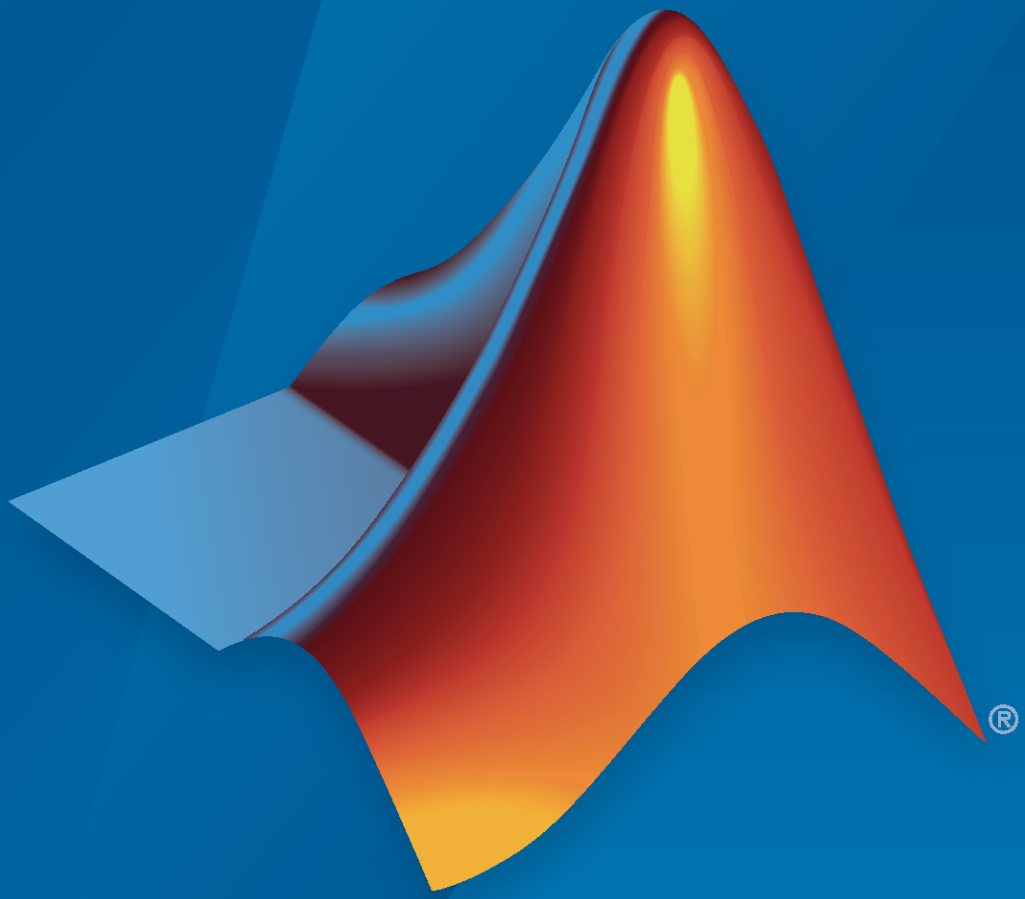
RF Toolbox™

Reference



MATLAB®

R2022b

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™ Reference

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)

Contents

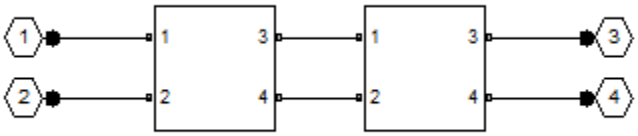
Objects — Alphabetical List

rfckt.cascade

Cascaded network

Description

Use the `cascade` object to represent cascaded networks of RF objects that are characterized by the components that make up the individual network. The following figure shows the configuration of a pair of cascaded networks.



Creation

Syntax

```
h = rfckt.cascade
h = rfckt.cascade('Ckts',value)
```

Description

`h = rfckt.cascade` returns a cascaded network object whose properties all have their default values.

`h = rfckt.cascade('Ckts',value)` returns a cascaded network with elements specified in the name-value pair property `Ckts`.

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

This property is read-only.

Computed S-parameters, noise figure, OIP3, and group delay values. For more information, see “Algorithms” on page 1-4.

Data Types: `function_handle`

`Ckts` — Circuit objects in network
cell array of object handles

Circuit objects in network. All circuits must be 2-port. By default, this property is empty.

Data Types: `char`

`Name` — Name of cascaded network
`1-by-N` character array

This property is read-only.

Name of cascaded network.

Data Types: `char`

`nport` — Number of ports of cascaded network
positive integer

This property is read-only.

Number of ports of cascaded network. The default value is `2`.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

Create RF Circuit Cascade Network

Create a cascade network using `rfckt.cascade` with amplifier and transmission lines as elements.

```
amp = rfckt.amplifier('IntpType','cubic');
tx1 = rfckt.txline;
tx2 = rfckt.txline;
casccircuit = rfckt.cascade('Ckts',{tx1,amp,tx2})
```

```
casccircuit =
   rfckt.cascade with properties:

              Ckts: {1x3 cell}
             nPort: 2
    AnalyzedResult: []
              Name: 'Cascaded Network'
```

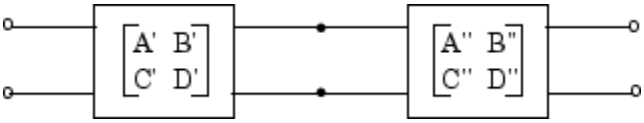
Algorithms

The `analyze` method computes the `AnalyzedResult` property using the data stored in the `Ckts` property as follows:

- The `analyze` method starts calculating the ABCD-parameters of the cascaded network by converting each component network's parameters to an ABCD-parameters matrix. The figure shows a cascaded network consisting of two 2-port networks, each represented by its ABCD matrix.

  The `analyze` method then calculates the ABCD-parameters matrix for the cascaded network by calculating the product of the ABCD matrices of the individual networks.

  The following figure shows a cascaded network consisting of two 2-port networks, each represented by its ABCD-parameters.

  The following equation illustrates calculations of the ABCD-parameters for two 2-port networks.

  $$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} A' & B' \\ C' & D' \end{bmatrix} \begin{bmatrix} A'' & B'' \\ C'' & D'' \end{bmatrix}$$

  Finally, `analyze` converts the ABCD-parameters of the cascaded network to S-parameters at the frequencies specified in the `analyze` input argument `freq`.

- The `analyze` method calculates the noise figure for an N-element cascade. First, the method calculates noise correlation matrices C_A' and C_A”, corresponding to the first two matrices in the cascade, using the following equation:

  $$C_A = 2kT \begin{bmatrix} R_n & \frac{NF_{\min} - 1}{2} - R_n Y_{opt}^* \\ \frac{NF_{\min} - 1}{2} - R_n Y_{opt} & R_n \left| Y_{opt} \right|^2 \end{bmatrix}$$

  where `k` is Boltzmann's constant, and `T` is the noise temperature in Kelvin.

  The method combines C_A' and C_A” into a single correlation matrix C_A using the equation

  $$C_A = C'_A + \begin{bmatrix} A' & B' \\ C' & D' \end{bmatrix} C''_A \begin{bmatrix} A' & B' \\ C' & D' \end{bmatrix}$$

By applying this equation recursively, the method obtains a noise correlation matrix for the entire cascade. The method then calculates the noise factor, `F`, from the noise correlation matrix of as follows:

$$F = 1 + \frac{z^{+} C_A z}{2kT\mathrm{Re}\{Z_S\}}$$

$$z = \begin{bmatrix} 1 \\ Z_S^* \end{bmatrix}$$

In the two preceding equations, Z_S is the nominal impedance, which is 50 ohms, and z^+ is the Hermitian conjugation of z.

- The `analyze` method calculates the output power at the third-order intercept point (OIP3) for an N-element cascade using the following equation:

$$OIP_3 = \frac{1}{\frac{1}{OIP_{3,N}} + \frac{1}{G_N \cdot OIP_{3,N-1}} + \ldots + \frac{1}{G_N \cdot G_{N-1} \cdot \ldots \cdot G_2 \cdot OIP_{3,1}}}$$

  where G_n is the gain of the nth element of the cascade and $OIP_{3,N}$ is the OIP_3 of the n^{th} element.
- The `analyze` method uses the cascaded S-parameters to calculate the group delay values at the frequencies specified in the `analyze` input argument `freq`, as described in the `analyze` reference page.

Version History

Introduced before R2006a

References

[1] Ludwig, R. and P. Bretchko, *RF Circuit Design: Theory and Applications*, Prentice Hall, 2000.

See Also

`rfckt.amplifier` | `rfckt.coaxial` | `rfckt.cpw` | `rfckt.datafile` | `rfckt.delay` | `rfckt.hybrid` | `rfckt.hybridg` | `rfckt.mixer` | `rfckt.microstrip` | `rfckt.passive` | `rfckt.parallel` | `rfckt.parallelplate` | `rfckt.rlcgline` | `rfckt.series` | `rfckt.seriesrlc` | `rfckt.shuntrlc` | `rfckt.twowire` | `rfckt.txline`

Topics

“Bandpass Filter Response Using RFCKT Objects”
“MOS Interconnect and Crosstalk Using RFCKT Objects”

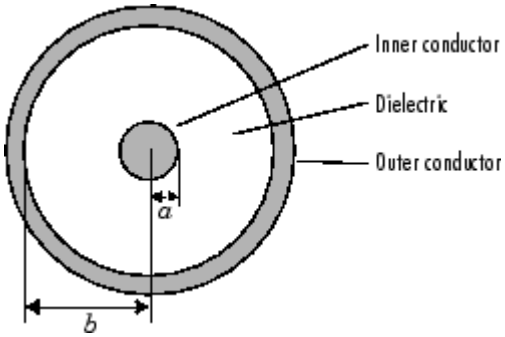
rfckt.coaxial

Coaxial transmission line

Description

Use the `coaxial` class to represent coaxial transmission lines that are characterized by line dimensions, stub type, and termination.

A coaxial transmission line is shown in cross-section in the following figure. Its physical characteristics include the radius of the inner conductor of the coaxial transmission line *a*, and the radius of the outer conductor *b*.



Creation

Syntax

```
h = rfckt.coaxial
h = rfckt.coaxial(Name,Value)
```

Description

`h = rfckt.coaxial` returns a coaxial transmission line object whose properties are set to their default values.

`h = rfckt.coaxial(Name,Value)` sets properties using one or more name-value pairs. For example, `rfckt.coaxial('OuterRadius',0.0043)` creates a coaxial transmission line object with outer radius of 0.0043 meters. You can specify multiple name-value pairs. Enclose each property name in quotes. Properties not specified retain their default values.

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as an `rfdata.data` object. This is a read-only property. For more information refer, “Algorithms” on page 1-9.

Data Types: `function_handle`

`EpsilonR` — Relative permittivity of dielectric
scalar

Relative permittivity of dielectric, specified as a scalar. The relative permittivity is the ratio of permittivity of the dielectric,ε, to the permittivity in free space, ε_0. The default value is`2.3`.

Data Types: `double`

`InnerRadius` — Inner conductor radius
scalar

Inner conductor radius, specified as a scalar in meters. The default value is `7.25e-4`.

Data Types: `double`

`LineLength` — Physical length of transmission line
scalar

Physical length of transmission line, specified as a scalar in meters. The default value is `0.01`.

Data Types: `double`

`LossTangent` — Tangent of loss angle of dielectric
scalar

Tangent of loss angle of dielectric, specified as a scalar. The default value is `0`.

Data Types: `double`

`MUR` — Relative permeability of dielectric
scalar

Relative permeability of dielectric, specified as a scalar. The ratio of permeability of dielectric, μ, to the permeability in free space, μ_0. The default value is `1`.

Data Types: `double`

`Name` — Object name
`'Coaxial Transmission Line'` (default) | `1-by-N` character array

This property is read-only.

Object name, specified as a `1-by-N` character array.

Data Types: `char`

`nport` — Number of ports
positive integer

This property is read-only.

Number of ports, specified as a positive integer.

Data Types: `double`

`OuterRadius` — Outer conductor radius
scalar in meters

Outer conductor radius, specified as a scalar in meters. The default value is `0.0026`.

Data Types: `double`

`SigmaCond` — Conductor conductivity
scalar

Conductor conductivity, specified as a scalar in Siemens per meter (S/m). The default value is `Inf`.

Data Types: `double`

`StubMode` — Type of stub
`'NotaStub'` | `'Series'` | `'Shunt'`

Type of stub, specified as one of the following values: `'NotaStub'`, `'Series'`, `'Shunt'`.

Data Types: `double`

`Termination` — Stub transmission line termination
`'NotApplicable'` (default) | `'Open'` | `'Short'`

Stub transmission line termination, specified as one of the following values: `'NotApplicable'`, `'Open'`, `'Short'`.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions

Examples

Create Coaxial Transmission Line

Create a coaxial transmission line with 0.0045 meters outer radius using rfckt.coaxial.

```
tx1=rfckt.coaxial('OuterRadius',0.0045)
```

```
tx1 =
   rfckt.coaxial with properties:

       OuterRadius: 0.0045
       InnerRadius: 7.2500e-04
               MuR: 1
          EpsilonR: 2.3000
       LossTangent: 0
         SigmaCond: Inf
        LineLength: 0.0100
          StubMode: 'NotAStub'
       Termination: 'NotApplicable'
              nPort: 2
    AnalyzedResult: []
              Name: 'Coaxial Transmission Line'
```

Algorithms

The `analyze` method treats the transmission line as a 2-port linear network. It computes the `AnalyzedResult` property of a stub or as a stubless line using the data stored in the `rfckt.coaxial` object properties as follows:

- If you model the transmission line as a stubless line, the `analyze` method first calculates the ABCD-parameters at each frequency contained in the modeling frequencies vector. It then uses the `abcd2s` function to convert the ABCD-parameters to S-parameters.

  The `analyze` method calculates the ABCD-parameters using the physical length of the transmission line, d, and the complex propagation constant, k, using the following equations:

  $$A = \frac{e^{kd} + e^{-kd}}{2}$$

  $$B = \frac{Z_0 * \left(e^{kd} - e^{-kd}\right)}{2}$$

  $$C = \frac{e^{kd} - e^{-kd}}{2 * Z_0}$$

  $$D = \frac{e^{kd} + e^{-kd}}{2}$$

  Z_0 and k are vectors whose elements correspond to the elements of f, the vector of frequencies specified in the `analyze` input argument `freq`. Both can be expressed in terms of the resistance (R), inductance (L), conductance (G), and capacitance (C) per unit length (meters) as follows:

  $$Z_0 = \sqrt{\frac{R + j2\pi fL}{G + j2\pi fC}}$$

  $$k = k_r + jk_i = \sqrt{(R + j2\pi fL)(G + j2\pi FC)}$$

  where

$$R = \frac{1}{2\pi\sigma_{cond}\delta_{cond}}\left(\frac{1}{a}+\frac{1}{b}\right)$$

$$L = \frac{\mu}{2\pi}\ln\left(\frac{b}{a}\right)$$

$$G = \frac{2\pi\omega\varepsilon''}{\ln\left(\frac{b}{a}\right)}$$

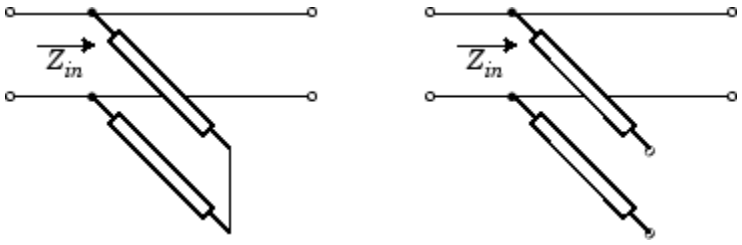
$$C = \frac{2\pi\varepsilon}{\ln\left(\frac{b}{a}\right)}$$

In these equations:

- a is the radius of the inner conductor.
- b is the radius of the outer conductor.
- σ_{cond} is the conductivity in the conductor.
- μ is the permeability of the dielectric.
- ε is the permittivity of the dielectric.
- ε'' is the imaginary part of ε, $\varepsilon'' = \varepsilon_0\varepsilon_r\tan\delta$, where:
  - ε_0 is the permittivity of free space.
  - ε_r is the `EpsilonR` property value.
  - $\tan\delta$ is the `LossTangent` property value.
- δ_{cond} is the skin depth of the conductor, which the method calculates as $1/\sqrt{\pi f\mu\sigma_{cond}}$.
- f is a vector of modeling frequencies determined by the Outport block.

- If you model the transmission line as a shunt or series stub, the `analyze` method first calculates the ABCD-parameters at the specified frequencies. It then uses the `abcd2s` function to convert the ABCD-parameters to S-parameters.

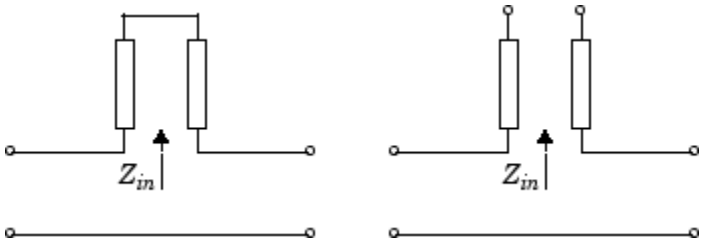
  When you set the `StubMode` property to `'Shunt'`, the 2-port network consists of a stub transmission line that you can terminate with either a short circuit or an open circuit as shown in the following figure.



  Z_{in} is the input impedance of the shunt circuit. The ABCD-parameters for the shunt stub are calculated as:

$$A = 1$$
$$B = 0$$
$$C = 1/Z_{in}$$
$$D = 1$$

When you set the `StubMode` property to `'Series'`, the 2-port network consists of a series transmission line that you can terminate with either a short circuit or an open circuit as shown in the following figure.



Z_{in} is the input impedance of the series circuit. The ABCD-parameters for the series stub are calculated as

$$A = 1$$
$$B = Z_{in}$$
$$C = 0$$
$$D = 1$$

Version History

Introduced before R2006a

References

[1] Pozar, David M. *Microwave Engineering*, John Wiley & Sons, Inc., 2005.

See Also

`rfckt.amplifier` | `rfckt.cascade` | `rfckt.cpw` | `rfckt.datafile` | `rfckt.delay` | `rfckt.hybrid` | `rfckt.hybridg` | `rfckt.mixer` | `rfckt.microstrip` | `rfckt.passive` | `rfckt.parallel` | `rfckt.parallelplate` | `rfckt.rlcgline` | `rfckt.series` | `rfckt.seriesrlc` | `rfckt.shuntrlc` | `rfckt.twowire` | `rfckt.txline`

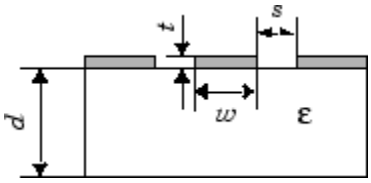
rfckt.cpw

Coplanar waveguide transmission line

Description

Use the `cpw` object to represent coplanar waveguide transmission lines that are characterized by line dimensions, stub type, and termination.

A coplanar waveguide transmission line is shown in cross-section in the following figure. Its physical characteristics include the conductor width (*w*), the conductor thickness (*t*), the slot width (*s*), the substrate height (*d*), and the permittivity constant (*ε*).



Creation

Syntax

```
h = rfckt.cpw
h = rfckt.cpw(Name,Value)
```

Description

`h = rfckt.cpw` returns a coplanar waveguide transmission line object whose properties are set to their default values.

`h = rfckt.cpw(Name,Value)` sets properties using one or more name-value pairs. For example, `rfckt.cpw('ConductorWidth',0.3)` creates an RF coplanar waveguide transmission line with a width of 0.3 meters. You can specify multiple name-value pairs. Enclose each property name in a quote. Properties not specified retain their default values.

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

This property is read-only.

Computed S-parameters, noise figure, OIP3, and group delay values, specified as an `rfdata.data` object. For more information refer, “Algorithms” on page 1-15.

Data Types: `function_handle`

`ConductorWidth` — Physical width of conductor
scalar in meters

Physical width of conductor, specified as a scalar in meters. By default, the value is `0.6e-4`.

Data Types: `double`

`EpsilonR` — Relative permittivity of dielectric
scalar

Relative permittivity of dielectric, specified as a scalar. The relative permittivity is the ratio of permittivity of the dielectric,ε, to the permittivity in free space, ε_0. By default, the value is `9.8`.

Data Types: `double`

`Height` — Dielectric thickness or physical height of conductor
scalar in meters

Dielectric thickness or physical height of the conductor, specified as a scalar in meters. The default value is `0.635e-4`.

Data Types: `double`

`LineLength` — Physical length of transmission
scalar in meters

Physical length of transmission, specified as a scalar in meters. The default value is `0.01`.

Data Types: `double`

`LossTangent` — Loss angle tangent of dielectric
scalar

Loss angle tangent of dielectric, specified as a scalar. The default value is `0`.

Data Types: `double`

`Name` — Name of coplanar waveguide transmission line object
`1-by-N` character array

This property is read-only.

Name of coplanar waveguide transmission line object, specified as a `1-by-N` character array.

Data Types: `char`

`nport` — Number of ports
positive integer

This property is read-only.

Number of ports, specified as a positive integer.

Data Types: `double`

`SigmaCond` — Conductor conductivity
scalar in Siemens per meter

Conductor conductivity, specified as a scalar in Siemens per meter (S/m). The default value is `Inf`.

Data Types: `double`

`StubMode` — Type of stub
`'NotaStub'` | `'Series'` | `'Shunt'`

Type of stub, specified as one of the following values: `'NotaStub'`, `'Series'`, `'Shunt'`.

Data Types: `double`

`SlotWidth` — Physical width of slot
scalar in meters

Physical width of slot, specified as a scalar in meters. The default value is `0.2e-4`.

Data Types: `double`

`Termination` — Stub transmission line termination
`'NotApplicable'` (default) | `'Open'` | `'Short'`

Stub transmission line termination, specified as one of the following values: `'NotApplicable'`, `'Open'`, `'Short'`.

Data Types: `double`

`Thickness` — Physical thickness of conductor
scalar in meters

Physical thickness of conductor, specified as a scalar in meters. The default value is `0.005e-6`.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions

Examples

Create Coplanar Waveguide Transmission Line

Create a coplanar waveguide transmission line using rfckt.cpw.

```
tx=rfckt.cpw('Thickness',0.0075e-6)
```

```
tx =
   rfckt.cpw with properties:

    ConductorWidth: 6.0000e-04
         SlotWidth: 2.0000e-04
            Height: 6.3500e-04
         Thickness: 7.5000e-09
          EpsilonR: 9.8000
       LossTangent: 0
         SigmaCond: Inf
        LineLength: 0.0100
           StubMode: 'NotAStub'
       Termination: 'NotApplicable'
              nPort: 2
    AnalyzedResult: []
              Name: 'Coplanar Waveguide Transmission Line'
```

Algorithms

The `analyze` method treats the transmission line as a 2-port linear network. It computes the `AnalyzedResult` property of a stub or as a stub less line using the data stored in the `rfckt.cpw` object properties as follows:

- If you model the transmission line as a stub less line, the `analyze` method first calculates the ABCD-parameters at each frequency contained in the modeling frequencies vector. It then uses the `abcd2s` function to convert the ABCD-parameters to S-parameters.

  The `analyze` method calculates the ABCD-parameters using the physical length of the transmission line, d, and the complex propagation constant, k, using the following equations:

  $$A = \frac{e^{kd} + e^{-kd}}{2}$$

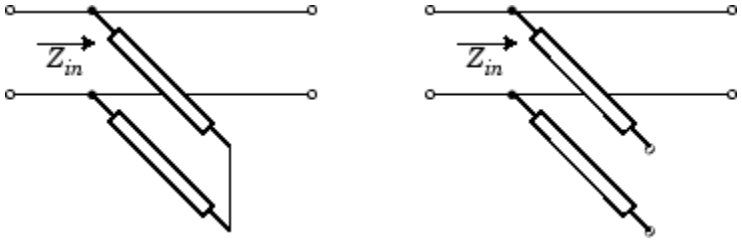
  $$B = \frac{Z_0 * \left(e^{kd} - e^{-kd}\right)}{2}$$

  $$C = \frac{e^{kd} - e^{-kd}}{2 * Z_0}$$

  $$D = \frac{e^{kd} + e^{-kd}}{2}$$

  Z_0 and k are vectors whose elements correspond to the elements of f, the vector of frequencies specified in the `analyze` input argument `freq`. Both can be expressed in terms of the specified conductor strip width, slot width, substrate height, conductor strip thickness, relative permittivity constant, conductivity and dielectric loss tangent of the transmission line, as described in [1].
- If you model the transmission line as a shunt or series stub, the `analyze` method first calculates the ABCD-parameters at the specified frequencies. It then uses the `abcd2s` function to convert the ABCD-parameters to S-parameters.

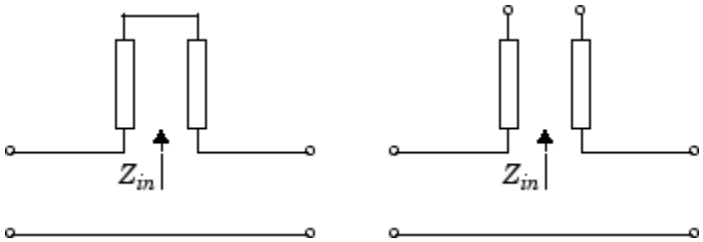
When you set the `StubMode` property to `'Shunt'`, the 2-port network consists of a stub transmission line that you can terminate with either a short circuit or an open circuit as shown in the following figure.



Z_{in} is the input impedance of the shunt circuit. The ABCD-parameters for the shunt stub are calculated as:

$$A = 1$$
$$B = 0$$
$$C = 1/Z_{in}$$
$$D = 1$$

When you set the `StubMode` property to `'Series'`, the 2-port network consists of a series transmission line that you can terminate with either a short circuit or an open circuit as shown in the following figure.



Z_{in} is the input impedance of the series circuit. The ABCD-parameters for the series stub are calculated as:

$$A = 1$$
$$B = Z_{in}$$
$$C = 0$$
$$D = 1$$

The `analyze` method uses the S-parameters to calculate the group delay values at the frequencies specified in the `analyze` input argument `freq`, as described in the `analyze` reference page.

Version History

Introduced before R2006a

References

[1] Gupta, K. C., R. Garg, I. Bahl, and P. Bhartia, *Microstrip Lines and Slotlines*, 2nd Edition, Artech House, Inc., Norwood, MA, 1996.

See Also

rfckt.amplifier | rfckt.cascade | rfckt.coaxial | rfckt.datafile | rfckt.delay | rfckt.hybrid | rfckt.hybridg | rfckt.mixer | rfckt.microstrip | rfckt.passive | rfckt.parallel | rfckt.parallelplate | rfckt.rlcgline | rfckt.series | rfckt.seriesrlc | rfckt.shuntrlc | rfckt.twowire | rfckt.txline

rfckt.datafile

Component or network from file data

Description

Use the `datafile` object to represent RF components and networks that are characterized by measured or simulated data in a file.

Use the `read` method to read the data from a file in one of the following formats:

- Touchstone
- Agilent® P2D
- Agilent S2D
- AMP

Creation

Syntax

```
h = rfckt.datafile
h = rfckt.datafile(Name,Value)
```

Description

`h = rfckt.datafile` returns a circuit object whose properties all have their default values.

`h = rfckt.datafile(Name,Value)` sets properties using one or more name-value pairs. For example, `rfckt.datafile('IntType','Cubic')` creates an RF component or network that uses cubic interpolation. You can specify multiple name-value pairs. Enclose each property name in a quote. Properties not specified retain their default values.

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

This property is read-only.

Computed S-parameters, noise figure, OIP3, and group delay values, specified as a `rfdata.data` object. For more information refer, “Algorithms” on page 1-20.

Data Types: `function_handle`

`File` — File name containing circuit data
`1-by-1` character array

File name containing circuit data, specified as a `1-by-1` character array.

Data Types: `char`

`IntpType` — Interpolation method
`1-by-N` character array

Interpolation method, specified as a `1-by-N` character array of the following values:

Method	Description
`Linear` (default)	Linear interpolation
`Spline`	Cubic spline interpolation
`Cubic`	Piecewise cubic Hermite interpolation

Data Types: `char`

`Name` — Object name
`1-by-N` character array

This property is read-only.

Object name, specified as a `1-by-N` character array.

Data Types: `char`

`nport` — Number of ports
positive integer

This property is read-only.

Number of ports, specified as a positive integer. The default value is 2.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions

groupdelay Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

Represent RF Components and Networks In Data File.

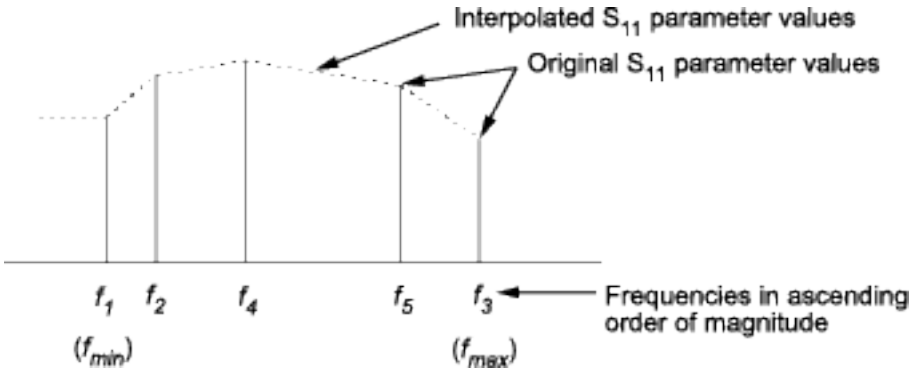
Represent RF components and networks that are characterized by measured or simulated data in a file using rfckt.datafile.

```
data=rfckt.datafile('File','default.s2p')
```

```
data =
   rfckt.datafile with properties:

          IntpType: 'Linear'
              File: 'default.s2p'
             nPort: 2
    AnalyzedResult: [1x1 rfdata.data]
              Name: 'Data File'
```

Algorithms

The `analyze` method computes the `AnalyzedResult` property using the data stored in the `File` object property. If the file you specify with this property contains network Y- or Z-parameters, `analyze` first converts these parameters, as they exist in the `rfckt.datafile` object, to S-parameters. Using the interpolation method you specify with the `'IntpType'` property, `analyze` interpolates the S-parameters to determine the S-parameters at the specified frequencies. Specifically, `analyze` orders the S-parameters according to the ascending order of their frequencies, f_n. It then interpolates the S-parameters, using the MATLAB® `interp1` function. For example, the curve in the following diagram illustrates the result of interpolating the S_{11} parameters at five different frequencies.



For more information, see “One-Dimensional Interpolation” and the `interp1` reference page in the MATLAB documentation.

Version History

Introduced before R2006a

References

[1] EIA/IBIS Open Forum, *Touchstone File Format Specification*, Rev. 1.1, 2002

See Also

`rfckt.amplifier` | `rfckt.cascade` | `rfckt.coaxial` | `rfckt.cpw` | `rfckt.delay` | `rfckt.hybrid` | `rfckt.hybridg` | `rfckt.mixer` | `rfckt.microstrip` | `rfckt.passive` | `rfckt.parallel` | `rfckt.parallelplate` | `rfckt.rlcgline` | `rfckt.series` | `rfckt.seriesrlc` | `rfckt.shuntrlc` | `rfckt.twowire` | `rfckt.txline`

rfckt.delay

Delay line

Description

Use the `delay` class to represent delay lines that are characterized by line loss and time delay.

Creation

Syntax

```
h = rfckt.delay
h = rfckt.delay(Name,Value)
```

Description

`h = rfckt.delay` returns a delay line object whose properties are set to their default values.

`h = rfckt.delay(Name,Value)` sets properties using one or more name-value pairs. For example, `rfckt.delay('Loss',2)`creates an RF delay line with a line loss of 2 dB. You can specify multiple name-value pairs. Enclose each property name in a quote. Properties not specified retain their default values.

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

This property is read-only.

Computed S-parameters, noise figure, OIP3, and group delay values, specified as a `rfdata.data` object. For more information refer, “Algorithms” on page 1-24.

Data Types: `function_handle`

`'Loss'` — Line loss value
`0` | positive scalar in dB

Line loss value, specified as a positive scalar in dB. Line loss is the reduction in strength of the signal as it travels over the delay line .

Data Types: `double`

`Name` — Object name
`1-by-N` character array

This property is read-only.

Object name, specified as a `1-by-N` character array.

Data Types: `char`

`nport` — Number of ports
`2` | positive integer

This property is read-only.

Number of ports, specified as a positive integer.

Data Types: `double`

`'TimeDelay'` — Amount of time delay
`1.0000e-012` | scalar in seconds

Amount of time delay introduced in the line, specified as a scalar in seconds.

Data Types: `double`

`'Z0'` — Characteristic impedance
`50` | scalar in ohms

Characteristic impedance of the delay line, specified as a scalar in ohms.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

Represent Delay Lines

Represent delay lines that are characterized by line loss and time delay using rfckt.delay.

```
del=rfckt.delay('TimeDelay',1e-11)
```

```
del =
   rfckt.delay with properties:

              Z0: 50.0000 + 0.0000i
            Loss: 0
       TimeDelay: 1.0000e-11
           nPort: 2
    AnalyzedResult: []
            Name: 'Delay Line'
```

Algorithms

The `analyze` method treats the delay line, which can be lossy or lossless, as a 2-port linear network. It computes the `AnalyzedResult` property of the delay line using the data stored in the `rfckt.delay` object properties by calculating the S-parameters for the specified frequencies. This calculation is based on the values of the delay line's `loss`, α, and time delay, D.

$$\begin{cases} S_{11} = 0 \\ S_{12} = e^{-p} \\ S_{21} = e^{-p} \\ S_{22} = 0 \end{cases}$$

Above, $p = \alpha_a + i\beta$, where α_a is the attenuation coefficient and β is the wave number. The attenuation coefficient α_a is related to the loss, α, by

$$\alpha_a = -\ln\left(10^{\alpha/20}\right)$$

and the wave number β is related to the time delay, D, by

$$\beta = 2\pi f D$$

where f is the frequency range specified in the `analyze` input argument `freq`.

Version History

Introduced before R2006a

References

[1] Ludwig, R. and P. Bretchko, *RF Circuit Design: Theory and Applications*, Prentice-Hall, 2000.

See Also

`rfckt.amplifier` | `rfckt.cascade` | `rfckt.coaxial` | `rfckt.cpw` | `rfckt.datafile` | `rfckt.hybrid` | `rfckt.hybridg` | `rfckt.mixer` | `rfckt.microstrip` | `rfckt.passive` | `rfckt.parallel` | `rfckt.parallelplate` | `rfckt.rlcgline` | `rfckt.series` | `rfckt.seriesrlc` | `rfckt.shuntrlc` | `rfckt.twowire` | `rfckt.txline`

rfckt.hybrid

Hybrid connected network

Description

Use the `hybrid` object to represent hybrid connected networks of linear RF objects characterized by the components that make up the network.

Creation

Syntax

```
h = rfckt.hybrid
h = rfckt.hybrid('Ckts',value)
```

Description

`h = rfckt.hybrid` returns a hybrid connected network object whose properties all have their default values.

`h = rfckt.hybrid('Ckts',value)` returns a cascaded network with elements specified in the name-value pair property `Ckts`.

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

This property is read-only.

Computed S-parameters, noise figure, OIP3, and group delay values, specified as an `rfdata.data` object. For more information, see “Algorithms” on page 1-27.

Data Types: `function_handle`

`Ckts` — Circuit objects in network
cell array of object handles

Circuit objects in network, specified as a cell array of object handles. All circuits must be 2-port. By default, this property is empty.

Data Types: `char`

`Name` — Object name
`1-by-N` character array

This property is read-only.

Object name, specified as a `1-by-N` character array.

Data Types: `char`

`nport` — Number of ports
positive integer

This property is read-only.

Number of ports, specified as a positive integer. The default value is 2.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

Create Hybrid Connected Networks

Create hybrid connected networks of linear RF objects with two transmission line objects using rfckt.hybrid.

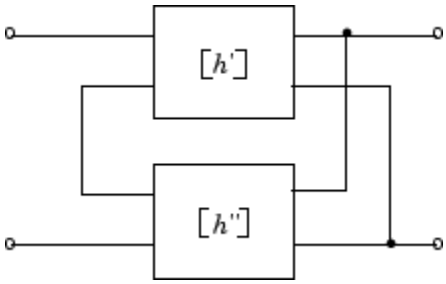
```
tx1 = rfckt.txline;
tx2 = rfckt.txline;
hyb = rfckt.hybrid('Ckts',{tx1,tx2})
```

```
hyb =
   rfckt.hybrid with properties:

            Ckts: {[1x1 rfckt.txline]  [1x1 rfckt.txline]}
           nPort: 2
    AnalyzedResult: []
            Name: 'Hybrid Connected Network'
```

Algorithms

The `analyze` method computes the S-parameters of the `AnalyzedResult` property using the data stored in the `Ckts` property as follows:

- The `analyze` method first calculates the *h* matrix of the hybrid network. It starts by converting each component network parameters to an *h* matrix. The following figure shows a hybrid connected network consisting of two 2-port networks, each represented by its *h* matrix,



  where

$$[h'] = \begin{bmatrix} h_{11}' & h_{12}' \\ h_{21}' & h_{22}' \end{bmatrix}$$

$$[h''] = \begin{bmatrix} h_{11}'' & h_{12}'' \\ h_{21}'' & h_{22}'' \end{bmatrix}$$

- The `analyze` method then calculates the *h* matrix for the hybrid network by calculating the sum of the *h* matrices of the individual networks. The following equation illustrates the calculations for two 2-port networks.

$$[h] = \begin{bmatrix} h_{11}' + h_{11}'' & h_{12}' + h_{12}'' \\ h_{21}' + h_{21}'' & h_{22}' + h_{22}'' \end{bmatrix}$$

- Finally, `analyze` converts the *h* matrix of the hybrid network to S-parameters at the frequencies specified in the `analyze` input argument `freq`.

Version History

Introduced before R2006a

References

[1] Ludwig, R. and P. Bretchko, *RF Circuit Design: Theory and Applications*, Prentice-Hall, 2000.

See Also

`rfckt.amplifier` | `rfckt.cascade` | `rfckt.coaxial` | `rfckt.cpw` | `rfckt.datafile` | `rfckt.delay` | `rfckt.hybridg` | `rfckt.mixer` | `rfckt.microstrip` | `rfckt.passive` | `rfckt.parallel` | `rfckt.parallelplate` | `rfckt.rlcgline` | `rfckt.series` | `rfckt.seriesrlc` | `rfckt.shuntrlc` | `rfckt.twowire` | `rfckt.txline`

rfckt.hybridg

Inverse hybrid connected network

Description

Use the `hybridg` object to represent hybrid connected networks of linear RF objects characterized by the components that make up the network.

Creation

Syntax

```
h = rfckt.hybridg
h = rfckt.hybridg('Ckts',value)
```

Description

`h = rfckt.hybridg` returns an inverse hybrid connected network object whose properties all have their default values.

`h = rfckt.hybridg('Ckts',value)` returns a cascaded network with elements specified in the name-value pair property `Ckts`.

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

This property is read-only.

Computed S-parameters, noise figure, OIP3, and group delay values, specified as an rfdata.data object. For more information, see “Algorithms” on page 1-30.

Data Types: `function_handle`

`Ckts` — Circuit objects in network
cell array of object handles

Circuit objects in network, specified as a cell array of object handles. All circuits must be 2-port. By default, this property is empty.

Data Types: `char`

`Name` — Object name
`1-by-N` character array

This property is read-only.

Object name, specified as a `1-by-N` character array.

Data Types: `char`

`nport` — Number of ports
positive integer

This property is read-only.

Number of ports, specified as a positive integer. The default value is 2.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

Create Inverse Hybrid Connected Networks

Create inverse hybrid connected networks of linear RF objects with two transmission line objects using rfckt.hybridg.

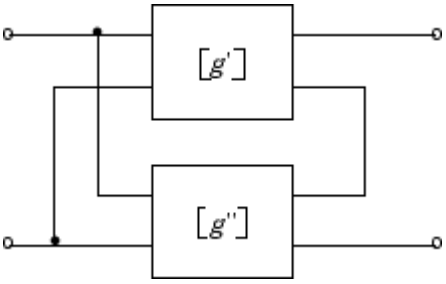
```
tx1 = rfckt.txline;
tx2 = rfckt.txline;
invhyb = rfckt.hybridg('Ckts',{tx1,tx2})
```

```
invhyb =
   rfckt.hybridg with properties:

              Ckts: {[1x1 rfckt.txline]  [1x1 rfckt.txline]}
             nPort: 2
    AnalyzedResult: []
              Name: 'Hybrid G Connected Network'
```

Algorithms

The `analyze` method computes the S-parameters of the `AnalyzedResult` property using the data stored in the `Ckts` property as follows:

1. The `analyze` method first calculates the *g* matrix of the inverse hybrid network. It starts by converting each component network's parameters to a *g* matrix. The following figure shows an inverse hybrid connected network consisting of two 2-port networks, each represented by its *g* matrix,



   where

$$[g'] = \begin{bmatrix} g_{11}' & g_{12}' \\ g_{21}' & g_{22}' \end{bmatrix}$$

$$[g''] = \begin{bmatrix} g_{11}'' & g_{12}'' \\ g_{21}'' & g_{22}'' \end{bmatrix}$$

2. The `analyze` method then calculates the *g* matrix for the inverse hybrid network by calculating the sum of the *g* matrices of the individual networks. The following equation illustrates the calculations for two 2-port networks.

$$[g] = \begin{bmatrix} g_{11}' + g_{11}'' & g_{12}' + g_{12}'' \\ g_{21}' + g_{21}'' & g_{22}' + g_{22}'' \end{bmatrix}$$

3. Finally, `analyze` converts the *g* matrix of the inverse hybrid network to S-parameters at the frequencies specified in the `analyze` input argument `freq`.

Version History

Introduced before R2006a

References

[1] Davis, A.M., *Linear Circuit Analysis*, PWS Publishing Company, 1998.

See Also

`rfckt.amplifier` | `rfckt.cascade` | `rfckt.coaxial` | `rfckt.cpw` | `rfckt.datafile` | `rfckt.delay` | `rfckt.hybrid` | `rfckt.mixer` | `rfckt.microstrip` | `rfckt.passive` | `rfckt.parallel` | `rfckt.parallelplate` | `rfckt.rlcgline` | `rfckt.series` | `rfckt.seriesrlc` | `rfckt.shuntrlc` | `rfckt.twowire` | `rfckt.txline`

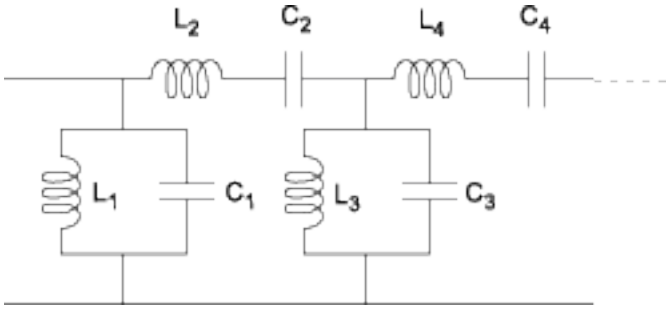
rfckt.lcbandpasspi

Bandpass pi filter

Description

Use the `lcbandpasspi` class to represent a bandpass pi filter as a network of inductors and capacitors.

The LC bandpass pi network object is a 2-port network as shown in the following circuit diagram.



In the diagram, [L_1, L_2, L_3, L_4, ...] is the value of the `'L'` object property, and [C_1, C_2, C_3, C_4, ...] is the value of the `'C'` object property.

Creation

Syntax

```
h = rfckt.lcbandpasspi
h = rfckt.lcbandpasspi('Property1',value1,'Property2',value2,...)
```

Description

`h = rfckt.lcbandpasspi`returns an LC bandpass pi network object whose properties all have their default values.

`h = rfckt.lcbandpasspi('Property1',value1,'Property2',value2,...)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote

Properties

`'AnalyzedResult'` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as a comma-separated pair consisting of `'AnalyzedResult'` and rfdata.data object. The values are computed over the specified frequency range. By default, this property is empty.

Data Types: `function_handle`

`'C'` — Capacitance value
positive vector in farads

Capacitance value from source to load of all capacitors in the network, specified as a comma separated pair consisting of `'C'` and a positive vector in farads. The length of the capacitance vector must be equal to the length of the vector you provide for `'L'`. The default value is `[0.3579e-10, 0.0118e-10, 0.3579e-10]`.

Data Types: `double`

`'L'` — Inductance value
positive vector

Inductance value from source to load of all inductors in the network, specified as a comma separated pair consisting of `'L'` and a positive vector in henries. The length of the inductance vector must be equal to the length of the vector you provide for `'C'`. The default value is `[0.0144e-7, 0.4395e-7, 0.0144e-7]`.

Data Types: `double`

`'Name'` — Object name
`'LC Bandpass Pi'` (default) | `1-by-N` character array

Object name, specified as a comma-separated pair consisting of `'Name'` and `1-by-N` character array. This is a read-only property.

Data Types: `char`

`'nport'` — Number of ports
positive integer

Number of ports, specified as a comma-separated pair consisting of `'nport'` and a positive integer. This is a read-only property. The default value is 2.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file

getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

Create LC BandPass Pi Filter

Create an LC bandpass filter of capacitor values 1e-12 and 4e12 farads, inductor values 2e-9 and 2.5e-9 henries.

```
filter = rfckt.lcbandpasspi('C',[1e-12 4e-12],'L',[2e-9 2.5e-9])
```

```
filter =
   rfckt.lcbandpasspi with properties:

                L: [2x1 double]
                C: [2x1 double]
            nPort: 2
     AnalyzedResult: []
             Name: 'LC Bandpass Pi'
```

Version History

Introduced before R2006a

References

[1] Ludwig, R. and P. Bretchko, *RF Circuit Design: Theory and Applications*, Prentice-Hall, 2000.

[2] Zverev, A.I., *Handbook of Filter Synthesis*, John Wiley & Sons, 1967.

See Also

`rfckt.lcbandpasstee` | `rfckt.lcbandstoppi` | `rfckt.lcbandstoptee` | `rfckt.lchighpasspi` | `rfckt.lchighpasstee` | `rfckt.lclowpasspi` | `rfckt.lclowpasstee`

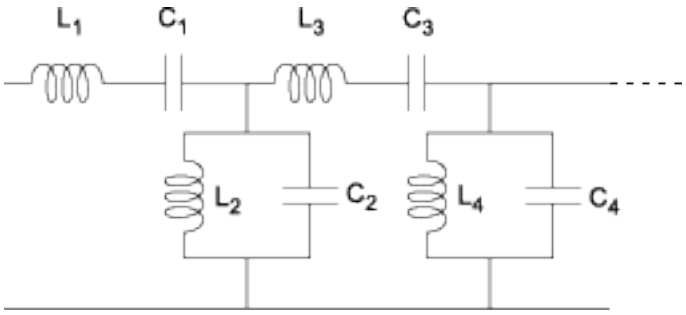
rfckt.lcbandpasstee

Bandpass tee filter

Description

Use the `lcbandpasstee` class to represent a bandpass tee filter as a network of inductors and capacitors.

The LC bandpass tee network object is a 2-port network as shown in the following circuit diagram.



In the diagram, [L_1, L_2, L_3, L_4, ...] is the value of the `'L'` object property, and [C_1, C_2, C_3, C_4, ...] is the value of the `'C'` object property.

Creation

Syntax

```
h = rfckt.lcbandpasstee
h = rfckt.lcbandpasstee('Property1',value1,'Property2',value2,...)
```

Description

`h = rfckt.lcbandpasstee`returns an LC bandpass tee network object whose properties all have their default values.

`h = rfckt.lcbandpasstee('Property1',value1,'Property2',value2,...)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote

Properties

`'AnalyzedResult'` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as rfdata.data object. The values are computed over the specified frequency range. By default, this property is empty.

Data Types: `function_handle`

`'C'` — Capacitance value
positive vector in farads

Capacitance value from source to load of all capacitors in the network, specified as a positive vector in farads. The length of the capacitance vector must be equal to the length of the vector you provide for `'L'`. The default value is `[0.0186e-10, 0.1716e-10, 0.0186e-10]`.

Data Types: `double`

`'L'` — Inductance value
positive vector in henries

Inductance value from source to load of all inductors in the network, specified as a positive vector in henries. The length of the inductance vector must be equal to the length of the vector you provide for `'C'`. The default value is `[0.2781e-7, 0.0301e-7, 0.2781e-7]`.

Data Types: `double`

`'Name'` — Object name
`'LC Bandpass Tee'` (default) | `1-by-N` character array

Object name, specified as a `1-by-N` character array. This is a read-only property.

Data Types: `char`

`'nport'` — Number of ports
positive integer

Number of ports, specified as a positive integer. This is a read-only property. The default value is 2.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions

groupdelay Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

LC Bandpass Tee Filter

Create a LC Bandpass Tee Filter using `rfckt.lcbandpasstee`.

```
filter = rfckt.lcbandpasstee('C',[1e-12 4e-12],'L',[2e-9 2.5e-9])
```

```
filter =
   rfckt.lcbandpasstee with properties:

                L: [2x1 double]
                C: [2x1 double]
            nPort: 2
     AnalyzedResult: []
             Name: 'LC Bandpass Tee'
```

Version History

Introduced before R2006a

References

[1] Ludwig, R. and P. Bretchko, *RF Circuit Design: Theory and Applications*, Prentice-Hall, 2000.

[2] Zverev, A.I., *Handbook of Filter Synthesis*, John Wiley & Sons, 1967.

See Also

`rfckt.lcbandpasspi` | `rfckt.lcbandstoppi` | `rfckt.lcbandstoptee` | `rfckt.lchighpasspi` | `rfckt.lchighpasstee` | `rfckt.lclowpasspi` | `rfckt.lclowpasstee`

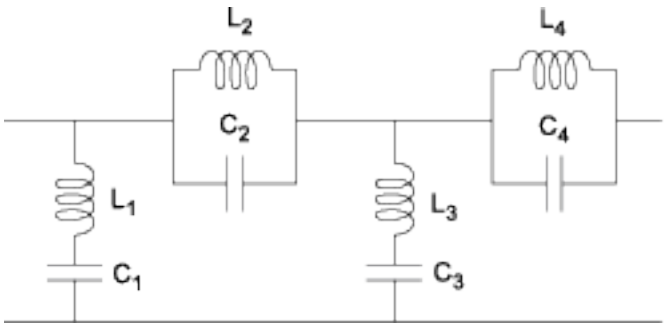
rfckt.lcbandstoppi

Bandstop pi filter

Description

Use the `lcbandstoppi` class to represent a bandstop pi filter as a network of inductors and capacitors.

The LC bandstop pi network object is a 2-port network as shown in the following circuit diagram.



In the diagram, [L_1, L_2, L_3, L_4, ...] is the value of the `'L'` object property, and [C_1, C_2, C_3, C_4, ...] is the value of the `'C'` object property.

Creation

Syntax

```
h = rfckt.lcbandstoppi
h = rfckt.lcbandstoppi('Property1',value1,'Property2',value2,...)
```

Description

`h = rfckt.lcbandstoppi` returns an LC bandstop pi network object whose properties all have their default values.

`h = rfckt.lcbandstoppi('Property1',value1,'Property2',value2,...)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote

Properties

`'AnalyzedResult'` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as rfdata.data object. The values are computed over the specified frequency range. By default, this property is empty.

Data Types: `function_handle`

`'C'` — Capacitance value
positive vector in farads

Capacitance value from source to load of all capacitors in the network, specified as a positive vector in farads. The length of the capacitance vector must be equal to the length of the vector you provide for `'L'`. The default value is `[0.0184e-10, 0.2287e-10, 0.0184e-10]`.

Data Types: `double`

`'L'` — Inductance value
positive vector in henries

Inductance value from source to load of all inductors in the network, specified as a positive vector in henries. The length of the inductance vector must be equal to the length of the vector you provide for `'C'`. The default value is `[0.2809e-7, 0.0226e-7, 0.2809e-7]`.

Data Types: `double`

`'Name'` — Object name
`'LC Bandstop Pi'` (default) | `1-by-N` character array

Object name, specified as a `1-by-N` character array. This is a read-only property.

Data Types: `char`

`'nport'` — Number of ports
positive integer

Number of ports, specified as a positive integer. This is a read-only property. The default value is `2`.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions

groupdelay Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

LC Bandstop Pi Filter

Create a LC Bandstop Pi Filter using `rfckt.lcbandstoppi`.

```
filter = rfckt.lcbandstoppi('C',[1e-12 4e-12],'L',[2e-9 2.5e-9])
```

```
filter =
   rfckt.lcbandstoppi with properties:

                L: [2x1 double]
                C: [2x1 double]
            nPort: 2
     AnalyzedResult: []
             Name: 'LC Bandstop Pi'
```

Version History

Introduced before R2006a

References

[1] Ludwig, R. and P. Bretchko, *RF Circuit Design: Theory and Applications*, Prentice-Hall, 2000.

[2] Zverev, A.I., *Handbook of Filter Synthesis*, John Wiley & Sons, 1967.

See Also

`rfckt.lcbandpasspi` | `rfckt.lcbandpasstee` | `rfckt.lcbandstoptee` | `rfckt.lchighpasspi` | `rfckt.lchighpasstee` | `rfckt.lclowpasspi` | `rfckt.lclowpasstee`

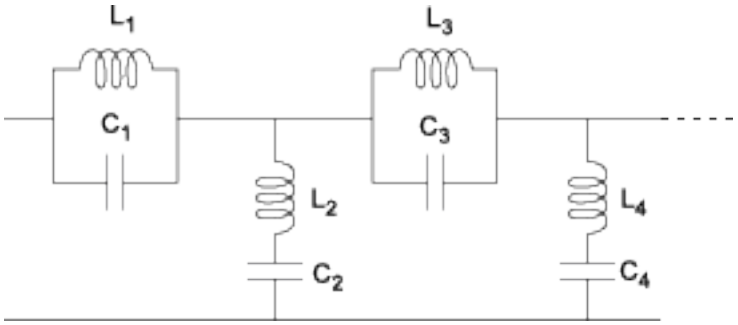
rfckt.lcbandstoptee

Bandstop tee filter

Description

Use the `lcbandstoptee` class to represent a bandstop tee filter as a network of inductors and capacitor.

The LC bandstop tee network object is a 2-port network as shown in the following circuit diagram.



In the diagram, [L_1, L_2, L_3, L_4, ...] is the value of the `'L'` object property, and [C_1, C_2, C_3, C_4, ...] is the value of the `'C'` object property.

Creation

Syntax

```
h = rfckt.lcbandstoptee
h = rfckt.lcbandstoptee('Property1',value1,'Property2',value2,...)
```

Description

`h = rfckt.lcbandstoptee` returns an LC bandstop tee network object whose properties all have their default values.

`h = rfckt.lcbandstoptee('Property1',value1,'Property2',value2,...)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote

Properties

`'AnalyzedResult'` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as rfdata.data object. The values are computed over the specified frequency range. By default, this property is empty.

Data Types: `function_handle`

`'C'` — Capacitance value
positive vector in farads

Capacitance value from source to load of all capacitors in the network, specified as a positive vector in farads. The length of the capacitance vector must be equal to the length of the vector you provide for `'L'`. The default value is `[0.0186e-10, 0.1716e-10, 0.0186e-10]`.

Data Types: `double`

`'L'` — Inductance value
positive vector in henries

Inductance value from source to load of all inductors in the network, specified as a positive vector in henries. The length of the inductance vector must be equal to the length of the vector you provide for `'C'`. The default value is `[0.2781e-7, 0.0301e-7, 0.2781e-7]`.

Data Types: `double`

`'Name'` — Object name
`'LC Bandstop Tee'` (default) | `1-by-N` character array

Object name, specified as a `1-by-N` character array. This is a read-only property.

Data Types: `char`

`'nport'` — Number of ports
positive integer

Number of ports, specified as a positive integer. This is a read-only property. The default value is 2.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions

groupdelay Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

LC Bandstop Tee Filter

Create a LC Bandstop Tee Filter using `rfckt.lcbandstoptee`.

```
filter = rfckt.lcbandstoptee('C',[1e-12 4e-12],'L',[2e-9 2.5e-9])
```

```
filter =
   rfckt.lcbandstoptee with properties:

                L: [2x1 double]
                C: [2x1 double]
            nPort: 2
     AnalyzedResult: []
             Name: 'LC Bandstop Tee'
```

Version History

Introduced before R2006a

References

[1] Ludwig, R. and P. Bretchko, *RF Circuit Design: Theory and Applications*, Prentice-Hall, 2000.

[2] Zverev, A.I., *Handbook of Filter Synthesis*, John Wiley & Sons, 1967.

See Also

`rfckt.lcbandpasspi` | `rfckt.lcbandpasstee` | `rfckt.lcbandstoppi` | `rfckt.lchighpasspi` | `rfckt.lchighpasstee` | `rfckt.lclowpasspi` | `rfckt.lclowpasstee`

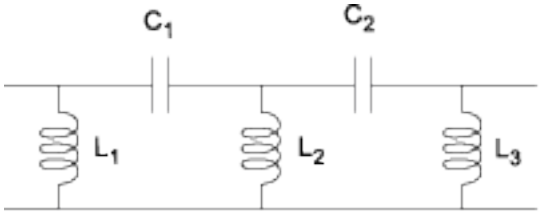
rfckt.lchighpasspi

Highpass pi filter

Description

Use the `lchighpasspi` class to represent a highpass pi filter as a network of inductors and capacitors.

The LC highpass pi network object is a 2-port network as shown in the following circuit diagram.



In the diagram, [L_1, L_2, L_3, ...] is the value of the `'L'` object property, and [C_1, C_2, C_3, ...] is the value of the `'C'` object property.

Creation

Syntax

```
h = rfckt.lchighpasspi
h = rfckt.lchighpasspi('Property1',value1,'Property2',value2,...)
```

Description

`h = rfckt.lchighpasspi` returns an LC highpass pi network object whose properties all have their default values.

`h = rfckt.lchighpasspi('Property1',value1,'Property2',value2,...)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote

Properties

`'AnalyzedResult'` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as rfdata.data object. The values are computed over the specified frequency range. By default, this property is empty.

Data Types: `function_handle`

`'C'` — Capacitance value
positive vector in farads

Capacitance value from source to load of all capacitors in the network, specified as a positive vector in farads. The default value is `[0.1188e-5, 0.1188e-5]`.

Data Types: `double`

`'L'` — Inductance value
positive vector in henries

Inductance value from source to load of all inductors in the network, specified as a positive vector in henries. The default value is `[2.2363e-9]`.

Data Types: `double`

`'Name'` — Object name
`'LC Highpass Pi'` (default) | `1-by-N` character array

Object name, specified as a `1-by-N` character array. This is a read-only property.

Data Types: `char`

`'nport'` — Number of ports
positive integer

Number of ports, specified as a positive integer. This is a read-only property. The default value is 2.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

LC Highpass Pi Filter

Create a LC Highpass Pi Filter using `rfckt.lchighpasspi`.

```
filter = rfckt.lchighpasspi('C',[1e-12 4e-12],'L',[2e-9 2.5e-9])

filter =
   rfckt.lchighpasspi with properties:

                L: [2x1 double]
                C: [2x1 double]
            nPort: 2
     AnalyzedResult: []
             Name: 'LC Highpass Pi'
```

Version History

Introduced before R2006a

References

[1] Ludwig, R. and P. Bretchko, *RF Circuit Design: Theory and Applications*, Prentice-Hall, 2000.

[2] Zverev, A.I., *Handbook of Filter Synthesis*, John Wiley & Sons, 1967.

See Also

`rfckt.lcbandpasspi` | `rfckt.lcbandpasstee` | `rfckt.lcbandstoppi` | `rfckt.lcbandstoptee` | `rfckt.lchighpasstee` | `rfckt.lclowpasspi` | `rfckt.lclowpasstee`

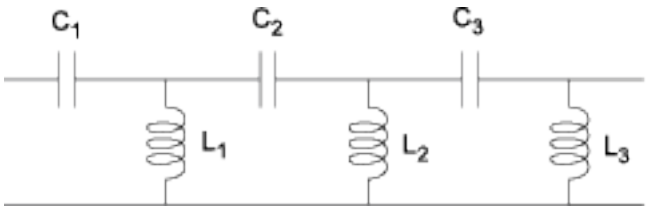
rfckt.lchighpasstee

Highpass tee filter

Description

Use the `lchighpasstee` class to represent a highpass tee filter as a network of inductors and capacitors.

The LC highpass tee network object is a 2-port network as shown in the following circuit diagram.



In the diagram, [L_1, L_2, L_3, ...] is the value of the `'L'` object property, and [C_1, C_2, C_3, ...] is the value of the `'C'` object property.

Creation

Syntax

```
h = rfckt.lchighpasstee
h = rfckt.lchighpasstee('Property1',value1,'Property2',value2,...)
```

Description

`h = rfckt.lchighpasstee` returns an LC highpass tee network object whose properties all have their default values.

`h = rfckt.lchighpasstee('Property1',value1,'Property2',value2,...)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote

Properties

`'AnalyzedResult'` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as rfdata.data object. The values are computed over the specified frequency range. By default, this property is empty.

Data Types: `function_handle`

`'C'` — Capacitance value
positive vector in farads

Capacitance value from source to load of all capacitors in the network, specified as a positive vector in farads. The default value is `[[0.4752e-9, 0.4752e-9]`.

Data Types: `double`

`'L'` — Inductance value
positive vector in henries

Inductance value from source to load of all inductors in the network, specified as a positive vector in henries. The default value is `[5.5907e-6]`.

Data Types: `double`

`'Name'` — Object name
`'LC Highpass Tee'` (default) | `1-by-N` character array

Object name, specified as a `1-by-N` character array. This is a read-only property.

Data Types: `char`

`'nport'` — Number of ports
positive integer

Number of ports, specified as a positive integer. This is a read-only property. The default value is 2.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

LC Highpass Tee Filter

Create a LC Highpass Tee Filter using `rfckt.lchighpasstee`.

```
filter = rfckt.lchighpasstee('C',[1e-12 4e-12],'L',[2e-9 2.5e-9])

filter =
   rfckt.lchighpasstee with properties:

                L: [2x1 double]
                C: [2x1 double]
            nPort: 2
     AnalyzedResult: []
             Name: 'LC Highpass Tee'
```

Version History

Introduced before R2006a

References

[1] Ludwig, R. and P. Bretchko, *RF Circuit Design: Theory and Applications*, Prentice-Hall, 2000.

[2] Zverev, A.I., *Handbook of Filter Synthesis*, John Wiley & Sons, 1967.

See Also

`rfckt.lcbandpasspi` | `rfckt.lcbandpasstee` | `rfckt.lcbandstoppi` | `rfckt.lcbandstoptee` | `rfckt.lchighpasspi` | `rfckt.lclowpasspi` | `rfckt.lclowpasstee`

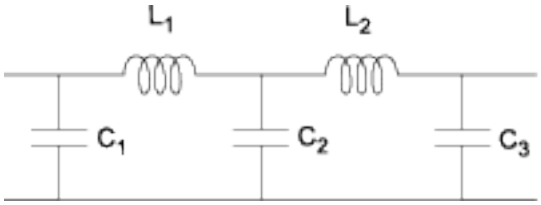
rfckt.lclowpasspi

Lowpass pi filter

Description

Use the `lclowpasspi` class to represent a lowpass pi filter as a network of inductors and capacitors.

The LC lowpass pi network object is a 2-port network as shown in the following circuit diagram.



In the diagram, [L_1, L_2, L_3, ...] is the value of the `'L'` object property, and [C_1, C_2, C_3, ...] is the value of the `'C'` object property.

Creation

Syntax

```
h = rfckt.lclowpasspi
h = rfckt.lclowpasspi('Property1',value1,'Property2',value2,...)
```

Description

`h = rfckt.lclowpasspi` returns an LC lowpass pi network object whose properties all have their default values.

`h = rfckt.lclowpasspi('Property1',value1,'Property2',value2,...)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote

Properties

`'AnalyzedResult'` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as rfdata.data object. The values are computed over the specified frequency range. By default, this property is empty.

Data Types: `function_handle`

`'C'` — Capacitance value
positive vector in farads

Capacitance value from source to load of all capacitors in the network, specified as a positive vector in farads. The default value is `[0.5330e-8, 0.5330e-8]`.

Data Types: `double`

`'L'` — Inductance value
positive vector in henries

Inductance value from source to load of all inductors in the network, specified as a positive vector in henries. The default value is `[2.8318e-6]`.

Data Types: `double`

`'Name'` — Object name
`'LC Lowpass Pi'` (default) | `1-by-N` character array

Object name, specified as a `1-by-N` character array. This is a read-only property.

