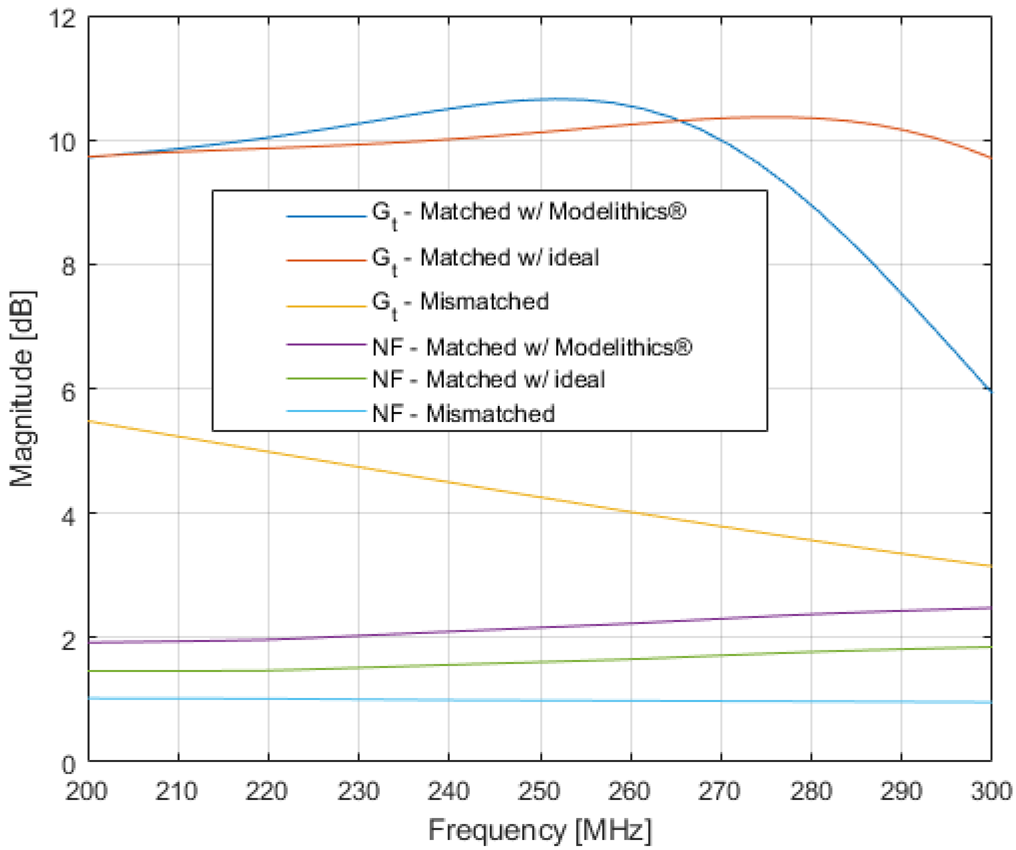
Use the `rfbudget` object to calculate the transducer gain and the noise figure across the bandwidth. Compute RF budget of the LNA network. The LNA network comprises of matching network and low noise amplifier.

```
bMatchedMdlx = rfbudget([npm1 clone(Mismatched_Amp) npm2],freq',-30,BW);
```

Verify Design

Plot the transducer gain and the noise figure of the real-world matching network and then compare the optimization results with the mismatched network.

```
figure
plot(freq*1e-6,bMatchedMdlx.TransducerGain(:,end))
hold all
plot(freq*1e-6,bMatched.TransducerGain(:,end))
plot(freq*1e-6,bMismatched.TransducerGain(:,end))
plot(freq*1e-6,bMatchedMdlx.NF(:,end))
plot(freq*1e-6,bMatched.NF(:,end))
plot(freq*1e-6,bMismatched.NF(:,end))
axis([freq(1)*1e-6 freq(end)*1e-6 0 12])
legend('G_t - Matched w/ Modelithics®','G_t - Matched w/ ideal',...
    'G_t - Mismatched','NF - Matched w/ Modelithics®','NF - Matched w/ ideal',...
       'NF - Mismatched','Position', [0.22643,0.47762,0.49643,0.25119])
xlabel('Frequency [MHz]')
ylabel('Magnitude [dB]')
grid on
hold off
```



The most noticeable aspect of the matching performance when using the Modelithics® library is the large deviation in transducer gain especially beyond 275 MHz. The noise figure performance has also worsened by approximately 0.25 dB. This can be explained by the fact that the inductor and capacitor models now represent real-world parasitic effects such as those due to the presence of a substrate, pads and orientation. This analysis highlights the importance of being able to predict such behaviors prior to realization of the design in hardware.

References

[1] Ludwig, Reinhold, and Gene Bogdanov. *RF Circuit Design: Theory and Applications*. Upper Saddle River, NJ: Prentice-Hall, 2009.

[2] Cuthbert, Thomas R. *Broadband Direct-Coupled and Matching RF Networks*. Greenwood, Ark.: T.R. Cuthbert, 1999.

[3] Cuthbert, T.R. “A Real Frequency Technique Optimizing Broadband Equalizer Elements.” In *2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No.00CH36353)*, 5:401–4. Geneva, Switzerland: Presses Polytech. Univ. Romandes, 2000. https://doi.org/10.1109/ISCAS.2000.857453.

[4] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.

[5] https://www.modelithics.com/model/freemodels

See Also

More About

- “Design Broadband Matching Networks for Antennas” on page 6-140
- “Impedance Matching of Small Monopole Antenna” on page 6-167

Impedance Matching of Small Monopole Antenna

This example shows how to design a double tuning L-section matching network between a resistive source and capacitive load in the form of a small monopole by using the `matchingnetwork` object. The L-section consists of two inductors. The network achieves conjugate match and guarantees maximum power transfer at a single frequency. This example requires the following product:

- Antenna Toolbox™

Create Monopole

Create a quarter-wavelength monopole antenna via the Antenna Toolbox with the resonant frequency around 1 GHz. For the purpose of this example, we choose a square ground plane of side 0.75λ.

```
fres = 1e9;
speedOfLight = physconst('lightspeed');
lambda = speedOfLight/fres;
L = 0.25*lambda;
dp = monopole('Height',L,'Width',L/50,...
    'GroundPlaneLength',0.75*lambda,...
    'GroundPlaneWidth',0.75*lambda);
```

Calculate Monopole Impedance

Specify the source (generator) impedance, the reference (transmission line) impedance and the load (antenna) impedance. In this example, the load `Zl0` will be the non-resonant (small) monopole at the frequency of 500 MHz, which is the half of the resonant frequency. The source has the equivalent impedance of 50 ohms.

```
f0 = fres/2;
Zs = 50;
Z0 = 50;
Zl0 = impedance(dp,f0);
Rl0 = real(Zl0);
Xl0 = imag(Zl0);
```

Define the number of frequency points for the analysis and set up a frequency band about 500 MHz.

```
Npts = 30;
fspan = 0.1;
fmin = f0*(1 - (fspan/2));
fmax = f0*(1 + (fspan/2));
freq = unique([f0 linspace(fmin,fmax,Npts)]);
```

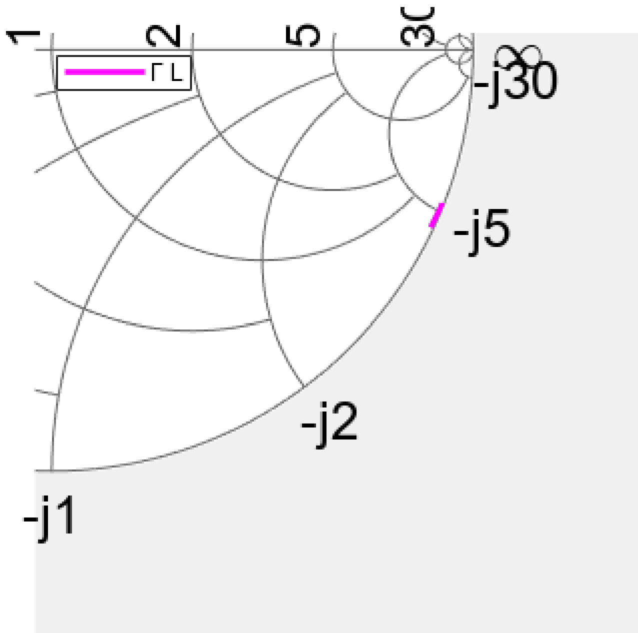
Understand Load Behavior using Reflection Coefficient and Power Gain

Calculate the load reflection coefficient and the power gain between the source and the antenna.

```
S = sparameters(dp, freq);
GammaL = rfparam(S, 1,1);
Gt = 10*log10(1 - abs(GammaL).^2);
```

Plotting the input reflection coefficient on a Smith chart shows the capacitive behavior of this antenna around the operating frequency of 500 MHz. The center of the Smith chart represents the matched condition to the reference impedance. The location of the reflection coefficient trace around $-j5.0\ \Omega$ confirms that there is a severe impedance mismatch.

```
fig1 = figure;
hsm = smithplot(fig1,freq,GammaL,'LineWidth',2.0,'Color','m',...
    'View','bottom-right','LegendLabels',{'#Gamma L'});
```



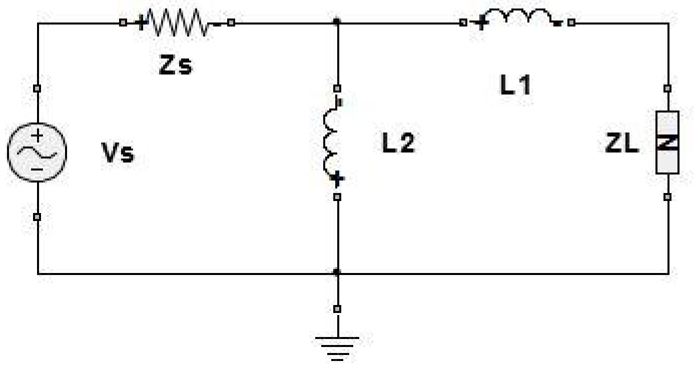
Plot the power delivered to the load.

```
fig2 = figure;
plot(freq*1e-6,Gt,'m','LineWidth',2);
grid on
xlabel('Frequency [MHz]')
ylabel('Magnitude (dB)')
title('Power delivered to load')
```

As the power gain plot shows, there is approximately 20 dB power loss around the operating frequency (500 MHz).

Design Matching Network

The matching network must ensure maximum power transfer at 500 MHz. The L-section double tuning network achieves this goal [1]. The network topology, shown in the figure consists of an inductor in series with the antenna, that cancels the large capacitance at 500 MHz, and a shunt inductor that further boosts the output resistance to match the source impedance of 50 Ω.



Use the `matchingnetwork` object to create various matching network circuits based on the source impedance, load impedance, and center frequency.

```
matchnw = matchingnetwork('CenterFrequency',f0,'LoadImpedance',Zl0,'Bandwidth',50e6);
matchnw.clearEvaluationParameter(1); % clear default constraint
```

The values of each element for the circuits generated are shown below.

```
[circuit_list, performance] = circuitDescriptions(matchnw)
```

```
circuit_list=4×5 table
                 circuitName    component1Type    component1Value    component2Type    component2
                 ___________    ______________    _______________    ______________    __________

    Circuit 1     "auto_1"       "Series L"        2.9822e-07         "Shunt L"         1.0466e
    Circuit 2     "auto_2"       "Series C"        3.3975e-13         "Shunt L"         6.1569e
    Circuit 3     "auto_3"       "Shunt C"         2.3684e-11         "Series L"        8.1505e
    Circuit 4     "auto_4"       "Shunt L"          4.278e-09         "Series L"        7.3525e

performance=4×4 table
                 circuitName    evaluationPassed    testsFailed     performanceScore
                 ___________    ________________    ____________    ________________

    Circuit 1     "auto_1"        {["Yes"]}         {0x0 double}         {[0]}
    Circuit 2     "auto_2"        {["Yes"]}         {0x0 double}         {[0]}
    Circuit 3     "auto_3"        {["Yes"]}         {0x0 double}         {[0]}
    Circuit 4     "auto_4"        {["Yes"]}         {0x0 double}         {[0]}
```

Create Matching Network and Calculate S-parameters

The matching network circuit is created via the RF Toolbox™ and it consists of the two inductors whose values have been calculated above. The S-parameters of this network are calculated over the frequency band centered at the operating frequency.

Calculate the S-parameters based on desired matching network (this example uses circuit #2).

```
ckt_no = 4;
Smatchnw = sparameters(matchnw, freq, Z0, ckt_no);
```

The circuit element representation of the matching network is shown below.

```
disp(matchnw.Circuit(ckt_no))
```

```
  circuit: Circuit element

    ElementNames: {'L'  'L_1'}
        Elements: [1x2 inductor]
           Nodes: [1 2 3]
            Name: 'auto_4'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Reflection Coefficient and Power Gain with Matching Network

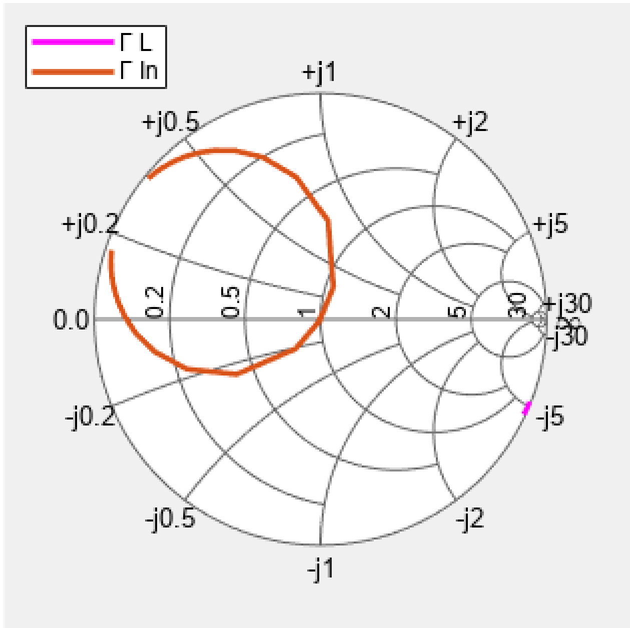
Calculate the input reflection coefficient/power gain for the antenna load with the matching network.

```
Zl =  impedance(dp,freq);
GammaIn = gammain(Smatchnw,Zl);
Gtmatch = powergain(Smatchnw,Zs,Zl,'Gt');
Gtmatch = 10*log10(Gtmatch);
```

Compare Results

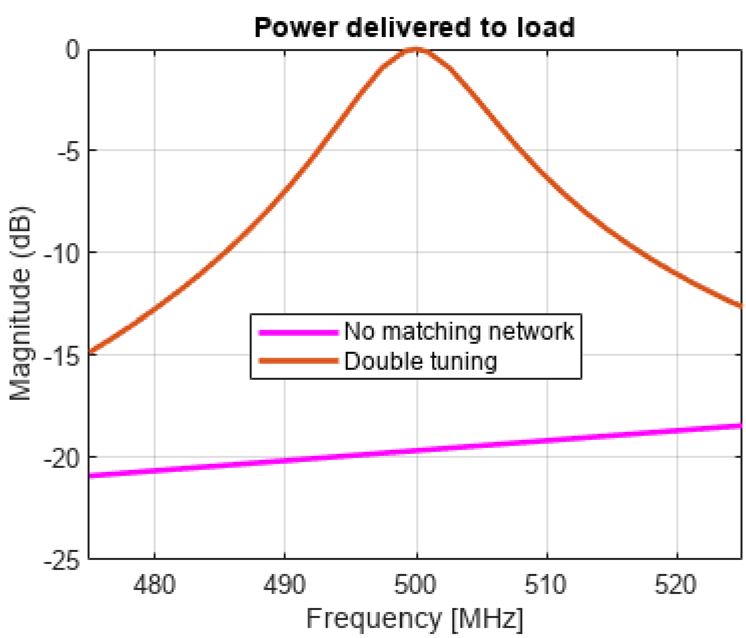
Plot the input reflection coefficient and power delivered to the antenna, with and without the matching network. The Smith® chart plot shows the reflection coefficient trace going through its center thus confirming the match. At the operation frequency of 500 MHz, the generator transfers maximum power to the antenna. The match degrades on either side of the operating frequency.

```
add(hsm,freq,GammaIn);
hsm.LegendLabels(2) = {'#Gamma In'};
hsm.View = 'full';
```



Plot power delivered to the load.

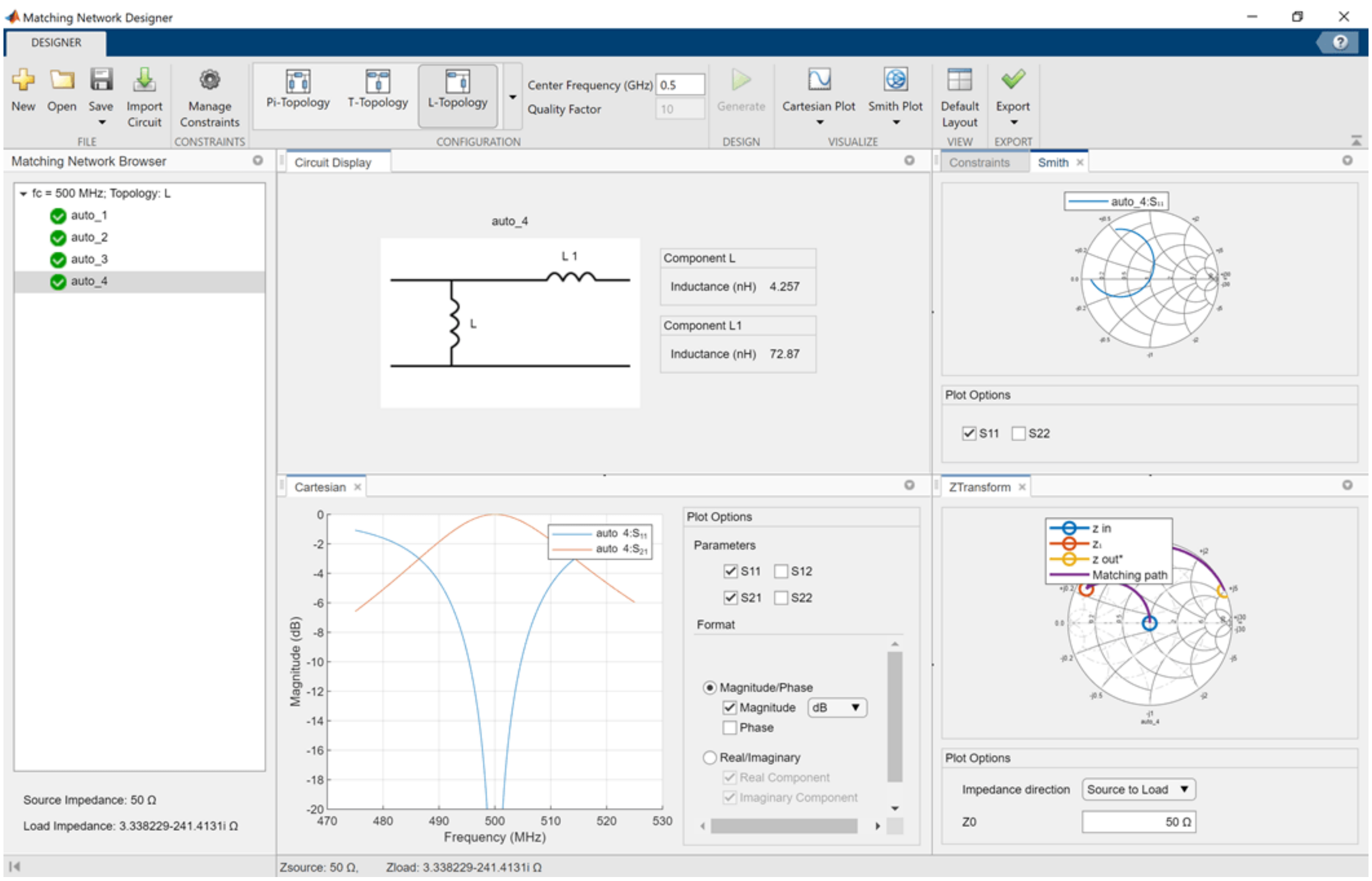
```
figure(fig2)
hold on
plot(freq*1e-6,Gtmatch,'LineWidth',2);
axis([min(freq)*1e-6,max(freq)*1e-6,-25,0])
legend('No matching network','Double tuning','Location','Best');
```



Matching Network Designer

The **Matching Network Designer** app allows to design matching networks or view an existing `matchingnetwork` object. Type this command at the command line to open the **Matching Network Designer** app. Use the `matchnw` object and select `auto_4` node to view the corresponding circuit.

```
matchingNetworkDesigner(matchnw)
```



References

[1] M. M. Weiner, Monopole Antennas, Marcel Dekker, Inc.,CRC Press, Rev. Exp edition, New York, pp.110-118, 2003.

See Also

More About

- "Design Broadband Matching Networks for Antennas" on page 6-140
- "Design Broadband Matching Networks for Amplifier" on page 6-149

Operations with RF Circuit Objects

This example shows how to create and use RF Toolbox™ circuit objects. In this example, you create three circuit (`rfckt`) objects: two transmission lines and an amplifier.

You visualize the amplifier data using RF Toolbox™ functions and retrieve the frequency data that was read from a file into the amplifier `rfckt` object. Then you analyze the amplifier over a different frequency range and visualize the results. Next, you cascade the three circuits, analyze the cascaded network and visualize its S-parameters over the original frequency range of the amplifier. Finally, you plot the `S11`, `S22`, and `S21` parameters, and noise figure of the cascaded network.

Create `rfckt` Objects

Create three circuit objects: two transmission lines, and an amplifier using data from `default.amp` data file.

```
FirstCkt = rfckt.txline;
SecondCkt = rfckt.amplifier('IntpType','cubic');
read(SecondCkt,'default.amp');
ThirdCkt = rfckt.txline('LineLength',0.025,'PV',2.0e8);
```

View Properties of `rfckt` Objects

You can use the `get` function to view an object's properties. For example,

```
PropertiesOfFirstCkt = get(FirstCkt)
```

```
PropertiesOfFirstCkt = struct with fields:
        LineLength: 0.0100
          StubMode: 'NotAStub'
       Termination: 'NotApplicable'
              Freq: 1.0000e+09
                Z0: 50.0000 + 0.0000i
                PV: 299792458
              Loss: 0
          IntpType: 'Linear'
             nPort: 2
    AnalyzedResult: []
              Name: 'Transmission Line'
```

```
PropertiesOfSecondCkt = get(SecondCkt)
```

```
PropertiesOfSecondCkt = struct with fields:
         NoiseData: [1x1 rfdata.noise]
     NonlinearData: [1x1 rfdata.power]
          IntpType: 'Cubic'
       NetworkData: [1x1 rfdata.network]
             nPort: 2
    AnalyzedResult: [1x1 rfdata.data]
              Name: 'Amplifier'
```

```
PropertiesOfThirdCkt = get(ThirdCkt)
```

```
PropertiesOfThirdCkt = struct with fields:
        LineLength: 0.0250
```

```
          StubMode: 'NotAStub'
       Termination: 'NotApplicable'
              Freq: 1.0000e+09
                Z0: 50.0000 + 0.0000i
                PV: 200000000
              Loss: 0
          IntpType: 'Linear'
             nPort: 2
    AnalyzedResult: []
              Name: 'Transmission Line'
```

List Methods of `rfckt` Objects

You can use the `methods` function to list an object's methods. For example,

```
MethodsOfThirdCkt = methods(ThirdCkt)
```

```
MethodsOfThirdCkt = 82x1 cell
    {'addlistener'         }
    {'analyze'             }
    {'calcgroupdelay'      }
    {'calckl'              }
    {'calcpout'            }
    {'calculate'           }
    {'calczin'             }
    {'checkbool'           }
    {'checkchar'           }
    {'checkenum'           }
    {'checkenumexact'      }
    {'checkfrequency'      }
    {'checkproperty'       }
    {'checkproptype'       }
    {'checkreadonlyproperty'}
    {'checkrealscalardouble'}
    {'circle'              }
    {'convertfreq'         }
    {'copy'                }
    {'delete'              }
    {'destroy'             }
    {'disp'                }
    {'eq'                  }
    {'extract'             }
    {'findimpedance'       }
    {'findobj'             }
    {'findprop'            }
    {'ge'                  }
    {'get'                 }
    {'getdata'             }
       ⋮
```

Change Properties of `rfckt` Objects

Use the `get` function or Dot Notation to get the line length of the first transmission line.

```
DefaultLength = FirstCkt.LineLength;
```

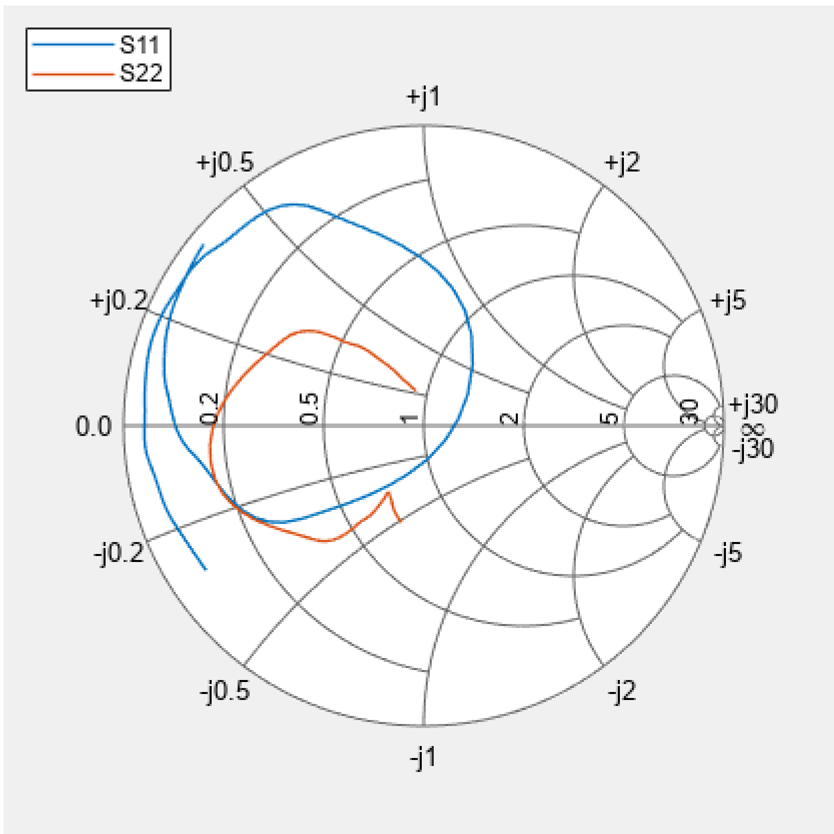
Use the `set` function or Dot notation to change the line length of the first transmission line.

```
FirstCkt.LineLength = .001;
NewLength = FirstCkt.LineLength;
```

Plot Amplifier S11 and S22 Parameters

Use the `smithplot` method of circuit object to plot the original `S11` and `S22` parameters of the amplifier (`SecondCkt`) on a Z Smith chart. The original frequencies of the amplifier's S-parameters range from 1.0 GHz to 2.9 GHz.

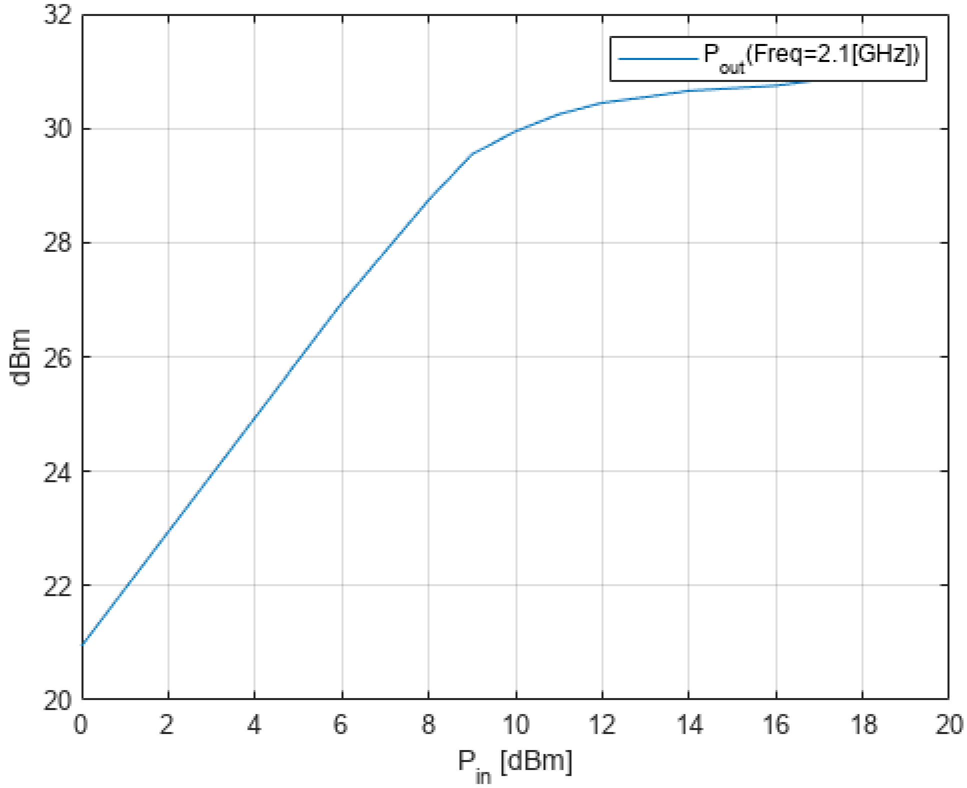
```
figure
smithplot(SecondCkt,[1 1;2 2]);
```



Plot Amplifier Pin-Pout Data

Use the `plot` method of circuit object to plot the amplifier (`SecondCkt`) Pin-Pout data, in dBm, at 2.1 GHz on an X-Y plane.

```
figure
plot(SecondCkt,'Pout','dBm')
```




```
legend('show','Location','northwest');
```

Get Original Frequency Data and Result of Analyzing Amplifier over Original Frequencies

When the RF Toolbox reads data from default.amp into an amplifier object (`SecondCkt`), it also analyzes the amplifier over the frequencies of network parameters in default.amp file and store the result at the property `AnalyzedResult`. Here are the original amplifier frequency and analyzed result over it.

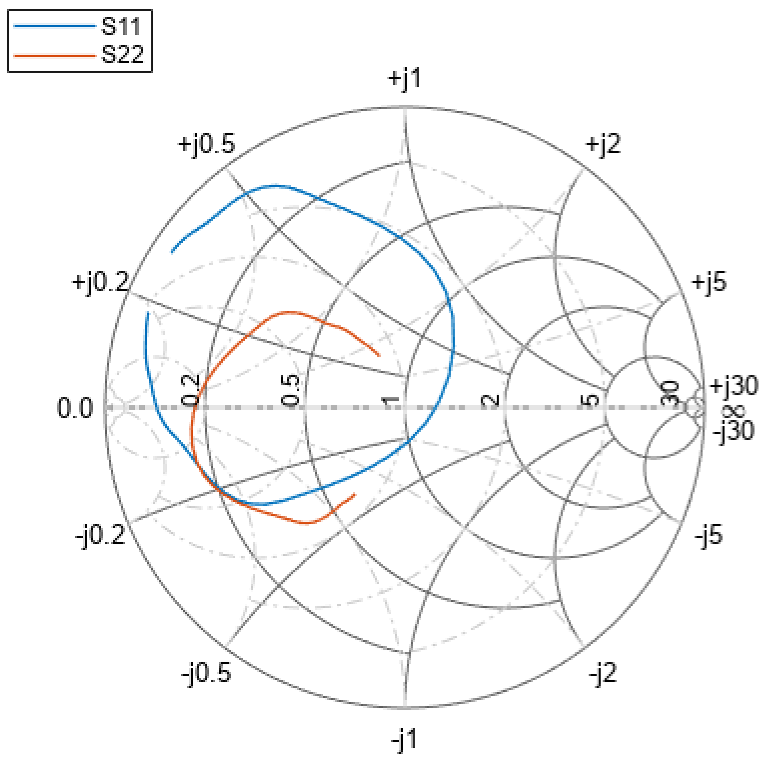
```
f = SecondCkt.AnalyzedResult.Freq;
data = SecondCkt.AnalyzedResult
```

```
data = 
   rfdata.data with properties:

            Freq: [191x1 double]
    S_Parameters: [2x2x191 double]
      GroupDelay: [191x1 double]
              NF: [191x1 double]
            OIP3: [191x1 double]
              Z0: 50.0000 + 0.0000i
              ZS: 50.0000 + 0.0000i
              ZL: 50.0000 + 0.0000i
        IntpType: 'Cubic'
            Name: 'Data object'
```

Analyze and Plot S11 and S22 of Amplifier Circuit with Different Frequencies

To visualize the S-parameters of a circuit over a different frequency range, you must first analyze it over the specified frequency range.

```
analyze(SecondCkt,1.85e9:1e7:2.55e9);
smithplot(SecondCkt,[1 1;2 2],'GridType','ZY')
```



Create and Analyze Cascaded `rfckt` Object

Cascade three circuit objects to create a cascaded circuit object, and then analyze it at the original amplifier frequencies which range from 1.0 GHz to 2.9 GHz.

```
CascadedCkt = rfckt.cascade('Ckts',{FirstCkt,SecondCkt,ThirdCkt});
analyze(CascadedCkt,f)
```

```
ans =
   rfckt.cascade with properties:

              Ckts: {[1x1 rfckt.txline]  [1x1 rfckt.amplifier]  [1x1 rfckt.txline]}
             nPort: 2
    AnalyzedResult: [1x1 rfdata.data]
              Name: 'Cascaded Network'
```

Figure 1: The cascaded circuit.

Plot S11 and S22 Parameters of Cascaded Circuit

Use the `smithplot` method of circuit object to plot `S11` and `S22` of the cascaded circuit (`CascadedCkt`) on a Z Smith chart.

```
smithplot(CascadedCkt,[1 1;2 2],'GridType','Z')
```

Plot S21 Parameters of Cascaded Circuit

Use the plot method of circuit object to plot `S21` of the cascaded circuit (`CascadedCkt`) on an X-Y plane.

```
plot(CascadedCkt,'S21','dB')
legend show;
```

Plot Budget S21 Parameters and Noise Figure of Cascaded Circuit

Use the `plot` method of circuit object to plot the budget `S21` parameters and noise figure of the cascaded circuit (`CascadedCkt`) on an X-Y plane.

```
plot(CascadedCkt,'budget','S21','NF')
legend show;
```

See Also

More About

- “Bandpass Filter Response Using RFCKT Objects” on page 6-35

Operations with RF Data Objects

This example shows you how to manipulate RF data directly using `rfdata` objects. First, you create an `rfdata.data` object by reading in the S-parameters of a two-port passive network stored in the Touchstone® format data file, `passive.s2p`. Next, you create a circuit object, `rfckt.amplifier`, and you update the properties of this object using three data objects.

Read Touchstone® Data File

Use the `read` method of the `rfdata.data` object to read the Touchstone data file `passive.s2p`. The parameters in this data file are the 50-Ohm S-parameters of a 2-port passive network at frequencies ranging from 315 kHz to 6.0 GHz.

```
data = rfdata.data;
data = read(data,'passive.s2p')
```

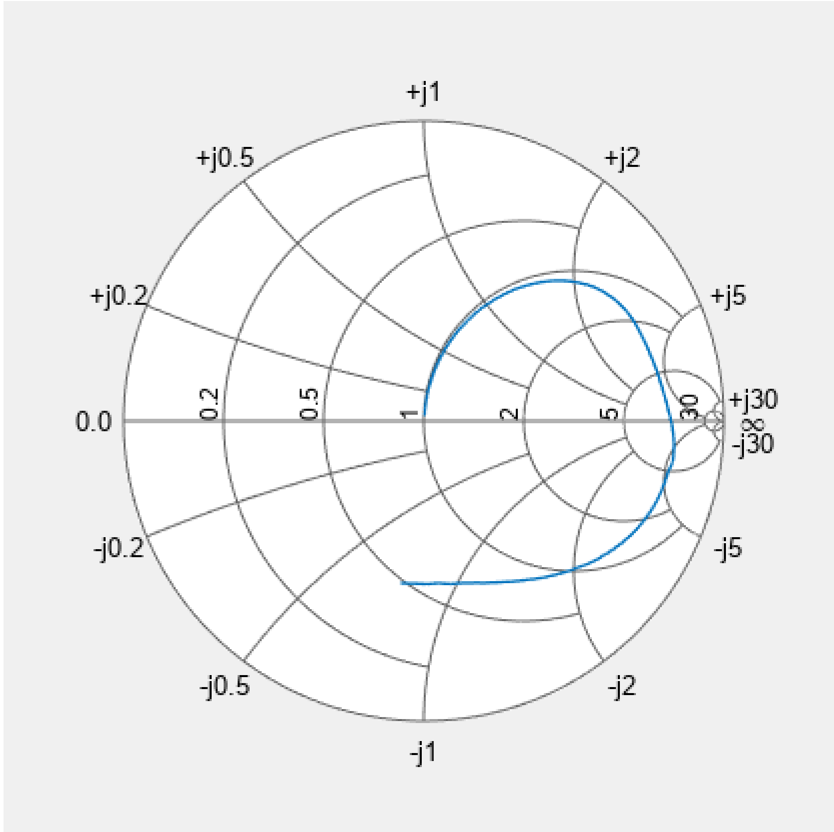
```
data = 
   rfdata.data with properties:

            Freq: [202x1 double]
    S_Parameters: [2x2x202 double]
      GroupDelay: [202x1 double]
              NF: [202x1 double]
            OIP3: [202x1 double]
              Z0: 50.0000 + 0.0000i
              ZS: 50.0000 + 0.0000i
              ZL: 50.0000 + 0.0000i
        IntpType: 'Linear'
            Name: 'Data object'
```

Use the `extract` method of the `rfdata.data` object to get other network parameters. For example, here are the frequencies, 75-Ohm S-parameters, and Y-parameters which are converted from the original 50-Ohm S-parameters in `passive.s2p` data file.

```
[s_params,freq] = extract(data,'S_PARAMETERS',75);
y_params = extract(data,'Y_PARAMETERS');
```

Use the RF utility function, `smithplot` to plot the 75-Ohm S11 on a Smith chart.

```
s11 = s_params(1,1,:);
figure
smithplot(freq, s11(:))
```



Here are the four 75-Ohm S-parameters and four Y-parameters at 6.0 GHz, the last frequency.

```
f = freq(end)
```

```
f = 6.0000e+09
```

```
s = s_params(:,:,end)
```

```
s = 2×2 complex

  -0.0764 - 0.5401i   0.6087 - 0.3018i
   0.6094 - 0.3020i  -0.1211 - 0.5223i
```

```
y = y_params(:,:,end)
```

```
y = 2×2 complex

   0.0210 + 0.0252i  -0.0215 - 0.0184i
  -0.0215 - 0.0185i   0.0224 + 0.0266i
```

Create RF Data Objects for Amplifier with Your Own Data

In this example, you create a circuit object, `rfckt.amplifier`. Then you create three data objects and use them to update the properties of the circuit object.

The `rfckt.amplifier` object has properties for network parameters, noise data and nonlinear data:

- `NetworkData` is an `rfdata.network` object for network parameters.
- `NoiseData` is for noise parameters which could be a scalar NF (dB), an `rfdata.noise`, or an `rfdata.nf` object.
- `NonlinearData` is for nonlinear parameters which could be a scalar OIP3 (dBm), an `rfdata.power`, or an `rfdata.ip3` object.

By default, these properties of `rfckt.amplifier` contain data from the `default.amp` data file. `NetworkData` is an `rfdata.network` object that contains 50-Ohm 2-port S-Parameters at 191 frequencies ranging from 1.0 GHz to 2.9 GHz. `NoiseData` is an `rfdata.noise` object that contains spot noise data at 9 frequencies ranging from 1.9 GHz to 2.48 GHz. The `NonlinearData` parameter is an `rfdata.power` object that contains `Pin/Pout` data at 2.1 GHz.

```
amp = rfckt.amplifier
```

```
amp = 
   rfckt.amplifier with properties:

          NoiseData: [1x1 rfdata.noise]
      NonlinearData: [1x1 rfdata.power]
           IntpType: 'Linear'
        NetworkData: [1x1 rfdata.network]
              nPort: 2
     AnalyzedResult: [1x1 rfdata.data]
               Name: 'Amplifier'
```

Use the following code to create an `rfdata.network` object that contains 2-port Y-parameters of an amplifier at 2.08 GHz, 2.10 GHz and 2.15 GHz. Later in this example, you use this data object to update the `NetworkData` property of the amplifier object.

```
f = [2.08 2.10 2.15] * 1.0e9;
y(:,:,1) = [-.0090-.0104i, .0013+.0018i; -.2947+.2961i, .0252+.0075i];
y(:,:,2) = [-.0086-.0047i, .0014+.0019i; -.3047+.3083i, .0251+.0086i];
y(:,:,3) = [-.0051+.0130i, .0017+.0020i; -.3335+.3861i, .0282+.0110i];
netdata = rfdata.network('Type','Y_PARAMETERS','Freq',f,'Data',y)
```

```
netdata = 
   rfdata.network with properties:

    Type: 'Y_PARAMETERS'
    Freq: [3x1 double]
    Data: [2x2x3 double]
      Z0: 50.0000 + 0.0000i
    Name: 'Network parameters'
```

Use the following code to create an `rfdata.nf` object that contains noise figures of the amplifier, in dB, at seven frequencies ranging from 1.93 GHz to 2.40 GHz. Later in this example, you use this data object to update the `NoiseData` property of the amplifier object.

```
f = [1.93 2.06 2.08 2.10 2.15 2.3 2.4] * 1.0e+009;
nf = [12.4521 13.2466 13.6853 14.0612 13.4111 12.9499 13.3244];
nfdata = rfdata.nf('Freq',f,'Data',nf)
```

```
nfdata = 
   rfdata.nf with properties:
```

```
    Freq: [7x1 double]
    Data: [7x1 double]
    Name: 'Noise figure'
```

Use the following code to create an `rfdata.ip3` object that contains the output third-order intercept points of the amplifier, which is 8.45 watts at 2.1 GHz. Later in this example, you use this data object to update the `NonlinearData` property of the amplifier object.

```
ip3data = rfdata.ip3('Type','OIP3','Freq',2.1e9,'Data',8.45)
```

```
ip3data = 
   rfdata.ip3 with properties:

    Type: 'OIP3'
    Freq: 2.1000e+09
    Data: 8.4500
    Name: '3rd order intercept'
```

Use the following code to update the properties of the amplifier object with three data objects you created in the previous steps. To get a good amplifier object, the data in these data objects must be accurate. These data could be obtained from RF measurements, or circuit simulation using other tools.

```
amp.NetworkData = netdata;
amp.NoiseData = nfdata;
amp.NonlinearData = ip3data
```

```
amp = 
   rfckt.amplifier with properties:

          NoiseData: [1x1 rfdata.nf]
      NonlinearData: [1x1 rfdata.ip3]
           IntpType: 'Linear'
        NetworkData: [1x1 rfdata.network]
              nPort: 2
     AnalyzedResult: [1x1 rfdata.data]
               Name: 'Amplifier'
```

See Also

More About

- "Create RF Objects" on page 2-2

Design IF Butterworth Bandpass Filter

This example shows how to design an Intermediate Frequency (IF) Butterworth bandpass filter with a center frequency of 400 MHz, bandwidth of 5 MHz, and Insertion Loss (IL) of 1dB [1] on page 6-186.

Account for Mismatch/Insertion Loss (IL)

Practical circuits suffer a certain degree of mismatch. Mismatch happens when an unmatched circuit is connected to an RF source leading to reflections that result in a loss of power delivered to the circuit. You can use IL to define this mismatch. Calculate the load impedance mismatch to account for the given IL. The IL and normalized load impedance (ZL) are related as follows [2] on page 6-186,[3] on page 6-186:

IL (dB) = -10*log10(1-| γ_{in} |^2) = -10*log10(4*ZL/(1+ZL)^2)

The roots of the resulting polynomial return the value of normalized load impedance. The unnormalized values are 132.986 Ohms and 18.799 Ohms. Choose the higher value for the filter design to account for the IL.

```
syms ZL IL
eqn = -10*log10(4*ZL/(1+ZL)^2) - IL == 0;
[solx, ~, ~] = solve(eqn,ZL,'ReturnConditions', true);
IL_desired_dB = 1;
Zload = double(subs(solx,IL,IL_desired_dB))*50;
```

Load impedance:

```
ZL = Zload(2);
```

Design Filter

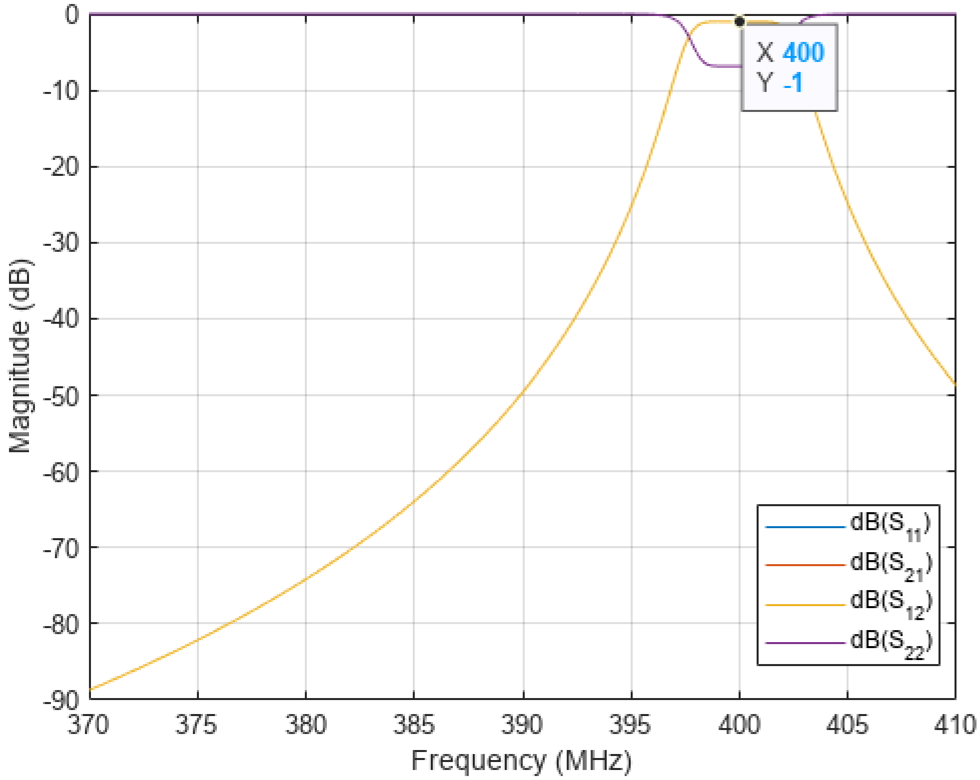
Use `rffilter` to design the filter for the desired specifications.

```
Fcenter = 400e6;
Bwpass  = 5e6;
if_filter = rffilter('ResponseType','Bandpass',...
    'FilterType','Butterworth','FilterOrder',4,...
    'PassbandAttenuation',10*log10(2),...
    'Implementation','Transfer function',...
    'PassbandFrequency',[Fcenter-Bwpass/2 Fcenter+Bwpass/2],'Zout',ZL);
```

Plot S-parameters and Group Delay of Filter

Calculate S-parameters.

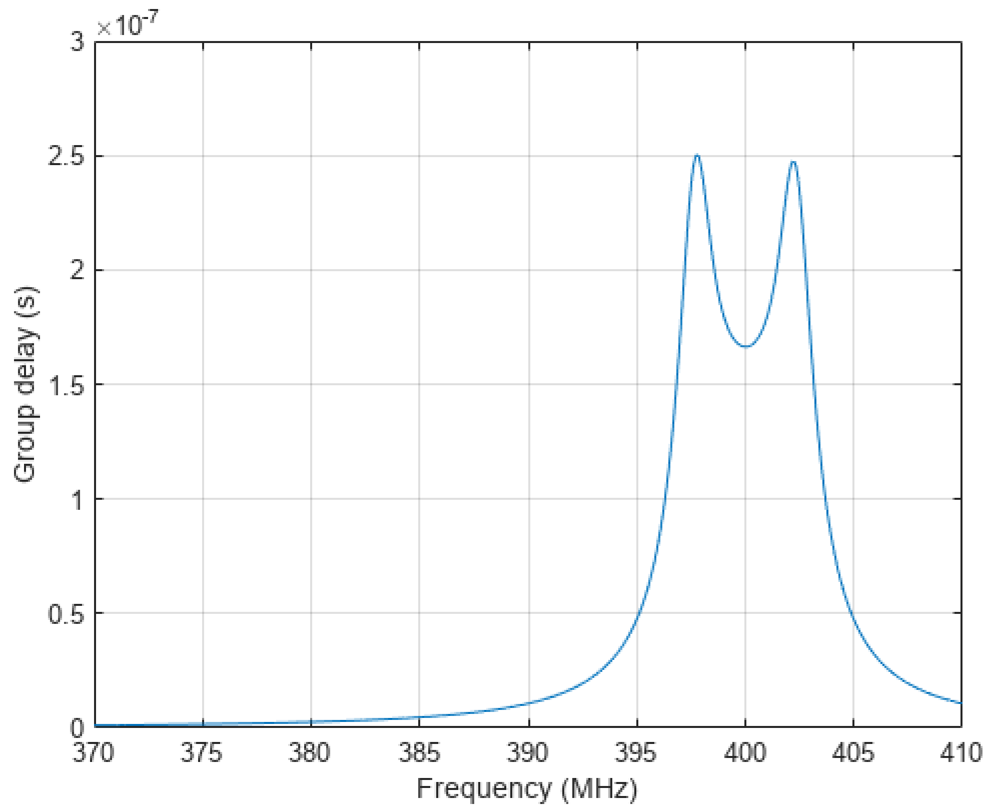
```
freq = linspace(370e6,410e6,2001);
Sf = sparameters(if_filter, freq);
figure;
line = rfplot(Sf);
lgd = legend;
lgd.Location = "best";
[~,freq_index] = min(abs(freq-Fcenter));
datatip(line(3),'DataIndex',freq_index);
```



A datatip shows a 1dB IL at Fcenter = 400 MHz.

Calculate groupdelay:

```
gd = groupdelay(if_filter, freq);
figure;
plot(freq/1e6, gd);
xlabel('Frequency (MHz)');
ylabel('Group delay (s)');
grid on;
```



Insert Filter into rfbudget Object

An rffilter object can be inserted directly into an `rfbudget` object to perform budget analysis.

```
rfb = rfbudget(if_filter,Fcenter,-30,Bwpass)
```

```
rfb = 
  rfbudget with properties:

               Elements: [1x1 rffilter]
         InputFrequency: 400 MHz
    AvailableInputPower: -30 dBm
        SignalBandwidth:   5 MHz
                 Solver: Friis
             AutoUpdate: true

   Analysis Results
        OutputFrequency:   400 (MHz)
            OutputPower:   -31 (dBm)
         TransducerGain:    -1 (dB)
                     NF:     0 (dB)
                   IIP2:    [] (dBm)
                   OIP2:    [] (dBm)
                   IIP3:   Inf (dBm)
                   OIP3:   Inf (dBm)
                    SNR: 76.99 (dB)
```

References

[1] Hongbao Zhou, Bin Luo. " Design and budget analysis of RF receiver of 5.8GHz ETC reader" Published at Communication Technology (ICCT), 2010 12th IEEE International Conference, Nanjing, China, November 2010.

[2] Electronic Filter Analysis and Synthesis, Michael G. Ellis, Sr., Artech House, Chapter 7.

[3] RF Circuit Design, R. Ludwig, G. Bogdanov, Pearson Education, Chapter 2.

See Also

“Superheterodyne Receiver Using RF Budget Analyzer App” on page 6-2

See Also

More About

- “Bandpass Filter Response” on page 6-23

Passivity: Test, Visualize, and Enforce Passivity of Rational Fit Output

This example shows how to test, visualize, and enforce the passivity of output from the `rationalfit` function.

S-Parameter Data Passivity

Time-domain analysis and simulation depends critically on being able to convert frequency-domain S-parameter data into causal, stable, and passive time-domain representations. Because the `rationalfit` function guarantees that all poles are in the left half plane, `rationalfit` output is both stable and causal by construction. The problem is passivity.

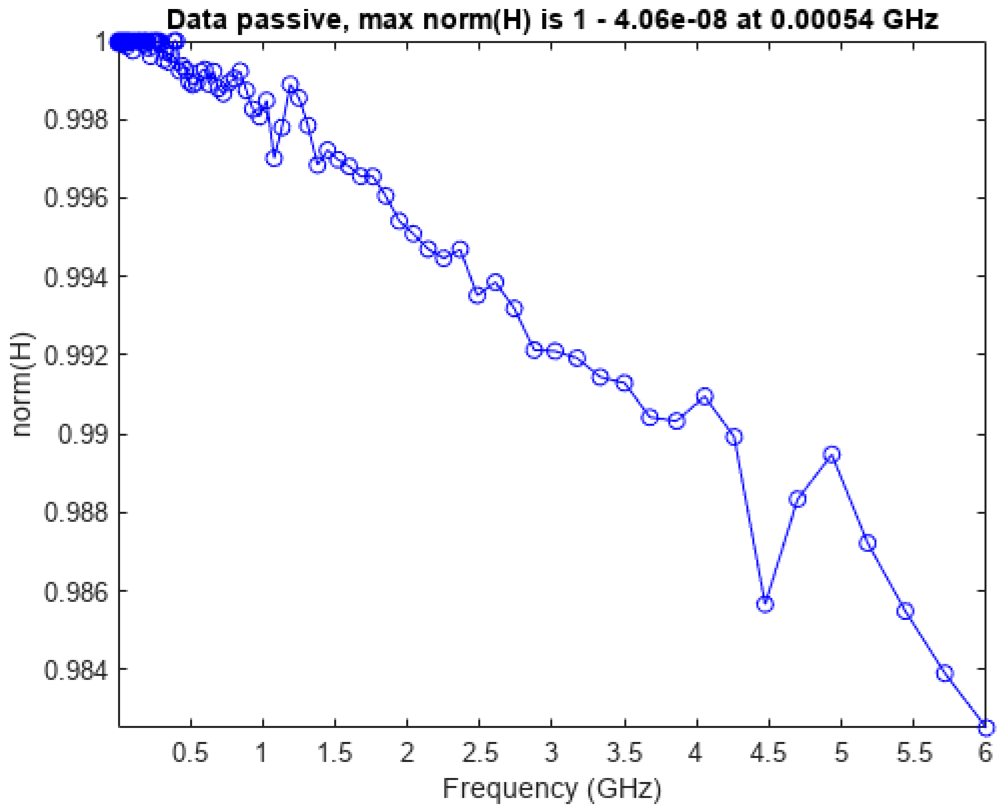
N-port S-parameter data represents a frequency-dependent transfer function H(f). You can create an S-parameters object in RF Toolbox™ by reading a Touchstone® file, such as `passive.s2p`, into the `sparameters` function.

You can use the `ispassive` function to check the passivity of the S-parameter data, and the `passivity` function to plot the 2-norm of the N x N matrices H(f) at each data frequency.

```
S = sparameters('passive.s2p');
ispassive(S)
```

```
ans = logical
   1
```

```
passivity(S)
```



Testing and Visualizing `rationalfit` Output Passivity

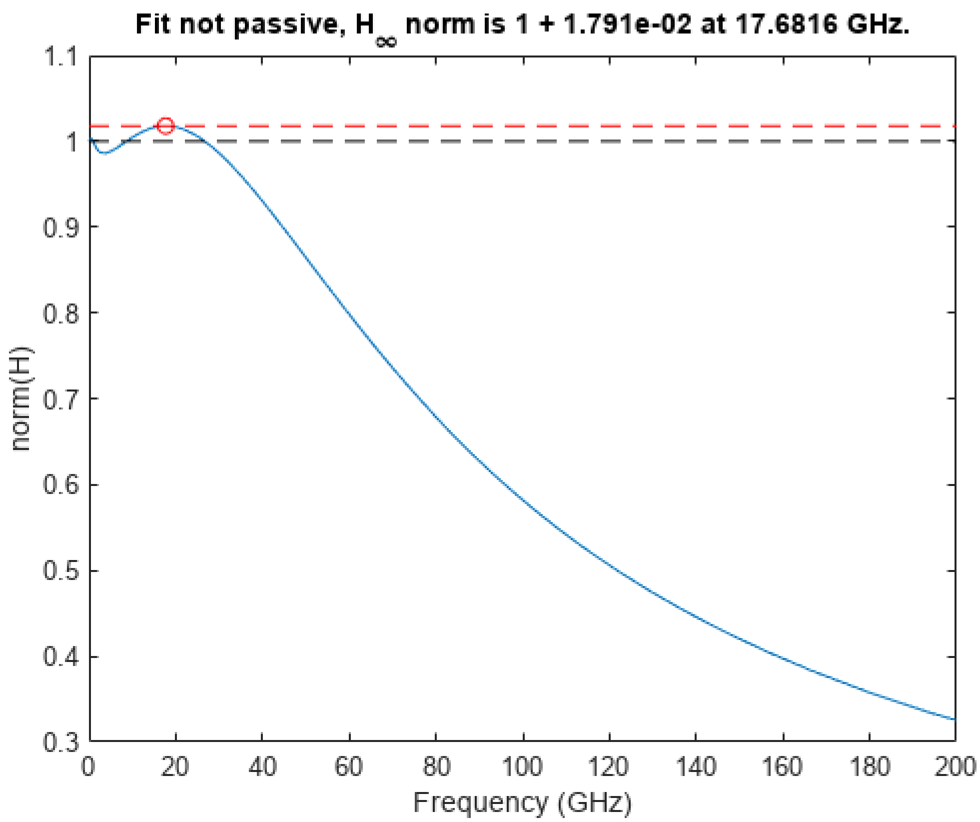
The `rationalfit` function converts N-port sparameter data, `S` into an NxN matrix of `rfmodel.rational` objects. Using the `ispassive` function on the N x N fit output reports that even if input data `S` is passive, the output fit is not passive. In other words, the norm H(f) is greater than one at some frequency in the range [0,Inf].

The `passivity` function takes an N x N fit as input and plots its passivity. This is a plot of the upper bound of the norm(H(f)) on [0,Inf], also known as the H-infinity norm.

```
fit = rationalfit(S);
ispassive(fit)
```

```
ans = logical
   0
```

```
passivity(fit)
```



The makepassive function takes as input an N x N array of fit objects and also the original S-parameter data, and produces a passive fit by using convex optimization techniques to optimally match the data of the S-parameter input S while satisfying passivity constraints. The residues C and feedthrough matrix D of the output pfit are modified, but the poles A of the output pfit are identical to the poles A of the input fit.

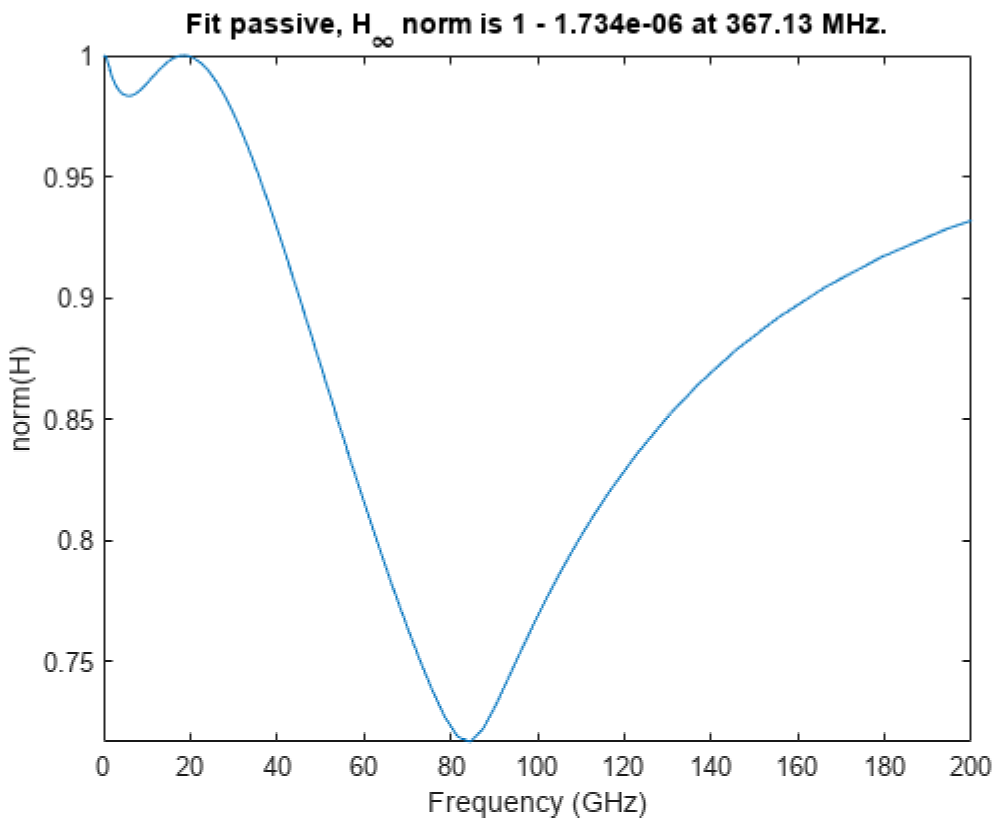
```
pfit = makepassive(fit,S,'Display','on');
```

```
ITER     H-INFTY NORM    FREQUENCY        ERRDB       CONSTRAINTS
0        1 + 1.791e-02    17.6816  GHz    -40.4702
1        1 + 2.681e-04    282.5    MHz    -40.9168    5
2        1 + 7.008e-05    377.725  MHz    -40.9076    8
3        1 + 2.178e-06    359.574  MHz    -40.9068    9
4        1 - 1.734e-06    367.13   MHz    -40.9062    10
```

```
ispassive(pfit)
```

```
ans = logical
   1
```

```
passivity(pfit)
```




```
all(vertcat(pfit(:).A) == vertcat(fit(:).A))
```

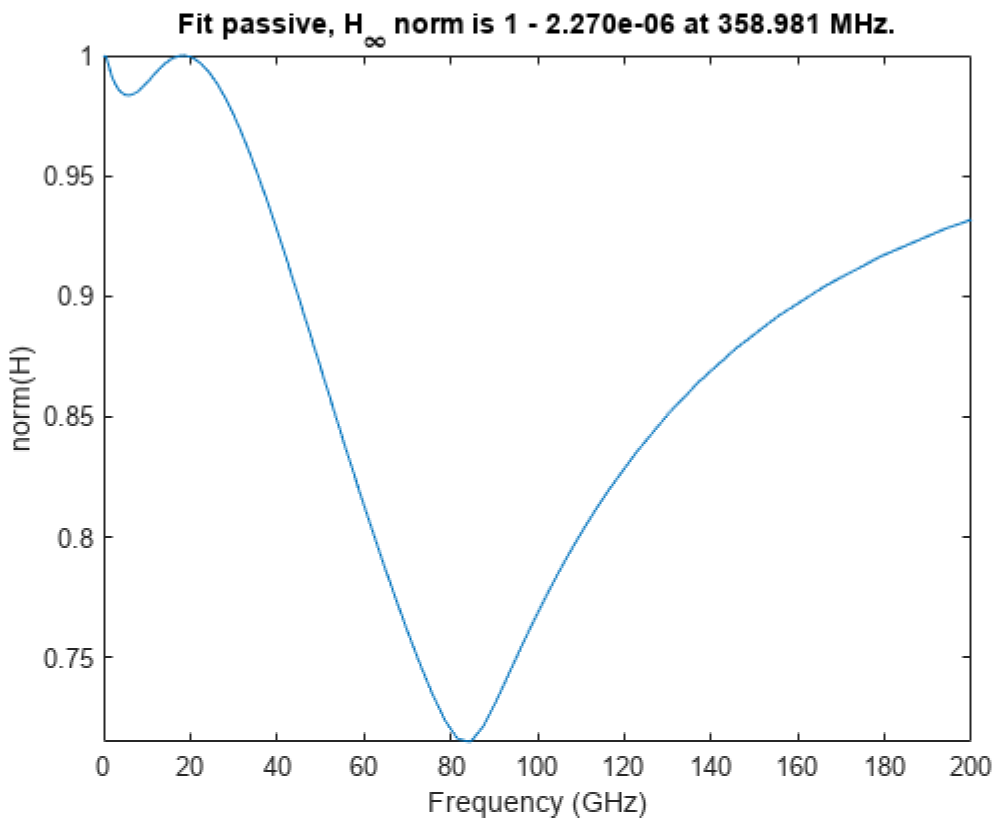
```
ans = logical
   1
```

Start makepassive with Prescribed Poles and Zero C and D

To demonstrate that only C and D are modified by `makepassive`, one can zero out C and D and re-run `makepassive`. The output, `pfit` still has the same poles as the input fit. The differences between `pfit` and `pfit2` arise because of the different starting points of the convex optimizations.

One can use this feature of the `makepassive` function to produce a passive fit from a prescribed set of poles without any idea of starting C and D.

```
for k = 1:numel(fit)
    fit(k).C(:) = 0;
    fit(k).D(:) = 0;
end
pfit2 = makepassive(fit,S);
passivity(pfit2)
```




```
all(vertcat(pfit2(:).A) == vertcat(fit(:).A))
```

```
ans = logical
   1
```

Generate Equivalent SPICE Circuit from Passive Fit

The `generateSPICE` function takes a passive fit and generates an equivalent circuit as a SPICE subckt file. The input fit is an N x N array of `rfmodel.rational` objects as returned by rationalfit with an S-parameters object as input. The generated file is a SPICE model constructed solely of passive R, L, C elements and controlled source elements E, F, G, and H.

```
generateSPICE(pfit2,'mypassive.ckt')
type mypassive.ckt
```

```
* Equivalent circuit model for mypassive.ckt
.SUBCKT mypassive po1 po2
Vsp1 po1 p1 0
Vsr1 p1 pr1 0
Rp1 pr1 0 50
Ru1 u1 0 50
Fr1 u1 0 Vsr1 -1
Fu1 u1 0 Vsp1 -1
Ry1 y1 0 1
Gy1 p1 0 y1 0 -0.02
Vsp2 po2 p2 0
Vsr2 p2 pr2 0
```

```
Rp2 pr2 0 50
Ru2 u2 0 50
Fr2 u2 0 Vsr2 -1
Fu2 u2 0 Vsp2 -1
Ry2 y2 0 1
Gy2 p2 0 y2 0 -0.02
Rx1 x1 0 1
Fxc1_2 x1 0 Vx2 18.8511829952899
Cx1 x1 xm1 3.95175907226011e-09
Vx1 xm1 0 0
Gx1_1 x1 0 u1 0 -0.092166465258608
Rx2 x2 0 1
Fxc2_1 x2 0 Vx1 -0.0833090255364873
Cx2 x2 xm2 3.95175907226011e-09
Vx2 xm2 0 0
Gx2_1 x2 0 u1 0 0.00767829840783715
Rx3 x3 0 1
Cx3 x3 0 2.73023889928382e-12
Gx3_1 x3 0 u1 0 -2.06210448771304
Rx4 x4 0 1
Cx4 x4 0 7.77758884882576e-12
Gx4_1 x4 0 u1 0 -2.91822063634408
Rx5 x5 0 1
Cx5 x5 0 2.29141629980399e-11
Gx5_1 x5 0 u1 0 -0.544252116941361
Rx6 x6 0 1
Cx6 x6 0 9.31845201627124e-11
Gx6_1 x6 0 u1 0 -0.65447079495975
Rx7 x7 0 1
Cx7 x7 0 4.89917764982128e-10
Gx7_1 x7 0 u1 0 -0.0811043328984435
Rx8 x8 0 1
Cx8 x8 0 1.25490425570277e-08
Gx8_1 x8 0 u1 0 -0.947605341201612
Rx9 x9 0 1
Fxc9_10 x9 0 Vx10 18.4757545197087
Cx9 x9 xm9 3.95175907226011e-09
Vx9 xm9 0 0
Gx9_2 x9 0 u2 0 -0.0931477754195798
Rx10 x10 0 1
Fxc10_9 x10 0 Vx9 -0.0850018700926301
Cx10 x10 xm10 3.95175907226011e-09
Vx10 xm10 0 0
Gx10_2 x10 0 u2 0 0.00791773510563261
Rx11 x11 0 1
Cx11 x11 0 2.73023889928382e-12
Gx11_2 x11 0 u2 0 -2.08590438543279
Rx12 x12 0 1
Cx12 x12 0 7.77758884882576e-12
Gx12_2 x12 0 u2 0 -2.92844449800997
Rx13 x13 0 1
Cx13 x13 0 2.29141629980399e-11
Gx13_2 x13 0 u2 0 -0.60704076676533
Rx14 x14 0 1
Cx14 x14 0 9.31845201627123e-11
Gx14_2 x14 0 u2 0 -0.692675539791758
Rx15 x15 0 1
Cx15 x15 0 4.89917764982128e-10
```

```
Gx15_2 x15 0 u2 0 -0.086056124835021
Rx16 x16 0 1
Cx16 x16 0 1.25490425570277e-08
Gx16_2 x16 0 u2 0 -0.948041381714949
Gyc1_1 y1 0 x1 0 -1
Gyc1_2 y1 0 x2 0 -1
Gyc1_3 y1 0 x3 0 -0.140208229154557
Gyc1_4 y1 0 x4 0 -0.0224130457738786
Gyc1_5 y1 0 x5 0 -1
Gyc1_6 y1 0 x6 0 -1
Gyc1_7 y1 0 x7 0 1
Gyc1_8 y1 0 x8 0 0.999881486167614
Gyc1_9 y1 0 x9 0 0.989930749917605
Gyc1_10 y1 0 x10 0 0.966001267019067
Gyc1_11 y1 0 x11 0 1
Gyc1_12 y1 0 x12 0 -1
Gyc1_13 y1 0 x13 0 0.810855526722219
Gyc1_14 y1 0 x14 0 0.94182086372261
Gyc1_15 y1 0 x15 0 -0.935895407236664
Gyc1_16 y1 0 x16 0 -0.999947034175042
Gyd1_1 y1 0 u1 0 0.604679239616378
Gyd1_2 y1 0 u2 0 -0.35333173208655
Gyc2_1 y2 0 x1 0 0.998726972393021
Gyc2_2 y2 0 x2 0 0.974259384727238
Gyc2_3 y2 0 x3 0 1
Gyc2_4 y2 0 x4 0 -1
Gyc2_5 y2 0 x5 0 0.90076447323033
Gyc2_6 y2 0 x6 0 0.997053363381364
Gyc2_7 y2 0 x7 0 -0.99261933063484
Gyc2_8 y2 0 x8 0 -1
Gyc2_9 y2 0 x9 0 -1
Gyc2_10 y2 0 x10 0 -1
Gyc2_11 y2 0 x11 0 -0.262453289009206
Gyc2_12 y2 0 x12 0 0.0673121121592632
Gyc2_13 y2 0 x13 0 -1
Gyc2_14 y2 0 x14 0 -1
Gyc2_15 y2 0 x15 0 1
Gyc2_16 y2 0 x16 0 1
Gyd2_1 y2 0 u1 0 -0.338057614833956
Gyd2_2 y2 0 u2 0 0.696902323249447
.ENDS
```

See Also

More About

- “Using 'NPoles' Parameter With rationalfit” on page 6-73
- “Using 'Weight' Parameter With rationalfit” on page 6-77
- “Using 'DelayFactor' Parameter With rationalfit” on page 6-83

Design, Visualize and Explore Inverse Chebyshev Filter - I

This example shows how to determine the transfer function for a fifth-order inverse Chebyshev lowpass filter with 1 dB passband attenuation, cutoff frequency of 1 rad/sec, and a minimum attenuation of 50 dB in the stopband. Determine the amplitude response at 2 rad/sec [1].

The `rffilter object` is used to design a RF Filter. A filter requires a minimum set of parameters for it to be completely defined. Refer to the table in the `rffilter` documentation page which reflects this set of required parameters. Each set of parameters result in its corresponding syntax. Input these parameters as name-value pairs to `rffilter` to design the specified filter. Note that the parameters which are required but are not defined assume default values.

After initialization of an `rffilter` object, the property `DesignData` contains the complete solution of the filter designed. It is a structure which contains fields such as the computed factorized polynomials for the construction of the transfer function.

Design Chebyshev Type II filter

```
N           = 5;           % Filter order
Fp          = 1/(2*pi);    % Passband cutoff frequency
Ap          = 1;           % Passband attenuation
As          = 50;          % Stopband attenuation
```

Use `rffilter` object to create a desired filter. The only implementation type for Inverse Chebyshev is 'Transfer function'.

```
r = rffilter('FilterType','InverseChebyshev','ResponseType','Lowpass',  ...
    'Implementation','Transfer function','FilterOrder',N,               ...
    'PassbandFrequency',Fp,'StopbandAttenuation',As,                    ...
    'PassbandAttenuation',Ap);
```

Generate and Visualize Transfer Function Polynomial

Use `tf` function to generate transfer function polynomials.

```
[numerator, denominator] = tf(r);
format long g
```

Display `Numerator21` polynomial coefficients.

```
disp('Numerator polynomial coefficients of Transfer function');
```

```
Numerator polynomial coefficients of Transfer function
```

```
disp(numerator{2,1});
```

```
        0.0347736250821381                         0         0.672768334081369
```

Display Denominator polynomial coefficients.

```
disp('Denominator polynomial coefficients of Transfer function');
```

```
Denominator polynomial coefficients of Transfer function
```

```
disp(denominator);
```

```
                         1          3.81150884154936           7.2631952221038           8.613445
```

Optionally, use Control System Toolbox™ to visualize all transfer functions.

```
G_s = tf(numerator,denominator)
```

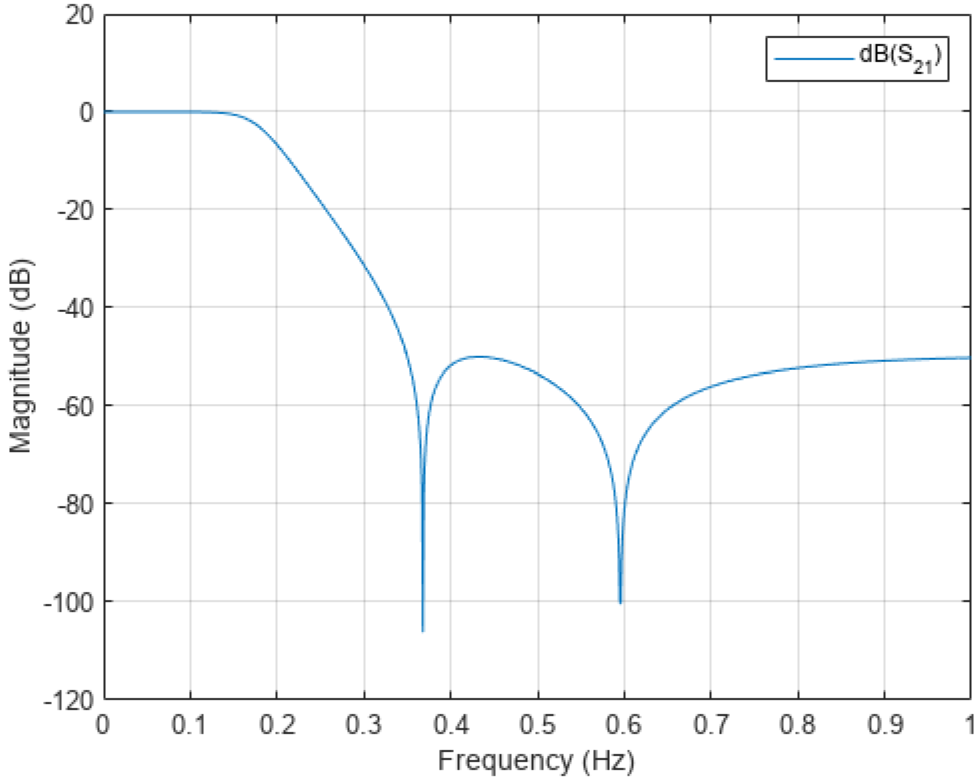
```
G_s =

  From input 1 to output...
                                s^5
   1:  --------------------------------------------------------
       s^5 + 3.812 s^4 + 7.263 s^3 + 8.613 s^2 + 6.43 s + 2.603

                   0.03477 s^4 + 0.6728 s^2 + 2.603
   2:  --------------------------------------------------------
       s^5 + 3.812 s^4 + 7.263 s^3 + 8.613 s^2 + 6.43 s + 2.603

  From input 2 to output...
                   0.03477 s^4 + 0.6728 s^2 + 2.603
   1:  --------------------------------------------------------
       s^5 + 3.812 s^4 + 7.263 s^3 + 8.613 s^2 + 6.43 s + 2.603

                                s^5
   2:  --------------------------------------------------------
       s^5 + 3.812 s^4 + 7.263 s^3 + 8.613 s^2 + 6.43 s + 2.603

Continuous-time transfer function.
```

Visualize Amplitude Response of Filter

```
frequencies = linspace(0,1,1001);
Sparam      = sparameters(r, frequencies);
```

Note: S-parameters computes the transfer function using quadratic (lowpass/highpass) or quartic (bandpass/bandstop) factorized forms. These factors are used to construct the polynomials. The polynomial form is numerically unstable for larger filter order so the preferred form is the factorized quadratic/quartic forms. These factorized parts are present in `r.DesignData`. For example, the `numerator21` can be accessed using `r.DesignData.Numerator21`.

```
l = rfplot(Sparam,2,1);
```

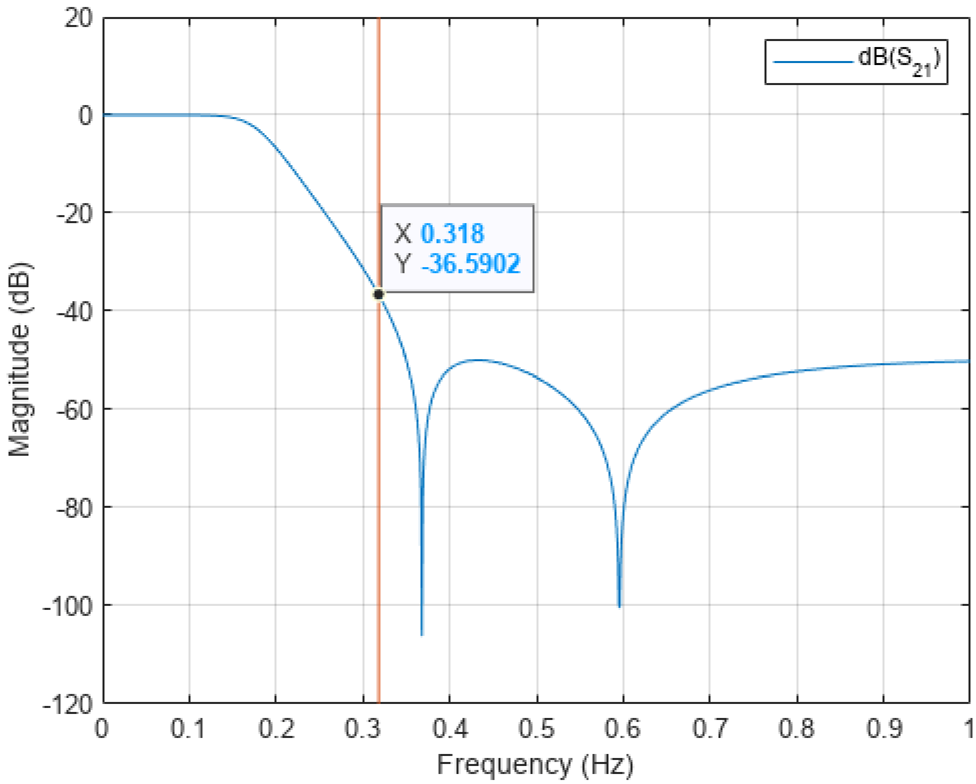


Amplitude Response of Filter at Specified Frequency

```
freq     = 2/(2*pi);
hold on;
setrfplot('noengunits',false);
```

Note: To use `rfplot` and plot on the same figure use `setrfplot`.

```
plot(freq*ones(1,101),linspace(-120,20,101));
setrfplot('engunits',false);
[~,freq_index]= min(abs(frequencies-freq));
datatip(l,'DataIndex',freq_index);
```



Using the data tip, the magnitude at 2 rad/sec is found to be -36.59 dB.

Evaluate the exact value at 2 rad/sec.

```
S_freq   = sparameters(r,freq);
As_freq  = 20*log10(abs(rfparam(S_freq,2,1)));
sprintf('Amplitude response at 2 rad/sec is %d dB',As_freq)
```

```
ans =
'Amplitude response at 2 rad/sec is -3.668925e+01 dB'
```

Calculate Stopband Frequency at As

```
Fs      = r.DesignData.Auxiliary.Wx*r.PassbandFrequency;
sprintf('Stopband frequency at -%d dB is: %d Hz',As, Fs)
```

```
ans =
'Stopband frequency at -50 dB is: 3.500241e-01 Hz'
```

References

[1] Ellis, Michael G. *Electronic Filter Analysis and Synthesis*. Boston: Artech House, 1994.

See Also

More About

- “Design, Visualize and Explore Inverse Chebyshev Filter - II” on page 6-199
- “Design IF Butterworth Bandpass Filter” on page 6-183

Design, Visualize and Explore Inverse Chebyshev Filter - II

This example shows how to design a fourth-order inverse Chebyshev low-pass filter with stopband frequency of 10000 rad/sec, and epsilon of 0.01 (please see the reference section) using `rffilter`. This `rffilter` could be used in a `circuit` or in a `rfbudget` object.

The `rffilter` object is used to design a RF filter. A filter requires a minimum set for parameters to completely define it.

The parameters to design an inverse Chebyshev filter can be one of the following:

- Filter order, Passband frequency, Passband and Stopband Attenuation
- Passband and Stopband frequencies, Passband and Stopband Attenuation
- Filter order, Stopband frequency, Stopband Attenuation

Design Filter

```
N           = 4;                                % Filter order
Fs          = 1000/(2*pi);                      % Stopband frequency
epsilon     = 0.01;
Rs          = 10*log10((1+epsilon^2)/epsilon^2);    % Stopband attenuation
```

Use the first set of parameters to define the filter.