Data Types: `char`

`'nport'` — Number of ports
positive integer

Number of ports, specified as a positive integer. This is a read-only property. By default, the value is `2`.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

LC Lowpass Pi Filter

Create a LC lowpass pi Filter using `rfckt.lclowpasspi`.

```
filter = rfckt.lclowpasspi('C',[1e-12 4e-12],'L',[2e-9 2.5e-9])
```

```
filter =
   rfckt.lclowpasspi with properties:

                L: [2x1 double]
                C: [2x1 double]
            nPort: 2
     AnalyzedResult: []
             Name: 'LC Lowpass Pi'
```

Version History

Introduced before R2006a

References

[1] Ludwig, R. and P. Bretchko, *RF Circuit Design: Theory and Applications*, Prentice-Hall, 2000.

[2] Zverev, A.I., *Handbook of Filter Synthesis*, John Wiley & Sons, 1967.

See Also

`rfckt.lcbandpasspi` | `rfckt.lcbandpasstee` | `rfckt.lcbandstoppi` | `rfckt.lcbandstoptee` | `rfckt.lchighpasspi` | `rfckt.lchighpasstee` | `rfckt.lclowpasstee`

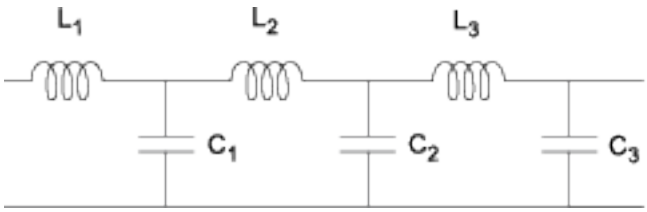
rfckt.lclowpasstee

Lowpass tee filter

Description

Use the `lclowpasstee` class to represent a lowpass tee filter as a network of inductors and capacitors

The LC lowpass tee network object is a 2-port network as shown in the following circuit diagram.



In the diagram, [L_1, L_2, L_3, ...] is the value of the `'L'` object property, and [C_1, C_2, C_3, ...] is the value of the `'C'` object property.

Creation

Syntax

```
h = rfckt.lclowpasstee
h = rfckt.lclowpasstee('Property1',value1,'Property2',value2,...)
```

Description

`h = rfckt.lclowpasstee` returns an LC lowpass tee network object whose properties all have their default values.

`h = rfckt.lclowpasstee('Property1',value1,'Property2',value2,...)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote

Properties

`'AnalyzedResult'` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as rfdata.data object. The values are computed over the specified frequency range. By default, this property is empty.

Data Types: `function_handle`

`'C'` — Capacitance value
positive vector in farads

Capacitance value from source to load of all capacitors in the network, specified as a positive vector in farads. The default value is `[1.1327e-9]`.

Data Types: `double`

`'L'` — Inductance value
positive vector in henries

Inductance value from source to load of all inductors in the network, specified as a positive vector in henries. The default value is `[0.1332e-4, 0.1332e-4]`.

Data Types: `double`

`'Name'` — Object name
`'LC Lowpass Tee'` (default) | `1-by-N` character array

Object name, specified as a `1-by-N` character array. This is a read-only property.

Data Types: `char`

`'nport'` — Number of ports
positive integer

Number of ports, specified as a positive integer. This is a read-only property. The default value is 2.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

LC Lowpass Tee Filter

Create a LC lowpass tee Filter using `rfckt.lclowpasstee`.

```
filter = rfckt.lclowpasstee('C',[1e-12 4e-12],'L',[2e-9 2.5e-9])

filter =
   rfckt.lclowpasstee with properties:

                L: [2x1 double]
                C: [2x1 double]
            nPort: 2
     AnalyzedResult: []
             Name: 'LC Lowpass Tee'
```

Version History

Introduced before R2006a

References

[1] Ludwig, R. and P. Bretchko, *RF Circuit Design: Theory and Applications*, Prentice-Hall, 2000.

[2] Zverev, A.I., *Handbook of Filter Synthesis*, John Wiley & Sons, 1967.

See Also

`rfckt.lcbandpasspi` | `rfckt.lcbandpasstee` | `rfckt.lcbandstoppi` | `rfckt.lcbandstoptee` | `rfckt.lchighpasspi` | `rfckt.lchighpasstee` | `rfckt.lclowpasspi`

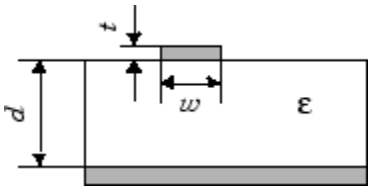
rfckt.microstrip

Microstrip transmission line

Description

Use the `microstrip` class to represent microstrip transmission lines characterized by line dimensions and optional stub properties.

A microstrip transmission line is shown in cross-section in the following figure. Its physical characteristics include the microstrip width (*w*), the microstrip thickness (*t*), the substrate height (*d*), and the relative permittivity constant (*ε*).



Creation

Syntax

```
h = rfckt.microstrip
h = rfckt.microstrip(Name,Value)
```

Description

`h = rfckt.microstrip` returns a microstrip transmission line object whose properties are set to their default values.

`h = rfckt.microstrip(Name,Value)` sets properties using one or more name-value pairs. For example, `rfckt.microstrip('Thickness',0.0075e-6)` creates a microstrip transmission line with thickness of `0.0075e-6` meters. You can specify multiple name-value pairs. Enclose each property name in a quote. Properties not specified retain their default values.

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as `rfdata.data` object. `AnalyzedResult` is a read-only property. For more information refer, “Algorithms” on page 1-58.

Data Types: `function_handle`

EpsilonR — Relative permittivity of dielectric
9.8 (default) | scalar

Relative permittivity of dielectric, specified as a scalar. The relative permittivity is the ratio of permittivity of the dielectric,ε, to the permittivity in free space, ε_0.

Data Types: double

Height — Dielectric thickness or physical height of conductor
6.35e-4 (default) | scalar

Dielectric thickness or physical height of the conductor, specified as a scalar in meters.

Data Types: double

LineLength — Physical length of transmission
0.01 (default) | scalar

Physical length of transmission, specified as a scalar in meters.

Data Types: double

LossTangent — Loss angle tangent of dielectric
0 (default) | scalar

Loss angle tangent of dielectric, specified as a scalar.

Data Types: double

Name — Object name
'Microstrip Transmission Line' (default) | 1-by-N character array | string scalar

This property is read-only.

Object name, specified as an 1-by-N character array or string scalar.

Data Types: char

nport — Number of ports
2 (default) | positive integer

Number of ports, specified as a positive integer. nportt is a read-only property.

Data Types: double

SigmaCond — Conductor conductivity
Inf (default) | scalar

Conductor conductivity, specified as a scalar in Siemens per meter (S/m).

Data Types: double

StubMode — Type of stub
'NotaStub' (default) | 'Series' | 'Shunt'

Type of stub, specified as one of the following values: 'NotaStub', 'Series', 'Shunt'.

Data Types: double

`Termination` — Stub transmission line termination
`'NotApplicable'` (default) | `'Open'` | `'Short'`

Stub transmission line termination, specified as one of the following values: `'NotApplicable'`, `'Open'`, `'Short'`.

Data Types: `double`

`Thickness` — Physical thickness of microstrip
`5.0e-6` (default) | scalar

Physical thickness of microstrip, specified as a scalar in meters.

Data Types: `double`

`Width` — Physical width of parallel-plate
`6.0e-4` (default) | scalar

Physical width of parallel-plate, specified as a scalar in meters.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

Microstrip Transmission Line

Create a microstrip transmission line using `rfckt.microstrip`.

```
tx1=rfckt.microstrip('Thickness',0.0075e-6)
```

```
tx1 =
   rfckt.microstrip with properties:
```

```
          Width: 6.0000e-04
         Height: 6.3500e-04
      Thickness: 7.5000e-09
        EpsilonR: 9.8000
    LossTangent: 0
      SigmaCond: Inf
     LineLength: 0.0100
        StubMode: 'NotAStub'
    Termination: 'NotApplicable'
          nPort: 2
 AnalyzedResult: []
           Name: 'Microstrip Transmission Line'
```

Algorithms

The `analyze` method treats the microstrip line as a 2-port linear network and models the line as a transmission line with optional stubs. The `analyze` method computes the `AnalyzedResult` property of the transmission line using the data stored in the `rfckt.microstrip` object properties as follows:

- If you model the transmission line as a stubless line, the `analyze` method first calculates the ABCD-parameters at each frequency contained in the modeling frequencies vector. It then uses the `abcd2s` function to convert the ABCD-parameters to S-parameters.

  The `analyze` method calculates the ABCD-parameters using the physical length of the transmission line, d, and the complex propagation constant, k, using the following equations:

  $$A = \frac{e^{kd} + e^{-kd}}{2}$$

  $$B = \frac{Z_0 * \left(e^{kd} - e^{-kd}\right)}{2}$$

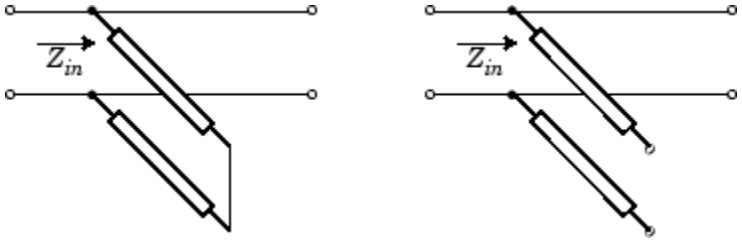
  $$C = \frac{e^{kd} - e^{-kd}}{2 * Z_0}$$

  $$D = \frac{e^{kd} + e^{-kd}}{2}$$

  Z_0 and k are vectors whose elements correspond to the elements of f, the vector of frequencies specified in the `analyze` input argument `freq`. Both can be expressed in terms of the specified conductor strip width, substrate height, conductor strip thickness, relative permittivity constant, conductivity, and dielectric loss tangent of the microstrip line, as described in [1].

- If you model the transmission line as a shunt or series stub, the `analyze` method first calculates the ABCD-parameters at the specified frequencies. It then uses the `abcd2s` function to convert the ABCD-parameters to S-parameters.

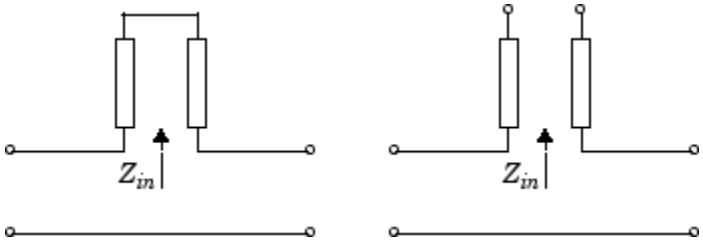
  When you set the `StubMode` property to `'Shunt'`, the 2-port network consists of a stub transmission line that you can terminate with either a short circuit or an open circuit.



Z_{in} is the input impedance of the shunt circuit. The ABCD-parameters for the shunt stub are calculated as:

$$A = 1$$
$$B = 0$$
$$C = 1/Z_{in}$$
$$D = 1$$

When you set the `StubMode` property to `'Series'`, the 2-port network consists of a series transmission line that you can terminate with either a short circuit or an open circuit.



Z_{in} is the input impedance of the series circuit. The ABCD-parameters for the series stub are calculated as:

$$A = 1$$
$$B = Z_{in}$$
$$C = 0$$
$$D = 1$$

Version History

Introduced before R2006a

References

[1] Gupta, K. C., R. Garg, I. Bahl, and P. Bhartia, *Microstrip Lines and Slotlines*, 2nd Edition, Artech House, Inc., Norwood, MA, 1996.

See Also

`rfckt.amplifier` | `rfckt.cascade` | `rfckt.coaxial` | `rfckt.cpw` | `rfckt.datafile` | `rfckt.delay` | `rfckt.hybrid` | `rfckt.hybridg` | `rfckt.mixer` | `rfckt.passive` |

rfckt.parallel | rfckt.parallelplate | rfckt.rlcgline | rfckt.series | rfckt.seriesrlc | rfckt.shuntrlc | rfckt.twowire | rfckt.txline

rfckt.mixer

2-port representation of RF mixer and its local oscillator

Description

Use the `mixer` class to represent RF mixers and their local oscillators characterized by network parameters, noise data, nonlinearity data, and local oscillator frequency.

Use the `read` method to read the mixer data from a data file in one of the following formats:

- Touchstone
- Agilent P2D
- Agilent S2D
- AMP

Note If you set `NonLinearData` using `rfdata.ip3` or `rfdata.power`, then the property is converted from scalar OIP3 format to the format of `rfdata.ip3` or `rfdata.power`.

Creation

Syntax

```
h = rfckt.mixer
h = rfckt.mixer(Name,Value)
```

Description

`h = rfckt.mixer` returns a mixer object whose properties all have their default values.

`h = rfckt.mixer(Name,Value)` sets properties using one or more name-value pairs. For example, `rfckt.mixer('IntpType','cubic')` creates RF mixer with piecewise cubic Hermite interpolation as interpolation method. You can specify multiple name-value pairs. Enclose each property name in a quote. Properties not specified retain their default values.

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as `rfdata.data` object. `AnalyzedResult` is a read-only property. For more information, see .“Algorithms” on page 1-64.

Data Types: `function_handle`

FLO — Local oscillator frequency
positive scalar

Local oscillator frequency, specified as a positive scalar in hertz. If the `MixerType` is set to `'DownConverter'`, the mixer output frequency is $f_{out} = f_{in} - f_{lo}$. If the `MixerType` is set to `'UpConverter'`, the mixer output frequency is $f_{out} = f_{in} + f_{lo}$.

Data Types: `double`

FreqOffset — Frequency offset data
positive vector

Frequency offset data, specified as a positive vector in hertz. The `'FreqOffset'` values correspond to phase noise level values specified by the `'PhaseNoiseLevel'` property. By default, this property is empty.

Data Types: `double`

IntpType — Interpolation method used in `rfckt.mixer`
`1-by-N` character array

Interpolation method used in `rfckt.mixer`, specified as a `1-by-N` character array of the following values:

Method	Description
`Linear` (default)	Linear interpolation
`Spline`	Cubic spline interpolation
`Cubic`	Piecewise cubic Hermite interpolation

Data Types: `char`

MixerSpurData — Data from mixer spur table
`rfdata.mixerspur` object

Data from mixer spur table, specified as an `rfdata.mixerspur` object.

Data Types: `function_handle`

MixerType — Type of mixer
`'DownConverter'` (default) | `'UpConverter'`

Type of mixer, specified as `'DownConverter'` or `'UpConverter'`.

Data Types: `char`

Name — Object name
`1-by-N` character array

Object name, specified as an `1-by-N` character array. `Name` is a read-only property.

Data Types: `char`

NoiseData — Noise information
scalar noise figure in decibels | `rfdata.noise` object | `rfdata.nf` object

Noise information, specified as one of the following:

- Scalar noise figure in dB
- `rfdata.noise` object
- `rfdata.nf` object

Data Types: `double` | `function_handle`

`NonlinearData` — Nonlinearity information
scalar OIP3 in dB | `rfdata.power` object | `rfdata.ip3` object

Noise information, specified as one of the following:

- Scalar OIP3 in dB
- `rfdata.power` object
- `rfdata.ip3` object

Data Types: `double` | `function_handle`

`NetworkData` — Network parameter data
`rfdata.network` object

Network parameter data, specified as a `rfdata.network` object .

Data Types: `double` | `function_handle`

`nport` — Number of ports
positive integer

Number of ports, specified as a positive integer. `nport`t is a read-only property. The default value is `2`.

Data Types: `double`

`PhaseNoiseLevel` — Phase noise data
vector

Phase noise data, specified as a vector in dbc/Hz.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis

smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

RF Mixer

Create an RF mixer using `rfckt.mixer`.

```
rfmixer = rfckt.mixer('IntpType','cubic')
```

```
rfmixer =
   rfckt.mixer with properties:

      MixerSpurData: []
          MixerType: 'Downconverter'
                FLO: 1.0000e+09
         FreqOffset: []
    PhaseNoiseLevel: []
          NoiseData: [1x1 rfdata.noise]
      NonlinearData: Inf
            IntpType: 'Cubic'
        NetworkData: [1x1 rfdata.network]
               nPort: 2
     AnalyzedResult: [1x1 rfdata.data]
               Name: 'Mixer'
```

Algorithms

The `analyze` method computes the `AnalyzedResult` property using the data stored in the `rfckt.amplifier` object properties as follows:

- The `analyze` method uses the data stored in the `'NoiseData'` property of the `rfckt.amplifier` object to calculate the noise figure.
- The `analyze` method uses the data stored in the `'NonlinearData'` property of the `rfckt.amplifier` object to calculate OIP3.

  If power data exists in the `'NonlinearData'` property, the block extracts the AM/AM and AM/PM nonlinearities from the power data.

  If the `'NonlinearData'` property contains only IP3 data, the method computes and adds the nonlinearity by:

  1. Using the third-order input intercept point value in dBm to compute the factor, *f*, that scales the input signal before the amplifier object applies the nonlinearity:

$$F_{AM/AM}(u) = u - \frac{u^3}{3}$$

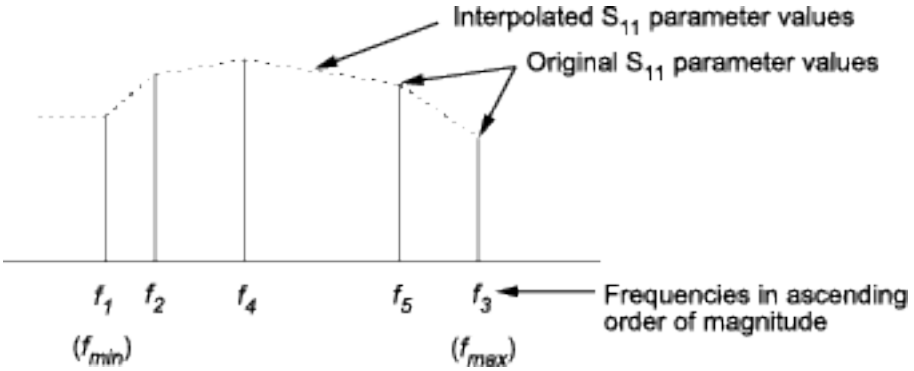
2. Computing the scaled input signal by multiplying the amplifier input signal by *f*.
3. Limiting the scaled input signal to a maximum value of 1.
4. Applying an AM/AM conversion to the amplifier gain, according to the following cubic polynomial equation:

$$F_{AM/AM}(u) = u - \frac{u^3}{3}$$

where *u* is the magnitude of the scaled input signal, which is a unitless normalized input voltage.

- The `analyze` function uses the data stored in the `'NetworkData'` property of the `rfckt.amplifier` object to calculate the group delay values of the amplifier at the frequencies specified in `freq`, as described in the `analyze` function reference page.
- The `analyze` method uses the data stored in the `'NetworkData'` property of the `rfckt.amplifier` object to calculate the S-parameter values of the amplifier at the frequencies specified in `freq`. If the `'NetworkData'` property contains network Y- or Z-parameters, the `analyze` method first converts the parameters to S-parameters. Using the interpolation method you specify with the `'IntpType'` property, the `analyze` method interpolates the S-parameter values to determine their values at the specified frequencies.

  Specifically, the `analyze` method orders the S-parameters according to the ascending order of their frequencies, f_n. It then interpolates the S-parameters, using the MATLAB `interp1` function. For example, the curve in the following diagram illustrates the result of interpolating the S_{11} parameters at five different frequencies.



  For more information, see “One-Dimensional Interpolation” and the `interp1` reference page in the MATLAB documentation.

  As shown in the preceding diagram, the `analyze` method uses the parameter values at f_{min}, the minimum input frequency, for all frequencies smaller than f_{min}. It uses the parameters values at f_{max}, the maximum input frequency, for all frequencies greater than f_{max}. In both cases, the results may not be accurate, so you need to specify network parameter values over a range of frequencies that is wide enough to account for the amplifier behavior.

Version History

Introduced before R2006a

References

[1] EIA/IBIS Open Forum, *Touchstone File Format Specification*, Rev. 1.1, 2002

See Also

`rfckt.amplifier` | `rfckt.cascade` | `rfckt.coaxial` | `rfckt.cpw` | `rfckt.datafile` | `rfckt.delay` | `rfckt.hybrid` | `rfckt.hybridg` | `rfckt.microstrip` | `rfckt.passive` | `rfckt.parallel` | `rfckt.parallelplate` | `rfckt.rlcgline` | `rfckt.series` | `rfckt.seriesrlc` | `rfckt.shuntrlc` | `rfckt.twowire` | `rfckt.txline`

Topics

“Visualize Mixer Spurs”

rfckt.passive

Passive component or network

Description

Use the `passive` class to represent passive RF components and networks that are characterized by passive network parameter data.

Use the `read` method to read the passive object data from a Touchstone data file. When you read S-parameter data into an `rfckt.passive` object, the magnitude of your S_{21} data must be less than or equal to 1.

Due to random numerical error, data measured from a passive device is not necessarily passive. However, `rfckt.passive` objects can only contain passive data. To import data with active regions, use the `rfckt.amplifier` object, even if the original data represents a passive device.

Creation

Syntax

```
h = rfckt.passive
h = rfckt.passive(Name,Value)
```

Description

`h = rfckt.passive` returns an passive-device object whose properties all have their default values.

`h = rfckt.passive(Name,Value)` sets properties using one or more name-value pairs. For example, `rfckt.passive('IntpType','cubic')` creates an passive-device object with piecewise cubic Hermite interpolation as interpolation method. You can specify multiple name-value pairs. Enclose each property name in a quote. Properties not specified retain their default values.

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as `rfdata.data` object. `Analyzed Result` is a read-only property. For more information refer, “Algorithms” on page 1-69.

Data Types: `function_handle`

`IntpType` — Interpolation method used in `rfckt.passive`
`1-by-N` character array

Interpolation method used in `rfckt.passive`, specified as a `1-by-N` character array of the following values:

Method	Description
`Linear` (default)	Linear interpolation
`Spline`	Cubic spline interpolation
`Cubic`	Piecewise cubic Hermite interpolation

Data Types: `char`

`Name` — Object name
`1-by-N` character array

Object name, specified as an `1-by-N` character array. `Name` is a read-only property.

Data Types: `char`

`NetworkData` — Network parameter data
`rfdata.network` object

Network parameter data, specified as a `rfdata.network` object.

Data Types: `function_handle`

`nport` — Number of ports
positive integer

Number of ports, specified as a positive integer. `nport`t is a read-only property. The default value is `2`.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

Passive RF Components

Create passive RF components using `rfckt.passive`.

```
pas = rfckt.passive('IntpType','cubic')
```

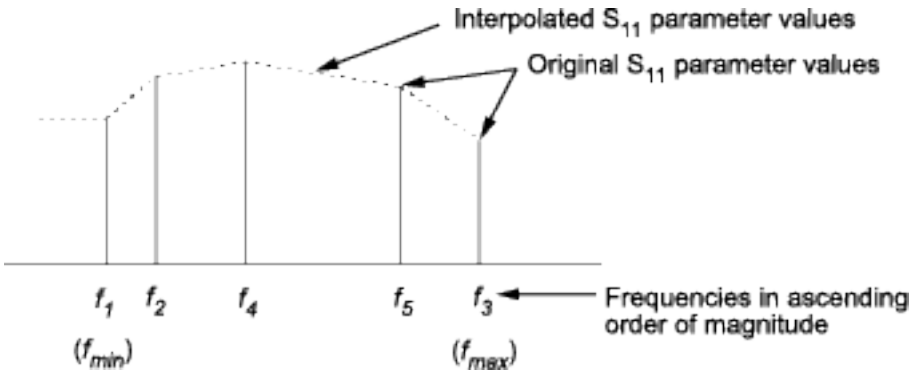
```
pas =
   rfckt.passive with properties:

          IntpType: 'Cubic'
       NetworkData: [1x1 rfdata.network]
             nPort: 2
    AnalyzedResult: [1x1 rfdata.data]
              Name: 'Passive'
```

Algorithms

The `analyze` method computes the `AnalyzedResult` property as follows:

The `analyze` method uses the data stored in the `'NetworkData'` property of the `rfckt.passive` object to calculate the S-parameter values of the passive component at the frequencies specified in `freq`. If the `'NetworkData'` property contains network Y- or Z-parameters, the `analyze` method first converts the parameters to S-parameters. Using the interpolation method you specify with the `'IntpType'` property, the `analyze` method interpolates the S-parameter values to determine their values at the specified frequencies.

Specifically, the `analyze` method orders the S-parameters according to the ascending order of their frequencies, f_n. It then interpolates the S-parameters, using the MATLAB `interp1` function. For example, the curve in the following diagram illustrates the result of interpolating the S_{11} parameters at five different frequencies.



For more information, see “One-Dimensional Interpolation” and the `interp1` reference page in the MATLAB documentation.

As shown in the preceding diagram, the `analyze` method uses the parameter values at f_{min}, the minimum input frequency, for all frequencies smaller than f_{min}. It uses the parameters values at f_{max}, the maximum input frequency, for all frequencies greater than f_{max}. In both cases, the results may not be accurate, so you need to specify network parameter values over a range of frequencies that is wide enough to account for the component behavior.

The `analyze` method uses the S-parameters to calculate the group delay values at the frequencies specified in the `analyze` input argument `freq`, as described in the `analyze` reference page.

Version History

Introduced in R2009a

References

[1] EIA/IBIS Open Forum, *Touchstone File Format Specification*, Rev. 1.1, 2002

See Also

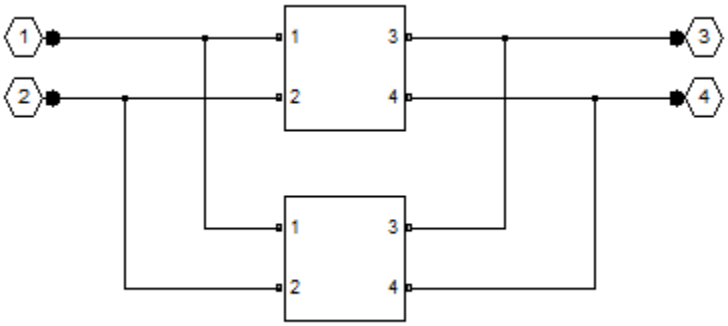
`rfckt.amplifier` | `rfckt.cascade` | `rfckt.coaxial` | `rfckt.cpw` | `rfckt.datafile` | `rfckt.delay` | `rfckt.hybrid` | `rfckt.hybridg` | `rfckt.mixer` | `rfckt.microstrip` | `rfckt.parallel` | `rfckt.parallelplate` | `rfckt.rlcgline` | `rfckt.series` | `rfckt.seriesrlc` | `rfckt.shuntrlc` | `rfckt.twowire` | `rfckt.txline`

rfckt.parallel

Parallel connected network

Description

Use the `parallel` class to represent networks of linear RF objects connected in parallel that are characterized by the components that make up the network. The following figure shows a pair of networks in a parallel configuration.



Creation

Syntax

```
h = rfckt.parallel
h = rfckt.parallel('Ckts',value)
```

Description

`h = rfckt.parallel` returns a parallel connected network object whose properties all have their default values.

`h = rfckt.parallel('Ckts',value)` returns a cascaded network with elements specified in the name-value pair property `Ckts`.

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as rfdata.data object. `Analyzed Result` is a read-only property. For more information, see “Algorithms” on page 1-73.

Data Types: `function_handle`

`Ckts` — Circuit objects in network

cell array of object handles

Circuit objects in network, specified as a cell array of object handles. All circuits must be 2-port. By default, this property is empty.

Data Types: `char`

`Name` — Object name

`1-by-N` character array

Object name, specified as an `1-by-N` character array. `Name` is a read-only property.

Data Types: `char`

`nport` — Number of ports

positive integer

Number of ports, specified as a positive integer. `nportt` is a read-only property. The default value is `2`.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

Network of RF Objects In Parallel

Create a network of transmission lines connected in parallel using `rfckt.parallel`.

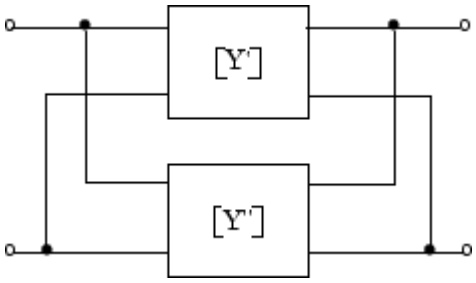
```
tx1 = rfckt.txline;
tx2 = rfckt.txline;
rfplel = rfckt.parallel('Ckts',{tx1,tx2})
```

```
rfplel =
   rfckt.parallel with properties:

            Ckts: {[1x1 rfckt.txline]  [1x1 rfckt.txline]}
           nPort: 2
    AnalyzedResult: []
            Name: 'Parallel Connected Network'
```

Algorithms

The `analyze` method computes the S-parameters of the `AnalyzedResult` property using the data stored in the `Ckts` property as follows:

1. The `analyze` method first calculates the admittance matrix of the parallel connected network. It starts by converting each component network's parameters to an admittance matrix. The following figure shows a parallel connected network consisting of two 2-port networks, each represented by its admittance matrix.



   where

$$[Y'] = \begin{bmatrix} Y_{11}' & Y_{12}' \\ Y_{21}' & Y_{22}' \end{bmatrix}$$

$$[Y''] = \begin{bmatrix} Y_{11}'' & Y_{12}'' \\ Y_{21}'' & Y_{22}'' \end{bmatrix}$$

2. The `analyze` method then calculates the admittance matrix for the parallel network by calculating the sum of the individual admittances. The following equation illustrates the calculations for two 2-port circuits.

$$[Y] = [Y'] + [Y''] = \begin{bmatrix} Y_{11}' + Y_{11}'' & Y_{12}' + Y_{12}'' \\ Y_{21}' + Y_{21}'' & Y_{22}' + Y_{22}'' \end{bmatrix}$$

3. Finally, `analyze` converts the admittance matrix of the parallel network to S-parameters at the frequencies specified in the `analyze` input argument `freq`.

Version History

Introduced before R2006a

References

[1] Ludwig, R. and P. Bretchko, *RF Circuit Design: Theory and Applications*, Prentice-Hall, 2000.

See Also

`rfckt.amplifier` | `rfckt.cascade` | `rfckt.coaxial` | `rfckt.cpw` | `rfckt.datafile` | `rfckt.delay` | `rfckt.hybrid` | `rfckt.hybridg` | `rfckt.mixer` | `rfckt.microstrip` | `rfckt.passive` | `rfckt.parallelplate` | `rfckt.rlcgline` | `rfckt.series` | `rfckt.seriesrlc` | `rfckt.shuntrlc` | `rfckt.twowire` | `rfckt.txline`

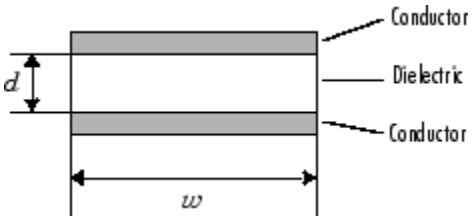
rfckt.parallelplate

Parallel-plate transmission line

Description

Use the `parallelplate` class to represent parallel-plate transmission lines that are characterized by line dimensions and optional stub properties.

A parallel-plate transmission line is shown in cross-section in the following figure. Its physical characteristics include the plate width *w* and the plate separation *d*.



Creation

Syntax

```
h = rfckt.parallelplate
h = rfckt.parallelplate(Name,Value)
```

Description

`h = rfckt.parallelplate` returns a parallel-plate transmission line object whose properties are set to their default values.

`h = rfckt.parallelplate(Name,Value)` sets properties using one or more name-value pairs. For example, `rfckt.parallelplate('LineLength',0.045)` creates a parallel-plate transmission line object with a physical length of 0.045 meters. You can specify multiple name-value pairs. Enclose each property name in a quote. Properties not specified retain their default values.

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as `rfdata.data` object. `AnalyzedResult` is a read-only property. For more information refer, “Algorithms” on page 1-78.

Data Types: `function_handle`

EpsilonR — Relative permittivity of dielectric
scalar

Relative permittivity of dielectric, specified as a scalar. The relative permittivity is the ratio of permittivity of the dielectric,ε, to the permittivity in free space, ε_0. The default value is `2.3`.

Data Types: `double`

LineLength — Physical length of parallel-plate transmission line
scalar

Physical length of parallel-plate transmission line, specified as a scalar in meters. The default value is `0.01`.

Data Types: `double`

LossTangent — Tangent of loss angle of dielectric
scalar

Tangent of loss angle of dielectric, specified as a scalar. The default value is `0`.

Data Types: `double`

SigmaCond — Conductor conductivity
scalar in Siemens per meter

Conductor conductivity, specified as a scalar in Siemens per meter (S/m). The default value is `Inf`.

Data Types: `double`

MUR — Relative permeability of dielectric
scalar

Relative permeability of dielectric, specified as a scalar. The ratio of permeability of dielectric, μ, to the permeability in free space, μ_0. The default value is `1`.

Data Types: `double`

Name — Object name
`1-by-N` character array

Object name, specified as an `1-by-N` character array. `Name` is a read-only property.

Data Types: `char`

nport — Number of ports
positive integer

Number of ports, specified as a positive integer. `nportt` is a read-only property. The default value is `2`.

Data Types: `double`

Separation — Thickness of dielectric
scalar

Thickness of the dielectric separating the plates, specified as a scalar in meters. The default value is `1.0e-3`.

Data Types: `double`

`StubMode` — Type of stub
`'NotaStub'` (default) | `'Series'` | `'Shunt'`

Type of stub, specified as one of the following values: `'NotaStub'`, `'Series'`, `'Shunt'`.

Data Types: `double`

`Termination` — Stub transmission line termination
`'NotApplicable'` (default) | `'Open'` | `'Short'`

Stub transmission line termination, specified as one of the following values: `'NotApplicable'`, `'Open'`, `'Short'`.

Data Types: `double`

`Width` — Physical width of parallel-plate transmission line
scalar

Physical width of parallel-plate transmission line, specified as a scalar in meters. The default value is `6.0e-4`.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

Parallel Plate Transmission Line

Create a parallel plate transmission line using `rfckt.parallelplate`.

```
tx1=rfckt.parallelplate('LineLength',0.045)
```

```
tx1 =
   rfckt.parallelplate with properties:

           Width: 0.0050
      Separation: 1.0000e-03
             MuR: 1
         EpsilonR: 2.3000
      LossTangent: 0
        SigmaCond: Inf
       LineLength: 0.0450
         StubMode: 'NotAStub'
      Termination: 'NotApplicable'
            nPort: 2
   AnalyzedResult: []
             Name: 'Parallel-Plate Transmission Line'
```

Algorithms

The `analyze` method treats the parallel-plate line as a 2-port linear network and models the line as a transmission line with optional stubs. The `analyze` method computes the `AnalyzedResult` property of the line using the data stored in the `rfckt.parallelplate` object properties as follows:

- If you model the transmission line as a stub less line, the `analyze` method first calculates the ABCD-parameters at each frequency contained in the modeling frequencies vector. It then uses the `abcd2s` function to convert the ABCD-parameters to S-parameters.

  The `analyze` method calculates the ABCD-parameters using the physical length of the transmission line, d, and the complex propagation constant, k, using the following equations:

  $$A = \frac{e^{kd} + e^{-kd}}{2}$$

  $$B = \frac{Z_0 * \left(e^{kd} - e^{-kd}\right)}{2}$$

  $$C = \frac{e^{kd} - e^{-kd}}{2 * Z_0}$$

  $$D = \frac{e^{kd} + e^{-kd}}{2}$$

  Z_0 and k are vectors whose elements correspond to the elements of f, the vector of frequencies specified in the `analyze` input argument `freq`. Both can be expressed in terms of the resistance (R), inductance (L), conductance (G), and capacitance (C) per unit length (meters) as follows:

  $$Z_0 = \sqrt{\frac{R + j2\pi fL}{G + j2\pi fC}}$$

  $$k = k_r + jk_i = \sqrt{(R + j2\pi fL)(G + j2\pi FC)}$$

  where

$$R = \frac{2}{w\sigma_{cond}\delta_{cond}}$$

$$L = \mu\frac{d}{w}$$

$$G = \omega\varepsilon''\frac{w}{d}$$

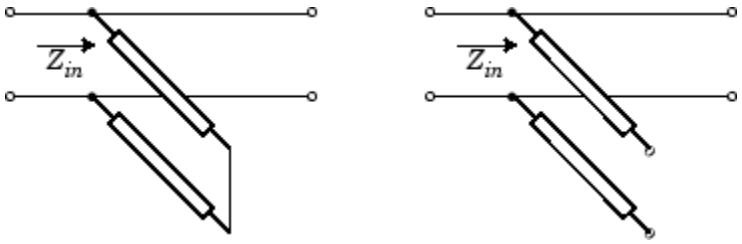
$$C = \varepsilon\frac{w}{d}$$

In these equations:

- w is the plate width.
- d is the plate separation.
- σ_{cond} is the conductivity in the conductor.
- μ is the permeability of the dielectric.
- ε is the permittivity of the dielectric.
- ε'' is the imaginary part of ε, $\varepsilon'' = \varepsilon_0\varepsilon_r\tan\delta$, where:
  - ε_0 is the permittivity of free space.
  - ε_r is the `EpsilonR` property value.
  - $\tan\delta$ is the `LossTangent` property value.
- δ_{cond} is the skin depth of the conductor, which the block calculates as $1/\sqrt{\pi f\mu\sigma_{cond}}$.
- f is a vector of modeling frequencies determined by the Outport block.

- If you model the transmission line as a shunt or series stub, the `analyze` method first calculates the ABCD-parameters at the specified frequencies. It then uses the `abcd2s` function to convert the ABCD-parameters to S-parameters.

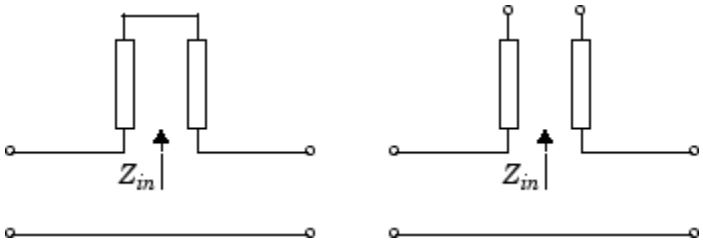
  When you set the `StubMode` property to `'Shunt'`, the 2-port network consists of a stub transmission line that you can terminate with either a short circuit or an open circuit as shown in the following figure.



  Z_{in} is the input impedance of the shunt circuit. The ABCD-parameters for the shunt stub are calculated as:

$$A = 1$$
$$B = 0$$
$$C = 1/Z_{in}$$
$$D = 1$$

When you set the `StubMode` property to `'Series'`, the 2-port network consists of a series transmission line that you can terminate with either a short circuit or an open circuit as shown in the following figure.



Z_{in} is the input impedance of the series circuit. The ABCD-parameters for the series stub are calculated as:

$$A = 1$$
$$B = Z_{in}$$
$$C = 0$$
$$D = 1$$

Version History

Introduced in R2009a

References

[1] Pozar, David M. *Microwave Engineering*, John Wiley & Sons, Inc., 2005.

See Also

`rfckt.amplifier` | `rfckt.cascade` | `rfckt.coaxial` | `rfckt.cpw` | `rfckt.datafile` | `rfckt.delay` | `rfckt.hybrid` | `rfckt.hybridg` | `rfckt.mixer` | `rfckt.microstrip` | `rfckt.passive` | `rfckt.parallel` | `rfckt.rlcgline` | `rfckt.series` | `rfckt.seriesrlc` | `rfckt.shuntrlc` | `rfckt.twowire` | `rfckt.txline`

rfckt.rlcgline

Passive component or network

Description

Use the `rlcgline` object to represent RLCG transmission lines that are characterized by line loss, line length, stub type, and termination.

Creation

Syntax

```
h = rfckt.rlcgline
h = rfckt.rlcgline(Name,Value)
```

Description

`h = rfckt.rlcgline` returns an RLCG transmission line object whose properties are set to their default values.

`h = rfckt.rlcgline(Name,Value)` sets properties using one or more name-value pairs. For example, `rfckt.rlcgline('LineLenght',0.04)` creates a RLGC transmission line with a physical length of 0.04 meters. You can specify multiple name-value pairs. Enclose each property name in a quote. Properties not specified retain their default values.

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as `rfdata.data` object. `AnalyzedResult` is a read-only property. For more information refer, ."Algorithms" on page 1-84.

Data Types: `function_handle`

`R` — Resistance values per length
vector

Resistance values per length, specified as a vector in ohms per meter. The resistance values correspond to the frequency values in `'Freq'` property. All values must be positive. The default value is `0`.

Data Types: `double`

`C` — Capacitance values per length
vector

Capacitance values per length, specified as a vector in farads per meter. The capacitance values correspond to the frequency values in `'Freq'` property. All values must be positive. The default value is `0`.

Data Types: `double`

`Freq` — Frequency data
M-element vector

Frequency data for the RLCG values, specified as a *M*-element vector. The values must be positive and correspond to the order of the RLCG values. The default value is `1e9`.

Data Types: `double`

`G` — Conductance values per length
vector

Conductance values per length, specified as a vector in Siemens per meter. The conductance values correspond to the frequency values in `'Freq'` property. All values must be positive. The default value is `0`.

Data Types: `double`

`IntpType` — Interpolation method used in `rfckt.rlcgline`
`'Linear'` (default) | `'Spline'` | `'Cubic'`

Interpolation method used in `rfckt.rlcgline`, specified as one of the following values:

Method	Description
`Linear`	Linear interpolation
`Spline`	Cubic spline interpolation
`Cubic`	Piecewise cubic Hermite interpolation

Data Types: `char`

`L` — Inductance values per length
vector

Inductance values per length, specified as vector in henries per meter. The inductance values correspond to the frequency values in `'Freq'` property. All values must be positive. The default value is `0`.

Data Types: `double`

`LineLength` — Physical length of transmission line
scalar

Physical length of transmission line, specified as a scalar in meters. The default value is `0.01`.

Data Types: `double`

`Name` — Object name
`'RLCG Transmission Line'` (default) | `1-by-N` character array

Object name, specified as a `1-by-N` character array. `Name` is a read-only property.

Data Types: `char`

`nport` — Number of ports
positive integer

Number of ports, specified as a positive integer. `nportt` is a read-only property. The default value is `2`.

Data Types: `double`

`StubMode` — Type of stub
`'NotaStub'` (default) | `'Series'` | `'Shunt'`

Type of stub, specified as one of the following values: `'NotaStub'`, `'Series'`, `'Shunt'`.

Data Types: `double`

`Termination` — Stub transmission line termination
`'NotApplicable'` (default) | `'Open'` | `'Short'`

Stub transmission line termination, specified as one of the following values: `'NotApplicable'`, `'Open'`, `'Short'`.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

RLCG Transmission Line

Create an RLCG transmission line using `rfckt.rlcgline`.

```
rlcgtx=rfckt.rlcgline('R',0.002,'C',8.8542e-12,'L',1.2566e-6,'G',0.002')
```

```
rlcgtx =
   rfckt.rlcgline with properties:

              Freq: 1.0000e+09
                 R: 0.0020
                 L: 1.2566e-06
                 C: 8.8542e-12
                 G: 0.0020
          IntpType: 'Linear'
        LineLength: 0.0100
          StubMode: 'NotAStub'
       Termination: 'NotApplicable'
             nPort: 2
    AnalyzedResult: []
              Name: 'RLCG Transmission Line'
```

Algorithms

The `analyze` method treats the transmission line, which can be lossy or lossless, as a 2-port linear network. It uses the interpolation method you specify in the `IntpType` property to find the R, L, C, and G values at the frequencies you specify when you call `analyze`. Then, it calculates the characteristic impedance, Z0, phase velocity, PV, and loss using these interpolated values. It computes the `AnalyzedResult` property of a stub or as a stubless line using the data stored in the `rfckt.rlcgline` object properties as follows:

- If you model the transmission line as a stubless line, the `analyze` method first calculates the ABCD-parameters at each frequency contained in the modeling frequencies vector. It then uses the `abcd2s` function to convert the ABCD-parameters to S-parameters.

  The `analyze` method calculates the ABCD-parameters using the physical length of the transmission line, d, and the complex propagation constant, k, using the following equations:

  $$A = \frac{e^{kd} + e^{-kd}}{2}$$

  $$B = \frac{Z_0 * \left(e^{kd} - e^{-kd}\right)}{2}$$

  $$C = \frac{e^{kd} - e^{-kd}}{2 * Z_0}$$

  $$D = \frac{e^{kd} + e^{-kd}}{2}$$

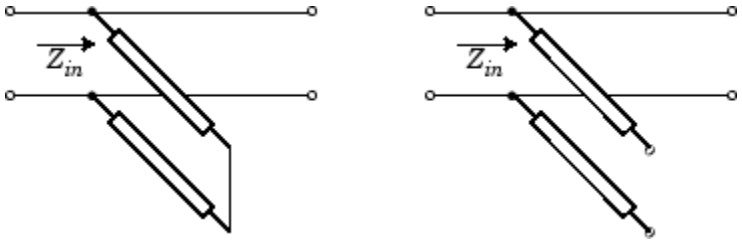
  Z_0 and k are vectors whose elements correspond to the elements of f, the vector of frequencies specified in the `analyze` input argument `freq`. Both can be expressed in terms of the resistance (R), inductance (L), conductance (G), and capacitance (C) per unit length (meters) as follows:

  $$Z_0 = \sqrt{\frac{R + j2\pi fL}{G + j2\pi fC}}$$

  $$k = k_r + jk_i = \sqrt{(R + j2\pi fL)(G + j2\pi FC)}$$

- If you model the transmission line as a shunt or series stub, the `analyze` method first calculates the ABCD-parameters at the specified frequencies. It then uses the `abcd2s` function to convert the ABCD-parameters to S-parameters.

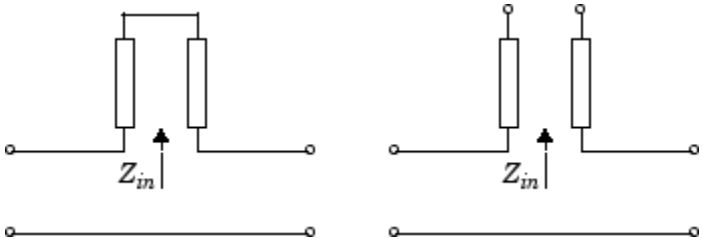
  When you set the `StubMode` property to `'Shunt'`, the 2-port network consists of a stub transmission line that you can terminate with either a short circuit or an open circuit as shown in the following figure.



  Z_{in} is the input impedance of the shunt circuit. The ABCD-parameters for the shunt stub are calculated as:

$$A = 1$$
$$B = 0$$
$$C = 1/Z_{in}$$
$$D = 1$$

  When you set the `StubMode` property to `'Series'`, the 2-port network consists of a series transmission line that you can terminate with either a short circuit or an open circuit as shown in the following figure.



  Z_{in} is the input impedance of the series circuit. The ABCD-parameters for the series stub are calculated as:

$$A = 1$$
$$B = Z_{in}$$
$$C = 0$$
$$D = 1$$

The `analyze` method uses the S-parameters to calculate the group delay values at the frequencies specified in the `analyze` input argument `freq`, as described in the `analyze` reference page.

Version History

Introduced in R2009a

References

[1] Ludwig, R. and P. Bretchko, *RF Circuit Design: Theory and Applications*, Prentice-Hall, 2000

See Also

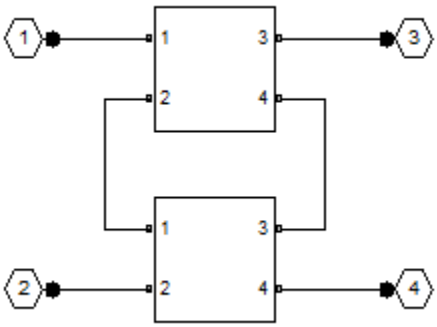
`rfckt.amplifier` | `rfckt.cascade` | `rfckt.coaxial` | `rfckt.cpw` | `rfckt.datafile` | `rfckt.delay` | `rfckt.hybrid` | `rfckt.hybridg` | `rfckt.mixer` | `rfckt.microstrip` | `rfckt.passive` | `rfckt.parallel` | `rfckt.parallelplate` | `rfckt.series` | `rfckt.seriesrlc` | `rfckt.shuntrlc` | `rfckt.twowire` | `rfckt.txline`

rfckt.series

Series connected network

Description

Use the `series` class to represent networks of linear RF objects connected in series that are characterized by the components that make up the network. The following figure shows a pair of networks in a series configuration.



Creation

Syntax

```
h = rfckt.series
h = rfckt.series('Ckts',value)
```

Description

`h = rfckt.series` returns a series connected network object whose properties all have their default values.

`h = rfckt.series('Ckts',value)` returns a series connected network with elements specified in the name-value pair property `Ckts`.

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as rfdata.data object. `Analyzed Result` is a read-only property. For more information, see “Algorithms” on page 1-89.

Data Types: `function_handle`

`Ckts` — Circuit objects in network

cell array of object handles

Circuit objects in network, specified as a cell array of object handles. All circuits must be 2-port. By default, this property is empty.

Data Types: `char`

`Name` — Object name

`'RLCG Transmission Line'` (default) | `1-by-N` character array

Object name, specified as a `1-by-N` character array. `Name` is a read-only property.

Data Types: `char`

`nport` — Number of ports

positive integer

Number of ports, specified as a positive integer. `nport` is a read-only property. The default value is `2`.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

Series Connected RF Network Object

Create a series connected RF network object using `rfckt.series`

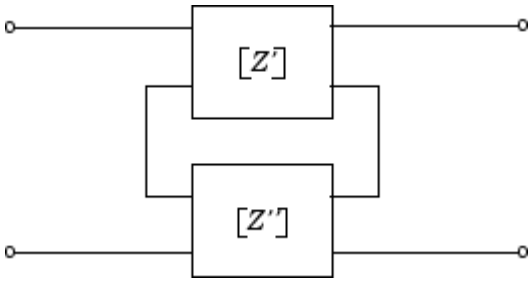
```
tx1 = rfckt.txline;
tx2 = rfckt.txline;
ser = rfckt.series('Ckts',{tx1,tx2})
```

```
ser =
   rfckt.series with properties:

            Ckts: {[1x1 rfckt.txline]  [1x1 rfckt.txline]}
           nPort: 2
    AnalyzedResult: []
            Name: 'Series Connected Network'
```

Algorithms

The `analyze` method computes the S-parameters of the `AnalyzedResult` property using the data stored in the `Ckts` property as follows:

1. The `analyze` method first calculates the impedance matrix of the series connected network. It starts by converting each component network's parameters to an impedance matrix. The following figure shows a series connected network consisting of two 2-port networks, each represented by its impedance matrix.



   where

$$[Z'] = \begin{bmatrix} Z_{11}' & Z_{12}' \\ Z_{21}' & Z_{22}' \end{bmatrix}$$

$$[Z''] = \begin{bmatrix} Z_{11}'' & Z_{12}'' \\ Z_{21}'' & Z_{22}'' \end{bmatrix}$$

2. The `analyze` method then calculates the impedance matrix for the series network by calculating the sum of the individual impedances. The following equation illustrates the calculations for two 2-port circuits.

$$[Z] = [Z'] + [Z''] = \begin{bmatrix} Z_{11}' + Z_{11}'' & Z_{12}' + Z_{12}'' \\ Z_{21}' + Z_{21}'' & Z_{22}' + Z_{22}'' \end{bmatrix}$$

3. Finally, `analyze` converts the impedance matrix of the series network to S-parameters at the frequencies specified in the analyze input argument `freq`.

Version History

Introduced in R2009a

References

[1] Ludwig, Reinhold and Pavel Bretchko, *RF Circuit Design: Theory and Applications*, Prentice-Hall, 2000.

See Also

`rfckt.amplifier` | `rfckt.cascade` | `rfckt.coaxial` | `rfckt.cpw` | `rfckt.datafile` | `rfckt.delay` | `rfckt.hybrid` | `rfckt.hybridg` | `rfckt.mixer` | `rfckt.microstrip` | `rfckt.passive` | `rfckt.parallel` | `rfckt.parallelplate` | `rfckt.rlcgline` | `rfckt.seriesrlc` | `rfckt.shuntrlc` | `rfckt.twowire` | `rfckt.txline`

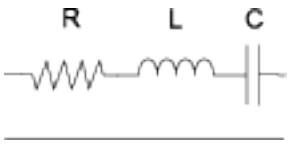
rfckt.seriesrlc

Series RLC component

Description

Use the `seriesrlc` class to represent a component as a resistor, inductor, and capacitor connected in series.

The series RLC network object is a 2-port network as shown in the following circuit diagram.



Creation

Syntax

```
h = rfckt.seriesrlc
h = rfckt.seriesrlc('R',Rvalue,'L',Lvalue,'C',Cvalue)
```

Description

`h = rfckt.seriesrlc` returns a series RLC network object whose properties all have their default values. The default object is equivalent to a pass-through 2-port network, i.e., the resistor, inductor, and capacitor are each replaced by a short circuit.

`h = rfckt.seriesrlc('R',Rvalue,'L',Lvalue,'C',Cvalue)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as rfdata.data object. `Analyzed Result` is a read-only property. For more information refer, “Algorithms” on page 1-89.

Data Types: `function_handle`

`R` — Resistance value
positive scalar

Resistance value, specified as a positive scalar in ohms. The default value is `0`.

Data Types: `double`

C — Capacitance value
positive scalar

Capacitance value, specified as a positive scalar in farads. The default value is `'Inf'`.

Data Types: `double`

L — Inductance value
positive scalar

Inductance value, specified as a positive scalar in henries. The default value is `0`.

Data Types: `double`

Name — Object name
`'RLCG Transmission Line'` (default) | `1-by-N` character array

Object name, specified as a `1-by-N` character array. `Name` is a read-only property.

Data Types: `char`

nport — Number of ports
positive integer

Number of ports, specified as a positive integer. `nport`t is a read-only property. The default value is `2`.

Data Types: `double`

Object Functions

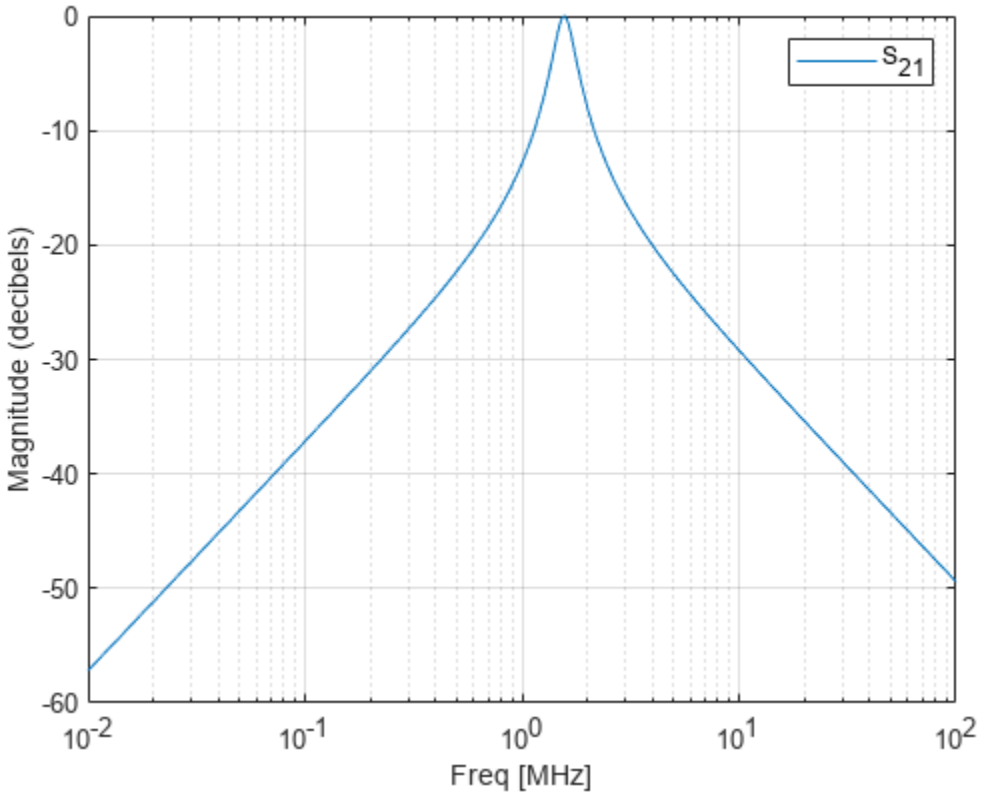
analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

Frequency Response of an LC Resonator

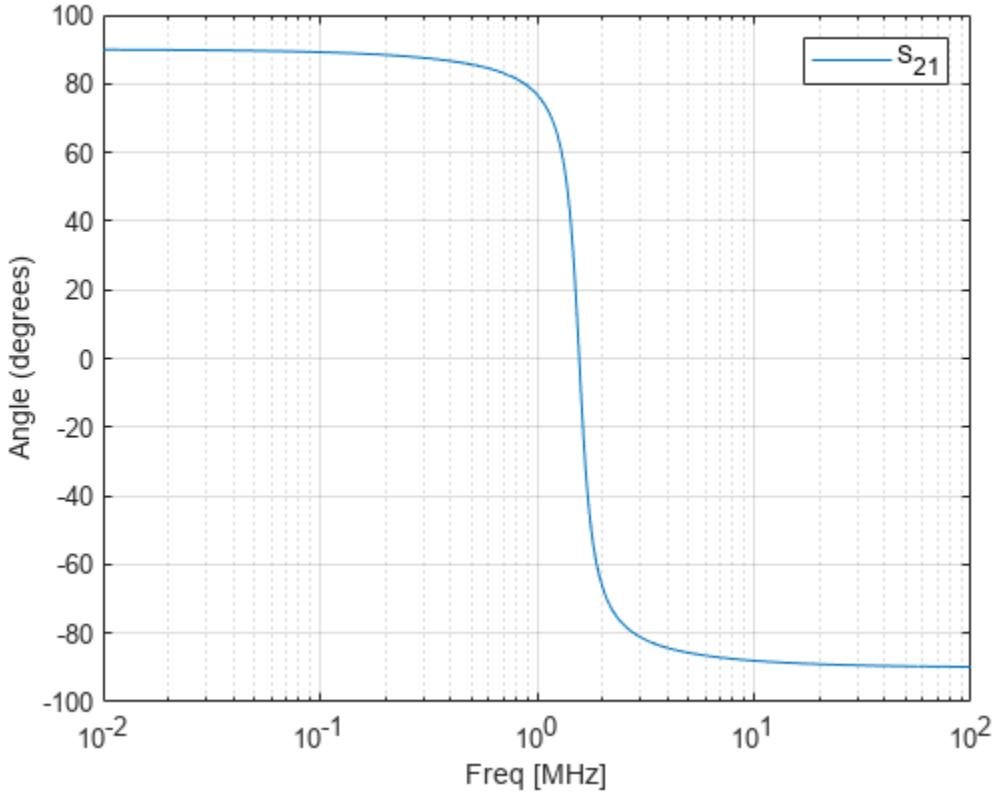
This example creates a series LC resonator and examines its frequency response. It first creates the circuit object and then uses the analyze method to calculate its frequency response. Finally, it plots the results - first, the magnitude in decibels (dB):

```
h = rfckt.seriesrlc('L',4.7e-5,'C',2.2e-10);
analyze(h,logspace(4,8,1000));
plot(h,'s21','dB')
ax = gca;
ax.XScale = 'log';
```



The example then plots the phase, in degrees:

```
figure
plot(h,'s21','angle')
ax = gca;
ax.XScale = 'log';
```



Algorithms

The `analyze` method computes the S-parameters of the `AnalyzedResult` property using the data stored in the `rfckt.seriesrlc` object properties by first calculating the ABCD-parameters for the circuit, and then converting the ABCD-parameters to S-parameters using the `abcd2s` function. For this circuit, A = 1, B = Z, C = 0, and D = 1, where

$$Z = \frac{-LC\omega^2 + jRC\omega + 1}{jC\omega}$$

and $\omega = 2\pi f$.

The `analyze` method uses the S-parameters to calculate the group delay values at the frequencies specified in the `analyze` input argument `freq`, as described in the `analyze` reference page.

Version History

Introduced in R2009a

References

[1] Ludwig, Reinhold and Pavel Bretchko, *RF Circuit Design: Theory and Applications*, Prentice-Hall, 2000.

See Also

rfckt.amplifier | rfckt.cascade | rfckt.coaxial | rfckt.cpw | rfckt.datafile | rfckt.delay | rfckt.hybrid | rfckt.hybridg | rfckt.mixer | rfckt.microstrip | rfckt.passive | rfckt.parallel | rfckt.parallelplate | rfckt.rlcgline | rfckt.series | rfckt.shuntrlc | rfckt.twowire | rfckt.txline

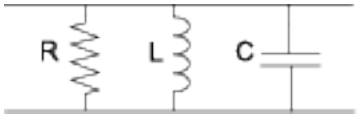
rfckt.shuntrlc

Shunt RLC component

Description

Use the `shuntrlc` class to represent a component as a resistor, inductor, and capacitor connected in a shunt configuration.

The shunt RLC network object is a 2-port network as shown in the following circuit diagram.



Creation

Syntax

```
h = rfckt.shuntrlc
h = rfckt.shuntrlc('R',Rvalue,'L',Lvalue,'C',Cvalue)
```

Description

`h = rfckt.shuntrlc` returns a shunt RLC network object whose properties all have their default values. The default object is equivalent to a pass-through 2-port network; i.e., the resistor, inductor, and capacitor are each replaced by a short circuit.

`h = rfckt.shuntrlc('R',Rvalue,'L',Lvalue,'C',Cvalue)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as rfdata.data object. `Analyzed Result` is a read-only property. For more information refer, “Algorithms” on page 1-99.

Data Types: `function_handle`

R — Resistance value
nonnegative scalar

Resistance value, specified as a positive scalar in ohms. The default value is `0`.

Data Types: `double`

`C` — Capacitance value
nonnegative scalar

Capacitance value, specified as a positive scalar in farads. The default value is `0`.

Data Types: `double`

`L` — Inductance value
positive scalar

Inductance value, specified as a positive scalar in henries. The default value is `'Inf'`.

Data Types: `double`

`Name` — Object name
`'RLCG Transmission Line'` (default) | `1-by-N` character array

Object name, specified as a `1-by-N` character array. `Name` is a read-only property.

Data Types: `char`

`nport` — Number of ports
positive integer

Number of ports, specified as a positive integer. `nport`t is a read-only property. The default value is `2`.

Data Types: `double`

Object Functions

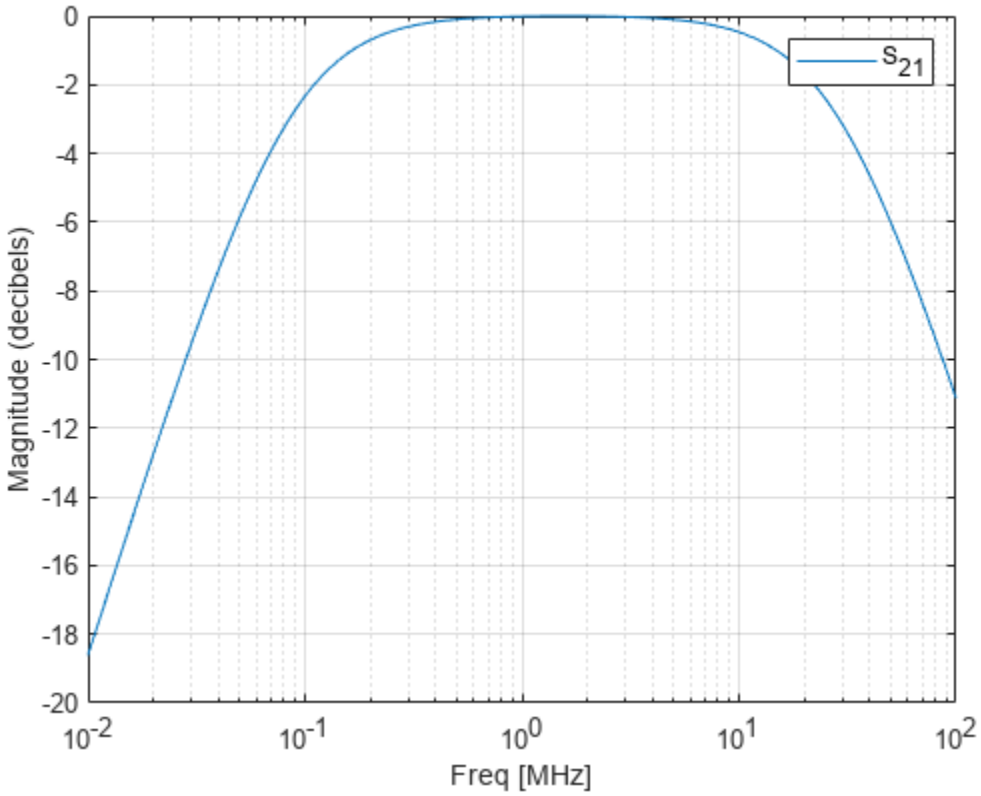
analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

Frequency Response of a Shunt LC Resonator

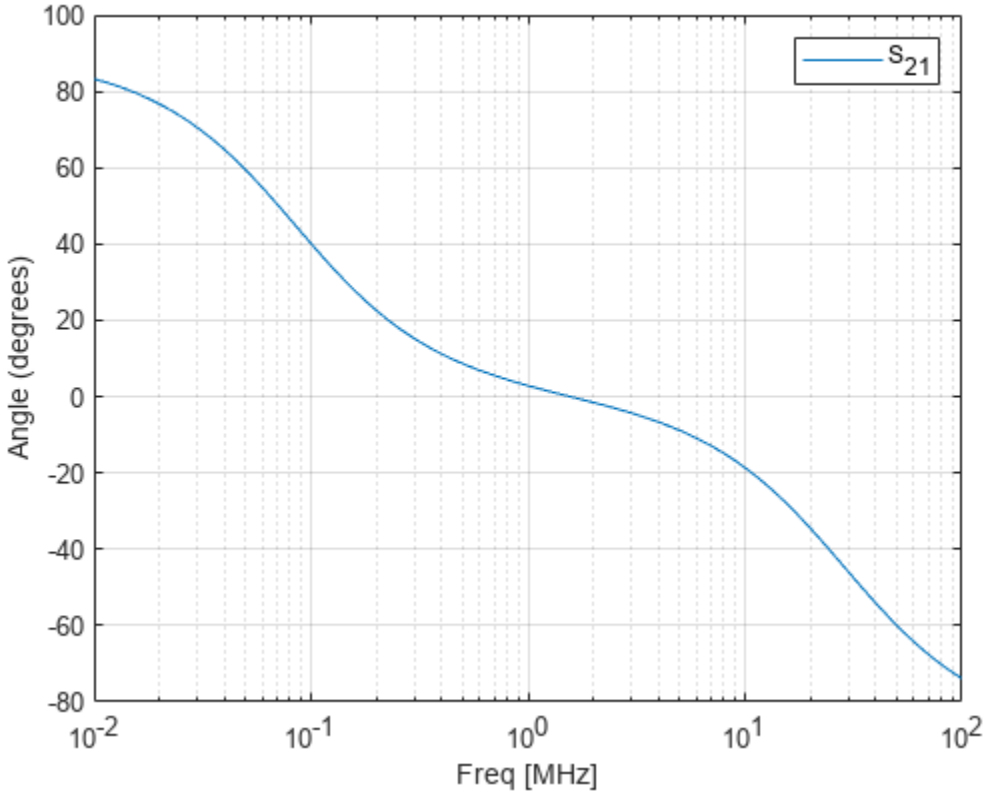
This example creates a shunt LC resonator and examines its frequency response. It first creates the circuit object and then uses the analyze method to calculate its frequency response. The plot is in decibels(dB).

```
h = rfckt.shuntrlc('L',4.7e-5,'C',2.2e-10);
analyze(h,logspace(4,8,1000));
plot(h,'s21','dB')
ax = gca;
ax.XScale = 'log';
```



The example then plots the phase, in degrees:

```
figure
plot(h,'s21','angle')
ax = gca;
ax.XScale = 'log';
```



Algorithms

The `analyze` method computes the S-parameters of the `AnalyzedResult` property using the data stored in the `rfckt.shuntrlc` object properties by first calculating the ABCD-parameters for the circuit, and then converting the ABCD-parameters to S-parameters using the `abcd2s` function. For this circuit, A = 1, B = 0, C = Y, and D = 1, where

$$Y = \frac{-LC\omega^2 + j(L/R)\omega + 1}{jL\omega}$$

and $\omega = 2\pi f$.

The `analyze` method uses the S-parameters to calculate the group delay values at the frequencies specified in the `analyze` input argument `freq`, as described in the `analyze` reference page.

Version History

Introduced in R2009a

References

[1] Ludwig, Reinhold and Pavel Bretchko, *RF Circuit Design: Theory and Applications*, Prentice-Hall, 2000.

See Also

rfckt.amplifier | rfckt.cascade | rfckt.coaxial | rfckt.cpw | rfckt.datafile | rfckt.delay | rfckt.hybrid | rfckt.hybridg | rfckt.mixer | rfckt.microstrip | rfckt.passive | rfckt.parallel | rfckt.parallelplate | rfckt.rlcgline | rfckt.series | rfckt.seriesrlc | rfckt.twowire | rfckt.txline

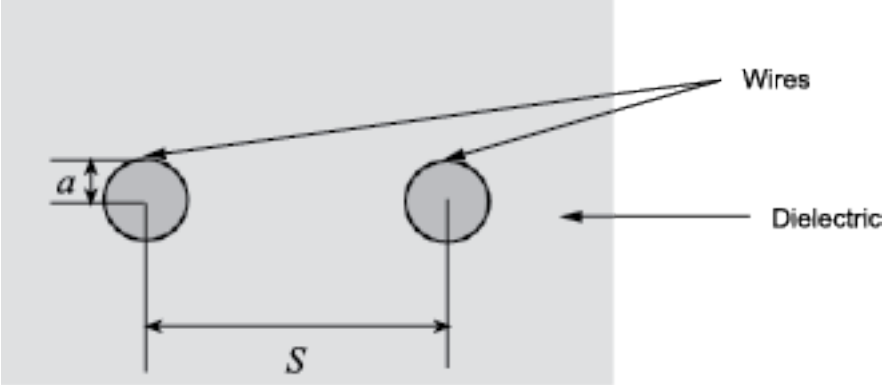
rfckt.twowire

Two-wire transmission line

Description

Use the `twowire` class to represent two-wire transmission lines that are characterized by line dimensions, stub type, and termination.

A two-wire transmission line is shown in cross-section in the following figure. Its physical characteristics include the radius of the wires *a*, the separation or physical distance between the wire centers *S*, and the relative permittivity and permeability of the wires. RF Toolbox software assumes the relative permittivity and permeability are uniform.



Creation

Syntax

```
h = rfckt.twowire
h = rfckt.twowire(Name,Value)
```

Description

`h = rfckt.twowire` returns a shunt RLC network object whose properties all have their default values. The default object is equivalent to a pass-through 2-port network; i.e., the resistor, inductor, and capacitor are each replaced by a short circuit.

`h = rfckt.twowire(Name,Value)` sets properties using one or more name-value pairs. For example, `rfckt.twowire('Radius',7.5e-4)` creates a two-wire transmission line with conducting wire radius of $7.5e^{-4}$ meters. You can specify multiple name-value pairs. Enclose each property name in a quote. Properties not specified retain their default values.

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as a `rfdata.data` object. This is a read-only property. For more information refer, “Algorithms” on page 1-104.

Data Types: `function_handle`

`Separation` — Distance between two wire centers
scalar

The separation or physical distance between the wire centers, specified as a scalar in meters. The default value is `0.0016`.

Data Types: `double`

`EpsilonR` — Relative permittivity of dielectric
scalar

Relative permittivity of dielectric, specified as a scalar. The relative permittivity is the ratio of permittivity of the dielectric,ε, to the permittivity in free space, ε_0. The default value is `2.3`.

Data Types: `double`

`LineLength` — Physical length of transmission line
scalar

Physical length of transmission line, specified as a scalar in meters. The default value is `0.01`.

Data Types: `double`

`LossTangent` — Tangent of loss angle of dielectric
scalar

Tangent of loss angle of dielectric, specified as a scalar. The default value is `0`.

Data Types: `double`

`MUR` — Relative permeability of dielectric
scalar

Relative permeability of dielectric, specified as a scalar. The ratio of permeability of dielectric, μ, to the permeability in free space,

$$\mu_0$$

. The default value is `1`.

Data Types: `double`

`Name` — Object name
`'Two-Wire Transmission Line'` (default) | `1-by-N` character array

Object name, specified as a `1-by-N` character array. This is a read-only property.

Data Types: `char`

`nport` — Number of ports
positive integer

Number of ports, specified as a positive integer. This is a read-only property. The default value is `2`.

Data Types: `double`

`Radius` — Conducting wire radius
scalar

Conducting wire radius, specified as a scalar in meters. The default value is `6.7e-4`.

Data Types: `double`

`SigmaCond` — Conductor conductivity
scalar in Siemens per meter

Conductor conductivity, specified as a scalar in Siemens per meter (S/m). The default value is `Inf`.

Data Types: `double`

`StubMode` — Type of stub
`'NotaStub'` (default) | `'Series'` | `'Shunt'`

Type of stub, specified as a one of the following values: `'NotaStub'`, `'Series'`, `'Shunt'`.

Data Types: `double`

`Termination` — Stub transmission line termination
`'NotApplicable'` (default) | `'Open'` | `'Short'`

Stub transmission line termination, specified as one of the following values: `'NotApplicable'`, `'Open'`, `'Short'`.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies

getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

Two-Wire Transmission Line

Create a two-wire transmission line object using `rfckt.twowire`.

```
tx1=rfckt.twowire('Radius',7.5e-4)
```

```
tx1 =
   rfckt.twowire with properties:

            Radius: 7.5000e-04
        Separation: 0.0016
               MuR: 1
          EpsilonR: 2.3000
       LossTangent: 0
         SigmaCond: Inf
        LineLength: 0.0100
         StubMode: 'NotAStub'
       Termination: 'NotApplicable'
             nPort: 2
    AnalyzedResult: []
              Name: 'Two-Wire Transmission Line'
```

Algorithms

- If you model the transmission line as a stubless line, the `analyze` method first calculates the ABCD-parameters at each frequency contained in the modeling frequencies vector. It then uses the `abcd2s` function to convert the ABCD-parameters to S-parameters.

  The `analyze` method calculates the ABCD-parameters using the physical length of the transmission line, d, and the complex propagation constant, k, using the following equations:

  $$A = \frac{e^{kd} + e^{-kd}}{2}$$

  $$B = \frac{Z_0 * \left(e^{kd} - e^{-kd}\right)}{2}$$

  $$C = \frac{e^{kd} - e^{-kd}}{2 * Z_0}$$

  $$D = \frac{e^{kd} + e^{-kd}}{2}$$

  Z_0 and k are vectors whose elements correspond to the elements of f, the vector of frequencies specified in the `analyze` input argument `freq`. Both can be expressed in terms of the resistance (R), inductance (L), conductance (G), and capacitance (C) per unit length (meters) as follows:

$$Z_0 = \sqrt{\frac{R + j2\pi fL}{G + j2\pi fC}}$$

$$k = k_r + jk_i = \sqrt{(R + j2\pi fL)(G + j2\pi FC)}$$

where

$$R = \frac{1}{\pi a \sigma_{cond} \delta_{cond}}$$

$$L = \frac{\mu}{\pi}\text{acosh}\left(\frac{D}{2a}\right)$$

$$G = \frac{\pi\omega\varepsilon''}{\text{acosh}\left(\frac{D}{2a}\right)}$$

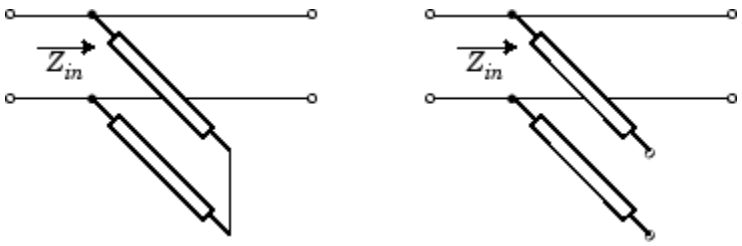
$$C = \frac{\pi\varepsilon}{\text{acosh}\left(\frac{D}{2a}\right)}$$

In these equations:

- w is the plate width.
- d is the plate separation.
- σ_{cond} is the conductivity in the conductor.
- μ is the permeability of the dielectric.
- ε is the permittivity of the dielectric.
- ε'' is the imaginary part of ε, $\varepsilon'' = \varepsilon_0\varepsilon_r\tan\delta$, where:
  - ε_0 is the permittivity of free space.
  - ε_r is the `EpsilonR` property value.
  - $\tan\delta$ is the `LossTangent` property value.
- δ_{cond} is the skin depth of the conductor, which the block calculates as $1/\sqrt{\pi f \mu \sigma_{cond}}$.
- f is a vector of modeling frequencies determined by the Outport block.

- If you model the transmission line as a shunt or series stub, the `analyze` method first calculates the ABCD-parameters at the specified frequencies. It then uses the `abcd2s` function to convert the ABCD-parameters to S-parameters.

  When you set the `StubMode` property to `'Shunt'`, the 2-port network consists of a stub transmission line that you can terminate with either a short circuit or an open circuit as shown in the following figure.



  Z_{in} is the input impedance of the shunt circuit. The ABCD-parameters for the shunt stub are calculated as:

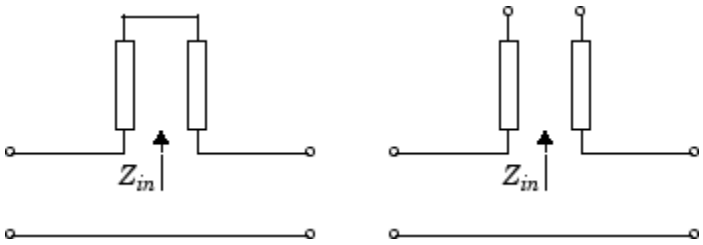
$A = 1$

$B = 0$

$C = 1/Z_{in}$

$D = 1$

When you set the `StubMode` property to `'Series'`, the 2-port network consists of a series transmission line that you can terminate with either a short circuit or an open circuit as shown in the following figure.



Z_{in} is the input impedance of the series circuit. The ABCD-parameters for the series stub are calculated as:

$A = 1$

$B = Z_{in}$

$C = 0$

$D = 1$

Version History

Introduced in R2009a

References

[1] Pozar, David M. *Microwave Engineering*, John Wiley & Sons, Inc., 2005.

See Also

`rfckt.amplifier` | `rfckt.cascade` | `rfckt.coaxial` | `rfckt.cpw` | `rfckt.datafile` | `rfckt.delay` | `rfckt.hybrid` | `rfckt.hybridg` | `rfckt.mixer` | `rfckt.microstrip` | `rfckt.passive` | `rfckt.parallel` | `rfckt.parallelplate` | `rfckt.rlcgline` | `rfckt.series` | `rfckt.seriesrlc` | `rfckt.shuntrlc`

rfckt.txline

General transmission line

Description

Use the `txline` class to represent transmission lines that are characterized by line loss, line length, stub type, and termination.

Creation

Syntax

```
h = rfckt.txline
h = rfckt.txline(Name,Value)
```

Description

`h = rfckt.txline`returns a transmission line object whose properties are set to their default values.

`h = rfckt.txline(Name,Value)` sets properties using one or more name-value pairs. For example, `rfckt.txline('Z0',75)` creates a transmission line object with characteristic impedance of 75 ohms. You can specify multiple name-value pairs. Enclose each property name in a quote. Properties not specified retain their default values.

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

Computed S-parameters, noise figure, OIP3, and group delay values, specified as `rfdata.data` object. This is a read-only property. For more information refer, “Algorithms” on page 1-110.

Data Types: `function_handle`

`Freq` — Frequency data
M-element vector

Frequency data for the RLCG values, specified as a *M*-element vector in Hz. The values must be positive and correspond to the order of loss and phase velocity values. By default, this property is empty.

Data Types: `double`

`IntpType` — Interpolation method used in `rfckt.rlcgline`
`'Linear'` (default) | `'Spline'` | `'Cubic'`

Interpolation method used in `rfckt.rlcgline`, specified as one of the following values:

Method	Description
`Linear`	Linear interpolation
`Spline`	Cubic spline interpolation
`Cubic`	Piecewise cubic Hermite interpolation

Data Types: `char`

`LineLength` — Physical length of transmission line
scalar

Physical length of transmission line, specified as a scalar in meters. The default value is `0.01`.

Data Types: `double`

`Loss` — Reduction in strength of signal
`0` (default) | nonnegative *M*-element vector

Reduction in strength of signal as it travels through the transmission line, specified as a nonnegative *M*-element vector in decibels per meter.

Data Types: `double`

`Name` — Object name
`'Transmission Line'` (default) | `1-by-N` character array

Object name, specified as a `1-by-N` character array. This is a read-only property.

Data Types: `char`

`nport` — Number of ports
positive integer

Number of ports, specified as a positive integer. This is a read-only property. The default value is 2.

Data Types: `double`

`PV` — Phase velocity
M-element vector

Phase velocity or propagation velocity of a uniform plane wave on the transmission line specified as a *M*-element vector in meters/sec. The phase velocity values correspond to the frequency values. The default value is 299792458.

Data Types: `double`

`StubMode` — Type of stub
`'NotaStub'` (default) | `'Series'` | `'Shunt'`

Type of stub, specified as one of the following values: `'NotaStub'`, `'Series'`, `'Shunt'`.

Data Types: `double`

`Termination` — Stub transmission line termination
`'NotApplicable'` (default) | `'Open'` | `'Short'`

Stub transmission line termination, specified as one of the following values: `'NotApplicable'`, `'Open'`, `'Short'`.

Data Types: `double`

`Z0` — Characteristic impedance
vector in ohms

Characteristic impedance, specified as a vector in ohms. The default value is `50` ohms.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
circle	Draw circles on Smith Chart
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
listparam	List valid parameters for specified circuit object
getz0	Calculate characteristic impedance of RFCKT transmission line object
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
polar	Plot specified object parameters on polar coordinates
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file

Examples

Frequency Domain Analysis of a Transmission Line

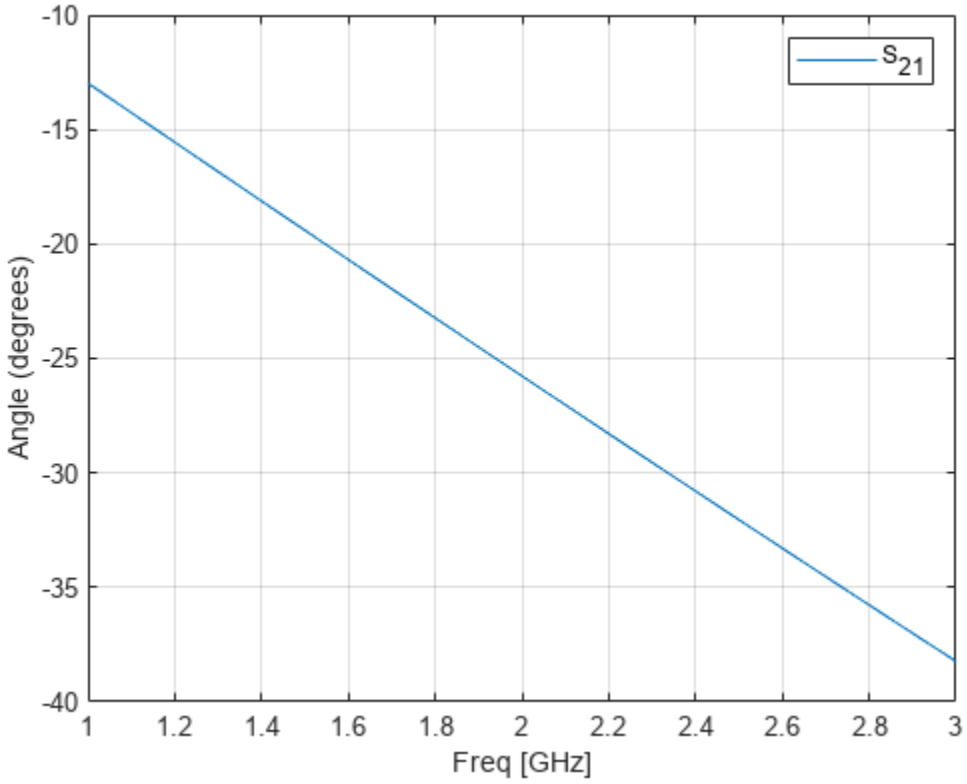
Transmission Line Properties

```
trl = rfckt.txline('Z0',75)
```

```
trl =
   rfckt.txline with properties:

        LineLength: 0.0100
          StubMode: 'NotAStub'
       Termination: 'NotApplicable'
              Freq: 1.0000e+09
                Z0: 75
                PV: 299792458
              Loss: 0
          IntpType: 'Linear'
             nPort: 2
    AnalyzedResult: []
              Name: 'Transmission Line'
```

Plot

```
f = [1e9:1.0e7:3e9];     % Simulation frequencies
analyze(trl,f);          % Do frequency domain analysis
figure
plot(trl,'s21','angle'); % Plot angle of S21
```



Algorithms

The `analyze` method treats the transmission line, which can be lossy or lossless, as a 2-port linear network. It computes the `AnalyzedResult` property of a stub or as a stubless line using the data stored in the `rfckt.txline` object properties as follows:

- If you model the transmission line as a stub less line, the `analyze` method first calculates the ABCD-parameters at each frequency contained in the modeling frequencies vector. It then uses the `abcd2s` function to convert the ABCD-parameters to S-parameters.

  The `analyze` method calculates the ABCD-parameters using the physical length of the transmission line, d, and the complex propagation constant, k, using the following equations:

  $$A = \frac{e^{kd} + e^{-kd}}{2}$$

  $$B = \frac{Z_0 * \left(e^{kd} - e^{-kd}\right)}{2}$$

  $$C = \frac{e^{kd} - e^{-kd}}{2 * Z_0}$$

  $$D = \frac{e^{kd} + e^{-kd}}{2}$$

  Z_0 is the specified characteristic impedance. k is a vector whose elements correspond to the elements of the input vector `freq`. The `analyze` method calculates k from the specified

properties as $k = \alpha_a + i\beta$, where α_a is the attenuation coefficient and β is the wave number. The attenuation coefficient α_a is related to the specified loss, α, by

$$\alpha_a = -\ln\left(10^{\alpha/20}\right)$$

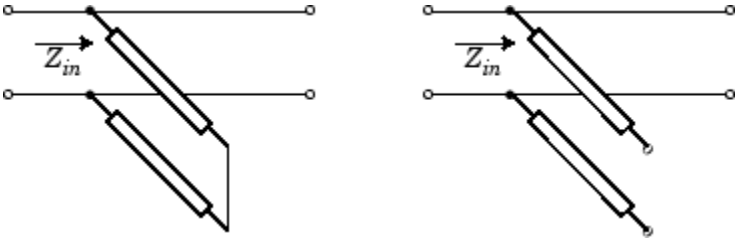
The wave number β is related to the specified phase velocity, V_p, by

$$\beta = \frac{2\pi f}{V_p},$$

where f is the frequency range specified in the `analyze` input argument `freq`. The phase velocity V_p is derived from the `rfckt.txline` object properties. It is also known as the *wave propagation velocity*.

- If you model the transmission line as a shunt or series stub, the `analyze` method first calculates the ABCD-parameters at the specified frequencies. It then uses the `abcd2s` function to convert the ABCD-parameters to S-parameters.

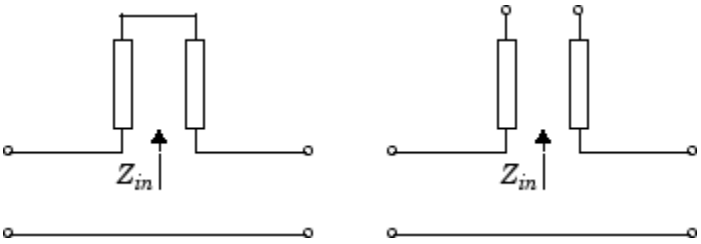
  When you set the `StubMode` property to `'Shunt'`, the 2-port network consists of a stub transmission line that you can terminate with either a short circuit or an open circuit as shown in the following figure.



  Z_{in} is the input impedance of the shunt circuit. The ABCD-parameters for the shunt stub are calculated as:

$$A = 1$$
$$B = 0$$
$$C = 1/Z_{in}$$
$$D = 1$$

  When you set the `StubMode` property to `'Series'`, the 2-port network consists of a series transmission line that you can terminate with either a short circuit or an open circuit as shown in the following figure.



  Z_{in} is the input impedance of the series circuit. The ABCD-parameters for the series stub are calculated as:

$$A = 1$$
$$B = Z_{in}$$
$$C = 0$$
$$D = 1$$

Version History

Introduced in R2009a

References

[1] Ludwig, R. and P. Bretchko, *RF Circuit Design: Theory and Applications*, Prentice-Hall, 2000.

See Also

`rfckt.amplifier` | `rfckt.cascade` | `rfckt.coaxial` | `rfckt.cpw` | `rfckt.datafile` | `rfckt.delay` | `rfckt.hybrid` | `rfckt.hybridg` | `rfckt.mixer` | `rfckt.microstrip` | `rfckt.passive` | `rfckt.parallel` | `rfckt.parallelplate` | `rfckt.rlcgline` | `rfckt.series` | `rfckt.seriesrlc` | `rfckt.shuntrlc` | `rfckt.twowire`

rfdata.data

Store result of circuit object analysis

Description

Use the `data` class to store S-parameters, noise figure in decibels, and frequency-dependent, third-order output (OIP3) intercept points.

There are three ways to create an `rfdata.data` object:

- You can construct it by specifying its properties from workspace data using the `rfdata.data` constructor.
- You can create it from file data using the `read` method.
- You can perform frequency domain analysis of a circuit object using the `analyze` method, and RF Toolbox software stores the results in an `rfdata.data` object.

Creation

Syntax

```
h = rfdata.data
h = rfdata.data('Property1',value1,'Property2',value2,...)
```

Description

`h = rfdata.data`returns a data object whose properties all have their default values.

`h = rfdata.data('Property1',value1,'Property2',value2,...)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote

Properties

`Freq` — Frequency data for S-parameters

M-element vector

Frequency data for the S-parameters in the `S-Parameters` property, specified as a M-element vector in hertz. The values must be positive and correspond to the order of the S-parameters. By default, this property is empty.

Data Types: `double`

`GroupDelayData` — Group delay data

M-element vector

Group delay data calculated at each frequency, specified as a M-element vector in seconds. By default, this property is empty.

Data Types: `double`

`IntpType` — Interpolation method used in `rfdata.data`
`1-by-N` character array | scalar string

Interpolation method used in `rfdata.data`, specified as a `1-by-N` character array of the following values:

Method	Description
`Linear` (default)	Linear interpolation
`Spline`	Cubic spline interpolation
`Cubic`	Piecewise cubic Hermite interpolation

Data Types: `char`

`NF` — Noise figure
scalar

Noise figure, specified as a scalar in dB. `'NF'` is the amount of noise relative to noise temperature of 290 degrees kelvin. The default value is zero indicating a noiseless system.

Data Types: `function_handle`

`OIP3` — Output third-order intercept
scalar

Output third-order intercept, specified as a scalar in watts. This property represents the hypothetical output signal level at which the third-order tones would reach the same amplitude level as the desired input tones. The default value is `Inf`.

Data Types: `double`

`S_Parameters` — S-parameter data
`2-by-2-by-M` array

S-parameter data, specified as a `2-by-2-by-M` array. `M` is the number of frequencies at which the network parameters are specified. By default, this property is empty.

Data Types: `double`

`Z0` — Reference impedance
scalar

Reference impedance, specified as a scalar in ohms. The default value is `50` ohms.

Data Types: `double`

`ZL` — Load impedance
scalar

Load impedance, specified as a scalar in ohms. The default value is `50` ohms.

Data Types: `double`

`ZS` — Source impedance
scalar

Source impedance, specified as a scalar in ohms. The default value is `50` ohms.

Data Types: `double`

`Name` — Object name
`1-by-N` character array | string

Object name, specified as a `1-by-N` character array or string. This is a read-only property.

Data Types: `char`

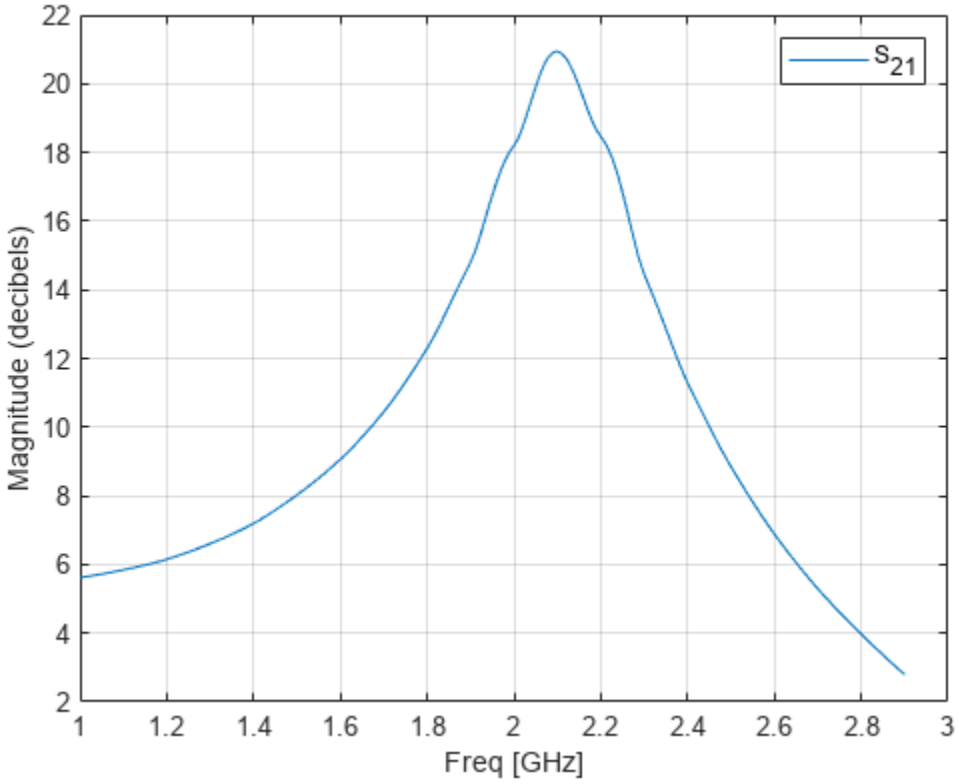
Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

RF Data Object From a .s2p Data File

```
file = 'default.s2p';
h = read(rfdata.data,file);  % Read file into data object.
figure
plot(h,'s21','db');   % Plot dB(S21) in XY plane.
```



Version History

Introduced in R2009a

See Also

`rfdata.mixerspur` | `rfdata.network` | `rfdata.nf` | `rfdata.noise` | `rfdata.power`

Topics

“Operations with RF Data Objects”

rfdata.ip3

Store frequency-dependent, third-order intercept points

Description

Use the `ip3` class to store third-order intercept point specifications for a circuit object.

Note If you set `NonLinearData` using `rfdata.ip3` or `rfdata.power`, then the property is converted from scalar OIP3 format to the format of `rfdata.ip3` or `rfdata.power`.

Creation

Syntax

```
h = rfdata.ip3
h = rfdata.ip3('Type',value1,'Freq',value2,'Data',value3)
```

Description

`h = rfdata.ip3` returns a data object for the frequency-dependent IP3, `h`, whose properties all have their default values.

`h = rfdata.ip3('Type',value1,'Freq',value2,'Data',value3)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote

Properties

`Data` — Third-order intercept values

M-element vector

Third-order intercept values, specified as a M-element vector in watts. The values correspond to the frequencies stored in the `'Freq'` property. The default value is `'Inf'`.

Data Types: `double`

`Freq` — Frequency data

M-element vector

Frequency data , specified as a M-element vector in hertz. The values must be positive and correspond to the order of the IP3 values. By default, this property is empty.

Data Types: `double`

Type — IP3 data type

`'OIP3'` (default) | `'IIP3'`

IP3 data type, specified as a `'OIP3'` or `'IIP3'`.

Data Types: `double`

`Name` — Object name
`1-by-N` character array | string

Object name, specified as a `1-by-N` character array or string. This is a read-only property.

Data Types: `char`

Examples

Store Third-Order Intercept Point Specifications

Create an object to store third-order intercept point specifications using `rfdata.ip3`.

```
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'
```

Version History

Introduced in R2009a

See Also

`rfdata.mixerspur` | `rfdata.network` | `rfdata.nf` | `rfdata.noise` | `rfdata.power`

rfdata.mixerspur

Store data from intermodulation table

Description

Use the `mixerspur` class to store mixer spur power specifications for a circuit object.

Creation

Syntax

```
h = rfdata.mixerspur
h = rfdata.mixerspur('Data',value1,'PLORef',value2,'PinRef','value3)
```

Description

`h = rfdata.mixerspur` returns a data object that defines an intermodulation table, `h`, whose properties all have their default values.

`h = rfdata.mixerspur('Data',value1,'PLORef',value2,'PinRef','value3)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote

Properties

`Data` — Mixer spur power values
matrix

Mixer spur power values, specified as a matrix in decibels. The values are such that the mixer spur power is less than the power at the fundamental output frequency. Values must be between `0` and `99`. By default, this property is empty.

Data Types: `double`

`PinRef` — Reference input power
scalar

Reference input power, specified as a scalar in decibels relative to 1 milliwatt. The default value is `0`.

Data Types: `double`

`PLORef` — Reference local oscillator power
scalar

Reference local oscillator power, specified as a scalar in decibels relative to 1 milliwatt. The default value is `0`.

Data Types: `double`

Name — Object name
1-by-N character array | string

Object name, specified as a 1-by-N character array or string. This property is a read-only.

Data Types: char

Examples

Store Mixer Spur Power Specifications

Create an object to store mixer spur power specifications using rfdata.mixerspur.

```
spurs = rfdata.mixerspur('Data',[2 5 3; 1 0 99; 10 99 99],...
          'PinRef',3,'PLORef',5)

spurs =
   rfdata.mixerspur with properties:

    PLORef: 5
    PinRef: 3
      Data: [3x3 double]
      Name: 'Intermodulation table'
```

Version History

Introduced in R2009a

See Also

rfdata.network | rfdata.nf | rfdata.noise | rfdata.power

rfdata.network

Store frequency-dependent network parameters

Description

Use the `network` class to store frequency-dependent S-, Y-, Z-, ABCD-, H-, G-, or T-parameters for a circuit object.

Creation

Syntax

```
h = rfdata.network
h = rfdata.network('Type',value1,'Freq',value2, Data',value3, 'Z0',value4)
```

Description

`h = rfdata.network` returns a data object for the frequency-dependent network parameters `h`, whose properties all have their default values.

`h = rfdata.network('Type',value1,'Freq',value2, Data',value3, 'Z0',value4)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote

Properties

`Data` — Network parameter data
`2-by-2-by-`*M* array

Network parameter data, specified as a `2-by-2-by-M` array. *M* is the number of frequencies. The values correspond to the frequencies stored in the `'Freq'` property. By default, this property is empty.

Data Types: `double`

`Freq` — Frequency data
M-element vector

Frequency data , specified as a M-element vector in hertz. The values must be positive and correspond to the order of the IP3 values. By default, this property is empty.

Data Types: `double`

Type — Type of network parameters
S-Parameters (default) | `'S'` | `'Y'` | `'Z'` | `'ABCD'` | `'H'` | `'G'` | `'T'`

Type of network parameters, specified as one of the following network parameters:

- 'S'
- 'Y'
- 'Z'
- 'ABCD'
- 'H'
- 'G'
- 'T'

Data Types: double

Z0 — Reference impedance
scalar

Reference impedance, specified as a scalar in ohms. This property is only available when the 'Type' is set to 'S'. The default value is 50 ohms.

Data Types: double

Name — Object name
1-by-N character array | string

Object name, specified as a 1-by-N character array or string. This is a read-only property.

Data Types: char

Examples

Store Frequency-Dependent RF Network Parameters.

Create an object to store frequency-dependent Y-parameters using rfdata.network.

```
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];

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

net =
   rfdata.network with properties:

    Type: 'Y_PARAMETERS'
    Freq: [3x1 double]
    Data: [2x2x3 double]
      Z0: 50.0000 + 0.0000i
    Name: 'Network parameters'
```

Version History

Introduced in R2009a

See Also

`rfdata.mixerspur` | `rfdata.nf` | `rfdata.noise` | `rfdata.power`

rfdata.nf

Store frequency-dependent noise figure data for amplifiers or mixers

Description

Use the `nf` class to store noise figure specifications for a circuit object.

Creation

Syntax

```
h = rfdata.nf
h = rfdata.nf('Freq',value1,'Data',value2)
```

Description

`h = rfdata.nf` returns a data object for the frequency-dependent noise figure, `h`, whose properties all have their default values.

`h = rfdata.nf('Freq',value1,'Data',value2)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote

Properties

`Data` — Noise figure values
M-element vector

Noise figure values, specified as a *M*-element vector in dB. The values correspond to the frequencies stored in the `'Freq'` property. The default value is `0`.

Data Types: `double`

`Freq` — Frequency data
M-element vector

Frequency data , specified as a M-element vector in hertz. The values must be positive and correspond to the order of the noise figure values. By default, this property is empty.

Data Types: `double`

`Name` — Object name
`1-by-N` character array | string

Object name, specified as a `1-by-N` character array or string. This is a read-only property.

Data Types: `char`

Examples

Store Noise Figure Specifications of RF Circuit Object.

Create an object to store noise figure specifications using `rfdata.nf`.

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

Version History

Introduced in R2009a

See Also

`rfdata.mixerspur` | `rfdata.network` | `rfdata.noise` | `rfdata.power`

rfdata.noise

Store frequency-dependent spot noise data for amplifiers or mixers

Description

Use the `noise` class to store spot noise specifications for a circuit object.

Creation

Syntax

```
h = rfdata.noise
h = rfdata.noise('Freq',value1,'FMIN',value2,'GAMMAOPT', value3,'RN',value4)
```

Description

`h = rfdata.noise` returns a data object for the frequency-dependent spot noise, `h`, whose properties all have their default values.

`h = rfdata.noise('Freq',value1,'FMIN',value2,'GAMMAOPT', value3,'RN',value4)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote

Properties

`FMIN` — Minimum noise figure data

M-element vector

Noise figure values, specified as a *M*-element vector in dB. . The values correspond to the frequencies stored in the `'Freq'` property. By default, the value is `1`.

Data Types: `double`

`Freq` — Frequency data

M-element vector

Frequency data , specified as a M-element vector in hertz. The values must be positive and correspond to the spot noise data in `'FMIN'`, `'GAMMAOPT'`, and `'RN'` properties. By default, this property is empty.

Data Types: `double`

`GAMMAOPT` — Optimum source reflection coefficients

M-element vector

Optimum source reflection coefficients , specified as a M-element vector. The values correspond to the frequencies stored in the `'Freq'` property. The default value is `1`.

Data Types: `double`

RN — Equivalent normalized noise resistance data
M-element vector

Equivalent normalized noise resistance data, specified as a M-element vector. The values correspond to the frequencies stored in the `'Freq'` property. The default value is `1`.

Data Types: `double`

Name — Object name
`1-by-N` character array | string

Object name, specified as a `1-by-N` character array or string. This is a read-only property.

Data Types: `char`

Examples

Store Spot Noise Specifications of RF Circuit Object.

Create an object to store spot noise specifications using `rfdata.noise`.

```
f = [2.08 2.10]*1.0e9;
fmin = [12.08 13.40];
gopt = [0.2484-1.2102j 1.0999-0.9295j];
rn = [0.26 0.45];
noisedata = rfdata.noise('Freq',f,'FMIN',fmin,...
                         'GAMMAOPT',gopt,'RN',rn)
```

```
noisedata =
   rfdata.noise with properties:

        Freq: [2x1 double]
        Fmin: [2x1 double]
    GammaOPT: [2x1 double]
          RN: [2x1 double]
        Name: 'Spot noise data'
```

Version History

Introduced in R2009a

See Also

`rfdata.mixerspur` | `rfdata.network` | `rfdata.nf` | `rfdata.power`

rfdata.power

Store output power and phase information for amplifiers or mixers

Description

Use the `power` class to store output power and phase specifications for a circuit object.

Creation

Syntax

```
h = rfdata.power
h = rfdata.power(`property1',value1,'property2',value2,...)
```

Description

`h = rfdata.power`returns a data object for the Pin/Pout power data, `h`, whose properties all have their default values.

`h = rfdata.power(`property1',value1,'property2',value2,...)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote

Properties

`Freq` — Frequency data
M-element vector

Frequency data , specified as a *M*-element vector in hertz. The values must be positive and correspond to the power data in `'Phase'`, `'Pin'`, and `'Pout'` properties. The order of frequencies is equal to the order of the phase and power values. By default, this property is empty.

Data Types: `double`

`Phase` — Phase shift data
M-element cell

Phase shift data, specified as a *M*-element cell in degrees. . The values correspond to the frequencies stored in the `'Freq'` property. The values within each element correspond to the input power values stored in the `'Pin'` property. The default value is `1`.

Data Types: `double`

`Pin` — Input power data
M-element cell in watts

Input power data , specified as a *M*-element vector cell in watts. The values correspond to the frequencies stored in the `'Freq'` property. For example,

$P_{in} = \{[A];[B];[C]\};$

where `A`, `B`, and `C` are column vectors that contain the first three frequencies stored in the `'Freq'` property.

The default value is `1`.

Data Types: `double`

`Pout` — Output power data
M-element vector

Output power data, specified as a *M*-element vector in watts. The values correspond to the frequencies stored in the `'Freq'` property. The values within each element correspond to the input power values stored in the `'Pin'` property. The default value is `1`.

Data Types: `double`

`Name` — Object name
`'Power data'` | `1-by-N` character array | string

Object name, specified as a `1-by-N` character array or string. This is a read-only property.

Data Types: `char`

Examples

Store Output Power and Phase Specifications of RF Circuit Object.

Create an object to store output power and phase specifications using `rfdata.power`.

```
f = [2.08 2.10]*1.0e9;
phase = {[27.1 35.3],[15.4 19.3 21.1]};
pin = {[0.001 0.002],[0.001 0.005 0.01]};
pout = {[0.0025 0.0031],[0.0025 0.0028 0.0028]};
powerdata = rfdata.power
```

```
powerdata =
   rfdata.power with properties:

     Freq: []
      Pin: {[1 10]}
     Pout: {[1 10]}
    Phase: {}
     Name: 'Power data'
```

```
powerdata.Freq = f;
powerdata.Phase = phase;
powerdata.Pin = pin;
powerdata.Pout = pout;
```

Version History

Introduced in R2009a

See Also

Topics

rfdata.ip3
rfdata.mixerspur
rfdata.network
rfdata.nf
rfdata.noise

rfmodel.rational

Perform rational fit using pole-residue representation of the component

Description

Use the `rational` class to represent RF components using a rational function object of the form:

$$F(s) = \left(\sum_{k=1}^{n} \frac{C_k}{s - A_k} + D \right) e^{-s\tau},\ s = j2\pi f$$

There are two ways to construct an rational function object:

- You can fit a rational function object 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.

Creation

Syntax

```
h = rfmodel.rational
h = rfmodel.rational('Property1',value1,'Property2',value2,...)
```

Description

`h = rfmodel.rational` returns a rational function object whose properties are set to their default values.

`h = rfmodel.rational('Property1',value1,'Property2',value2,...)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote

Properties

A — Poles of rational function object

complex vector

Poles of rational function object, specified as a complex vector in radians/second. The property length is shown in:

$$F(s) = \left(\sum_{k=1}^{n} \frac{C_k}{s - A_k} + D \right) e^{-s\tau},\ s = j2\pi f$$

where, *n* must be equal to the length of the vector you provide for `'C'`. *n* is the number of poles in the rational function object. By default, this property is empty.

Data Types: `double`

C — Residues of rational function object

complex vector

Residues of the rational function object, specified as a complex vector in radians/second. The property length is shown in

$$F(s) = \left(\sum_{k=1}^{n} \frac{C_k}{s - A_k} + D \right) e^{-s\tau},\ s = j2\pi f$$

as `n`, must be equal to the length of the vector you provide for `'A'`. `n` is the number of residues in the rational function object. By default, this property is empty.

Data Types: `double`

D — Frequency response offset

scalar

Frequency response offset, specified as a scalar. The default value is `0`.

Data Types: `double`

Delay — Frequency response time delay

scalar

Frequency response time delay, specified as a scalar. The default value is `0`.

Data Types: `double`

Name — Object name

`'Rational Function'` (default) | `1-by-N` character array

Object name, specified as a `1-by-N` character array. This is a read-only property.

Data Types: `char`

Object Functions

timeresp	Time response for rational objects
stepresp	Step-signal response for rational object and rationalfit function object
freqresp	Frequency response of rational object and rationalfit function object
impulse	Impulse response for rational function object
ispassive	Return true if rationalfit output is passive at all frequencies
makepassive	Enforce passivity of rationalfit output or a rational object
passivity	Plot passivity of N-by-N rationalfit function output
pwlresp	Calculate time response of piecewise linear input signal
generateSPICE	Generate SPICE file from rationalfit of S-parameters
writeva	Generate Verilog-A description of rational object
abcd	Construct state-space matrices from rational object
zpk	Compute zeros, poles, and gain of rational object

Examples

Fit a Rational Function to Data

Fit a rational function to data from an `rfdata.data` object.

```
S = sparameters('defaultbandpass.s2p');
freq = S.Frequencies;
data = rfparam(S,2,1);
fit = rationalfit(freq,data)

fit =
   rfmodel.rational with properties:

        A: [10x1 double]
        C: [10x1 double]
        D: 0
    Delay: 0
     Name: 'Rational Function'
```

Define, Evaluate and Visualize a Rational Function

Construct a rational function object, `rat`, with poles at -4 Mrad/s, -3 Grad/s,and -5 Grad/s and residues of 600 Mrad/s,2 Grad/s and 4 Grad/s.

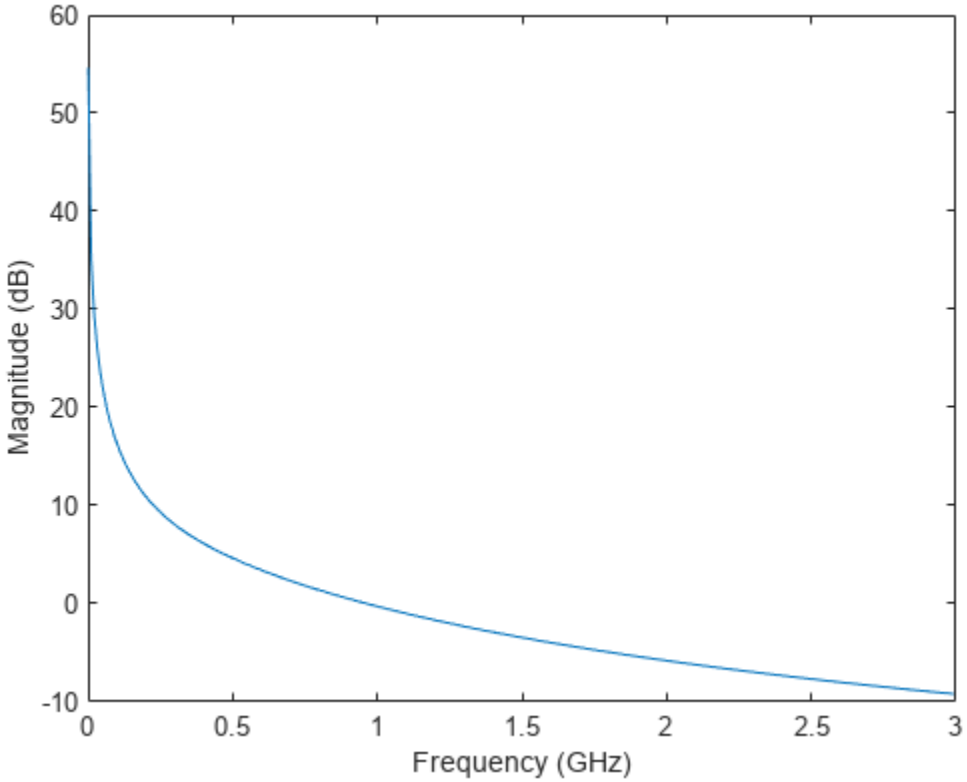
```
rat=rfmodel.rational('A',[-5e9,-3e9,-4e6],'C',[6e8,2e9,4e9]);
```

Perform frequency-domain analysis from 1.0 MHz to 3.0 GHz.

```
f = [1e6:1.0e7:3e9];
```

Plot the resulting frequency response in decibels on the X-Y plane.

```
[resp,freq]=freqresp(rat,f);
figure
plot(freq/1e9,20*log10(abs(resp)));
xlabel('Frequency (GHz)')
ylabel('Magnitude (dB)')
```



Generate SPICE File of 2-by-2 S-parameters

Read a file named `passive.s2p` and fit the 2-by-2 S-parameters. Generate a SPICE file of these S-parameters.

```
S = sparameters('passive.s2p');
fit = rationalfit(S);
generateSPICE(fit,'passive.ckt')
```

The circuit is saved in your current folder.

Version History

Introduced in R2009a

See Also

`rationalfit` | `ispassive` | `makepassive` | `passivity`

rfbudget

Create RF budget object and compute RF budget results for chain of 2-port elements

Description

Use the `rfbudget` object to create an RF budget object and compute the RF budget results for a chain of 2-port elements. In this RF chain, you can use a 2-port element such as `amplifier`, `nport`, or `modulator`. You can also open the `rfbudget` object in an **RF Budget Analyzer** app and then export the completed circuit to RF Blockset™ for circuit envelope analysis.

Creation

Syntax

```
rfobj = rfbudget
rfobj = rfbudget(elements,inputfreq,inputpwr,bandwidth)
rfobj = rfbudget( ___ ,autoupdate)
rfobj = rfbudget(Name=value)
```

Description

`rfobj = rfbudget` creates an `rfbudget` object, `rfobj`, with default empty property values.

`rfobj = rfbudget(elements,inputfreq,inputpwr,bandwidth)` sets Elements, InputFrequency, AvailableInputPower, and SignalBandwidth properties and computes the RF budget analysis. By default, if any of the input properties are changed, the object recomputes results.

`rfobj = rfbudget( ___ ,autoupdate)` sets the AutoUpdate property. You can use this syntax with any of the previous syntaxes.

`rfobj = rfbudget(Name=value)` sets "Properties" on page 1-135 using one or more name-value arguments. You can specify multiple name-value arguments.

Properties

`Elements` — RF budget elements

`[ ]` (default) | RF budget object | array of RF budget objects

RF budget elements, specified as an RF budget object or an array of RF budget objects. Use an array of RF budget objects when you perform RF budget analysis on an RF chain.

This table lists supported RF budget elements you can use to design an RF chain.

Element Type	RF Budget Elements
Linear Elements	`attenuator`
	`rfantenna`

Element Type	RF Budget Elements
	rffilter
	nport
	seriesRLC
	shuntRLC
	phaseshift
	txlineCoaxial
	txlineCPW
	txlineMicrostrip
	txlineParallelPlate
	txlineRLCGLine
	txlineStripline
	txlineTwoWire
	txlineEquationBased
	txlineDelayLossless
	txlineDelayLossy
Nonlinear Elements	amplifier
	modulator
	rfelement
	mixerIMT

Example: `a = amplifier; m = modulator; rfbudget(Elements=[a m])` calculates the RF budget analysis of the amplifier and modulator circuit.

`InputFrequency` — Input frequency of signal
`[]` (default) | scalar or column vector in Hz

Input frequency of the signal, specified as a scalar or column vector of size *M*-by-1 in Hz. *M* represents number of frequencies. If the input frequency is a vector, then the RF budget object analyzes each input frequency separately.

Example: `InputFrequency=2e9`

Data Types: `double`

`AvailableInputPower` — Power applied at input of cascade
`[]` (default) | scalar in dBm

Power applied at the input of the cascade, specified as a scalar in dBm.

Example: `AvailableInputPower=-30`

Data Types: `double`

`SignalBandwidth` — Signal bandwidth at input of cascade
`[]` (default) | scalar in Hz

Signal bandwidth at the input of the cascade, specified as a scalar in Hz.

Example: `SignalBandwidth=10`

Data Types: `double`

`AutoUpdate` — Automatically recompute RF budget analysis
`true` (default) | `false`

Automatically recompute the RF budget analysis by incorporating changes made to the existing circuit, specified as `true` or `false`.

Setting `AutoUpdate` to `false` turns off automatic budget recomputation as parameters change. To compute the budget result of an `rfbudget` object when you set the `AutoUpdate` property to `false`, use the `computeBudget` function.

Example: `AutoUpdate=false`

Data Types: `logical`

`Solver` — Computation Method
`Friis` (default) | `HarmonicBalance`

Computation method, specified as `Friis` or `HarmonicBalance`. The `Friis` solver is faster and the `HarmonicBalance` solver supports computation of second-order nonlinearities such as OIP2.

When you set the `Solver` type to `HarmonicBalance`, the tone and harmonic-dependent properties are displayed.

Note The `HarmonicBalance` solver does not support architectures where the input or output frequencies at any stage in the cascade are nonzero and less than SignalBandwidth.

Example: `Solver='Friis'`

Data Types: `string`

`HarmonicOrder` — Number of harmonics to use for all tones in HB analyses
`[]` (default) | positive integer

Number of harmonics to use for one-tone harmonic balance (HB) analysis, specified as a positive integer. For each two-tone analysis, `max(3,HarmonicOrder)` harmonics is used. Use the default value for automatic determination of harmonics.

Use this property to

- Accelerate the HB analysis by reducing the number of harmonics needed for a mildly nonlinear system.
- Ensure harmonic balance accuracy by increasing the number of harmonics used in a highly nonlinear system.

Dependencies

To enable this property, set Solver to `HarmonicBalance`.

Data Types: `double`

`OutputFrequency` — Output frequencies
scalar | vector | matrix

This property is read-only.

Output frequencies in Hz, returned as one of the following:

- Scalar when M and $N = 1$
- Vector when M or $N = 1$
- Matrix when M and $N > 1$

where M represents the number of frequencies in the input and N represents the number of stages in the cascade.

Data Types: `double`

`OutputPower` — Output power

scalar | vector | matrix

This property is read-only.

Output power in dBm, returned as one of the following:

- Scalar when M and $N = 1$
- Vector when M or $N = 1$
- Matrix when M and $N > 1$

where M represents the number of frequencies in the input and N represents the number of stages in the cascade.

Data Types: `double`

`TransducerGain` — Transducer power gains

scalar | vector | matrix

This property is read-only.

Transducer power gains in dB, returned as one of the following:

- Scalar when M and $N = 1$
- Vector when M or $N = 1$
- Matrix when M and $N > 1$

where M represents the number of frequencies in the input and N represents the number of stages in the cascade.

Data Types: `double`

`NF` — Noise figures

scalar | vector | matrix

This property is read-only.

Noise figures in dB, returned as one of the following:

- Scalar when M and $N = 1$
- Vector when M or $N = 1$

- Matrix when M and $N > 1$

where M represents the number of frequencies in the input and N represents the number of stages in the cascade.

Note If AvailableInputPower is very large, it can result in negative NF values during harmonic balance analysis [1].

Data Types: `double`

`IIP2` — Input-referred second-order intercept

scalar | vector | matrix

This property is read-only.

Input-referred second-order intercept (IIP2) in dBm, returned as one of the following:

- Scalar when M and $N = 1$
- Vector when M or $N = 1$
- Matrix when M and $N > 1$

where M represents the number of frequencies in the input and N represents the number of stages in the cascade.

Dependencies

To compute IIP2 values, set Solver to `HarmonicBalance`.

Data Types: `double`

`OIP2` — Output-referred second-order intercept

scalar | vector | matrix

This property is read-only.

Output-referred second-order intercept (OIP2) in dBm, returned as one of the following:

- Scalar when M and $N = 1$
- Vector when M or $N = 1$
- Matrix when M and $N > 1$

where M represents the number of frequencies in the input and N represents the number of stages in the cascade.

Dependencies

To compute OIP2 values, set Solver to `HarmonicBalance`.

Data Types: `double`

`IIP3` — Input-referred third-order intercept

scalar | vector | matrix

This property is read-only.

The Input-referred third-order intercept (IIP3) in dBm, returned as one of the following:

- Scalar when M and $N = 1$
- Vector when M or $N = 1$
- Matrix when M and $N > 1$

where M represents the number of frequencies in the input and N represents the number of stages in the cascade.

Data Types: `double`

`OIP3` — Output-referred third-order intercept
scalar | vector | matrix

This property is read-only.

The Output-referred third-order intercept (OIP3) in dBm, returned as one of the following:

- Scalar when M and $N = 1$
- Vector when M or $N = 1$
- Matrix when M and $N > 1$

where M represents the number of frequencies in the input and N represents the number of stages in the cascade.

Data Types: `double`

`SNR` — Signal-to-noise ratio
scalar | vector | matrix

This property is read-only.

Signal-to-noise ratio (SNR) in dB, returned as one of the following:

- Scalar when M and $N = 1$
- Vector when M or $N = 1$
- Matrix when M and $N > 1$

where M represents the number of frequencies in the input and N represents the number of stages in the cascade.

Data Types: `double`

`WaitBar` — Display progress bar
true (default) | false

Display a progress bar with a cancel button during harmonic balance analysis, specified as `true` or `false`.

Data Types: `logical`

Object Functions

show	Display RF budget object in RF Budget Analyzer app
computeBudget	Compute results of RF budget object

exportScript	Export MATLAB code that generates RF budget object
exportRFBlockset	Create RF Blockset model from RF budget object
exportTestbench	Create measurement testbench from RF budget object
rfplot	Plot cumulative RF budget result versus cascade input frequency
smithplot	Plot measurement data on Smith chart
polar	Plot specified object parameters on polar coordinates

Examples

Default RF Budget

Open a default RF budget object.

```
obj = rfbudget
```

```
obj =
  rfbudget with properties:

               Elements: []
         InputFrequency: [] Hz
    AvailableInputPower: [] dBm
        SignalBandwidth: [] Hz
                 Solver: Friis
             AutoUpdate: true
```

RF Budget Analysis of Series of RF Elements

Create an amplifier with a gain of 4 dB.

```
a = amplifier(Gain=4);
```

Create a modulator with an OIP3 of 13 dBm.

```
m = modulator(OIP3=13);
```

Create an N-port element using `passive.s2p`.

```
n = nport('passive.s2p');
```

Create an RF element with a gain of 10 dB.

```
r = rfelement(Gain=10);
```

Calculate the RF budget of a series of RF elements at an input frequency of 2.1 GHz, an available input power of –30 dBm, and a bandwidth of 10 MHz.

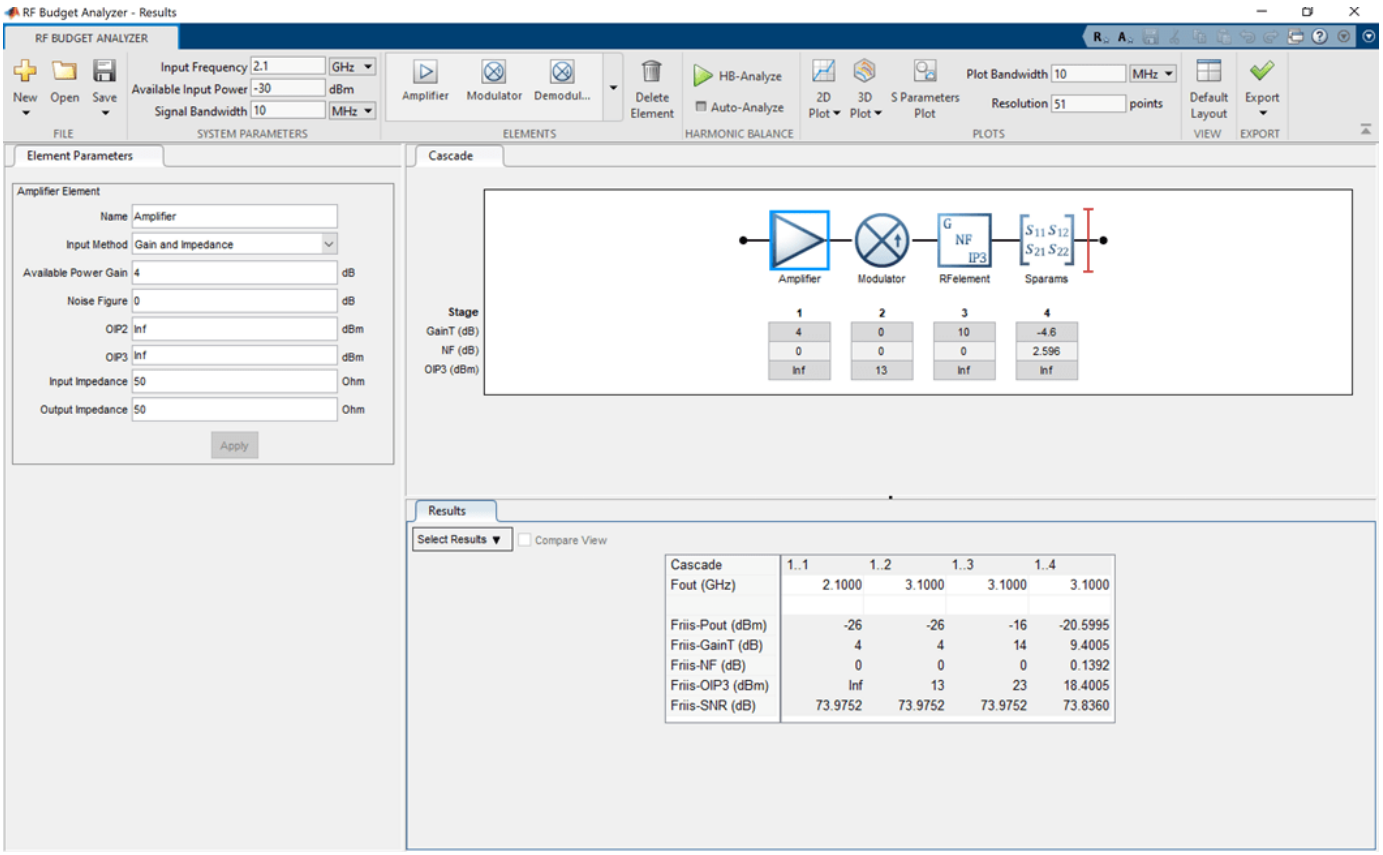
```
b = rfbudget([a m r n],2.1e9,-30,10e6)
```

```
b =
  rfbudget with properties:

               Elements: [1x4 rf.internal.rfbudget.Element]
         InputFrequency: 2.1 GHz
```

```
    AvailableInputPower: -30 dBm
         SignalBandwidth:  10 MHz
                  Solver: Friis
              AutoUpdate: true

Analysis Results
     OutputFrequency: (GHz) [  2.1    3.1    3.1     3.1]
         OutputPower: (dBm) [  -26    -26    -16   -20.6]
      TransducerGain: (dB)  [    4      4     14     9.4]
                  NF: (dB)  [    0      0      0  0.1392]
                IIP2: (dBm) []
                OIP2: (dBm) []
                IIP3: (dBm) [  Inf      9      9       9]
                OIP3: (dBm) [  Inf     13     23    18.4]
                 SNR: (dB)  [73.98  73.98  73.98   73.84]
```

Type the `show` command at the command window to display the analysis in the **RF Budget Analyzer** app.

```
show(b)
```



Plot Cumulative Output Power and Gain of RF System

Create an RF system.

Create an RF bandpass filter using the Touchstone® file `RFBudget_RF`.

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

Create an amplifier with a gain of 11.53 dB, a noise figure (NF) of 1.53 dB, and an output third-order intercept (OIP3) of 35 dBm.

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

Create a demodulator with a gain of –6 dB, a NF of 4 dB, and an OIP3 of 50 dBm.

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

Create an IF bandpass filter using the Touchstone file `RFBudget_IF`.

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

Create an amplifier with a gain of 30 dB, a NF of 8 dB, and an OIP3 of 37 dBm.