```
r = rffilter('FilterType','InverseChebyshev','ResponseType','Lowpass',  ...
    'Implementation','Transfer function','FilterOrder',N,               ...
    'PassbandFrequency',Fs,'PassbandAttenuation',Rs,                    ...
    'StopbandAttenuation',Rs);
```

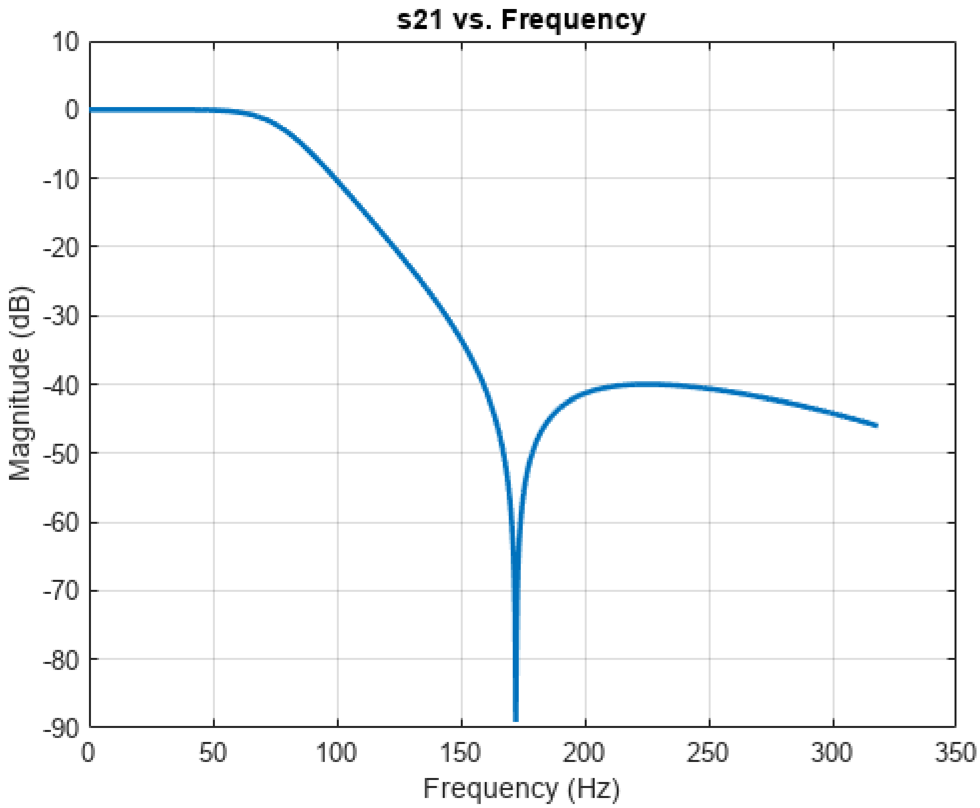
Note: Alternative, you can also use the third set of parameters to design the same filter:

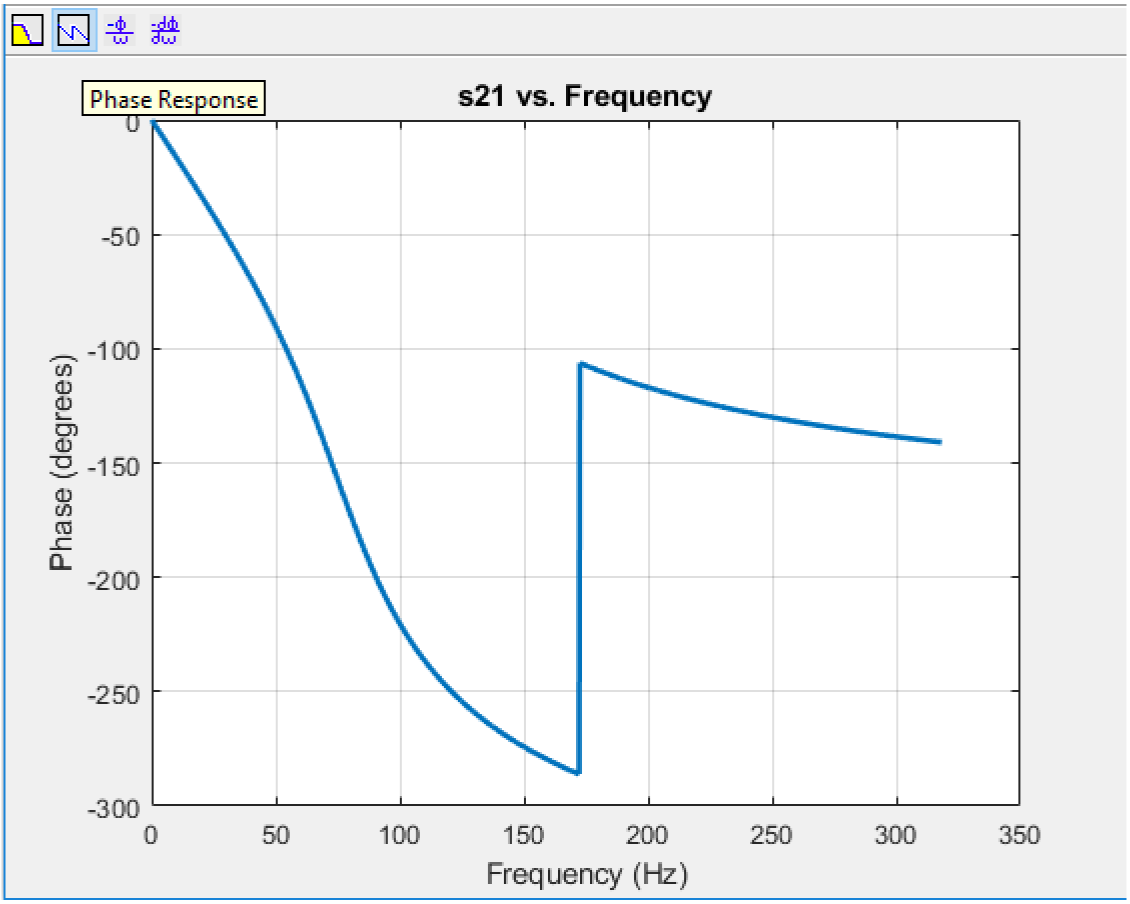
```
r = rffilter('FilterType','InverseChebyshev','ResponseType','Lowpass',  ...
'Implementation','Transfer function','FilterOrder',N,                   ...
'StopbandFrequency',Fs,'StopbandAttenuation',Rs);
```

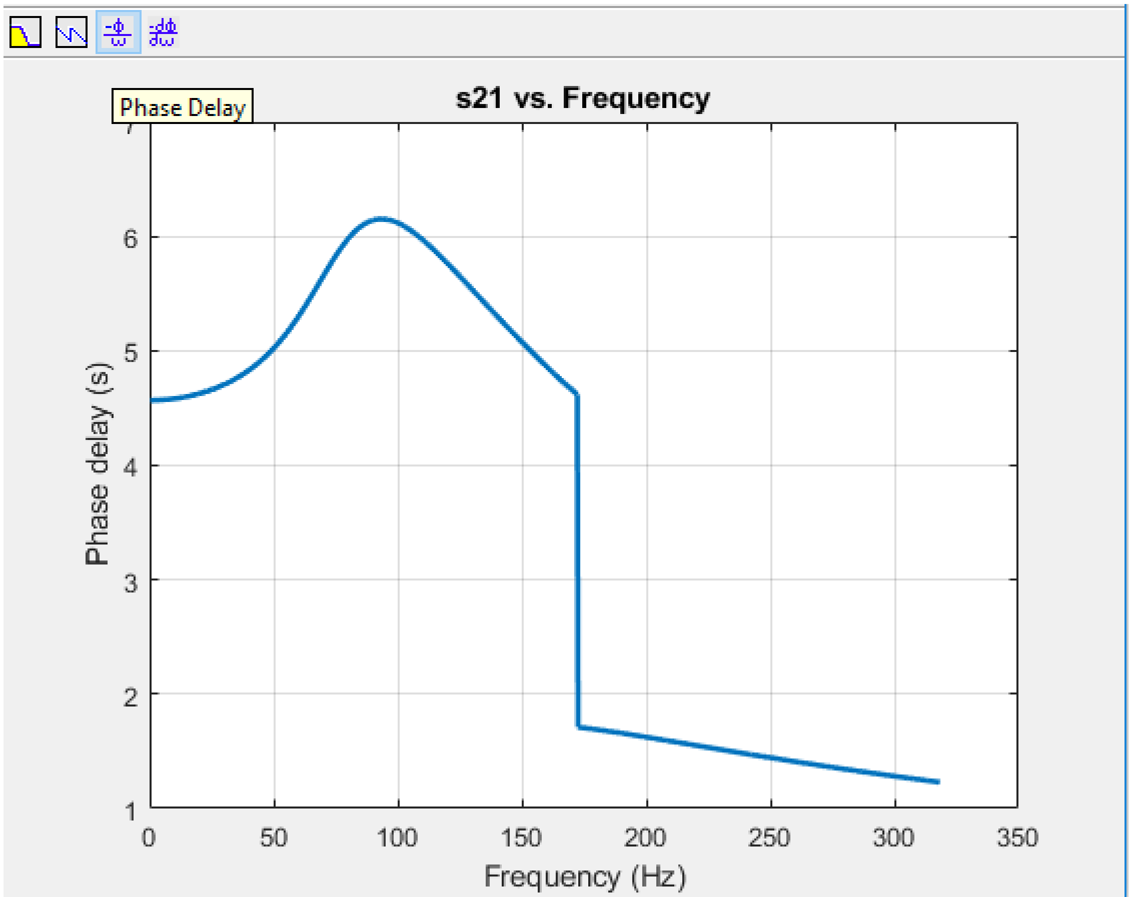
The limitation of this parameter set is that it assumes the passband attenuation to be fixed at 10*log10(2) dB.

Visualize magnitude response, phase response, and phase delay of filter

```
frequencies = linspace(0,2*Fs,1001);
rfplot(r, frequencies);
```


s21 vs. Frequency
Magnitude (dB)
Frequency (Hz)
10
0
-10
-20
-30
-40
-50
-60
-70
-80
-90
0
50
100
150
200
250
300
350


Phase Response
s21 vs. Frequency
Phase (degrees)
Frequency (Hz)
0
-50
-100
-150
-200
-250
-300
0
50
100
150
200
250
300
350



Optionally, you can also use Signal Processing Toolbox to visualize the analog filter using:

```
freqs(numerator{2,1},denominator)
```

Find zeros, poles, and gain

```
[z,p,k] = zpk(r);
```

You can obtain zeros, poles, and gain of Transfer function (S21) by:

```
format long g
zeros_21 = z{2,1}
```

```
zeros_21 = 4×1 complex

                         0 +      1082.39220029239i
                         0 -      1082.39220029239i
                         0 +      2613.12592975275i
                         0 -      2613.12592975275i
```

```
poles_21 = p                   % Same denominator for S11, S12, S21 and S22
```

```
poles_21 = 4×1 complex

          -171.158733950657 +      476.096694464131i
```

```
 -171.158733950657 -      476.096694464131i
 -504.530434776367 +      240.786480832184i
 -504.530434776367 -      240.786480832184i
```

```
k_21     = k{2,1}
```

```
k_21 =
       0.00999950003749688
```

View transfer function in factorized form

View these factor forms directly from the filter **r**.

```
disp('Numerator of Transfer function as factors:');
```

```
Numerator of Transfer function as factors:
```

```
r.DesignData.Numerator21
```

```
ans = 2×3

                      1                         0          1171572.87525381
       0.00999950003749688                         0          68280.8572899443
```

```
disp('Denominator of Transfer function as factors:');
```

```
Denominator of Transfer function as factors:
```

```
r.DesignData.Denominator
```

```
ans = 2×3

                         1          342.317467901314          255963.374687264
                         1          1009.06086955273          312529.088967178
```

Alternatively, use |zpk| from Control System Toolbox to view the transfer function in factorized form.

```
G_s = zpk(zeros_21,poles_21,k_21)
```

```
G_s =

     0.0099995 (s^2 + 1.172e06) (s^2 + 6.828e06)
  -------------------------------------------------
  (s^2 + 1009s + 3.125e05) (s^2 + 342.3s + 2.56e05)

Continuous-time zero/pole/gain model.
```

References

[1] Paarmann, L. D. *Design and Analysis of Analog Filters: A Signal Processing Perspective.* SECS 617. Boston: Kluwer Academic Publishers, 2001.

See Also

More About

- “Design, Visualize and Explore Inverse Chebyshev Filter - I” on page 6-194
- “Design IF Butterworth Bandpass Filter” on page 6-183

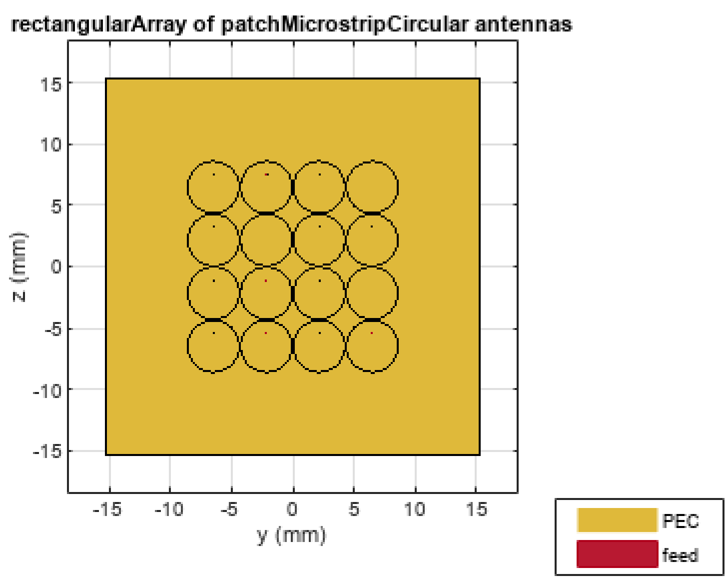
Design Matching Networks for Passive Multiport Network

This example shows how to design matching networks for 16-port passive network at 39 GHz for 5G mmWave systems. Matching networks are designed independently for each port, and each generated matching network is intended to function between two 1-port terminations.

Design Multiport Passive Network

Compute the S-Parameters of a patch antenna array designed at 39 GHz. Load the `sparams_patchArray.mat` file. The `s_params_circ_array` function is obtained from the supporting file `designmultiport.mlx`.

```
Fcenter = 39e9;
load('sparams_patchArray.mat')
Sparam_array = s_params_circ_array;
show(patchArray)
view([90 0])
```



Determine the index corresponding to the center frequency.

```
freq    = Sparam_array.Frequencies;
fIndex  = find(freq == Fcenter);
```

Create Matching Networks

Generate matching networks for each corresponding port independently, with a Loaded Q of 20 and configure the topology to 'Pi'. This Q-factor is aligned with half power bandwidth of the patch antenna array, which is approximately 2 GHz.

Define the number of ports in the network and specify the termination impedance.

```
numport     = s_params_circ_array.NumPorts;
ZT          = 50;
loadedQ     = 20;
topology    = 'Pi';
for i = 1 : numport
    % reflection coefficient/Sii
    gam_array       = s_params_circ_array.Parameters(i,i,fIndex);
    % Load impedance
    Zout            = gamma2z(gam_array);
    % Matching networks generation
    match_net(i)    = matchingnetwork('SourceImpedance', ZT, ...
        'LoadImpedance', Zout, 'CenterFrequency', Fcenter, ...
        'LoadedQ', loadedQ, 'Components', topology);
end
```

The source is connected to the component located on left of the matching network circuit and the load is connected to the component connected to the right of the matching network circuit. For the matching networks generated, the source is terminated with ZT (50 Ohm) and the load impedance is the impedance seen at the ith-port given by Zout.

View and Select Circuits

Select a topology from the sixteen `matchingnetwork` objects. To get an overview of the available circuits, see `circuitDescriptions` function.

In this example, a Shunt C-Series L-Shunt C topology is used. If this topology is not available in your network, use the best available matching network circuit.

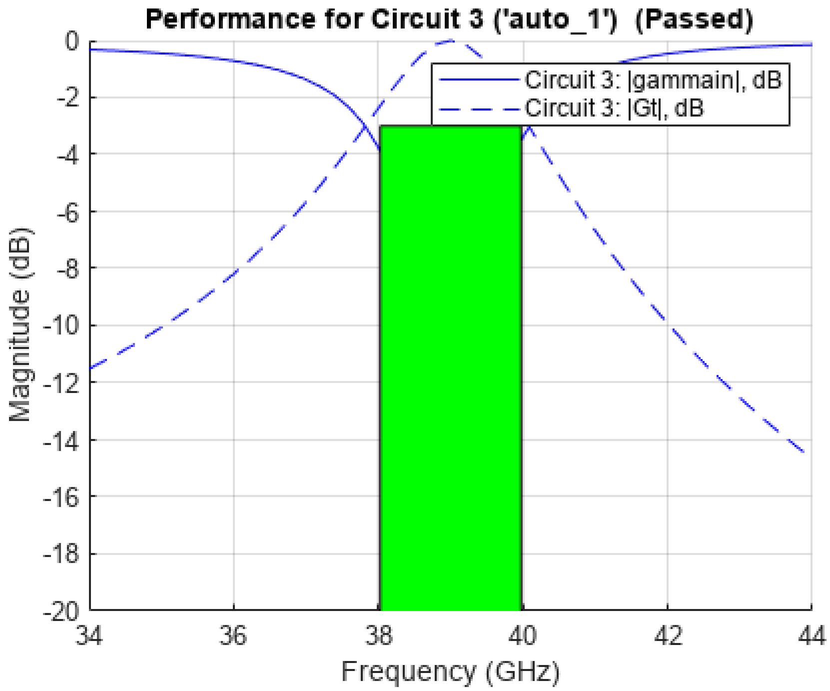
```
selectedCircuits = repmat(circuit,1,numport);
cIndex           = zeros(1,numport);
```

View the list of circuits generated.

```
for i = 1:numel(match_net)
    c = circuitDescriptions(match_net(i));
    % Perform a text search to choose the circuit with Shunt C-Series L-Shunt C topology
    Index = strcmp(c.component1Type,"Shunt C") & ...
        strcmp(c.component2Type,"Series L") & ...
        strcmp(c.component3Type,"Shunt C");
    if any(Index)
        % ShuntC-SeriesL-ShuntC topology
        cIndex(i)   = find(Index, 1, 'first');
        selectedCircuits(i) = match_net(i).Circuit(cIndex(i));
    else
        % Best available matchingnetwork
        selectedCircuits(i) = match_net(i).Circuit(1);
    end
    selectedCircuits(i).Name = "N"+i;
end
```

To view the performance of a selected matching network circuit, use `rfplot`. For instance, to plot the performance of the first matching network for the circuit with Shunt C-Series L-Shunt C topology type this command.

```
rfplot(match_net(1),freq,cIndex(1));
```



Add Matching Network Circuits to 16-Port Network

Create Circuit Object

Create a circuit object and an n-port object for the 16-port network.

```
ckt          = circuit('patchArray');
array_net    = nport(Sparam_array);
```

In this example, number of circuit nodes are shown as 17, as nodes 1 through 16 will be used for adding the matching networks.

```
cktnodes     = (1+numport):(numport+numport);
```

Add the n-port object to circuit object.

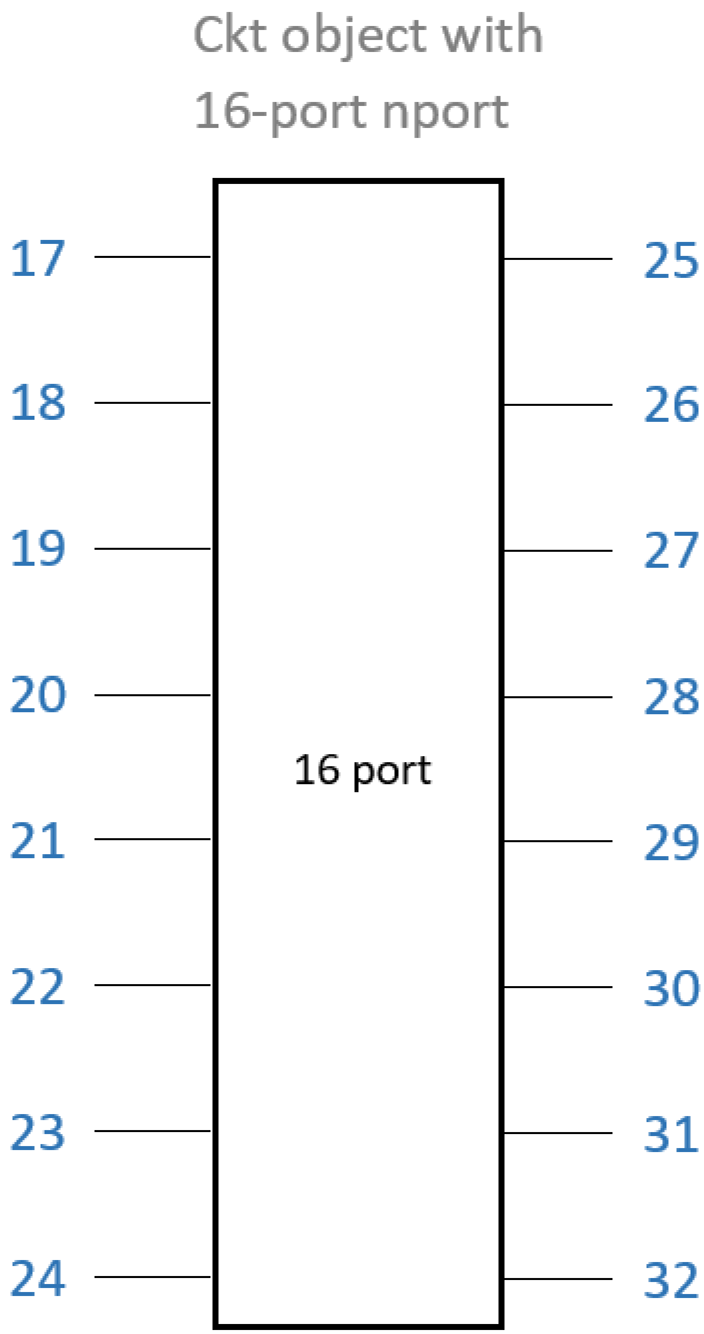
```
add(ckt, cktnodes, array_net);
```

View parent nodes of the 16-port network.

```
disp(array_net)
```

```
  nport: N-port element

    NetworkData: [1×1 sparameters]
           Name: 'Sparams'
       NumPorts: 16
      Terminals: {1×32 cell}
    ParentNodes: [17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 0 0 0 0 0 0 0 0 0 0 0 0 0 … ]
     ParentPath: 'patchArray'
```

An illustration of the circuit object with 16-port n-port is provided.



Initialize the ports.

```
ports = cell(1,numport);
```

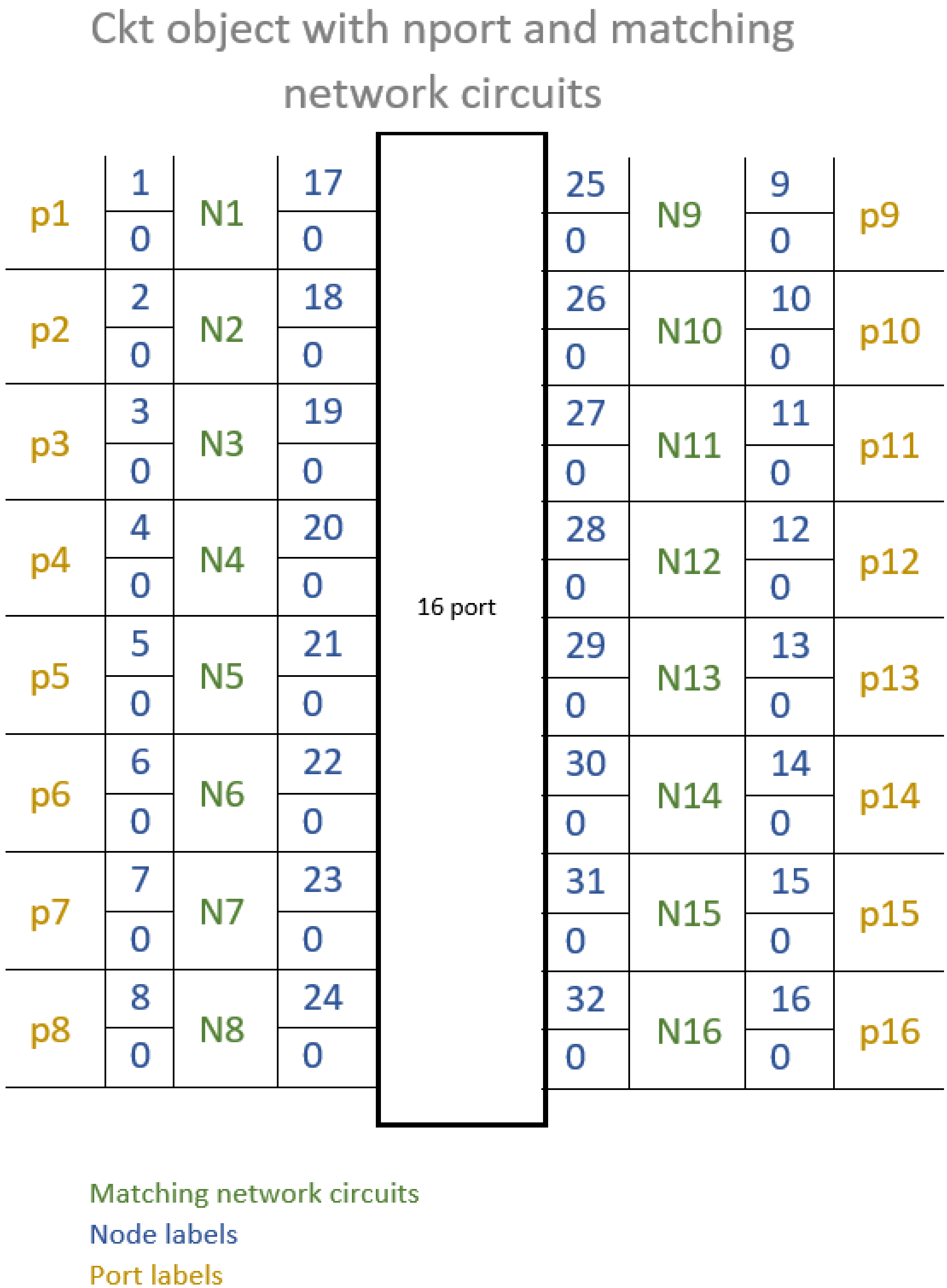
Add each matching network circuit to its corresponding port one at a time. Port numbers for corresponding matching network circuit are also generated.

```
for i=1:length(selectedCircuits)
    add(ckt, [i, 0, i+numport, 0], selectedCircuits(i), ...
        {'p1+', 'p1-', 'p2+', 'p2-'});
    ports{i} = [i, 0];
end
% ports = arrayfun(@(x) [x 0],1:10,'UniformOutput',false);
```

Use the `setports` function to define the ports for each of the circuits.

```
setports(ckt,ports{:});
```

An illustration of the circuit object with n-port and matching network circuits are provided.


Ckt object with nport and matching network circuits
p1
1
0
N1
17
0
p2
2
0
N2
18
0
p3
3
0
N3
19
0
p4
4
0
N4
20
0
p5
5
0
N5
21
0
p6
6
0
N6
22
0
p7
7
0
N7
23
0
p8
8
0
N8
24
0
16 port
25
0
N9
9
0
p9
26
0
N10
10
0
p10
27
0
N11
11
0
p11
28
0
N12
12
0
p12
29
0
N13
13
0
p13
30
0
N14
14
0
p14
31
0
N15
15
0
p15
32
0
N16
16
0
p16
Matching network circuits
Node labels
Port labels

Generate and Plot S-Parameters

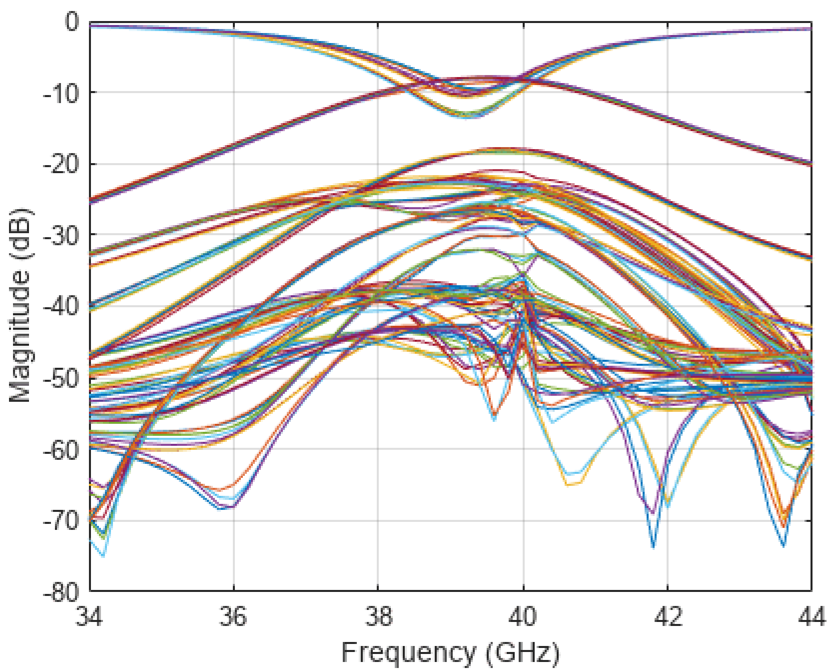
Generate and plot the S-Parameters of the passive 16-port matching network.

```
Sparam = sparameters(ckt, freq);
```

Plot Frequency Responses

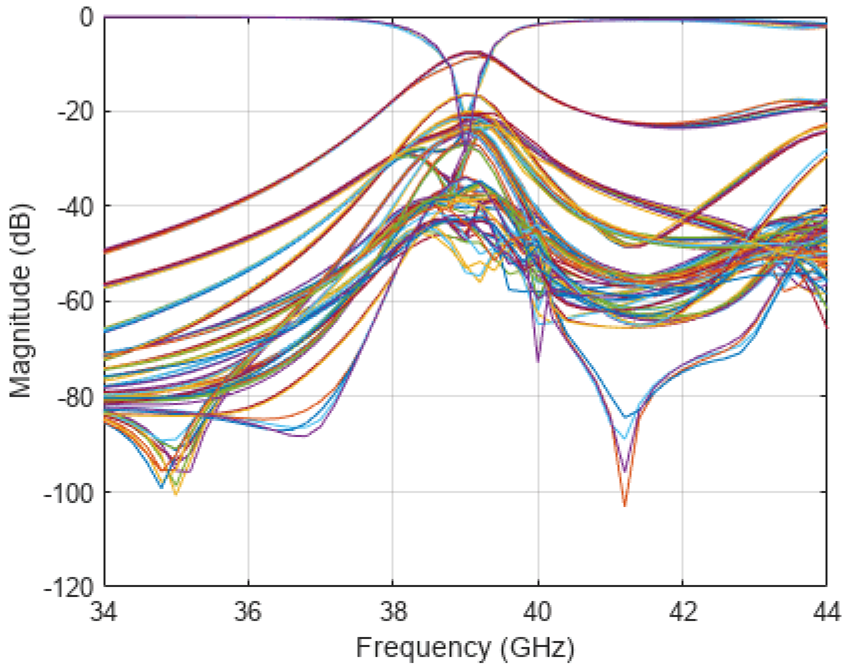
Plot the frequency response of the 16-port network before matching.

```
figure; rfplot(s_params_circ_array); legend off
```



Plot the frequency response of the 16-port network after matching.

```
figure; rfplot(Sparam); legend off
```



See Also

More About

- "Design Two-Stage Low Noise Amplifier Using Microstrip Transmission Line Matching Network" on page 6-219

Frequency Sweep in RF Budget Analysis

This example shows how to sweep through frequency-dependent properties of the elements in an RF Budget Analysis.

Use the `nport` and `amplifier` objects to specify the 2-port RF elements in the design and build an RF budget element by cascading the elements together into an RF system with `rfbudget`.

Building Elements of RF Budget Cascade

Build and parameterize each of the 2-port RF elements.

```
f1 = nport('RFBudget_RF.s2p','RFBandpassFilter');
a1 = amplifier('Name','RFAmplifier', ...
    'Gain',11.53, ...
    'NF',1.53, ...
    'OIP3',35);
```

Use `rfbudget` object to cascade the elements with input frequency 2.1 GHz, input power -30 dBm, and input bandwidth 45 MHz. This example cascades a filter and an amplifier.

```
b = rfbudget('Elements',[f1 a1], ...
    'InputFrequency',2.1e9, ...
    'AvailableInputPower',-30, ...
    'SignalBandwidth',45e6);
```

Read Frequency-Dependent Noise Figure

Read frequency-dependent Noise Figure (NF) values of the amplifier from the data-sheet. A similar approach can be followed if the Output third-order intercept (OIP3) or Gain is frequency-dependent.

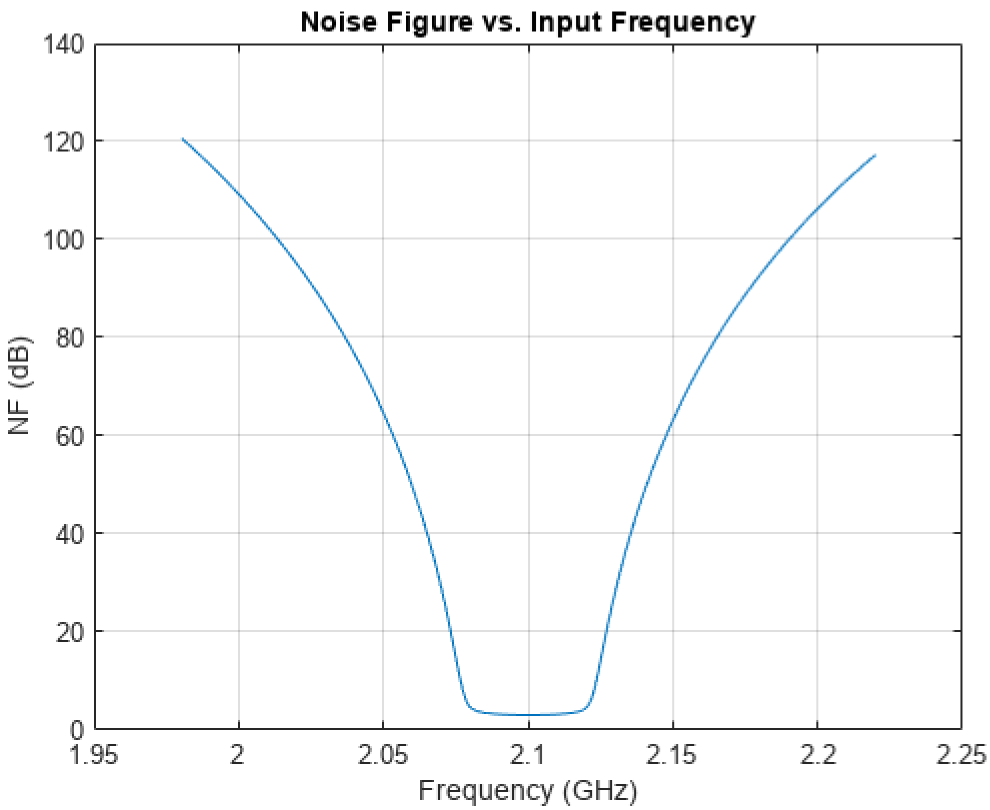
```
% Inputs from the data-sheet
freq_datasheet = [1.98;1.99;2.0;2.01;2.02;2.03;2.04;2.05;2.06;2.07;2.08;....
    2.09;2.10].*1e9;

NF_datasheet = [1.0000;1.0442;1.0883;1.1325;1.1767;1.2208;1.2650;1.3092;...
    1.3533;1.3975;1.4417;1.4858;1.5300];

% Interpolate the amplifier NF data based on existing filter frequencies
Freq = f1.NetworkData.Frequencies;
RFAmplifier_NF = interp1(freq_datasheet,NF_datasheet,Freq);
```

Plot RF Budget Results Versus Input Frequency

Loop over the desired frequencies, by setting NF of the RF Amplifier element in the `rfbudget` object.

```
TotalNF = zeros(size(Freq));
for i = 1:numel(Freq)
    b.InputFrequency = Freq(i);

    % Adjust frequency-dependent NF of the RF Amplifier
    elems(2).NF = RFAmplifier_NF(i);

    % Compute NF of the cascade
    TotalNF(i) = b.NF(end);
```

```
end
plot(Freq/1e9,TotalNF)
grid on;
xlabel('Frequency (GHz)')
ylabel('NF (dB)')
title('Noise Figure vs. Input Frequency')
```



See Also

More About

- "RF Budget Harmonic Balance Analysis of Low-IF Receiver, IP2 and NF" on page 6-226
- "Visualizing RF Budget Analysis over Bandwidth" on page 6-16

Using Rational Object to Fit S-Parameters

This example shows how to use the rational object to create a rational fit to S-parameter data, and the various properties and methods that are included in the rational object.

Create Rational Object

Read in the `sparameters`, and create the rational object from them. The rational function automatically fits all entries of the S-parameter matrices.

```
S = sparameters('sawfilter.s2p')
```

```
S = 
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [334x1 double]
     Parameters: [2x2x334 double]
      Impedance: 50

  rfparam(obj,i,j) returns S-parameter Sij
```

```
r = rational(S)
```

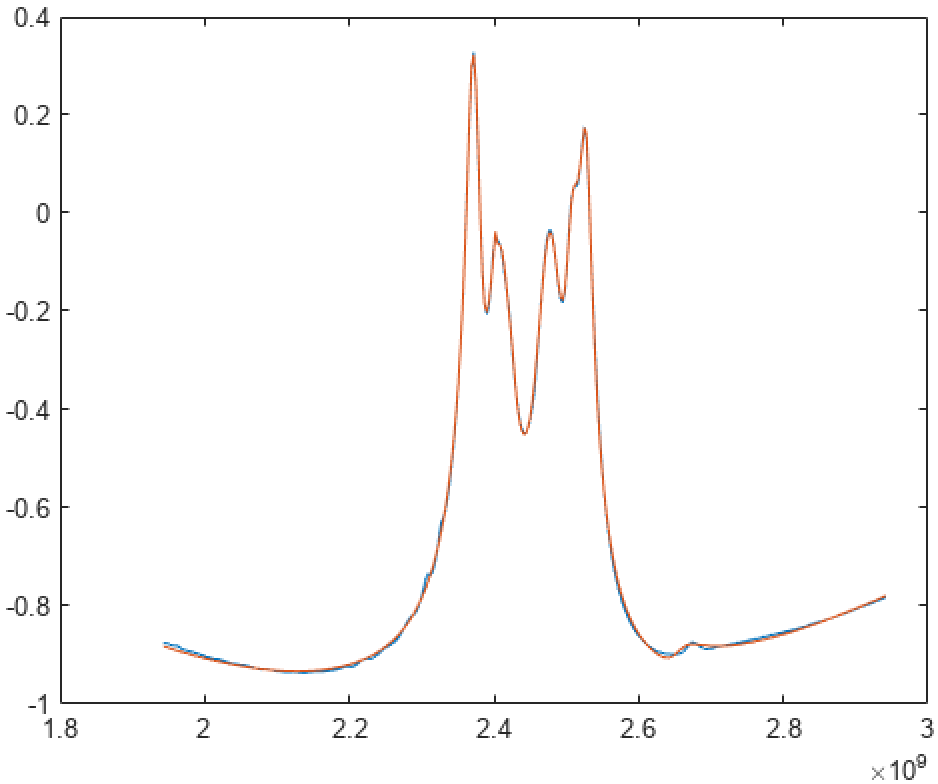
```
r = 
  rational with properties:

      NumPorts: 2
      NumPoles: 24
         Poles: [24x1 double]
      Residues: [2x2x24 double]
    DirectTerm: [2x2 double]
         ErrDB: -40.9658
```

With the default settings on this example, the rational function achieves an accuracy of about -26 dB, using 30 poles. By construction, the rational object is causal, with a non-zero direct term.

Compare Fit with Original Data

Generate the frequency response from the rational object, and compare one of the entries with the original data.

```
resp = freqresp(r, S.Frequencies);
plot(S.Frequencies, real(rfparam(S, 1, 1)), ...
    S.Frequencies, real(squeeze(resp(1,1,:))))
```



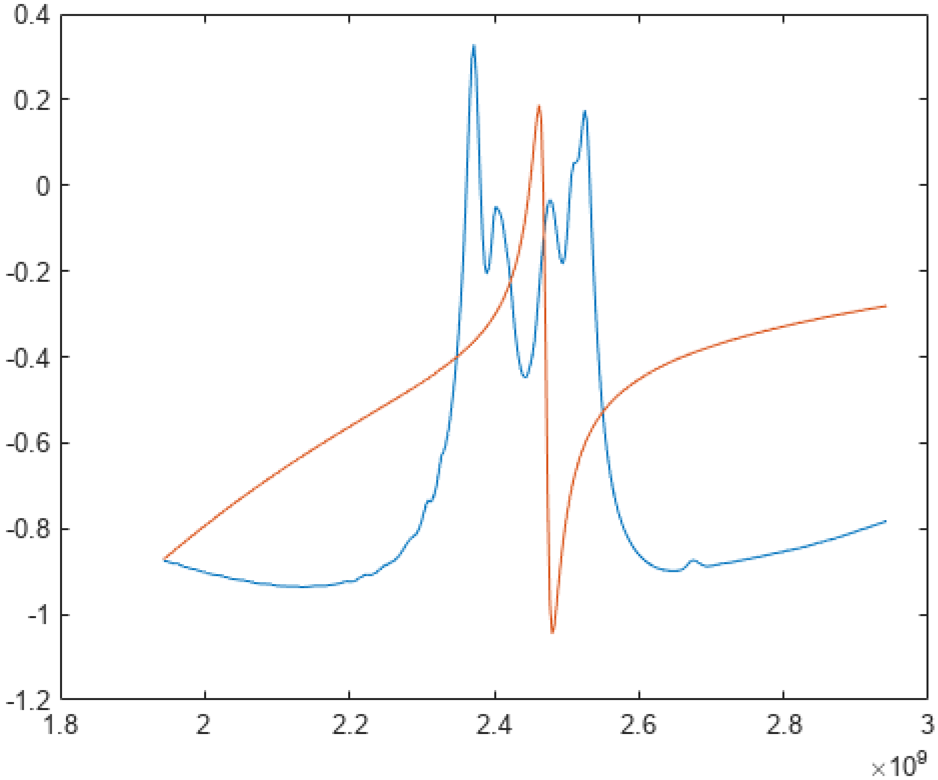
Limit Number of Poles

Redo the fit, limiting the number of poles to a maximum of 5. The rational object may use fewer poles than specified. Notice that the quality of the fit is degraded as opposed to the original 30-pole fit.

```
r5 = rational(S, 'MaxPoles', 5)
```

```
r5 = 
  rational with properties:

      NumPorts: 2
      NumPoles: 4
         Poles: [4x1 double]
      Residues: [2x2x4 double]
    DirectTerm: [2x2 double]
         ErrDB: -1.7376
```

```
resp5 = freqresp(r5, S.Frequencies);
plot(S.Frequencies, real(rfparam(S, 1, 1)), ...
    S.Frequencies, real(squeeze(resp5(1,1,:))))
```



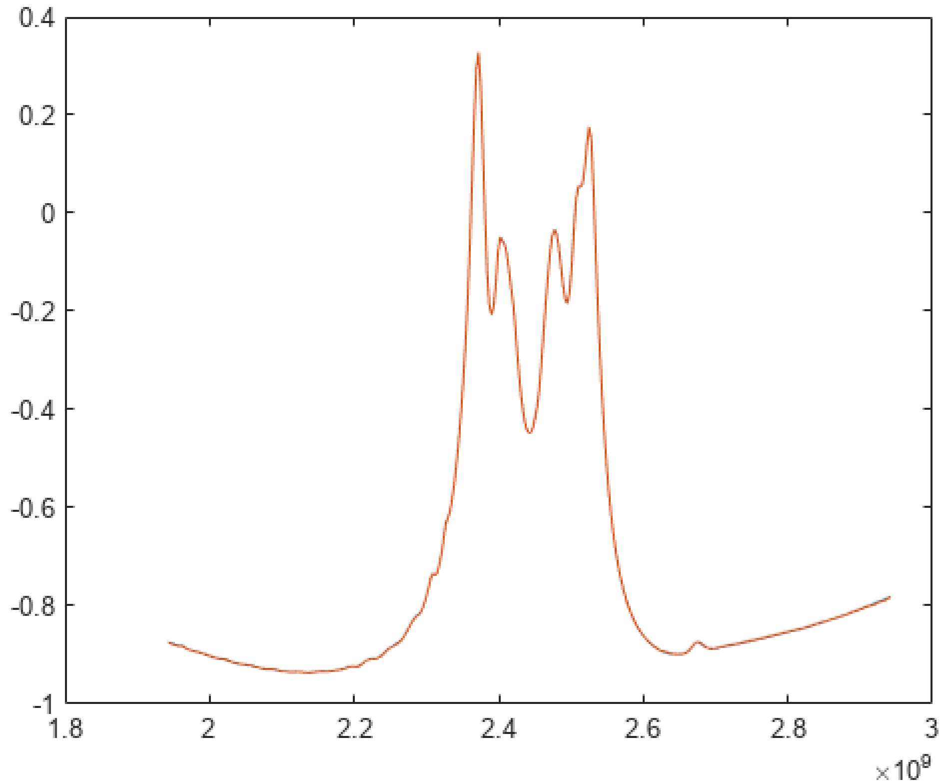
Tighten Target Accuracy

Redo the fit, asking for a tighter tolerance (-60 dB), Notice that the fit is significantly improved, particularly in the stopbands of the SAW filter.

```
rgood = rational(S, -60)
```

```
rgood = 
  rational with properties:

      NumPorts: 2
      NumPoles: 188
         Poles: [188x1 double]
      Residues: [2x2x188 double]
    DirectTerm: [2x2 double]
         ErrDB: -53.6396
```

```
respgood = freqresp(rgood, S.Frequencies);
plot(S.Frequencies, real(rfparam(S, 1, 1)), ...
    S.Frequencies, real(squeeze(respgood(1,1,:))))
```



See Also

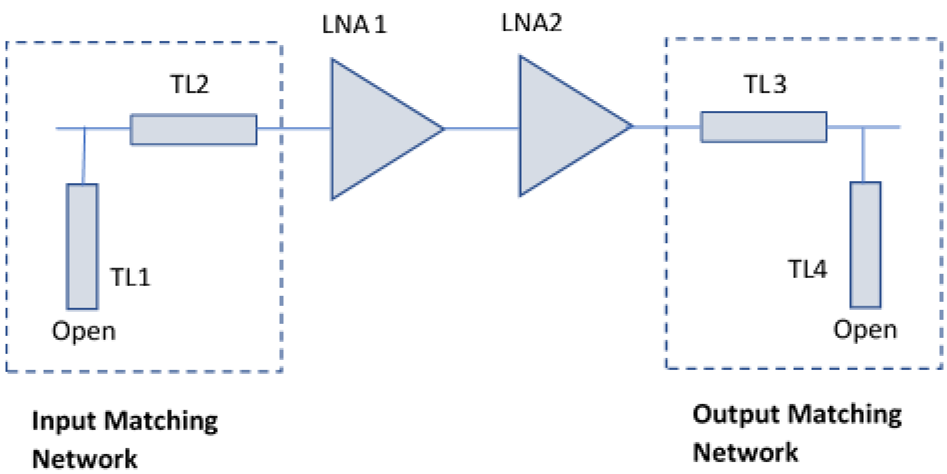
More About

- "Passivity: Test, Visualize, and Enforce Passivity of Rational Fit Output" on page 6-187
- "Using 'NPoles' Parameter With rationalfit" on page 6-73
- "Using 'Weight' Parameter With rationalfit" on page 6-77

Design Two-Stage Low Noise Amplifier Using Microstrip Transmission Line Matching Network

This example shows how to use the RF Toolbox™ microstrip transmission line element to design two-stage low noise amplifier (LNA) for wireless local area network (WLAN) with an input and the output matching network (MNW) to maximize the power delivered through a 50-ohm load and the system.

Designing an input and output MNW is an important part of the amplifier design. The amplifier in this example has high gain and low noise. To minimize parasitic effect, this example uses the microstrip transmission line MNW with a single stub.



Define Microstrip Transmission Line Parameters

The microstrip transmission line parameters are chosen as follows.

- Physical Height of conductor or dielectric thickness — 1.524 mm
- Relative permittivity of dielectric — 3.48
- Loss angle tangent of dielectric — 0.0037
- Physical thickness of microstrip transmission line — 3.5 um

Design Input Matching Network Using Microstrip Transmission Line

The input matching network consists of one shunt stub and one series microstrip transmission line.

Create an input shunt stub microstrip transmission line with the physical length of 8.9 mm.

```
TL1 = txlineMicrostrip('Width',3.41730e-3,'Height',1.524e-3,'EpsilonR',3.48,'LossTangent',0.0037
    'LineLength',8.9e-3,'Thickness',0.0035e-3,'StubMode','Shunt','Termination','Open');
```

Create an input series microstrip transmission line with the physical length of 14.7 mm.

```
TL2 = txlineMicrostrip('Width',3.41730e-3,'Height',1.524e-3,'EpsilonR',3.48, 'LossTangent',0.0037
    'LineLength',14.7e-3,'Thickness',0.0035e-3);
```

Create and Extract Amplifier Object

Create and extract an amplifier object from the frequency dependent S-parameter data available in the specified file.

```
amp1 = nport('f551432p.s2p');
```

Define the frequency range.

```
freq = 2e9:10e6:3e9;
```

Create a two-stage amplifier and plot its S-parameter.

```
casamp = circuit([amp1,clone(amp1)],'amplifiers'); % amplifier ciruit without MNW.
```

Plot the S-Parameter over the frequency range from 2 – 3 GHz.

```
S2 = sparameters(casamp,freq);
```

Design Output Matching Network Using Microstrip Transmission Line

The output matching network consists of one shunt stub and one series microstrip transmission line.

Create an output series microstrip transmission line with the physical length of 22.47 mm.

```
TL3 = txlineMicrostrip('Width',3.41730e-3,'Height',1.524e-3,'EpsilonR',3.48, 'LossTangent',0.0037
    'LineLength',22.47e-3,'Thickness',0.0035e-3);
```

Create an output shunt stub microstrip transmission line with the physical length of 5.66 mm.

```
TL4 = txlineMicrostrip('Width',3.41730e-3, 'Height',1.524e-3,'EpsilonR',3.48, 'LossTangent',0.003
    'LineLength',5.66e-3,'Thickness',0.0035e-3,'StubMode','Shunt','Termination','Open');
```

Plot Input Reflection Coefficients of Two-Stage LNA

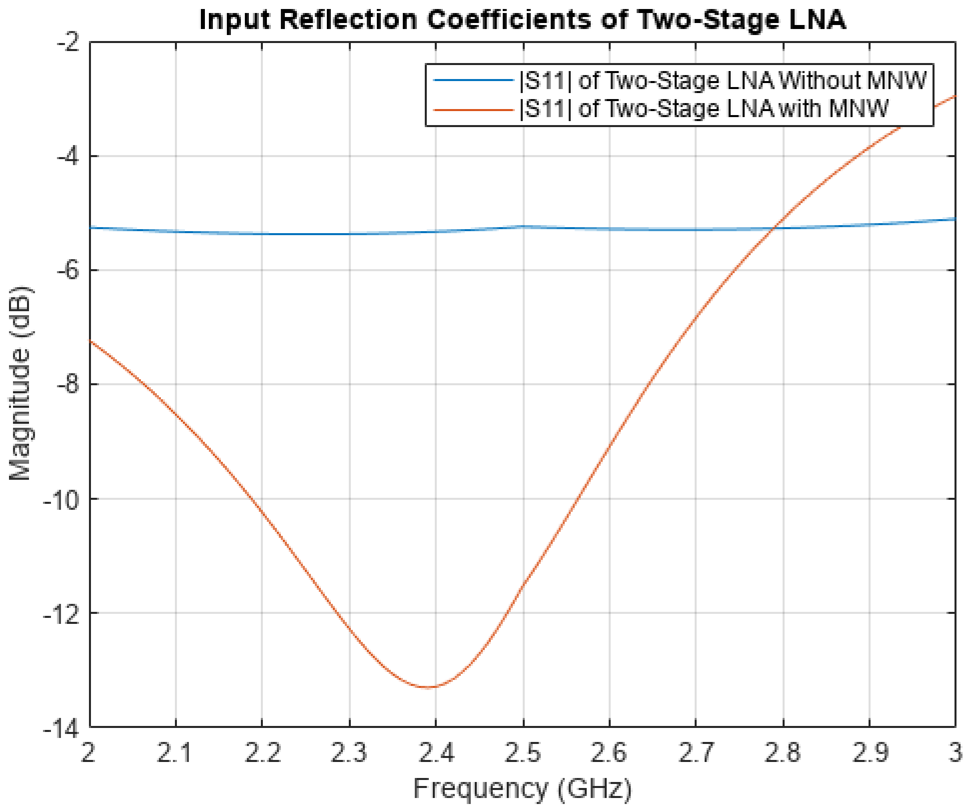
To verify the simultaneous conjugate match at the input of the amplifier, plot the input reflection coefficients in dB for the amplifier circuit with and without a matching network.

Cascade the circuit elements by adding the input and the output MNW to the two-stage amplifier.

```
c = circuit([TL1, TL2,clone(amp1),clone(amp1),TL3, TL4]); % two-stage LNA with MNW
```

Plot the S-parameters and analyze the amplifier with and without the matching networks over the frequency range of 2.4 – 2.5 GHz.

```
figure
S3 = sparameters(c,freq);
rfplot(S2,1,1)
hold on;
rfplot(S3,1,1)
legend('|S11| of Two-Stage LNA Without MNW','|S11| of Two-Stage LNA with MNW');
title('Input Reflection Coefficients of Two-Stage LNA');
grid on;
```

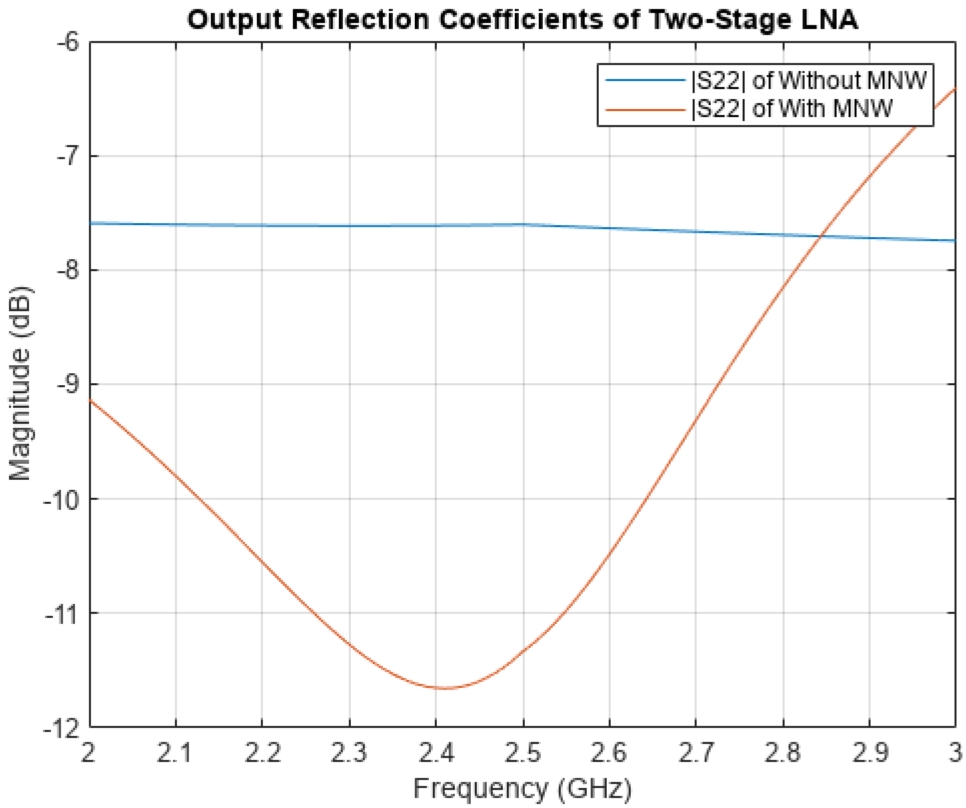


The calculated input return loss for the two-stage LNA with the input MNW is around 13 dB.

Plot Output Reflection Coefficients of Two-Stage LNA

To verify the simultaneous conjugate match at the output of the amplifier, plot output reflection coefficients in dB for both the two-stage LNA with and without a MNW.

```
figure
rfplot(S2,2,2)
hold on;
rfplot(S3,2,2)
legend('|S22| of Without MNW','|S22| of With MNW');
title('Output Reflection Coefficients of Two-Stage LNA');
grid on;
```

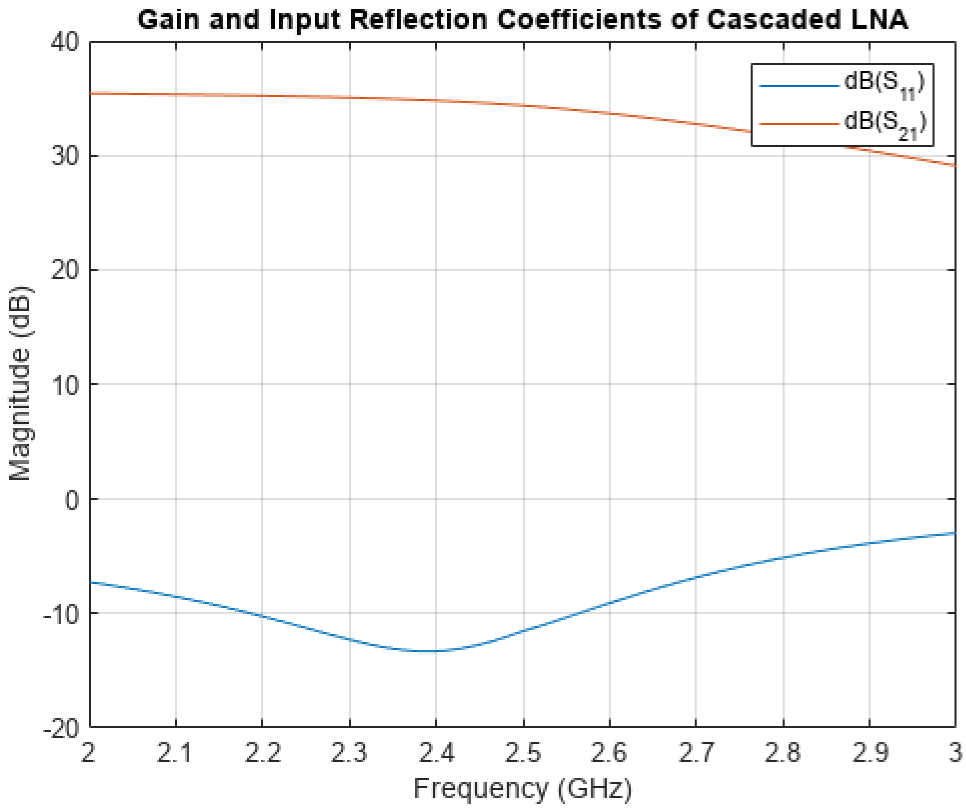


The calculated output return loss for the two-stage LNA with the output MNW is aroung 11 dB.

Plot Gain and Input Reflection Coefficients of Cascaded LNA

To verify the simultaneous conjugate match at the input and output of the amplifier, plot the input reflection coefficient and the gain parameters in dB for the two-stage LNA with the MNW.

```
figure;
rfplot(S3,1,1)
hold on;
rfplot(S3,2,1)
title('Gain and Input Reflection Coefficients of Cascaded LNA');
grid on;
```

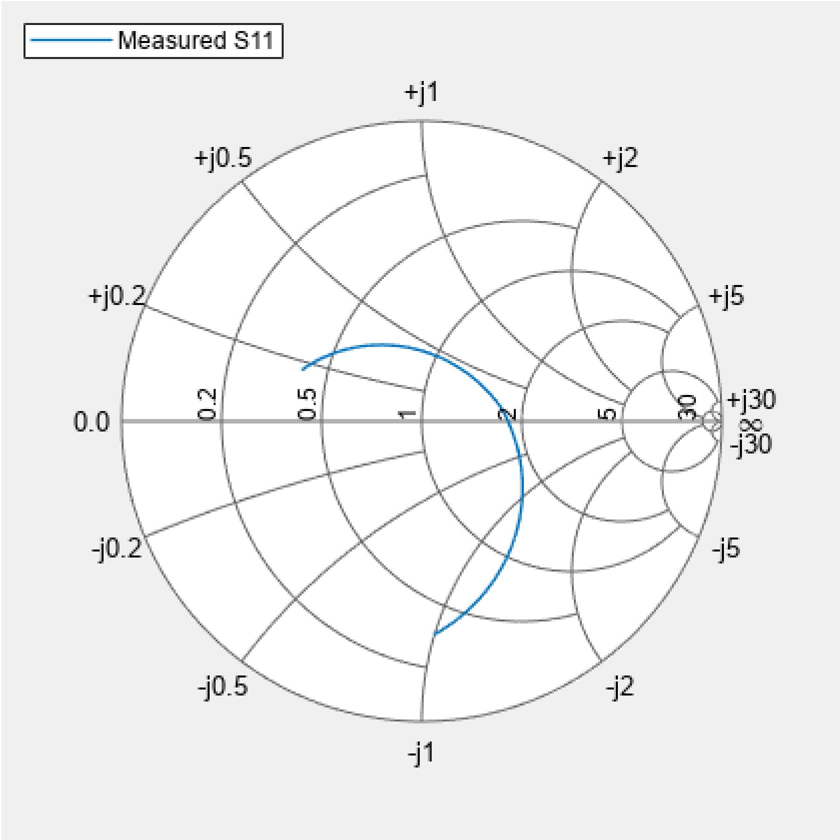


The calculated amplifier gain, S21 is 34.5 dB, and the input reflection coefficient, S11 is around 13 dB.

Calculate and Plot Complex Load and Source Reflection Coefficients

Calculate and plot all the complex load and source reflection coefficients for simultaneous conjugate match at all measured frequency data points that are unconditionally stable. These reflection coefficients are measured at the amplifier interfaces.

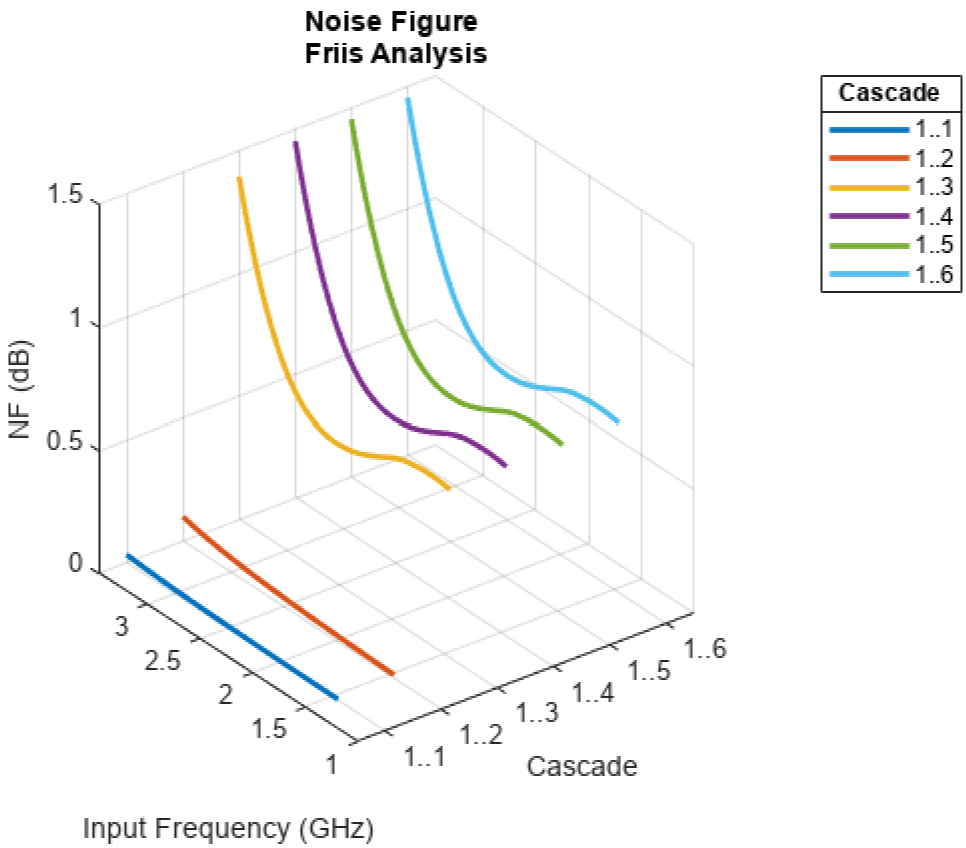
```
figure
smithplot(S3,1,1,'LegendLabels','Measured S11')
```



Calculate Ampifier Noise Figure

Use an `rfbudget` object to calculate the amplifier noise figure.

```
b = rfbudget( ...
    'Elements',[TL1 TL2 amp1 clone(amp1) TL3 TL4], ...
    'InputFrequency',2.45e9, ...
    'AvailableInputPower',0, ...
    'SignalBandwidth',2e9, ...
    'Solver','Friis', ...
    'AutoUpdate',1);
rfplot(b,'NF')
```



The amplifier noise figure is calculated as 0.7 dB.

Reference

[1] Maruddani, B, M Ma'sum, E Sandi, Y Taryana, T Daniati, and W Dara. "Design of Two Stage Low Noise Amplifier at 2.4 - 2.5 GHz Frequency Using Microstrip Line Matching Network Method." *Journal of Physics: Conference Series* 1402 (December 2019): 044031.

See Also

More About

- "Analysis of Coplanar Waveguide Transmission Line in X-Band Application" on page 6-232
- "Design Matching Networks for Passive Multiport Network" on page 6-205
- "Design Broadband Matching Networks for Antennas" on page 6-140
- "Impedance Matching of Small Monopole Antenna" on page 6-167

RF Budget Harmonic Balance Analysis of Low-IF Receiver, IP2 and NF

This example shows how to use the rfbudget object's harmonic balance solver to analyze a low-IF (intermediate frequency) receiver RF budget for second-order intercept point (IP2), the second-order intercept point and to compute a more accurate noise figure (NF) that correctly accounts for system nonlinearity and noise-folding.

Use `amplifier` and `modulator` objects to construct the 2-port RF elements in a low-IF receiver design, along with their output second-order intercept point (OIP2) specifications. You can turn off the default ideal image reject and channel select filtering in the modulator with the `ImageReject` and `ChannelSelect` logical name-value pairs.

Compute RF budget results by cascading the elements together into an RF system with `rfbudget`. The rfbudget object enables design exploration and visualization at the MATLAB command-line. It also enables automatic RF Blockset model and measurement testbench generation.

```
a1 = amplifier('Name','RFAmplifier', ...
    'Gain',11.53, ...
    'NF',1.53, ...
    'OIP2',35);

d = modulator('Name','Demodulator', ...
    'Gain',-6, ...
    'NF',4, ...
    'OIP2',50, ...
    'LO',2.03e9, ...
    'ConverterType','Down', ...
    'ImageReject',false, ...
    'ChannelSelect',false);

a2 = amplifier('Name','IFAmplifier', ...
    'Gain',30, ...
    'NF',8, ...
    'OIP2',37);

b = rfbudget('Elements',[a1 d a2], ...
    'InputFrequency',2.1e9, ...
    'AvailableInputPower',-30, ...
    'SignalBandwidth',45e6)
```

```
b = 
  rfbudget with properties:

               Elements: [1x3 rf.internal.rfbudget.RFElement]
         InputFrequency: 2.1 GHz
    AvailableInputPower: -30 dBm
        SignalBandwidth:  45 MHz
                 Solver: Friis
             AutoUpdate: true

   Analysis Results
        OutputFrequency: (GHz) [   2.1    0.07    0.07]
            OutputPower: (dBm) [-18.47  -24.47    5.53]
         TransducerGain: (dB)  [ 11.53    5.53   35.53]
```

```
                NF: (dB)  [  1.53   1.843  4.793]
              IIP2: (dBm) []
              OIP2: (dBm) []
              IIP3: (dBm) [   Inf     Inf    Inf]
              OIP3: (dBm) [   Inf     Inf    Inf]
               SNR: (dB)  [ 65.91    65.6  62.65]
```

Why are OIP2 and IIP2 Empty in the Results?

The default `Solver` property of the rfbudget object is 'Friis', an equivalent baseband approximation which is unable to compute IP2. To see the IP2 results, you can set the `Solver` property of the budget object to 'HarmonicBalance'. This performs nonlinear circuit analysis to compute the steady-state operating point, from which it is possible to compute IP2.

You can also select the 'HarmonicBalance' solver at rfbudget construction time by passing in a `Solver` name-value pair after the other positional or name-value pair arguments, e.g.

```
b = rfbudget([a1 d a2],2.1e9,-30,45e6,'Solver','HarmonicBalance')
```

In general, the 'HarmonicBalance' solver is not as fast as the 'Friis' solver and does not compute noise figure (NF) or signal-to-noise ratio (SNR).

```
b.Solver = 'HarmonicBalance'
```

```
b = 
  rfbudget with properties:

             Elements: [1x3 rf.internal.rfbudget.RFElement]
       InputFrequency: 2.1 GHz
  AvailableInputPower: -30 dBm
      SignalBandwidth:  45 MHz
               Solver: HarmonicBalance
              WaitBar: true
           AutoUpdate: true

   Analysis Results
      OutputFrequency: (GHz) [   2.1    0.07    0.07]
          OutputPower: (dBm) [-18.47  -24.47    5.53]
       TransducerGain: (dB)  [ 11.53    5.53   35.53]
                   NF: (dB)  [  1.53     4.7   6.487]
                 IIP2: (dBm) [ 23.47   44.47  -4.581]
                 OIP2: (dBm) [    35      50   30.95]
                 IIP3: (dBm) [   Inf     Inf   19.45]
                 OIP3: (dBm) [   Inf     Inf   54.98]
                  SNR: (dB)  [ 65.91   62.74   60.96]
```

The rfbudget display above shows the results of the cascade computed by the 'HarmonicBalance' solver. Comparing them to the 'Friis' results, the vector properties showing the OutputPower and TransducerGain along the cascade match well.

As expected, the OIP2 and IIP2 properties have nonempty values. In addition, the output third-order intercept point (OIP3) and input third-order intercept point (IIP3) properties have changed. The 'Friis' solver is unable to capture the nonlinear bleeding through the IP2 properties of the cascade to affect the third-order intercept point. Mathematically, this happens because cascading two second-order polynomials results in a polynomial with a third-order term.

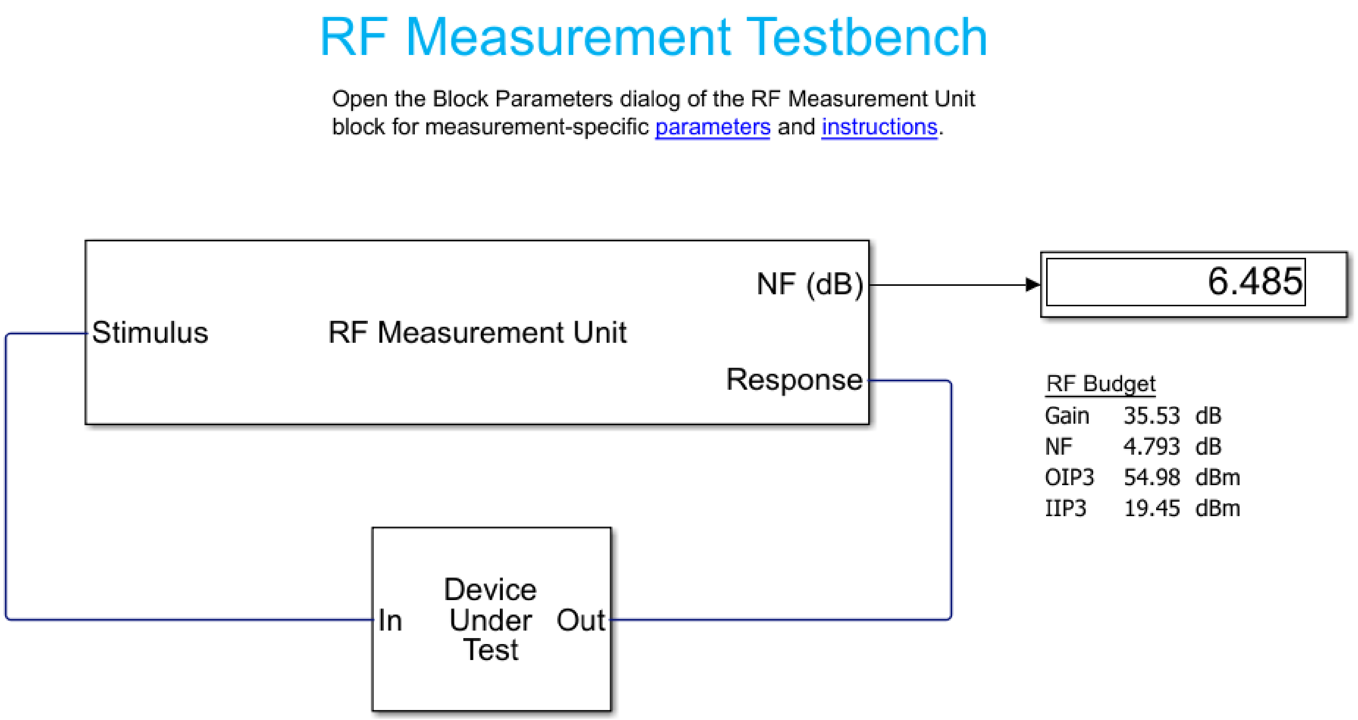
Similarly, the NF results of Harmonic Balance are different (and more accurate) than the Friis results because Harmonic Balance correctly captures the noise folding effects of nonlinearities.

Verifying HB Results Using RF Blockset Circuit Envelope Simulation

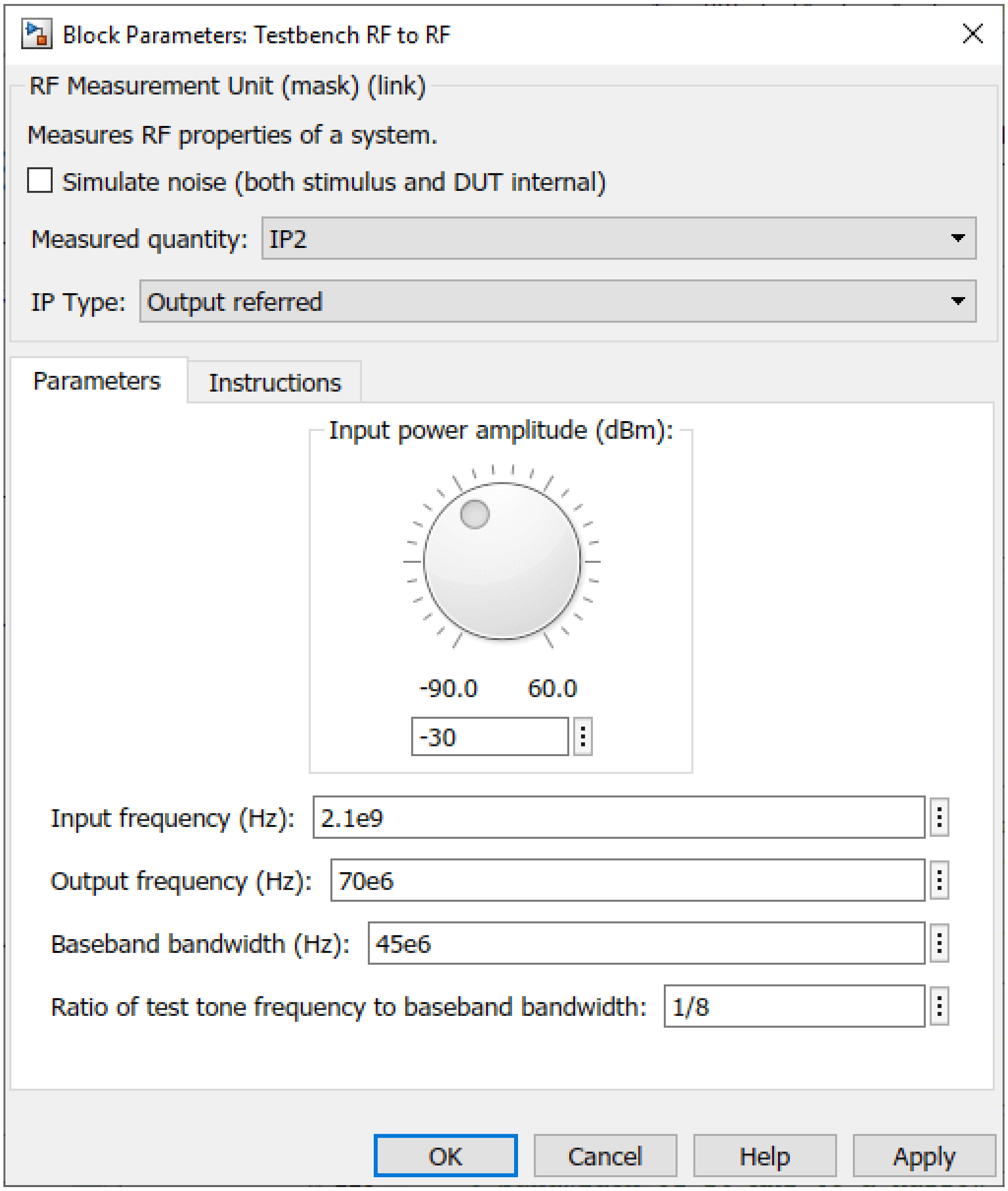
You can verify the harmonic balance NF, IP2 and IP3 results by exporting the budget to an RF Blockset testbench model using the following command:

```
exportTestbench(b)
```

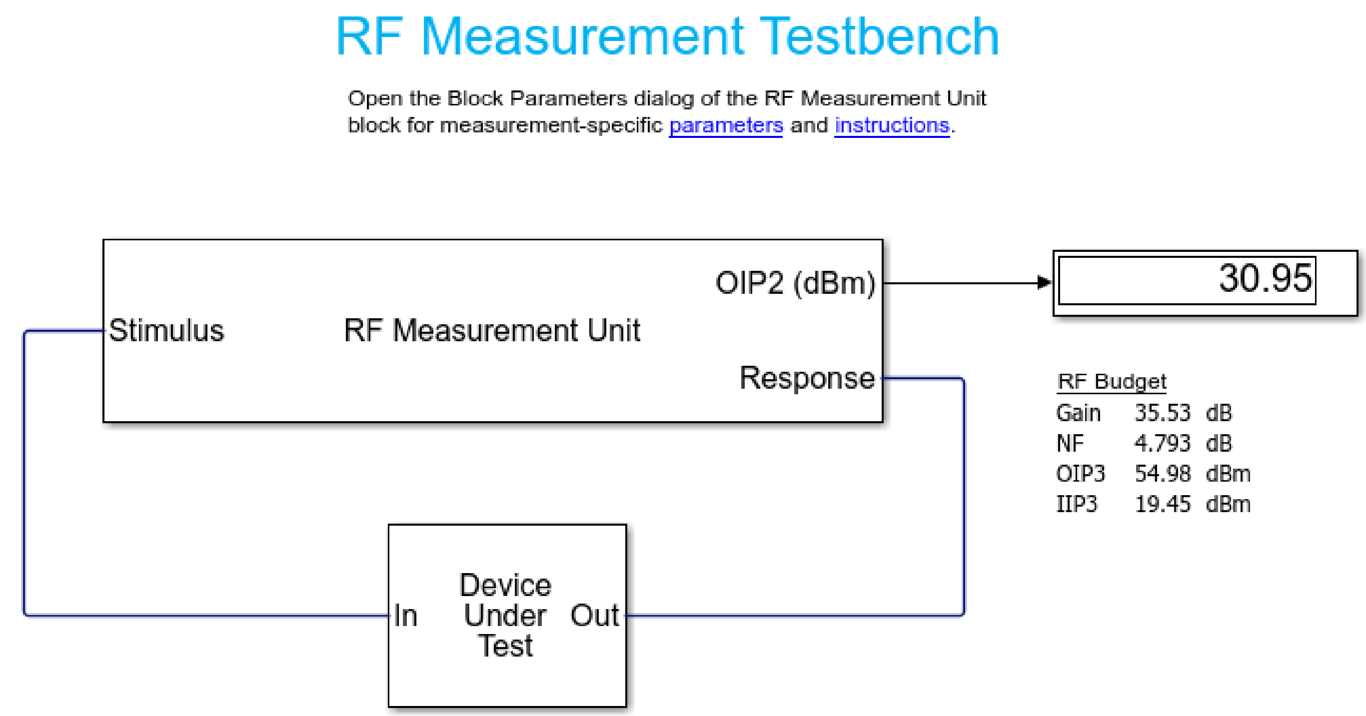
To verify NF, double-click on the RF Measurement Unit to open the mask, then select NF from the Measured quantity pulldown. Then run the model. This verifies the Harmonic Balance NF calculation.



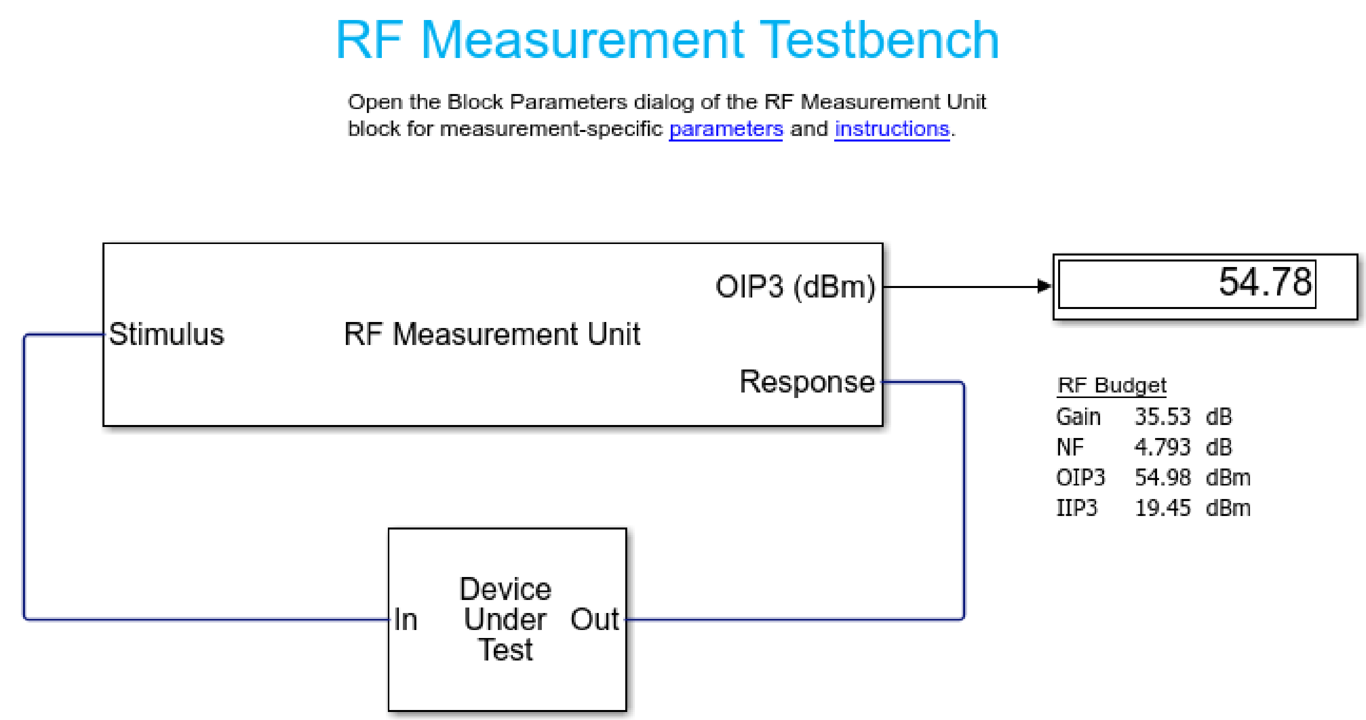
To verify IP2, double-click on the RF Measurement Unit to open its mask, then select IP2 from the Measured quantity pulldown.



Also uncheck the Simulate noise checkbox. Then run the model.

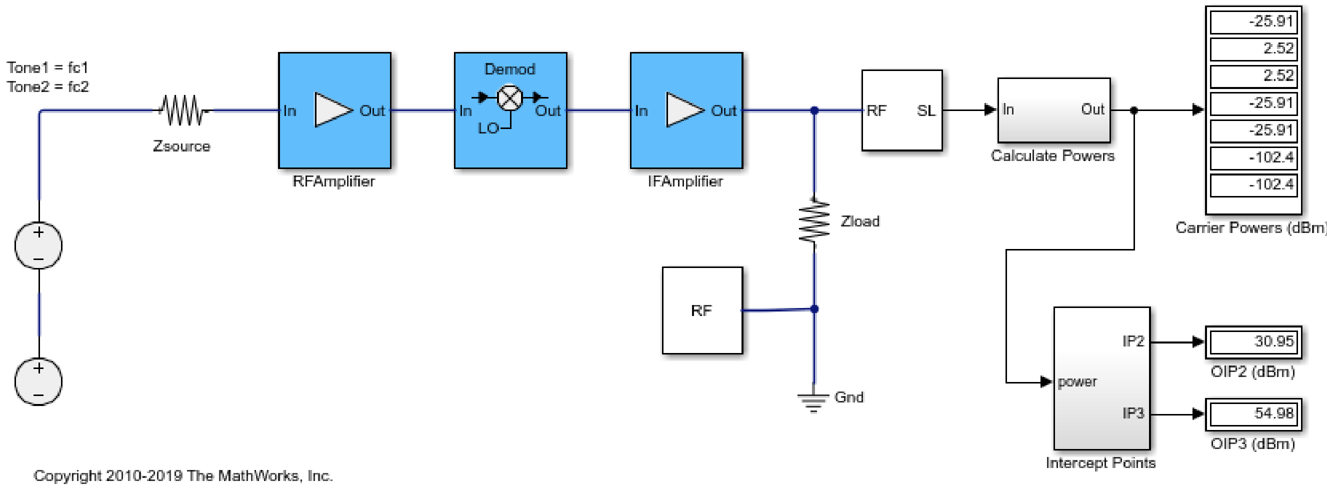


To verify IP3, select IP3 from the Measured quantity pulldown and run the model again.



Verifying HB results with RF Blockset Harmonic Balance

Rather than using the large machinery of circuit envelope and the RF Testbench, it is possible to build a simpler model that computes the IP2 and IP3 using two tones and harmonic balance. Open the model `oipHB.slx` found in the MATLAB/Examples folder. Simulate the model.



See Also

More About

- "Superheterodyne Receiver Using RF Budget Analyzer App" on page 6-2
- "Visualizing RF Budget Analysis over Bandwidth" on page 6-16
- "Frequency Sweep in RF Budget Analysis" on page 6-213

Analysis of Coplanar Waveguide Transmission Line in X-Band Application

This example shows how to analyze a coplanar waveguide (cpw) transmission line for X-band applications. CPW transmission line consists of a central metal strip separated by a narrow gap from two ground planes on either side. The dimensions of the center strip, the gap, the thickness, and permittivity of the dielectric substrate determine the characteristic impedance, group delay, and noise. The gap in the cpw is usually very small and supports electric fields primarily concentrated in the dielectric.

Define Parameters

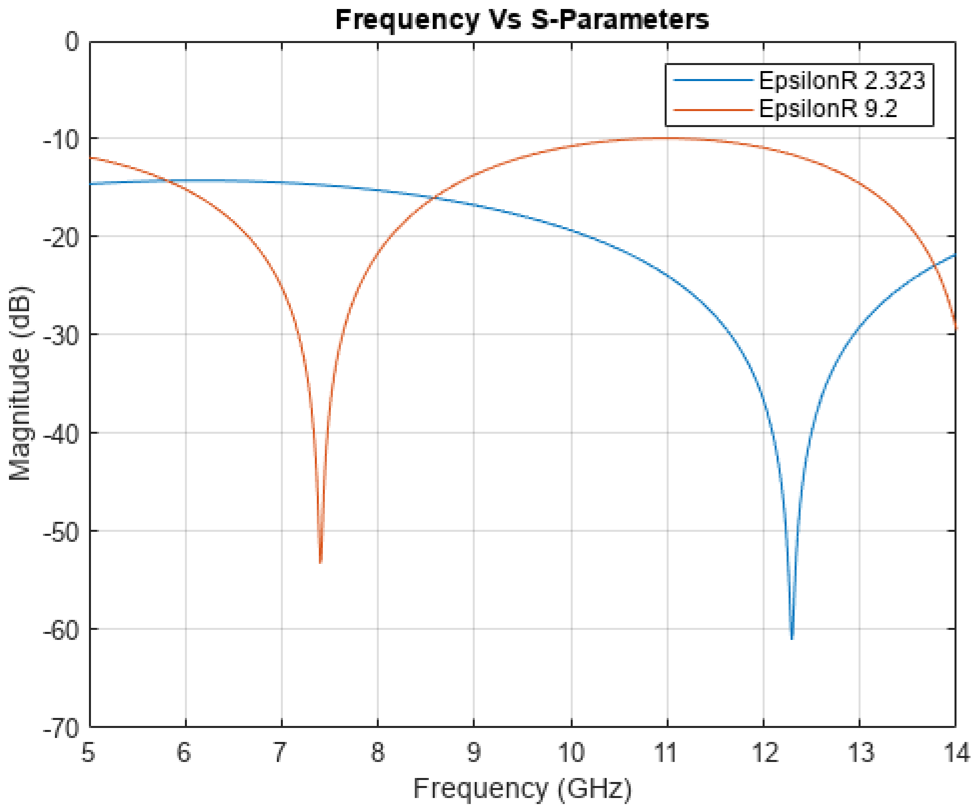
The cpw transmission line has 200 mm slot width, 1600 mm conductor width, 635 mm height, 0.005 loss tangent, and 17 um of thickness. This example uses two different dielectric constants to simulate the cpw transmission line. The dielectric constant values are 2.323 and 9.2.