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

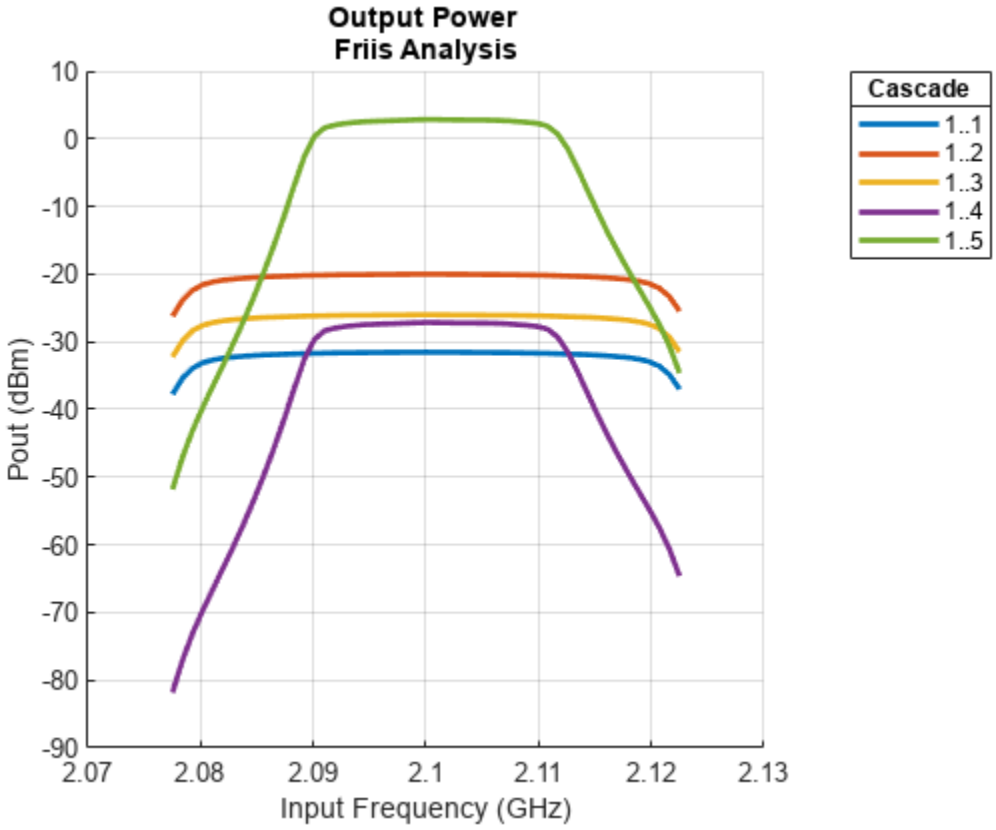
Calculate the RF budget of the system using an input frequency of 2.1 GHz, an input power of –30 dBm, and a bandwidth of 45 MHz.

```
b = rfbudget([f1 a1 d f2 a2],2.1e9,-30,45e6)
```

```
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]
```

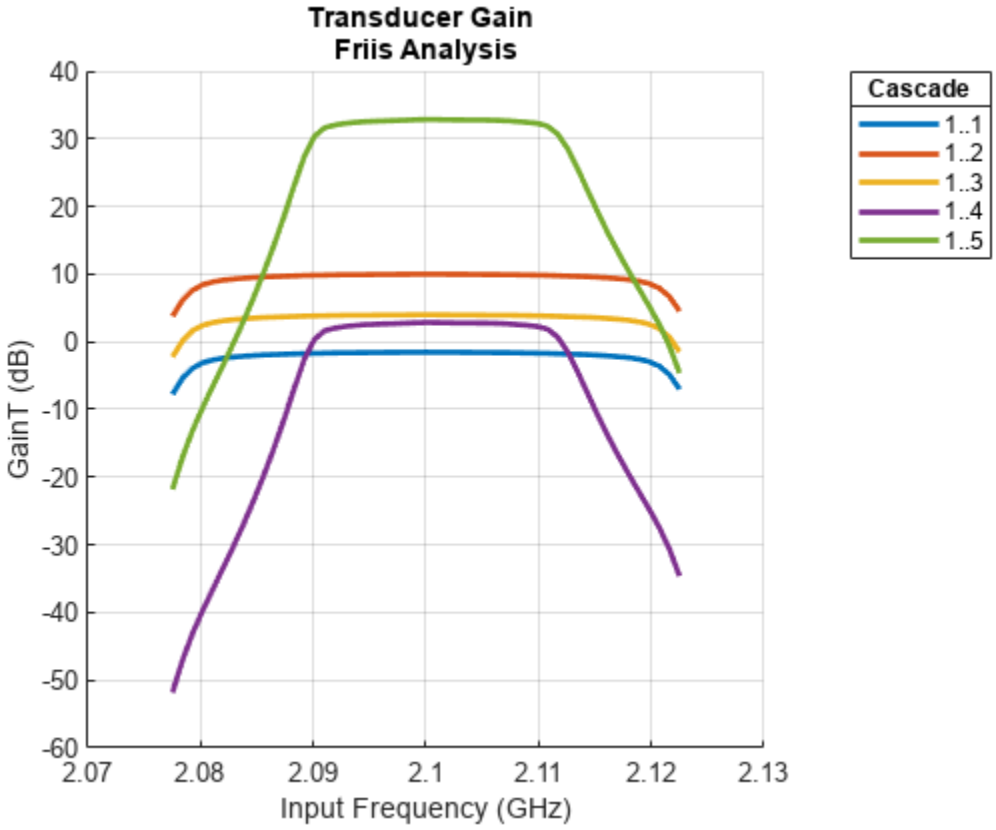
Plot the available output power.

```
rfplot(b,'Pout')
view(90,0)
```



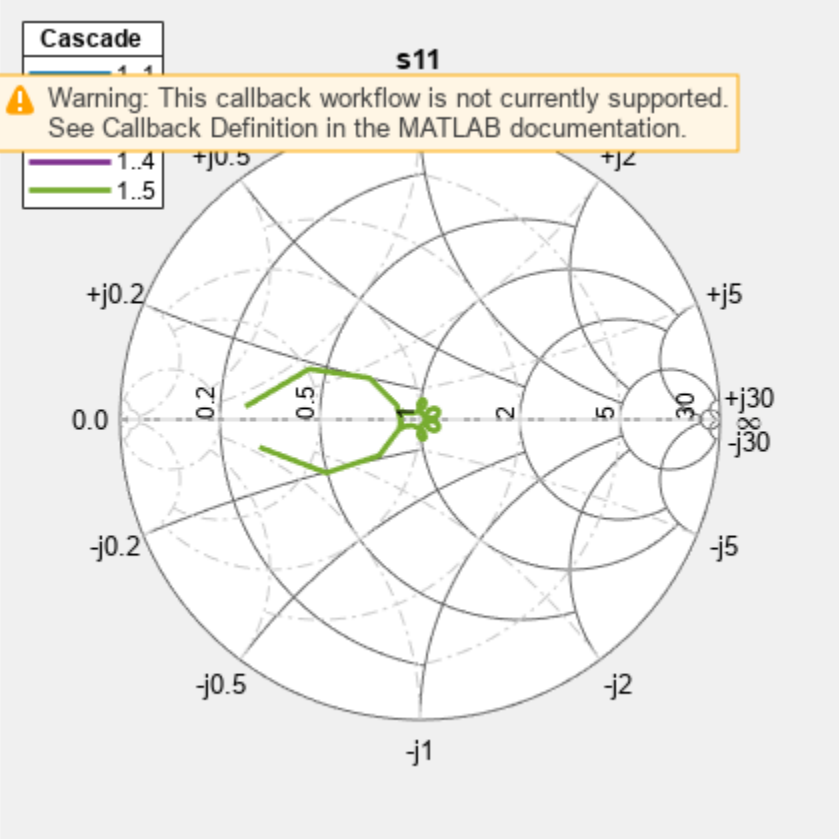
Plot the transducer gain.

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

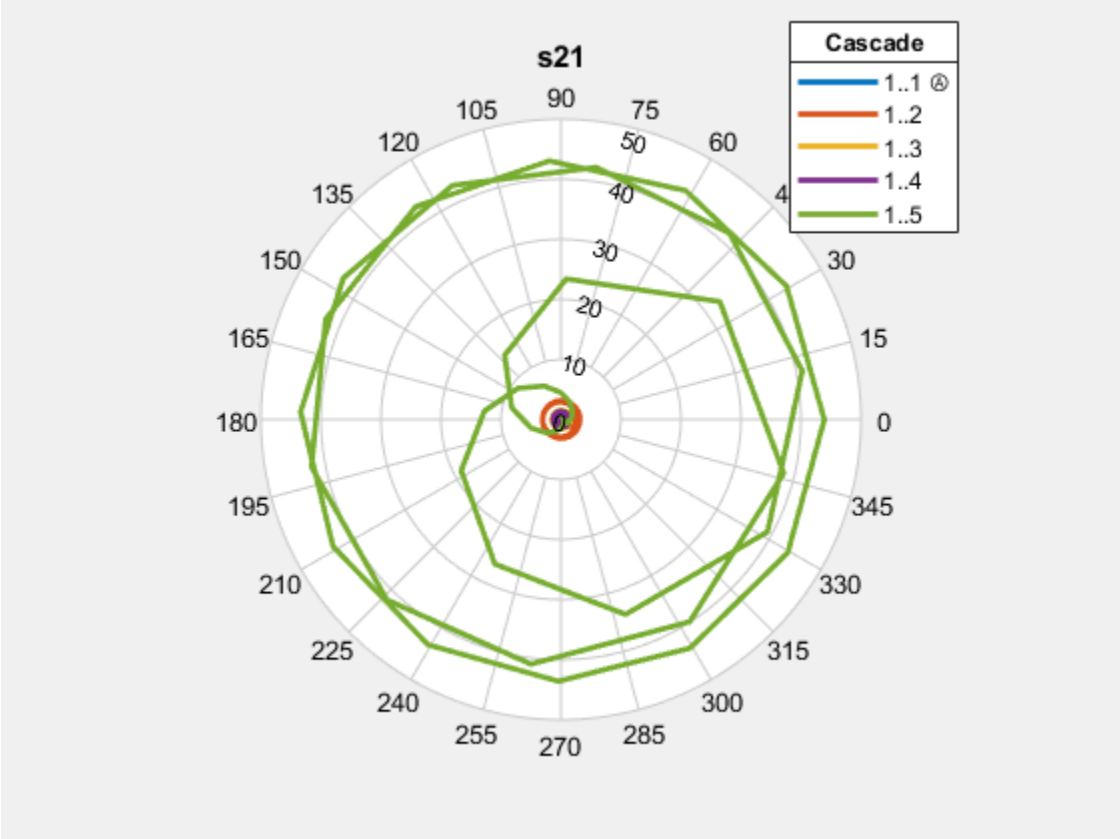


Plot S-parameters of an RF system on a Smith Chart and a Polar plot.

```
s = smithplot(b,1,1,'GridType','ZY');
```


Cascade
1..4
1..5
s11
Warning: This callback workflow is not currently supported.
See Callback Definition in the MATLAB documentation.
+j0.5
+j2
+j0.2
+j5
0.0
+j30
-j30
-j0.2
-j5
-j0.5
-j2
-j1
0.2
0.5
1
2
5
30
∞

```
p = polar(b,2,1);
```



Harmonic Balance Solver for Nonlinear RF Budget Analysis

Create two modulators with output-referred second-order intercept set to 20 and available power gain set to 3.

```
m = modulator(Gain=3,OIP2=20,ImageReject=false,ChannelSelect=false);
m2 = modulator(Gain=3,OIP2=20,ImageReject=false,ChannelSelect=false);
```

Create an RF budget object specifying the input frequency of the signal, power applied at cascade, and signal bandwidth. Select `HarmonicBalance` as solver method to compute nonlinear effects such as IIP2 and OIP2.

```
b = rfbudget([m m2],2.1e9,-30,100e6,Solver='HarmonicBalance')
```

```
b =
  rfbudget with properties:

              Elements: [1x2 modulator]
        InputFrequency: 2.1 GHz
    AvailableInputPower: -30 dBm
        SignalBandwidth: 100 MHz
                Solver: HarmonicBalance
               WaitBar: true
            AutoUpdate: true
```

```
   Analysis Results
        OutputFrequency: (GHz) [  3.1    4.1]
            OutputPower: (dBm) [  -27    -24]
         TransducerGain: (dB)  [    3      6]
                     NF: (dB)  [ 3.01  7.783]
                   IIP2: (dBm) [   17  4.457]
                   OIP2: (dBm) [   20  10.46]
                   IIP3: (dBm) [  Inf    Inf]
                   OIP3: (dBm) [  Inf    Inf]
                    SNR: (dB)  [60.96  56.19]
```

Number of Harmonics in HB Analysis

Create an amplifier with a gain of 10 dB.

```
a = amplifier(Gain=10);
```

Create a modulator with an OIP3 of 13 dBm.

```
m = modulator(OIP3=13);
```

Create an N-port circuit element using `passive.s2p`.

```
n = nport('passive.s2p');
```

Calculate the RF budget of a series of RF elements at an input frequency of 2.1 GHz, an available input power of –30 dBm, and a bandwidth of 10 MHz using HB analysis. Set the number of harmonics that the `rfbudget` object should use for all the tones in HB analyses.

```
b = rfbudget([a m n],2.1e9,-30,10e6,...
    Solver="HarmonicBalance",HarmonicOrder=3)
```

```
b =
  rfbudget with properties:

                Elements: [1x3 rf.internal.rfbudget.Element]
          InputFrequency: 2.1 GHz
     AvailableInputPower: -30 dBm
          SignalBandwidth:  10 MHz
                  Solver: HarmonicBalance
           HarmonicOrder: 3
                 WaitBar: true
              AutoUpdate: true

   Analysis Results
        OutputFrequency: (GHz) [       2.1        3.1     3.1]
            OutputPower: (dBm) [       -20        -20   -24.6]
         TransducerGain: (dB)  [        10      9.996   5.396]
                     NF: (dB)  [-2.842e-14  -0.004353  0.3376]
                   IIP2: (dBm) [       Inf        Inf     Inf]
                   OIP2: (dBm) [       Inf        Inf     Inf]
                   IIP3: (dBm) [       Inf      2.993   2.995]
                   OIP3: (dBm) [       Inf      12.98   8.382]
                    SNR: (dB)  [     73.98      73.98   73.64]
```

Plot Phase and Group Delay of RF System

Create an amplifier with a gain of 4 dB.

```
a = amplifier(Gain=4);
```

Create a modulator with an OIP3 of 13 dBm.

```
m = modulator(OIP3=13);
```

Create an N-port element using `passive.s2p`.

```
n = nport('passive.s2p');
```

Create an RF element with a gain of 10 dB.

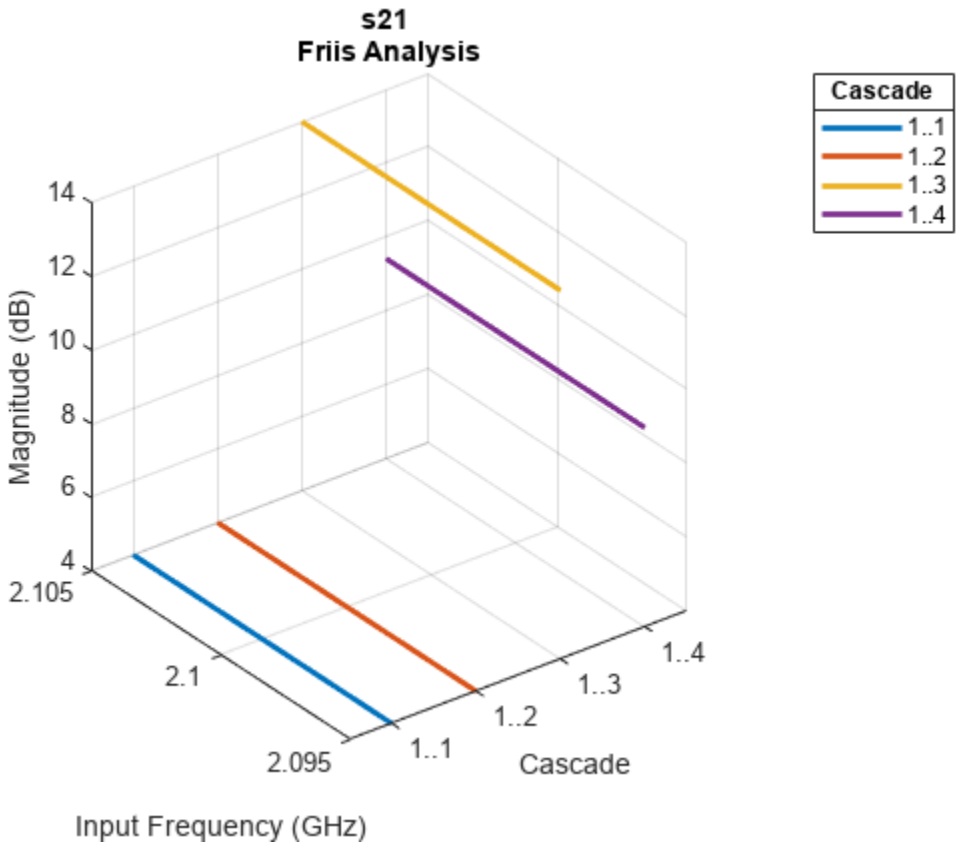
```
r = rfelement(Gain=10);
```

Calculate the RF budget of a series of RF elements at an input frequency of 2.1 GHz, an available input power of –30 dB, and a bandwidth of 10 MHz.

```
b = rfbudget([a m r n],2.1e9,-30,10e6);
```

Show the analysis in the RF plot.

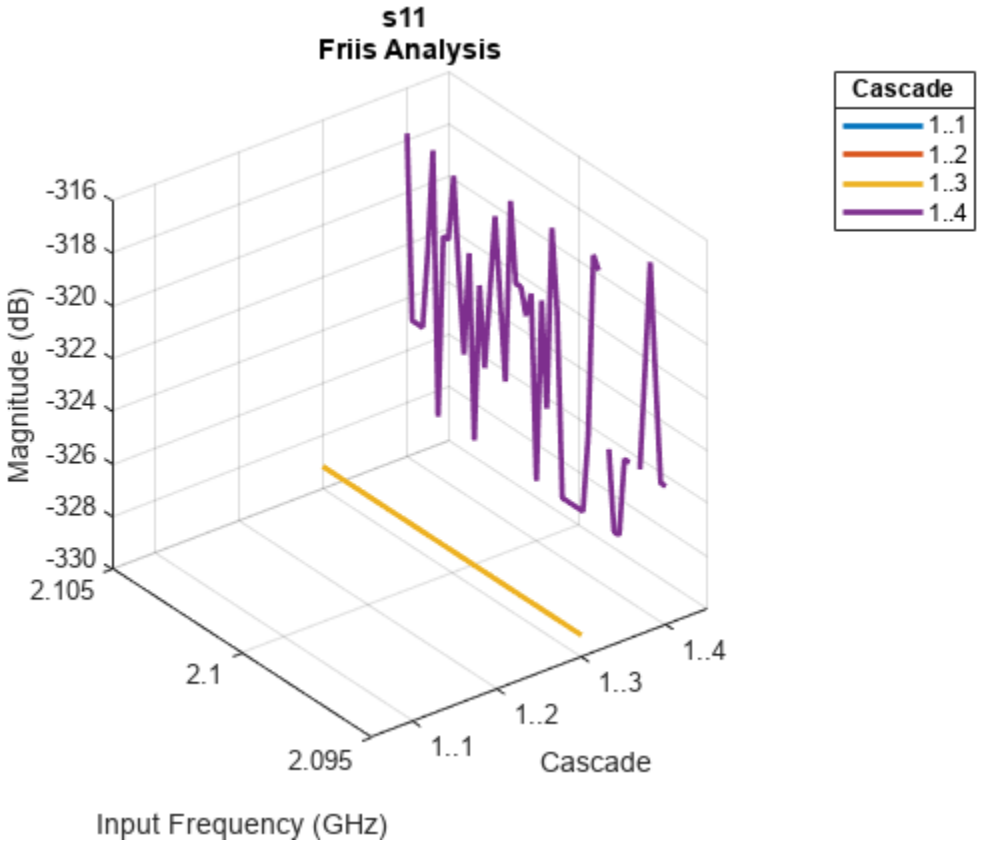
```
rfplot(b)
```



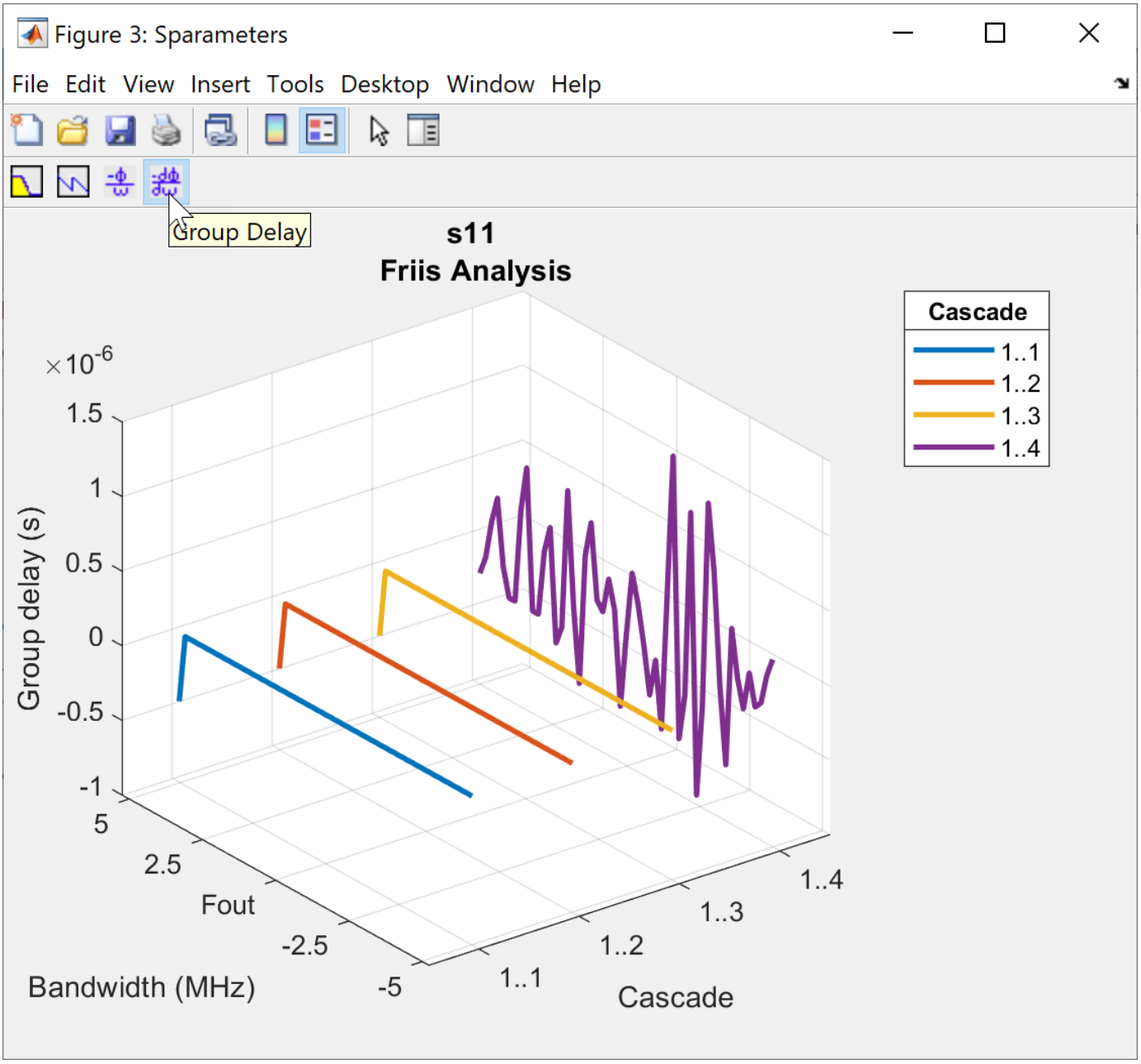
Group Delay

To plot the group delay, first plot the S11 data for the RF System.

```
rfplot(b,1,1)
```

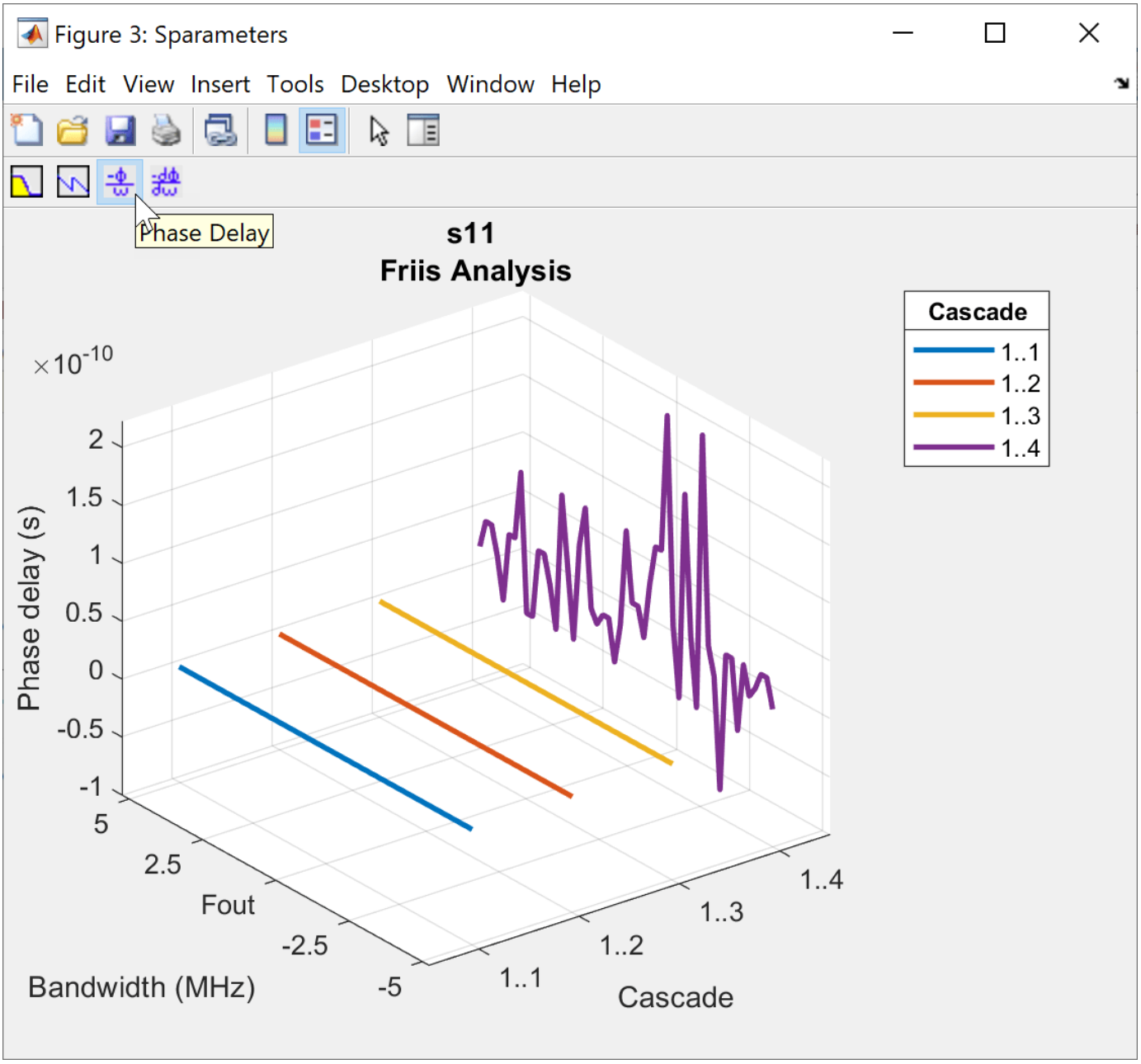


Use the `Group Delay` option on the plot graph to plot the group delay of the RF system.



Phase Delay

Use the `Phase Delay` option on the plot graph to plot the phase delay of the RF System.



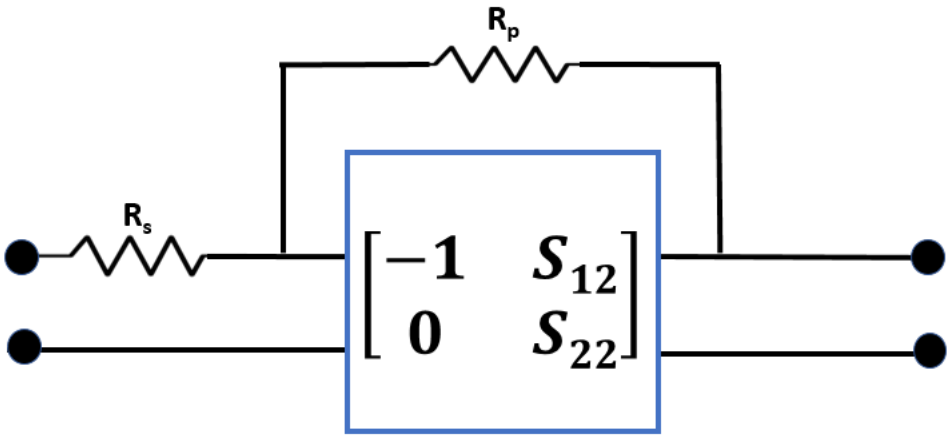
Tips

- The Touchstone file in the `nport` object must be passive at all specified frequencies. To make N-port S-parameters passive, use the `makepassive` function.

Algorithms

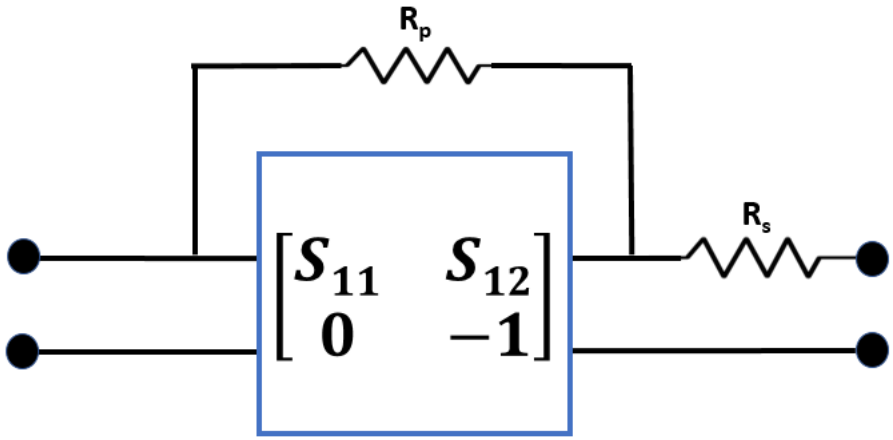
ABCD parameters are used in the computation of S-parameters of the cascade for Friis Solver. When S21 = 0, conversion to ABCD results in NaNs. For such cases, modifications to the S-parameters are made as follows:

S_{21} = 0, S_{11} = -1, and S_{22} ≠ -1



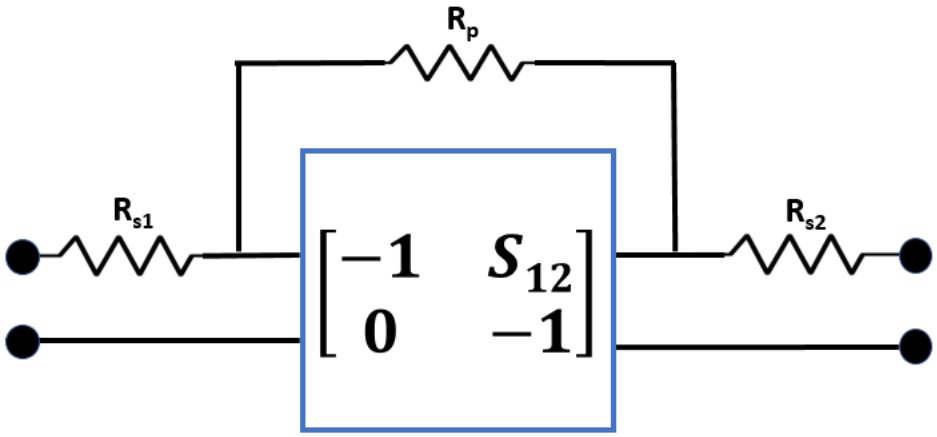
- Connected large resistance ($R_p = 10^{12}$ ohm) in parallel with the network.
- Connected small resistance ($R_s = 10^{-12}$ ohm) in series to the beginning of the network.

S_{21} = 0, S_{22} = -1, and S_{11} ≠ -1



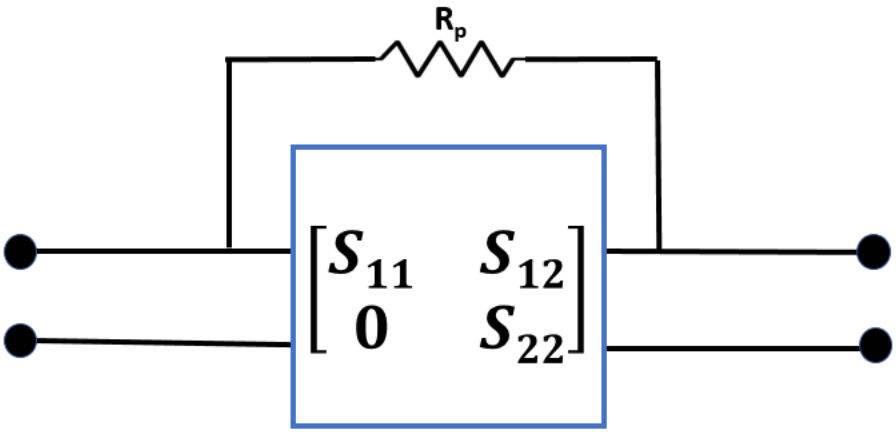
- Connected large resistance ($R_p = 10^{12}$ ohm) in parallel with the network.
- Connected small resistance ($R_s = 10^{-12}$ ohm) in series after the network.

S_{21} = 0, S_{22} = -1, and S_{11} = -1



- Connected large resistance (R_p = 10^{12} ohm) in parallel with the network.
- Connected small resistance (R_s = 10^{-12} ohm) in series to the beginning of the network.
- Connected small resistance (R_s = 10^{-12} ohm) in series after the network.

S_{21} = 0



- Connected large resistance (R_p = 10^{12} ohm) in parallel with the network.

Version History

Introduced in R2017a

Specify number of harmonics to use for all tones in HB analysis

Specify the number of harmonics for all tones in HB analyses by setting the HarmonicOrder property in the `rfbudget` object.

References

[1] Roychowdhury, J., D. Long, and P. Feldmann. “Cyclostationary Noise Analysis of Large RF Circuits with Multitone Excitations.” *IEEE Journal of Solid-State Circuits* 33, no. 3 (March 1998): 324–36. https://doi.org/10.1109/4.661198.

See Also

`amplifier` | `nport` | `modulator`

Topics

“Visualizing RF Budget Analysis over Bandwidth”

amplifier

Create two-port amplifier element

Description

Use the `amplifier` object to create a two-port amplifier element or to analyze a commercial off-the-shelf (COTS) amplifier. You can also use the `amplifier` object to model an amplifier in an RF system created using an `rfbudget` object or the **RF Budget Analyzer** app and, then export this element to RF Blockset or to `rfsystem` System object for circuit envelope analysis.

Creation

Syntax

```
amp = amplifier
amp = amplifier(Name=Value)
```

Description

`amp = amplifier` creates an amplifier object with default property values.

`amp = amplifier(Name=Value)` sets amplifier object properties using one or more name-value arguments. You can specify multiple name-value arguments.

Properties

`Name` — Name of amplifier
`'Amplifier'` (default) | character vector | string scalar

Name of amplifier, specified as a character vector. All names must be valid MATLAB variable names.

Example: `Name='amp'`

`UseNetworkData` — S-parameter characterization in amplifier
`false` or `0` (default) | `true` or `1`

S-parameter characterization in amplifier, specified as a logical `true` or `1` or `false` or `0`. When you set this property to `true`, the `amplifier` object uses the `NetworkData` and `NoiseData` properties to compute gain, matching, and noise. When you set this property to `false`, the `amplifier` object use the `Gain`, `Zin`, `Zout`, and `NF` properties to compute gain, matching, and noise.

Example: `UseSparameters=1`

`Gain` — Available power gain
`0` (default) | real finite scalar

Available power gain, specified as a real finite scalar in dB.

Example: `Gain=10`

`NF` — Noise figure
`0` (default) | real finite nonnegative scalar

Noise figure, specified as a real finite nonnegative scalar dB.

Example: `NF=-10`

`OIP2` — Second-order output-referred intercept point
`Inf` (default) | real scalar

Second-order output-referred intercept point, specified as a real scalar in dBm.

Example: `OIP2=8`

`OIP3` — Third-order output-referred intercept point
`Inf` (default) | real scalar

Third-order output-referred intercept point, specified as a real scalar in dBm.

Example: `OIP3=10`

`Zin` — Input impedance
`50` (default) | positive real part finite scalar

Input impedance, specified as a positive real part finite scalar in ohms. You can also use a complex value with a positive real part.

Example: `Zin=40`

`Zout` — Output impedance
`50` (default) | positive real part finite scalar

Output impedance, specified as a scalar in ohms. You can also use a complex value with a positive real part.

Example: `Zout=40`

`FileName` — Name of two-port Touchstone file
`''` (default) | string scalar | character vector

Name of the two-port Touchstone file from which to extract the `NetworkData` and the `NoiseData` properties, specified as a string scalar or character vector.

Example: `FileName='default.s2p'`

`NetworkData` — Network Data
network parameter object

Network data of the amplifier, specified as a two-port network parameter object. The network parameter objects are of the type:

- `sparameters`
- `yparameters`
- `zparameters`
- `gparameters`
- `hparameters`

- `abcdparameters`
- `tparameters`

Network parameter object defines the frequency-dependent gain and impedance matching for the amplifier, typically, a `sparameters` object from a two-port Touchstone file. To specify a frequency-independent network data, set the `NetworkData` property to `[]`. This resets network data to a frequency-independent two-port network object defined by the `Gain`, `Zin`, and `Zout` properties.

Example: `NetworkData=nd`

`NoiseData` — Noise Data
`noiseParameter` object

Noise data of the amplifier, specified as a `noiseParameters` object. The `noiseParameter` object contains a frequency-dependent noise figure loaded from a two-port Touchstone file or built at the MATLAB® command line. To specify a frequency-independent noise figure, set the `NoiseData` to `[]`.

Example: `NoiseData=np`

`NumPorts` — Number of ports
2 (default) | scalar integer

This property is read-only.

Number of ports, returned as a scalar integer.

`Terminals` — Names of port terminals
`{'p1+' 'p2+' 'p1-' 'p2-'}` (default) | cell vector

This property is read-only.

Names of port terminals, returned as a cell vector.

Object Functions

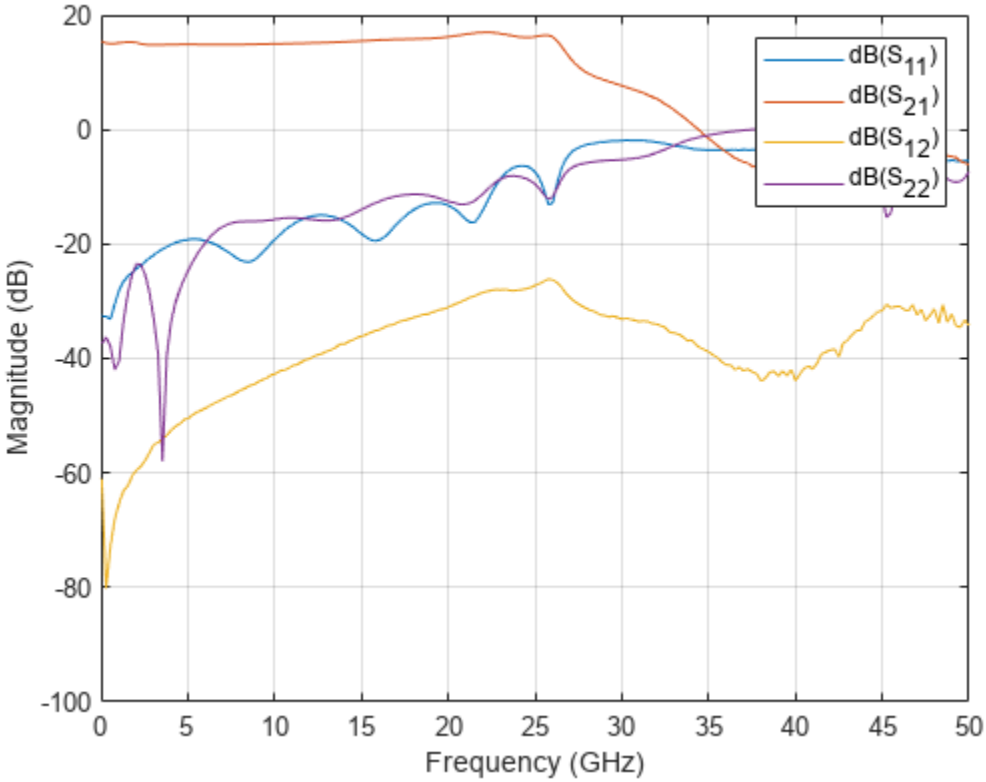
clone Create copy of existing circuit element or circuit object

Examples

Analyze COTS Amplifier

Use the RF Toolbox `amplifier` object to model a Qorvo CMD240 COTS amplifier. First, use the `sparameter` object to capture the S-parameter data from the CMD240 data file (Copyright (c) Qorvo, Inc., reproduced with permission).

```
S = sparameters('cmd240-sparameters.s2p');
rfplot(S)
```



Then use the `noiseParameters` object to build the noise data.

```
NF = [4 2.9 2.2 1.8 2.2 2 2.1 2.3 2.4 3.1 3.7];
freqs = (2:2:22)*1e9;
nd = noiseParameters(NF,freqs,50)
```

```
nd =
  noiseParameters with properties:

    Frequencies: [11x1 double]
           Fmin: [11x1 double]
       GammaOpt: [11x1 double]
             Rn: [11x1 double]
```

Create an amplifier using the CMD240 data file and add the noise data to the amplifier.

```
a1 = amplifier('FileName','cmd240-sparameters.s2p','OIP3',27.8);
a1.NoiseData = nd % clears FileName since there is no noise in the file.
```

```
a1 =
  amplifier: Amplifier element

              Name: 'Amplifier'
    UseNetworkData: 1
          FileName: ''
       NetworkData: [1x1 sparameters]
         NoiseData: [1x1 noiseParameters]
```

```
              OIP2: Inf
              OIP3: 27.8000
          NumPorts: 2
         Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Alternatively, you can use the `nport` object to hold both the S-parameter and the noise data and then use the `rfwrite` function to create a Touchstone file.

```
n = nport(NetworkData=S,NoiseData=nd);
rfwrite(n,'CMD240withNF.s2p','Format','RI')
a = amplifier('FileName','CMD240withNF.s2p','OIP3',27.8);
```

Use the `rfbudget` object to compare harmonic balance analysis with Friis analysis.

```
b = rfbudget(a,10e9,-30,1e3,'Solver','HarmonicBalance');
b.Friis
```

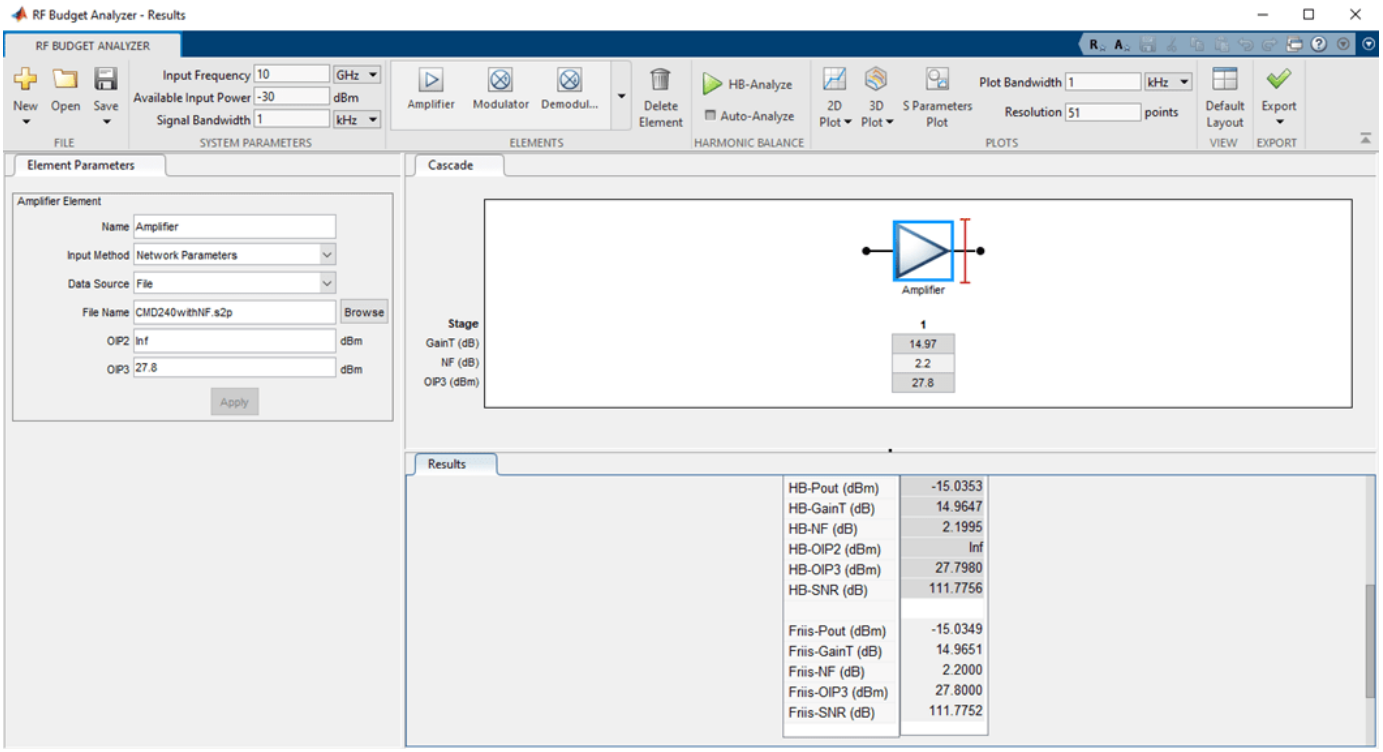
```
ans = struct with fields:
       OutputPower: -15.0349
    TransducerGain: 14.9651
                NF: 2.2000
              IIP2: []
              OIP2: []
              IIP3: 12.7828
              OIP3: 27.8000
               SNR: 111.7752
```

```
b.HarmonicBalance
```

```
ans = struct with fields:
       OutputPower: -15.0353
    TransducerGain: 14.9647
                NF: 2.1995
              IIP2: Inf
              OIP2: Inf
              IIP3: 12.7821
              OIP3: 27.7980
               SNR: 111.7756
```

Gain is about 15 dB and noise figure is about 2.2 dB. This matches the product data sheet. For more information, see CMD240 product data sheet. You can verify this in **RF Budget Analyzer** app.

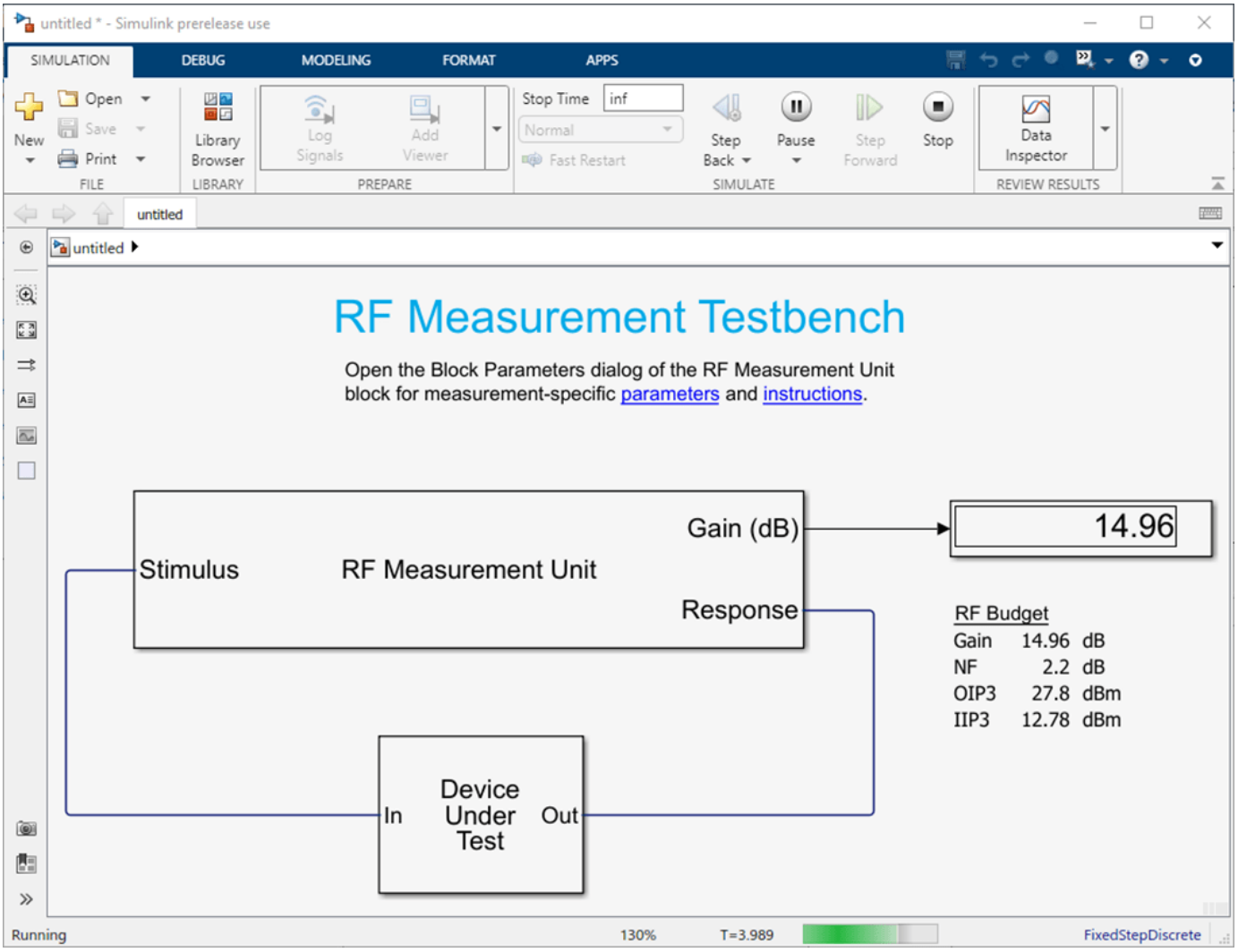
```
show(b)
```



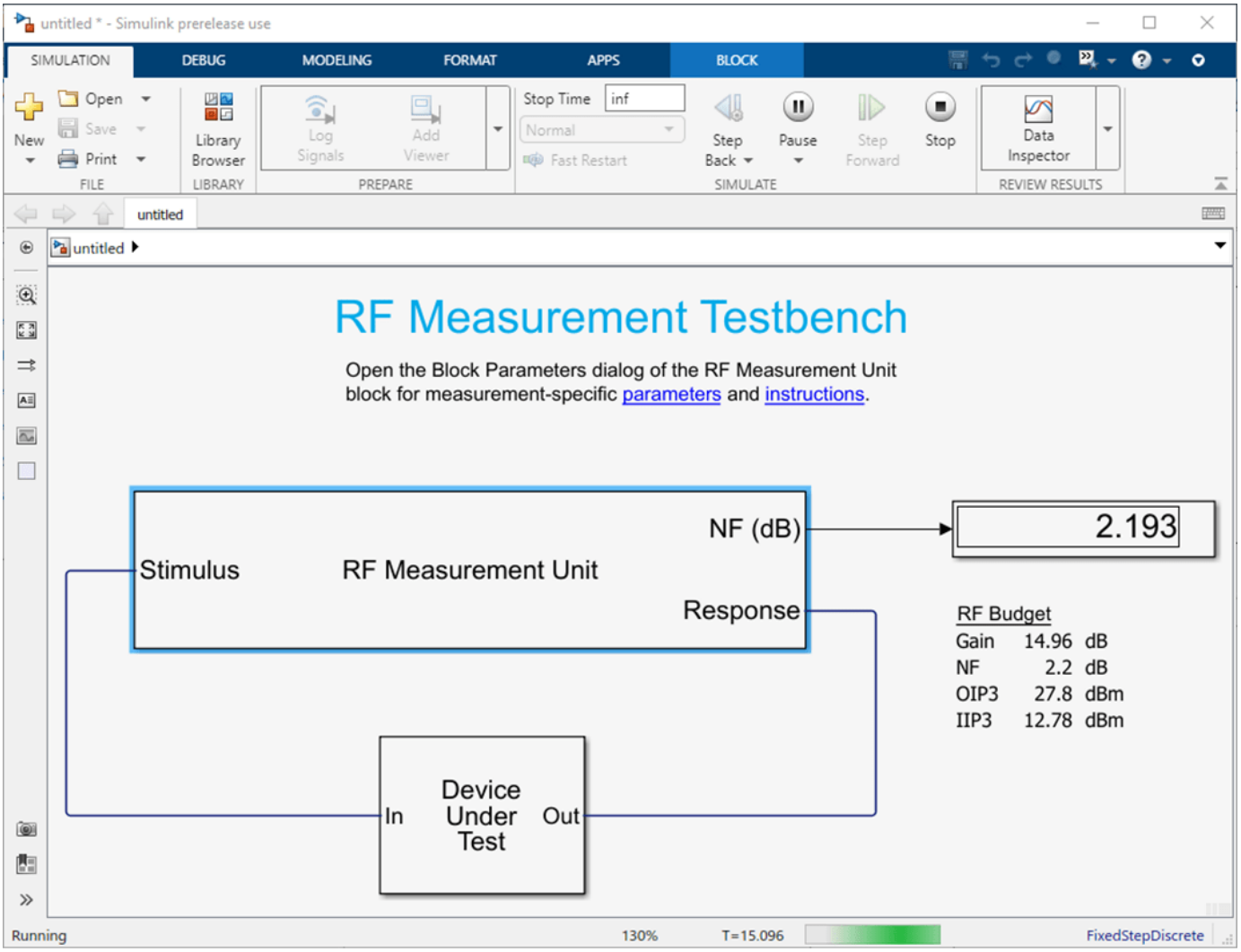
You can also use the `exportTestbench` function to verify with RF Blockset simulation. Type `exportTestbench(b)` command at the command line to open measurement testbench from RF budget object.

```
exportTestbench(b)
```

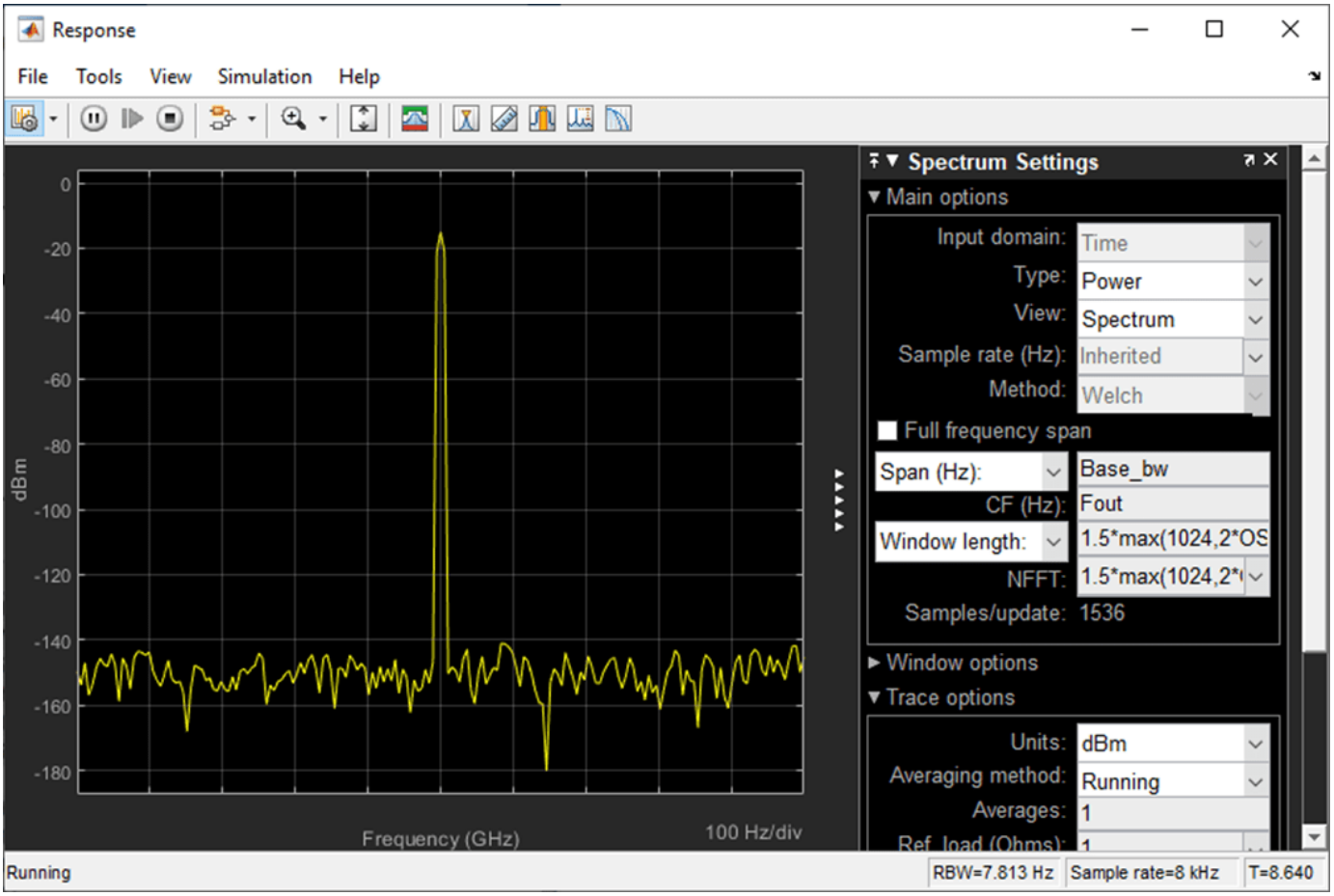
Click **Run** to see the Gain is about 15 dB.



Set the **Measured quantity** of the RF Measurement Unit to `NF` and then click **Run** to see the noise figure is about 2.2 dB.



The amplifier response is displayed in the Spectrum Analyzer window.



Create Amplifier Element from Touchstone File Containing Noise Data

Create an amplifier from the `default.s2p` Touchstone file.

```
a = amplifier(FileName='default.s2p')
```

```
a =
  amplifier: Amplifier element

               Name: 'Amplifier'
    UseNetworkData: 1
           FileName: 'default.s2p'
      NetworkData: [1x1 sparameters]
        NoiseData: [1x1 noiseParameters]
             OIP2: Inf
             OIP3: Inf
         NumPorts: 2
        Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Define a measured noise figure, noise frequencies, and the reference impedance data.

```
NF = [4 3 2 2 2 2 2 2.5 2.5 3 3.5];
freqs = (2:2:22)*1e9;
z0 = 50;
```

Build the noise parameters from the measured NF data.

```
np = noiseParameters(NF,freqs,z0);
```

Add this noise data to the `amplifier` object.

```
a = amplifier(FileName='default.s2p',NoiseData=np)
```

```
a =
  amplifier: Amplifier element

              Name: 'Amplifier'
    UseNetworkData: 1
          FileName: ''
       NetworkData: [1x1 sparameters]
         NoiseData: [1x1 noiseParameters]
              OIP2: Inf
              OIP3: Inf
          NumPorts: 2
         Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Create Amplifier Element

Create an `amplifier` object named 'LNA' and has a gain of 10 dB.

```
a = amplifier(Name='LNA',Gain=10)
```

```
a =
  amplifier: Amplifier element

              Name: 'LNA'
    UseNetworkData: 0
              Gain: 10
                NF: 0
              OIP2: Inf
              OIP3: Inf
               Zin: 50
              Zout: 50
          NumPorts: 2
         Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Amplifier Circuit

Create an amplifier object with a gain of 4 dB. Create another amplifier object that has an output third-order intercept (OIP3) 13 dBm.

```
amp1 = amplifier('Gain',4);
amp2 = amplifier('OIP3',13);
```

Build a 2-port circuit using the amplifiers.

```
c = circuit([amp1 amp2])
```

```
c =
  circuit: Circuit element

    ElementNames: {'Amplifier'  'Amplifier_1'}
        Elements: [1x2 amplifier]
           Nodes: [0 1 2 3]
            Name: 'unnamed'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

RF Budget Analysis of Series of RF Elements

Create an amplifier with a gain of 4 dB.

```
a = amplifier(Gain=4);
```

Create a modulator with an OIP3 of 13 dBm.

```
m = modulator(OIP3=13);
```

Create an N-port element using `passive.s2p`.

```
n = nport('passive.s2p');
```

Create an RF element with a gain of 10 dB.

```
r = rfelement(Gain=10);
```

Calculate the RF budget of a series of RF elements at an input frequency of 2.1 GHz, an available input power of –30 dBm, and a bandwidth of 10 MHz.

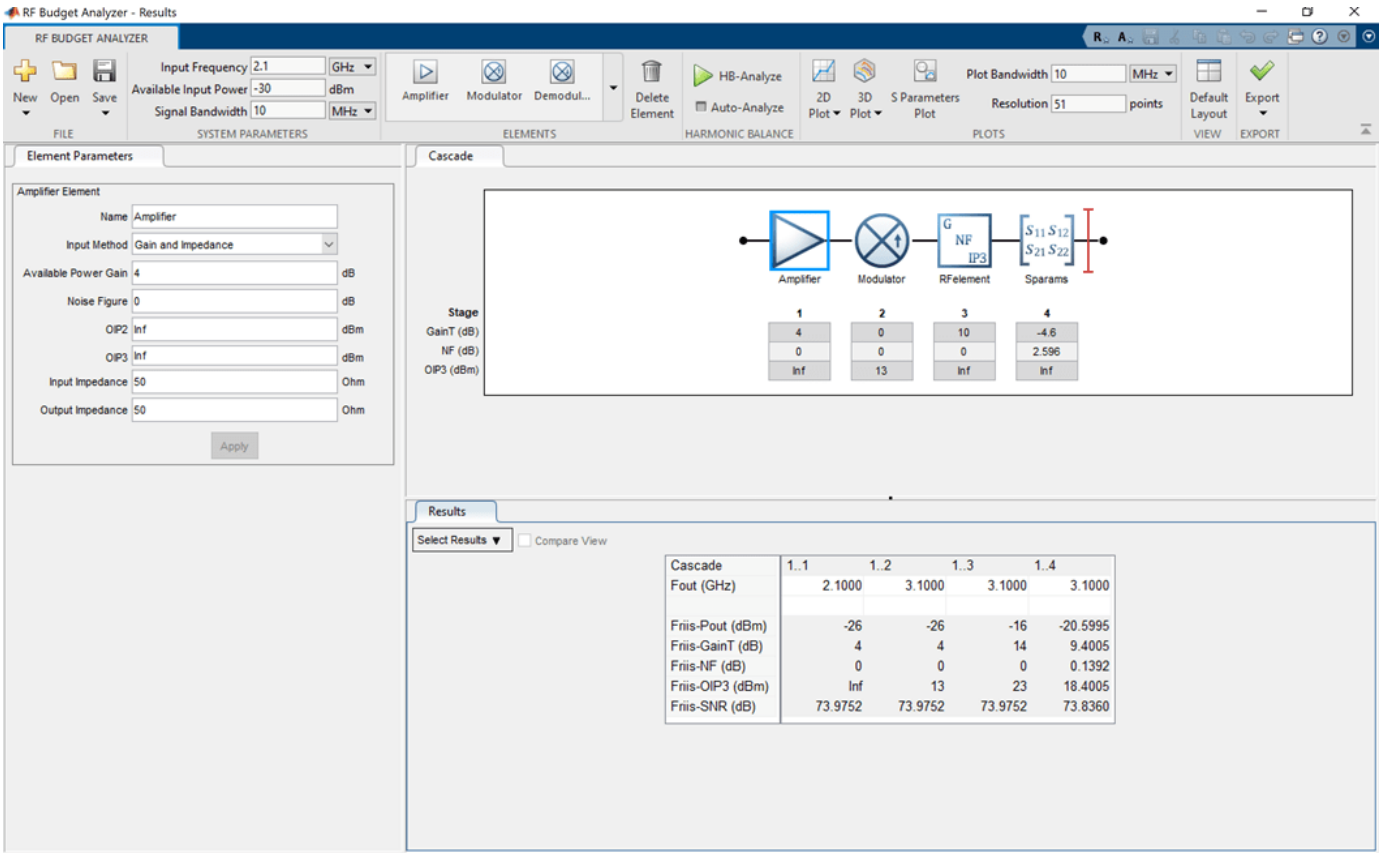
```
b = rfbudget([a m r n],2.1e9,-30,10e6)
```

```
b =
  rfbudget with properties:

              Elements: [1x4 rf.internal.rfbudget.Element]
        InputFrequency: 2.1 GHz
    AvailableInputPower: -30 dBm
        SignalBandwidth:  10 MHz
                Solver: Friis
            AutoUpdate: true

   Analysis Results
        OutputFrequency: (GHz) [  2.1    3.1    3.1     3.1]
            OutputPower: (dBm) [  -26    -26    -16   -20.6]
         TransducerGain: (dB)  [    4      4     14     9.4]
                     NF: (dB)  [    0      0      0  0.1392]
                  IIP2: (dBm) []
                  OIP2: (dBm) []
                  IIP3: (dBm) [  Inf      9      9       9]
```

```
OIP3: (dBm) [  Inf     13     23    18.4]
 SNR: (dB)  [73.98  73.98  73.98   73.84]
```

Type the `show` command at the command window to display the analysis in the **RF Budget Analyzer** app.