```
cptxline1 = txlineCPW('EpsilonR',2.323,'SlotWidth',200e-6,'ConductorWidth',...
            1600e-6,'Height',635e-6,'LossTangent',0.005,'Thickness',17e-6);
cptxline2 = txlineCPW('EpsilonR',9.2,'SlotWidth',200e-6,'ConductorWidth',...
            1600e-6,'Height',635e-6,'LossTangent',0.005,'Thickness',17e-6);
% x band Frequency range 8 to 12GHz
freq = 5e9:10e6:14e9;
```

Plot Input Return Loss

The results for two different dielectric substrates indicates the impedance bandwidth increases with a lower dielectric constant. The measurement results are for a frequency range of 5 GHz to 14GHz, and magnitude of S11 < 10 dB.

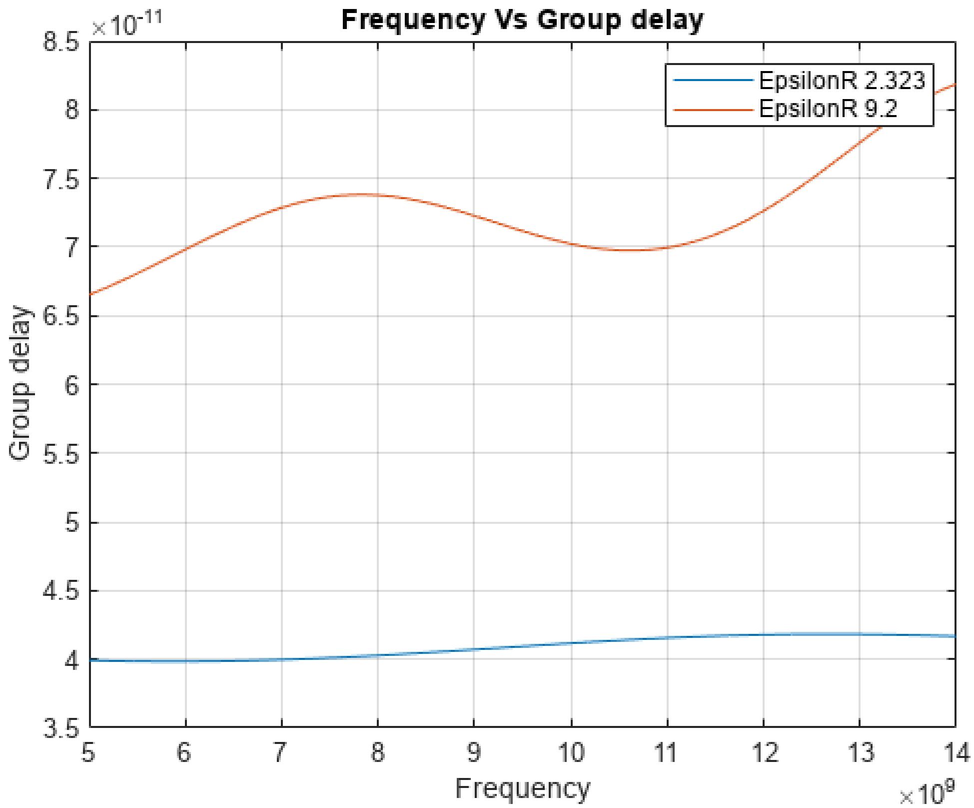
```
figure;
sp1 = sparameters(cptxline1,freq);
sp2 = sparameters(cptxline2,freq);
rfplot(sp1,1,1);hold on;
rfplot(sp2,1,1);
title('Frequency Vs S-Parameters');
legend('EpsilonR 2.323','EpsilonR 9.2');
grid on;
```



Group Delay

Group delay variations versus frequency is an essential factor when using phase modulation and high data rates. This impairment causes distortion and degradation in wideband applications. In a cpw transmission line the group delay increases with increase in the frequency for both dielectric substrates.

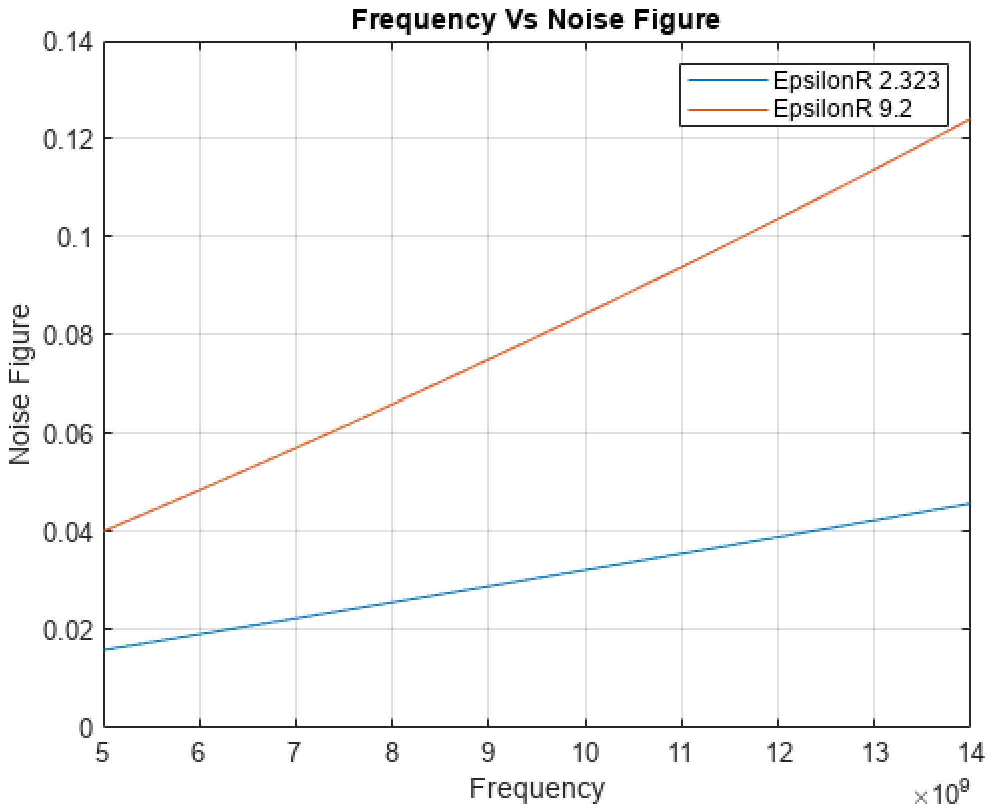
```
gd1 = groupdelay(cptxline1,freq,'Impedance',50);
gd2 = groupdelay(cptxline2,freq,'Impedance',50);
figure;plot(freq,gd1);hold on;
plot(freq,gd2);
title('Frequency Vs Group delay');
legend('EpsilonR 2.323','EpsilonR 9.2');
xlabel('Frequency');
ylabel('Group delay');
grid on;
```



Noise Figure

The noise is generated primarily within the input stages of the receiver system itself. Cascaded stages are not noisier than others. The noise generated at the input and amplified by the receiver's full gain amplifier greatly exceeds the noise generated further along the receiver chain. In the results using both lower and higher dielectric constant, noise figure increases with increasing frequency. The variation is very less over the frequency range when using a lower dielectric constant.

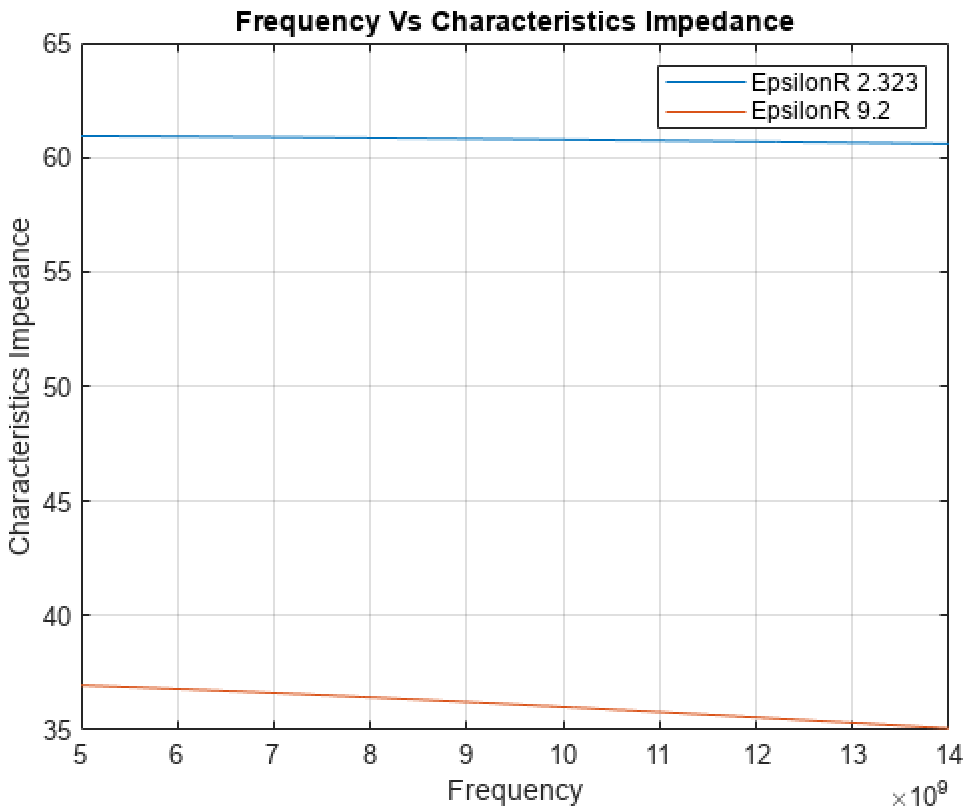
```
nf1 = noisefigure(cptxline1,freq);
nf2 = noisefigure(cptxline2,freq);
figure;plot(freq,nf1);hold on;
plot(freq,nf2);
title('Frequency Vs Noise Figure');
legend('EpsilonR 2.323','EpsilonR 9.2');
xlabel('Frequency');
ylabel('Noise Figure');
grid on;
```



Characteristic Impedance

Relative permittivity for a homogeneous dielectric affects the characteristic impedance of cpw transmission line. You can compute this approximately by using the electrical model of the cpw to clarify impedance behavior along the frequency band. Characteristic impedance determines the amount of power transfer and attenuation effect along the cpw transmission line. The characteristic impedance of a transmission line is usually written as Z0. In the simulation, the resulting characteristic impedance decreases with increasing frequency in both dielectric constants. With lower dielectric constant impedance value is below 50 ohms, with higher dielectric constant impedance value is above 50 ohms.

```
ChImp1 = getZ0(cptxline1,freq);
ChImp2 = getZ0(cptxline2,freq);
figure; plot(freq,ChImp1);hold on;
plot(freq,ChImp2);
title('Frequency Vs Characteristics Impedance');
xlabel('Frequency');
ylabel('Characteristics Impedance');
legend('EpsilonR 2.323','EpsilonR 9.2');
grid on;
```



Conclusion

In RF and microwave circuit design the dielectric permittivity of the substrate plays an important role and requires precise evaluation over a broad range of frequencies. With the above simulation you see that, lower dielectric constant gives wider bandwidth, lower noise figure, and lower group delay.

Reference:

Sova, M., and I. Bogdan. "Coplanar Waveguide Resonator Design for Array Antenna Applications." In 6th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Service, 2003. TELSIKS 2003.,1:57 to 59. Serbia, Montenegro, Nis: IEEE, 2003.

See Also

More About

- “Design Two-Stage Low Noise Amplifier Using Microstrip Transmission Line Matching Network” on page 6-219

Extract S-Parameters from Circuit

This example uses the Symbolic Math Toolbox™ to explain how the RF Toolbox™ extracts two-port S-parameters from an RF Toolbox circuit object.

Consider a two-port network as shown in the figure 1that you want to characterize with S-parameters. S-parameters are defined as $V - I \times Z_0 = S(V + I \times Z_0)$.

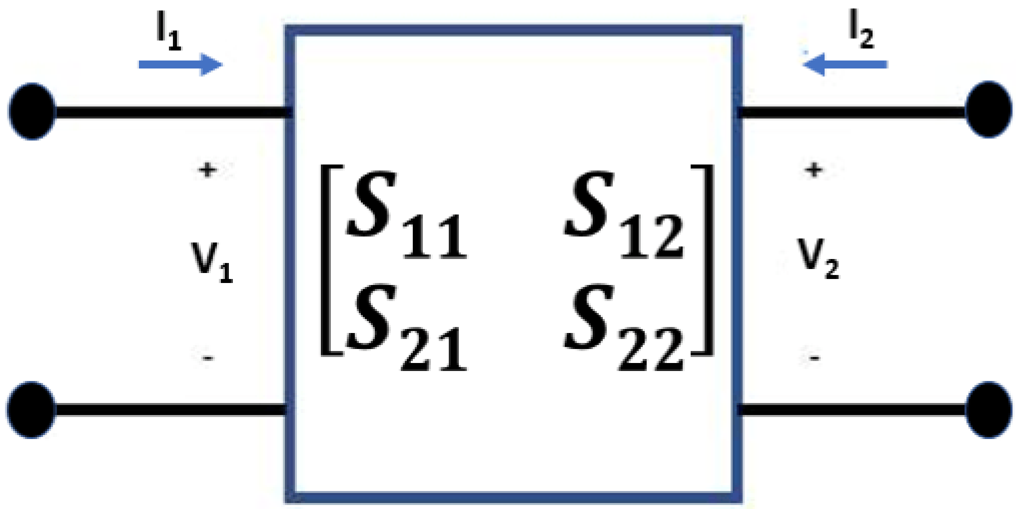


Figure 1: Two-Port Network

To extract the S-parameters from a circuit into an `sparameters` object, the RF Toolbox terminates each port with the reference impedance Z_0. Then, the RF Toolbox independently drives each port *j*, with $\frac{1}{Z_0}$ and solves for the port voltages V_{ij}. Driving with current sources is the Norton equivalent of driving with a 1 V source and a series resistance of Z_0.

Measure the port voltage V_{ij} at node *i* when node *j* is driven.

- If $i \neq j$, the S-parameter entry S_{ij} is simply twice the port voltage V_{ij}, and this is given using the equation $S_{ij} = 2 \times V_{ij}$.
- The diagonal entries of S-parameters when $i = j$ are given using the equation $S_{ij} = 2 \times V_{ij} - 1$.

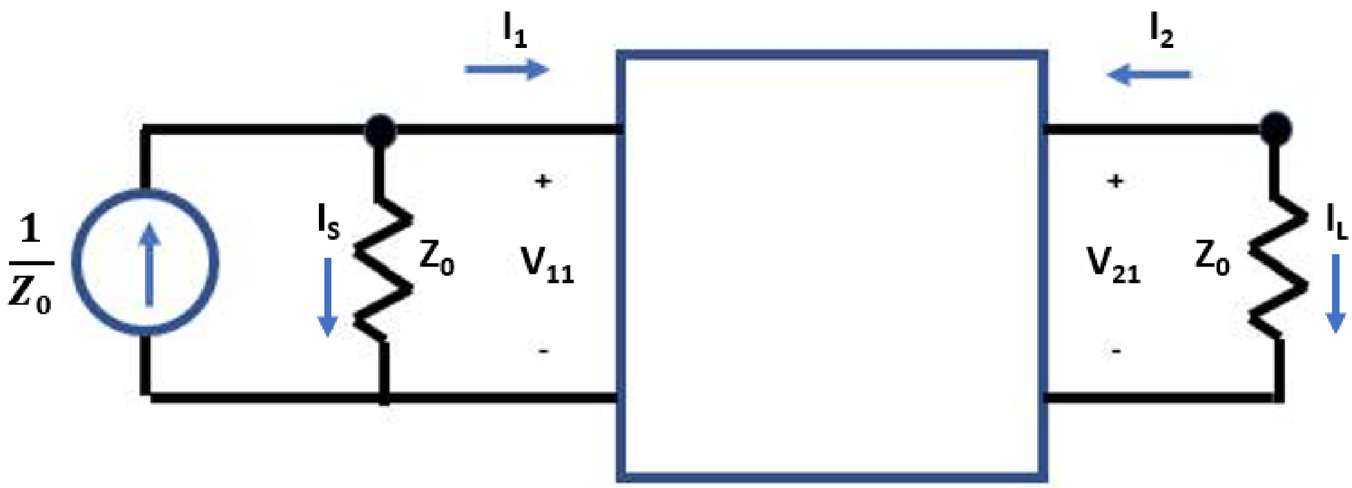


Figure 2: Circuit Driven at Port 1 with Current Source

Write Constitutive and Conservative Equations of Circuit

Circuits are represented in node-branch form in the RF Toolbox. There are four branches in the circuit represented in figure 2, one for the input port, two for the two-port `nport` object, and one for the output port. This means that the circuit has four branch current unknowns I_S, I_1, I_2, and I_L and two node voltages V_{11} and V_{21}. To represent the circuit described in figure 2 in node-branch form, you need four constitutive equations to represent the branch currents and two conservative equations to represent the node voltages.

```
syms F IS I1 I2 IL V1 V2 Z0
syms S11 S12 S21 S22

nI = 4; % number of branch currents
nV = 2; % number of node voltages

% F = [Fconstitutive; Fconservative]
F = [
    V1 - Z0*IS
    V1 - Z0*I1 - S11*(V1+Z0*I1) - S12*(V2+Z0*I2)
    V2 - Z0*I2 - S21*(V1+Z0*I1) - S22*(V2+Z0*I2)
    V2 - Z0*IL
    IS+I1
    I2+IL
    ]
```

F =

$$\begin{pmatrix} V_1 - \mathrm{IS}\, Z_0 \\ V_1 - I_1 Z_0 - S_{11}(V_1 + I_1 Z_0) - S_{12}(V_2 + I_2 Z_0) \\ V_2 - I_2 Z_0 - S_{21}(V_1 + I_1 Z_0) - S_{22}(V_2 + I_2 Z_0) \\ V_2 - \mathrm{IL}\, Z_0 \\ I_1 + \mathrm{IS} \\ I_2 + \mathrm{IL} \end{pmatrix}$$

Jacobian Evaluation of Circuit

Use the `jacobian` function from the Symbolic Math Toolbox to compute the matrix of derivatives of the function `F` with respect to the six unknowns (four branch currents and two node voltages)

```
J = jacobian(F,[IS; I1; I2; IL; V1; V2])
```

J =

$$\begin{pmatrix} -Z_0 & 0 & 0 & 0 & 1 & 0 \\ 0 & -Z_0 - S_{11} Z_0 & -S_{12} Z_0 & 0 & 1 - S_{11} & -S_{12} \\ 0 & -S_{21} Z_0 & -Z_0 - S_{22} Z_0 & 0 & -S_{21} & 1 - S_{22} \\ 0 & 0 & 0 & -Z_0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Solve S-Parameters of Circuit

Create a two-column right-hand side vector, `rhs`, to represent the driving of each port.

```
syms rhs [nI+nV 2]
syms x v S

% Compute S-parameters of cascade
rhs(:,:) = 0;
rhs(nI+1,1) = 1/Z0;  % rhs for driving input port
rhs(nI+nV,2) = 1/Z0  % rhs for driving output port
```

rhs =

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{Z_0} & 0 \\ 0 & \frac{1}{Z_0} \end{pmatrix}$$

To solve for the voltages, back solve the `rhs` with the Jacobian. The S-parameter matrix that MATLAB outputs represents the two-port circuit shown in figure 1.

```
x = J \ rhs;
v = x(nI+[1 nV],:);
S = 2*v - eye(2)
```

S =

$$\begin{pmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{pmatrix}$$

See Also

More About

- "Extract S-Parameters from Mutual Inductor" on page 6-241

- "De-Embedding S-Parameters" on page 6-113
- "Bisect S-Parameters of Cascaded Probes" on page 6-117

Extract S-Parameters from Mutual Inductor

This example shows how to build a user-defined element from S-parameters and add it to an `rfbudget` object for link budget analysis using the Symbolic Math Toolbox™. The user-defined element in this example is a mutual inductor.

Consider a mutual inductor as shown in figure 1 with the inductors $L_a$and L_b. This examples uses the Symbolic Math Toolbox to extract the analytical S-parameters of the mutual inductor and write them an RF Toolbox™ object. To extract S-parameters from a circuit, see “Extract S-Parameters from Circuit” on page 6-237.

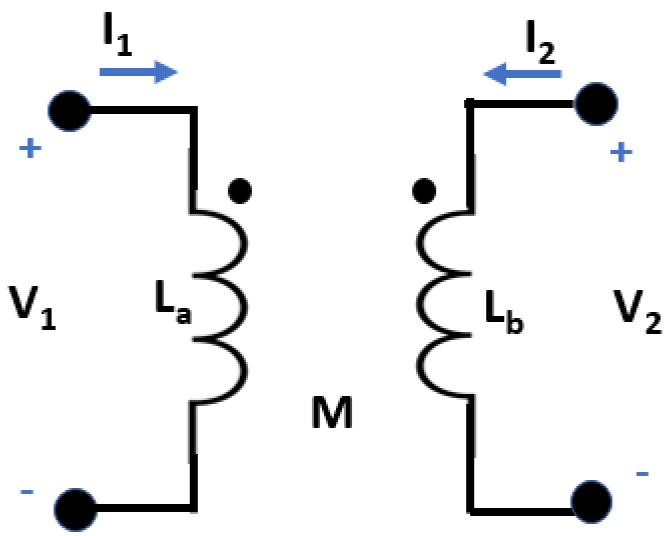


Figure 1: Mutual Inductor

One way to model a mutual inductor in the RF Toolbox is to draw the mutual inductor as an equivalent of a two-port network of inductors in a T configuration. Such a mutual inductor is shown in figure 2 with the mutual inductance M and the coupling coefficient k. Mutual inductance is given by the equation $M = k\sqrt{L_a \times L_b}$ relates M and `k`. Inductors in a T configuration can have negative values when there is a strong coupling between the inductors or if the M is greater than $L_a$or L_b.

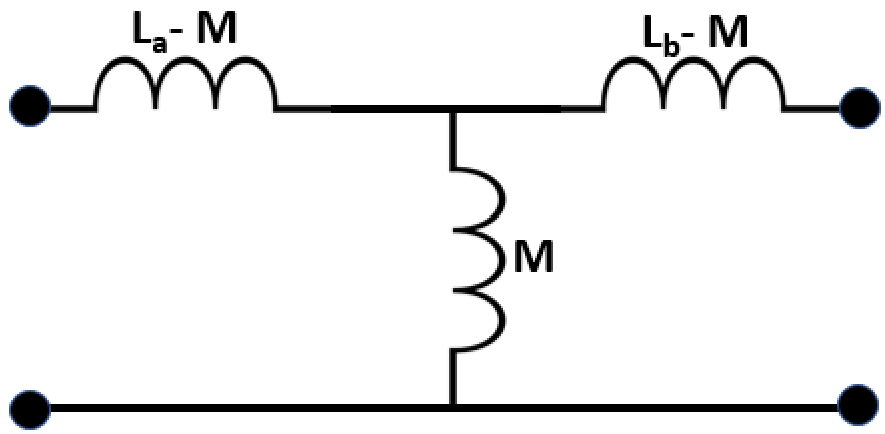


Figure 2: T-Circuit Representation of Mutual Inductor

Represent Circuit in Node-Branch Form

As discussed in the “Extract S-Parameters from Circuit” on page 6-237 example, to extract S-parameters from a circuit you need to drive one port while terminating the other. This is shown in figure 3. Use constitutive and conservative equations to represent the circuit in node-branch form. There are eight unknowns, five branch currents and three node voltages. Therefore there are eight equations in the node-form, five constitutive equations for the branches and three conservative equations obtained from the Kirchoff's Current Law for the nodes. The constitutive equation for a resistor is derived from Ohm's Law,$V = IR$, and the constitutive equation for an inductor is given by $V = sLR$, where `s` is a complex frequency.

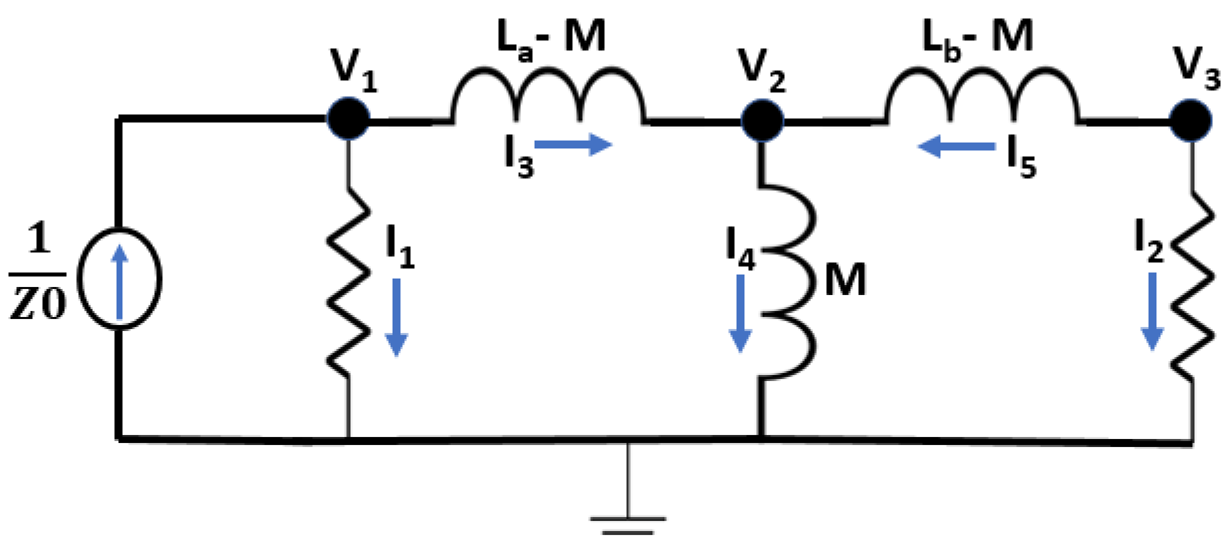


Figure 3: Mutual Inductor Driven at Port 1 with Current Source

```
syms F
syms I [5 1]
```

```
syms V [3 1]
syms Z0 La Lb M s

nI=5; % number of branch currents
nV=3; % number of node voltages

% F = [Fconstitutive; Fconservative]
F = [
    V1 - Z0*I1
    V1 - V2 - (La-M)*I3*s
    V2 - M*I4*s
    V2 - V3 + (Lb-M)*I5*s
    V3 - I2*Z0
    I1 + I3
    I4 - I5 - I3
    I2 + I5
    ]
```

F =

$$\begin{pmatrix} V_1 - I_1 Z_0 \\ V_1 - V_2 - I_3 s (\mathrm{La} - M) \\ V_2 - I_4 M s \\ V_2 - V_3 + I_5 s (\mathrm{Lb} - M) \\ V_3 - I_2 Z_0 \\ I_1 + I_3 \\ I_4 - I_3 - I_5 \\ I_2 + I_5 \end{pmatrix}$$

Compute Jacobian

Determine the Jacobian with respect to the unknowns, the five branch currents and three node voltages.

```
J = jacobian(F,[I; V]);
```

Solve S-parameters for Further Analysis

As shown in the **Extract S-Parameter from a Circuit** example, create the right-hand side, `rhs` vector to the drive and terminate ports.

```
syms rhs [nI+nV 2]
syms x v S t

% Compute S-parameters of cascade
rhs(:,:) = 0;
rhs(nI+1,1) = 1/Z0;  % rhs for driving input port
rhs(nI+nV,2) = 1/Z0  % rhs for driving output port
```

rhs =

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{Z_0} & 0 \\ 0 & 0 \\ 0 & \frac{1}{Z_0} \end{pmatrix}$$

By backsolving `rhs`, solve for the voltages using Jacobian.

```
x = J \ rhs;
v = x(nI+[1 nV],:);
S = (2*v - eye(2));
```

Create Object for RF Toolbox

In order to create a `sparameters` object, the parameters must be determined at a set of frequencies. To do so, define the variables for your mutual inductor. If you would like to test multiple values for your variables and automatically update your `sparameters` object, use **Numeric Sliders** in the **Control** drop-down under the **Live Editor** tab. Then, use the Symbolic Math Toolbox's `matlabFunction` to automatically generate a function, `mutualInductorS` to compute the analytic S-parameters at a set of frequencies. Finally, use the `sparameters` object to create a S-parameters object.

```
matlabFunction(S,'file','mutualInductorS.m','Optimize',false);
```

```
La = 0.000001;
Lb = 0.000001;
Z0 = 50;
k = 0.763;
M = k*((La*Lb)^(1/2));

freq = linspace(1e9,2e9,10);
s = 2i*pi*freq;
s_param = zeros(2,2,10);
for index = 1:numel(freq)
    s_param(:,:,index) = mutualInductorS(Lb,Lb,M,Z0,s(index));
end

Sobj = sparameters(s_param,freq);
```

Create Object for RF Budget

Use an `rfwrite` function to create a Touchstone® file from the `sparameters` object.

```
rfwrite(Sobj,'mutualInductor.s2p');
```

Create a `nport` object.

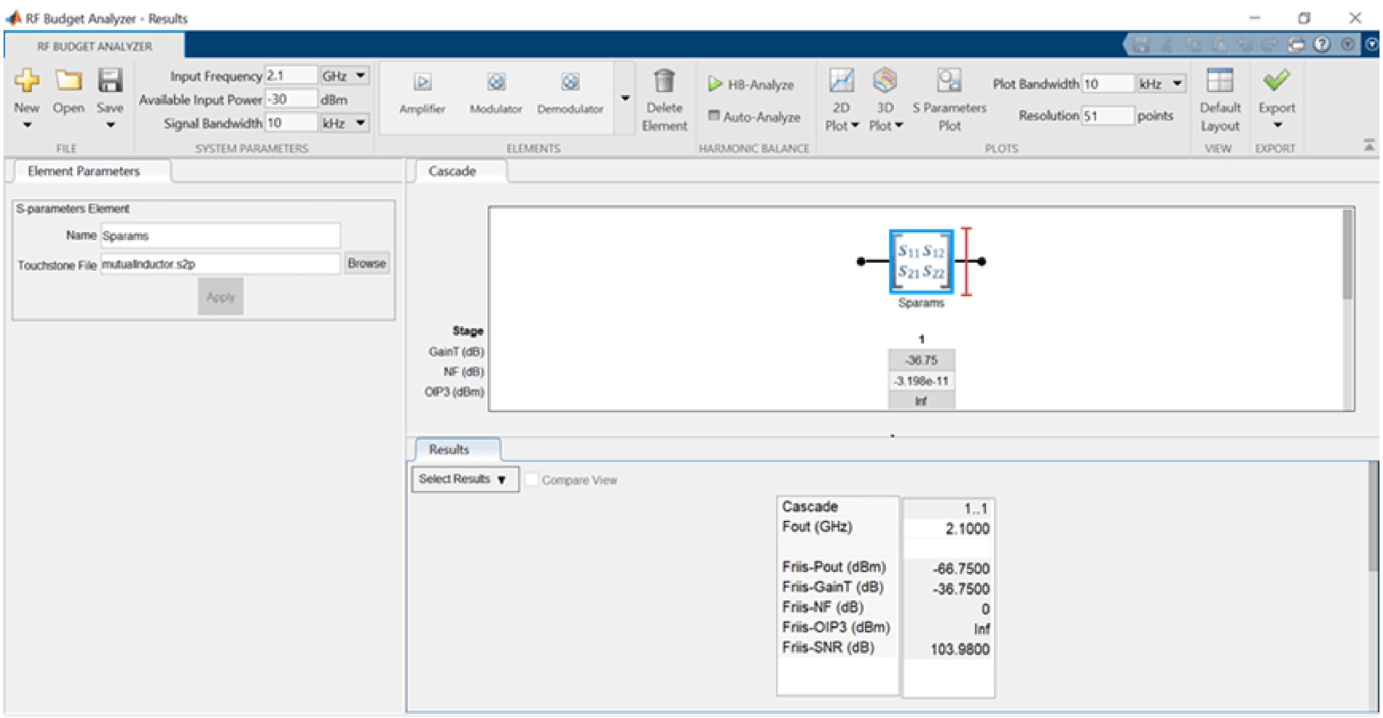
```
n = nport('mutualInductor.s2p');
```

Provide the `nport` object as an input to `rfbudget` object.

```
b = rfbudget(n,2.1e9,-30,10e3);
```

Type this command at the MATLAB Command Window to open the mutual inductor as a S-parameter element in the **RF Budget Analyzer** app.

```
show(b)
```



Using this method you can build your own components for RF budget analysis.

See Also

More About

- "Extract S-Parameters from Circuit" on page 6-237
- "De-Embedding S-Parameters" on page 6-113
- "Bisect S-Parameters of Cascaded Probes" on page 6-117

Lossy Multiconductor Transmission Line Circuit

This example shows how to analyze a lossy multiconductor transmission line circuit. The transmission lines are modeled using RLGC distributed elements which are used in signal integrity analysis for accurately capturing high-speed interconnect effects. You can analyze this circuit using RF Toolbox™ `pwlresp` function or using RF Blockset™ Circuit Envelope model. In this example you first compute the time-domain response of the circuit excited by a periodic pulse signal and then you compare the result to a circuit simulated using the RF Blockset Circuit Envelope simulation.

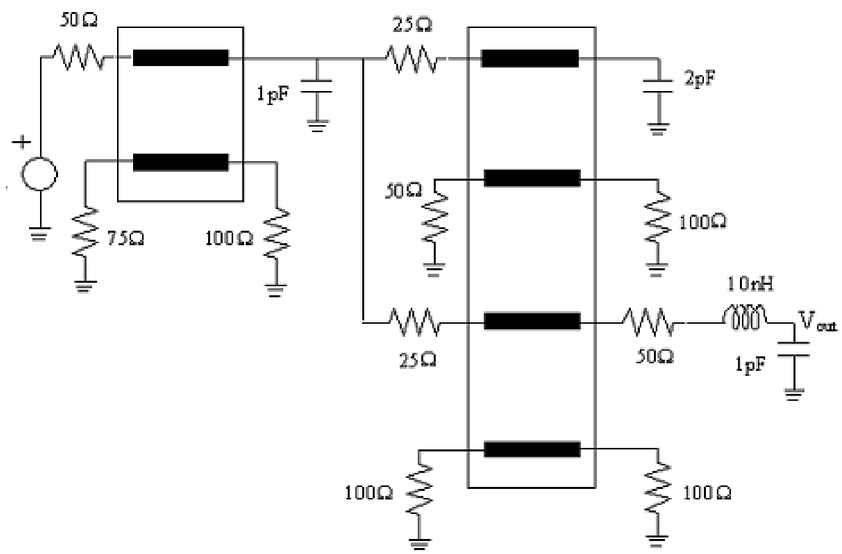


Figure 1: Interconnect circuit with lossy multiconductor transmission lines

Representation of Distributed Transmission Line Models with N-Port Elements

The circuit in this example contains two sets of lossy coupled transmission lines. The RLGC line parameter matrices of the two lines are as follows [1].

TLine 1 (len1 = 0.1m)

$$\boldsymbol{C} = \begin{bmatrix} 62.8 & -4.9 \\ -4.9 & 62.8 \end{bmatrix} \text{pF/m} \quad \boldsymbol{L} = \begin{bmatrix} 494.6 & 63.3 \\ 63.3 & 494.6 \end{bmatrix} \text{nH/m}$$

$$\boldsymbol{R} = \begin{bmatrix} 75 & 15 \\ 15 & 75 \end{bmatrix} \Omega/\text{m} \quad \boldsymbol{G} = \begin{bmatrix} 0.1 & -0.01 \\ -0.01 & 0.1 \end{bmatrix} \text{S/m}$$

TLine 2 (len2 = 0.1m)

$$\boldsymbol{C} = \begin{bmatrix} 62.8 & -4.9 & -0.3 & 0.0 \\ -4.9 & 62.8 & -4.9 & -0.3 \\ -0.3 & -4.9 & 62.8 & -4.9 \\ 0.0 & -0.3 & -4.9 & 62.8 \end{bmatrix} \text{pF/m} \quad \boldsymbol{L} = \begin{bmatrix} 494.6 & 63.3 & 7.8 & 0.0 \\ 63.3 & 494.6 & 63.3 & 7.8 \\ 7.8 & 63.3 & 494.6 & 63.3 \\ 0.0 & 7.8 & 63.3 & 494.6 \end{bmatrix} \text{nH/m}$$

$$\boldsymbol{R} = \begin{bmatrix} 50 & 10 & 1 & 0.0 \\ 10 & 50 & 10 & 1 \\ 1 & 10 & 50 & 10 \\ 0.0 & 1 & 10 & 50 \end{bmatrix} \Omega/\text{m} \quad \boldsymbol{G} = \begin{bmatrix} 0.1 & -0.01 & -0.001 & 0.0 \\ -0.01 & 0.1 & -0.01 & -0.001 \\ -0.001 & -0.01 & 0.1 & -0.01 \\ 0.0 & -0.001 & -0.01 & 0.1 \end{bmatrix} \text{S/m}$$

Figure 2: RLGC line parameters for the 4-ports and 8-ports transmission lines

Extract the S-parameters of each coupled transmission lines using `rlgc2s` function. Then represent the S-parameters using an `nport` element.