```
show(b)
```



Version History

Introduced in R2017a

Model RF amplifier using network parameters from two-port Touchstone file

To model an RF amplifier element using network parameters from a two-port Touchstone file, specify the name of the file in the FileName property. The object extracts the values of the NetworkData and the NoiseData properties from this file.

See Also

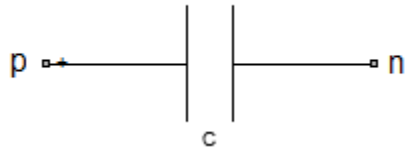
`modulator` | `nport` | `circuit` | `noiseParameters` | `add`

capacitor

Capacitor object

Description

Use the `capacitor` class to create a capacitor object that you can add to an existing circuit.



Creation

Syntax

```
cobj = capacitor(cvalue)
cobj = capacitor(cvalue,cname)
```

Description

`cobj = capacitor(cvalue)` creates a capacitor object, `cobj`, with a capacitance of `cvalue` and default name, `C`. `cvalue` must be a non-negative scalar.

`cobj = capacitor(cvalue,cname)` creates a capacitor object, `cobj`, with a capacitance of `cvalue` and name `cname`. `cname` must be a character vector.

Properties

`Capacitance` — Capacitance value

scalar

Capacitance value specified as a scalar in farads.

Example: `1e-12`

Example: `cobj.Capacitance = 1e-12`

`Name` — Name of capacitor object

`C` (default) | character vector

Name of capacitor object, specified as a character vector. Two elements in the same circuit cannot have the same name.

Example: `'cap'`

Example: `cobj.Name = 'cap'`

`Terminals` — Names of terminals of capacitor object

cell vector

Names of the terminals of capacitor object, specified as a cell vector. These names are always `p` and `n`.

Example: `{'p' 'n'}`

Example: `cobj.Terminals = {'p' 'n'}`

`ParentPath` — Full path of the circuit to which the capacitor object belongs

character vector

Full path of the circuit to which the capacitor object belongs, specified as character vector. This path appears only after the capacitor is added to the circuit.

Note "ParentPath" is only displayed after the capacitor has been added

into a circuit.

`ParentNodes` — Circuit nodes in the parent nodes connect to capacitor terminals

vector of integers.

Circuit nodes in the parent nodes connect to capacitor terminals, specified as a vector of integers. This property appears only after the capacitor is added to a circuit.

Example: `[1 2]`

Example: `lobj.ParentNodes = [1 2]`

Note "ParentNodes" are only displayed after the capacitor has been added

into a circuit.

Object Functions

clone Create copy of existing circuit element or circuit object

Examples

Create Capacitor and Display Properties

Create a capacitor of capacitance 2 microfarad and display its properties.

```
hC1 = capacitor(2e-6);
disp(hC1)
```

```
  capacitor: Capacitor element

    Capacitance: 2.0000e-06
           Name: 'C'
      Terminals: {'p'  'n'}
```

Create and Extract S-parameters of a Capacitor

Create a capacitor and extract S-parameters of the capacitor.

```
hC = capacitor(2e-6,'C2uf');
hckt = circuit('example2');
add(hckt,[1 2],hC)
setports(hckt, [1 0],[2 0])
freq = linspace(1e3,2e3,100);
S = sparameters(hckt,freq);
disp(S)
```

```
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [100x1 double]
     Parameters: [2x2x100 double]
      Impedance: 50

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

Add Capacitor to Circuit and Display Properties

Add capacitor to a circuit, display the parent path and parent nodes.

```
hC3 = capacitor(3e-6,'C3uf');
hckt3 = circuit('example3');
add(hckt3,[1 2],hC3)
setports(hckt3, [1 0],[2 0])
disp(hC3)
```

```
  capacitor: Capacitor element

    Capacitance: 3.0000e-06
           Name: 'C3uf'
      Terminals: {'p'  'n'}
    ParentNodes: [1 2]
     ParentPath: 'example3'
```

Version History

Introduced in R2013b

See Also

`resistor` | `inductor` | `lcladder` | `circuit`

rfckt.amplifier

RF Amplifier

Description

Use the `rfckt.amplifier` object to represent RF amplifiers that are characterized by network parameters, noise data, and nonlinear data

Use the `read` object function to read the amplifier data from a data file in one of the following formats:

- Touchstone
- Agilent P2D
- Agilent S2D
- AMP

Note If you set `NonLinearData` using `rfdata.ip3` or `rfdata.power`, then the property is converted from scalar OIP3 format to the format of `rfdata.ip3` or `rfdata.power`.

Creation

Syntax

```
h = rfckt.amplifier
h = rfckt.amplifier(Name,Value)
```

Description

`h = rfckt.amplifier` returns an amplifier circuit object whose properties all have their default values.

`h = rfckt.amplifier(Name,Value)` sets properties using one or more name-value pairs. For example, `rfckt.amplifier ('IntpType','Cubic')` creates an RF amplifier circuit that uses cubic interpolation. You can specify multiple name-value pairs. Enclose each property name in quotes. Properties not specified retain their default values.

Properties

`AnalyzedResult` — Computed S-parameters, noise figure, OIP3, and group delay values
`rfdata.data` object

This property is read-only.

Computed S-parameters, noise figure, OIP3, and group delay values, specified as an `rfdata.data` object. For more information refer, “Algorithms” on page 1-174.

Data Types: `function_handle`

`IntpType` — Interpolation method used in `rfckt.amplifier`
`'Linear'` (default) | `'Spline'` | `'Cubic'`

Interpolation method specified as one of the following values:

Method	Description
`Linear`	Linear interpolation
`Spline`	Cubic spline interpolation
`Cubic`	Piecewise cubic Hermite interpolation

Data Types: `char`

`Name` — Name of amplifier object
`1-by-N` character array

This property is read-only.

Name of amplifier object.

Data Types: `char`

`NetworkData` — Network parameter data
`rfdata.network` object

Network parameter data.

Data Types: `function_handle`

`NoiseData` — Noise information
scalar noise figure in decibels | `rfdata.noise` object | `rfdata.nf` object

Noise information, specified as one of the following:

- Scalar noise figure in dB
- `rfdata.noise` object
- `rfdata.nf` object

Data Types: `double` | `function_handle`

`NonlinearData` — Non-linearity information
scalar OIP3 in dB | `rfdata.power` object | `rfdata.ip3` object

Noise information, specified as one of the following:

- Scalar OIP3 in dBm
- `rfdata.power` object
- `rfdata.ip3` object

Data Types: `double` | `function_handle`

`nport` — Number of ports
positive integer

This property is read-only.

Number of ports. The default value is 2.

Data Types: `double`

Object Functions

analyze	Analyze RFCKT object in frequency domain
calculate	Calculate specified parameters for rfckt objects or rfdata objects
circle	Draw circles on Smith Chart
extract	Extract specified network parameters from rfckt object or data object
listformat	List valid formats for specified circuit object parameter
listparam	List valid parameters for specified circuit object
loglog	Plot specified circuit object parameters using log-log scale
plot	Plot circuit object parameters on X-Y plane
plotyy	Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes
getop	Display operating conditions
polar	Plot specified object parameters on polar coordinates
semilogx	Plot RF circuit object parameters using log scale for x-axis
semilogy	Plot RF circuit object parameters using log scale for y-axis
smith	Plot circuit object parameters on Smith chart
write	Write RF data from circuit or data object to file
getz0	Calculate characteristic impedance of RFCKT transmission line object
read	Read RF data from file to new or existing circuit or data object
restore	Restore data to original frequencies
getop	Display operating conditions
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Examples

Create RF Circuit Amplifier

Create an Amplifier using rfckt.amplifier object.

```
amp = rfckt.amplifier('IntpType','cubic')
```

```
amp =
   rfckt.amplifier with properties:

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

Set Nonlinearity Information in RF Circuit Amplifier Object

Create an RF amplifier using `rfckt.amplifier` object.

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

Create an `rfdata.power` object to store output power and phase information.

```
powerdata = rfdata.power;
```

Define frequency, phase shift, input power, and output power data.

```
f = [2.08 2.10]*1.0e9;
phase = {[27.1 35.3],[15.4 19.3 21.1]};
pin = {[0.001 0.002],[0.001 0.005 0.01]};
pout = {[0.0025 0.0031],[0.0025 0.0028 0.0028]};
```

Assign frequency, phase shift, input power, and output power data to an `rfdata.power` object.

```
powerdata.Freq = f;
powerdata.Phase = phase;
powerdata.Pin = pin;
powerdata.Pout = pout;
```

Set nonlinearity information parameter in the RF circuit amplifier object.

```
amp.NonlinearData = powerdata
```

```
amp =
   rfckt.amplifier with properties:

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

Algorithms

The `analyze` function computes the `AnalyzedResult` property using the data stored in the `rfckt.amplifier` object properties as follows:

- The `analyze` function uses the data stored in the `NoiseData` property of the `rfckt.amplifier` object to calculate the noise figure.
- The `analyze` function uses the data stored in the `NonlinearData` property of the `rfckt.amplifier` object to calculate OIP3.

  If power data exists in the `NonlinearData` property, the block extracts the AM/AM and AM/PM nonlinearities from the power data.

  If the `NonlinearData` property contains only IP3 data, the method computes and adds the nonlinearity by:

  1. Using the third-order input intercept point value in dBm to compute the factor, *f*, that scales the input signal before the amplifier object applies the nonlinearity:

     $$F_{AM/AM}(u) = u - \frac{u^3}{3}$$

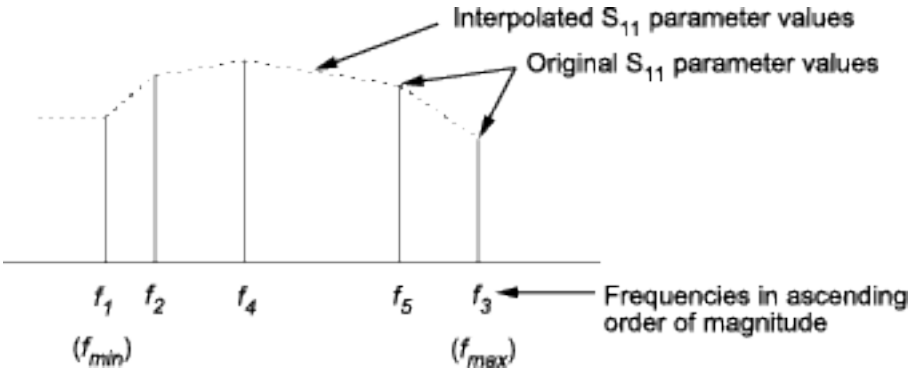
2. Computing the scaled input signal by multiplying the amplifier input signal by *f*.
3. Limiting the scaled input signal to a maximum value of 1.
4. Applying an AM/AM conversion to the amplifier gain, according to the following cubic polynomial equation:

$$F_{AM/AM}(u) = u - \frac{u^3}{3}$$

where *u* is the magnitude of the scaled input signal, which is a unitless normalized input voltage.

- The `analyze` function uses the data stored in the `'NetworkData'` property of the `rfckt.amplifier` object to calculate the group delay values of the amplifier at the frequencies specified in `freq`, as described in the `analyze` function reference page.
- The `analyze` function uses the data stored in the `NetworkData` property of the `rfckt.amplifier` object to calculate the S-parameter values of the amplifier at the frequencies specified in `freq`. If the `'NetworkData'` property contains network Y-parameters or Z-parameters, the `analyze` function first converts the parameters to S-parameters. Using the interpolation method you specify with the `IntpType` property, the `analyze` method interpolates the S-parameter values to determine their values at the specified frequencies.

  Specifically, the `analyze` function orders the S-parameters according to the ascending order of their frequencies, f_n. It then interpolates the S-parameters, using the MATLAB `interp1` function. For example, the curve in the following diagram illustrates the result of interpolating the S_{11} parameters at five different frequencies.



  For more information, see “One-Dimensional Interpolation” and the `interp1` reference page.

  As shown in the preceding diagram, the `analyze` function uses the parameter values at f_{min}, the minimum input frequency, for all frequencies smaller than f_{min}. It uses the parameters values at f_{max}, the maximum input frequency, for all frequencies greater than f_{max}. In both cases, the results may not be accurate, so you need to specify network parameter values over a range of frequencies that is wide enough to account for the amplifier behavior.

Version History

Introduced before R2006a

References

[1] EIA/IBIS Open Forum. *Touchstone File Format Specification*, Rev. 1.1, 2002 (https://ibis.org/connector/touchstone_spec11.pdf).

See Also

`rfckt.cascade` | `rfckt.coaxial` | `rfckt.cpw` | `rfckt.datafile` | `rfckt.delay` | `rfckt.hybrid` | `rfckt.hybridg` | `rfckt.mixer` | `rfckt.microstrip` | `rfckt.passive` | `rfckt.parallel` | `rfckt.parallelplate` | `rfckt.rlcgline` | `rfckt.series` | `rfckt.seriesrlc` | `rfckt.shuntrlc` | `rfckt.twowire` | `rfckt.txline`

Topics

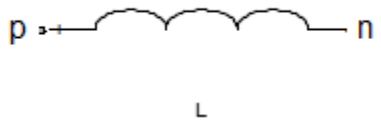
“Operations with RF Circuit Objects”
“Operations with RF Data Objects”

inductor

Inductor object

Description

Use `inductor` class to create an inductor object that you can add to an existing circuit.



Creation

Syntax

```
lobj = inductor(lvalue)
lobj = inductor(lvalue,lname)
```

Description

`lobj = inductor(lvalue)` creates a inductor object, `lobj`, with a inductance of `lvalue` and default name, `L`. `lvalue` must be a numeric positive scalar.

`lobj = inductor(lvalue,lname)` creates a inductor object, `lobj`, with a inductance of `lvalue` and name `lname`. `lname` must be a character vector.

Properties

`Inductance` — Inductance

scalar

Inductance, specified as a scalar in henries.

Example: `1e-9`

Example: `lobj.Inductance = 1e-9`

`Name` — Object name

L (default) | character vector

Name of inductor object, specified as a character vector. Two elements in the same circuit cannot have the same name.

Example: `'cap'`

Example: `lobj.Name = 'indcutor1'`

`Terminals` — Names of terminals of inductor object

cell vector

Names of the terminals of inductor object, specified as a cell vector These names are always `p` and `n`.

Example: `{'p' 'n'}`

Example: `lobj.Terminals = {'p' 'n'}`

`ParentPath` — Full path of the circuit

character vector

Full path of the circuit to which the inductor object belongs, specified as character vector. This path appears only after the inductor is added to the circuit.

Note "ParentPath" is only displayed after the capacitor has been added

into a circuit.

`ParentNodes` — Parent nodes connected to inductor terminals

vector of integers.

Parent nodes connected to inductor terminals, specified as a vector of integers. This property appears only after the inductor is added to a circuit.

Example: `[1 2]`

Example: `lobj.ParentNodes = [1 2]`

Note "ParentNodes" are only displayed after the capacitor has been added

into a circuit.

Object Functions

clone Create copy of existing circuit element or circuit object

Examples

Create and Display Inductor

Create an inductor of 3e-9 henry and display the properties.

```
hL1 = inductor(3e-9);
disp(hL1)
```

```
  inductor: Inductor element

    Inductance: 3.0000e-09
          Name: 'L'
     Terminals: {'p'  'n'}
```

Create and Extract S-parameters of Inductor

Create an inductor object and extract the s-parameters of this inductor.

```
hL = inductor(3e-9,'L3nh');
hckt = circuit('example2');
add(hckt,[1 2],hL)
setports (hckt, [1 0],[2 0])
freq = linspace (1e3,2e3,100);
S = sparameters(hckt,freq);
disp(S)
```

```
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [100x1 double]
     Parameters: [2x2x100 double]
      Impedance: 50

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

Add Inductor to Circuit and Display Properties

Add an inductor to a circuit, display the parent path and parent nodes.

```
hL = inductor(3e-9,'L3n9');
hckt = circuit('example3');
add(hckt,[1 2],hL)
setports(hckt, [1 0],[2 0])
disp(hL)
```

```
  inductor: Inductor element

     Inductance: 3.0000e-09
           Name: 'L3n9'
      Terminals: {'p'  'n'}
    ParentNodes: [1 2]
     ParentPath: 'example3'
```

Version History

Introduced in R2013b

See Also

`resistor` | `capacitor` | `inductor` | `circuit`

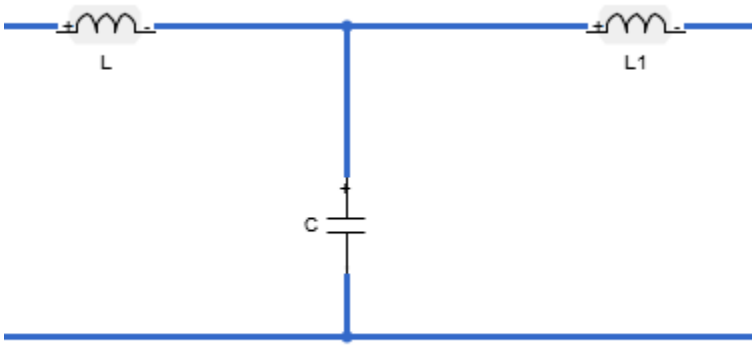
lcladder

Create LC ladder network

Description

Use an `lcladder` object:

- To create an LC ladder filter.
- To convert an `rffilter` object to an LC ladder.
- To model an LC circuit in cascaded stages using an `rfbudget` object or the **RF Budget Analyzer** app.



Creation

Syntax

```
lcobj = lcladder(top,L,C)
lcobj = lcladder(rffilterobj)
lcobj = lcladder( ___ ,lcname)
```

Description

`lcobj = lcladder(top,L,C)` creates an LC ladder object `lcobj` and sets the “Topology” on page 1-0 , “Inductances” on page 1-0 , and “Capacitances” on page 1-0 properties.

`lcobj = lcladder(rffilterobj)` creates a LC ladder object `lcobj` from an RF filter object `rffilterobj`.

`lcobj = lcladder( ___ ,lcname)` creates an LC ladder object `lcobj` and sets the “Name” on page 1-0 property. Specify `lcname` after all other input arguments.

Input Arguments

`rffilterobj` — RF filter object

`rffilter` object

RF filter object, specified as an `rffilter` object.

Example: `lcobj = lcladder(rffilterobj)`

Properties

`Topology` — Topology type of LC ladder

`'lowpasspi'` | `'lowpasstee'` | `'highpasspi'` | `'highpasstee'` | `'bandpasspi'` | `'bandpasstee'` | `'bandpasstee'` | `'bandstoppi'` | `'bandstoptee'`

Topology type of the LC ladder, specified as one of the following:

- `'lowpasspi'` — Low-pass pi filter
- `'lowpasstee'` —Low-pass tee filter
- `'highpasspi'` — High-pass pi filter
- `'highpasstee'` — High-pass tee filter
- `'bandpasspi'` — Band-pass pi filter
- `'bandpasstee'` — Band-pass tee filter
- `'bandstoppi'` — Band-stop pi filter
- `'bandstoptee'` — Band-stop tee filter

Example: `'lowpasspi'`

`Inductances` — Inductor values in LC ladder

`inductor` object | numeric scalar | two-element vector

Inductor values in the LC ladder, specified as an `inductor` object or as a numeric scalar or two-element vector in henries.

Example: `1e-9`

`Capacitances` — Capacitor values in LC ladder

`capacitor` object | numeric scalar | two-element vector

Capacitor values in the LC ladder, specified as a `capacitor` object or as a numeric scalar or two-element vector in farads.

Example: `[2e-12 3e-12]`

`Name` — Name of LC ladder object

`'lcfilt'` (default) | character vector | string scalar

Name of the LC ladder object, specified as a character vector or string scalar.

Example: `'lcfilter'`

`NumPorts` — Number of ports in LC ladder object

2 (default) | positive scalar

This property is read-only.

Number of ports in LC ladder object. specified as a positive scalar.

`Terminals` — Terminal names of LC ladder object

{'p1+' 'p2+' 'p1-' 'p2-'} | cell vector

This property is read-only.

Terminal names of LC ladder object, specified as the cell vector, `{'p1+' 'p2+' 'p1-' 'p2-'}`. An LC ladder object always has four terminals: two terminals associated with the first port (`'p1+'` and `'p1-'`) and two terminals associated with the second port (`'p2+'` and `'p2-'`).

Example: `{'p1+' 'p2+' 'p1-' 'p2-'}`

`ParentNodes` — Parent circuit nodes connected to LC ladder object terminals

vector of integers

Parent circuit nodes connected to LC ladder object terminals, specified as a vector of integers. This property appears only after the LC ladder object is added to a circuit.

Note `ParentNodes` are only displayed after the capacitor has been added into a circuit.

`ParentPath` — Full path of the circuit to which the LC ladder object belongs

character vector

Full path of the circuit to which the LC ladder object belongs, specified as character vector. This path appears only after the inductor is added to the circuit.

Note `ParentPath` is only displayed after the capacitor has been added into a circuit.

Object Functions

sparameters	Calculate S-parameters for RF data, network, circuit, and matching network objects
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object
clone	Create copy of existing circuit element or circuit object
rfplot	Plot S-parameter data

Examples

Create Low-Pass Pi LC Ladder Object and Plot S-Parameters

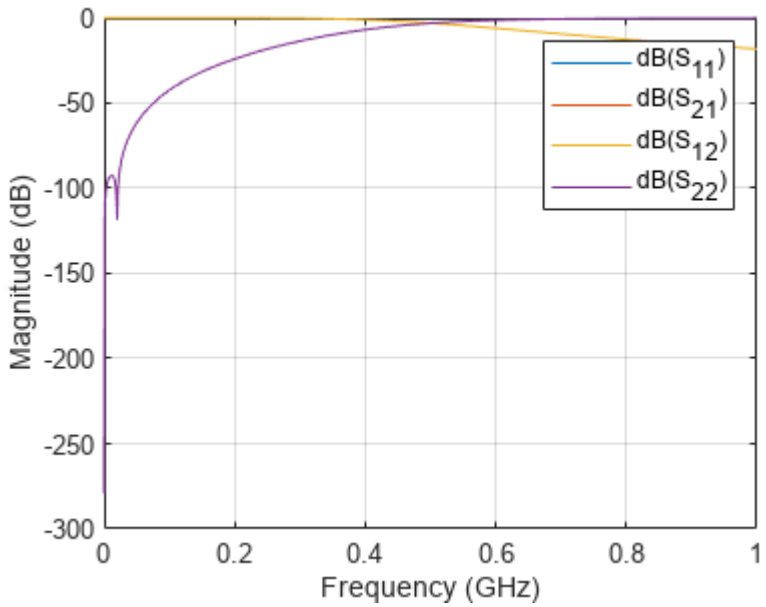
Create a low-pass pi LC ladder object with an inductor value of 3.18e-8 H and a capacitor value of 6.37e-12 F. Calculate and plot the s-parameters.

```
L = 3.18e-8;
C = [6.37e-12 6.37e-12];
lpp = lcladder('lowpasspi',L,C)
```

```
lpp =
  lcladder: LC Ladder element
```

```
        Topology: 'lowpasspi'
     Inductances: 3.1800e-08
    Capacitances: [6.3700e-12 6.3700e-12]
            Name: 'lcfilt'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

```
freq = 0:1e6:1e9;
S = sparameters(lpp,freq);
rfplot(S)
```



You can also add this LC ladder to a circuit.

```
c = circuit;
add(c,[1 2 0 0],lpp)
setports(c,[1 0],[2 0])
```

Design RF Chain with LC Ladder

Design a low-pass pi LC ladder circuit.

```
L  = 1e-9;
C = [1e-12 1e-12];
lpp = lcladder('lowpasspi',L,C);
```

Design a default transmission line delay lossy and lossless objects.

```
tx1 = txlineDelayLossless;
tx2 = txlineDelayLossy;
```

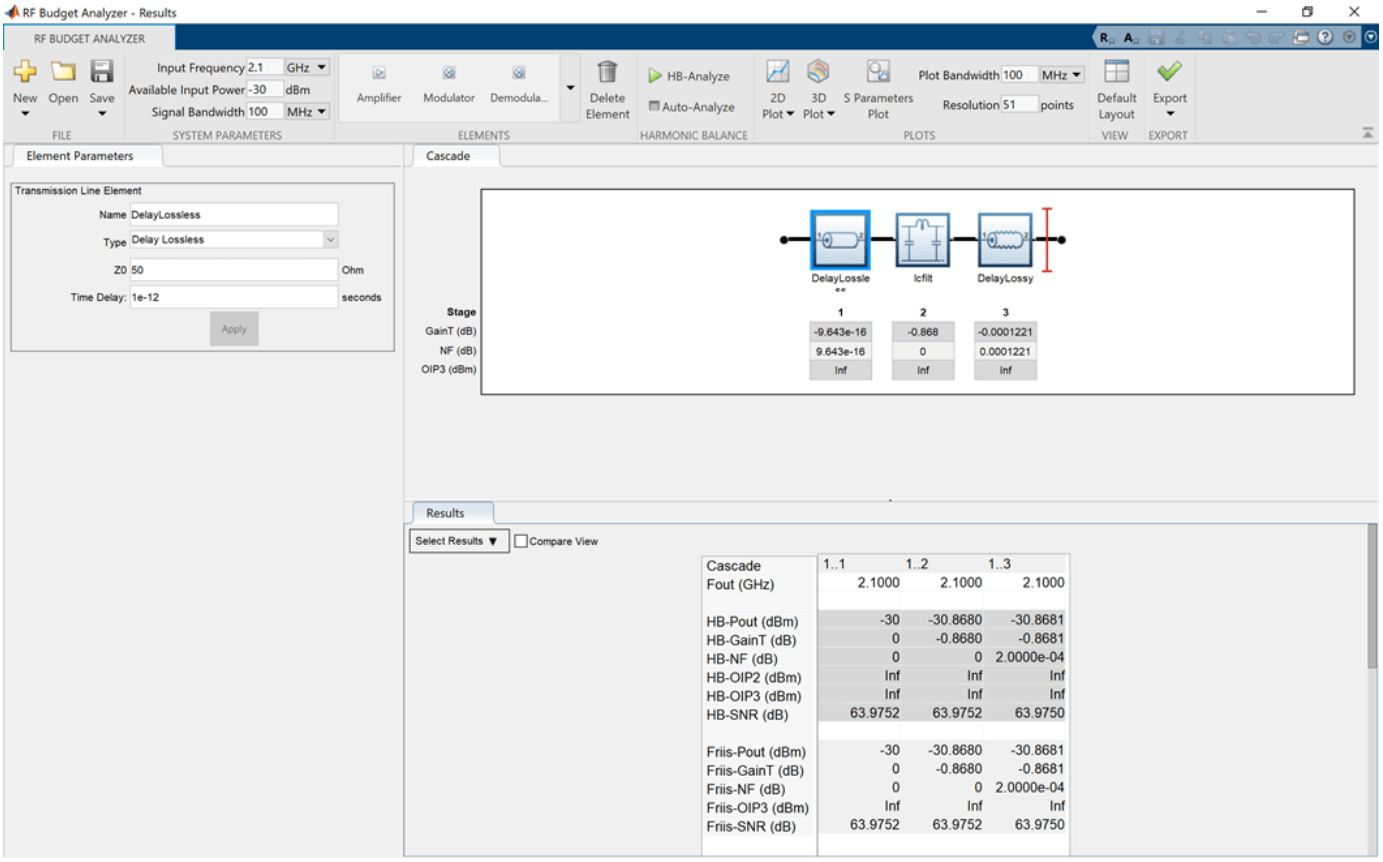
Create an `rfbudget` object to design an RF chain.

```
b = rfbudget([tx1 lpp tx2 ],2.1e9,-30,100e6,'Solver','HarmonicBalance')
```

```
b =
  rfbudget with properties:

              Elements: [1x3 rf.internal.rfbudget.Element]
        InputFrequency: 2.1 GHz
    AvailableInputPower: -30 dBm
        SignalBandwidth: 100 MHz
                 Solver: HarmonicBalance
               WaitBar: true
            AutoUpdate: true

   Analysis Results
         OutputFrequency: (GHz) [       2.1        2.1        2.1]
             OutputPower: (dBm) [       -30     -30.87     -30.87]
          TransducerGain: (dB)  [-6.514e-10     -0.868    -0.8681]
                      NF: (dB)  [-2.132e-14 -2.132e-14  0.0001761]
                    IIP2: (dBm) [       Inf        Inf        Inf]
                    OIP2: (dBm) [       Inf        Inf        Inf]
                    IIP3: (dBm) [       Inf        Inf        Inf]
                    OIP3: (dBm) [       Inf        Inf        Inf]
                     SNR: (dB)  [     63.98      63.98      63.98]
```

Use the `show` command at the command line to visualize the RF budget chain in the **RF Budget Analyzer** app. You can also do further analysis on this chain using the app. For more information, see RF Budget Analyzer.

```
show(b)
```



Version History

Introduced in R2013b

See Also

`resistor` | `capacitor` | `inductor` | `circuit` | `rffilter`

nport

Create linear n-port circuit element

Description

The `nport` class creates an n-port object that can be added into an RF Toolbox circuit. The n-port S-parameters define the n-port object.

Creation

Syntax

```
nport_obj = nport(filename)
nport_obj = nport(netparamobj)
nport_obj = nport( ___ ,name)
nport_obj = nport(Name=Value)
```

Description

`nport_obj = nport(filename)` creates an n-port object and sets the FileName property.

`nport_obj = nport(netparamobj)` creates an n-port object and sets the NetworkData.

`nport_obj = nport( ___ ,name)` creates an n-port object and sets the Name property. Specify the name-value argument after any of the input argument combinations in the previous syntaxes.

`nport_obj = nport(Name=Value)` sets the "Properties" on page 1-186 of an `nport` object using one or more name-value arguments. For example, `nport_obj = nport(FileName='passive.s2p')` creates an `nport` object from `passive.s2p`. Properties you do not specify retain their default values.

Properties

`FileName` — Touchstone file
string scalar | character vector

Touchstone file, specified as a string scalar or character vector.

Note The Touchstone file must be passive at all specified frequencies. To make n-port S-parameters passive, use the `makepassive` function.

Data Types: `char` | `string`

`NumPorts` — Number of ports
scalar

Number of ports, specified as a scalar.

Example: 2

`NetworkData` — Network data
scalar

Network data, specified as a scalar. Network data can be of S, Z, Y, ABCD, h, or g-parameters.

Example: [1x1 sparameters]

`Name` — Name of n-port object
scalar

Name of n-port object, specified as a scalar.

Example: obj

`Ports` — Port names
cell vector

Port names, stored as a cell vector. This property is a read only.

Example: {’p1’ ‘p2’}

`Terminals` — Terminal names
cell vector

Terminal names, stored as a cell vector. There are two terminals per port. The positive terminal names are listed first ('p1+', 'p2+'...) followed by the negative terminal ('p1–', 'p2–'...). This property is read only.

`ParentNodes` — Parent circuit nodes connected to n-port object terminals
vector of integers.

Parent circuit nodes connected to n-port object terminals, stored as a vector of integers. `ParentNodes` is same length as `Terminals`. This property is read only and appears only after you add the n-port data.

`ParentPath` — Full path of circuit to which n-port object belongs
character vector

Full path of the circuit to which the n-port object belongs, stored as character vector. This property is read only and appear only after you add the n-port object is added to the circuit.

Object Functions

smithplot	Plot impedance transformation for selected matching network on smith chart
clone	Create copy of existing circuit element or circuit object

Examples

Create N-port Object

Create and display N-port data object.

```
npass = nport('passive.s2p')

npass =
  nport: N-port element

       FileName: 'passive.s2p'
    NetworkData: [1x1 sparameters]
           Name: 'Sparams'
       NumPorts: 2
      Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Add N-Port Object to Circuit

Add a N-port object to a circuit. Display the object.

```
nobj = nport('passive.s2p');
ckt = circuit('example');
add(ckt,[1 2],nobj)
disp(nobj)

  nport: N-port element

       FileName: 'passive.s2p'
    NetworkData: [1x1 sparameters]
           Name: 'Sparams'
       NumPorts: 2
      Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
    ParentNodes: [1 2 0 0]
     ParentPath: 'example'
```

Version History

Introduced in R2013b

See Also

`resistor` | `capacitor` | `inductor` | `circuit` | `rfbudget`

resistor

Resistor object

Description

Use the `resistor` class to create a resistor object that you can add to an existing circuit.

Creation

Syntax

```
robj = resistor(rvalue)
robj = resistor(rvalue,rname)
```

Description

`robj = resistor(rvalue)` with a resistance of `rvalue` and default name, `R`. `rvalue` must be a numeric non-negative scalar.

`robj = resistor(rvalue,rname)` creates a resistor object, `robj`, with a resistance of `rvalue` and name `rname`. `rname` must be a character vector.

Properties

`Resistance` — Resistance value
scalar

Resistance value specified as a scalar in ohms.

Example: `50`

Example: `robj.Resistance = 50`

`Name` — Name of resistor object
`R` (default) | character vector

Name of resistor object, specified as a character vector. Two elements in the same circuit cannot have the same name.

Example: `'resis'`

Example: `robj.Name = 'resis'`

`Terminals` — Names of terminals of resistor object

cell vector

Names of the terminals of resistor object, specified as a cell vector. This property is read-only. The names `p` and `n` stand for positive and negative terminals, respectively.

`ParentPath` — Full path of the circuit to which the resistor object belongs

character vector

Full path of the circuit to which the resistor object belongs, specified as character vector. This path appears only after the resistor is added to the circuit.

Note "ParentPath" is only displayed after the resistor has been added into a circuit.

`ParentNodes` — Circuit nodes in the parent nodes connect to resistor terminals

vector of integers.

Circuit nodes in the parent nodes connect to resistor terminals, specified as a vector of integers. This property is read-only and appears only after the resistor is added to a circuit.

Note "ParentNodes" are only displayed after the resistor has been added into a circuit.

Object Functions

clone Create copy of existing circuit element or circuit object

Examples

Create Resistor and Display Properties

Create a resistor of resistance 100 ohms and display its properties.

```
hR1 = resistor(100);
disp(hR1)
```

```
  resistor: Resistor element

    Resistance: 100
          Name: 'R'
     Terminals: {'p'  'n'}
```

Create and Extract S-parameters of Resistor

Create an resistor object and extract the s-parameters of this resistor.

```
hR = resistor(50,'R50');
hckt = circuit('example2');
add(hckt,[1 2],hR)
setports (hckt, [1 0],[2 0])
```

```
freq = linspace (1e3,2e3,100);
S = sparameters(hckt,freq);
disp(S)
```

```
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [100x1 double]
     Parameters: [2x2x100 double]
      Impedance: 50

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

Add Resistor to Circuit and Display Properties

Add a resistor to a circuit, display the parent path and parent nodes.

```
hR = resistor(150,'R150');
hckt = circuit('resistorcircuit');
add(hckt,[1 2],hR)
setports(hckt, [1 0],[2 0])
disp(hR)
```

```
  resistor: Resistor element

     Resistance: 150
           Name: 'R150'
      Terminals: {'p'  'n'}
    ParentNodes: [1 2]
     ParentPath: 'resistorcircuit'
```

Version History

Introduced in R2013b

See Also

`capacitor` | `inductor` | `circuit`

rfchain

Create `rfchain` object

Description

Use the `rfchain` object to create an RF circuit cascade analysis object to calculate gain, noise figure, OIP3 (output third-order intercept), and IIP3 (input third order intercept).

Creation

Syntax

```
rfobj = rfchain()
obj = rfchain(g, nf, o3, 'Name', nm)
obj = rfchain(g, nf, 'IIP3', i3, 'Name', nm)
```

Description

`rfobj = rfchain()` creates an RF chain object `obj` having zero stages. To add stages to the RF chain, use `addstage` method.

`obj = rfchain(g, nf, o3, 'Name', nm)` creates an RF chain object `obj` having `N` stages. The gain `g`, noise figure `nf` and the OIP3 `o3` are vectors of length `N` . The name `nm` is a cell array of length `N` .

`obj = rfchain(g, nf, 'IIP3', i3, 'Name', nm)` creates an RF chain object having `N` stages, specifying an IIP3 for each stage, instead of an OIP3.

Properties

`Numstages` — Number of stages
scalar

Number of stages in an RF chain, specified as a scalar.

Data Types: `double`

`Name` — Name of each stage
character vector | string scalar

Name of each stage of an RF chain, specified as a character vector. This will always be a name-value pair.

Data Types: `char` | `string`

`Gain` — Gain of each stage
vector

Gain, in dB, of each stage in an RF chain, specified as a vector.

Data Types: `double`

`NoiseFigure` — Noise figure of each stage
vector

Noise figure, in dB, of each stage in an RF chain, specified as a vector.

Data Types: `double`

`OIP3` — Output-referred third-order intercept
vector

Output-referred third-order intercept, in dB, of each stage in an RF chain, specified as a vector.

Data Types: `double`

`IIP3` — Input-referred third-order intercept
vector

Input-referred third-order intercept, in dB, of each stage in an RF chain, specified as a vector.

Data Types: `double`

Examples

Create RF Chain Object, Add Stages, and View Results

Create an RF chain object.

```
rfch = rfchain;
```

Add stage 1 and stage 2 with gain, noise figure, oip3.

```
addstage(rfch, 21, 15, 30, 'Name', 'amp1');
addstage(rfch, -5, 6, Inf, 'Name', 'filt1');
```

Add stage 3 and stage 4 with gain, noise figure, iip3.

```
addstage(rfch, 7, 5, 'IIP3', 10, 'Name', 'lna1');
addstage(rfch, 12, 14, 'IIP3', 20, 'Name', 'amp2');
```

Calculate the gain, noise figure, oip3, and iip3 of each stage.

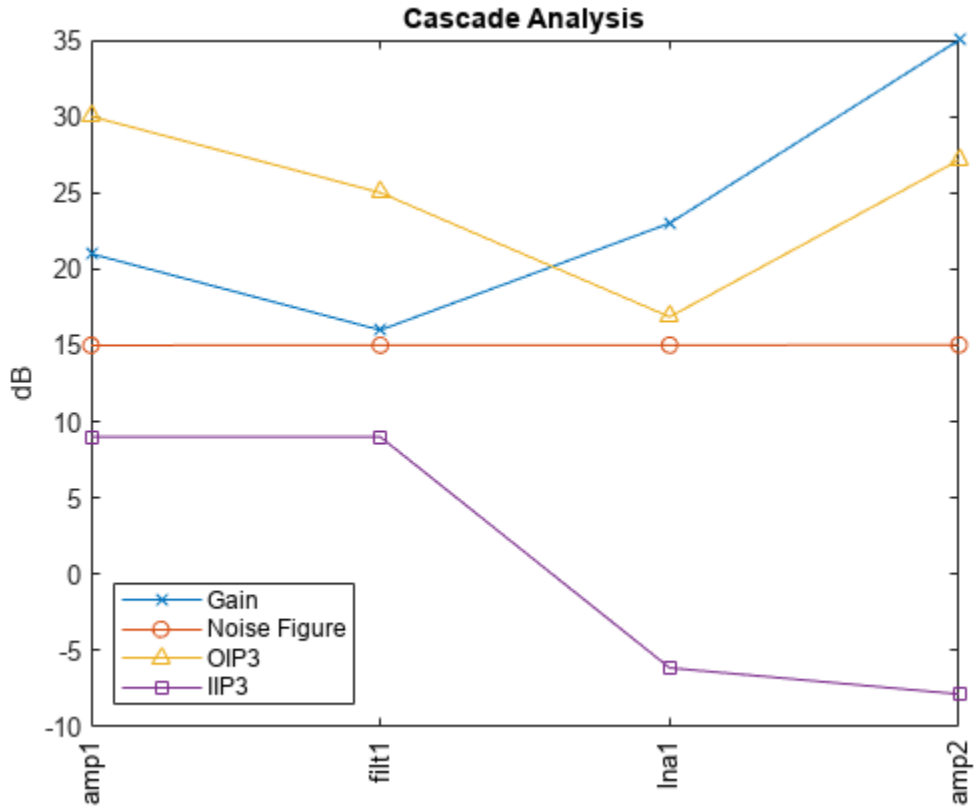
```
g = cumgain(rfch);
nf = cumnoisefig(rfch);
oip3val = cumoip3(rfch);
iip3val = cumiip3(rfch);
```

View the results on a table and plot it.

```
worksheet(rfch)
```

	amp1	filt1	lna1	amp2
Stage Gain	21	-5	7	12
Stage Noise Figure	15	6	5	14
Stage OIP3	30	Inf	17	32
Stage IIP3	9	Inf	10	20
Cascaded Gain	21	16	23	35
Cascaded Noise Figure	15	15.0033	15.0107	15.0272
Cascaded OIP3	30	25	16.8648	27.1451
Cascaded IIP3	9.0000	9.0000	-6.1352	-7.8549

```
figure
plot(rfch)
```



Create RF Chain Adding Stage-By-Stage Values

Assign three stage-by-stage values of gain, noise figure, OIP3 and stage names.

```
g = [11 -3 7];
nf = [25 3 5];
o3 = [30 Inf 10];
nm = {'amp1','filt1','lna1'};
```

Create an RF chain object.

```
rfch = rfchain(g,nf,o3,'Name',nm);
```

View results in a worksheet.

```
worksheet(rfch)
```

	amp1	filt1	lna1
Stage Gain	11	-3	7
Stage Noise Figure	25	3	5
Stage OIP3	30	Inf	10
Stage IIP3	19	Inf	3
Cascaded Gain	11	8	15
Cascaded Noise Figure	25	25.0011	25.0058
Cascaded OIP3	30	27	9.9827
Cascaded IIP3	19	19	-5.0173

Version History

Introduced in R2013b

See Also

`resistor` | `capacitor` | `inductor` | `circuit`

modulator

Modulator object

Description

Use the `modulator` object to create a modulator element. A modulator is a 2-port RF circuit object. You can use this element in the `rfbudget` object and the `circuit` object.

Creation

Syntax

```
mod = modulator
mod = modulator(Name,Value)
```

Description

`mod = modulator` creates a modulator object, `mod`, with default property values.

`mod = modulator(Name,Value)` creates a modulator object with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

Properties

`Name` — Name of modulator
`'Modulator'` (default) | character vector

Name of modulator, specified as the comma-separated pair consisting of `'Name'` and a character vector. All names must be valid MATLAB variable names.

Example: `'Name','mod'`

`Gain` — Available power gain
`0` (default) | nonnegative scalar

Available power gain, specified as a nonnegative scalar in dB.

Example: `'Gain',10`

`NF` — Noise figure
`0` (default) | real finite nonnegative scalar

Noise figure, specified as a real finite nonnegative scalar in dB.

Example: `'NF',-10`

`OIP2` — Second -order output-referred intercept point
`Inf` (default) | real scalar

Second -order output-referred intercept point, specified as a real scalar in dBm.

Example: `'OIP2',8`

Example: `amplifier.OIP2 = 8`

`OIP3` — Third -order output-referred intercept point
`Inf` (default) | real scalar

Third -order output-referred intercept point, specified as a real scalar in dBm.

Example: `'OIP3',10`

Example: `amplifier.OIP3 = 10`

`LO` — Local oscillator frequency
`1e9` (default) | real finite positive scalar

Local oscillator frequency, specified as a real finite positive scalar in Hz.

Example: `'LO',2e9`

`ConverterType` — Type of modulator
`'Up'` (default) | `'Down'`

Type of modulator, specified as `'Down'` or `'Up'`

Example: `'ConverterType','Up'`

`ImageReject` — Ideal image reject filtering
`true or 1` (default) | `false or 0`

Ideal image reject filtering at the input of the modulator, specified as a numeric or logical `1 (true)` or `0 (false)` . Setting this property to `false or 0` might affect harmonic balance results.

Example: `'ImageReject',1`

Example: `'ImageReject',true`

`ChannelSelect` — Ideal channel select filtering
`true or 1` (default) | `false or 0`

Ideal channel select filtering at the output of the modulator, specified as a numeric or logical `1 (true) or 0 (false)`. Setting this property to `false or 0` might affect harmonic balance results.

Example: `'ChannelSelect',1`

Example: `'ChannelSelect',false`

`Zin` — Input impedance
`50` (default) | positive real part finite scalar

Input impedance, specified as a positive real part finite scalar in ohms. You can also use a complex value with a positive real part.

Example: `'Zin',40`

`Zout` — Output impedance

`50` (default) | positive real part finite scalar

Output impedance, specified as a scalar in ohms. You can also use a complex value with a positive real part.

Example: `'Zout',40`

`NumPorts` — Number of ports

2 (default) | scalar integer

Number of ports, specified as a scalar integer. This property is read-only.

`Terminals` — Names of port terminals

`{'p1+' 'p2+' 'p1-' 'p2-'}` (default) | cell vector

Names of port terminals, specified as a cell vector. This property is read-only.

Object Functions

clone Create copy of existing circuit element or circuit object

Examples

Modulator Element

Create a downconverter modulator with a local oscillator (LO) frequency of 100 MHz.

```
m = modulator('ConverterType','Down','LO',100e6)
```

```
m =
  modulator: Modulator element

               Name: 'Modulator'
               Gain: 0
                 NF: 0
               OIP2: Inf
               OIP3: Inf
                Zin: 50
               Zout: 50
                 LO: 100000000
    ConverterType: 'Down'
      ImageReject: 1
    ChannelSelect: 1
         NumPorts: 2
        Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Modulator Circuit

Create a modulator object with a gain of 4 dB and local oscillator (LO) frequency of 2 GHz. Create another modulator object that is an upconverter and has an output third-order intercept (OIP3) of 13 dBm.

```
mod1 = modulator('Gain',4,'LO',2e9);
mod2 = modulator('OIP3',13,'ConverterType','Up');
```

Build a 2-port circuit using the modulators.

```
c = circuit([mod1 mod2])
```

```
c =
  circuit: Circuit element

    ElementNames: {'Modulator'  'Modulator_1'}
        Elements: [1x2 modulator]
           Nodes: [0 1 2 3]
            Name: 'unnamed'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

RF Budget Analysis of Series of RF Elements

Create an amplifier with a gain of 4 dB.

```
a = amplifier(Gain=4);
```

Create a modulator with an OIP3 of 13 dBm.

```
m = modulator(OIP3=13);
```

Create an N-port element using `passive.s2p`.

```
n = nport('passive.s2p');
```

Create an RF element with a gain of 10 dB.

```
r = rfelement(Gain=10);
```

Calculate the RF budget of a series of RF elements at an input frequency of 2.1 GHz, an available input power of –30 dBm, and a bandwidth of 10 MHz.

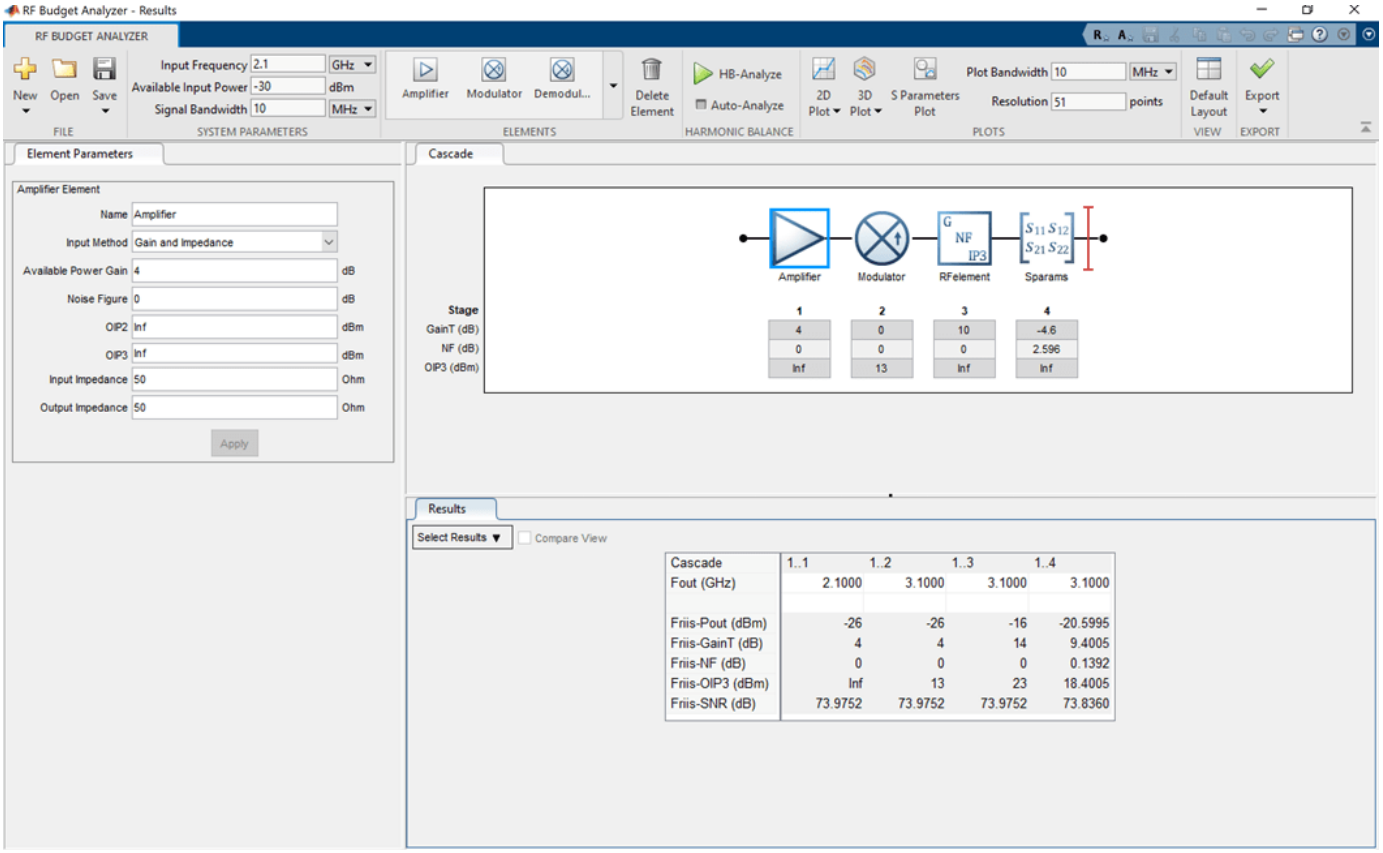
```
b = rfbudget([a m r n],2.1e9,-30,10e6)
```

```
b =
  rfbudget with properties:

              Elements: [1x4 rf.internal.rfbudget.Element]
        InputFrequency: 2.1 GHz
    AvailableInputPower: -30 dBm
        SignalBandwidth:  10 MHz
                Solver: Friis
            AutoUpdate: true

   Analysis Results
        OutputFrequency: (GHz) [  2.1    3.1    3.1     3.1]
            OutputPower: (dBm) [  -26    -26    -16   -20.6]
         TransducerGain: (dB)  [    4      4     14     9.4]
                     NF: (dB)  [    0      0      0  0.1392]
```

```
IIP2: (dBm) []
OIP2: (dBm) []
IIP3: (dBm) [  Inf      9      9       9]
OIP3: (dBm) [  Inf     13     23    18.4]
 SNR: (dB)  [73.98  73.98  73.98   73.84]
```

Type the `show` command at the command window to display the analysis in the **RF Budget Analyzer** app.

```
show(b)
```



Version History

Introduced in R2017a

See Also

`amplifier` | `nport` | `rfbudget`

circuit

Circuit object

Description

Use `circuit` object to build a circuit object which can contain elements like resistor, capacitor, and inductor.

Creation

Syntax

```
cktobj = circuit
cktobj = circuit(cktname)

cktobj = circuit([elem1,...,elemN])
cktobj = circuit([elem1,...,elemN],cktname)

cktobj = circuit(rfb)
cktobj = circuit(rfb,cktname)
```

Description

`cktobj = circuit` creates a circuit object `cktobj` with a default name.

`cktobj = circuit(cktname)` creates a circuit object `cktobj` with name of `cktname`.

`cktobj = circuit([elem1,...,elemN])` creates a circuit object `cktobj` by cascading the specified 2-port elements.

`cktobj = circuit([elem1,...,elemN],cktname)` creates a cascaded circuit object `cktobj` with the name, `cktname`.

`cktobj = circuit(rfb)` creates a circuit object `cktobj` by cascading the elements in the RF object, `rfb`.

`cktobj = circuit(rfb,cktname)` creates a circuit object `cktobj` by cascading the elements in the RF object, `rfb`, using name, `cktname`.

Input Arguments

`elem1,...,elemN` — 2-port RF elements
character vector

2-port RF elements, specified as character vectors. The possible elements are `amplifier`, `nport`, and `modulator`

`rfb` — RF budget object
object handle

RF budget object, specified as an object handle.

Properties

Name — Object Name

'unnamed' (default) | character vector

Name of circuit, specified as a character vector. Default name is 'unnamed'. Two circuit elements attached together or belonging to the same circuit cannot have the same name

Data Types: char | string

Elements — Heterogeneous array of elements in circuit

resistor object | capacitor object | inductor object | lcladder object | nport object | modulator object | rffilter object | amplifier object

Heterogeneous array of elements present in the circuit, specified as any one of the following objects: amplifier, resistor, capacitor, inductor, lcladder, nport, modulator, and rffilter objects.

Data Types: char | string

ElementNames — Name of elements in the circuit

cell vector

Name of elements in the circuit, specified as a vector of cell vector. The possible elements here are resistor, capacitor, inductor, and circuit.

Data Types: char | string

Terminals — Names of terminals in the circuit

cell vector

Names of terminals in the circuit, specified as a cell vector. Use setports or setterminals function to define the terminals. The terminals of the circuit are only displayed once it is defined.

Data Types: char | string

Ports — Names of ports in a circuit

character vector

Names of ports in a circuit specified as a character vector. Use setports function to define the ports. The ports of the circuit are only displayed once it is defined.

Data Types: char | string

Nodes — List of nodes defined in circuit

vector of integers

List of nodes defined in the circuit, specified as a vector of integers. These nodes are created when a new element is attached to the circuit.

Data Types: double

ParentPath — Full path of parent circuit

character vector

Full path of parent circuit, specified as a character vector. This path appears only once the child circuit is added to the parent circuit.

Data Types: `char` | `string`

`ParentNodes` — Nodes of parent circuit
vector of integers.

Nodes of parent circuit, specified as a vector of integers. This vector of integers is the same length as the `Terminals` property. This property is read-only and appears only after the child circuit is added to the parent circuit.

Data Types: `double`

Object Functions

sparameters	Calculate S-parameters for RF data, network, circuit, and matching network objects
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object
add	Insert circuit element or circuit object into circuit
delete	Delete circuit object and decouple its elements
setports	Set ports of circuit object
setterminals	Set terminals of circuit object
clone	Create copy of existing circuit element or circuit object
richards	Convert lumped element circuit to distributed element circuit using Richards' transformation

Examples

Create Circuit with Elements and Terminals

Create a circuit called new_circuit. Add a resistor and capacitor to the circuit. Set the terminals and display the results.

```
hckt = circuit('new_circuit1');
hC1= add(hckt,[1 2],capacitor(3e-9));
hR1 = add(hckt,[2 3],resistor(100));
setterminals (hckt,[1 3]);
disp(hckt)
```

```
  circuit: Circuit element

    ElementNames: {'C'  'R'}
        Elements: [1x2 rf.internal.circuit.RLC]
           Nodes: [1 2 3]
            Name: 'new_circuit1'
       Terminals: {'t1'  't2'}
```

Create Circuit with Two Parallel Elements

Create a circuit called new_circuit. Add a capacitor and inductor parallel to the circuit.

```
hckt = circuit('new_circuit');
hC = add(hckt,[1 2],capacitor(1e-12));
```

```
hL = add(hckt,[1 2],inductor(1e-9));
disp(hckt)
```

```
  circuit: Circuit element

    ElementNames: {'C'  'L'}
        Elements: [1x2 rf.internal.circuit.RLC]
           Nodes: [1 2]
            Name: 'new_circuit'
```

Version History

Introduced in R2013b

See Also

`resistor` | `capacitor` | `inductor` | `add`

Topics

“Bandpass Filter Response”
“MOS Interconnect and Crosstalk”

rfelement

Generic RF element object

Description

Use the `rfelement` object to create a generic RF element. An RF element is a 2–port RF circuit object. You can use this element in the `rfbudget` object and the `circuit` object.

Creation

Syntax

```
rfel = rfelement
rfel = rfelement(Name,Value)
```

Description

`rfel = rfelement` creates an RF element object with default property values.

`rfel = rfelement(Name,Value)` sets properties using one or more name-value pairs. You can specify multiple name-value pairs. Enclose each property name in a quote.

Properties

`Name` — Name given to identify RF element

`'RFelement'` (default) | character vector

Name given to identify rf element, specified as a character vector. All names must be valid MATLAB variable names.

Example: `'Name','rfel'`

Example: `rfel.Name = 'rfel'`

`Gain` — Available power gain

`0` (default) | scalar

Available power gain, specified as a scalar in dB.

Example: `'Gain',10`

Example: `rfel.Gain = 10`

`NF` — Noise figure

`0` (default) | real finite nonnegative scalar

Noise figure, specified as a real finite nonnegative scalar dB.

Example: `'NF',-10`

Example: `rfel.NF = -10`

`OIP2` — Second -order output-referred intercept point
`Inf` (default) | real scalar

Second -order output-referred intercept point, specified as a real scalar in dBm.

Example: `'OIP2',8`

Example: `amplifier.OIP2 = 8`

`OIP3` — Third -order output-referred intercept point
`Inf` (default) | real scalar

Third -order output-referred intercept point, specified as a real scalar in dBm.

Example: `'OIP3',10`

Example: `amplifier.OIP3 = 10`

`Zin` — Input impedance
`50` (default) | positive real part finite scalar

Input impedance, specified as a positive real part finite scalar in Ohms. You can also use a complex value with a positive real part.

Example: `'Zin',40`

Example: `rfel.Zin = 40`

`Zout` — Output impedance
`50` (default) | positive real part finite scalar

Output impedance, specified as a scalar in Ohms. You can also use a complex value with a positive real part.

Example: `'Zout',40`

Example: `rfel.Zout = 40`

`NumPorts` — Number of ports
2 (default) | scalar integer

Number of ports, specified as a scalar integer. This property is read-only.

`'Terminals'` — Names of port terminals
`{'p1+' 'p2+' 'p1-' 'p2-'}` (default) | cell vector

Names of port terminals, specified as a cell vector. This property is read-only.

Object Functions

clone Create copy of existing circuit element or circuit object

Examples

RF Element

Create an rfelement object with a gain of 10 dB, noise figure of 3 dB, and OIP3 (output third-order intercept) of 2 dBm.

```
r = rfelement('Gain',10,'NF',3,'OIP3',2)

r =
  rfelement: RF element

         Name: 'RFelement'
         Gain: 10
           NF: 3
         OIP2: Inf
         OIP3: 2
          Zin: 50
         Zout: 50
     NumPorts: 2
    Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

RF Element Circuit

Create an rf element with a gain of 4 dB. Create another rf element with an output third-order intercept(OIP3) of 13 dBm.

```
rfel1 = rfelement('Gain',4);
rfel2 = rfelement('OIP3',13);
```

Build a 2-port circuit using the above defined rf elements.

```
c = circuit([rfel1 rfel2])

c =
  circuit: Circuit element

    ElementNames: {'RFelement'  'RFelement_1'}
        Elements: [1x2 rfelement]
           Nodes: [0 1 2 3]
            Name: 'unnamed'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

RF Budget Analysis of Series of RF Elements

Create an amplifier with a gain of 4 dB.

```
a = amplifier(Gain=4);
```

Create a modulator with an OIP3 of 13 dBm.

```
m = modulator(OIP3=13);
```

Create an N-port element using `passive.s2p`.

```
n = nport('passive.s2p');
```

Create an RF element with a gain of 10 dB.

```
r = rfelement(Gain=10);
```

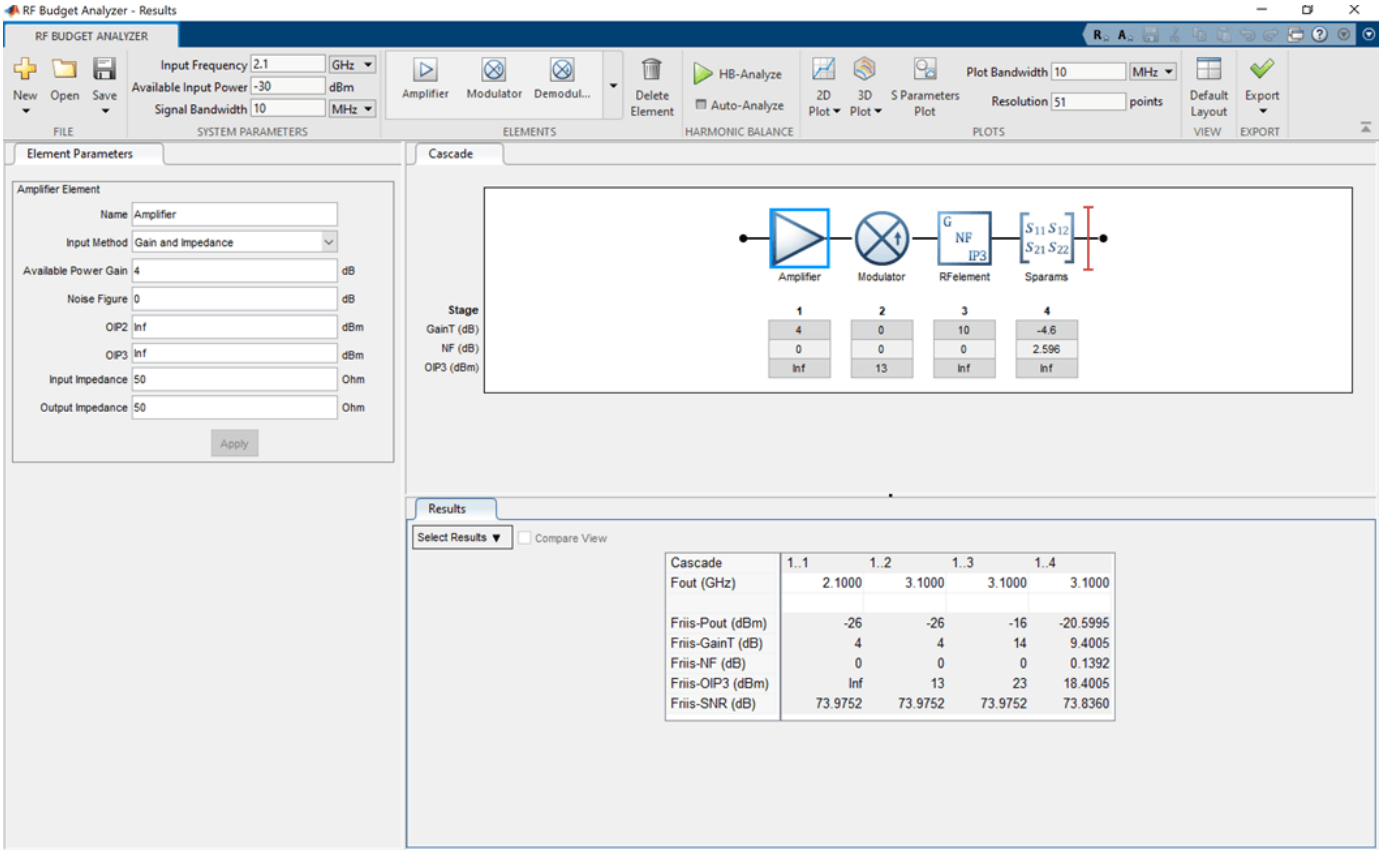
Calculate the RF budget of a series of RF elements at an input frequency of 2.1 GHz, an available input power of –30 dBm, and a bandwidth of 10 MHz.

```
b = rfbudget([a m r n],2.1e9,-30,10e6)
```

```
b =
  rfbudget with properties:

              Elements: [1x4 rf.internal.rfbudget.Element]
          InputFrequency: 2.1 GHz
     AvailableInputPower: -30 dBm
         SignalBandwidth:  10 MHz
                  Solver: Friis
              AutoUpdate: true

   Analysis Results
         OutputFrequency: (GHz) [  2.1    3.1    3.1     3.1]
             OutputPower: (dBm) [  -26    -26    -16   -20.6]
          TransducerGain: (dB)  [    4      4     14     9.4]
                      NF: (dB)  [    0      0      0  0.1392]
                    IIP2: (dBm) []
                    OIP2: (dBm) []
                    IIP3: (dBm) [  Inf      9      9       9]
                    OIP3: (dBm) [  Inf     13     23    18.4]
                     SNR: (dB)  [73.98  73.98  73.98   73.84]
```

Type the `show` command at the command window to display the analysis in the **RF Budget Analyzer** app.

```
show(b)
```



Version History

Introduced in R2017a

See Also

`nport` | `modulator` | `amplifier` | `rfbudget`

OpenIF

Find open intermediate frequencies (IFs) in multiband transmitter or receiver architecture

Description

Use the `OpenIF` object to analyze the spurs and spur-free zones in a multiband transmitter or receiver. This information helps you determine intermediate frequencies (IFs) that do not produce interference in operating bands.