```
freq = linspace(1e1,8e9,1001);
N = length(freq);
len1 = 0.1;
R1 = ones(2,2,N).*[75,15;15,75];
L1 = ones(2,2,N).*[494.6,63.3;63.3,494.6]*1e-9;
G1 = ones(2,2,N).*[0.1,-0.01;-0.01,0.1];
C1 = ones(2,2,N).*[62.8,-4.9;-4.9,62.8]*1e-12;

s_params1 = rlgc2s(R1,L1,G1,C1,len1,freq);
stlobj1 = sparameters(s_params1,freq);
nport_s1 = nport(stlobj1);

len2 = 0.1;
R2 = ones(4,4,N).*[50,10,1,0.0;10,50,10,1;1,10,50,10;0.0,1,10,50];
L2 = ones(4,4,N).*[494.6,63.3,7.8,0.0;63.3,494.6,63.3,7.8;7.8,63.3,494.6,63.3;0.0,7.8,63.3,494.6]
G2 = ones(4,4,N).*[0.1,-0.01,-0.001,0.0;-0.01,0.1,-0.01,-0.001;-0.001,-0.01,0.1,-0.01;0.0,-0.001
C2 = ones(4,4,N).*[62.8,-4.9,-0.3,0.0;-4.9,62.8,-4.9,-0.3;-0.3,-4.9,62.8,-4.9;0.0,-0.3,-4.9,62.8]

s_params2 = rlgc2s(R2,L2,G2,C2,len2,freq);
stlobj2 = sparameters(s_params2,freq);
nport_s2 = nport(stlobj2);
```

Calculation of S-Parameter Object of Interconnect Circuit

Calculate the S-parameter object of the circuit given in Figure.1 by using the `sparameters` function.

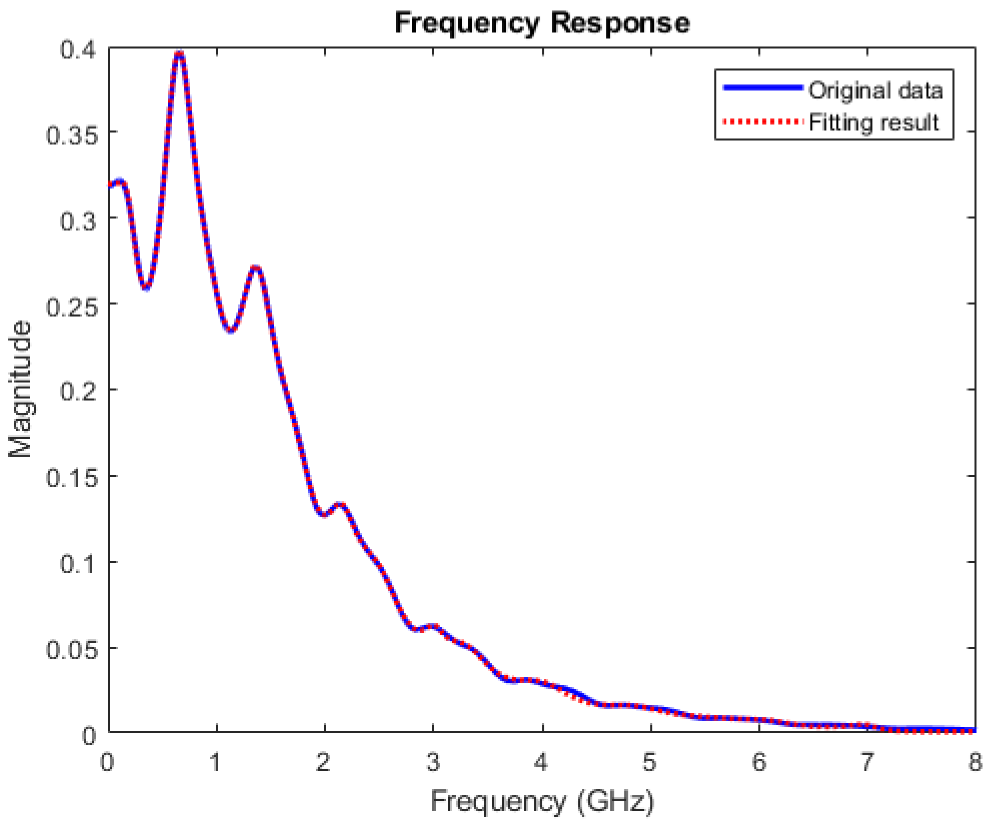
```
ckt = circuit('interconnect');

add(ckt,[1 2],resistor(50))
add(ckt,[2 3 4 5],nport_s1,{'p1+' 'p2+' 'p3+' 'p4+'})
add(ckt,[3 0],resistor(75))
add(ckt,[4 0],capacitor(1e-12))
add(ckt,[5 0],resistor(100))
add(ckt,[4 6],resistor(25))
add(ckt,[7 0],resistor(50))
add(ckt,[4 8],resistor(25))
add(ckt,[9 0],resistor(100))
add(ckt,[6 7 8 9 10 11 12 13],nport_s2,{'p1+' 'p2+' 'p3+' 'p4+' 'p5+' 'p6+' 'p7+' 'p8+'})
add(ckt,[10 0],capacitor(2e-12))
add(ckt,[11 0],resistor(100))
add(ckt,[13 0],resistor(100))
add(ckt,[12 14],resistor(50))
add(ckt,[14 15],inductor(10e-9))
add(ckt,[15 0],capacitor(1e-12))

freqfit = linspace(1e1,8e9,1001)';
setports(ckt,[1 0],[15 0])
S_vout = sparameters(ckt,freqfit);
```

Generation of Rational Object of Multiconductor Transmission Line Circuit

Convert the transfer function of the S-parameter object of the circuit to a `rational` object over the specified frequencies, . Then calculate the frequency response using the `freqresp` function.

```
tfblkout = s2tf(S_vout,0,Inf,2);
fitblkout = rational(freqfit,tfblkout);
```

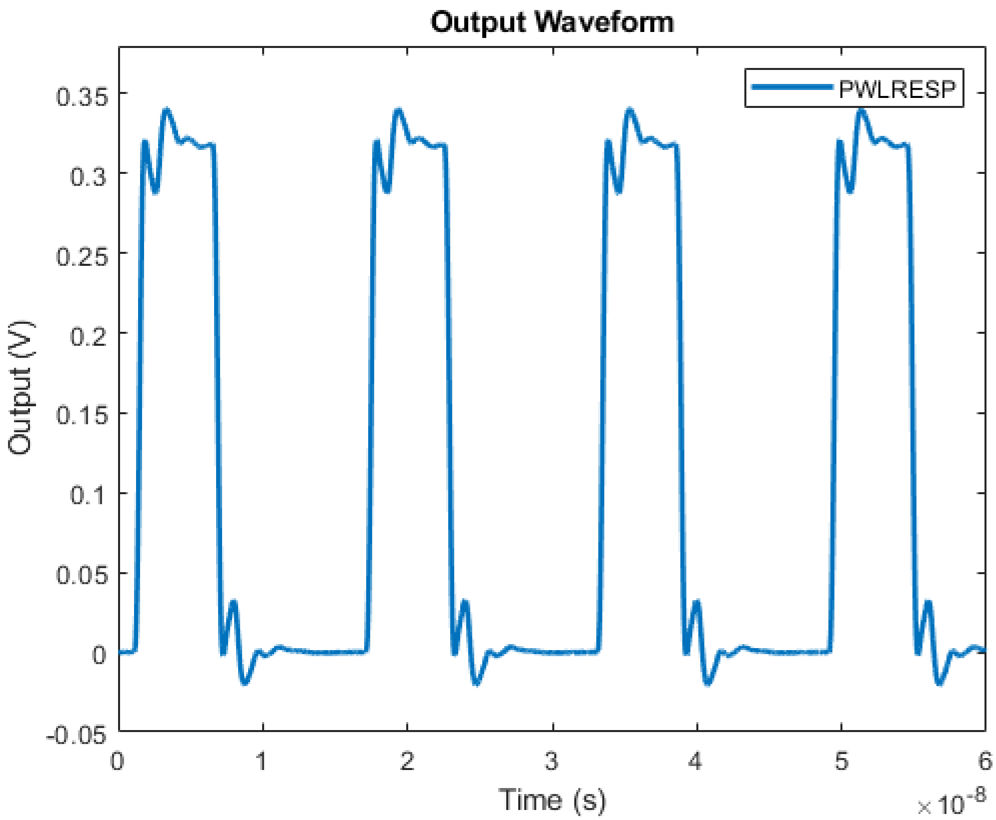
```
plot(freqfit/1e9,abs(tfblkout),'b',freqfit/1e9,abs(freqresp(fitblkout,freqfit)),'r:','LineWidth'
legend('Original data','Fitting result')
title('Frequency Response')
ylabel('Magnitude')
xlabel('Frequency (GHz)')
```



Computation of Transient Waveform Excited by Periodic Pulse Source

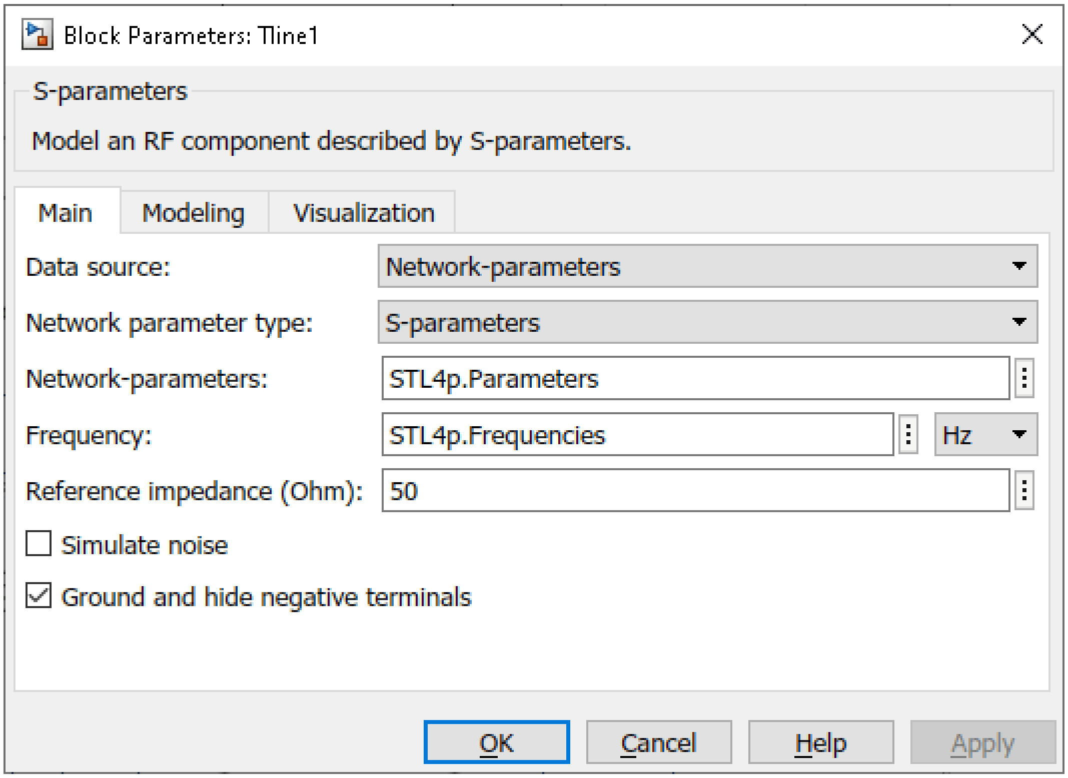
Apply a voltage source using a 1 V periodic pulse with a rise/fall time of 0.4 ns, a duration of 5 ns, and a period of 1.6 ns. Then compute the time-domain response of the circuit by using the `pwlresp` function.

```
SignalTime = [0,0.4,5.4,5.8]*1e-9;
SignalValue = [0,1,1,0];
Tsim = (0:1e-11:6e-8);
TP = 1.6e-8;
[WAVEOUT, tout] = pwlresp(fitblkout,SignalTime,SignalValue,Tsim,TP);

plot(tout,WAVEOUT, 'LineWidth',2)
legend('PWLRESP')
title('Output Waveform')
xlabel('Time (s)')
ylabel('Output (V)')
ylim([-0.05 0.38])
```



Comparison of Time-Domain Response of Coupled Transmission Line Circuit Using RF Blockset

A Simulink model is built to simulate the multiconductor transmission line circuit shown in Figure 1. With the S-parameters calculated from previous session, the multiconductor transmission lines can be represented by S-parameter blocks from RF Blockset Circuit Envelope Library. For instance, the 4-port transmission lines are modeled by defining the S-parameter block mask as shown below, where `STL4p` is a S-parameter object converted from reordering ports of S-parameter object `stlobj1` with `snp2smp` function. The output of the model is the time-domain response in the form of a passband signal. For more information on passband signal implementation in the Circuit Envelope model, see “Passband Signal Representation in Circuit Envelope” (RF Blockset).

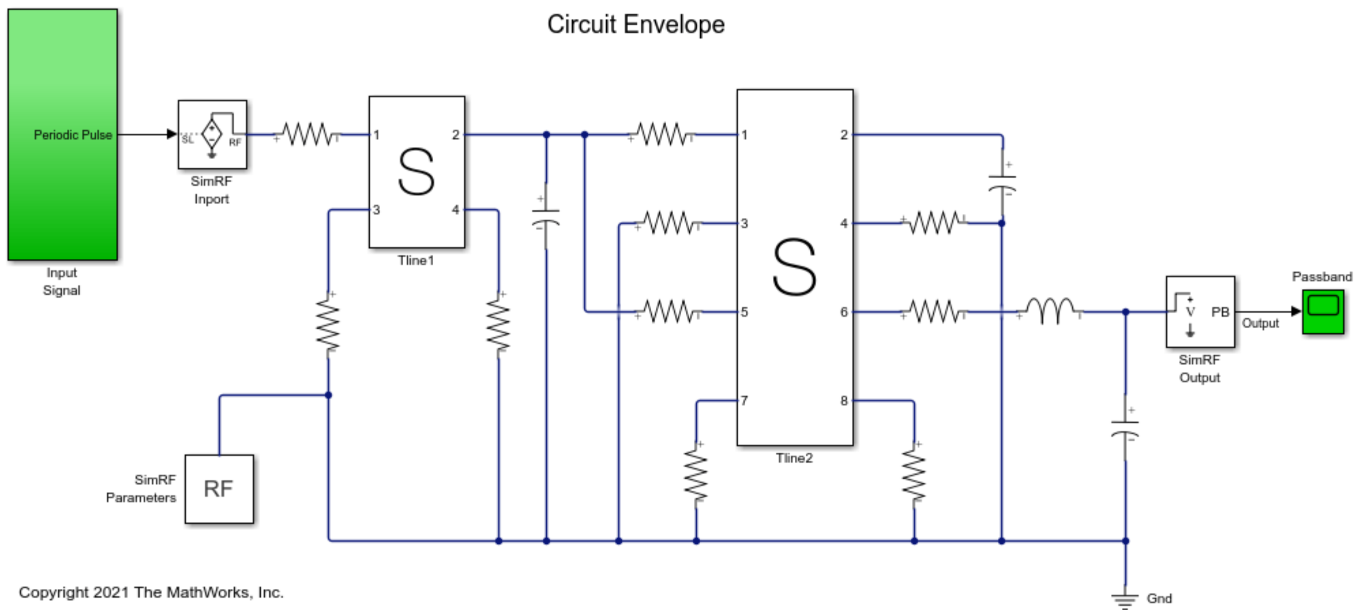


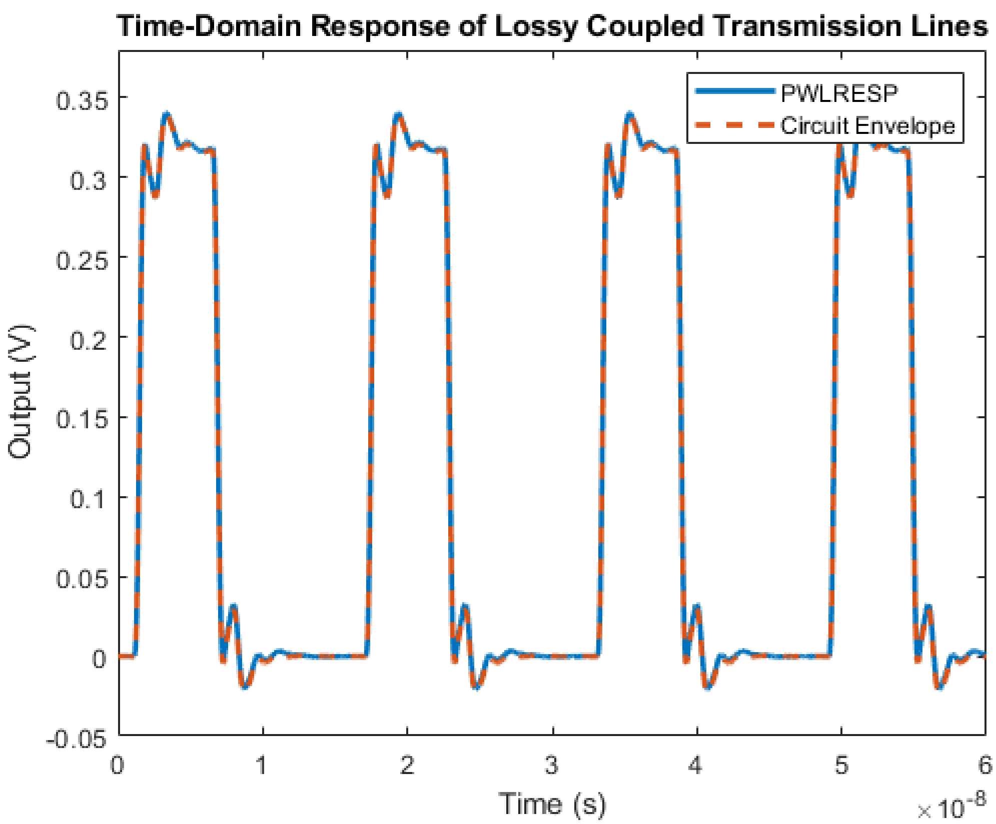
To simulate the model:

1. Type `open_system('simrf_coupled_pb')` at the command prompt.
2. Select **Simulation** > **Run**.

After you finish simulating the model, compare the obtained transient waveform with that calculated using RF Toolbox.

```
open_system('simrf_coupled_pb');
sim('simrf_coupled_pb');
plot(tout,WAVEOUT,CE_Data(:,1),CE_Data(:,2),'--','LineWidth',2)
legend('PWLRESP', 'Circuit Envelope')
title('Time-Domain Response of Lossy Coupled Transmission Lines')
xlabel('Time (s)')
ylabel('Output (V)')
ylim([-0.05 0.38])
```


Circuit Envelope
Periodic Pulse
Input
Signal
SimRF
Inport
Tline1
Tline2
SimRF
Parameters
RF
SimRF
Output
Output
Passband
Gnd
Copyright 2021 The MathWorks, Inc.


Time-Domain Response of Lossy Coupled Transmission Lines
PWLRESP
Circuit Envelope
Output (V)
Time (s)
×10^-8

```
bdclose('simrf_coupled_pb');
```

References

[1] Tang, Tak K., and Michel Nakhla. “Analysis of High-Speed VLSI Interconnects Using the Asymptotic Waveform Evaluation Technique.” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 11, no. 3 (March 1992): 341–52. https://doi.org/10.1109/43.124421.

See Also

Related Examples

- “Using Rational Object to Fit S-Parameters” on page 6-215
- “Passband Signal Representation in Circuit Envelope” (RF Blockset)

Richards-Kuroda Workflow for RF Filter Circuit

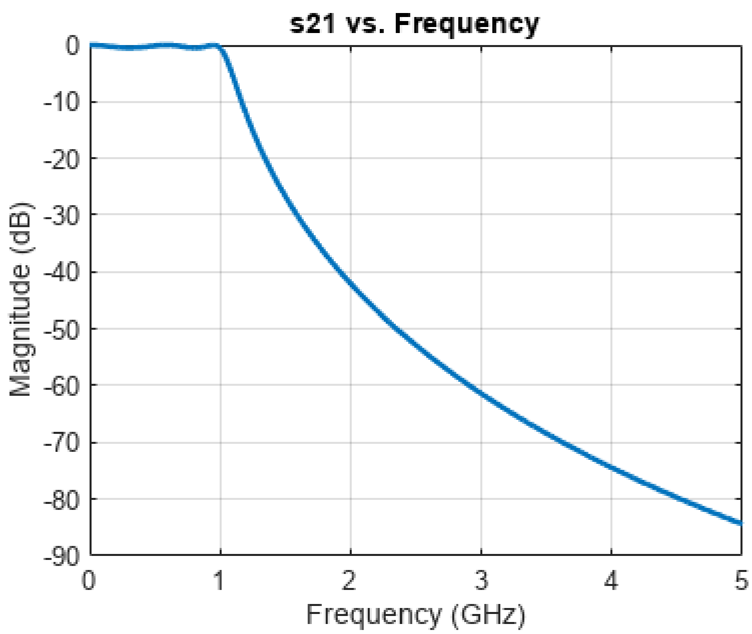
This example shows how to apply Richards-Kuroda workflow to an RF filter circuit.

Create a lowpass LC-Pi Chebyshev filter with the passband frequency of 1 GHz, passband attenuation of 0.5 dB, and filter order of 5.

```
Fp = 1e9;
Ap = 0.5;
Ord = 5;
r = rffilter("FilterType","Chebyshev","ResponseType","Lowpass","Implementation","LC Pi","Passband
    Fp,"PassbandAttenuation",Ap,"FilterOrder",Ord);
```

Plot the S21 parameter of the RF filter.

```
frequencies = linspace(0,5*Fp,1001);
rfplot(r, frequencies)
```



Convert the lumped-element filter to a distributed-element-based circuit at the operating frequency of 1 GHz using the `richards` function.

```
 txCkt = richards(r,1e9)
```

```
txCkt = 
  circuit: Circuit element

    ElementNames: {'C_tx'  'L_tx'  'C_1_tx'  'L_1_tx'  'C_2_tx'}
        Elements: [1x5 txlineElectricalLength]
           Nodes: [0 1 2 3 4 5 6]
            Name: 'unnamed'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

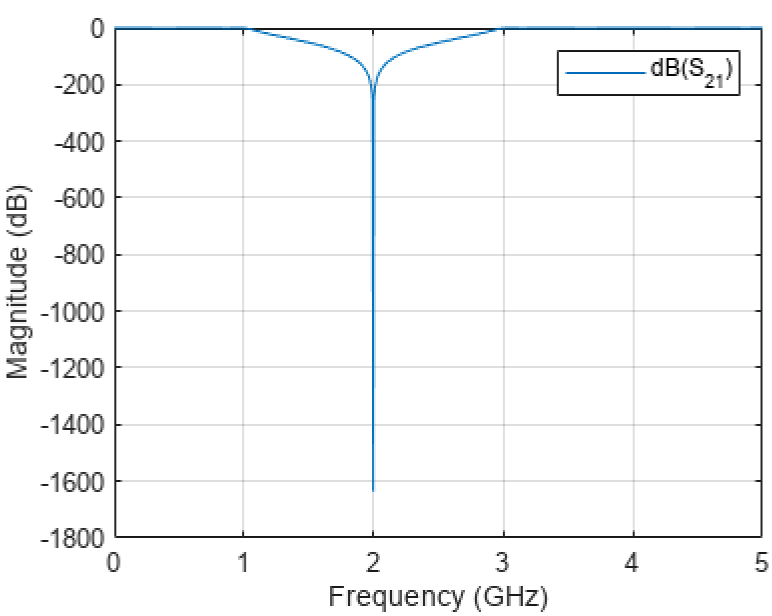
Show the circuit properties in a table using the `tableCircuitProperties` function. You can find the source code for this helper function in the Supporting Functions on page 6-260 section at the end of this example.

```
tableCircuitProperties(txCkt,'Name','StubMode','Termination','Z0')
```

```
       Name          StubMode       Termination      Z0  
    ___________    ___________    ____________    ______

    {'C_tx'   }    {'Shunt' }      {'Open' }      29.312
    {'L_tx'   }    {'Series'}      {'Short'}      61.481
    {'C_1_tx'}     {'Shunt' }      {'Open' }      19.679
    {'L_1_tx'}     {'Series'}      {'Short'}      61.481
    {'C_2_tx'}     {'Shunt' }      {'Open' }      29.312
```

Plot the S21 parameter of distributed-element-based filter circuit. The RF plot shows that distributed and lumped filter behavior overlap close to the operating frequency and diverge significantly at higher frequencies. This occurs due to the frequency-periodic nature of distributed elements.

```
rfplot(sparameters(txCkt, frequencies),2,1)
```



The distributed-element-based circuit in `txCkt` circuit is not practical since all stubs essentially stem from the same point in space. To separate the stubs and use only shunt stubs that are easier to implement as microstrip lines, insert unit elements in a sequence and apply Kuroda's identities.

Add unit elements at the edges of the `txCkt`, operating at 1 GHz with the characteristic impedance of 50 ohms. The edges of the circuit are port 1 of the first circuit element `C_tx` and port 2 of the last circuit element `C_2_tx`.

```
txCkt_UE = insertUnitElement(txCkt,'C_tx',1,1e9,50);
txCkt_UE = insertUnitElement(txCkt_UE,'C_2_tx',2,1e9,50)
```

```
txCkt_UE = 
  circuit: Circuit element
```

```
    ElementNames: {'C_tx_p1_elem_UE'  'C_tx'  'L_tx'  'C_1_tx'  'L_1_tx'  'C_2_tx'  'C_2_tx_p2_e
        Elements: [1x7 txlineElectricalLength]
           Nodes: [0 1 2 3 4 5 6 7 8]
            Name: 'unnamed'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

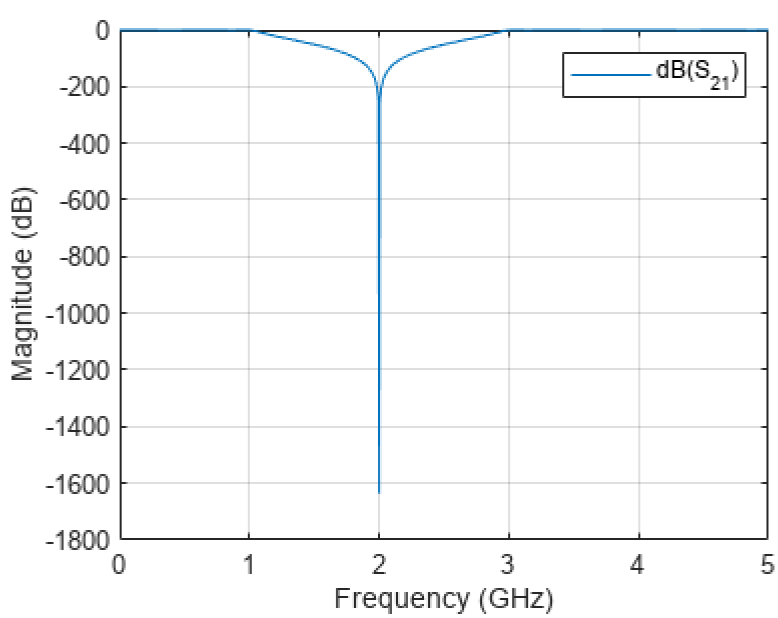
Show the circuit properties in a table.

```
tableCircuitProperties(txCkt_UE,'Name','StubMode','Termination','Z0')
```

Name	StubMode	Termination	Z0
{'C_tx_p1_elem_UE' }	{'NotAStub'}	{'NotApplicable'}	50
{'C_tx' }	{'Shunt' }	{'Open' }	29.312
{'L_tx' }	{'Series' }	{'Short' }	61.481
{'C_1_tx' }	{'Shunt' }	{'Open' }	19.679
{'L_1_tx' }	{'Series' }	{'Short' }	61.481
{'C_2_tx' }	{'Shunt' }	{'Open' }	29.312
{'C_2_tx_p2_elem_UE'}	{'NotAStub'}	{'NotApplicable'}	50

Plot the S21 parameter of the new circuit `txCkt_UE`. The RF plot shows that the addition of unit elements does not change the power magnitude behavior of the circuit, and thus this RF plot shows the same characteristics of the S21 parameter as the distributed-element-based filter circuit.

```
rfplot(sparameters(txCkt_UE, frequencies),2,1)
```



Apply Kuroda's identities to the first two and last two elements of the circuit. For more information, see “Kuroda's Transformation”.

```
txCkt_Kur = kuroda(txCkt_UE,'C_tx_p1_elem_UE','C_tx');
txCkt_Kur = kuroda(txCkt_Kur,'C_2_tx','C_2_tx_p2_elem_UE')
```

```
txCkt_Kur = 
  circuit: Circuit element

    ElementNames: {'Kuroda2_R2L_of_C_tx_p1_elem_UE'  'Kuroda2_R2L_of_C_tx'  'L_tx'  'C_1_tx'  'L_
        Elements: [1x7 txlineElectricalLength]
           Nodes: [0 1 2 3 4 5 6 7 8]
            Name: 'unnamed'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

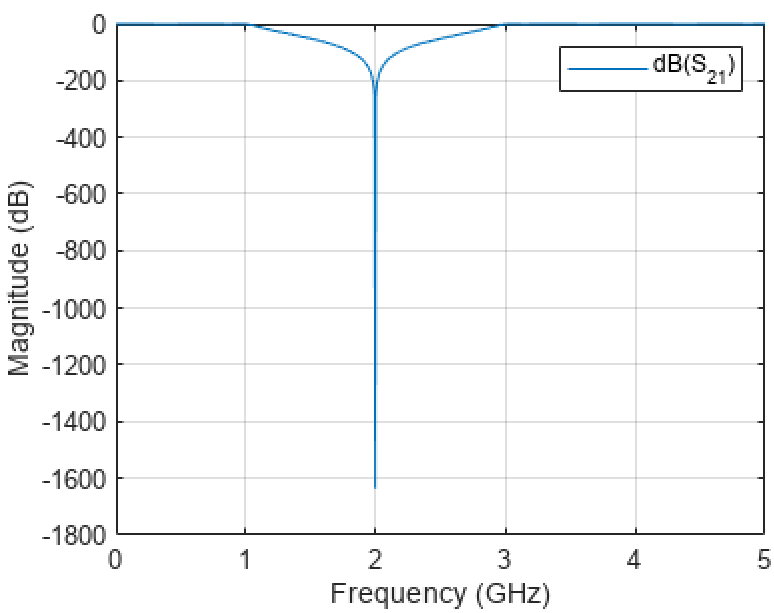
Show the circuit properties in a table.

```
tableCircuitProperties(txCkt_Kur,'Name','StubMode','Termination','Z0')
```

Name	StubMode	Termination	Z0
{'Kuroda2_R2L_of_C_tx_p1_elem_UE' }	{'Series' }	{'Short' }	31.521
{'Kuroda2_R2L_of_C_tx' }	{'NotAStub'}	{'NotApplicable'}	18.479
{'L_tx' }	{'Series' }	{'Short' }	61.481
{'C_1_tx' }	{'Shunt' }	{'Open' }	19.679
{'L_1_tx' }	{'Series' }	{'Short' }	61.481
{'Kuroda1_L2R_of_C_2_tx' }	{'NotAStub'}	{'NotApplicable'}	18.479
{'Kuroda1_L2R_of_C_2_tx_p2_elem_UE'}	{'Series' }	{'Short' }	31.521

Plot the S21 parameter of `txCkt_Kur`. The RF plot shows that, as expected, applying Kuroda's identities does not change the behavior of the circuit (as opposed to adding a unit element, applying Kuroda's identities retains both magnitude and phase behavior of the circuit).

```
rfplot(sparameters(txCkt_Kur, frequencies),2,1)
```



Add a unit elements at the edges of this circuit operating at 1 GHz with the characteristic impedance of 50 ohms.

```
txCkt_UE2 = insertUnitElement(txCkt_Kur,'Kuroda2_R2L_of_C_tx_p1_elem_UE',1,1e9,50);
txCkt_UE2 = insertUnitElement(txCkt_UE2,'Kuroda1_L2R_of_C_2_tx_p2_elem_UE',2,1e9,50)
```

```
txCkt_UE2 = 
  circuit: Circuit element

    ElementNames: {1x9 cell}
        Elements: [1x9 txlineElectricalLength]
           Nodes: [0 1 2 3 4 5 6 7 8 9 10]
            Name: 'unnamed'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

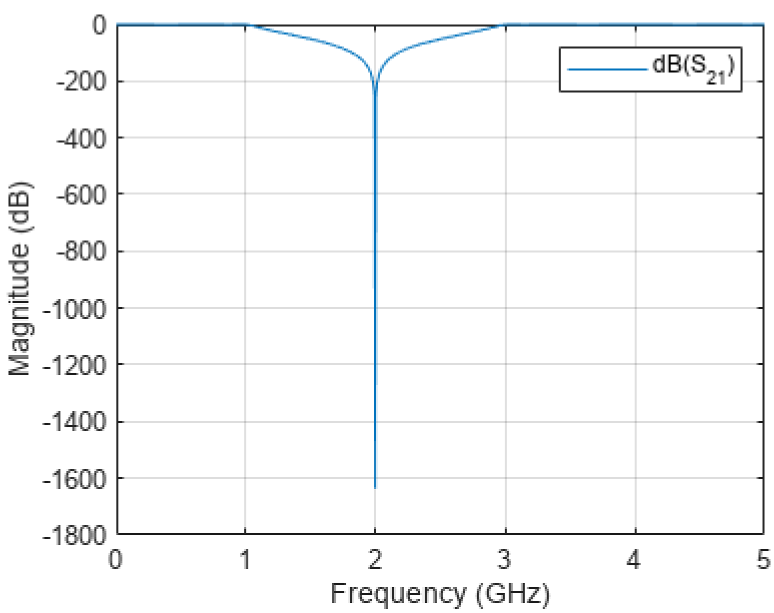
Show the circuit properties in a table.

```
tableCircuitProperties(txCkt_UE2,'Name','StubMode','Termination','Z0')
```

Name	StubMode	Termination	Z0
{'Kuroda2_R2L_of_C_tx_p1_elem_UE_p1_elem_UE' }	{'NotAStub'}	{'NotApplicable'}	5
{'Kuroda2_R2L_of_C_tx_p1_elem_UE' }	{'Series' }	{'Short' }	31.52
{'Kuroda2_R2L_of_C_tx' }	{'NotAStub'}	{'NotApplicable'}	18.47
{'L_tx' }	{'Series' }	{'Short' }	61.48
{'C_1_tx' }	{'Shunt' }	{'Open' }	19.67
{'L_1_tx' }	{'Series' }	{'Short' }	61.48
{'Kuroda1_L2R_of_C_2_tx' }	{'NotAStub'}	{'NotApplicable'}	18.47
{'Kuroda1_L2R_of_C_2_tx_p2_elem_UE' }	{'Series' }	{'Short' }	31.52
{'Kuroda1_L2R_of_C_2_tx_p2_elem_UE_p2_elem_UE'}	{'NotAStub'}	{'NotApplicable'}	5

Plot the S21 parameter of `txCkt_UE2`.

```
rfplot(sparameters(txCkt_UE2, frequencies),2,1)
```



Apply Kuroda's identities to the first, second, second to last, and last pair of elements in the circuit.

```
txCkt_Kur2 = kuroda(txCkt_UE2,'Kuroda2_R2L_of_C_tx_p1_elem_UE_p1_elem_UE','Kuroda2_R2L_of_C_tx_p1
txCkt_Kur2 = kuroda(txCkt_Kur2,'Kuroda2_R2L_of_C_tx','L_tx');
txCkt_Kur2 = kuroda(txCkt_Kur2,'Kuroda1_L2R_of_C_2_tx_p2_elem_UE','Kuroda1_L2R_of_C_2_tx_p2_elem_
txCkt_Kur2 = kuroda(txCkt_Kur2,'L_1_tx','Kuroda1_L2R_of_C_2_tx')
```

```
txCkt_Kur2 = 
  circuit: Circuit element

    ElementNames: {1x9 cell}
        Elements: [1x9 txlineElectricalLength]
           Nodes: [0 1 2 3 4 5 6 7 8 9 10]
            Name: 'unnamed'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

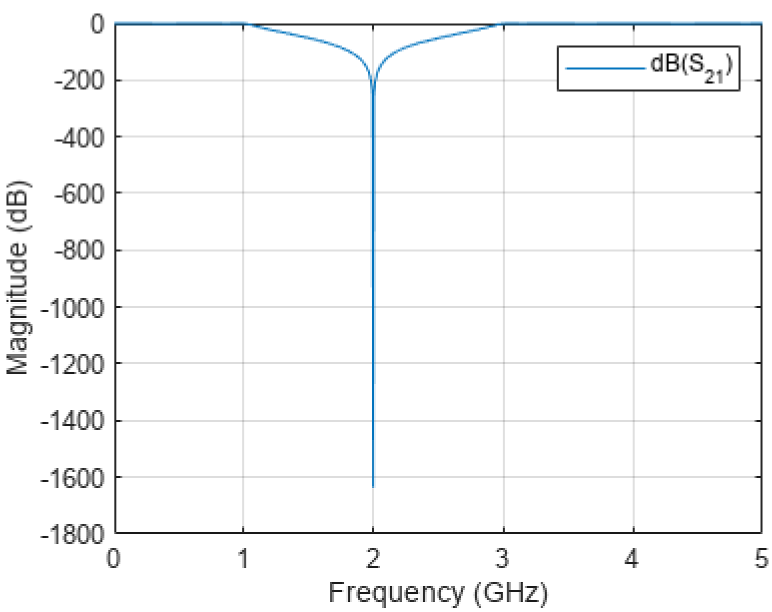
Show the circuit properties in a table.

```
tableCircuitProperties(txCkt_Kur2,'Name','StubMode','Termination','Z0')
```

```
                                 Name                                 StubMode        Terminat
    ______________________________________________________________    _____________   ________

    {'Kuroda1_R2L_of_Kuroda2_R2L_of_C_tx_p1_elem_UE_p1_elem_UE'  }    {'Shunt'    }   {'Open'
    {'Kuroda1_R2L_of_Kuroda2_R2L_of_C_tx_p1_elem_UE'             }    {'NotAStub'}    {'NotApplic
    {'Kuroda1_R2L_of_Kuroda2_R2L_of_C_tx'                        }    {'Shunt'    }   {'Open'
    {'Kuroda1_R2L_of_L_tx'                                       }    {'NotAStub'}    {'NotApplic
    {'C_1_tx'                                                    }    {'Shunt'    }   {'Open'
    {'Kuroda2_L2R_of_L_1_tx'                                     }    {'NotAStub'}    {'NotApplic
    {'Kuroda2_L2R_of_Kuroda1_L2R_of_C_2_tx'                      }    {'Shunt'    }   {'Open'
    {'Kuroda2_L2R_of_Kuroda1_L2R_of_C_2_tx_p2_elem_UE'           }    {'NotAStub'}    {'NotApplic
    {'Kuroda2_L2R_of_Kuroda1_L2R_of_C_2_tx_p2_elem_UE_p2_elem_UE'}    {'Shunt'    }   {'Open'
```

Plot the S21 parameter of `txCkt_Kur2`.

```
rfplot(sparameters(txCkt_Kur2, frequencies),2,1)
```



Create a microstrip transmission line. Then use the `realize` function to realize a circuit containing the electrical-length-based transmission line `txCkt_Kur2`.

```
txln = txlineMicrostrip('Height',0.0015748,'EpsilonR',4.6, 'LossTangent',0.026,'SigmaCond',596000
txMS = realize(txCkt_Kur2,txln)
```

```
txMS = 
  circuit: Circuit element

    ElementNames: {1x9 cell}
        Elements: [1x9 txlineMicrostrip]
           Nodes: [0 1 2 3 4 5 6 7 8 9 10]
            Name: 'unnamed'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

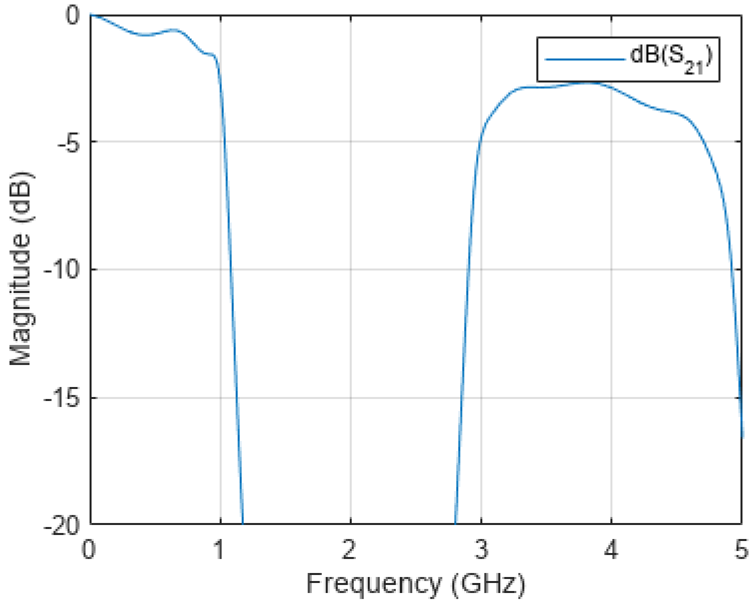
Show the circuit properties in a table.

```
tableCircuitProperties(txMS,'Name','StubMode','Termination','LineLength','Width')
```

```
                              Name                                         StubMode        Te
    ____________________________________________________________________    ____________    ______

    {'txlMs_of_Kuroda1_R2L_of_Kuroda2_R2L_of_C_tx_p1_elem_UE_p1_elem_'}    {'Shunt'   }    {'Oper
    {'txlMs_of_Kuroda1_R2L_of_Kuroda2_R2L_of_C_tx_p1_elem_UE'          }    {'NotAStub'}    {'Not/
    {'txlMs_of_Kuroda1_R2L_of_Kuroda2_R2L_of_C_tx'                     }    {'Shunt'   }    {'Oper
    {'txlMs_of_Kuroda1_R2L_of_L_tx'                                    }    {'NotAStub'}    {'Not/
    {'txlMs_of_C_1_tx'                                                 }    {'Shunt'   }    {'Oper
    {'txlMs_of_Kuroda2_L2R_of_L_1_tx'                                  }    {'NotAStub'}    {'Not/
    {'txlMs_of_Kuroda2_L2R_of_Kuroda1_L2R_of_C_2_tx'                   }    {'Shunt'   }    {'Oper
    {'txlMs_of_Kuroda2_L2R_of_Kuroda1_L2R_of_C_2_tx_p2_elem_UE'        }    {'NotAStub'}    {'Not/
    {'txlMs_of_Kuroda2_L2R_of_Kuroda1_L2R_of_C_2_tx_p2_elem_UE_p2_ele'}    {'Shunt'   }    {'Oper
```

Plot the S21 parameter of `txMS`. The S21 parameter of the microstrip-based circuit deviates from the S21 of `txCkt_Kur2`. This is due to the practical realization considerations. Remove losses to improve the agreement between the two plots.

```
rfplot(sparameters(txMS, frequencies),2,1)
set(gca,'YLim',[-20 0]);
```



Supporting Functions

tableCircuitProperties:

```
type("tableCircuitProperties.m")
```

```
function tableCircuitProperties(ckt,varargin)
c = cell(numel(ckt.Elements),nargin-1);
for col = 1:nargin-1
    c(:,col) = {ckt.Elements.(varargin{col})};
end
disp(cell2table(c, 'VariableNames',varargin));
end
```

See Also

richards | kuroda | insertUnitElement | realize | txlineElectricalLength

Design RF Chain Using RF Antenna Object

This example shows how to design an RF chain using an `rfantenna` object.

RF Antenna Transmitter and Receiver Elements

Create an RF antenna transmitter element. The properties of the RF antenna transmitter are as shown.

```
Txant = rfantenna
```

```
Txant = 
  rfantenna: Antenna element

         Name: 'Antenna'
         Type: 'Transmitter'
         Gain: 0
            Z: 50
     NumPorts: 1
    Terminals: {'p1+'  'p1-'}
```

Create an RF antenna receiver element. The properties of the RF antenna receiver are as shown.

```
Rxant = rfantenna("Type","Receiver")
```

```
Rxant = 
  rfantenna: Antenna element

         Name: 'Antenna'
         Type: "Receiver"
         Gain: 0
            Z: 50
       TxEIRP: -30
     PathLoss: 0
     NumPorts: 1
    Terminals: {'p1+'  'p1-'}
```

The `TxEIRP` property of the RF antenna receiver element is the effective isotropic radiated power (EIRP) from the transmitter to the receiver. This is essentially a power value falling on the receiver. The EIRP also takes into account the path loss. Path loss is specified in the `PathLoss` property of the receiver. Transmitter EIRP is calculated using this equation:

TxEIRP = Path Loss + Gain

Use RF Antenna Receiver Element in RF Budget Chain

Calculate the RF budget of a series of RF elements by tying this command at the command line at the input frequency of 2.1 GHz, available input power of –30 dBm, and bandwidth of 10 MHz. The RF antenna receiver element must be the first element in the RF chain.

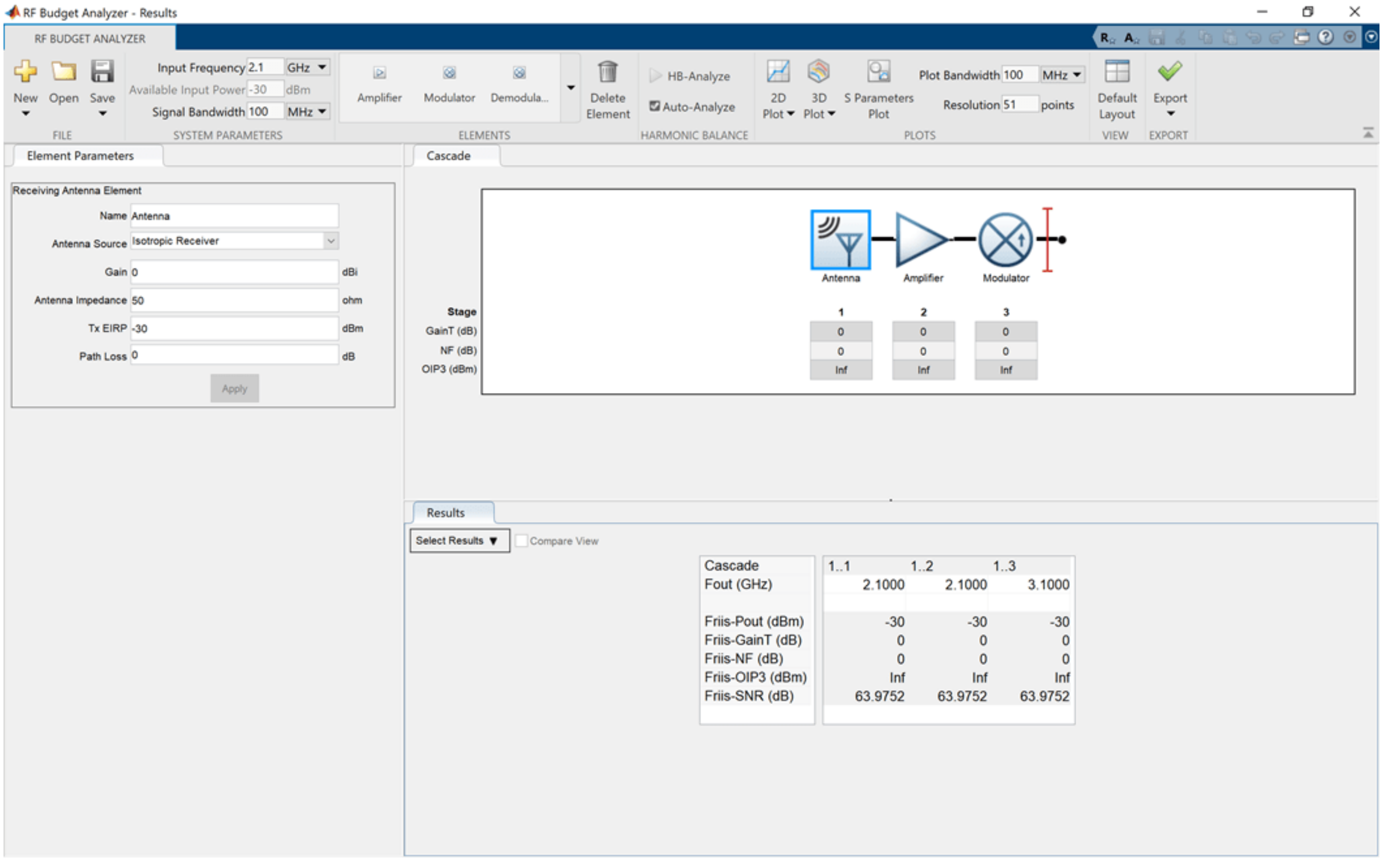
```
a = amplifier;
m = modulator;
rfb = rfbudget([Rxant a m],2.1e9,-30,100e6)
```

```
Warning: Available Input Power will be ignored because of the receiver.
Warning: Available Input Power will be ignored because of the receiver.
rfb =
  rfbudget with properties:

               Elements: [1x3 rf.internal.rfbudget.Element]
         InputFrequency: 2.1 GHz
    AvailableInputPower: -30 dBm
        SignalBandwidth: 100 MHz
                 Solver: Friis
             AutoUpdate: true

   Analysis Results
        OutputFrequency: (GHz) [  2.1    2.1    3.1]
            OutputPower: (dBm) [  -30    -30    -30]
         TransducerGain: (dB)  [    0      0      0]
                     NF: (dB)  [    0      0      0]
                   IIP2: (dBm) []
                   OIP2: (dBm) []
                   IIP3: (dBm) [  Inf    Inf    Inf]
                   OIP3: (dBm) [  Inf    Inf    Inf]
                    SNR: (dB)  [63.98  63.98  63.98]
```

MATLAB displays the warning message because the available input power is replaced by `TxEIRP`, the EIRP of the of the receiver.

Open the **RF Budget Analyzer** app to visualize RF budget chain using the `show` command at the command line.

```
show(rfb)
```



The **Available Input Power** option is dimmed in the app when you add a receiver element. By default, the **TxEIRP** property has the same value as the Available Input Power option.

Use RF Antenna Transmitter Element in RF Budget Chain

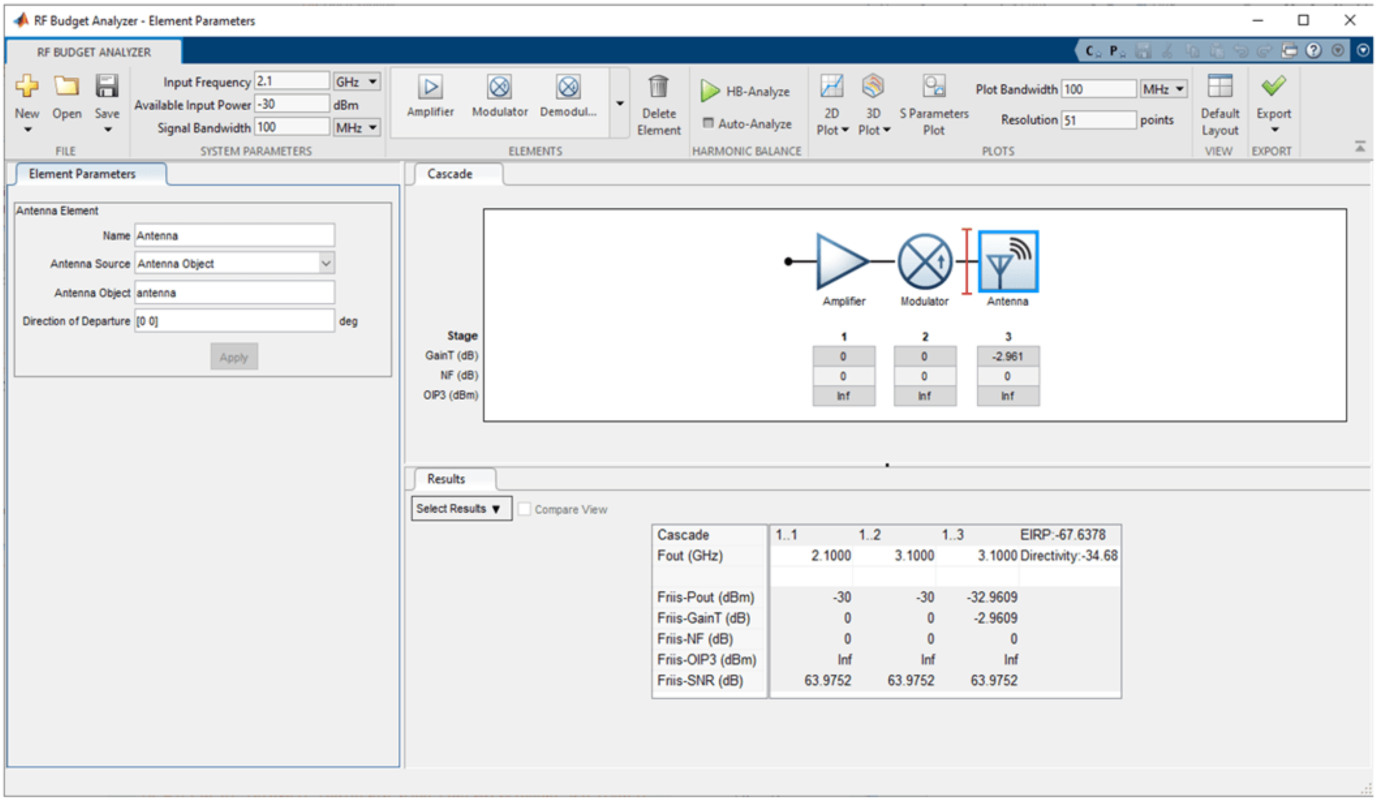
Calculate the RF budget of a series of RF elements by using the RF antenna transmitter as one of the elements. The RF antenna transmitter element must be the last element in the RF chain.

```
a = amplifier;
m = modulator;
rfb = rfbudget([a m Txant],2.1e9,-30,100e6)
```

```
rfb =
  rfbudget with properties:

               Elements: [1x3 rf.internal.rfbudget.Element]
         InputFrequency: 2.1 GHz
    AvailableInputPower: -30 dBm
        SignalBandwidth: 100 MHz
                 Solver: Friis
             AutoUpdate: true

   Analysis Results
        OutputFrequency: (GHz) [  2.1     3.1     3.1]
            OutputPower: (dBm) [  -30     -30     -30]
         TransducerGain: (dB)  [    0       0       0]
                     NF: (dB)  [    0       0       0]
                   IIP2: (dBm) []
                   OIP2: (dBm) []
                   IIP3: (dBm) [  Inf     Inf     Inf]
                   OIP3: (dBm) [  Inf     Inf     Inf]
                    SNR: (dB)  [63.98   63.98   63.98]
                   EIRP: (dBm) [  -30]
            Directivity: (dBi) [    0]
```

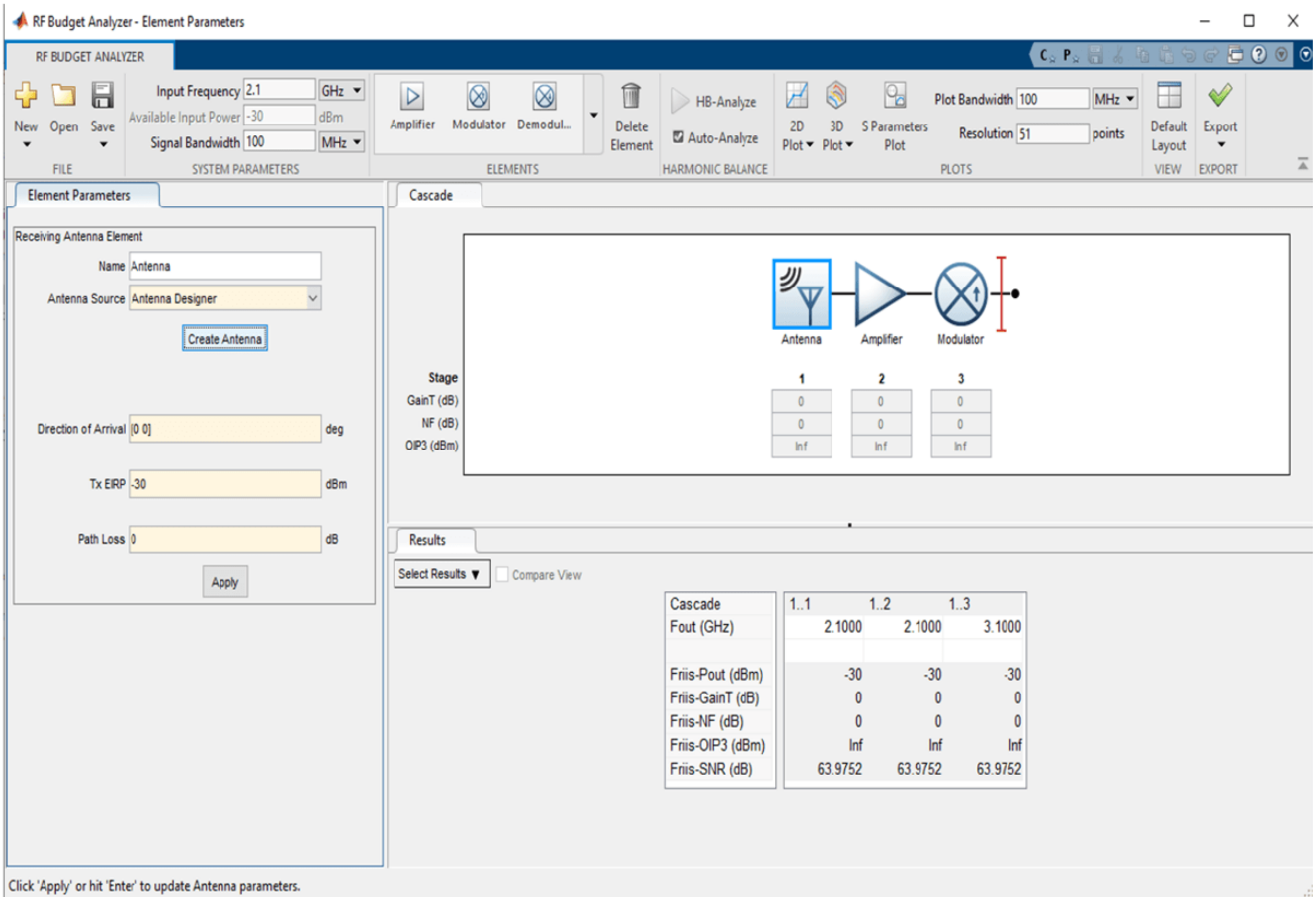
Open the **RF Budget Analyzer** app to visualize RF budget chain using the `show` command at the command line.

```
show(rfb)
```



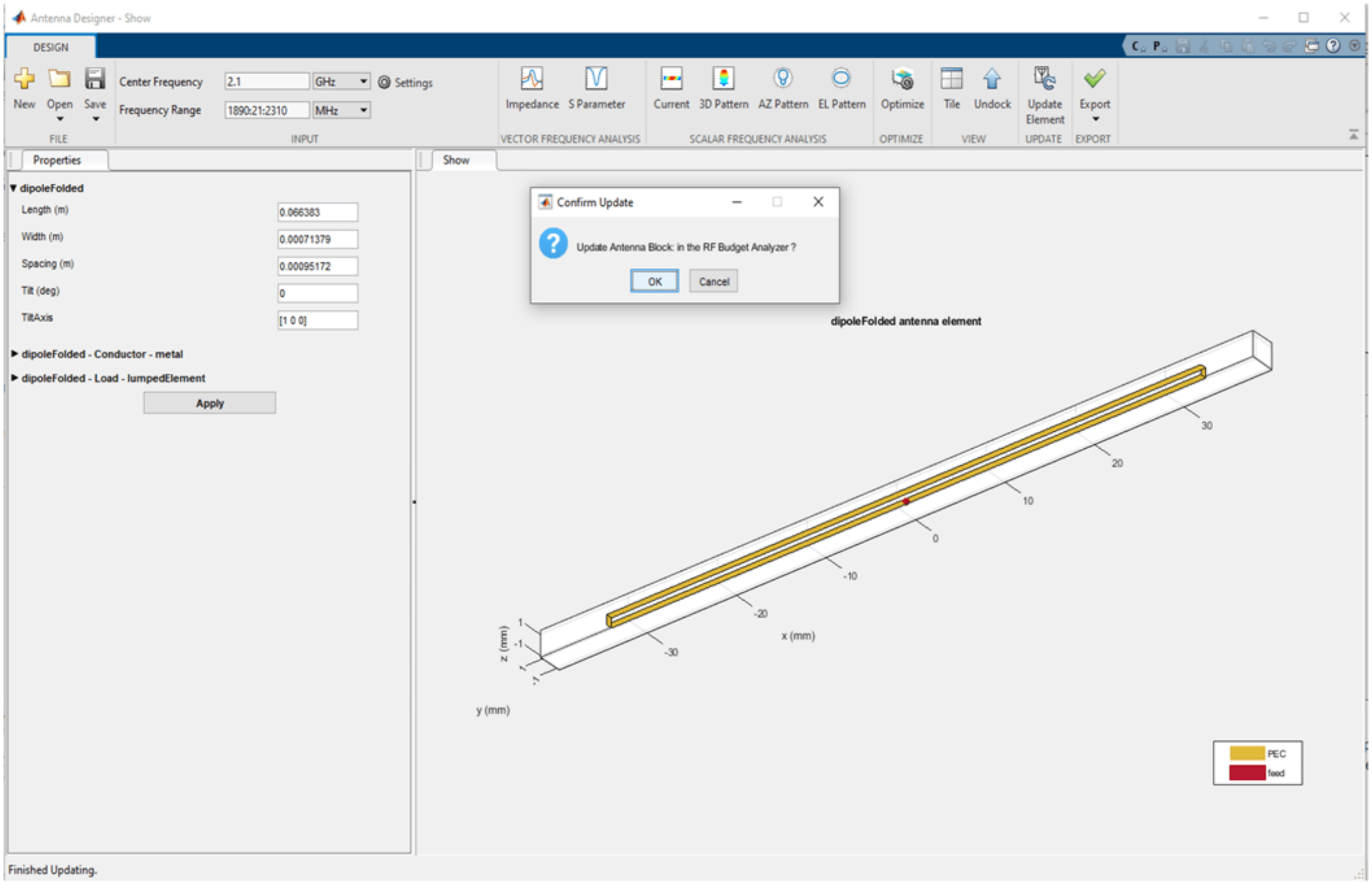
Design RF Receiver Antenna Using Antenna Designer App

By default, the **Antenna Source** property of the RF antenna receiver element in the app set to `Isotropic Receiver`. Set the **Antenna Source** to `Antenna Designer`, which gives you the option of designing the antenna in the **Antenna Designer** app of the **Antenna Toolbox.** To use the **Antenna Designer** app or the **Antenna Object**, you need Antenna Toolbox™ license.

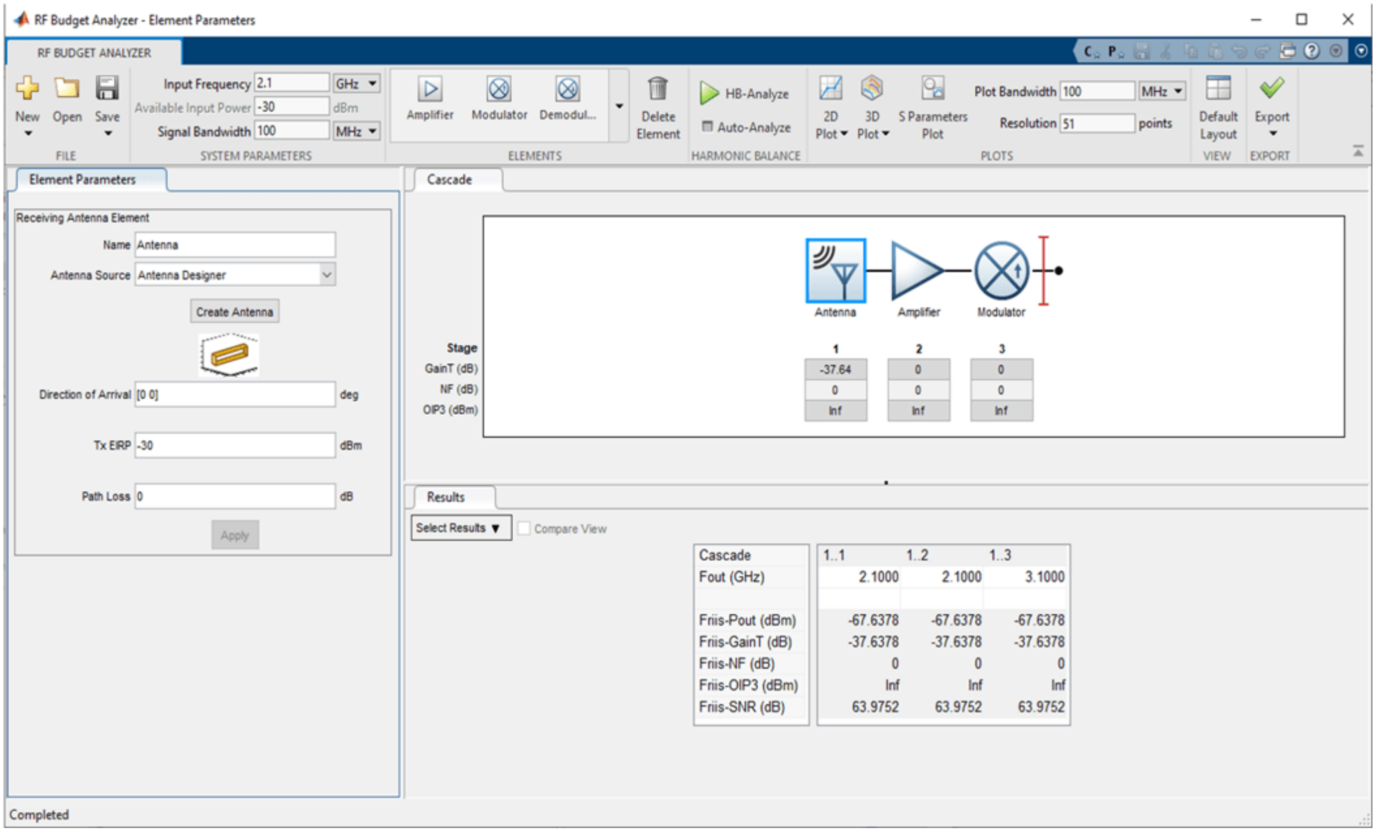


Click **Create Antenna** in the **Element Parameters** pane. The **Antenna Designer** app opens. Select **New** to explore the antenna library. This example uses a `dipoleFolded` antenna element with a center frequency of `2.1` GHz. Select the `Folded` element from the **Antenna Gallery**, set the **Design Frequency** to `2.1` MHz, and click **Accept**.

Select **Update Element** to update the `Antenna` element in the **RF Budget Analyzer** app. Select **OK** in the Confirm Update dialog box.



After importing the design to the **RF Budget Analyzer** app, select **Apply**.

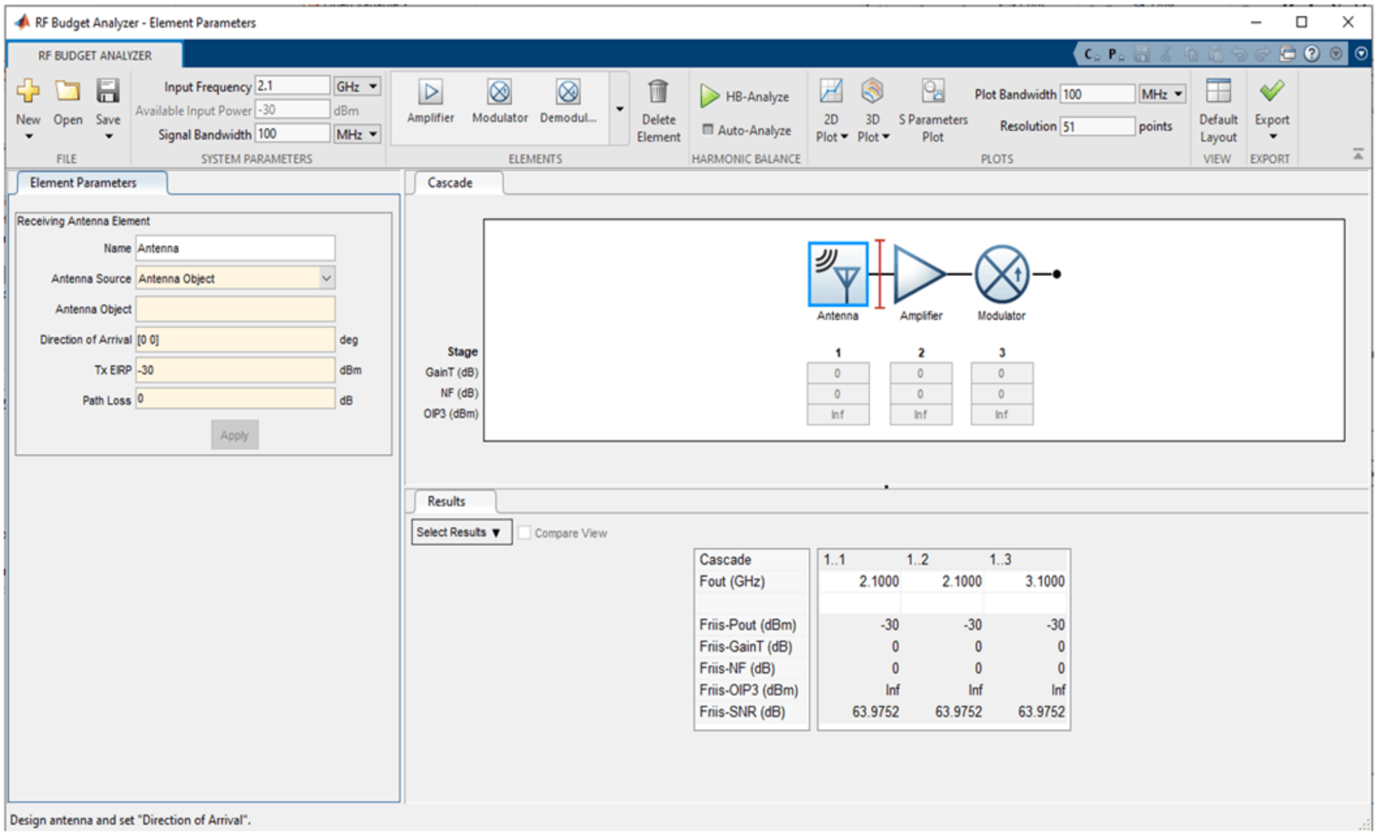


Design RF Receiver Antenna Using Antenna Object

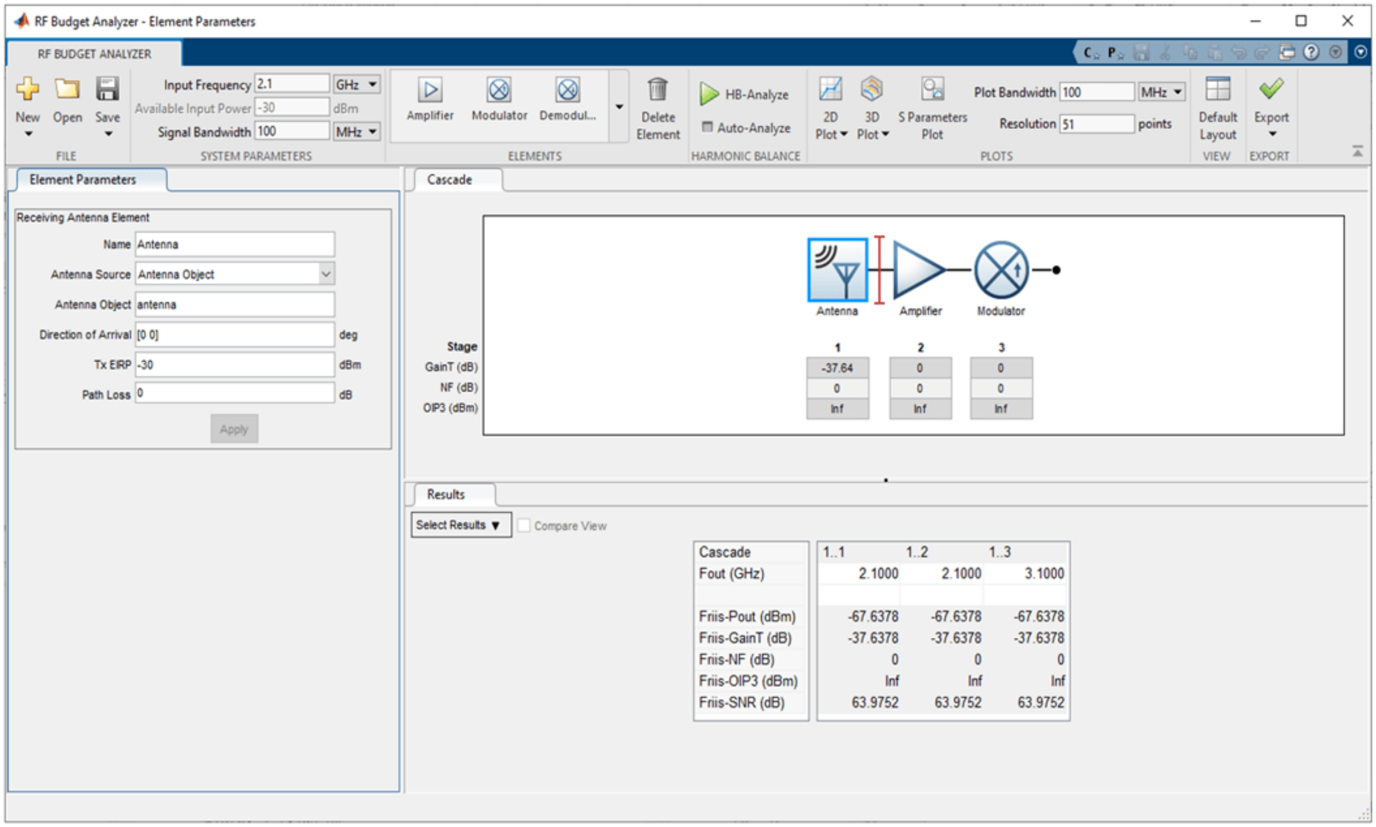
Create a folded dipole antenna at 2.1 GHz in the MATLAB workspace.

```
antenna = design(dipoleFolded,2.1e9)
```

Set the **Antenna Source** to `Antenna Object` in the **RF Budget Analyzer** app to design an antenna using antenna object**.**

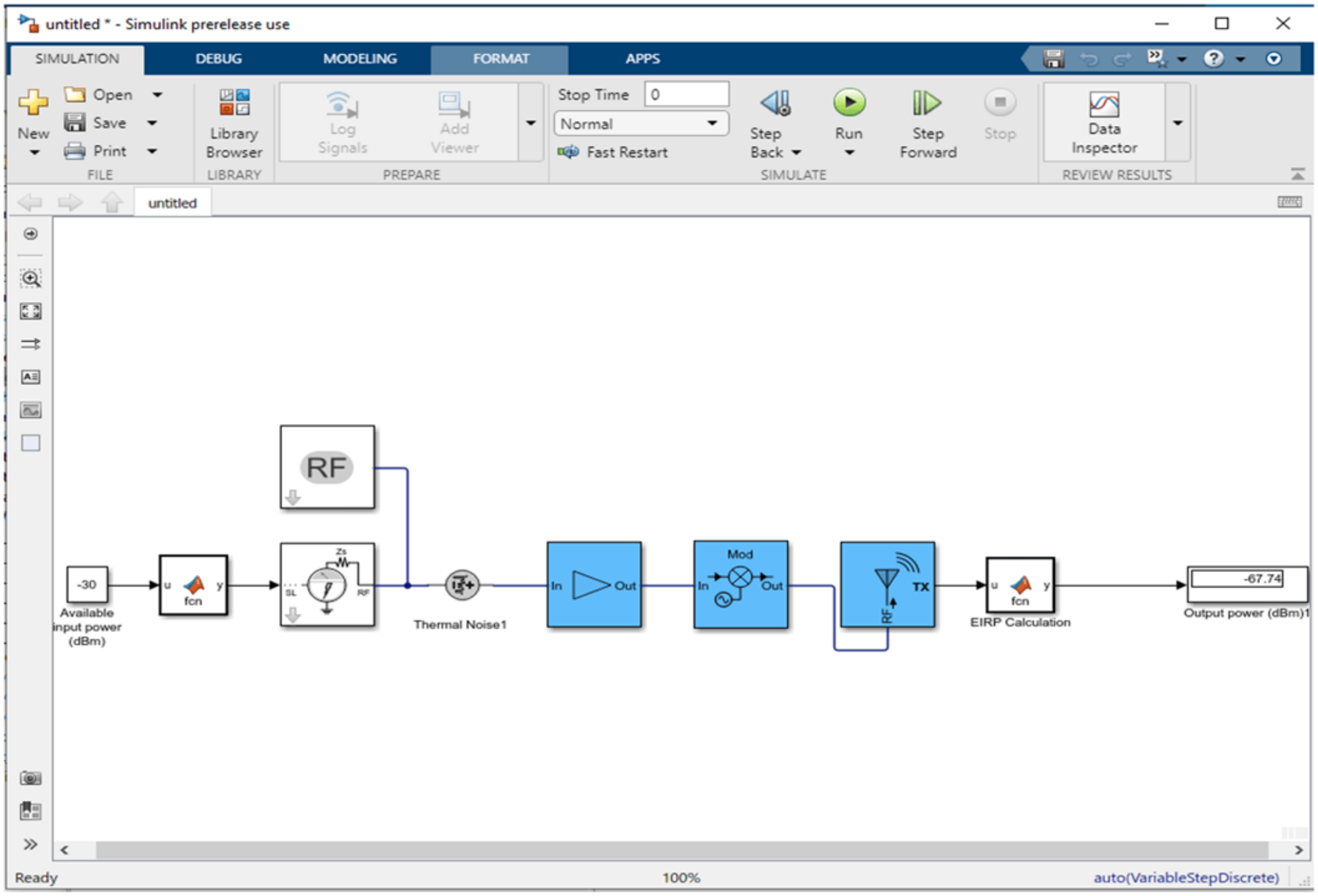


Select the antenna in the **Antenna Object** field and click **Apply**.

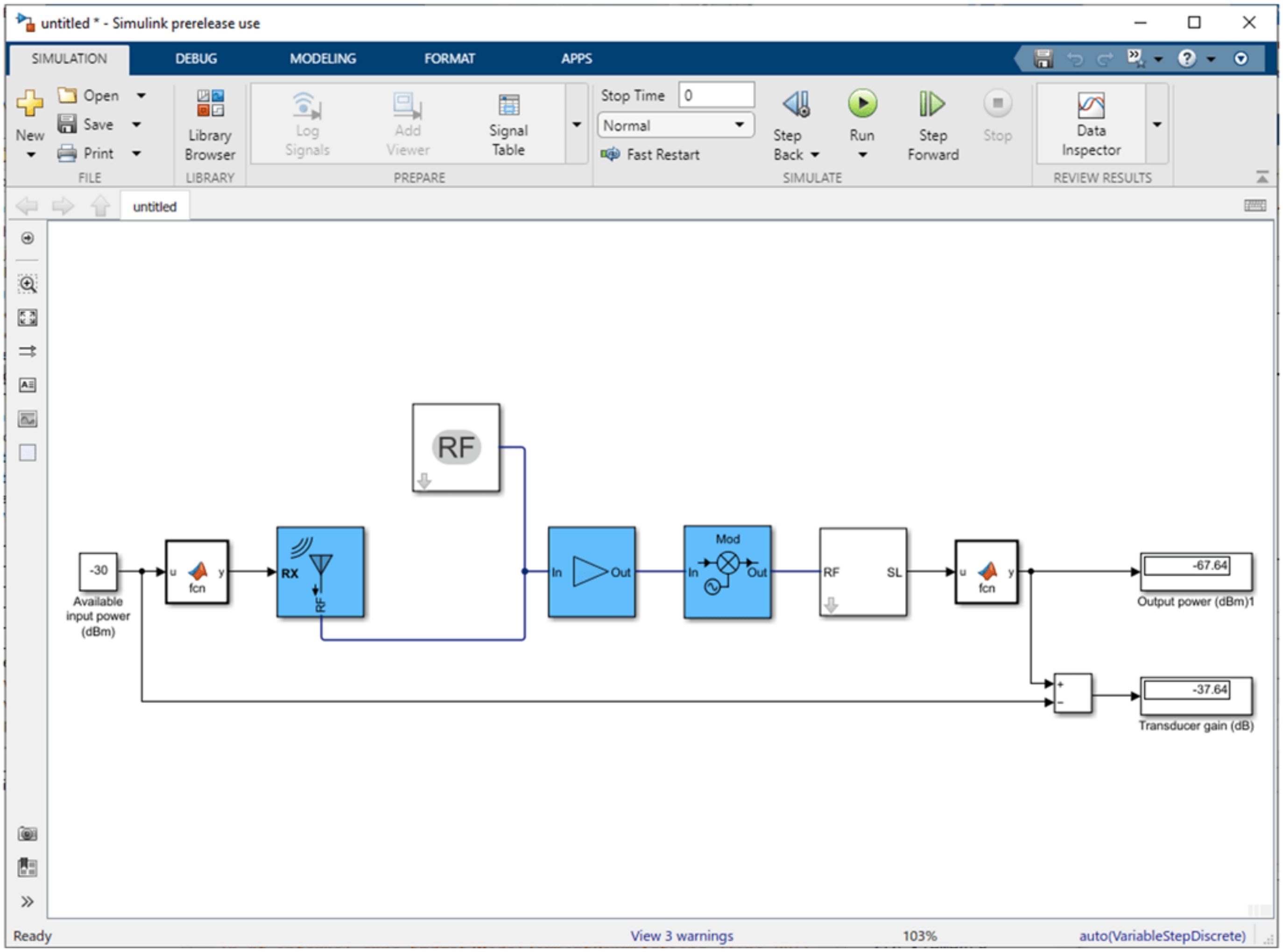


Export Your RF Transmitter and Receiver Chain

Export the RF transmitter chain to RF Blockset™ by selecting the **Export to Blockset** option in the **Export** drop-down list.



Now export the RF receiver chain to RF Blockset by selecting the **Export to Blockset** option in the **Export** drop-down list.



The exported RF Blockset model shows that the RF transmitter chain does not have `TxEIRP` and `PathLoss` as input properties, and similarly, the RF receiver chain does not have `EIRP` and `Directivity` as its outputs.

When you export an RF receiver, the exported RF Blockset model only shows the values of the `Gain` and `Impedance` parameters, whereas, in case of an RF transmitter, all the transmit antenna parameters of an `rfantenna` object are exported to the model.

See Also

RF Budget Analyzer

Related Examples

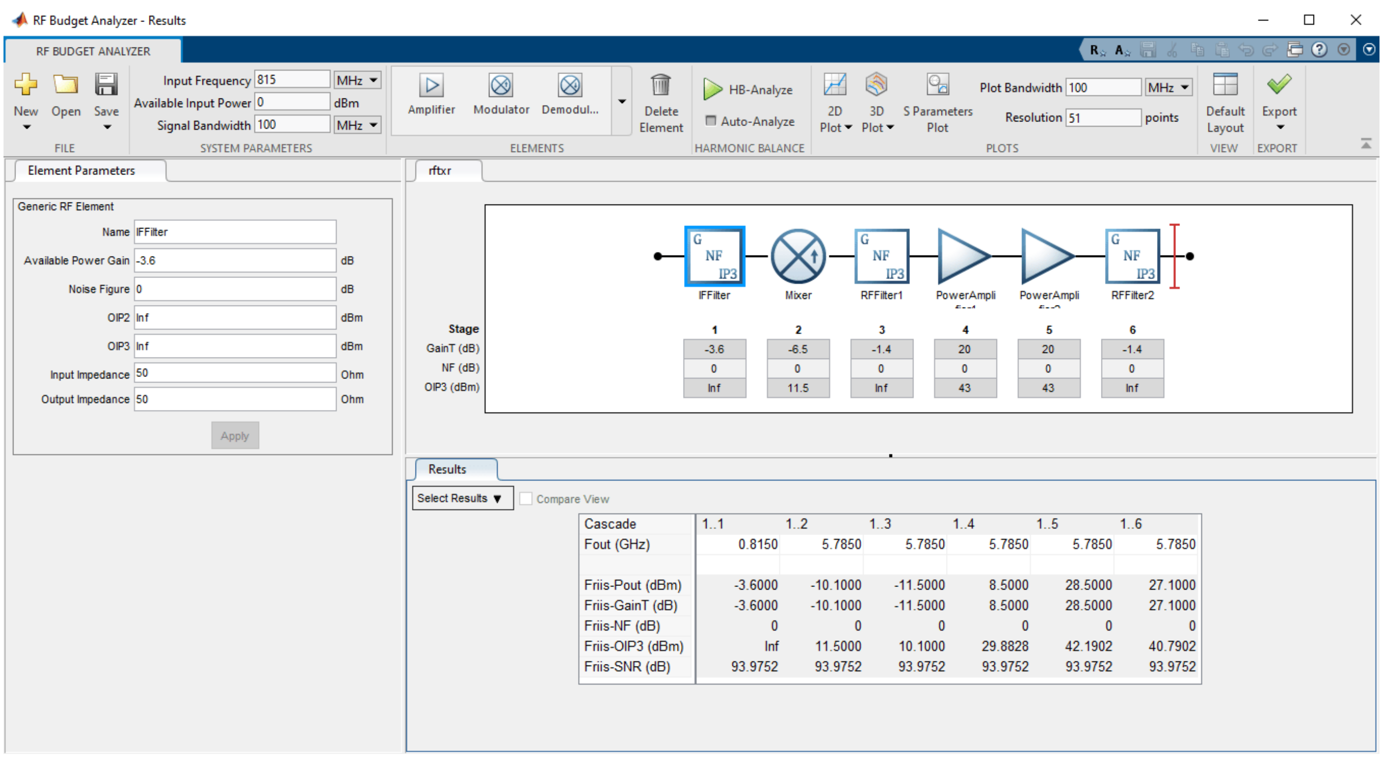
- “Superheterodyne Receiver Using RF Budget Analyzer App” on page 6-2
- “RF Budget Harmonic Balance Analysis of Low-IF Receiver, IP2 and NF” on page 6-226

Circuit Envelope Simulation at MATLAB Command Line

You can perform circuit envelope simulation on your RF system either in Simulink® or in MATLAB®. This example shows how to perform circuit envelope simulation at the MATLAB command line using the `rfsystem` System™ object. The circuit envelope simulation technique allows you to perform time-varying harmonic balance analysis of an RF system around multiple carrier frequencies. You can use this technique for simulating RF systems operating with narrowband signals. These narrowband signals are simulated around carrier frequencies that are either harmonic multiples of fundamental carrier frequencies or intermodulation frequencies that result from mixing products and nonlinear effects. This added level of fidelity allows you to predict out-of-band spectral regrowth or take into account the impact of interferers. For more information, see “Circuit Envelope Basics” (RF Blockset).

In this example, you first design your RF system using an `rfbudget` object or the **RF Budget Analyzer** app and then you will export your system to an `rfsystem` System object. You then provide provide a stimulus to this System object, set its sample rate, and view the associated RF Blockset™ model. You can also modify this model to increase the level of modeling fidelity.

Design an RF system. This example uses an RF transmitter designed in the “RF Transmitter System Analysis” example. Type `rfBudgetAnalyzer("rftxr.mat")` command at the command line to visualize the RF system in the **RF Budget Analyzer** app.



In the app, select **Export** and then select **RF system** to export this RF transmitter system to a `rfsystem` System object. This imported `rfsystem` System object is stored in a variable, `rfs`. For this example, the `rfs` variable is saved in `rfsystem_txmr.mat` data file using `open_system (rfs)` and `save_system(rfs,'rfsystem_txmr')` commands.

Load the `rfs` variable from the `rfsystem_txmr.mat` data file.