Creation

Syntax

```
hif = OpenIF
hif = OpenIF(Name,Value)
hif = OpenIF(bandwidth)
hif = OpenIF(bandwidth,Name,Value)
```

Description

`hif = OpenIF` creates an intermediate-frequency (IF) planning object with properties set to their default values.

`hif = OpenIF(Name,Value)` creates an intermediate-frequency (IF) planning object with properties with additional options specified by one or more `Name,Value` pair arguments.

`hif = OpenIF(bandwidth)` creates an intermediate-frequency (IF) planning object with a specified IF bandwidth.

`hif = OpenIF(bandwidth,Name,Value)` creates an IF-planning object with a specified IF bandwidth and additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

`bandwidth` — Bandwidth of IF signal

real positive scalar

Bandwidth of IF signal, specified as a real positive scalar. The value you provide sets the `IFBW` property of your object.

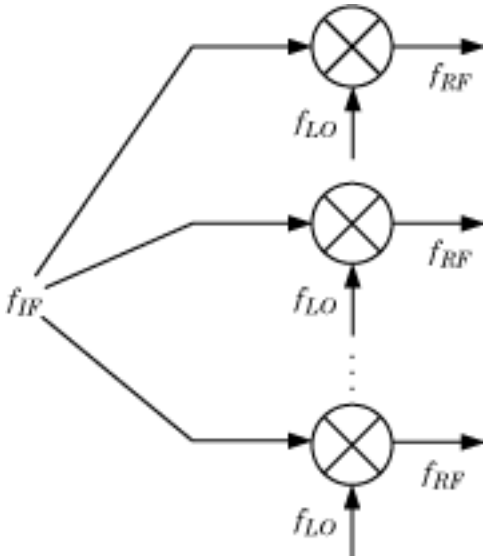
Properties

`IF Location` — Location of IF

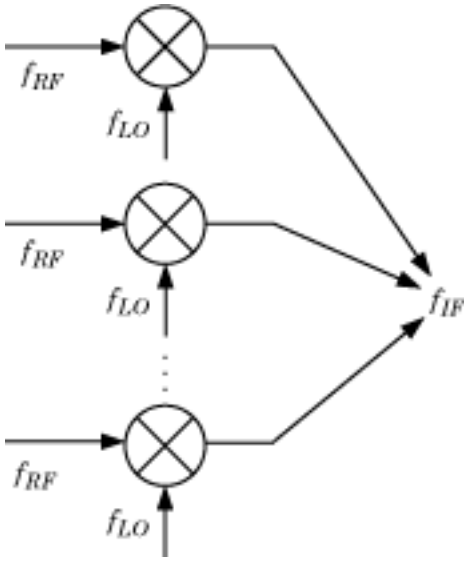
`'MixerOutput'` (default) | `'MixerInput'`

Location of IF, specified as a `'MixerOutput'` or `'MixerInput'`.

- Setting `IFLocation` to `'MixerInput'` specifies an up-converting (transmitting) configuration, where one IF is mixed up to multiple RFs. The following figure shows this convention.



- Setting `IFLocation` to `'MixerOutput'` specifies a down-converting (receiving) configuration, where multiple RFs are mixed down to one IF. The following figure shows this convention.



The setting of `IFLocation` determines the available values for the `injection` argument of the `addMixer` function.

Example: `'IFLocation','MixerInput'`

Example: `amplifier.IFLocation = 'MixerInput'`

`SpurFloor` — Maximum spur value
99 (default) | scalar

Maximum difference in magnitude between a signal at 0 dBc and an intermodulation product that the `OpenIF` object considers a spur, specified as a scalar in dBc.

Example: `'SpurFloor',80`

Example: `amplifier.SpurFloor = 80`

`IFBW` — System wide IF bandwidth
99 (default) | scalar

System wide IF bandwidth, specified as a scalar in hertz. You can also set this property using the optional `bandwidth` input argument.

Example: `'IFBW',80`

Example: `amplifier.IFBW = 80`

Examples

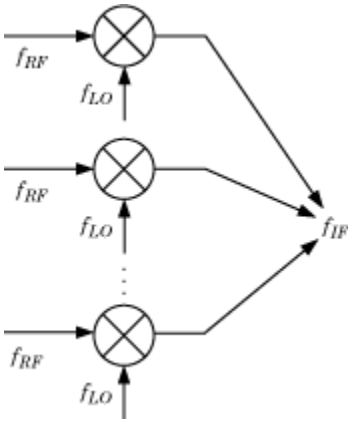
Spur-Free Zones of Multiband Receiver

Set up an `OpenIF` object as a multiband receiver, add three mixers to it, and obtain information about its spur-free zones.

Define an `OpenIF` object. The first input is the bandwidth of the IF signal (50 MHz). The `'IFLocation','MixerOutput'` name-value pair specifies a down-converting configuration.

```
hif = OpenIF(50e6,'IFLocation','MixerOutput');
```

The following figure shows the down-converting configuration.



Define the first mixer with an intermodulation table and add it to the `OpenIF` object. Mixer 1 has a RF center frequency at 2.4 GHz, has a RF bandwidth of 100 MHz, and uses low-side injection.

```
IMT1 = [99 00 21 17 26; ...
        11 00 29 29 63; ...
        60 48 70 65 41; ...
        90 89 74 68 87; ...
        99 99 95 99 99];
addMixer(hif,IMT1,2.4e9,100e6,'low');
```

Mixer 2 has an RF center frequency at 3.7 GHz, has a bandwidth of 150 MHz, and uses low-side injection.

```
IMT2 = [99 00 09 12 15; ...
        20 00 26 31 48; ...
        55 70 51 70 53; ...
        85 90 60 70 94; ...
        96 95 94 93 92];
addMixer(hif,IMT2,3.7e9,150e6,'low');
```

Mixer 3 has an RF center frequency at 5 GHz, has a bandwidth of 200 MHz, and uses low-side injection.

```
IMT3 = [99 00 15 23 36; ...
        10 00 34 27 59; ...
        67 61 56 59 68; ...
        97 82 81 60 77; ...
        99 99 99 99 96];
addMixer(hif,IMT3,5e9,200e6,'low');
```

The multiband receiver is fully defined and ready for spur-free-zone analysis. Use the `report` method to analyze and display spur and spur-free zone information at the command line. The method also returns information about the mixers in the receiver.

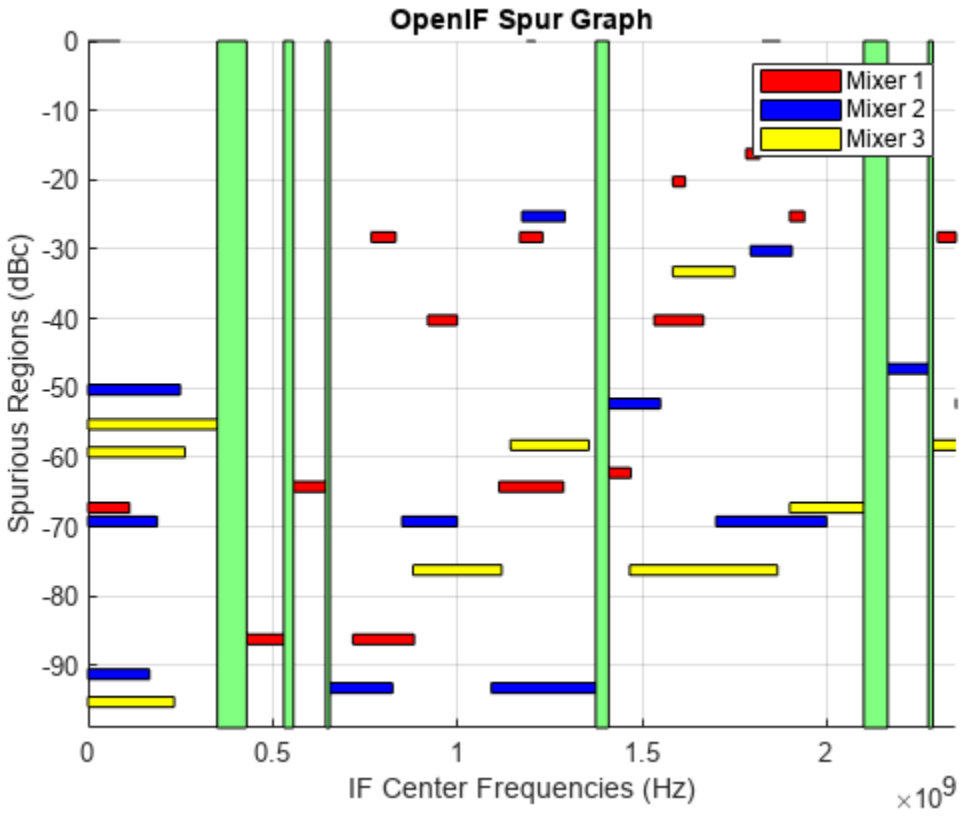
```
hif.report
```

```
     Intermediate Frequency (IF) Planner
     IF Location: MixerOutput

     -- MIXER 1 --
     RF Center Frequency: 2.4 GHz
     RF Bandwidth: 100 MHz
     IF Bandwidth: 50 MHz
     MixerType: low
     Intermodulation Table:   99   0  21  17  26
                              11   0  29  29  63
                              60  48  70  65  41
                              90  89  74  68  87
                              99  99  95  99  99

     -- MIXER 2 --
     RF Center Frequency: 3.7 GHz
     RF Bandwidth: 150 MHz
     IF Bandwidth: 50 MHz
     MixerType: low
     Intermodulation Table:   99   0   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 GHz
     RF Bandwidth: 200 MHz
     IF Bandwidth: 50 MHz
     MixerType: low
     Intermodulation Table:   99   0  15  23  36
                              10   0  34  27  59
                              67  61  56  59  68
                              97  82  81  60  77
                              99  99  99  99  96
```

```
Spur-Free Zones:
 350.00 -  430.00 MHz
 530.00 -  556.25 MHz
 643.75 -  655.00 MHz
   1.38 -    1.41 GHz
   2.10 -    2.17 GHz
   2.28 -    2.29 GHz
```

Use the `show` method to analyze the receiver and produce an interactive spur graph. Generating a spur graph is a convenient way to summarize the results of the analysis graphically.

```
figure;
hif.show
```



Version History

Introduced before R2006a

See Also

`addMixer` | `show`

Topics
“Finding Free IF Bandwidths”

rffilter

Create RF filter object

Description

Use the `rffilter` object to create a Butterworth, Chebyshev or an Inverse Chebyshev RF filter. The RF filter is a 2-port circuit object, and you can include this object as an element of a circuit.

For more design information see, “Parameters to Define Filter and Design Tips” on page 1-229.

You can also convert the `rffilter` object to an lcladder by using the `lcladder` object. `LCLad = lcladdder(rffiltobj)` where `rffilterobj` is an `rffilter` object.

Creation

Syntax

```
rffiltobj = rffilter
rffiltobj = rffilter(Name,Value)
```

Description

`rffiltobj = rffilter` creates a 2-port filter with default properties.

`rffiltobj = rffilter(Name,Value)` sets properties using one or more name-value pairs. For example, `rffiltobj = rffilter('FilterType','Chebyshev')` creates a 2-port Chebyshev RF filter. You can specify multiple name-value pairs. Enclose each property name in a quote.

Properties

FilterType — Filter type

`'Butterworth'` (default) | `'Chebyshev'` | `'InverseChebyshev'`

Filter type, specified as `'Butterworth'`, `'Chebyshev'`, or `'InverseChebyshev'`.

Example: `'FilterType','Chebyshev'`

Example: `rfobj.FilterType = 'Chebyshev'`

Data Types: `char` | `string`

ResponseType — Filter response type

`'Lowpass'` (default) | `'Highpass'` | `'Bandpass'` | `'Bandstop'`

Filter response type, specified as `'Lowpass'`, `'Highpass'`, `'Bandpass'`, or `'Bandstop'`. For more information, see “Frequency Responses” on page 1-227.

Example: `'ResponseType','Highpass'`

Example: `rfobj.ResponseType = 'Highpass'`

Data Types: `char` | `string`

`Implementation` — Filter implementation

`'LC Tee'` (default) | `'LC Pi'` | `'Transfer function'`

Filter implementation, specified as `'LC Tee'`, `'LC Pi'`, or `'Transfer function'`.

Example: `'Implementation','Transfer function'`

Example: `rfobj.Implementation = 'Transfer function'`

Dependencies

For `'Inverse Chebyshev'` type filter, you can only use `'Transfer function'` implementation.

Data Types: `char` | `string`

`FilterOrder` — Filter order

3 (default) | real finite non-negative integer

Filter order, specified as a real finite non-negative integer. In a lowpass or highpass filter, the order specifies the number of lumped storage elements. In a bandpass or bandstop filter, the number of lumped storage elements is twice the value of the order.

Note `FilterOrder` has the highest precedence among all the name-value pairs in the filter design. Using this property sets the `UseFilterOrder` read-only property to true.

Example: `'FilterOrder',4`

Example: `rfobj.FilterOrder = 4`

Data Types: `double`

`PassbandFrequency` — Passband frequency

scalar | two-element vector

Passband frequency, specified as:

- A scalar in hertz for lowpass and highpass filters.
- A two-element vector in hertz for bandpass or bandstop filters.

By default, the values are `1e9` for lowpass filter, `2e9` for highpass filter, and `[2e9 3e9]` for bandpass and `[[1e9 4e9]` for bandstop filters.

Example: `'PassbandFrequency',[3e6 5e6]`

Example: `rfobj.PassbandFrequency = [3e6 5e6]`

Data Types: `double`

`StopbandFrequency` — Stopband frequency

scalar | two-element vector

Stopband frequency, specified as:

- A scalar in hertz for lowpass and highpass filters.

- A two-element vector in hertz for bandpass or bandstop filters.

By default, the values are `2e9` for lowpass filter, `1e9` for highpass filter, `[1.5e9 3.5e9]` for bandpass filters, and `[2.1e9 2.9e9]` bandstop filters.

Example: `rffilter('ResponseType','lowpass','StopbandFrequency',[3e6 5e6])`

Example: `rfobj.StopbandFrequency = [3e6 5e6]`

Data Types: `double`

`PassbandAttenuation` — Passband attenuation
`10*log10(2)` (default) | scalar

Passband attenuation, specified as a scalar in dB. For bandpass filters, this value is applied equally to both edges of the passband.

Example: `'PassbandAttenuation',5`

Example: `rfobj.PassbandAttenuation = 5`

Data Types: `double`

`StopbandAttenuation` — Stopband attenuation
`40` (default) | scalar

Stopband attenuation, specified as a scalar in dB. For bandstop filters, this value is applied equally to both edges of the stopband.

Example: `'StopbandAttenuation',30`

Example: `rfobj.StopbandAttenuation = 30`

Data Types: `double`

`Zin` — Source impedance
`50` (default) | positive real part finite scalar

Source impedance, specified as a positive real part finite scalar in ohms.

Example: `'Zin',70`

Example: `rfobj.Zin = 70`

Data Types: `double`

`Zout` — Load impedance
`50` (default) | positive real part finite scalar

Load impedance, specified as a positive real part finite scalar in ohms.

Example: `'Zout',70`

Example: `rfobj.Zout = 70`

Data Types: `double`

`Name` — Name of RF filter object
`'Filter'` (default) | character vector

Name of RF filter object, specified as a character vector. Two elements in the same circuit cannot have the same name. All names must be valid MATLAB variable names.

Example: `'Name','filter1'`

Example: `rfobj.Name = 'filter1'`

Data Types: `char` | `string`

`NumPorts` — Number of ports
`2`

Number of ports, specified as a `2`. This property is read-only.

Data Types: `double`

`Terminals` — Names of terminals
`{'p1+','p2+','p1-','p2-'}`

Names of the terminals, specified as a {'p1+','p2+','p1-','p2-'}. This property is read-only.

Data Types: `char`

`DesignData` — Filter design data
structure

Filter design data, specified as a structure. This property is read-only. For more information, see “Design Data for LC Tee and LC Pi Topologies” on page 1-225 and “Design Data for Transfer Function Implementation” on page 1-225.

Data Types: `struct`

`UseFilterOrder` — Use of filter order for filter design
`true` | `false`

Use of filter order for filter design, specified as a `true` or `false`. This property is a read-only.

Data Types: `logical`

Object Functions

groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object
sparameters	Calculate S-parameters for RF data, network, circuit, and matching network objects
set	Set rffilter object property values
zpk	Converts rffilter to zero-pole-gain representation
tf	Converts rffilter to transfer function
lcladder	Create LC ladder network
rfplot	Plot input reflection coefficient and transducer gain of matching network
clone	Create copy of existing circuit element or circuit object
circuit	Circuit object

Examples

Default RF Filter

Create and view the properties of a default RF filter object.

```
rfobj = rffilter
```

```
rfobj =
  rffilter: Filter element

             FilterType: 'Butterworth'
           ResponseType: 'Lowpass'
         Implementation: 'LC Tee'
            FilterOrder: 3
      PassbandFrequency: 1.0000e+09
    PassbandAttenuation: 3.0103
                    Zin: 50
                   Zout: 50
             DesignData: [1x1 struct]
         UseFilterOrder: 1
                   Name: 'Filter'
               NumPorts: 2
              Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

```
rfobj.DesignData
```

```
ans = struct with fields:
             FilterOrder: 3
               Inductors: [7.9577e-09 7.9577e-09]
              Capacitors: 6.3662e-12
                Topology: 'lclowpasstee'
      PassbandFrequency: 1.0000e+09
    PassbandAttenuation: 3.0103
```

S-Parameters of Butterworth passband filter (LC Tee Implementation Type)

Create a Butterworth passband filter object named `BFCG_162W` with passband frequencies between 950 and 2200 MHz, stopband frequencies between 770 and 3000 MHz, passband attenuation of 3.0 dB, and stopband attenuation of 40 dB using 'LC Tee' implementation type. Calculate the S-parameters of the filter at 2.1 GHz.

```
robj = rffilter('ResponseType','Bandpass','Implementation','LC Tee','PassbandFrequency',[950e6 22
    'StopbandFrequency',[770e6 3000e6],'PassbandAttenuation',3,'StopbandAttenuation',40);
robj.Name = 'BFCG_162W'
```

```
robj =
  rffilter: Filter element

             FilterType: 'Butterworth'
           ResponseType: 'Bandpass'
         Implementation: 'LC Tee'
      PassbandFrequency: [950000000 2.2000e+09]
    PassbandAttenuation: 3
      StopbandFrequency: [770000000 3.0000e+09]
    StopbandAttenuation: 40
                    Zin: 50
                   Zout: 50
             DesignData: [1x1 struct]
         UseFilterOrder: 0
                   Name: 'BFCG_162W'
```

```
               NumPorts: 2
              Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Calculate the S-parameters at 2.1 GHz.

```
s = sparameters(robj,2.1e9)
```

```
s =
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: 2.1000e+09
     Parameters: [2x2 double]
      Impedance: 50

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

Build a `lcladder` object from the `rffilter` object. This lcladder object can be used in a circuit directly and could also be used for parametric analysis across inductor and capacitance values.

```
l = lcladder(robj)
```

```
l =
  lcladder: LC Ladder element

        Topology: 'bandpasstee'
     Inductances: [1.8116e-09 5.7297e-09 8.3361e-09 2.8294e-09 ... ]
    Capacitances: [6.6900e-12 2.1152e-12 1.4539e-12 4.2835e-12 ... ]
            Name: 'lcfilt'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Alternatively, to access the inductors and capacitors directly from the filter object use:

```
L = robj.DesignData.Inductors;
C = robj.DesignData.Capacitors;
```

Group Delay of Chebyshev Lowpass Filter

Create a Chebyshev lowpass filter with a passband frequency of 2 GHz.

```
robj = rffilter('FilterType','Chebyshev','PassbandFrequency',2e9);
```

Set the filter order to 5 and the implementation to `LC Pi`.

```
set(robj,'FilterOrder',5,'Implementation','LC Pi');
```

Calculate the group delay of the filter at 1.9 GHz.

```
groupdelay(robj,1.9e9)
```

```
ans = 1.4403e-09
```

Design Butterworth Filter and Determine Filter Order

This example shows how to design a low-pass Butterworth filter with passband frequency of 3 kHz, stopband frequency 7 kHz, passband attenuation of 2 dB, and stopband attenuation 60 dB. Display the filter order of such a designed filter and determine the passband frequency at 3.0103 dB. See [2] in `rffilter` object page.

Filter Parameters

```
Fp = 3e3;           % Passband frequency, Hz
Ap = 2;             % Passband attenuation, dB
Fs = 7e3;           % Stopband frequency, Hz
As = 60;            % Stopband attenuation, dB
```

Design Filter

```
r = rffilter("FilterType","Butterworth","ResponseType","Lowpass","Implementation","Transfer funct
    "PassbandAttenuation",Ap,"StopbandFrequency",Fs,"StopbandAttenuation",As);
```

Filter Order of Designed Filter

```
N = r.DesignData.FilterOrder;
sprintf('Calculated filter order is %d',N)
```

```
ans =
'Calculated filter order is 9'
```

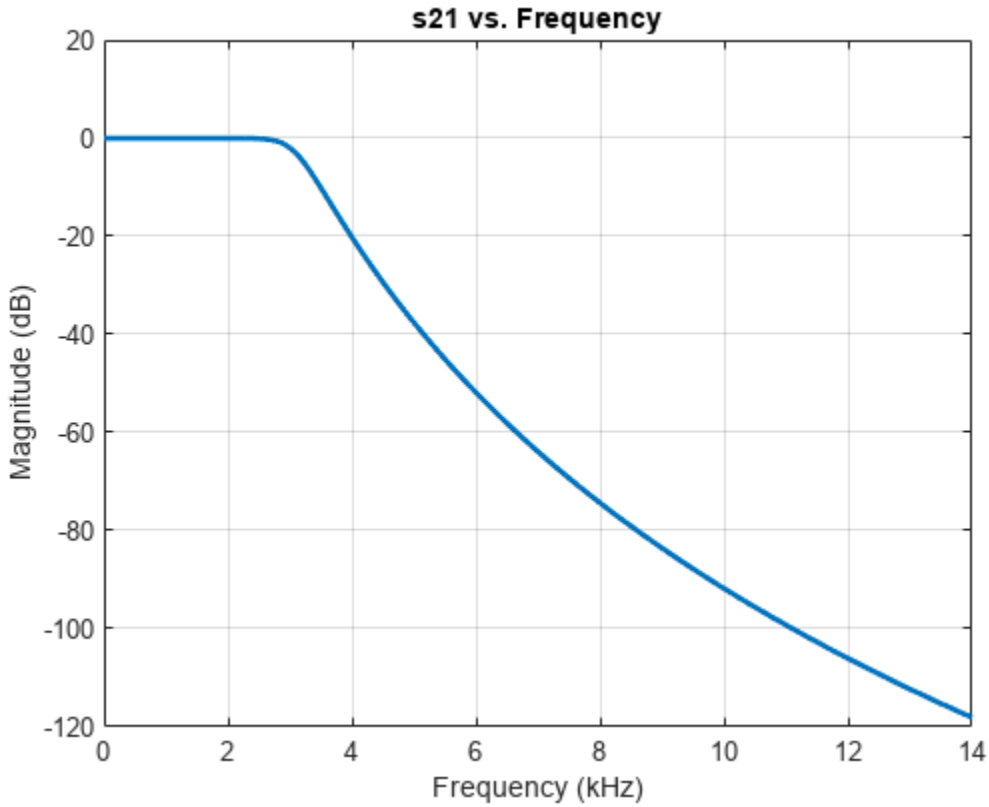
Frequency at 3.0103 dB

```
F_3dB = r.DesignData.PassbandFrequency/1e3;
sprintf('Frequency at 3.0103 dB is %d kHz',F_3dB)
```

```
ans =
'Frequency at 3.0103 dB is 3.090733e+00 kHz'
```

Visualize Magnitude Response

```
frequencies = linspace(0,2*Fs,1001);
rfplot(r, frequencies)
```



Note: To use `rfplot` and `plot` on the same figure use `setrfplot`. Type `'help setrfplot'` in command window for information.

Reference

1 Larry D. Paarmann, Design and Analysis of Analog Filters: A Signal Processing Perspective, Kluwer Academic Publishers

Design Chebyshev Filter and Determine Filter Order

Design a low-pass Chebyshev filter with 0.1 dB bandpass ripple, cut-off frequency of 1 rad/sec, and 50 dB attenuation at 1.1 rad/sec. Display the filter order of this designed filter [1].

Define Parameters

```
Fp = 1/(2*pi);          % Passband frequency, Hz
Rp = 0.1;               % Ripple in Passband, dB
Fs = 1.1/(2*pi);        % Stopband frequency, Hz
As = 50;                % Stopband attenuation, dB
```

Design Filter

```
r = rffilter("FilterType","Chebyshev","ResponseType","Lowpass","Implementation","Transfer functio
    "PassbandAttenuation",Rp,"StopbandFrequency",Fs,"StopbandAttenuation",As)
```

```
r =
  rffilter: Filter element
```

```
            FilterType: 'Chebyshev'
          ResponseType: 'Lowpass'
        Implementation: 'Transfer function'
      PassbandFrequency: 0.1592
    PassbandAttenuation: 0.1000
      StopbandFrequency: 0.1751
    StopbandAttenuation: 50
                   Zin: 50
                  Zout: 50
            DesignData: [1x1 struct]
         UseFilterOrder: 0
                   Name: 'Filter'
               NumPorts: 2
              Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Filter Order of Designed Filter

```
N = r.DesignData.FilterOrder;
sprintf('Calculated filter order is %d',N)

ans =
'Calculated filter order is 19'
```

Reference

1 G.Ellis, Michael,Sr.Electronic Filter Analysis and Synthesis,Artech House, 1994

Frequency Response of Even Order Chebyshev Filter

Design even order Chebyshev filter and plot the filter's frequency response.

Frequency Response of Even Order Chebyshev Filter using Transfer Function Implementation

Design even order Chebyshev filter with specified parameters.

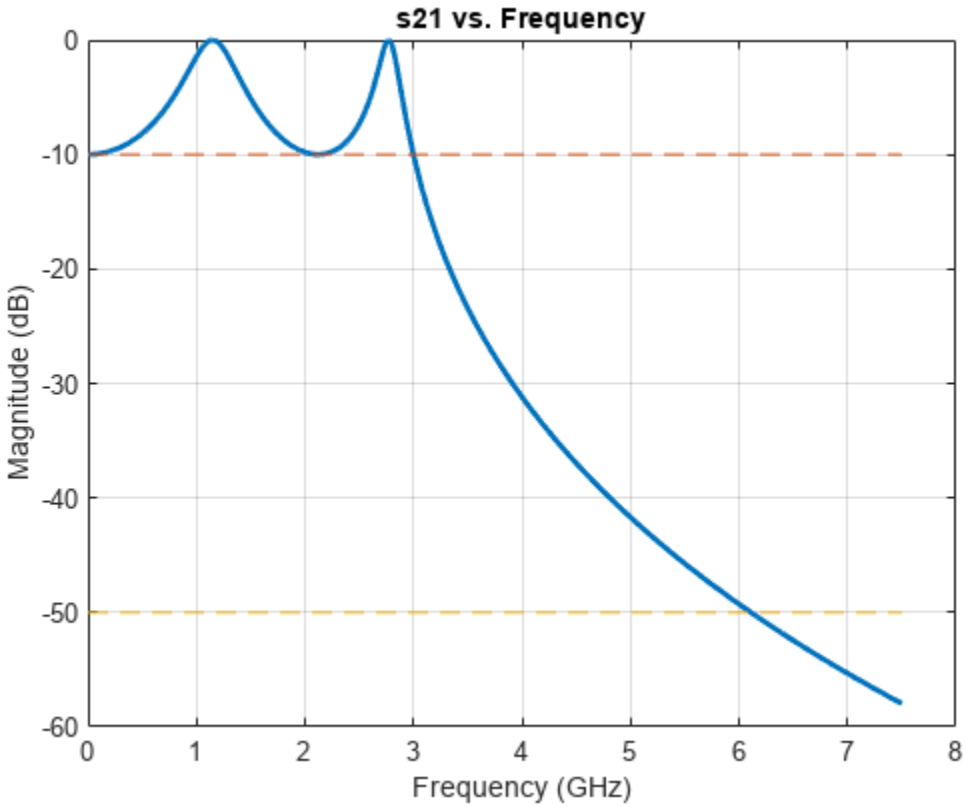
```
Fp = 3e9;          % Passband frequency, GHz
Rp = 10;           % Passband attenuation, dB
Fs = 7e9;          % Stopband frequency, GHz
As = 50;           % Stopband attenuation, dB
rffiltobj = rffilter("FilterType","Chebyshev","ResponseType","Lowpass","Implementation","Transfe
    "PassbandAttenuation",Rp,"StopbandAttenuation",As, "StopbandFrequency",Fs);
```

Plot the frequency response of even order Chebyshev filter using `rfplot` function.

```
rfplot(rffiltobj,linspace(0,7.5e9,1001))
```

Visualize the stopband and passband attenuation using `plot` function.

```
hold on;
plot([0 7.5], repmat([-rffiltobj.PassbandAttenuation -rffiltobj.StopbandAttenuation],2,1),'--')
```



Use `DesignData` parameter to verify the order of your filter.

```
N = rffiltobj.DesignData.FilterOrder;
sprintf('Calculated filter order is %d',N)
```

```
ans =
'Calculated filter order is 4'
```

More About

Design Data for LC Tee and LC Pi Topologies

For LC Tee or Pi topologies, `DesignData` returns inductor and capacitor values. In addition, `DesignData` includes other design parameters relevant to response type.

- Lowpass/Highpass Response: Filter order, Passband Frequency, Passband Attenuation
- Bandpass Response: Filter order, Passband Frequency, Passband Attenuation, Auxiliary (W_x).
- Bandstop Response: Filter order, Stopband Frequency, Passband Attenuation, Auxiliary (W_x).

For bandstop response, W_x is an adjustment for the first frequency at which the lowpass prototype meets the prescribed bandstop loss. For bandpass response, W_x is an adjustment of specification of passband attenuation not equal to 3 dB. For more information, see [1].

Design Data for Transfer Function Implementation

For transfer function implementation, `DesignData` returns factored polynomial coefficients for S-parameters. These factors group the complex conjugate terms to preserve precision. All S-parameters

have a common denominator present in `Denominator`. The numerator terms for S_{11}, S_{22}, and S_{21} (S_{21} = S_{12}) can be evaluated using the factored polynomial present in numerators `Numerator11`, `Numerator22`, and `Numerator21`, respectively.

For example, consider a default lowpass filter at 1 GHz. You can find the S_{21} data at 1 GHz for the filter as follows:

```
r = rffilter('Implementation','Transfer function');
f = 1e9;
num21 = [polyval(r.DesignData.Numerator21(1,:),1i*2*pi*f) ...
        polyval(r.DesignData.Numerator21(2,:),1i*2*pi*f)];
den = [polyval(r.DesignData.Denominator(1,:),1i*2*pi*f) ...
      polyval(r.DesignData.Denominator(2,:),1i*2*pi*f)];
s21_1GHz = prod(num21./den,2)
```

```
s21_1GHz =

  -0.5000 - 0.5000i
```

Alternatively, you can use the `sparameters` function to calculate the example:

```
S = sparameters(r,1e9);
S.Parameters(2,1)
```

```
ans =

  -0.5000 - 0.5000i
```

In addition, `DesignData` includes other design parameters relevant to response type for:

- Lowpass/Highpass Response: Filter order, Passband Frequency, Auxiliary (Numerator21 Polynomial)

  Note Passband frequency is at 3 dB for Butterworth filter.

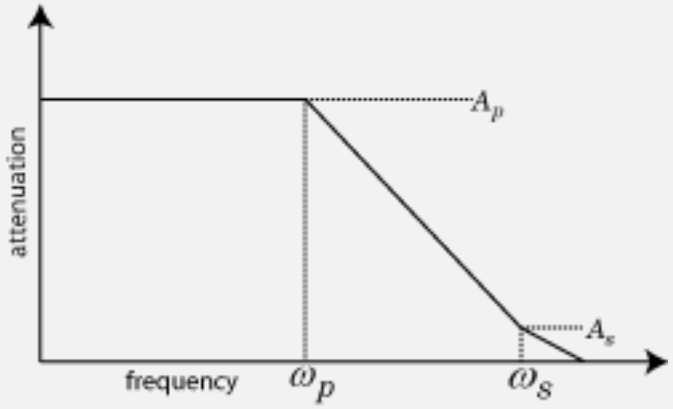
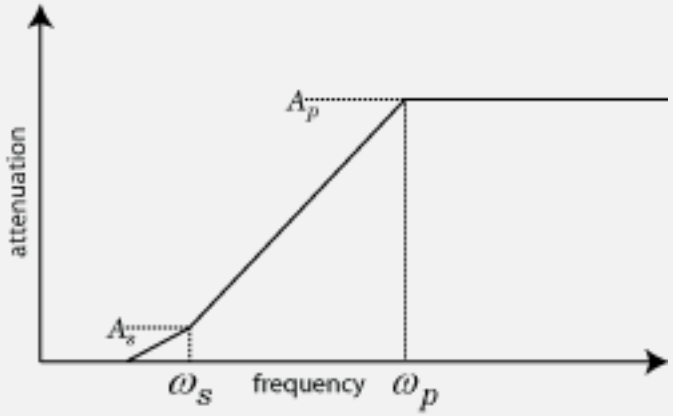
- Bandpass Response: Filter order, Passband Frequency, Auxiliary (W_x, Numerator21 Polynomial)
- Bandstop Response: Filter order, Stopband Frequency, Auxiliary (W_x, Numerator21 Polynomial)

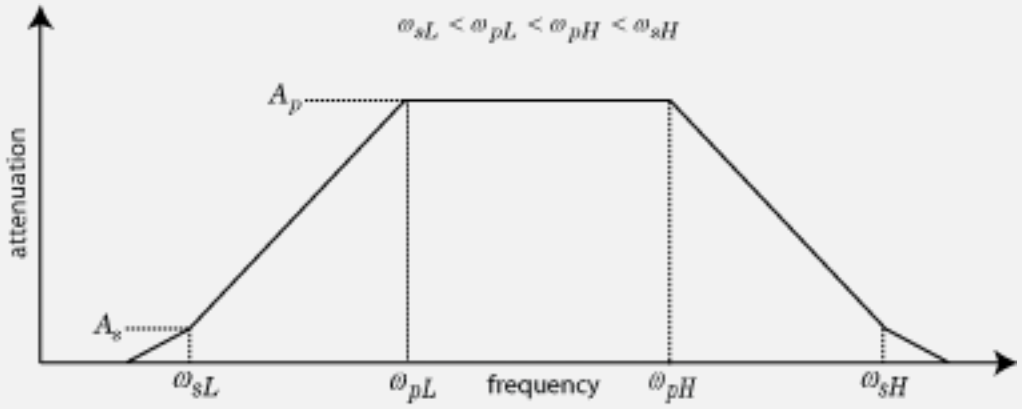
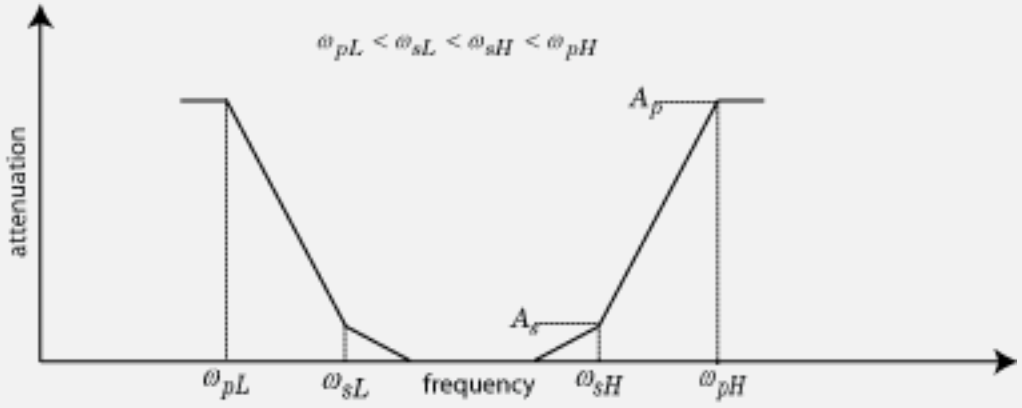
For bandstop response, W_x is an adjustment for the first frequency at which the lowpass prototype meets the prescribed bandstop loss. For bandpass response, W_x is an adjustment of specification of passband attenuation not equal to 3 dB.

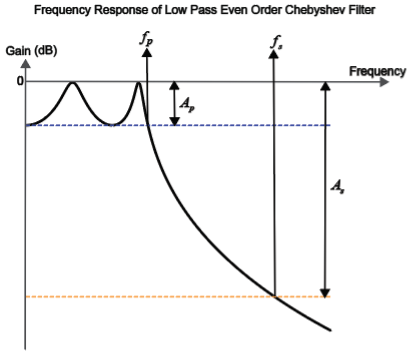
Some additional design tips:

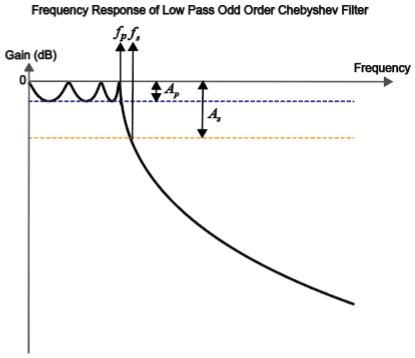
<table>
<tr><th></th><th colspan="2">Low-pass</th><th>High-pass</th><th>Band-pass</th><th>Band-stop</th></tr>
<tr><td>Butterworth</td><td colspan="2">Order, f_p $(3dB)$, Auxiliary (Polynomial of Numerator21)</td><td rowspan="2">Order, f_p, Auxiliary (Polynomial of Numerator21, Wx)</td><td>Order, f_s, Auxiliary (Wx)</td></tr>
<tr><td>Chebyshev</td><td colspan="2">Order, f_p, Auxiliary (Polynomial of Numerator21)</td><td>Order, f_s, Auxiliary (Quartics of Numerator21, Wx)</td></tr>
<tr><td>Chebyshev Inverse</td><td colspan="3">Order, f_s, Auxiliary (Wx)</td><td>Order, f_s</td></tr>
</table>

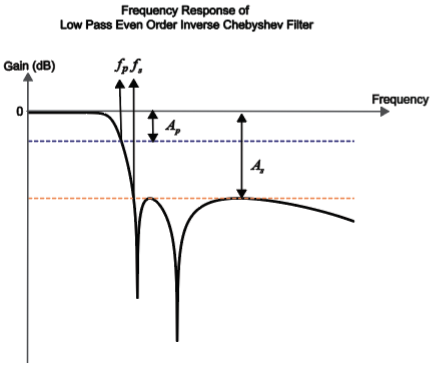
Frequency Responses

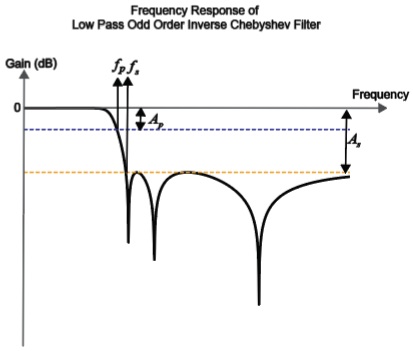
Filter Type	Frequency Response
Lowpass	 ω_p = passband frequency ω_s = stopband frequency A_p = passband attenuation @ ω_p A_s = stopband attenuation @ ω_s
Highpass	 ω_p = passband frequency ω_s = stopband frequency A_p = passband attenuation @ ω_p A_s = stopband attenuation @ ω_s

Filter Type	Frequency Response
Bandpass	 ω_{pL}, ω_{pH} = passband frequencies ω_{sL}, ω_{sH} = stopband frequencies A_p = passband attenuation at specified passband frequencies A_s = stopband attenuation at specified stopband frequencies
Bandstop	 ω_{pL}, ω_{pH} = passband frequencies ω_{sL}, ω_{sH} = stopband frequencies A_p = passband attenuation at specified passband frequencies A_s = stopband attenuation at specified stopband frequencies









Parameters to Define Filter and Design Tips

This table shows all the parameters required to design each filter correctly:

	Low-pass	High-pass	Band-pass	Band-stop
Butterworth	Order, f_p, A_p	Order, f_p, A_p	Order, f_p, A_p	Order, f_s, A_s
	f_p, f_s, A_p, A_s	f_p, f_s, A_p, A_s	f_p, f_s, A_p, A_s	f_p, f_s, A_p, A_s
Chebyshev	Order, f_p, R_p	Order, f_p, R_p	Order, f_p, R_p	Order, f_s, R_p, A_s
	f_p, f_s, R_p, A_s	f_p, f_s, R_p, A_s	f_p, f_s, R_p, A_s	f_p, f_s, R_p, A_s
Chebyshev Inverse	Order, f_p, A_p, A_s	Order, f_p, A_p, A_s	Order, f_p, A_p, A_s	Order, f_s, A_s
	f_p, f_s, A_s, A_p	f_p, f_s, A_s, A_p	f_p, f_s, A_s, A_p	f_p, f_s, A_s, A_p
	Order, f_s, A_s	Order, f_s, A_s	Order, f_s, A_s	
Legend	passband frequency - f_p, passband attenuation/passband ripple – A_p/R_p Note: Passband ripple is parsed in as passband attenuation.		stopband frequency - f_s, stopband attenuation/ stopband ripple - A_s/R_s Note: Stopband ripple is parsed in as stopband attenuation.	

Version History

Introduced in R2018b

References

[1] G.Ellis, Michael,Sr.*Electronic Filter Analysis and Synthesis*,Artech House, 1994

[2] Larry D. Paarmann, *Design and Analysis of Analog Filters, A Signal Processing Perspective with MATLAB Examples*, Kluwer Academic Publishers, 2001.

See Also

`sparameters` | `lcladder` | `rfbudget` | `nport`

Topics

“Design IF Butterworth Bandpass Filter”
“Design, Visualize and Explore Inverse Chebyshev Filter - I”
“Design, Visualize and Explore Inverse Chebyshev Filter - II”

matchingnetwork

Create matching network for 1-port network and generate circuit object

Description

Use the `matchingnetwork` object to create a matching network circuit for a 1-port network which match the impedance of given source to the impedance of given load at a specified center frequency. The `matchingnetwork` object stores the generated network as a `circuit` object in the Circuit property. The function `exportCircuits` could be also used to export the selected circuit(s) generated.

You can use **Matching Network Designer** app to design, visualize, and compare matching networks for one-port load. For more information, see **Matching Network Designer**.

Creation

Syntax

```
matchnet = matchingnetwork
matchnet = matchingnetwork(Name,Value)
```

Description

`matchnet = matchingnetwork` creates a matching network object with default property values.

`matchnet = matchingnetwork(Name,Value)` sets properties using one or more name-value pairs. For example, `matchnet = matchingnetwork('SourceImpedance','60')` creates a matching network with a source impedance of 60 ohms.

Properties

`SourceImpedance` — Source impedance as seen at terminals

`50` (default) | constant complex scalar | S-parameter object | Y-parameter object | Z-parameter object | Touchstone file | one-port circuit object | antenna object | function handle

Source impedance as seen at the terminals looking from the network into the source, specified as one of the following:

- Constant complex scalar in ohms
- `sparameters` object
- `yparameters` object
- `zparameters` object
- File name of a Touchstone file
- One-port circuit object
- Antenna Toolbox™ antenna object

- Function handle to a function that computes an impedance list from a frequency list

Example: `'SourceImpedance',60`

Example: `matchnet.SourceImpedance = 60`

Example: `'SourceImpedance','default.s1p'`

Data Types: `double` | `char` | `string` | `function_handle`

`LoadImpedance` — Load impedance as seen at terminals
`50` (default) | constant complex scalar | s-parameter object | y-parameter object | z-parameter object | Touchstone file | one-port circuit object | antenna object | function handle

Load impedance as seen at the terminals looking from the matching network into the load, specified as one of the following:

- Constant complex scalar in ohms
- `sparameters` object
- `yparameters` object
- `zparameters` object
- File name of a Touchstone file
- One-port circuit object
- Antenna Toolbox antenna object
- Function handle to a function that computes an impedance list from a frequency list

Example: `'LoadImpedance',60`

Example: `matchnet.LoadImpedance = 60`

Data Types: `double` | `char` | `string` | `function_handle`

`CenterFrequency` — Frequency to calculate impedance match between source and load
`1 GHz` (default) | real positive scalar

Frequency to calculate the impedance match between the source and the load, specified as a real positive scalar in hertz

Example: `'CenterFrequency',1e9`

Example: `matchnet.CenterFrequency = 1e9`

Data Types: `double`

`BandWidth` — Desired bandwidth
`100 MHz` (default) | real positive scalar

Desired bandwidth (transducer gain >= minus 3 dB over this bandwidth centered on `CenterFrequency`), specified as a real positive scalar in hertz.

Example: `'BandWidth',100e6`

Example: `matchnet.BandWidth = 100e6`

Data Types: `double`

`LoadedQ` — Desired loaded quality factor
`10` (default) | real positive scalar

Desired loaded quality factor, specified as a real positive scalar. Setting `LoadedQ` updates the bandwidth. If you specify `CenterFrequency`, `LoadedQ` is recalculated from `CenterFrequency` and `BandWidth`.

Example: `'LoadedQ',2`

Example: `matchnet.LoadedQ = 2`

Data Types: `double`

Note The addition of a third element introduces an added degree of freedom allowing you to control the `LoadedQ` property. Hence, the `Bandwidth` and the `LoadedQ` are hidden when there are two components. For more information please see, [1].

`Components` — Number of components or type of topology for matching network design

2 (default) | `3` | `'Pi'` | `'Tee'` | `'L'`

Number of components or type of topology for the matching network design, specified as `2` or `3` for the number of components and `'Pi'`, `'Tee'`, or, `'L'` for the type of topology.

Example: `'Components','Pi'`

Example: `matchnet.Components = 'Pi'`

Data Types: `double` | `char` | `string`

`Circuit` — Set of possible matching network designs

`[1x2 circuit]` (default)

An array of circuit objects containing possible matching network designs for the given set of parameters.

Note This is a read-only property.

Object Functions

addEvaluationParameter	Adds performance goal for sort, pass, or fail matching network design
circuitDescriptions	Tables describing each created matching network's topology and performance
getEvaluationParameters	Table of evaluation parameters currently used to rank and pass or fail matching network designs
clearEvaluationParameter	Delete one or more performance goals
exportCircuits	Select and export generated matching networks as circuit objects from an existing matching network object
rfplot	Plot input reflection coefficient and transducer gain of matching network
smithplot	Plot measurement data on Smith chart
sparameters	Calculate S-parameters for RF data, network, circuit, and matching network objects

Examples

Default Matching Network

Create a default matching network using the object, `matchingnetwork`.

```
matchnet = matchingnetwork
```

```
matchnet =
  matchingnetwork with properties:

       SourceImpedance: 50 Ohms
         LoadImpedance: 50 Ohms
       CenterFrequency: 1 GHz
            Components: 2
               Circuit: [1x2 circuit]
```

Matching Network with Specified Properties

Create a matching network with source impedance, 100 ohms, load impedance, 75 ohms, center frequency, 2 GHz, desired loaded quality factor, 5, and the number of components, 3.

```
mnobj = matchingnetwork('SourceImpedance',100,'LoadImpedance',...
    75,'CenterFrequency',2e9,'LoadedQ',5,'Components',3)
```

```
mnobj =
  matchingnetwork with properties:

       SourceImpedance: 100 Ohms
         LoadImpedance: 75 Ohms
       CenterFrequency: 2 GHz
             Bandwidth: 400 MHz
            Components: 3
               LoadedQ: 5
               Circuit: [1x8 circuit]
```

Display the list of matching network circuits generated and their corresponding performance

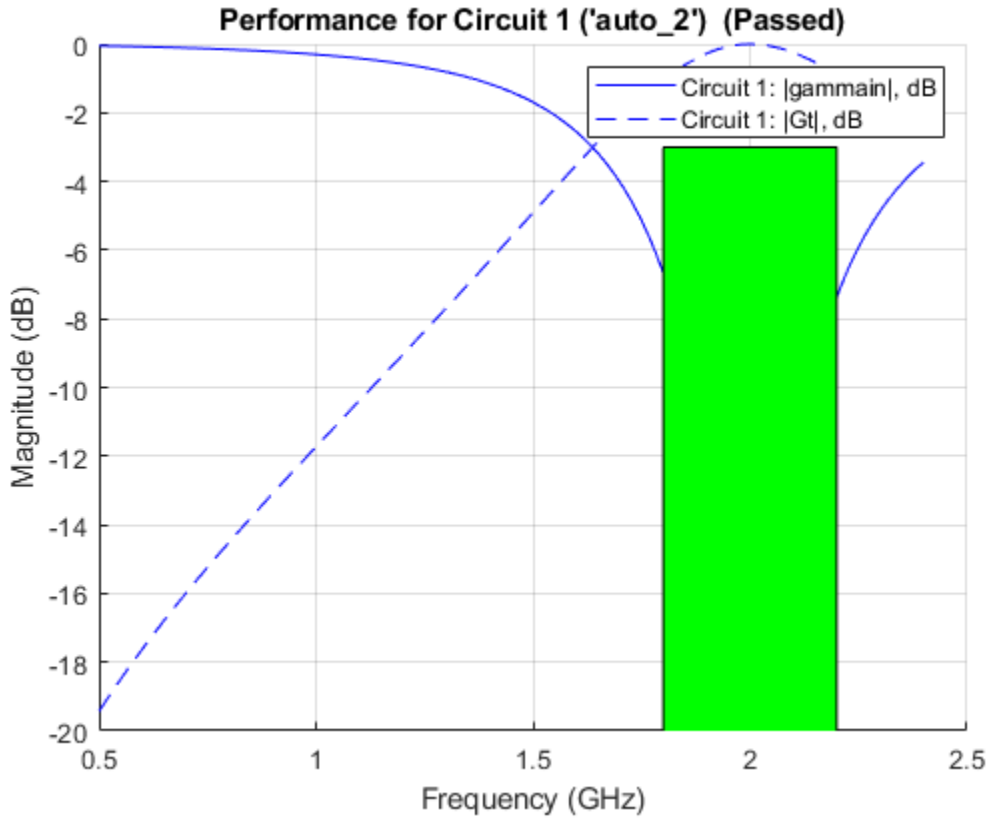
```
[circuit_list, performance] = circuitDescriptions(mnobj)
```

```
circuit_list=8×7 table
                circuitName    component1Type    component1Value    component2Type    component2

    Circuit 1     "auto_2"        "Shunt C"         3.9789e-12        "Series L"        2.1389e-
    Circuit 2     "auto_7"        "Series C"        1.8501e-13        "Shunt C"         2.8519e-
    Circuit 3     "auto_3"        "Shunt L"         1.5915e-09        "Series C"        2.9607e-
    Circuit 4     "auto_6"        "Series L"        3.4228e-08        "Shunt L"         2.2205e-
    Circuit 5     "auto_1"        "Shunt C"         3.9789e-12        "Series L"        2.8468e-
    Circuit 6     "auto_5"        "Series L"        3.4228e-08        "Shunt C"         3.7957e-
    Circuit 7     "auto_4"        "Shunt L"         1.5915e-09        "Series C"        2.2245e-
    Circuit 8     "auto_8"        "Series C"        1.8501e-13        "Shunt L"         1.6684e-


performance=8×4 table
                circuitName    evaluationPassed    testsFailed    performanceScore
```

```
Circuit 1     "auto_2"         {["Yes"]}        {0×0 double}      {[ 1.9447]}
Circuit 2     "auto_7"         {["Yes"]}        {0×0 double}      {[ 1.9447]}
Circuit 3     "auto_3"         {["Yes"]}        {0×0 double}      {[ 1.9443]}
Circuit 4     "auto_6"         {["Yes"]}        {0×0 double}      {[ 1.9443]}
Circuit 5     "auto_1"         {["No" ]}        {[       1]}      {[-0.1254]}
Circuit 6     "auto_5"         {["No" ]}        {[       1]}      {[-0.1254]}
Circuit 7     "auto_4"         {["No" ]}        {[       1]}      {[-0.6947]}
Circuit 8     "auto_8"         {["No" ]}        {[       1]}      {[-0.6947]}
```

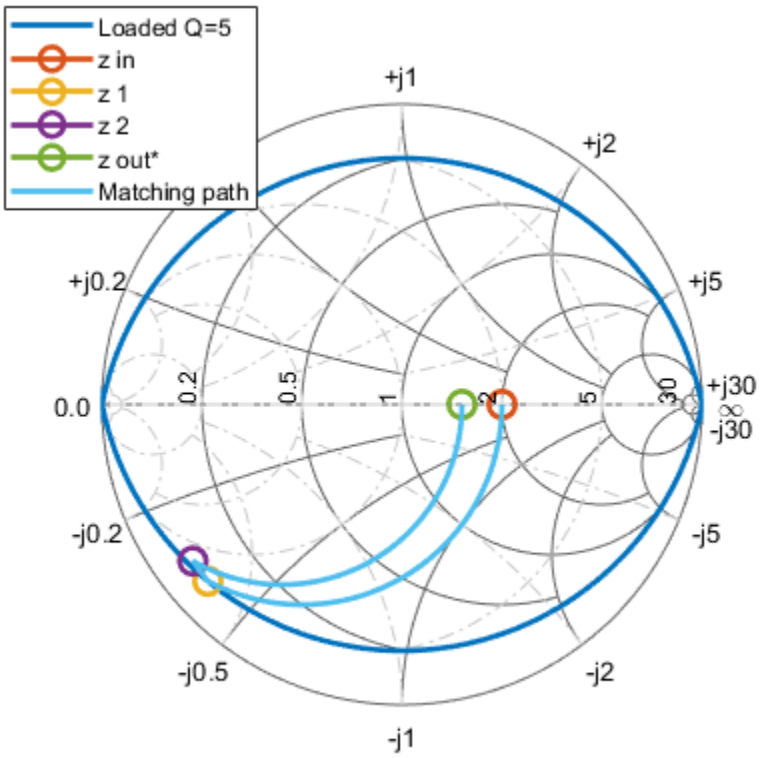
Plot the frequency response of the best circuit (Circuit #1) between 0.5 GHz and 2.5 GHz.

```
frequencies     = linspace(0.5e9,2.4e9);
CircuitIndex    = 1; % Best circuit is sorted to the top
rfplot(mnobj,frequencies,CircuitIndex)
```



Plot impedance transformation for the best matching network generated (Circuit#1). For more information, see `smithplot`.

```
smithplot(mnobj)
```



To export a selected matching network circuit, for example, Circuit #5:

```
CircuitIndex    = 5;
mn_circuit      = mnobj.Circuit(CircuitIndex)
```

```
mn_circuit =
  circuit: Circuit element

    ElementNames: {'C'  'L'  'C_1'}
        Elements: [1×3 rf.internal.circuit.RLC]
           Nodes: [1 2 3]
            Name: 'unnamed'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Alternatively, use exportCircuits(m,CircuitIndex).