```
S = load('rfsystem_txmr.mat','rfs');
rfs = S.rfs;
```

Specify an input time-domain signal for the RF system.

```
in = [1e-3*ones(8,1); zeros(8,1)] .* ones(1,10);
in = in(:);
```

Calculate the output time-domain signal of the RF system.

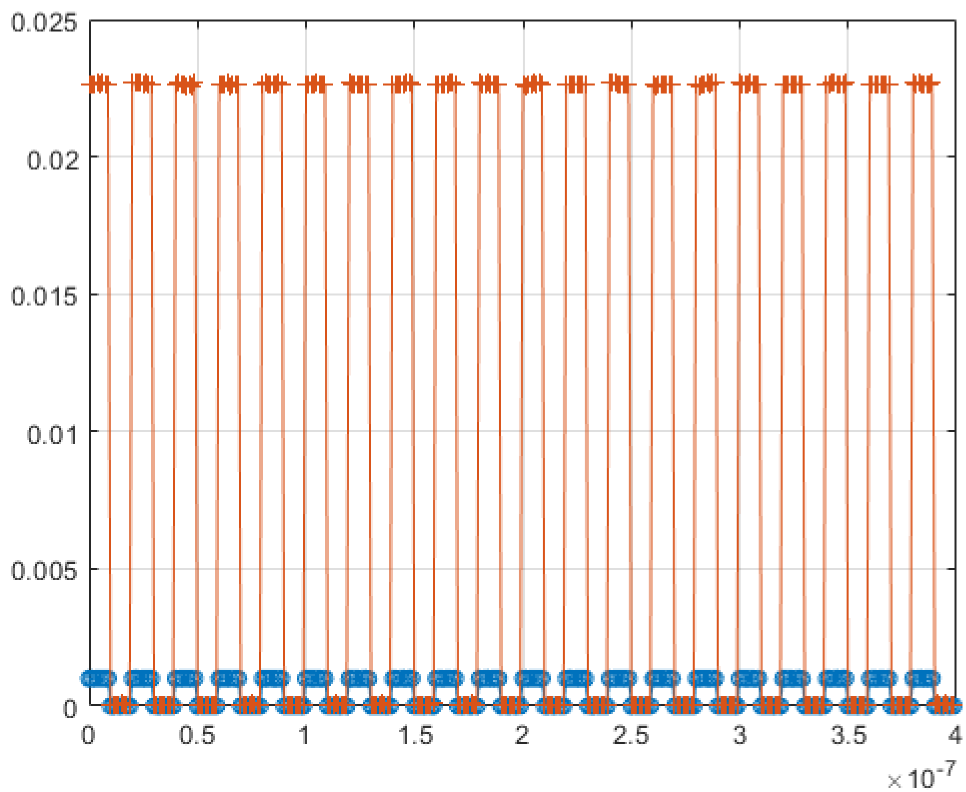
```
out = rfs(in);
out = [out; rfs(in)];
```

Specify the sample time of the RF system.

```
t = rfs.SampleTime*(0:length(out)-1);
```

Plot the simulated output.

```
plot(t,[in; in],'-o',t,abs(out),'-+')
grid on
```

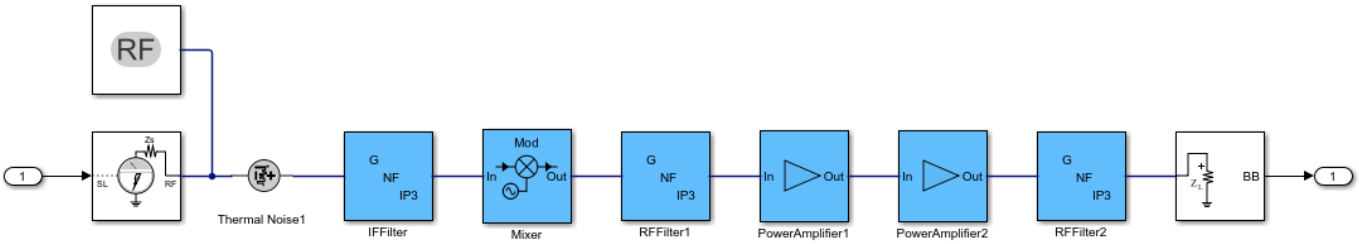


Release system resources and turn off fast restart.

```
release(rfs)
```

Open an RF Blockset model of the RF system using the `open_system` object function.

```
open_system(rfs)
```



You can also expand your system to a multiple input single output (MISO) or a single input multiple output (SIMO) system. For more information, see “Model MISO Receiver and SIMO Transmitter Systems”.

See Also

`open_system` | `load_system`

Related Examples

- “RF Receiver Modeling for LTE Reception” (RF Blockset)

Design Matching Network Using Lumped Components from Modelithics Library

This example shows you how to design a matching network with real-world lumped components from Modelithics SELECT+ Library™. The example also shows you how to analyze this matching network by matching it to a reference antenna and comparing its performance to a matching network with ideal lumped elements. The reference antenna in the example is an inset-fed microstrip patch antenna with an operating frequency of 2.4 GHz. This allows you to consider a matching frequency of 2.35 GHz.

The presence of parasitics changes the response of the matching network with real-world lumped components. The non-ideal behavior results from variations in the material properties of the substrate and packing material, solder and pad properties, and orientation.

Create Inset-fed Patch Antenna

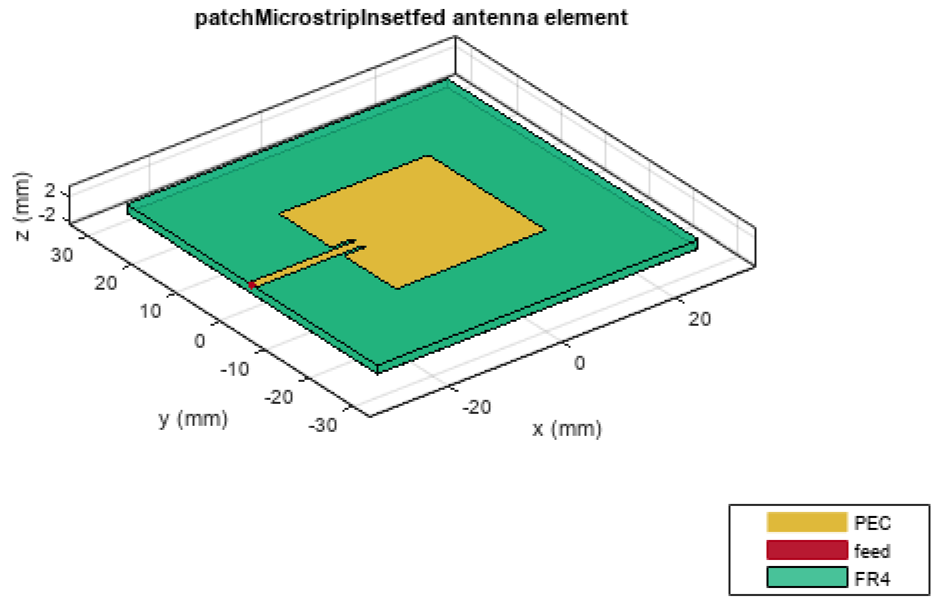
Create simple inset-fed patch antenna using the `design` function from Antenna Toolbox™. Set the antenna dimensions and use an FR4 substrate in this design.

```
antennaObject = design(patchMicrostripInsetfed, 2400*1e6);
antennaObject.Length = 0.0265;
antennaObject.Width = 0.0265;
antennaObject.Height = 0.0014;
antennaObject.Substrate.Name = 'FR4';
antennaObject.Substrate.EpsilonR = 4.8;
antennaObject.Substrate.LossTangent = 0.026;
antennaObject.Substrate.Thickness = 0.0014;
antennaObject.FeedOffset = [-0.02835, 0];
antennaObject.StripLineWidth = 0.0016223;
antennaObject.NotchLength = 0.0037853;
antennaObject.NotchWidth = 0.002839;
antennaObject.GroundPlaneLength = 0.0567;
antennaObject.GroundPlaneWidth = 0.0567;
```

Visualize Inset-Fed Patch Antenna

Use the `show` function to visualize the structure of this patch antenna.

```
figure;
show(antennaObject)
```



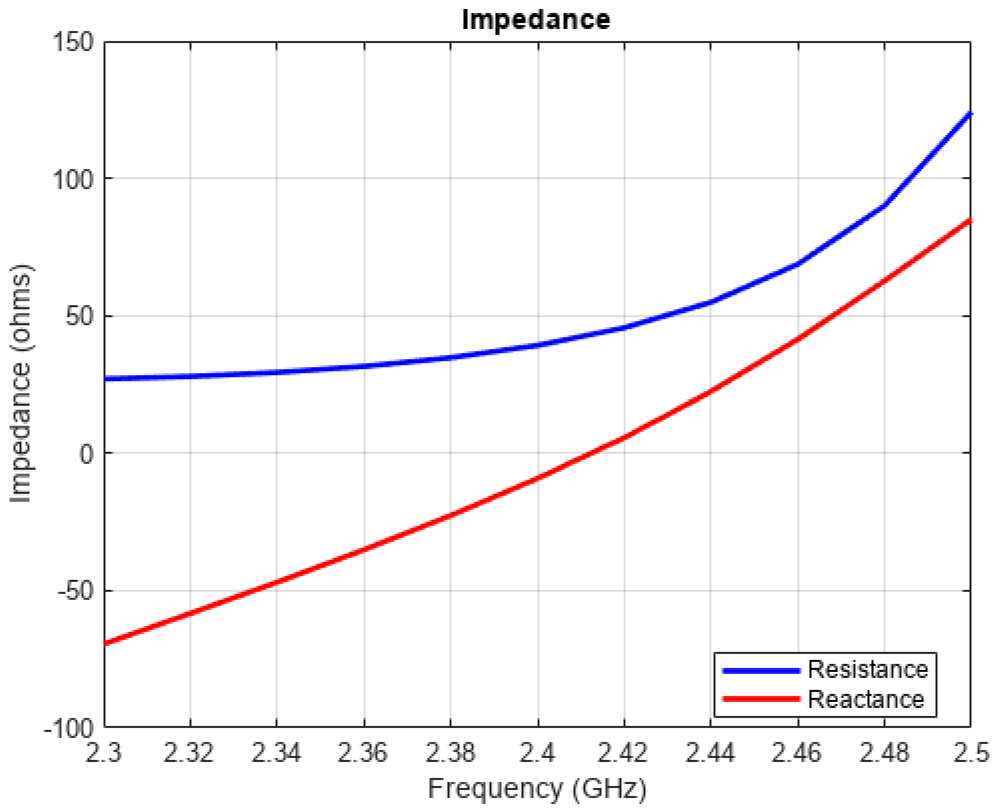
Analyze Inset-Fed Patch Antenna

Perform full-wave analysis on the inset-fed patch antenna over 2.3–2.5 GHz and save the S-parameter results to a MAT file. Load the MAT file to view the impedance and reflection coefficient response.

```
load InsetPatchAntenna.mat
plotFrequency = 2400*1e6;
freqRange = linspace(2.3e9, 2.5e9, 11);
```

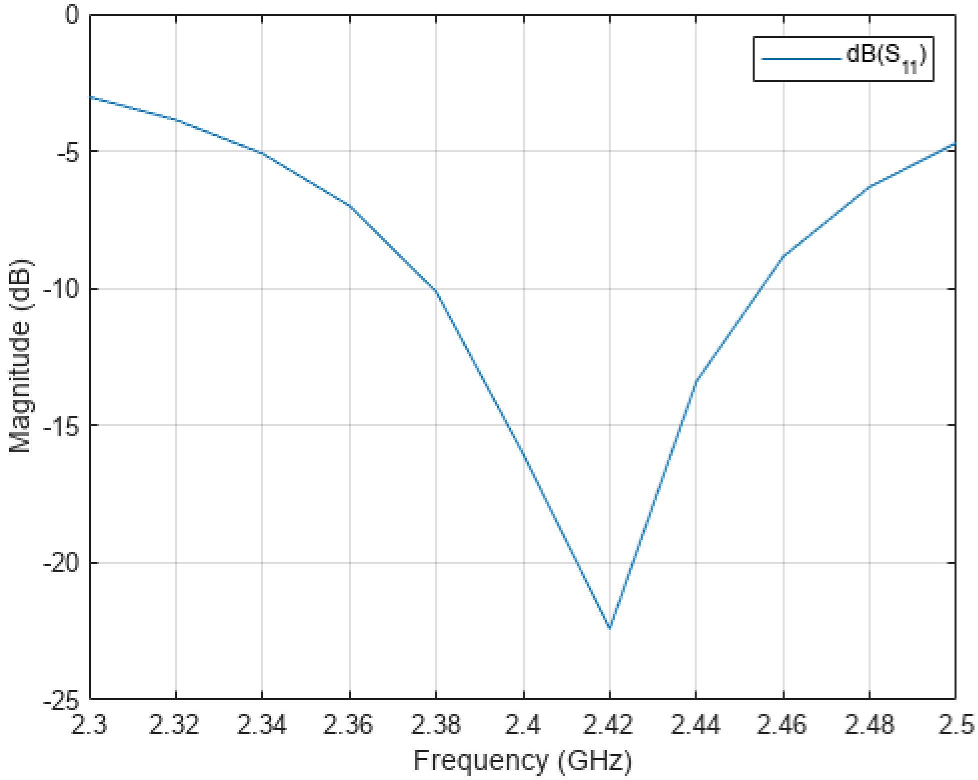
Plot the impedance of the inset-fed patch antenna.

```
figure;
impedance(antennaObject, freqRange);
```



Plot the reflection coefficient response of the inset-fed patch antenna.

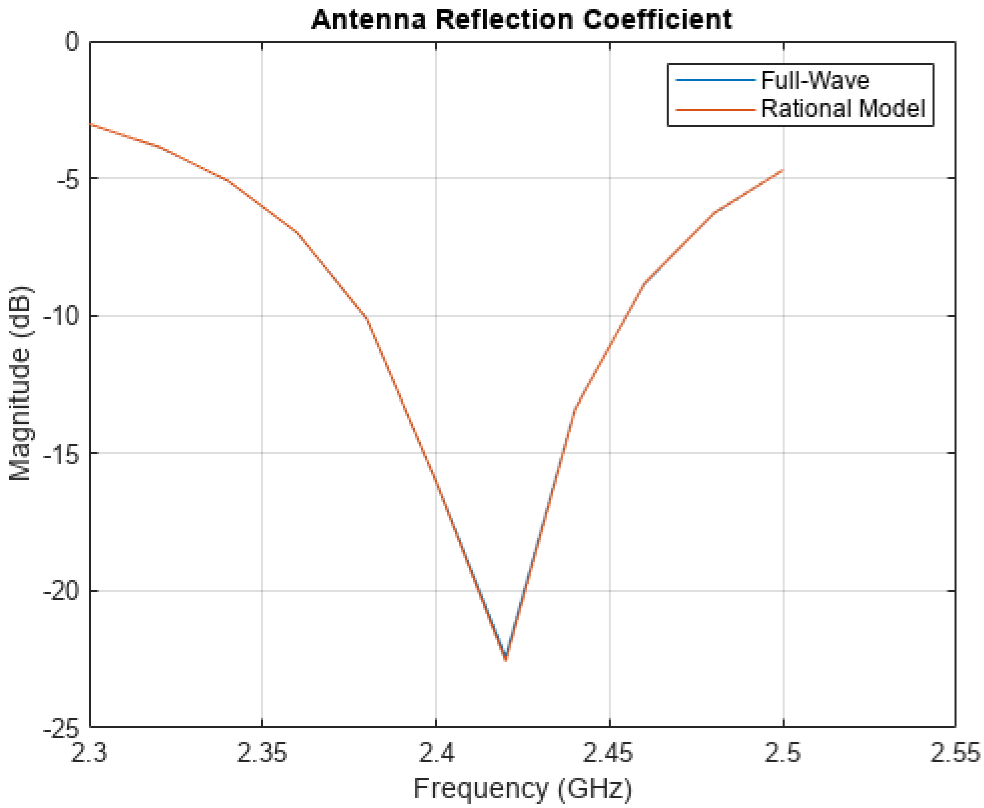
```
figure;
s = sparameters(antennaObject, freqRange);
rfplot(s)
```



Build Rational Model

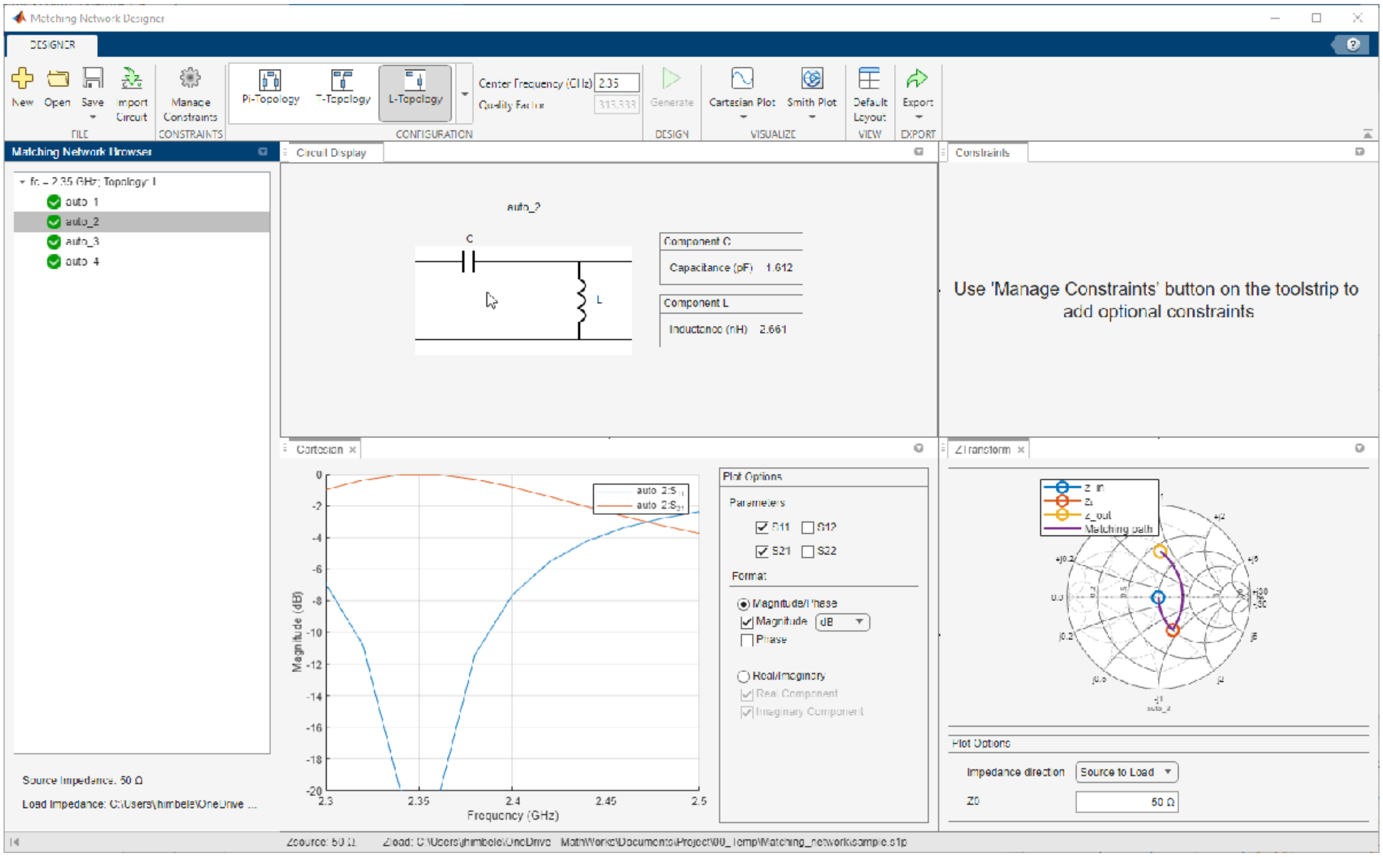
Build a rational model for the antenna S-parameter data. This allows you to refine the frequency points in the analysis range and not simulate the antenna again during full-wave analysis.

```
s_rat = rational(s);
[resp,~] = freqresp(s_rat,freqRange);
hold on
plot(freqRange/1e9,20*log10(abs(resp)))
title('Antenna Reflection Coefficient')
legend('Full-Wave','Rational Model')
```



Design Matching Network

Load the S-parameter data from the workspace to the **Matching Network Designer** app. Set the matching frequency to 2.35 GHz and the network topology to L-Topology. Once the app generates the networks, select series C, shunt L from the network list and export the network. The app saves the network to MAT file.



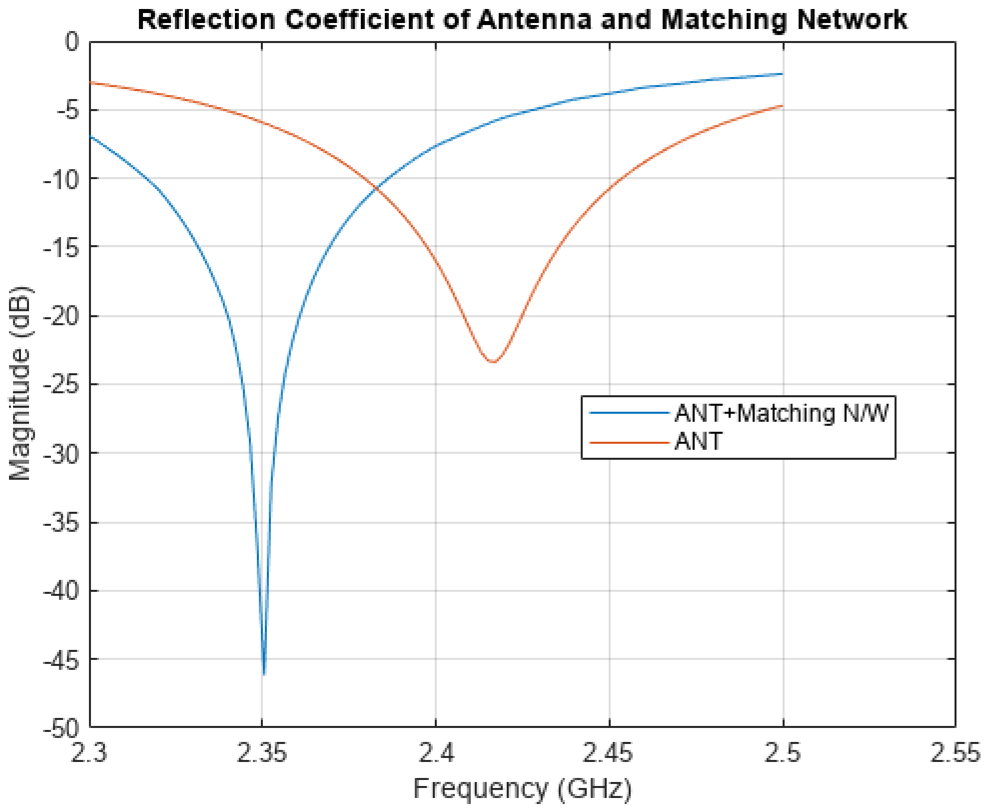
To build the matching network, convert the S-parameter data to an `nport` object and add it to a circuit. Assign ports to the circuit for RF analysis.

```
ant = nport(s);
load mnapp_LTopo_CserLsh.mat
ckt_lumped = circuit;
add(ckt_lumped,[1 2 0 0],mnckt)
add(ckt_lumped,[2 0],ant)
setports(ckt_lumped,[1 0])
```

Perfrom S-Parameters Frequency Sweep

Select 100 points in the analysis frequency range and analyze the matching network response with the antenna S-parameters as the load. Overlay the antenna-only port reflection coefficient over this response. This shifts the antenna response in the lower band to 2.35 GHz.

```
freqRange = linspace(2.3e9, 2.5e9, 100);
lumped_s = sparameters(ckt_lumped,freqRange);
[resp,freq] = freqresp(s_rat,freqRange);
figure
rfplot(lumped_s,1,1)
hold on
plot(freqRange/1e9,20*log10(abs(resp)))
legend('ANT+Matching N/W','ANT','Location','Best')
title('Reflection Coefficient of Antenna and Matching Network')
```



Build Matching Network with Non-Ideal Lumped Component Models

Select the non-ideal lumped components from Modelithics Select+ Library. You must have the Modelithics Select+ Library license to run the following code.

Create RF Component using Modelithics SELECT+ Library

Set up the Modelithics Select+ Library by specifying the full path to the library.

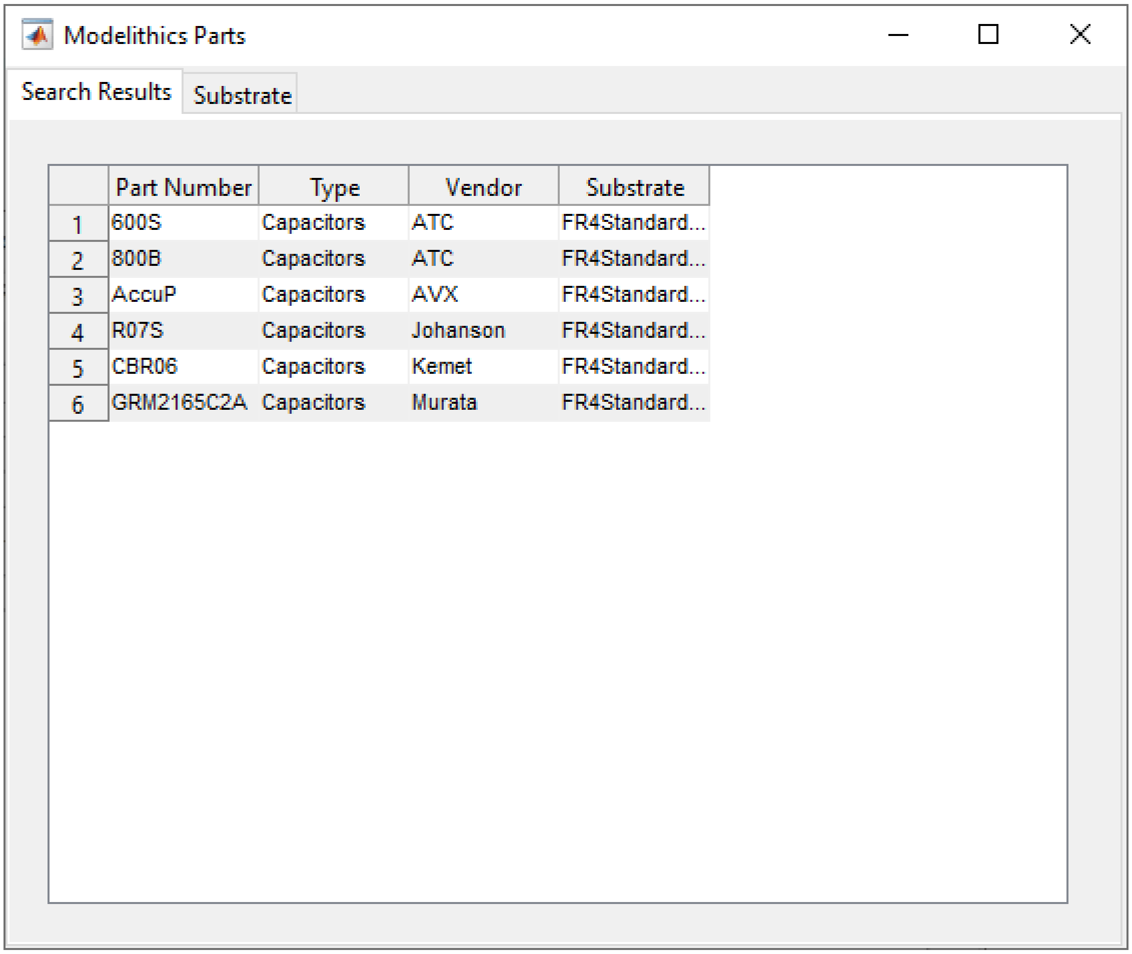
```
mdlxSetup('C:\mdlx_library\SELECT')
```

Create the Modelithics library object.

```
mdlx = mdlxLibrary;
```

Search the library for a 1.6117 pF capacitor mounted on a 59 mil FR4 substrate.

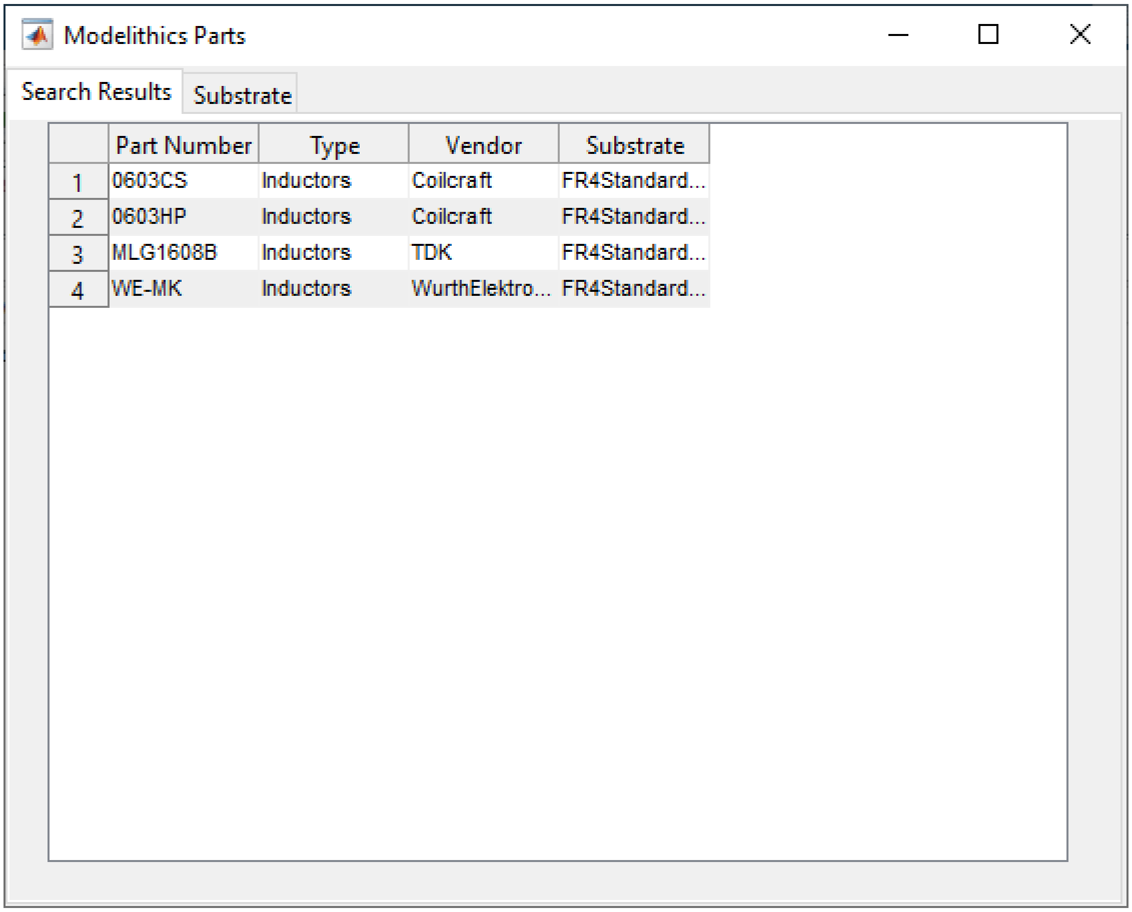
```
search(mdlx,'FR4Standard59mil',Type='Capacitors',Value=1.6117e-12)
```



	Part Number	Type	Vendor	Substrate
1	600S	Capacitors	ATC	FR4Standard...
2	800B	Capacitors	ATC	FR4Standard...
3	AccuP	Capacitors	AVX	FR4Standard...
4	R07S	Capacitors	Johanson	FR4Standard...
5	CBR06	Capacitors	Kemet	FR4Standard...
6	GRM2165C2A	Capacitors	Murata	FR4Standard...

Search the library for a 2.6611 nH inductor mounted on a 59 mil FR4 substrate.

```
search(mdlx,'FR4Standard59mil',Type='Inductors',Value=2.6611e-9)
```



Create an array of Modelithics capacitors.

```
cList = search(mdlx,'FR4Standard59mil',Type='Capacitors',Value=1.6117e-12);
```

Create an array of Modelithics inductors.

```
lList = search(mdlx,'FR4Standard59mil',Type='Inductors',Value=2.6611e-9);
```

Create Matching Network with Modelithics Components

Create a matching network with L-Topology, series C, and shunt L using the first element in the array of Modelithics capacitors and inductors. Most Modelithics lumped components have two ports.

```
mdlxckt = circuit;
add(mdlxckt,[1 2 0 0],cList(1));
add(mdlxckt,[2 0 0 0],lList(1));
setports(mdlxckt,[1 0],[2 0]);
```

As with the matching network with ideal lumped components, add the matching network with the Modelithics lumped components and the S-parameters of the inset-fed patch antenna to a circuit. Assign ports to the circuit for RF analysis.

```
mdlxckt_lumped = circuit;
add(mdlxckt_lumped,[1 2 0 0],mdlxckt)
add(mdlxckt_lumped,[2 0],nport(s))
setports(mdlxckt_lumped,[1 0])
```

Analyze Matching Network and Antenna

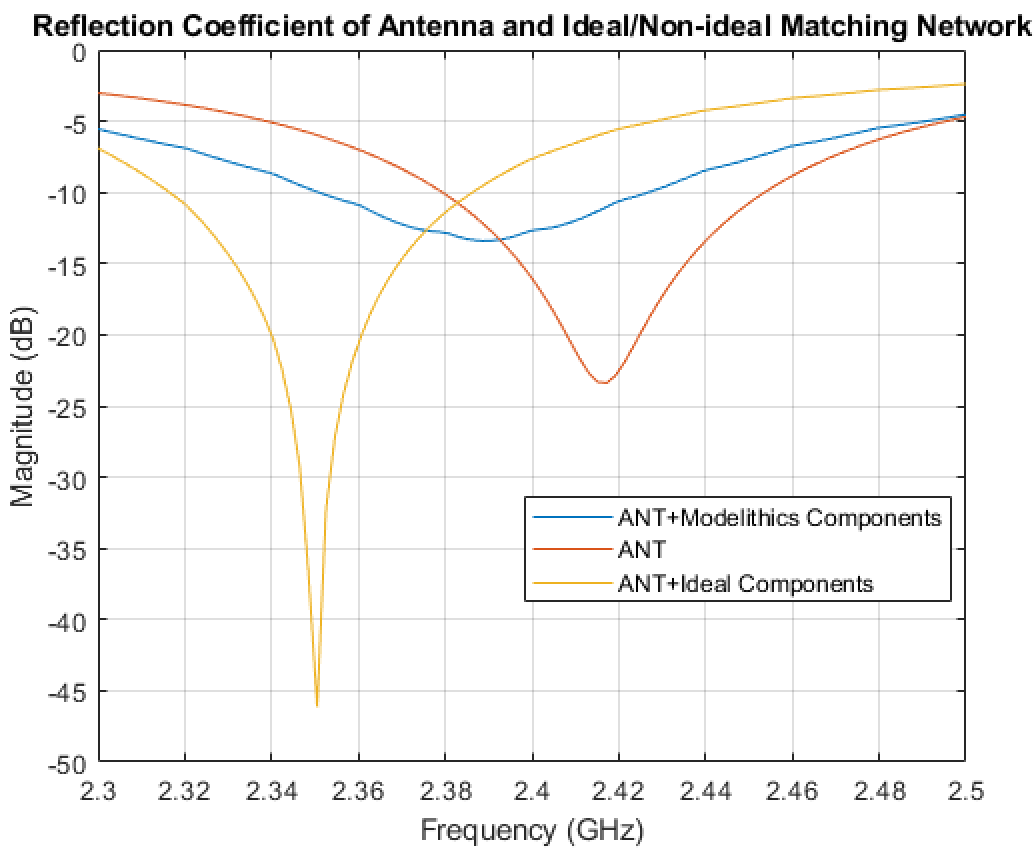
Analyze the matching network response with the antenna S-parameters as the load.

```
mdlxlumped_s = sparameters(mdlxckt_lumped,freqRange);
```

Compare Ideal and Non-Ideal Reflection Coefficients

Overlay the reflection coefficient from the antenna S-parameters with the ones using ideal matching network and the one with real-world components from the Modelithics Select+ Library.

```
figure
rfplot(mdlxlumped_s,1,1)
hold on
plot(freqRange/1e9,20*log10(abs(resp)))
rfplot(lumped_s,1,1)
legend('ANT+Modelithics Components','ANT','ANT+Ideal Components','Location','Best')
title('Reflection Coefficient of Antenna and Ideal/Non-ideal Matching Network')
```



See Also

`mdlxLibrary` | `search` | `mdlxPart` | `mdlxSetup`

Related Examples

- "Impedance Matching of Small Monopole Antenna" on page 6-167