Show the default evaluation parameters used by the matching network.

```
ep = getEvaluationParameters(mnobj)
```

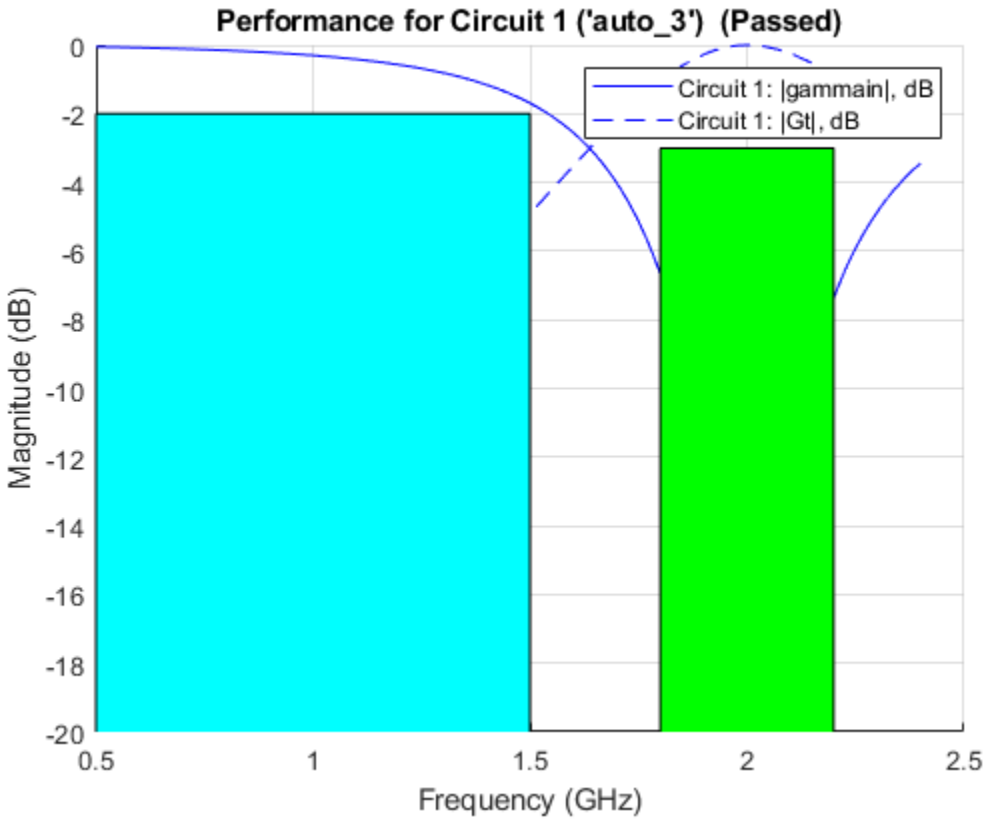
```
ep=1×6 table
    Parameter    Comparison     Goal         Band         Weight       Source
    _________    __________    ______    _____________    ______    _____________

     {'Gt'}        {'>'}       {[-3]}    {1×2 double}     {[1]}     {'Automatic'}
```

Add a new evaluation parameter and plot the frequency response of Circuit #1.

```
mnobj = mnobj.addEvaluationParameter('gammain','>',-2,[0.5e9 1.5e9],1)
```

```
mnobj =
  matchingnetwork with properties:

       SourceImpedance: 100 Ohms
         LoadImpedance: 75 Ohms
       CenterFrequency: 2 GHz
             Bandwidth: 400 MHz
            Components: 3
               LoadedQ: 5
               Circuit: [1x8 circuit]
```

```
rfplot(mnobj,frequencies,1)
```



Matching Network From Dipole Antenna

Create a dipole antenna and create the S-parameters of the antenna. This example requires Antenna Toolbox.

```
d       = dipole('Length', 0.103, 'Width',0.0022);
freq    = linspace(0.5e9,2.5e9,1001);
sd      = sparameters(d, freq);
```

Alternatively, load S-Parameters from the MAT file

```
% load('sparams_dipole.mat')
```

Create a matching network from the S-parameters.

```
n = matchingnetwork('LoadImpedance',sd,'Components',3,...
    'LoadedQ',7,'CenterFrequency',2e9);
```

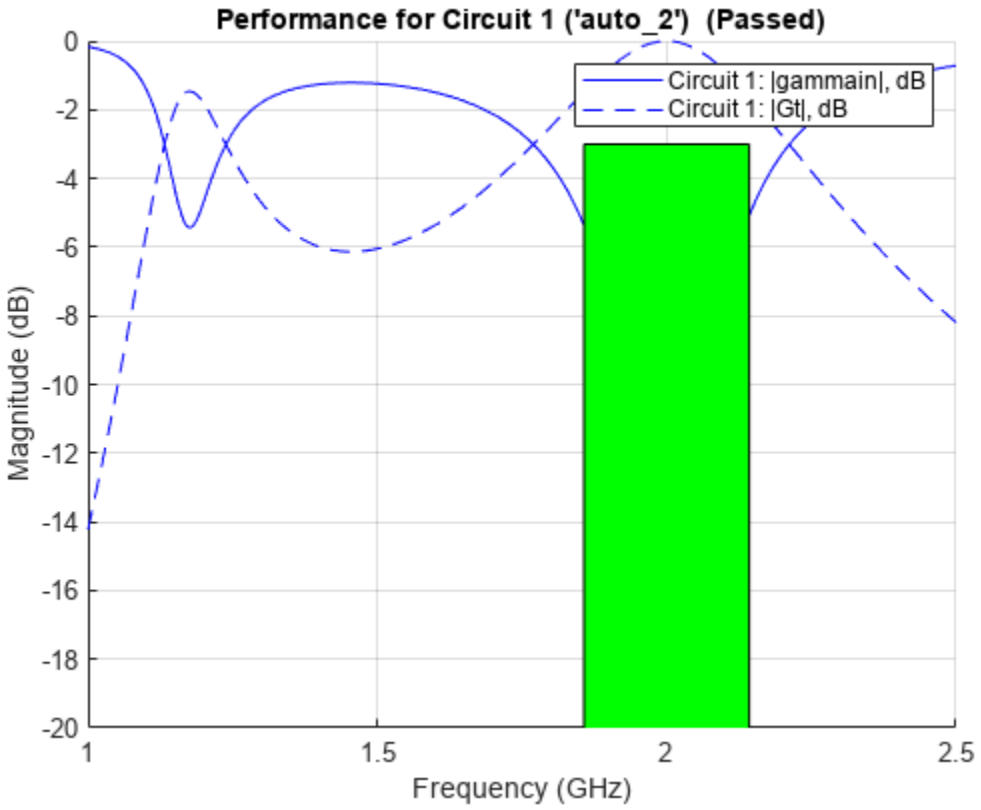
Get the evaluation parameters of the network.

```
t = getEvaluationParameters(n)
```

```
t=1×6 table
    Parameter    Comparison     Goal              Band                    Weight       Source
    _________    __________    ______    ____________________________    ______    _____________

     {'Gt'}        {'>'}       {[-3]}    {[1.8571e+09 2.1429e+09]}       {[1]}     {'Automatic'}
```

Plot the reflection coefficient and transducer gain of the matching network circuit 1 , at a frequency range of 1 GHz to 2.5 GHz.

```
rfplot(n, (1e9:0.001e9:2.5e9),1);
```

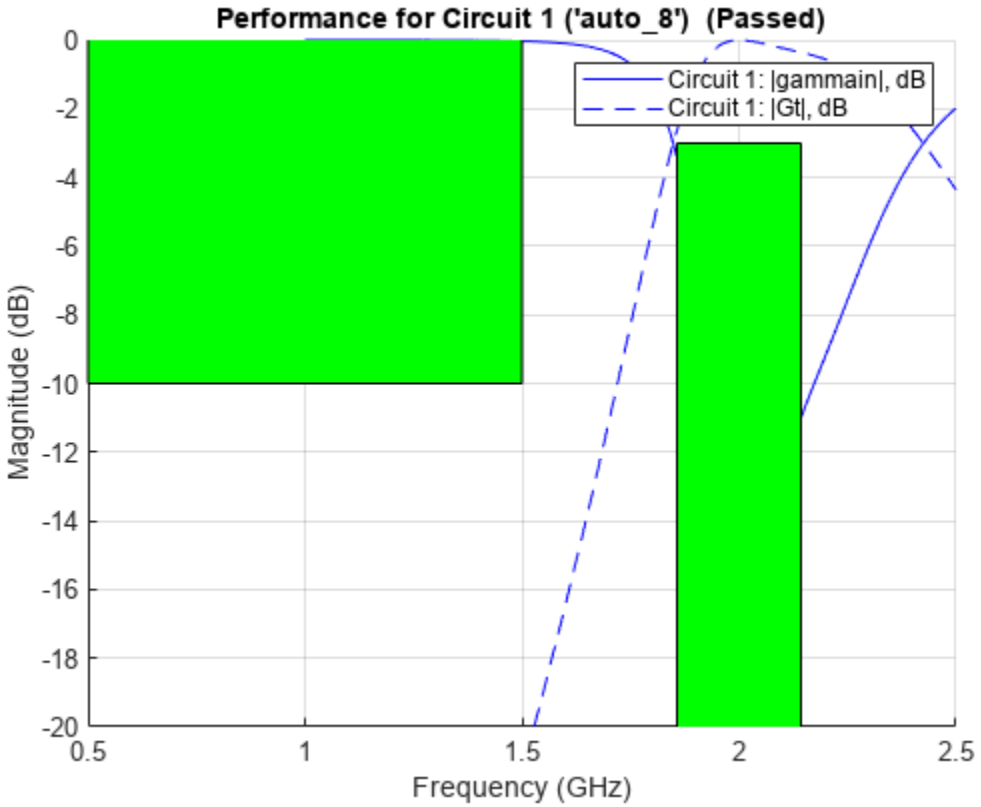


Add a new evaluation parameter to compare the transducer gain to have a cut-off of less than -10 dB. Use a frequency range of 0.5 GHz to 1.5 GHz. Plot the comparisons.

```
n = addEvaluationParameter(n, 'Gt', '<', -10, [0.5e9 1.5e9], 1);
t = getEvaluationParameters(n)
```

```
t=2×6 table
    Parameter    Comparison     Goal              Band                    Weight          Source
    _________    __________    _______    ____________________________    ______    _______________

     {'Gt'}        {'>'}       {[ -3]}    {[1.8571e+09 2.1429e+09]}       {[1]}     {'Automatic'
     {'Gt'}        {'<'}       {[-10]}    {[ 500000000 1.5000e+09]}       {[1]}     {'User-specifie
```

```
rfplot(n, (1e9:0.001e9:2.5e9),1);
```



Clear evaluation parameters.

```
n = clearEvaluationParameter(n,1);
t = getEvaluationParameters(n)
```

```
t=1×6 table
    Parameter    Comparison     Goal              Band                    Weight          Source
    _________    __________    _______    ____________________________    ______    _______________

     {'Gt'}        {'<'}       {[-10]}    {[500000000 1.5000e+09]}        {[1]}     {'User-specified
```

Calculate the S-Parameters of a Matching Network Circuit

This example shows how to calculate the S-Parameters for a newly created matching network for the auto-generated circuit #2 with a reference impedance of 100 Ohm.

```
n       = matchingnetwork('LoadImpedance',100,'Components',3);
freq    = linspace(n.CenterFrequency-n.Bandwidth/2,n.CenterFrequency+n.Bandwidth/2);
RefZ0   = 100;
ckt_no  = 2;
s       = sparameters(n,freq,RefZ0,ckt_no)

s =
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [100x1 double]
     Parameters: [2x2x100 double]
      Impedance: 100

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

Version History

Introduced in R2019a

References

[1] Ludwig, Reinhold, and Gene Bogdanov. *RF Circuit Design: Theory and Applications*. Prentice-Hall, 2009.

[2] Bowick, Chris, et al. *RF Circuit Design*. 2nd ed, 2008.

See Also

Matching Network Designer | `circuit`

Topics

“Design Matching Networks for Passive Multiport Network”
“Impedance Matching of Small Monopole Antenna”

rational

Perform rational fitting to complex frequency-dependent data

Description

Use the `rational` object and an interpolative algorithm to create a rational fit to frequency-dependent data.

The fit is given by the equation:

$$F(s) = \sum_{k=1}^{n} \frac{C_k}{s - A_k} + D$$

where, $s = j \times 2\pi f$

C − residues

A − poles

D − direct term

Creation

Syntax

```
fit = rational(freq,data)
fit = rational(s)
[fit,error] = rational( ___ )
fit = rational( ___ ,Name=Value)
```

Description

`fit = rational(freq,data)` returns a rational object with complex frequencies using the given frequency vector and network parameter data.

`fit = rational(s)` returns a rational object for N-port S-parameters.

`[fit,error] = rational( ___ )` also returns the error of the fit. Use any of input argument combinations from the previous syntaxes.

`fit = rational( ___ ,Name=Value)` sets properties using one or more name-value arguments. For example, `fit = rational(s,MaxPoles=1002)` sets the maximum number of poles for the `fit`. Specify name-value arguments after any of the input arguments from the previous syntaxes.

Input Arguments

`freq` — Nonnegative frequencies

vector

Nonnegative frequencies, specified as a vector of nonnegative frequencies in Hz.

Data Types: `double`

`data` — Network parameter data
vector | 2-D array | 3-D array

Network parameter data, specified as a vector, a 2-D array or a 3-D array. The length of the data values must equal the length of the frequency values.

`s` — N-port S-parameters
N-by-*N* matrix of elements of S

N-port S-parameters, specified as an *N*-by-*N* matrix of elements of S sharing identical poles.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `TendsToZero = true` | `'TendsToZero',true`

`Tolerance` — Relative error tolerance
`-40` (default) | scalar

Relative error tolerance, specified as a scalar less than or equal to zero in dB.

Example: `Tolerance=true`

Data Types: `double`

`TendsToZero` — Behavior of fit for large S-parameters
`true` (default) | `false`

Behavior of fit for large S-parameters, specified as `true` or `false`. When `true`, the direct term in the fit is set to zero so that the rational fit `F(S)` tends to zero as `S` approaches infinity. When `false`, a nonzero direct term is allowed.

Example: `TendsToZero=true`

Data Types: `logical`

`MaxPoles` — Maximum number of poles
`1000` (default) | scalar nonnegative integer

Maximum number of poles, specified as a scalar nonnegative integer.

Example: `MaxPoles=1002`

Data Types: `double`

`ErrorMetric` — Error metrics in `rational` object
`'default'` (default) | `'Relative'`

Error metrics in the `rational` object, specified as one of the following:

- If you specify `'ErrorMetric'` as `'default'`, the `rational` object distributes the error evenly.

- If you specify `'ErrorMetric'` as `'Relative'`, the `rational` object fits both peaks and valleys or gets smaller error for smaller values.

Example: `ErrorMetric='Relative'`

Data Types: `char`

`NoiseFloor` — Ignores low-level noise in data
`-inf` (default) | scalar

Ignores low-level noise in the data, specified as a scalar.

Example: `NoiseFloor=-60`

Data Types: `double`

`Causal` — Pole stability
`true or 1` (default) | `false or 0`

Pole stability, specified as a logical `true` (`1`) or `false` (`0`). When you specify true, all the poles of the fit are stable. When you specify false,the poles can be anywhere in the complex plane.

Example: `Causal=false`

`Qlimit` — Maximum value of quality factor
`1000` (default) | positive scalar

Maximum value of the quality factor of the poles of the fit, specified as a positive scalar.

Example: `Qlimit=1100`

Data Types: `double`

`ColumnReduce` — Data reduction
`true or 1` (default) | `false or 0` | logical

Data reduction, specified as a logical `1` (`true`) or `0` (`false`). When you specify true, the function reduces the data for the fitter to save memory and computation time. Reduce data for the fitter in order to save memory and computation time, specified as a logical `true` or `false`.

Example: `ColumnReduce=false`

`Display` — Display options for fitting algorithm
`'off'` (default) | `'on'` | `'plot'` | `'both'`

Display options for the fitting algorithm of the rational object, specified as one of the following:

- `'off'`— No display
- `'on'`— Printed information
- `'plot'`— Plots of the interpolation progress
- `'both'`— Both printed information and plots.

Example: `Display='on'`

Data Types: `char`

Properties

`NumPorts` — Number of ports

scalar integer

This property is read-only.

Number of ports in the original S-parameter data, returned as a scalar integer. The value of the `NumPorts` is determined from the S-parameter data.

`NumPoles` — Number of poles

scalar integer

This property is read-only.

Number or length of poles in the fit, returned as a scalar integer. The value of the `NumPoles` is determined by the fitter.

`Poles` — Poles of fit

row vector

This property is read-only.

Poles of the fit or complex values of A in the equation provided in the description, returned as a `NumPoles`-by-1 row vector.

Data Types: `double`

`Residues` — Residues in fit

three-dimensional matrix

This property is read-only.

Residues in the fit or complex values of C in the equation provided in the description, returned as a `NumPorts`-by-`NumPorts`-by-`NumPoles` matrix.

Data Types: `double`

`DirectTerm` — Direct term in fit

two-dimensional matrix

This property is read-only.

Direct term in the fit or value of D in the equation provided in the description, returned as a `NumPorts`-by-`NumPorts` matrix.

Data Types: `double`

`ErrDB` — Error between fit and original data

scalar

This property is read-only.

Error between the fit and the original data, returned as a scalar in dB.

Object Functions

timeresp	Time response for rational objects
stepresp	Step-signal response for rational object and rationalfit function object
freqresp	Frequency response of rational object and rationalfit function object
pwlresp	Calculate time response of piecewise linear input signal
impulse	Impulse response for rational function object
ispassive	Return true if rationalfit output is passive at all frequencies
makepassive	Enforce passivity of rationalfit output or a rational object
passivity	Plot passivity of N-by-N rationalfit function output
generateSPICE	Generate SPICE file from rationalfit of S-parameters
abcd	Construct state-space matrices from rational object
zpk	Compute zeros, poles, and gain of rational object

Examples

Fit `passive.s2p` File

Create S-Parameters from the file named `passive.s2p`.

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

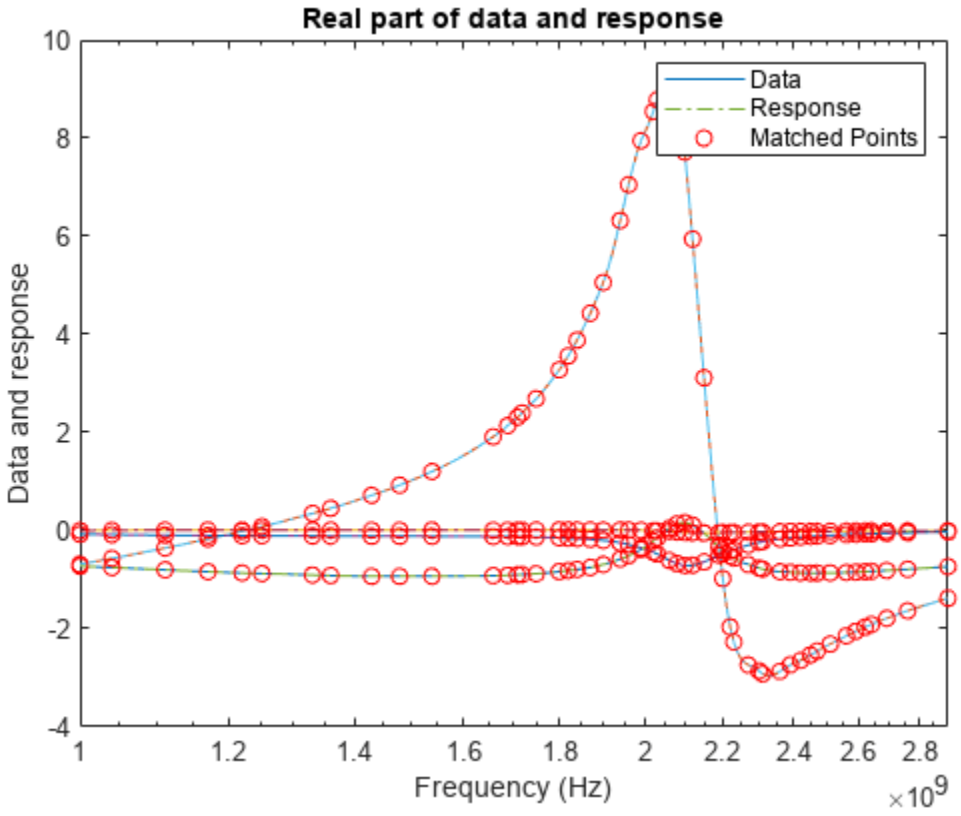
Perform rational fitting of the S-parameters.

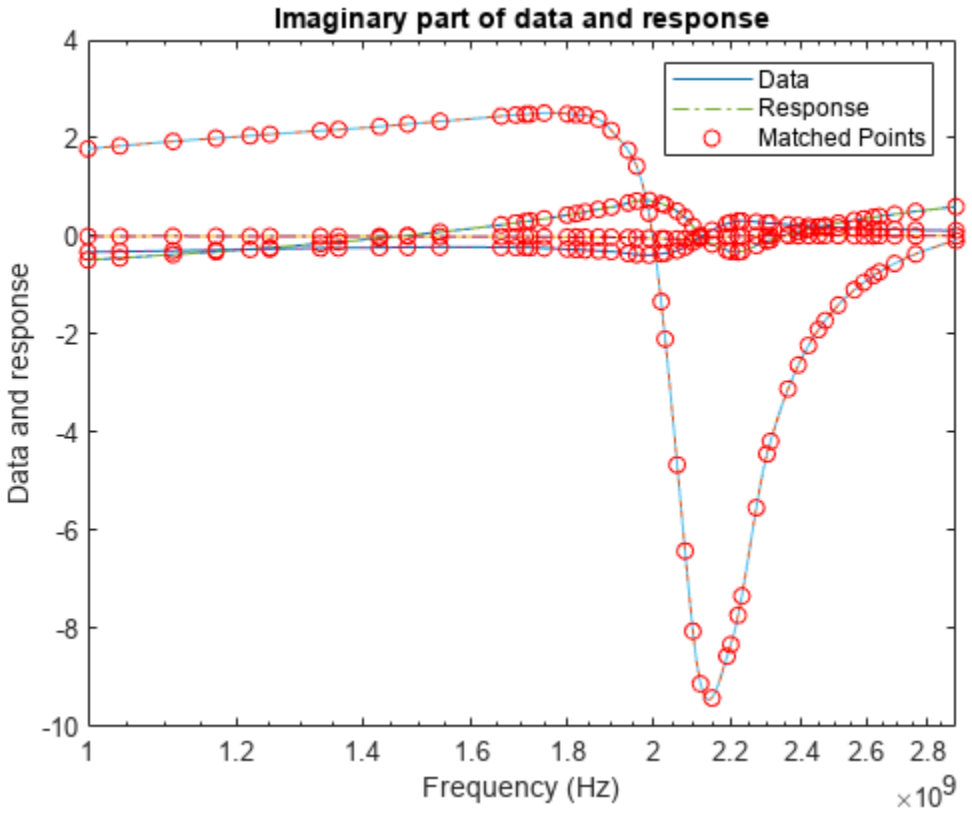
```
fit = rational(S);
```

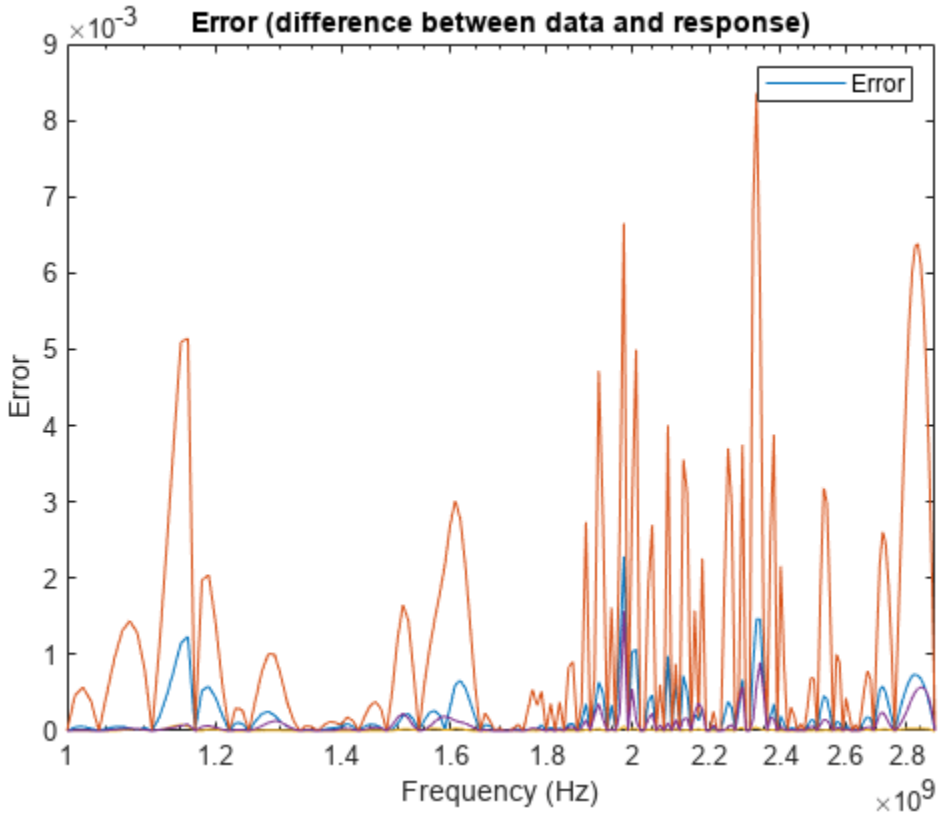
Zeros, Poles, Gain, and DC Gain of Fit

Create an S-Parameters object from the file named `default.s2p`. Perform rational fitting of the S-Parameters.

```
S = sparameters('default.s2p');
fit = rational(S,Display='plot')
```


Real part of data and response
Data
Response
Matched Points
Data and response
Frequency (Hz)
×10^9
10
8
6
4
2
0
-2
-4
1
1.2
1.4
1.6
1.8
2
2.2
2.4
2.6
2.8


Imaginary part of data and response
Data
Response
Matched Points
Data and response
Frequency (Hz)
×10^9
4
2
0
-2
-4
-6
-8
-10
1
1.2
1.4
1.6
1.8
2
2.2
2.4
2.6
2.8




```
fit =
  rational with properties:

      NumPorts: 2
      NumPoles: 52
         Poles: [52x1 double]
      Residues: [2x2x52 double]
    DirectTerm: [2x2 double]
         ErrDB: -22.6872
```

Calculate the zeros, poles, gain, and DC gain of the rational object.

```
[z,p,k,dcgain] = zpk(fit)
```

```
z=2×2 cell array
    {51x1 double}    {51x1 double}
    {51x1 double}    {51x1 double}


p=2×2 cell array
    {52x1 double}    {52x1 double}
    {52x1 double}    {52x1 double}


k = 2×2
10^10 ×
```

```
    1.0544   -0.0194
    0.9158    0.0377


dcgain = 2×2

    0.1289   -0.0838
   -0.1209    0.7649
```

Fit Peaks and Valleys in `rational` Object

Create an S-parameters object from the specified S2P file.

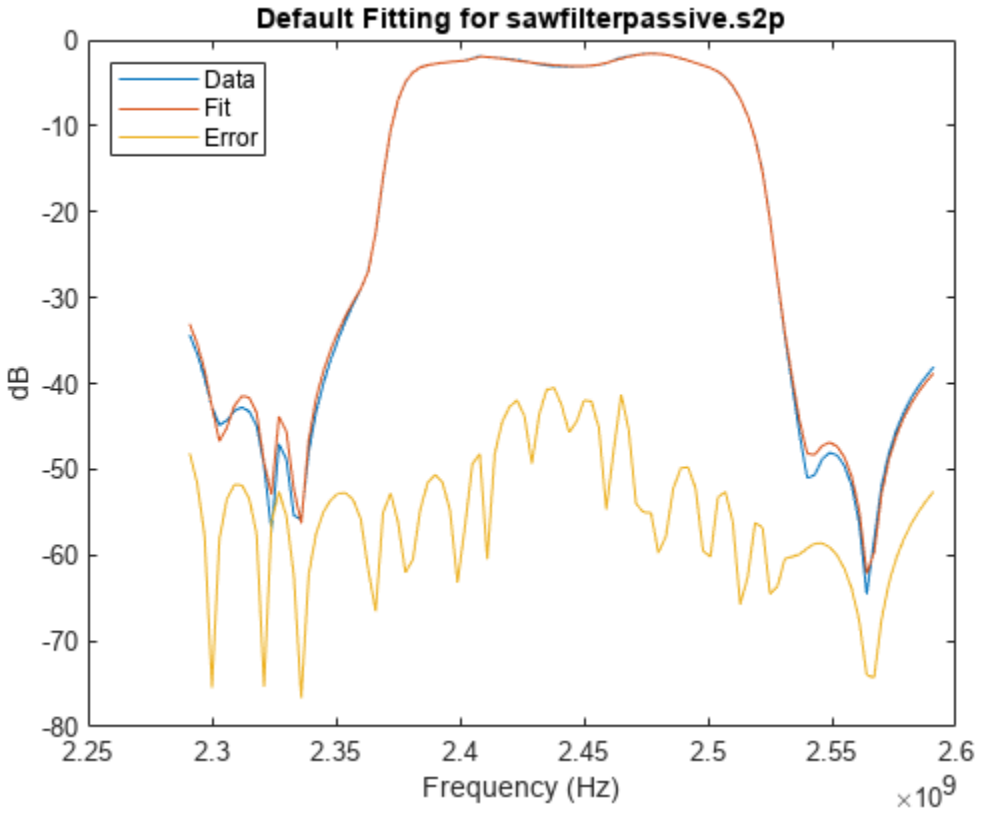
```
S = sparameters('sawfilterpassive.s2p');
f = S.Frequencies;
```

Create a `rational` object with the tolerance of -40 dB.

```
fit = rational(S,Tolerance=-40);
```

Compare the fit to the data. You can see the deviations at the smaller values on a semi-log plot.

```
dresp = freqresp(fit,f);
plot(f,20*log10(abs(squeeze(S.Parameters(2,1,:))))),...
    f,20*log10(abs(squeeze(dresp(2,1,:)))),f,20*log10(abs(squeeze(S.Parameters(2,1,:)-dresp(2,1,
title('Default Fitting for sawfilterpassive.s2p');
ylabel('dB');
xlabel('Frequency (Hz)');
legend('Data','Fit','Error','Location','northwest');
```

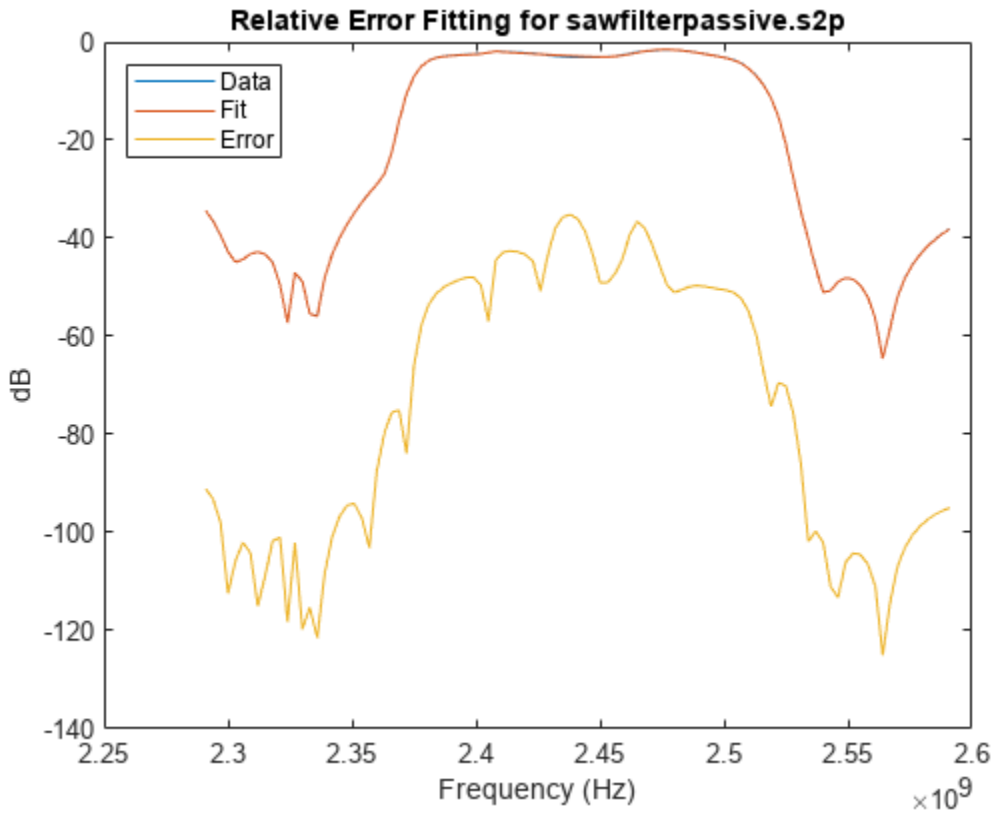


Create a `rational` object with `'ErrorMetric'` set to `'Relative'` to fit both peaks and valleys.

```
rfit = rational(S,Tolerance=-40,ErrorMetric='Relative');
```

Compare the fit to the data. The peaks and valleys are fitted.

```
rresp = freqresp(rfit,f);
figure(2)
plot(f,20*log10(abs(squeeze(S.Parameters(2,1,:)))),...
    f,20*log10(abs(squeeze(rresp(2,1,:)))), f, 20*log10(abs(squeeze(S.Parameters(2,1,:)-rresp(2,1
title('Relative Error Fitting for sawfilterpassive.s2p');
ylabel('dB');
xlabel('Frequency (Hz)');
legend('Data','Fit','Error','Location','northwest');
```



Ignore Low-Level Noise in Fitter Data

Create an S-parameters object from the specified S2P file.

```
S = sparameters('passive.s2p');
f = S.Frequencies;
data = S.Parameters;
```

Set one of the data entries to zero.

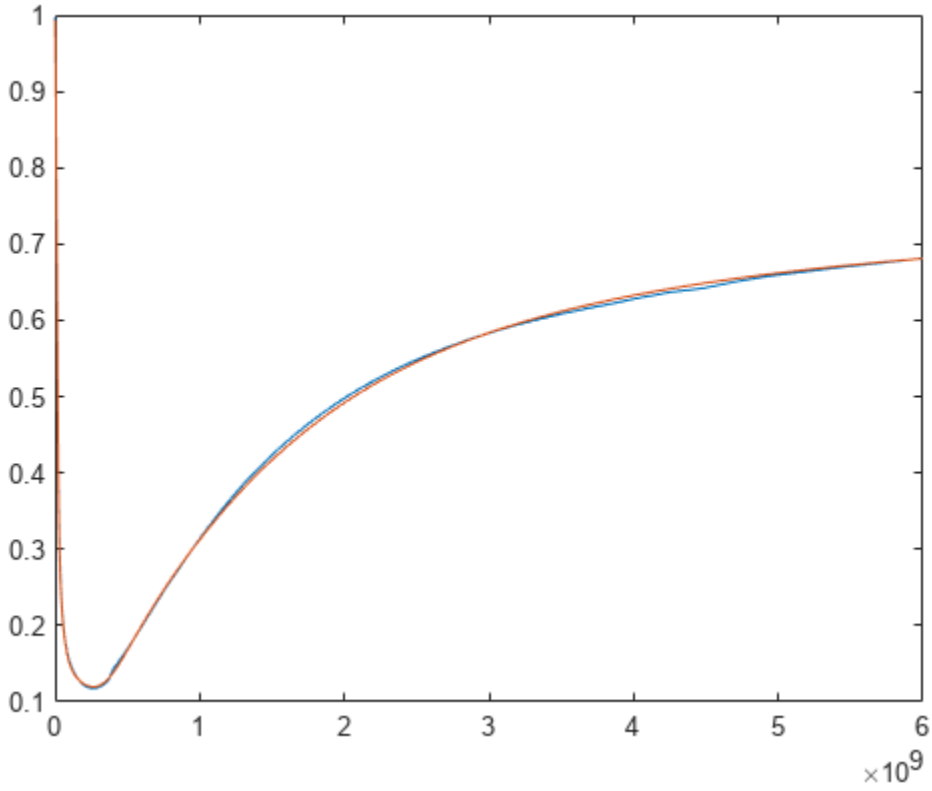
```
data(2,2,:) = 0;
```

Create a `rational` object with the tolerance of `-40` dB.

```
fit = rational(f,data,Tolerance=-40);
```

Compare the fit to the data. The fit and the data match closely.

```
xresp = freqresp(fit,f);
figure(3)
plot(f,abs(squeeze(data(2,1,:))),f,abs(squeeze(xresp(2,1,:))))
```

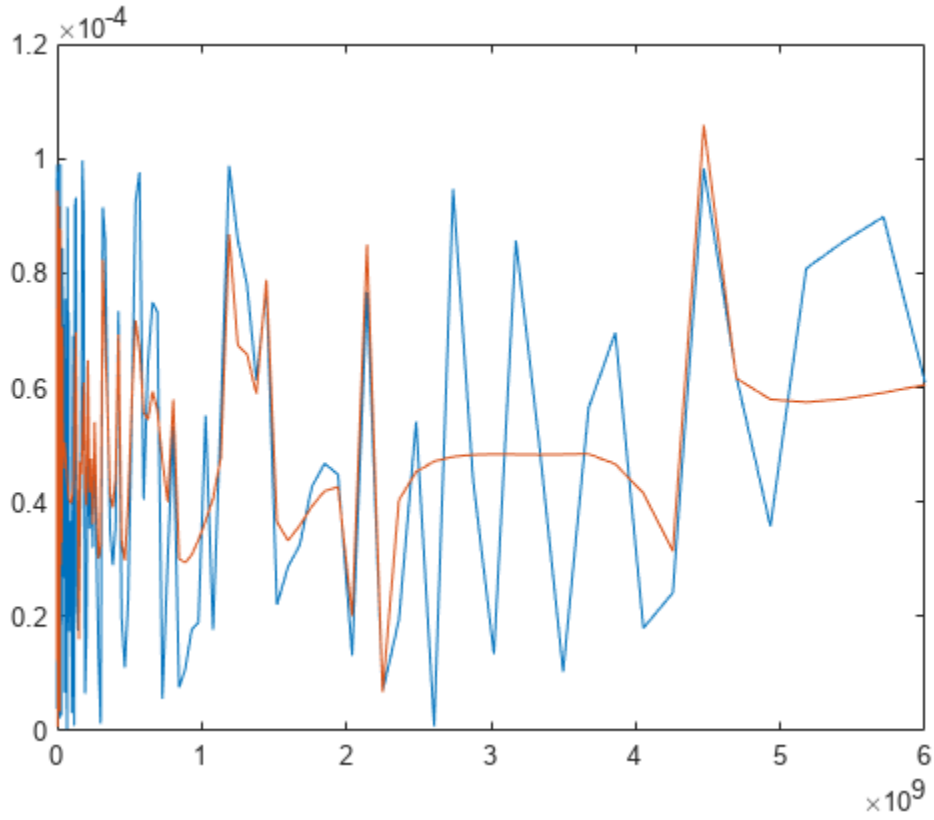


Add noise to the data and create a `rational` object.

```
rng(1);
noisyData = data + 1e-4 * rand(size(S.Parameters));
nfit = rational(f,noisyData,Tolerance=-40);
```

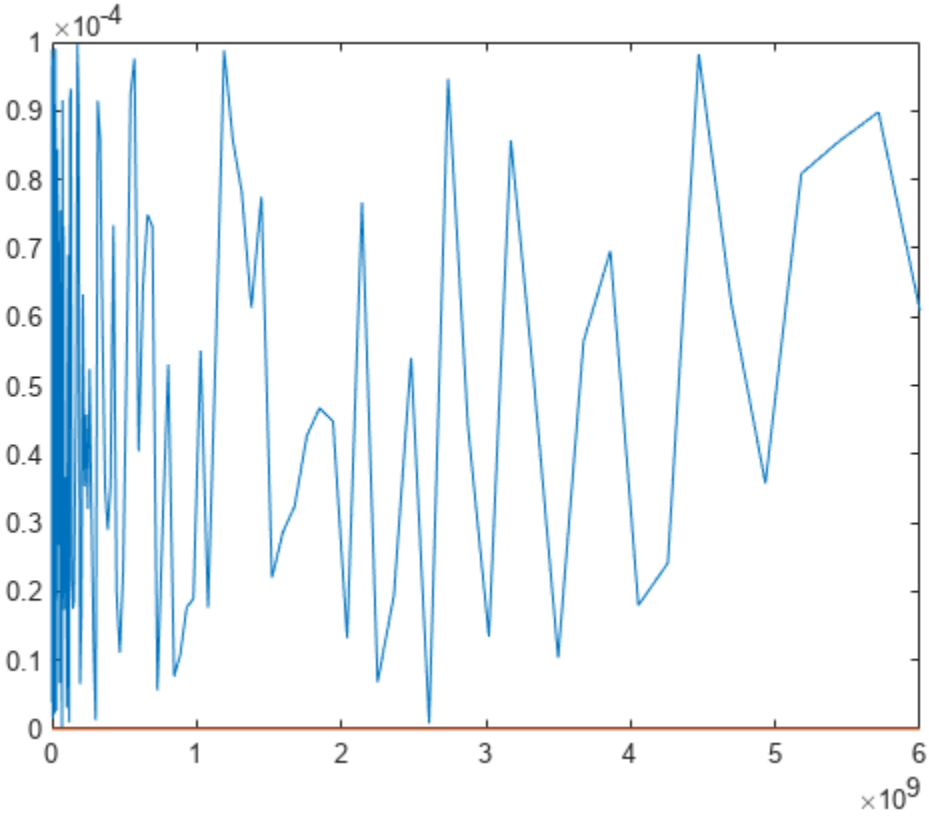
Compare the fit to the data with noise. Noise cannot be fitted because this is a data with a higher order fit with a worse error metric.

```
nresp = freqresp(nfit,f);
figure(4)
plot(f,abs(squeeze(noisyData(2,2,:))),f,abs(squeeze(nresp(2,2,:))))
```



Create a `rational` object with the noise floor of `-60` dB and plot the fit. The fitter ignores low-level noise.

```
ffit = rational(f,noisyData,Tolerance=-40,NoiseFloor=-60);
fresp = freqresp(ffit,f);
figure(5)
plot(f,abs(squeeze(noisyData(2,2,:))),f,abs(squeeze(fresp(2,2,:))))
```



Version History

Introduced in R2020a

References

[1] Nakatsukasa, Yuji, Olivier Sète, and Lloyd N. Trefethen. “The AAA Algorithm for Rational Approximation.” *SIAM Journal on Scientific Computing* 40, no. 3 (January 2018): A1494–1522. https://doi.org/10.1137/16M1106122.

See Also

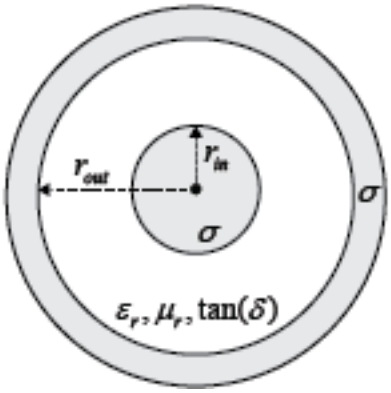
`rationalfit`

txlineCoaxial

Create coaxial transmission line

Description

Use the `txlineCoaxial` object to create a coaxial transmission line. A cross-section of a coaxial transmission line is shown in this figure. The physical characteristics of a coaxial transmission line include the radius of the inner conductor, r_{in} and the radius of the outer conductor, r_{out}.



Creation

Syntax

```
coaxialtxline = txlineCoaxial
coaxialtxline = txlineCoaxial(Name,Value)
```

Description

`coaxialtxline = txlineCoaxial` creates a default coaxial transmission line object.

`coaxialtxline = txlineCoaxial(Name,Value)` sets "Properties" on page 1-255 using one or more name-value pairs. For example, `txline = txlineCoaxial('OuterRadius',0.0046)` creates a coaxial transmission line with an outer radius of `0.0046` meters.

Properties

Name — Name of coaxial transmission line

`'coaxial'` (default) | string scalar | character vector

Name of the coaxial transmission line, specified as a string scalar or a character vector.

Example: `'Name','coaxial1'`

Example: `coaxialtxline.Name = 'coaxial1'`

Data Types: `char` | `string`

`OuterRadius` — Outer conductor radius
`0.0026` (default) | positive scalar

Outer conductor radius, specified as a positive scalar in meters.

Example: `'OuterRadius',0.0074`

Example: `coaxialtxline.OuterRadius = 0.0074`

Data Types: `double`

`InnerRadius` — Inner conductor radius
`0.000725` (default) | positive scalar

Inner conductor radius, specified as a positive scalar in meters.

Example: `'InnerRadius',0.000875`

Example: `coaxialtxline.InnerRadius = 0.000875`

Data Types: `double`

`MuR` — Relative permeability of dielectric
`1` (default) | positive scalar

Relative permeability of the dielectric, specified as a positive scalar.

Relative permeability is the ratio of the permeability of the dielectric, μ, to the permeability in free space, μ_0.

Example: `'MuR',1.5`

Example: `coaxialtxline.MuR = 1.5`

Data Types: `double`

`EpsilonR` — Relative permittivity of dielectric
`2.3` (default) | positive scalar

Relative permittivity of the dielectric, specified as a positive scalar.

Example: `'EpsilonR',1.4`

Example: `coaxialtxline.EpsilonR = 1.4`

Data Types: `double`

`LossTangent` — Loss angle tangent of dielectric
`0` (default) | nonnegative scalar

Loss angle tangent of the dielectric, specified as a nonnegative scalar.

Example: `'LossTangent',1`

Example: `coaxialtxline.LossTangent = 1`

Data Types: `double`

`SigmaCond` — Conductivity of conductor
`Inf` (default) | nonnegative scalar

Conductivity of the conductor, specified as a nonnegative scalar in Siemens per meter (S/m).

Example: `'SigmaCond',2`

Example: `coaxialtxline.SigmaCond = 2`

Data Types: `double`

`LineLength` — Physical length of transmission line

`0.0100` (default) | positive scalar

Physical length of the coaxial transmission line, specified as a positive scalar in meters.

Example: `'LineLength',0.2`

Example: `coaxialtxline.LineLength = 0.2`

Data Types: `double`

`Termination` — Stub transmission line termination

`'NotApplicable'` (default) | `'Open'` | `'Short'`

Stub transmission line termination, specified as `'NotApplicable'`, `'Open'`, or `'Short'`.

Example: `'Termination','Short'`

Example: `coaxialtxline.Termination = 'Short'`

Data Types: `char`

`StubMode` — Type of stub

`'NotAStub'` (default) | `'Series'` | `'Shunt'`

Type of stub, specified as `'NotAStub'`, `'Series'`, or `'Shunt'`.

Example: `'StubMode','Series'`

Example: `coaxialtxline.StubMode = 'Series'`

Data Types: `char`

`NumPorts` — Number of input and output ports

`2` (default) | positive scalar

This property is read-only.

Number of input and output ports, specified as a positive scalar.

Data Types: `double`

`Terminals` — Terminals of coaxial transmission line

`{'p1+' p2+' 'p1-' 'p2-'}` (default) | cell array of strings

This property is read-only.

Terminals of coaxial transmission line, specified as cell array of strings.

Data Types: `char` | `string`

Object Functions

sparameters	Calculate S-parameters for RF data, network, circuit, and matching network objects
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object
noisefigure	Calculate noise figure of transmission lines, series RLC, and shunt RLC circuits
getZ0	Calculate characteristic impedance with and without dispersion for transmission line
circuit	Circuit object
clone	Create copy of existing circuit element or circuit object

Examples

Calculate Characteristic Impedance with Dispersion for Coaxial Transmission Line

Create a coaxial transmission line.

```
txline = txlineCoaxial('OuterRadius',1.47e-3,'InnerRadius',0.45e-3,...
            'EpsilonR',3.4,'SigmaCond',5.8e7);
```

Calculate the characteristic impedance with dispersion for the coaxial transmission line at at the frequency of 6 GHz.

```
z0_f = getZ0(txline,6e9)
```

```
z0_f = 38.4927 - 0.0201i
```

Version History

Introduced in R2020b

See Also

`txlineCPW` | `txlineMicrostrip` | `txlineParallelPlate` | `txlineRLCGLine` | `txlineTwoWire`

Topics

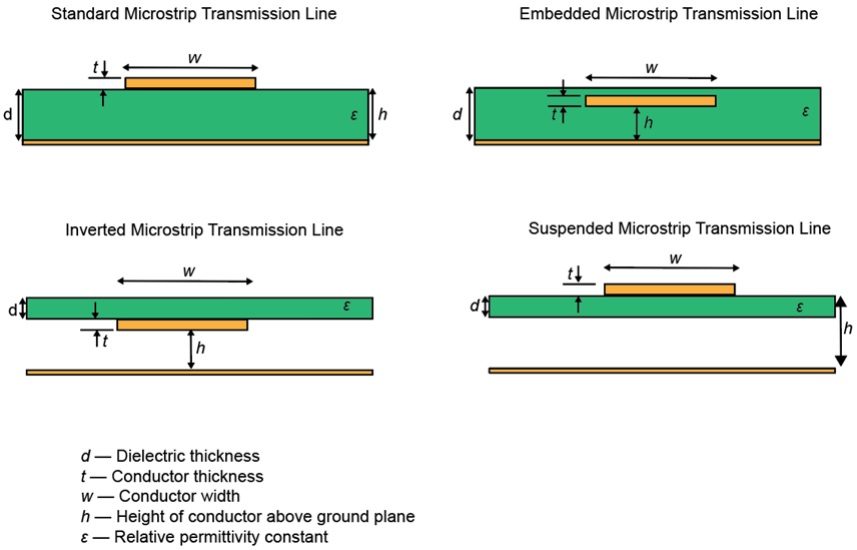
“Design Two-Stage Low Noise Amplifier Using Microstrip Transmission Line Matching Network”

txlineMicrostrip

Create microstrip transmission line

Description

Use the `txlineMicrostrip` object to create a standard, embedded, inverted, or suspended microstrip transmission line. This figure shows the cross sections of the four types of microstrip transmission lines you can create using the `txlineMircostrip` object. The physical characteristics of the microstrip transmission line include the conductor width (*w*), the conductor thickness (*t*), the dielectric thickness (*d*), the relative permittivity constant (*ε*), and the height of the conductor above the ground plane (*h*) .



Creation

Syntax

```
txline = txlineMicrostrip
txline = txlineMicrostrip(Name,Value)
```

Description

`txline = txlineMicrostrip` creates a standard microstrip transmission line object.

`txline = txlineMicrostrip(Name,Value)` sets “Properties” on page 1-260 using one or more name-value pairs. For example, `txline = txlineMicrostrip('Width',0.0046)` creates a standard microstrip transmission line with a width of `0.0046` meters.

Properties

`Name` — Name of microstrip transmission line

`'Microstrip'` (default) | string scalar | character vector

Name of the microstrip transmission line, specified as a string scalar or a character vector.

Example: `'Name','microstrip1'`

Example: `txline.Name = 'microstrip1'`

Data Types: `char` | `string`

`Type` — Type of microstrip transmission line

`'Standard'` (default) | `'Embedded'` | `'Inverted'` | `'Suspended'`

Type of microstrip transmission line, specified as one of the following:

- `'Standard'` — Standard microstrip transmission line
- `'Embedded'` — Embedded microstrip transmission line
- `'Inverted'` — Inverted microstrip transmission line
- `'Suspended'` — Suspended microstrip transmission line

Note When you create a default `txlineMicrostrip` object or set the Type property to 'Standard', the Type property does not show in MATLAB

Example: `'Type','Embedded'`

Example: `txline.Type = 'Embedded'`

Data Types: `char` | `string`

`LineLength` — Physical length of microstrip transmission line

`0.0100` (default) | positive scalar

Physical length of the microstrip transmission line, specified as a positive scalar in meters.

Example: `'LineLength',0.0200`

Example: `txline.LineLength = 0.0200`

Data Types: `double`

`Width` — Physical width of microstrip transmission line

`0.0006` (default) | positive scalar

Physical width of the microstrip transmission line, specified as a positive scalar in meters.

Example: `'Width',0.0008`

Example: `txline.Width = 0.0008`

Data Types: `double`

`Height` — Physical height of conductor

`0.000635` (default) | positive scalar

Physical height of the conductor, specified as a positive scalar in meters.

Example: `'Height',0.000835`

Example: `txline.Height = 0.000835`

Data Types: `double`

`DielectricThickness` — Dielectric thickness of microstrip transmission line

positive scalar

Dielectric thickness of the inverted, embedded, or suspended microstrip transmission line, specified as a positive scalar in meters. Default values of dielectric thickness for the embedded, inverted, and suspended microstrip transmission lines are is listed in the table.

Type of Microstrip	Default value of `DielctricThickness`
`'Embedded'`	`Height*2`
`'Inverted'`	`Height`
`'Suspended'`	`Height/2`

Note

- When you create a standard microstrip transmission line, the `DielectricThickness` property does not show in MATLAB.
- By default `txlineMicrostrip` object sets the dielectric thickness of the standard microstrip transmission line to the value of the “Height” on page 1-0 .

Example: `'DielectricThickness',0.0012`

Example: `txline.DielectricThickness = 0.0012`

Dependencies

To enable this property, set `Type` as `'Embedded'` or `'Inverted'`, or `'Suspended'`.

Data Types: `double`

`Thickness` — Physical thickness of microstrip transmission line

`0.000005` (default) | positive scalar

Physical thickness of the microstrip transmission line, specified as a positive scalar in meters. You can now model microstrip with thickness set to `0` mm.

Example: `'Thickness',0.000008`

Example: `txline Thickness = 0.000008`

Data Types: `double`

`EpsilonR` — Relative permittivity of dielectric

`9.8` (default) | positive scalar

Relative permittivity of the dielectric, specified as a positive scalar.

Example: `'EpsilonR',8.8`

Example: `txline.EpsilonR = 8.8`

Data Types: `double`

`LossTangent` — Loss angle tangent of dielectric
`0` (default) | nonnegative scalar

Loss angle tangent of the dielectric, specified as a nonegative scalar.

Example: `'LossTangent',1`

Example: `txline.LossTangent = 1`

Data Types: `double`

`SigmaCond` — Conductivity of conductor
`Inf` (default) | nonnegative scalar

Conductivity of the conductor, specified as a nonnegative scalar in Siemens per meter (S/m).

Example: `'SigmaCond',2`

Example: `txline.SigmaCond = 2`

Data Types: `double`

`Termination` — Stub transmission line termination
`'NotApplicable'` (default) | `'Open'` | `'Short'`

Stub transmission line termination, specified as `'NotApplicable'`, `'Open'`, or `'Short'`.

Example: `'Termination','Short'`

Example: `txline.Termination = 'Short'`

Data Types: `char`

`StubMode` — Type of stub
`'NotAStub'` (default) | `'Series'` | `'Shunt'`

Type of stub, specified as `'NotAStub'`, `'Series'`, or `'Shunt'`.

Example: `'StubMode','Series'`

Example: `txline.StubMode = 'Series'`

Data Types: `char`

`NumPorts` — Number of input and output ports
`2` (default) | positive scalar

This property is read-only.

Number of input and output ports, returned as a positive scalar.

Data Types: `double`

`Terminals` — Terminals of microstrip transmission line
`{'p1+' p2+' 'p1-' 'p2-'}` (default) | cell array of strings

This property is read-only.

Terminals of the microstrip transmission line, returned as a cell array of strings.

Data Types: `char` | `string`

Object Functions

sparameters	Calculate S-parameters for RF data, network, circuit, and matching network objects
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object
noisefigure	Calculate noise figure of transmission lines, series RLC, and shunt RLC circuits
getZ0	Calculate characteristic impedance with and without dispersion for transmission line
circuit	Circuit object
clone	Create copy of existing circuit element or circuit object

Examples

S-Parameters and Group Delay of Microstrip Transmission Line

Create a microstrip transmission line using these specifications:

- Width: `0.08` mm
- Height: `1.6` mm
- Line length: `12.2777` mm
- Thickness: `10e-6` m
- Conductivity: `5.88e7` S/m
- Relative permittivity of the dielectric: `3.9`

```
microstriptxline = txlineMicrostrip('Width',0.08e-3,'Height',1.6e-3,...
    'LineLength',12.2777e-3,'Thickness',10e-6,'EpsilonR',3.9,'SigmaCond',5.88e7);
```

Calculate the S-parameters of the transmission line at `10` GHz.

```
sparam = sparameters(microstriptxline,10e9,50);
```

Calculate the group delay of the transmission line at `10` GHz.

```
gd = groupdelay(microstriptxline,10e9,'Impedance',50)
```

```
gd = 4.2440e-11
```

S-Parameters and Characteristic Impedance of Suspended Microstrip Line

This example uses RF PCB Toolbox to calculate electromagnetic (EM) solver S-parameters of the microstrip line.

Create Suspended Microstrip Line

Create a suspended microstrip transmission line with a copper conductor and Teflon substrate.

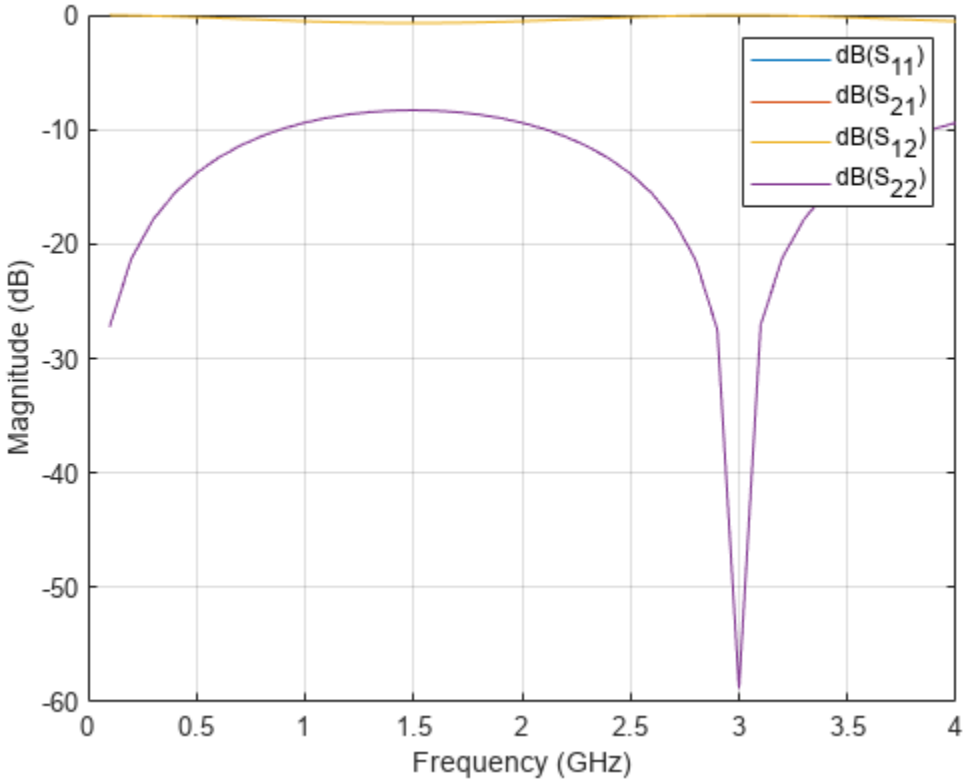
```
tx = txlineMicrostrip('Type','Suspended',...
    'LineLength',0.04705,'Width',3.5e-3,...
    'Height',1.6e-3,"DielectricThickness",0.8e-3,...
```

```
    "EpsilonR",2.1,"LossTangent",0.2e-3,...
    'SigmaCond',596e5,"Thickness",3.556e-5,...
    "StubMode","NotAStub","Termination","NotApplicable");
```

Behavioral Modeling

Calculate and plot the S-parameters with the reference impedance of 50Ω.

```
freq = (1:40)*100e6;
Srf = sparameters(tx,freq,50);
rfplot(Srf)
```



Calculate the characteristic impedance.

```
Zc_rf = getZ0(tx)
```

```
Zc_rf = 75.0279
```

EM Modeling

Input the microstrip transmission line to the microstripLine object from the RF PCB Toolbox for EM modeling.

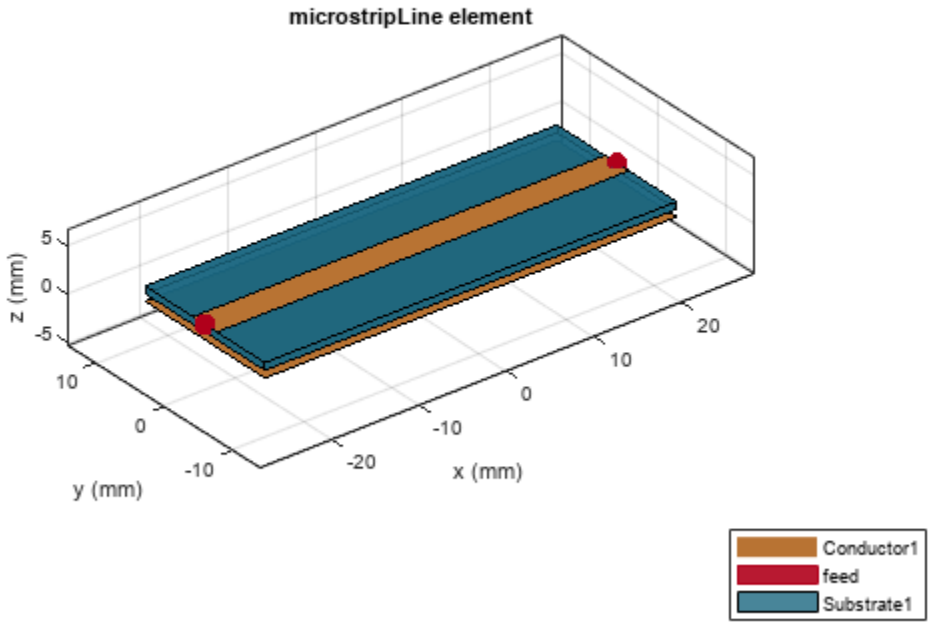
```
tx_em = microstripLine(tx)
```

```
tx_em =
  microstripLine with properties:
```

```
            Length: 0.0471
             Width: 0.0035
            Height: 0.0016
    GroundPlaneWidth: 0.0175
           Substrate: [1x1 dielectric]
           Conductor: [1x1 metal]
```

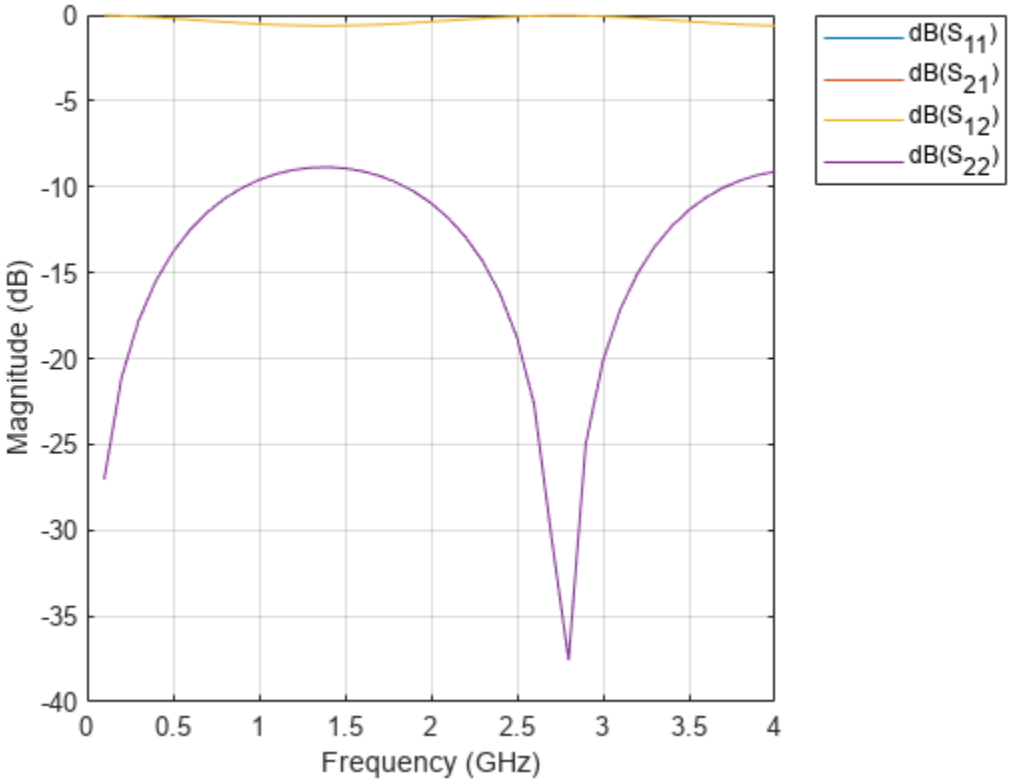
View the suspended microstrip transmission line.

```
show(tx_em)
```



Calculate and plot the S-parameters using EM solver from RF PCB Toolbox.

```
Sem = sparameters(tx_em,freq,50);
rfplot(Sem)
```




```
Zc_em = getZ0(tx_em)
```

```
Zc_em = 72.6236 - 0.2017i
```

S-Parameters for Inverted Microstrip Transmission Line

Select the dielectric and metal layers for an inverted microstrip transmission line from the dielectric and metal libraries, respectively, of the RF PCB Toolbox.

```
dFR4 = dielectric('FR4');
dFR4.Thickness = 3.2e-4;
mCopper = metal('Copper');
```

Create an inverted microstrip transmission line with a copper conductor and an FR4 substrate at 6 GHz with the line length of 0.5λ and the reference impedance of 75Ω. The air to substrate thickness ratio is calculated using:

$$\frac{\text{Thickness of air layer}}{\text{Thickness of substrate}} = \frac{12.8e-4}{3.2e-4} = 4$$

```
prototype_behavioral = txlineMicrostrip('Type','Inverted',...
    'DielectricThickness',dFR4.Thickness,"EpsilonR",dFR4.EpsilonR, ...
    'Height',12.8e-4,"LossTangent",dFR4.LossTangent, ...
    "SigmaCond",mCopper.Conductivity,"Thickness",mCopper.Thickness);
```

Input the inverted microstrip transmission line to the `microstripLine` object from the RF PCB Toolbox for EM modeling.

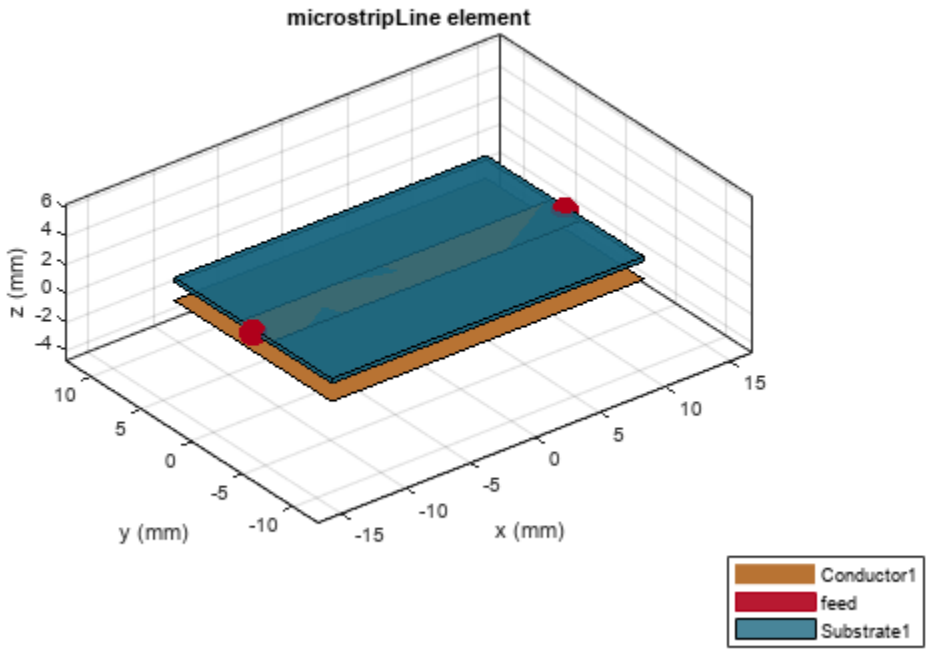
```
prototype_em = microstripLine(prototype_behavioral);
```

Use the `design` (Antenna Toolbox) function to design the `microstripLine` (RF PCB Toolbox) object at 6 GHz with the line length of 0.5λ and reference impedance of 75Ω.

```
tx = design(prototype_em,6e9,'Z0',75,'LineLength',0.5);
```

View the `microstipLine` object.

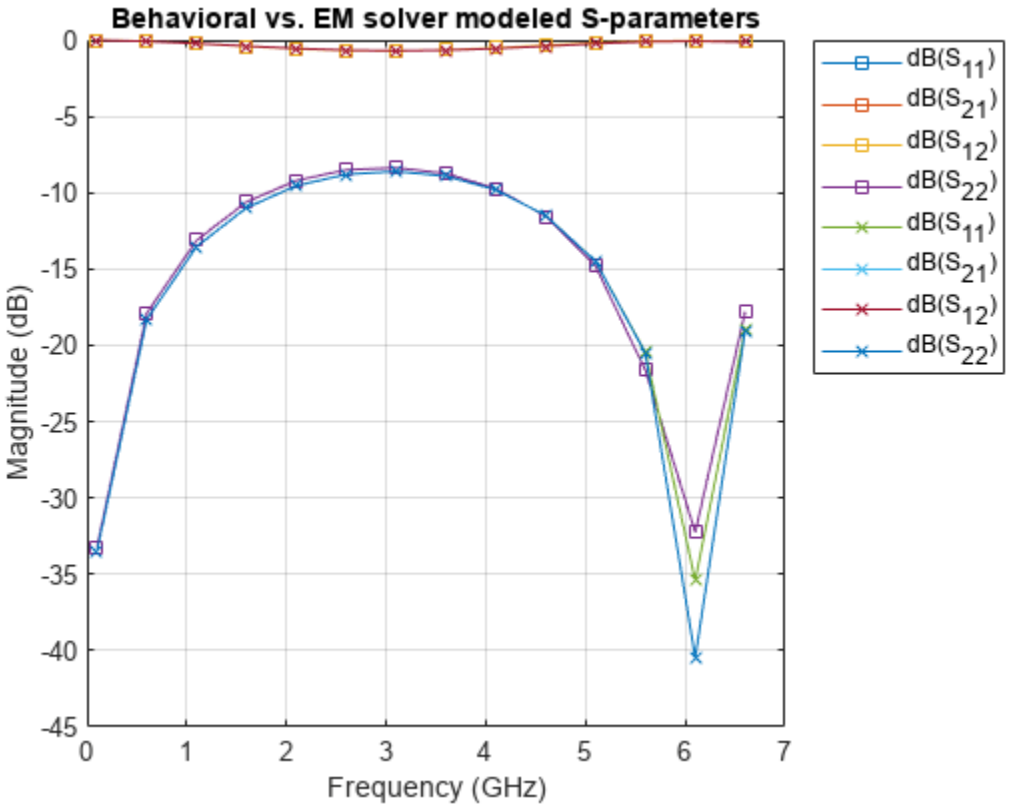
```
show(tx)
```



Plot S-Parameters

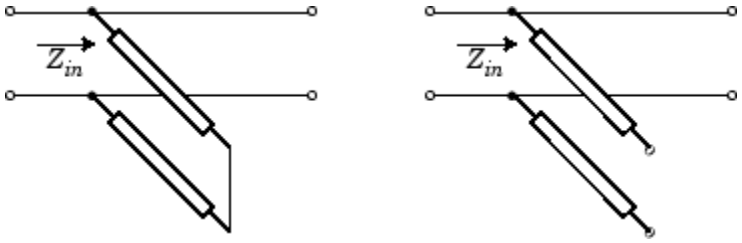
Calculate and plot the behavioral and electromagnetic (EM) solver modeled S-parameters of the line with the reference impedance of 50Ω. Use the `Behavioral` name-value argument of the `sparameters` (RF PCB Toolbox) function to compute the behavioral S-parameters.

```
freq = (1:5:66)*100e6;
Srf = sparameters(tx,freq,50,'Behavioral',true);
Sem = sparameters(tx,freq,50);
rfplot(Srf,'-s','db')
hold on
rfplot(Sem,'-x','db')
title('Behavioral vs. EM solver modeled S-parameters');
```



Algorithms

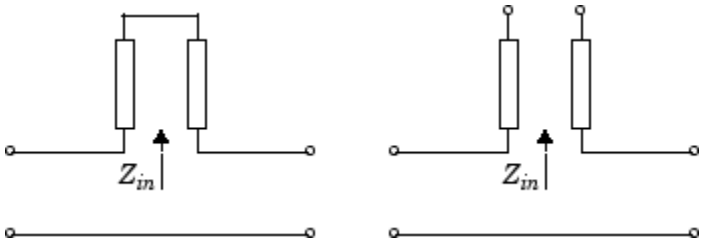
- When you set the `StubMode` property to `'Shunt'`, the 2-port network consists of a stub transmission line that you can terminate with either a short circuit or an open circuit.



Z_{in} is the input impedance of the shunt circuit. The ABCD-parameters for the shunt stub are calculated as:

$$A = 1$$
$$B = 0$$
$$C = 1/Z_{in}$$
$$D = 1$$

When you set the `StubMode` property to `'Series'`, the 2-port network consists of a series transmission line that you can terminate with either a short circuit or an open circuit.



Z_{in} is the input impedance of the series circuit. The ABCD-parameters for the series stub are calculated as:

$$A = 1$$
$$B = Z_{in}$$
$$C = 0$$
$$D = 1$$

Version History

Introduced in R2020b

References

[1] Garg, Ramesh, I. J. Bahl, and Maurizio Bozzi. Microstrip Lines and Slotlines . 3rd ed. Artech House Microwave Library. Boston: Artech House, 2013.

[2] Wadell, Brian C. *Transmission Line Design Handbook*. The Artech House Microwave Library. Boston: Artech House, 1991.

See Also

`microstripLine` | `txlineCoaxial` | `txlineCPW` | `txlineParallelPlate` | `txlineRLCGLine` | `txlineTwoWire`

Topics

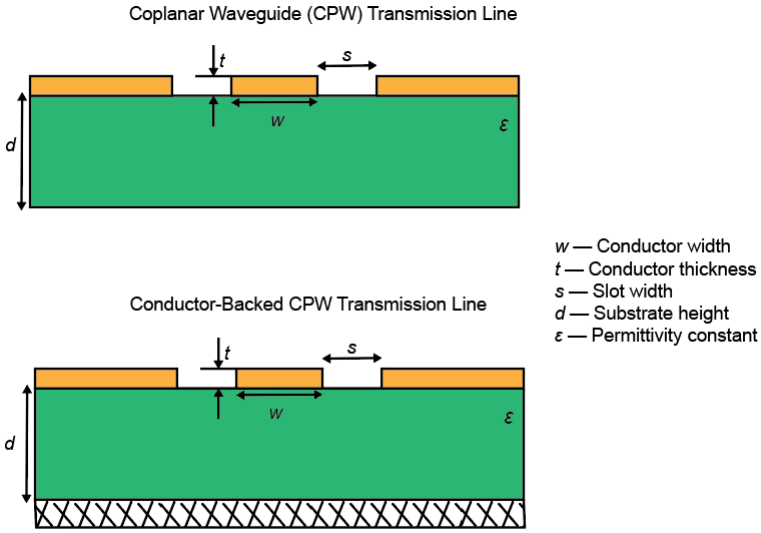
“Design Two-Stage Low Noise Amplifier Using Microstrip Transmission Line Matching Network”

txlineCPW

Create coplanar waveguide transmission line

Description

Use the `txlineCPW` object to create a coplanar waveguide (CPW) transmission line. You can also use the `txlineCPW` object to create an infinite-bottom conductor or ground plane CPW transmission line. The cross-section of a CPW transmission line is shown in this figure. The physical characteristics of the CPW transmission line include conductor width (w), conductor thickness (t), slot width (s), substrate height (d), and permittivity constant (ε).



Creation

Syntax

```
txline = txlineCPW
txline = txlineCPW(Name,Value)
```

Description

`txline = txlineCPW` creates a default CPW transmission line object.

`txline = txlineCPW(Name,Value)` sets "Properties" on page 1-271 using one or more name-value pairs. For example, `txline = txlineCPW('SlotWidth',0.0046)` creates a CPW transmission line with a slot width of `0.0046` meters.

Properties

`Name` — Name of CPW transmission line

`'cpw'` (default) | string scalar | character vector

Name of the CPW transmission line, specified as a string scalar or a character vector.

Example: `'Name','cpw1'`

Example: `txline.Name = 'cpw1'`

Data Types: `char` | `string`

`ConductorWidth` — Physical width

`0.0006` (default) | positive scalar

Physical width of the conductor, specified as a positive scalar in meters.

Example: `'ConductorWidth',0.0200`

Example: `txline.ConductorWidth = 0.0200`

Data Types: `double`

`SlotWidth` — Physical width of slot

`0.0002` (default) | positive scalar

Physical width of the slot, specified as a positive scalar in meters.

Example: `'SlotWidth',0.0008`

Example: `txline.SlotWidth = 0.0008`

Data Types: `double`

`Height` — Physical height of conductor or dielectric thickness

`0.000635` (default) | positive scalar

Physical height of the conductor or dielectric thickness, specified as a positive scalar in meters.

Example: `'Height',0.000835`

Example: `txline.Height = 0.000835`

Data Types: `double`

`Thickness` — Physical thickness

`0.000005` (default) | positive scalar

Physical thickness of the CPW transmission line, specified as a positive scalar in meters.

Example: `'Thickness',0.000008`

Example: `txline Thickness = 0.000008`

Data Types: `double`

`EpsilonR` — Relative permittivity of dielectric

`9.8` (default) | positive scalar

Relative permittivity of the dielectric, specified as a positive scalar.

Example: `'EpsilonR',8.8`

Example: `txline.EpsilonR = 8.8`

Data Types: `double`

`LossTangent` — Loss angle tangent of dielectric
`0` (default) | nonnegative scalar

Loss angle tangent of the dielectric, specified as a nonnegative scalar.

Example: `'LossTangent',1`

Example: `txline.LossTangent = 1`

Data Types: `double`

`SigmaCond` — Conductivity of conductor
`Inf` (default) | nonnegative scalar

Conductivity of the conductor, specified as a nonnegative scalar in Siemens per meter (S/m).

Example: `'SigmaCond',2`

Example: `txline.SigmaCond = 2`

Data Types: `double`

`LineLength` — Physical length
`0.0100` (default) | positive scalar

Physical length of the CPW transmission line, specified as a positive scalar in meters.

Example: `'LineLength',0.0200`

Example: `txline.LineLength = 0.0200`

Data Types: `double`

`ConductorBacked` — Infinite bottom conductor or ground plane
`0 (false)` (default) | `1 (true)`

Infinite-bottom conductor or ground plane, specified as a numeric or logical. When you specify `0`, you create a transmission line without any conductor backing.

Example: `'ConductorBacked',1`

Example: `txline.ConductorBacked = 1`

Data Types: `double`

`Termination` — Stub transmission line termination
`'NotApplicable'` (default) | `'Open'` | `'Short'`

Stub transmission line termination, specified as `'NotApplicable'`, `'Open'`, or `'Short'`.

Example: `'Termination','Short'`

Example: `txline.Termination = 'Short'`

Data Types: `char`

`StubMode` — Type of stub
`'NotAStub'` (default) | `'Series'` | `'Shunt'`

Type of stub, specified as `'NotAStub'`, `'Series'`, or `'Shunt'`.

Example: `'StubMode','Series'`

Example: `txline.StubMode = 'Series'`

Data Types: `char`

`NumPorts` — Number of input and output ports
2 (default) | positive scalar

This property is read-only.

Number of input and output ports, returned as positive scalar.

Data Types: `double`

`Terminals` — Terminals of coplanar waveguide transmission line
`{'p1+' p2+' 'p1-' 'p2-'}` (default) | cell array of strings

This property is read-only.

Terminals of the CPW transmission line, returned as a cell array of strings.

Data Types: `char` | `string`

Object Functions

sparameters	Calculate S-parameters for RF data, network, circuit, and matching network objects
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object
noisefigure	Calculate noise figure of transmission lines, series RLC, and shunt RLC circuits
getZ0	Calculate characteristic impedance with and without dispersion for transmission line
circuit	Circuit object
clone	Create copy of existing circuit element or circuit object

Examples

S-Parameters of Coplanar Waveguide Transmission Line

Create a coplanar waveguide transmission line using these specifications:

- Conductor width : 45 mu m
- Slot width : 50 mu m
- Height of substrate : 525 mu m
- Thickness : 1 mu m
- GaAS permittivity or Epsilon R : 2.5 F/m
- Conductivity : 3.33e7 S/m

```
cpwtxline = txlineCPW('ConductorWidth',45e-6,'SlotWidth',50e-6,'Height',525e-6,...
    'Thickness',1e-6,'EpsilonR',2.5,'SigmaCond',3.33e7)
```

```
cpwtxline =
  txlineCPW: CPW element

              Name: 'CPW'
    ConductorWidth: 4.5000e-05
         SlotWidth: 5.0000e-05
            Height: 5.2500e-04
         Thickness: 1.0000e-06
          EpsilonR: 2.5000
       LossTangent: 0
         SigmaCond: 33300000
        LineLength: 0.0100
   ConductorBacked: 0
       Termination: 'NotApplicable'
          StubMode: 'NotAStub'
          NumPorts: 2
         Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Calculate the S-parameters of the transmission line at 20 GHz.

```
sparam = sparameters(cpwtxline,20e9);
```

Behavioral vs EM Solver Model of CPW Transmission Line

This example uses RF PCB Toolbox to calculate an electromagnetic (EM) solver S-parameter model of a conductor-backed coplanar waveguide (CPW) transmission line.

Behavioral Modeling

Select the FR4 dielectric and silver metal the metal and dielectric libraries, respectively, of RF PCB Toolbox.

```
dFR4 = dielectric('FR4');
mSilver = metal('Silver');
```

Design an infinite-bottom conductor-backed CPW transmission line at 6 GHz and Z_0 of 75Ω.

```
tx = txlineCPW('ConductorBacked',true, ...
    'LineLength',0.0145,'ConductorWidth',2.6482e-4, ...
    'SlotWidth',2e-4,'Height',6.35e-4, ...
    "EpsilonR",dFR4.EpsilonR,"LossTangent",dFR4.LossTangent, ...
    "SigmaCond",mSilver.Conductivity,'Thickness',2e-7, ...
    'StubMode',"NotAStub","Termination","NotApplicable");
```

Calculate the S-parameters using a behavioral model from RF Toolbox with the reference impedance of 50Ω. The behavioral model is also called an analytical or closed-form model.

```
freq = (1:5:66)*100e6;
Srf = sparameters(tx,freq,50);
```

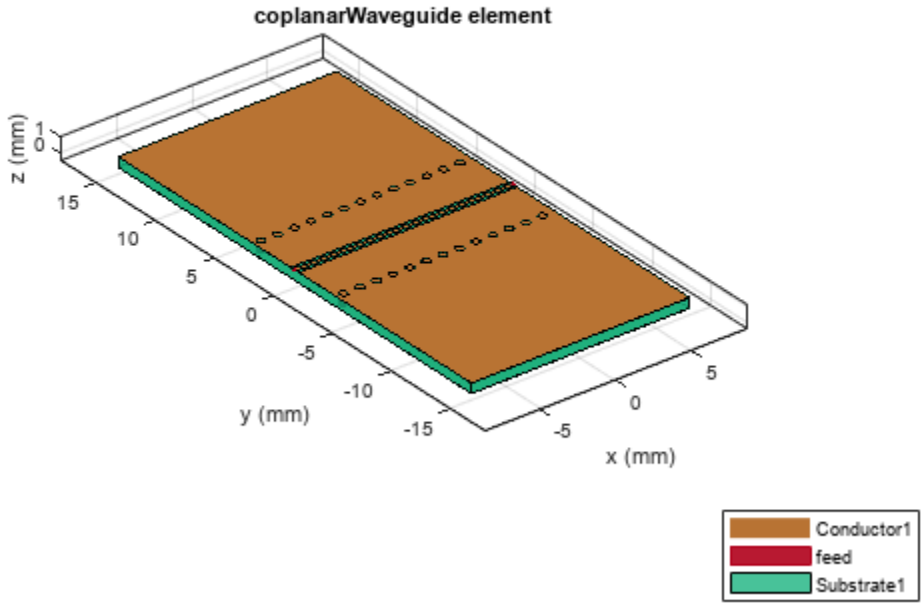
EM Modeling

Create an RF PCB Toolbox `coplanarWaveguide` object.

```
tx_em = coplanarWaveguide(tx);
```

View the conductor-backed CPW transmission line.

```
show(tx_em)
```



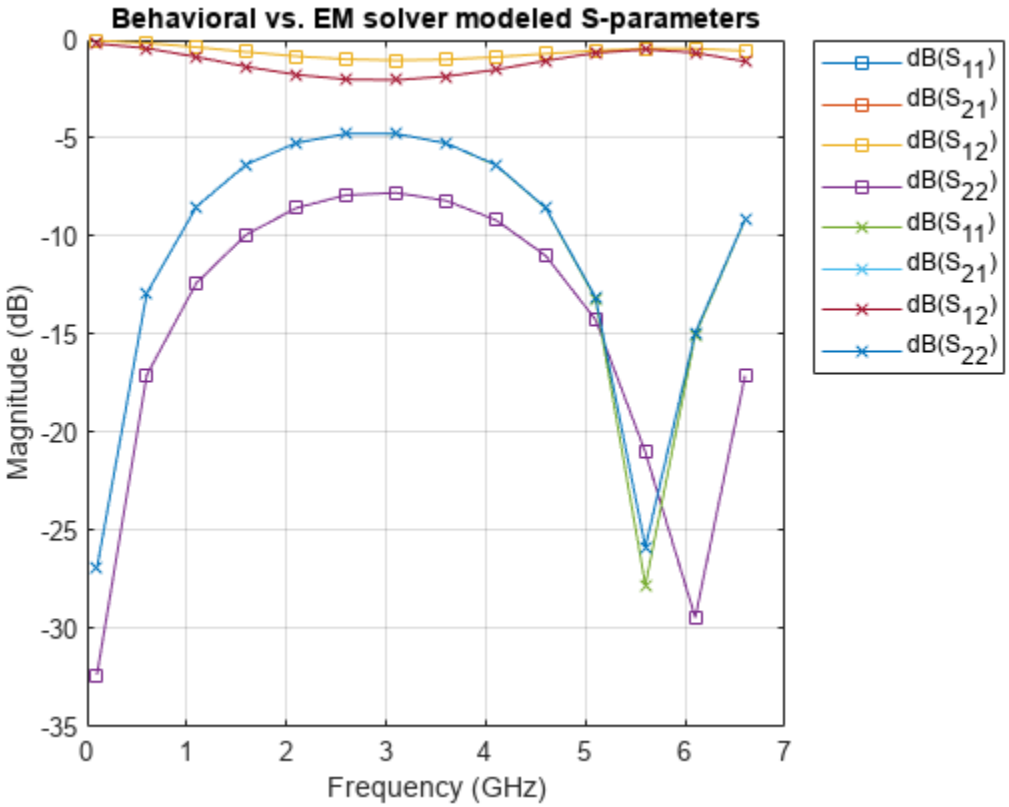
Calculate the S-parameters using the EM solver from RF PCB Toolbox.

```
Sem = sparameters(tx_em,freq,50);
```

Compare S-parameters

Compare the S-parameters from the behavioral and EM solver models.

```
rfplot(Srf,'-s','db')
hold on
rfplot(Sem,'-x','db')
title('Behavioral vs. EM solver modeled S-parameters')
```



Version History

Introduced in R2020b

Coplanar waveguide transmission line object changed

Behavior changed in R2021b

Starting in R2021b, the `txlineCPW` circuit object has a new property, `ConductorBacked`. Use this property to design a CPW transmission line with an infinite-bottom conductor or ground plane.

When you open a model created before R2021b containing a `txlineCPW` circuit object, the software replaces the object with the R2021b version of the object and sets the `ConductorBacked` property to `0`.

References

[1] Garg, Ramesh, I. J. Bahl, and Maurizio Bozzi. Microstrip Lines and Slotlines . 3rd ed. Artech House Microwave Library. Boston: Artech House, 2013.

See Also

`coplanarWaveguide` | `txlineCoaxial` | `txlineMicrostrip` | `txlineParallelPlate` | `txlineRLCGLine` | `txlineTwoWire`

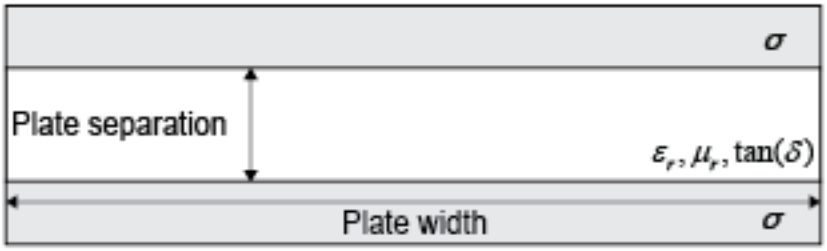
Topics
“Design Two-Stage Low Noise Amplifier Using Microstrip Transmission Line Matching Network”

txlineParallelPlate

Create parallel-plate transmission line

Description

Use the `txlineParallelPlate` object to create a parallel-plate transmission line. A cross-section of a parallel-plate transmission line is shown this figure. The physical characteristics of the parallel-plate transmission line include the plate width and the plate separation.



Creation

Syntax

```
paralleltxline = txlineParallelPlate
paralleltxline = txlineParallelPlate(Name,Value)
```

Description

`paralleltxline = txlineParallelPlate` creates a parallel-plate transmission line object.

`paralleltxline = txlineParallelPlate(Name,Value)` sets properties using one or more name-value pairs. For example, `txline = txlineParallelPlate('Separation',0.0046)` creates a parallel-plate transmission line with a separation of `0.0046` meters.

Properties

`Name` — Name of parallel-plate transmission line

`'parallelplate'` (default) | string scalar | character vector

Name of the parallel-plate transmission line, specified as a string scalar or a character vector.

Example: `'Name','parallelplate1'`

Example: `paralleltxline.Name = 'parallelplate1'`

Data Types: `char` | `string`

`LineLength` — Physical length
`'0.0100'` (default) | positive scalar

Physical length of the transmission line, specified as a positive scalar in meters.

Example: `'LineLength',0.0200`

Example: `paralletxline.LineLength = 0.0200`

Data Types: `double`

`Width` — Physical width
`0.0050` (default) | positive scalar

Physical width of the transmission line, specified as a positive scalar in meters.

Example: `'Width',0.00200`

Example: `paralleltxline.Width = 0.00200`

Data Types: `double`

`Separation` — Thickness of dielectric
`0.001` (default) | positive scalar

Thickness of dielectric, specified as a positive scalar in meters.

Example: `'Separation',0.007`

Example: `paralleltxline.Separation = 0.007`

Data Types: `double`

`MuR` — Relative permeability of dielectric
`1` (default) | positive scalar

Relative permeability of a dielectric, specified as a positive scalar.

The ratio of permeability of dielectric, μ, to the permeability in free space, μ_0.

Example: `'MuR',1.5`

Example: `paralleltxline.MuR = 1.5`

Data Types: `double`

`EpsilonR` — Relative permittivity of dielectric
`2.3` (default) | positive scalar

Relative permittivity of the dielectric, specified as a positive scalar in Farad per meter (F/m).

Example: `'EpsilonR',3.3`

Example: `paralleltxline.EpsilonR = 3.3`

Data Types: `double`

`LossTangent` — Loss angle tangent of dielectric
`0` (default) | nonnegative scalar

Loss angle tangent of the dielectric, specified as a nonnegative scalar.

Example: `'LossTangent',1`

Example: `paralleltxline.LossTangent = 1`

Data Types: `double`

`SigmaCond` — Conductivity of conductor
`Inf` (default) | positive scalar

Conductivity of the conductor, specified as a positive scalar in Siemens per meter (S/m).

Example: `'SigmaCond',2`

Example: `paralleltxline.SigmaCond = 2`

Data Types: `double`

`Termination` — Stub transmission line termination
`'NotApplicable'` (default) | `'Open'` | `'Short'`

Stub transmission line termination, specified as `'NotApplicable'`, `'Open'`, or `'Short'`.

Example: `'Termination','Short'`

Example: `paralleltxline.Termination = 'Short'`

Data Types: `char`

`StubMode` — Type of stub
`'NotAStub'` (default) | `'Series'` | `'Shunt'`

Type of stub, specified as `'NotAStub'`, `'Series'`, or `'Shunt'`.

Example: `'StubMode','Series'`

Example: `paralleltxline.StubMode = 'Series'`

Data Types: `char`

`NumPorts` — Number of input and output ports
2 (default) | positive scalar

This property is read-only.

Number of input and output ports, specified as positive scalar.

Data Types: `double`

`Terminals` — Terminals of coaxial transmission line
`{'p1+' p2+' 'p1-' 'p2-'}` (default) | cell array of strings

This property is read-only.

Terminals of coaxial transmission line, specified as a cell array of strings.

Data Types: `char` | `string`

Object Functions

sparameters	Calculate S-parameters for RF data, network, circuit, and matching network objects
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object
noisefigure	Calculate noise figure of transmission lines, series RLC, and shunt RLC circuits
getZ0	Calculate characteristic impedance with and without dispersion for transmission line
circuit	Circuit object
clone	Create copy of existing circuit element or circuit object

Examples

S-Parameters of Parallel-Plate Transmission Line

Create a parallel-plate transmission line using these specifications:

- Width : 300 mm
- Separation between plates : 10 mm
- EpsilonR : 4.2 F/m

```
parallelplatetxline = txlineParallelPlate('Width',300e-3,'Separation',10e-3,'EpsilonR',4.2)
```

```
parallelplatetxline =
  txlineParallelPlate: ParallelPlate element

           Name: 'ParallelPlate'
          Width: 0.3000
     Separation: 0.0100
            MuR: 1
       EpsilonR: 4.2000
    LossTangent: 0
      SigmaCond: Inf
     LineLength: 0.0100
    Termination: 'NotApplicable'
       StubMode: 'NotAStub'
       NumPorts: 2
      Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Calculate the S-parameters of the transmission line at 6 GHz.

```
sparam = sparameters(parallelplatetxline,6e9);
```

Version History

Introduced in R2020b

See Also

`txlineCoaxial` | `txlineCPW` | `txlineMicrostrip` | `txlineRLCGLine` | `txlineTwoWire`

Topics

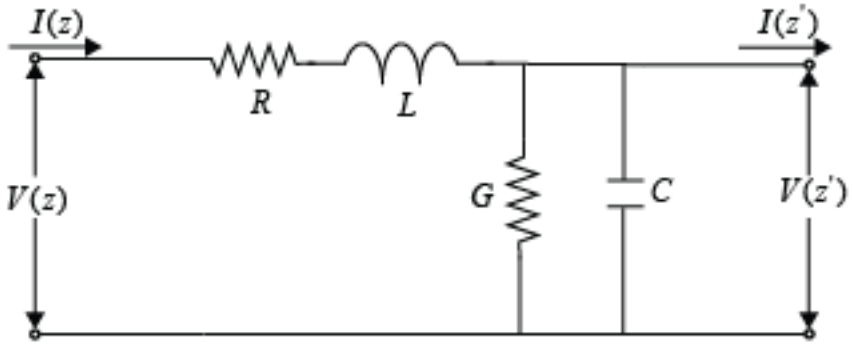
“Design Two-Stage Low Noise Amplifier Using Microstrip Transmission Line Matching Network”

txlineRLCGLine

Create RLCG line transmission line

Description

Use the `txlineRLCGLine` object to create a RLCG transmission line that is characterized by line loss, line length, stub type, and termination.



Creation

Syntax

```
rlcgtxline = txlineRLCGLine
rlcgtxline = txlineRLCGLine(Name=Value)
```

Description

`rlcgtxline = txlineRLCGLine` creates a default RLCG line transmission line object.

`rlcgtxline = txlineRLCGLine(Name=Value)` sets "Properties" on page 1-282 using one or more name-value pairs. For example, `rlcgtxline = txlineRLCGLine(Freq=2.0e9)` creates an RLCG line transmission line at a frequency of 2 GHz.. Properties you do not specify retain their default values.

Properties

Name — Name of RLCG line transmission line

`'rlcgline'` (default) | string scalar | character vector

Name of the RLCG line transmission line, specified as a string scalar or a character vector.

Example: `Name='rlcgline1'`

Example: `rlcgtxline.Name = 'rlcgline1'`

Data Types: `char` | `string`

Freq — Frequency

`1.0e9` (default) | nonnegative scalar or vector

Frequency of the RLCG of the transmission line, specified as a nonnegative scalar in hertz.

Example: `Freq=2.0e9`

Example: `rlcgtxline.Freq = 2.0e9`

Data Types: `double`

R — Resistor values

`0` (default) | nonnegative scalar or vector

Resistor values per length of the transmission line, specified as a nonnegative scalar in ohms per meter or a nonnegative vector with each element unit in ohms per meter.

Example: `R=0.5`

Example: `rlcgtxline.R = 0.5`

Data Types: `double`

L — Inductance values

`0` (default) | nonnegative scalar or vector

Inductance values per length of the transmission line, specified as a nonnegative scalar in Henry per meter or a nonnegative vector with each element unit in Henry per meter.

Example: `L=0.5`

Example: `rlcgtxline.L = 0.5`

Data Types: `double`

C — Capacitance values

`0` (default) | nonnegative scalar or vector

Capacitance values per length of the transmission line, specified as a nonnegative scalar in Faraday per meter or a nonnegative vector with each element unit in Faraday per meter.

Example: `C=0.5`

Example: `rlcgtxline.C = 0.5`

Data Types: `double`

G — Conductance values

`0` (default) | nonnegative scalar or vector

Conductance values per length of the transmission line, specified as a nonnegative scalar in Siemens per meter or a nonnegative vector with each element unit in Siemens per meter.

Example: `G=0.5`

Example: `rlcgtxline.G = 0.5`

Data Types: `double`

`IntpType` — Type of interpolation
`'Linear'` (default) | `'Spline'` | `'Cubic'`

Type of interpolation, specified as `'Linear'`, `'Spline'`, or `'Cubic'`.

Example: `IntpType='Spline'`

Example: `rlcgtxline.IntpType = 'Spline'`

Data Types: `char`

`LineLength` — Physical length
`'0.0100'` (default) | positive scalar

Physical length of the transmission line, specified as a positive scalar in meters.

Example: `LineLength=,0.0200`

Example: `rlcgtxline.LineLength = 0.0200`

Data Types: `double`

`Termination` — Stub transmission line termination
`'NotApplicable'` (default) | `'Open'` | `'Short'`

Stub transmission line termination, specified as `'NotApplicable'`, `'Open'`, or `'Short'`.

Example: `Termination='Short'`

Example: `rlcgtxline.Termination = 'Short'`

Data Types: `char`

`StubMode` — Type of stub
`'NotAStub'` (default) | `'Series'` | `'Shunt'`

Type of stub, specified as `'NotAStub'`, `'Series'`, or `'Shunt'`.

Example: `StubMode='Series'`

Example: `rlcgtxline.StubMode = 'Series'`

Data Types: `char`

`NumPorts` — Number of input and output ports
2 (default) | positive scalar

This property is read-only.

Number of input and output ports, specified as positive scalar.

Data Types: `double`

`Terminals` — Terminals of coaxial transmission line
`{'p1+' p2+' 'p1-' 'p2-'}` (default) | cell array of strings

This property is read-only.

Number of input and output ports, specified as a cell array of strings.

Data Types: `char` | `string`

Object Functions

sparameters	Calculate S-parameters for RF data, network, circuit, and matching network objects
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object
noisefigure	Calculate noise figure of transmission lines, series RLC, and shunt RLC circuits
getZ0	Calculate characteristic impedance with and without dispersion for transmission line
circuit	Circuit object
clone	Create copy of existing circuit element or circuit object

Examples

S-parameters of RLCG Transmission Line

Create an RLCG transmission line using these specifications:

- Resistor : 100 ohms
- Capacitor : 1 pF

```
rlcglinetxline = txlineRLCGLine(R=100,C=1e-12)
```

```
rlcglinetxline =
  txlineRLCGLine: RLCGLine element

           Name: 'RLCGLine'
      Frequency: 1.0000e+09
              R: 100
              L: 0
              C: 1.0000e-12
              G: 0
       IntpType: 'Linear'
     LineLength: 0.0100
    Termination: 'NotApplicable'
       StubMode: 'NotAStub'
       NumPorts: 2
      Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Calculate the S-parameters of the transmission line at 1 GHz.

```
sparam = sparameters(rlcglinetxline,1e9);
```

Version History

Introduced in R2020b

See Also

`txlineCoaxial` | `txlineCPW` | `txlineMicrostrip` | `txlineParallelPlate` | `txlineTwoWire`

Topics

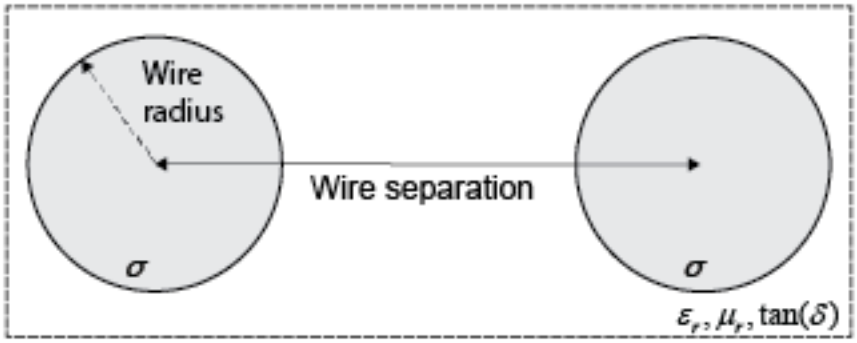
“Design Two-Stage Low Noise Amplifier Using Microstrip Transmission Line Matching Network”

txlineTwoWire

Create two-wire transmission line

Description

Use the `txlineTwoWire` object to create a two-wire transmission line. A cross-section of a two-wire transmission line is shown in this figure. The physical characteristics of a two-wire transmission line include the radii of the conducting wires, the separation or physical distance between the wire centers, and the relative permittivity and permeability of the wires. RF Toolbox software assumes that the relative permittivity and permeability are uniform.



Creation

Syntax

```
twowiretxline = txlineTwoWire
twowiretxline = txlineTwoWire(Name,Value)
```

Description

`twowiretxline = txlineTwoWire` creates a default two-wire transmission line object.

`twowiretxline = txlineTwoWire(Name,Value)` sets "Properties" on page 1-286 using one or more name-value pairs. For example, `txline = txlineTwoWire('Separation',0.0046)` creates a two-wire transmission line with a dielectric thickness of `0.0046` meters.

Properties

Name — Name of two-wire transmission line

`'twowire'` (default) | string scalar | character vector

Name of the two-wire transmission line, specified as a string scalar or a character vector.

Example: `'Name','twowire1'`

Example: `twowiretxline.Name = 'twowire1'`

Data Types: `char` | `string`

`LineLength` — Physical length
`'0.0100'` (default) | positive scalar

Physical length of the transmission line, specified as a positive scalar in meters.

Example: `'LineLength',0.0200`

Example: `twowiretxline.LineLength = 0.0200`

Data Types: `double`

`Radius` — Conducting wire radius
`0.000670` (default) | positive scalar

Conducting wire radius in the two-wire transmission line, specified as a positive scalar in meters.

Example: `'Radius',0.000970`

Example: `twowiretxline.Radius = 0.000970`

Data Types: `double`

`Separation` — Thickness of dielectric
`0.0016` (default) | positive scalar

Thickness of the dielectric, specified as a positive scalar in meters.

Example: `'Separation',0.0025`

Example: `twowiretxline.Separation = 0.0025`

Data Types: `double`

`MuR` — Relative permeability of dielectric
`1` (default) | positive scalar

Relative permeability of the dielectric, specified as a positive scalar. Relative permeability of the dielectric, μ, to the permeability in free space, μ_0.

Example: `'MuR',1.5`

Example: `twowiretxline.MuR = 1.5`

Data Types: `double`

`EpsilonR` — Relative permittivity of dielectric
`2.3` (default) | positive scalar

Relative permittivity of the dielectric, specified as a positive scalar.

Example: `'EpsilonR',3.3`

Example: `twowiretxline.EpsilonR = 3.3`

Data Types: `double`

`LossTangent` — Loss angle tangent of dielectric
`0` (default) | nonnegative scalar

Loss angle tangent of the dielectric, specified as a nonnegative scalar in degrees

Example: `'LossTangent',1`

Example: `twowiretxline.LossTangent = 1`

Data Types: `double`

`SigmaCond` — Conductivity of conductor

`Inf` (default) | scalar

Conductivity of the conductor, specified as a scalar in Siemens per meter (S/m).

Example: `'SigmaCond',2`

Example: `twowiretxline.SigmaCond = 2`

Data Types: `double`

`Termination` — Stub transmission line termination

`'NotApplicable'` (default) | `'Open'` | `'Short'`

Stub transmission line termination, specified as `'NotApplicable'`, `'Open'` or `'Short'`.

Example: `'Termination','Short'`

Example: `twowiretxline.Termination = 'Short'`

Data Types: `char`

`StubMode` — Type of stub

`'NotAStub'` (default) | `'Series'` | `'Shunt'`

Type of stub, specified as `'NotAStub'`, `'Series'` or `'Shunt'`.

Example: `'StubMode','Series'`

Example: `twowiretxline.StubMode = 'Series'`

Data Types: `char`

`NumPorts` — Number of input and output ports

2 (default) | positive scalar

This property is read-only.

Number of input and output ports, specified as a positive scalar.

Data Types: `double`

`Terminals` — Terminals of coaxial transmission line

`{'p1+' p2+' 'p1-' 'p2-'}` (default) | cell array of strings

This property is read-only.

Terminals of coaxial transmission line, specified as a cell array of strings.

Data Types: `char` | `string`

Object Functions

sparameters	Calculate S-parameters for RF data, network, circuit, and matching network objects
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object
noisefigure	Calculate noise figure of transmission lines, series RLC, and shunt RLC circuits
getZ0	Calculate characteristic impedance with and without dispersion for transmission line
circuit	Circuit object
clone	Create copy of existing circuit element or circuit object

Examples

Group Delay and Noise Figure of Two-Wire Transmission Line

Create a two-wire transmission line using these specifications:

- Radius - 0.5 mm
- Dielectric - air
- Thickness of dielectric or separation - 1.088 mm
- Permittivity or `EpsilonR` - 1.0054

```
twowiretxline = txlineTwoWire('Radius',0.5e-3,'EpsilonR',1.0054,'Separation',1.088e-3);
```

Calculate the noise figure and the group delay of the transmission line at 2.5 GHz.

```
nf = noisefigure(twowiretxline,2.5e9)
```

```
nf = 0
```

```
gd = groupdelay(twowiretxline,2.5e9)
```

```
gd = 3.3446e-11
```

Version History

Introduced in R2020b

See Also

`txlineCoaxial` | `txlineCPW` | `txlineMicrostrip` | `txlineParallelPlate` | `txlineRLCGLine`

Topics

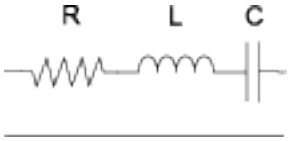
“Design Two-Stage Low Noise Amplifier Using Microstrip Transmission Line Matching Network”

seriesRLC

Create two-port series RLC circuit

Description

Use the `seriesRLC` object to create a circuit representing as a resistor, inductor, and capacitor connected in series. The series RLC circuit object is a two-port network as shown in this circuit diagram.



You can also use the `seriesRLC` object to model a series RLC circuit in cascaded stages using an `rfbudget` object or the **RF Budget Analyzer** app.

Creation

Syntax

```
rlc = seriesRLC
rlc = seriesRLC(Name,Value)
```

Description

`rlc = seriesRLC` creates a default series RLC object.

`rlc = seriesRLC(Name,Value)` sets “Properties” on page 1-290 of the `serisRLC` object using one or more name-value arguments. For example, `rlc = seriesRLC('R',80)` creates a series RLC object with the resistance set to `80` ohms. Properties you do not specify retain their default values.

Properties

Name — Name of series RLC circuit

`seriesRLC` (default) | string scalar | character vector

Name of the series RLC circuit, specified as a string scalar or a character vector.

Example: `rlc = seriesRLC('Name','RLCseriescircuit')` creates a series RLC circuit called `RLCseriescircuit`.

R — Resistance value

`0` (default) | positive scalar

Resistance value of the series RLC circuit, specified as a positive scalar in ohms.

Example: `rlc = seriesRLC('R',75)` creates a series RLC circuit with the resistance set to 75 ohms.

`L` — Inductance value
`0` (default) | positive scalar

Inductance value of the series RLC circuit, specified as a positive scalar in henries.

Example: `rlc = seriesRLC('L',1e-6)` creates a series RLC circuit with the inductance set to `1e-6` henries.

`C` — Capacitance value
`0` (default) | positive scalar

Capacitance value of the series RLC circuit, specified as a positive scalar in farads.

Example: `rlc = seriesRLC('C',2.2e-9)` creates a series RLC circuit with the capacitance set to `2.2e-9` farads.

`NumPorts` — Number of input and output ports
`2` (default) | positive scalar

This property is read-only.

Number of input and output ports of the series RLC circuit, returned as a positive scalar.

`Terminals` — Terminals of series RLC circuit
`{'p1+' 'p2+' 'p1-' 'p2-'}` (default) | cell array

This property is read-only.

Terminals of the series RLC circuit, returned as a cell array.

Object Functions

sparameters	Calculate S-parameters for RF data, network, circuit, and matching network objects
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object
noisefigure	Calculate noise figure of transmission lines, series RLC, and shunt RLC circuits

Examples

Calculate S-Parameters and Noise Figure of Series RLC Circuit

Create a series RLC circuit with the specified parameters.

```
rlc = seriesRLC('R',2e3,'L',40e-3,'C',1e-6);
```

Calculate the S-parameters of the series RLC circuit at 1 GHz.

```
spar = sparameters(rlc,1e9)
```

```
spar =
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: 1.0000e+09
```

```
     Parameters: [2x2 double]
      Impedance: 50

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

Calculate the noise figure of the series RLC circuit at 1 GHz.

```
nf = noisefigure(rlc,1e9)
```

```
nf = 16.1278
```

Version History

Introduced in R2021a

See Also

`shuntRLC`

shuntRLC

Create two-port shunt RLC circuit

Description

Use the `shuntRLC` object to create a circuit representing a resistor, inductor, and capacitor connected in parallel or shunt. The shunt RLC circuit object is a two-port network as shown in this circuit diagram.

You can also use the `shuntRLC` object to model a shunt RLC circuit in cascaded stages using the `rfbudget` object or the **RF Budget Analyzer** app.

Creation

Syntax

```
rlc = shuntRLC
rlc = shuntRLC(Name,Value)
```

Description

`rlc = shuntRLC` creates a default shunt RLC object.

`rlc = shuntRLC(Name,Value)` sets "Properties" on page 1-293 of the `shuntRLC` object using one or more name-value arguments. For example, `rlc = shuntRLC('R',80)` creates a shunt RLC object with the resistance set to `80` ohms. Properties you do not specify retain their default values.

Properties

`Name` — Name of shunt RLC circuit

`shuntRLC` (default) | string scalar | character vector

Name of the shunt RLC circuit, specified as a string scalar or a character vector.

Example: `rlc = shuntRLC('Name','RLCshuntcircuit')` creates a shunt RLC circuit called `RLCshuntcircuit`.

`R` — Resistance value

`inf` (default) | positive scalar

Resistance value of the shunt RLC circuit, specified as a positive scalar in ohms.

Example: `rlc = shuntRLC('R',75)` creates a shunt RLC circuit with the resistance set to `75` ohms.

L — Inductance value

`inf` (default) | positive scalar

Inductance value of the shunt RLC circuit, specified as a positive scalar in henries.

Example: `rlc = shuntRLC('L',1e-6)` creates a shunt RLC circuit with the inductance set to `1e-6` henries.

C — Capacitance value

`0` (default) | positive scalar

Capacitance value of the shunt RLC circuit, specified as a positive scalar in farads.

Example: `rlc = shuntRLC('C',2.2e-9)` creates a shunt RLC circuit with the capacitance set to `2.2e-9` farads.

NumPorts — Number of input and output ports

`2` (default) | positive scalar

This property is read-only.

Number of input and output ports of the shunt RLC circuit, returned as a positive scalar.

Terminals — Terminals of shunt RLC circuit

`{'p1+' 'p2+' 'p1-' 'p2-'}` (default) | cell array

This property is read-only.

Terminals of the shunt RLC circuit, returned as a cell array.

Object Functions

sparameters	Calculate S-parameters for RF data, network, circuit, and matching network objects
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object
noisefigure	Calculate noise figure of transmission lines, series RLC, and shunt RLC circuits

Examples

Calculate S-Parameters and Noise Figure of Shunt RLC Circuit

Create a shunt RLC circuit with the specified parameters.

```
rlc = shuntRLC('R',60,'L',1e-3,'C',50e-6);
```

Calculate the S-parameters of the shunt RLC circuit at 1 GHz.

```
spar = sparameters(rlc,1e9)
```

```
spar =
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: 1.0000e+09
     Parameters: [2x2 double]
      Impedance: 50
```

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

Calculate the noise figure of the shunt RLC circuit at 1 GHz.

```
nf = noisefigure(rlc,1e9)
```

```
nf = 2.6324
```

Version History

Introduced in R2021a

See Also

`seriesRLC`

attenuator

Create two-port attenuator element

Description

Use the `attenuator` object to attenuate the signal power by the insertion loss or attenuation factor that you specify. You can also use the `attenuator` object to compute attenuation in cascaded stages using an `rfbudget` object or the **RF Budget Analyzer** app.

Creation

Syntax

```
att = attenuator
att = attenuator(Name,Value)
```

Description

`att = attenuator` creates a default attenuator object with the attenuation of 3 dB.

`att = attenuator(Name,Value)` sets "Properties" on page 1-296 of the `attenuator` object using one or more name-value arguments. For example, `att = attenuator('Attenuation',3.2)` creates an attenuator object with attenuation set to `3.2` dB. Properties you do not specify retain their default values.

Properties

`Name` — Name of attenuator element

`Attenuator` (default) | string scalar | character vector

Name of the attenuator element, specified as a string scalar or a character vector.

Example: `att = attenuator('Name','Attenuator1')` creates an attenuator element called `Attenuator1`.

`Attenuation` — Insertion loss or attenuation

`3` (default) | scalar

Insertion loss or attenuation applied to the signal, specified as a scalar in dB.

Example: `att = attenuator('Attenuation',20)` creates a 20 dB attenuator element.

`Zin` — Input impedance

`50` (default) | positive scalar

Input impedance of the attenuator element, specified as a positive scalar in ohms.

Example: `att = attenuator('Zin',200)` creates an attenuator element with the input impedance of `200` ohms.

`Zout` — Output impedance

`50` (default) | positive scalar

Output impedance of the attenuator element, specified as a positive scalar in ohms.

Example: `att = attenuator('Zout',200)` creates an attenuator element with the output impedance of `200` ohms.

`NumPorts` — Number of input and output ports

2 (default) | positive scalar

This property is read-only.

Number of input and output ports in attenuator element, returned as a positive scalar.

`Terminals` — Terminals of attenuator element

`{'p1+' 'p2+' 'p1-' 'p2-'}` (default) | cell array

This property is read-only.

Terminals of the attenuator element, returned as a cell array.

Object Functions

sparameters Calculate S-parameters for RF data, network, circuit, and matching network objects

Examples

Design and Calculate S-Parameters of Attenuator Element

Design an attenuator element to reduce the amplitude level of an audio signal by 18 dB while matching the impedance of a network of `600` ohms.

```
att = attenuator('Attenuation',18,'Zin',600,'Zout',600);
```

Calculate the S-parameters of the attenuator element at 3 KHz.

```
spar = sparameters(att,3e3)
```

```
spar =
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: 3000
     Parameters: [2x2 double]
      Impedance: 50

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

Version History

Introduced in R2021a

See Also

`seriesRLC` | `shuntRLC`

rfantenna

Create antenna element modeled as transmitter, receiver, or transmit-receive

Description

Use the `rfantenna` object to create an antenna element that you can model as a transmitter, receiver, or transmit-receive configuration. If you design an RF system using an `rfbudget` object or the **RF Budget Analyzer** app you can model an RF antenna with the `rfantenna` object and add it to the RF system. You can export the element to RF Blockset or to an `rfsystem` System object for circuit envelope analysis.

Creation

Syntax

```
ant = rfantenna
ant = rfantenna(Name=Value)
```

Description

`ant = rfantenna` creates a default RF antenna object with a gain of `1` dBi.

`ant = rfantenna(Name=Value)` sets the “Properties” on page 1-299 of an `rfantenna` object using one or more name-value arguments. For example, `ant = attenuator(Gain=10)` creates an RF antenna object with a `10` dBi gain. Properties you do not specify retain their default values.

Properties

`Name` — Name of RF antenna element

`Antenna` (default) | string scalar | character vector

Name of the RF antenna element, specified as a string scalar or a character vector.

Example: `rfantenna(Name='Antenna_20dB')`

`Gain` — Antenna Gain

`0` | `[0 0]` | nonnegative scalar | two-element vector

Antenna gain, specified as either: nonnegative scalar (default value is `0`) or two-element vector (default value is `[0 0]`) in dBi. Specify `Gain` as a two-element vector when you set the `Type` property to `TransmitReceive`. To control the effective isotropically radiated power (EIRP), vary the gain of the antenna element.

Example: `rfantenna(Gain=20)`

`Z` — Input impedance

`50` | `[50 50]` | nonnegative scalar | two-element vector

Input impedance of the RF antenna element, specified as either: positive scalar (default value is `50`) or two-element vector (default value is `[50 50]`) in ohms. Specify `Z` as a two-element vector when you set the `Type` property to `TransmitReceive`.

Example: `rfantenna(Z=20)`

Type — Type of antenna element
`'Transmitter'` (default) | `'Receiver'` | `'TransmitReceive'`

Type of the antenna element, specified as a `'Transmitter'`, `'Receiver'`, or `'TransmitReceive'`. When you set `Type` to `'TransmitReceive'`, the transmitter and receiver antennas operate simultaneously.

Example: `rfantenna(Type='Receiver')`

TxEIRP — EIRP value of transmitting antenna
`-30` (default) | numerical scalar

EIRP value of the transmitting antenna that the receiver is tuned to, specified as a numerical scalar in dBm. You must set this property when you are designing a receiver antenna element.

Example: `rfantenna(TxEIRP=24)`

PathLoss — Loss encountered by signal before reaching receiver
`0` (default) | nonnegative scalar

Loss encountered by a signal before reaching the receiver, specified as a nonnegative scalar in dB.

Example: `rfantenna(PathLoss=2)`

NumPorts — Number of input and output ports
`1` (default) | positive scalar

This property is read-only.

Number of input and output ports in RF antenna element, specified as a positive scalar.

Terminals — Terminals of RF antenna element
`{'p1+' 'p1-' }` (default) | cell array

This property is read-only.

Terminals of RF antenna, specified as a cell array.

Object Functions

sparameters Calculate S-parameters for RF data, network, circuit, and matching network objects

Examples

Calculate EIRP Using `rfantenna` Object

Create an amplifier with the gain of 4 dB.

```
a = amplifier(Gain=4);
```

Create a modulator with OIP3 of 13 dBm.

```
m = modulator(OIP3=13);
```

Create an n-port element using `passive.s2p`.

```
n = nport('passive.s2p');
```

Create an RF antenna with the gain of 10 dB.

```
ant = rfantenna(Gain=10);
```

Calculate the RF budget of a series of RF elements at the input frequency of 2.1 GHz, available input power of –30 dBm, and bandwidth of 10 MHz. EIRP is computed under `Analyzed Results` of the `rfbudget` object.

```
b = rfbudget([a m n ant],2.1e9,-30,10e6)
```

```
b =
  rfbudget with properties:

              Elements: [1x4 rf.internal.rfbudget.Element]
        InputFrequency: 2.1 GHz
    AvailableInputPower: -30 dBm
        SignalBandwidth:  10 MHz
                Solver: Friis
            AutoUpdate: true

   Analysis Results
         OutputFrequency: (GHz) [  2.1    3.1      3.1      3.1]
             OutputPower: (dBm) [  -26    -26    -30.6    -30.6]
          TransducerGain: (dB)  [    4      4  -0.5995  -0.5995]
                      NF: (dB)  [    0      0    1.224    1.224]
                    IIP2: (dBm) []
                    OIP2: (dBm) []
                    IIP3: (dBm) [  Inf      9        9        9]
                    OIP3: (dBm) [  Inf     13      8.4      8.4]
                     SNR: (dB)  [73.98  73.98    72.75    72.75]
                    EIRP: (dBm) [-20.6]
            Directivity: (dBi) [   10]
```

Use RF Receiver Antenna Element in RF Budget Chain

Design an RF receiver antenna element given a transmitting antenna with an EIRP value of –35 dBm and a pathloss of 2 dB.

```
antR = rfantenna(Type ='Receiver',TxEIRP =-35,PathLoss=2);
```

Create an amplifier with a gain of 4 dB.

```
a = amplifier(Gain=4);
```

Create a modulator with an OIP3 value of 13 dBm.

```
m = modulator(OIP3=13);
```

Create an n-port element using `passive.s2p`.

```
n = nport('passive.s2p');
```

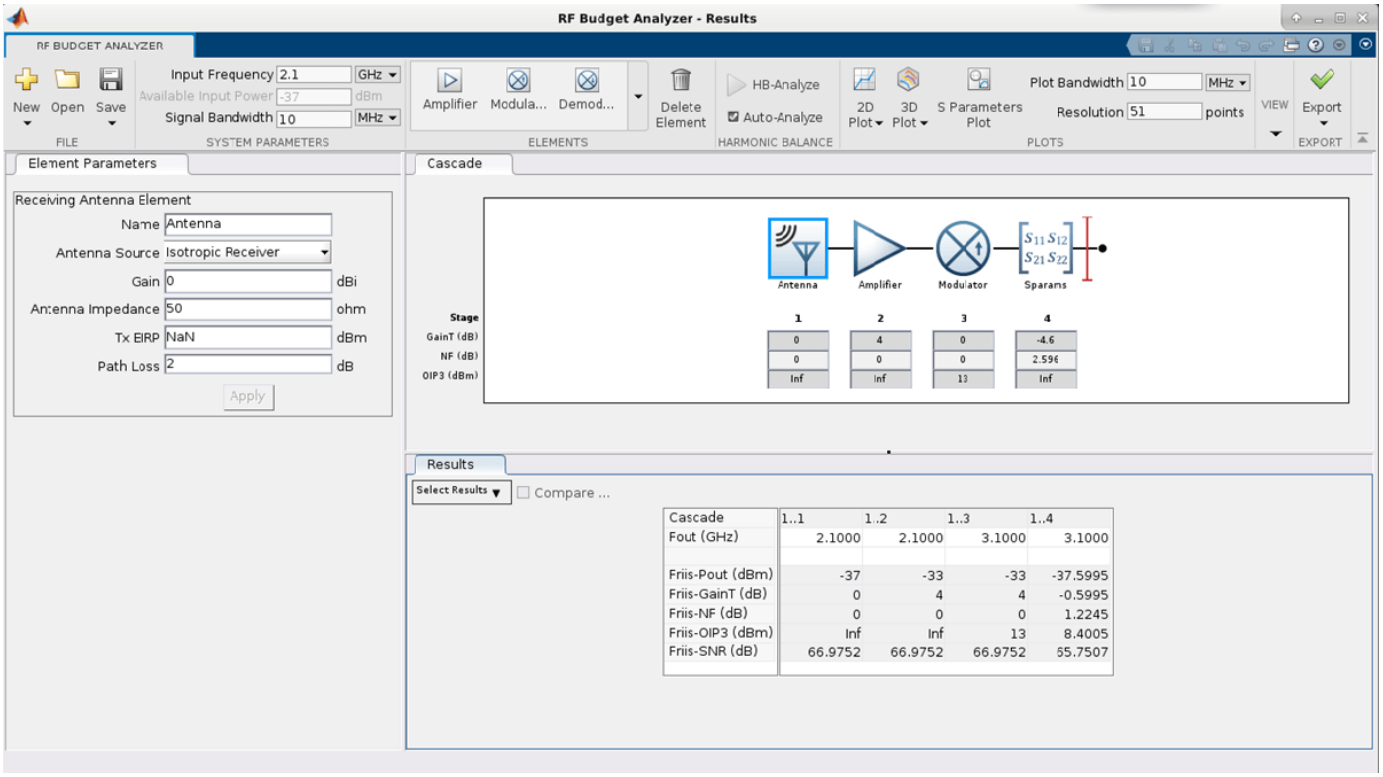
Calculate the RF budget of a series of RF elements by tying this command at the command line with an input frequency of 2.1 GHz, an available input power of –30 dBm, and a bandwidth of 10 MHz.

```
b = rfbudget([antR a m n],2.1e9,-30,10e6);
```

```
Warning: Available Input Power will be ignored because of the receiver.
Warning: Available Input Power will be ignored because of the receiver.
```

Use the `show` command at the command line to visualize the RF budget chain in the **RF Budget Analyzer** app. To do further analysis on this chain using this app see RF Budget Analyzer.

```
show(b)
```



Create Transmit-Receive Antenna Element

Create an RF antenna element in a transmit-receive configuration. Set the gain and the impedance of the transmitter to 5 dB and 45 ohms, respectively, and the gain and impedance of the receiver antenna to 6 dB and 55 ohms, respectively.

```
ant = rfantenna('Type','TransmitReceive','Gain',[5 6],'Z',[45 55])
```

```
ant =
  rfantenna: Antenna element
```

```
        Name: 'Antenna'
        Type: 'TransmitReceive'
        Gain: [5 6]
           Z: [45 55]
    PathLoss: 0
    NumPorts: 1
   Terminals: {'p1+'  'p1-'}
```

Calculate the S-parameters of the RF antenna element in the transmit-receive configuration at 1 GHz.

```
sparam = sparameters(ant,1e9)
```

```
sparam =
  sparameters: S-parameters object

       NumPorts: 1
    Frequencies: 1.0000e+09
     Parameters: -0.0025
      Impedance: 50

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

Version History

Introduced in R2021a

Model transmit-receive antenna element

Use the `rfantenna` object to model an antenna in the transmit-receive configuration by setting `Type` to `'TransmitReceive'`.

See Also

`attenuator`

Topics

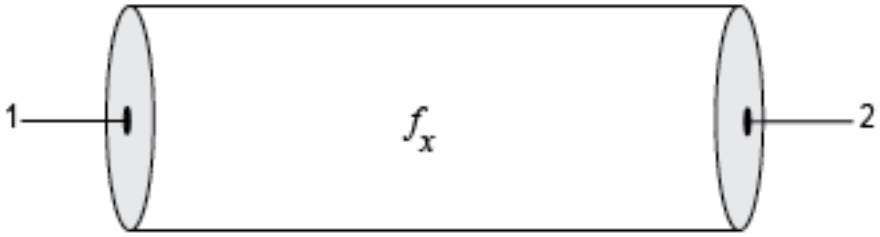
“Design RF Chain Using RF Antenna Object”

txlineEquationBased

Create equation-based transmission line

Description

Use the `txlineEquationBased` object to create an equation-based transmission line. You can set the loss, the phase velocity, and the interpolation type in this object. You can also use the `txlineEquationBased` object to model an equation-based transmission line in an RF systems using an `rfbudget` object or the **RF Budget Analyzer** app.



Creation

Syntax

```
eqtxline = txlineEquationBased
eqtxline = txlineEquationBased(Name,Value)
```

Description

`eqtxline = txlineEquationBased` creates a default equation-based transmission line object.

`eqtxline = txlineEquationBased(Name,Value)` sets "Properties" on page 1-304 of the equation-based transmission line using one or more name-value arguments. For example, `eqtxline = txlineEquationBased('Z0',75)` creates an equation-based transmission line with the impedance of 75 ohms.

Properties

Name — Name of equation-based transmission line

`'EquationBased'` (default) | string scalar | character vector

Name of the equation-based transmission line, specified as a string scalar or character vector.

Example: `'Name','EQbased'`

Example: `eqtxline.Name = 'EQbased'`

`Frequency` — Operating frequency of equation-based transmission line
`1.0e9` (default) | positive scalar

Operating frequency of the equation-based transmission line, specified as a positive scalar in Hz.

Example: `'Frequecny',2e9`

Example: `eqtxline.Frequency = 2e9`

`Z0` — Characteristic impedance
`50` (default) | positive scalar

Characteristic impedance of the equation-based transmission line, specified as a positive scalar in ohms.

Example: `'Z0',75`

Example: `eqtxline.Z0 = 75`

`LossDB` — Loss of equation-based transmission line
`0` (default) | positive scalar

Loss of the equation-based transmission line, specified as a scalar in dB.

Example: `'LossDB',25`

Example: `eqtxline.LossDB = 25`

`PhaseVelocity` — Phase velocity of equation-based transmission line
`299792458` (default) | positive scalar

Phase velocity of the equation-based transmission line, specified as a positive scalar in m/sec.

Example: `'PhaseVelocity',2e9`

Example: `eqtxline.PhaseVelocity = 2e9`

`IntType` — Type of interpolation
`'Linear'` (default) | `'Spline'` | `'Cubic'`

Type of interpolation set in the equation-based transmission line, specified as `'Linear'`, `'Spline'`, or `'Cubic'`.

Example: `'IntType','Spline'`

Example: `eqtxline.IntType = 'Spline'`

`LineLength` — Physical length of equation-based transmission line
`0.0100` (default) | positive scalar

Physical length of the equation-based transmission line, specified as a positive scalar in meters.

Example: `'LineLength',0.020`

Example: `eqtxline.LineLength = 0.020`

`Termination` — Stub transmission line termination
`'NotApplicable'` (default) | `'Open'` | `'Short'`

Stub transmission line termination in the equation-based transmission line, specified as `'NotApplicable'`, `'Open'`, or `'Short'`.

Example: `'Termination','Short'`

Example: `eqtxline.Termination = 'Short'`

`StubMode` — Type of stub
`'NotAStub'` (default) | `'Series'` | `'Shunt'`

Type of stub in the equation-based transmission line, specified as `'NotAStub'`, `'Series'`, or `'Shunt'`.

Example: `'StubMode','Series'`

Example: `eqtxline.StubMode = 'Series'`

`NumPorts` — Number of input and output ports
2 (default) | positive scalar

This property is read-only.

Number of input and output ports in equation-based transmission line, returned as positive scalar.

`Terminals` — Terminals of equation-based transmission line
`{'p1+' p2+' 'p1-' 'p2-'}` (default) | cell array of strings

This property is read-only.

Terminals of the equation-based transmission line, returned as a cell array of strings.

Object Functions

sparameters	Calculate S-parameters for RF data, network, circuit, and matching network objects
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object
noisefigure	Calculate noise figure of transmission lines, series RLC, and shunt RLC circuits

Examples

Calculate S-Parameters and Group Delay of Equation-Based Transmission Line

Create an equation-based transmission line `50` cm in length.

```
eqtxline = txlineEquationBased('LineLength',0.05);
```

Calculate the S-parameters of the equation-based transmission line over the frequency range `1`–`4` GHz.

```
freq = linspace(1e9,4e9,51);
sparam = sparameters(eqtxline, freq)
```

```
sparam =
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [51x1 double]
```

```
     Parameters: [2x2x51 double]
      Impedance: 50

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

Calculate the group delay of the equation-based transmission line at `2.4` GHz.

```
 gd = groupdelay(eqtxline,2.4e9)
```

```
gd = 1.6678e-10
```

Version History

Introduced in R2021a

See Also

`txlineDelayLossless` | `txlineDelayLossy` | `txlineCPW` | `txlineMicrostrip` | `txlineParallelPlate` | `txlineRLCGLine` | `txlineTwoWire` | `txlineCoaxial`

Topics

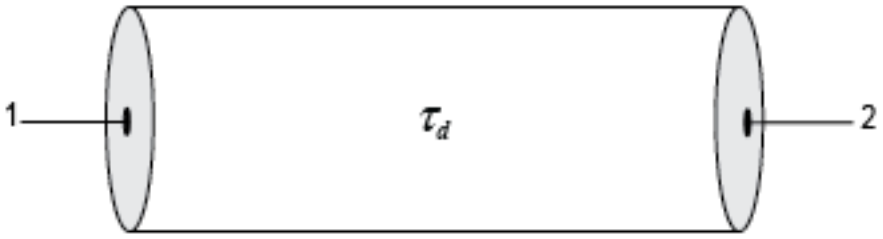
“Design Two-Stage Low Noise Amplifier Using Microstrip Transmission Line Matching Network”

txlineDelayLossless

Create delay lossless transmission line

Description

Use the `txlineDelayLossless` object to create a delay lossless transmission line. You can also use the `txlineDelayLossless` object to model a delay lossless transmission line in an RF system using the `rfbudget` object or the **RF Budget Analyzer** app.



Creation

Syntax

```
dltxline = txlineDelayLossless
dltxline = txlineDelayLossless(Name,Value)
```

Description

`dltxline = txlineDelayLossless` creates a default delay lossless transmission line object.

`dltxline = txlineDelayLossless(Name,Value)` sets "Properties" on page 1-308 of the delay lossless transmission line using one or more name-value arguments. For example, `dltxline = txlineDelayLossless('Z0',75)` creates a delay lossless transmission line with an impedance of 75 ohms.

Properties

`Name` — Name of delay lossless transmission line

`'DelayLossless'` (default) | string scalar | character vector

Name of the delay lossless transmission line, specified as a string scalar or character vector.

Example: `'Name','DLbased'`

Example: `dltxline.Name = 'DLbased'`

`Z0` — Characteristic impedance
`50` (default) | positive scalar

Characteristic impedance of the delay lossless transmission line, specified as a positive scalar in ohms.

Example: `'Z0',75`

Example: `dltxline.Z0 = 75`

`TimeDelay` — Time delay in delay lossless transmission line
`1e-12` (default) | positive scalar

Time delay (τ_d) in the delay lossless transmission line, specified as a positive scalar in seconds.

Example: `'TimeDelay',1e-9`

Example: `dltxline.TimeDelay = 1e-9`

`NumPorts` — Number of input and output ports
`2` (default) | positive scalar

This property is read-only.

Number of input and output ports of the delay lossless transmission line, returned as positive scalar.

`Terminals` — Terminals of delay lossless transmission line
`{'p1+' p2+' 'p1-' 'p2-'}` (default) | cell array

This property is read-only.

Terminals of the delay lossless transmission line, returned as a cell array.

Object Functions

sparameters	Calculate S-parameters for RF data, network, circuit, and matching network objects
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object
noisefigure	Calculate noise figure of transmission lines, series RLC, and shunt RLC circuits

Examples

Calculate Group Delay and Noise Figure of Delay Lossless Transmission Line

Create a delay lossless transmission line with a transmission delay of `5e-12` sec.

```
dltxline = txlineDelayLossless('TimeDelay',5e-12);
```

Calculate the group delay at `10` MHz.

```
gd = groupdelay(dltxline,10e6)
```

```
gd = 5.0000e-12
```

Calculate the noise figure at `10` MHz.

```
nf = noisefigure(dltxline,10e6)
```

```
nf = 0
```

Version History

Introduced in R2021a

See Also

`txlineDelayLossy` | `txlineEquationBased` | `txlineCPW` | `txlineMicrostrip` | `txlineParallelPlate` | `txlineRLCGLine` | `txlineTwoWire` | `txlineCoaxial`

Topics

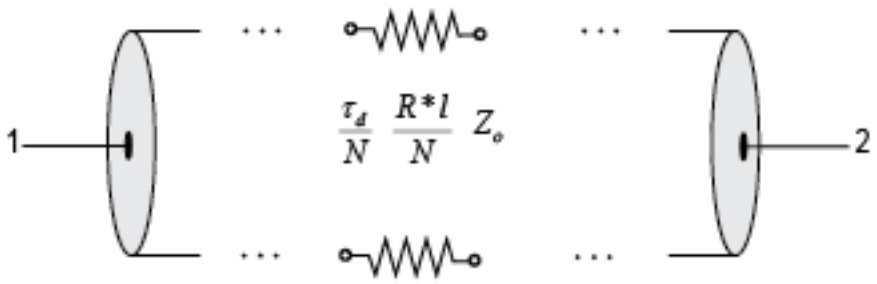
“Design Two-Stage Low Noise Amplifier Using Microstrip Transmission Line Matching Network”

txlineDelayLossy

Create delay lossy transmission line

Description

Use the `txlineDelayLossy` object to create a delay lossy transmission line. You can also use the `txlineDelayLossy` object to model a delay lossy transmission line in an RF system using the `rfbudget` object or the **RF Budget Analyzer** app.



Creation

Syntax

```
dlytxline = txlineDelayLossy
dlytxline = txlineDelayLossy(Name,Value)
```

Description

`dlytxline = txlineDelayLossy` creates a default delay lossy transmission line object.

`dlytxline = txlineDelayLossy(Name,Value)` sets “Properties” on page 1-311 of the delay lossy transmission line using one or more name-value arguments. For example, `dlytxline = txlineDelayLossy('Z0',75)` creates a delay lossy transmission line with an impedance of 75 ohms.

Properties

`Name` — Name of delay lossy transmission line

`'DelayLossy'` (default) | string scalar | character vector

Name of the delay lossy transmission line, specified as a string scalar or character vector.

Example: `'Name','DLbased'`

Example: `dlytxline.Name = 'DLbased'`

`Z0` — Characteristic impedance
`50` (default) | positive scalar

Characteristic impedance of the delay lossy transmission line, specified as a positive scalar in ohms.

Example: `'Z0',75`

Example: `dlytxline.Z0 = 75`

`LineLength` — Physical length of delay lossy transmission line
`0.0100` (default) | positive scalar

Physical length of the delay lossy transmission line, specified as a positive scalar in meters.

Example: `'LineLength',0.0200`

Example: `dlytxline.LineLength = 0.0200`

`TimeDelay` — Time delay in delay lossy transmission line
`4.7e-9` (default) | positive scalar

Time delay in the delay lossy transmission line, specified as a positive scalar in seconds.

Example: `'TimeDelay',3.7e-9`

Example: `dlytxline.TimeDelay = 3.7e-9`

`Resistance` — Resistance value per unit of length
`0.3000` (default) | positive scalar

Resistance value per unit of length of the delay lossy transmission line, specified as a positive scalar in ohm/meter.

Example: `'Resistance',0.3400`

Example: `dlytxline.Resistance = 0.3400`

`NumPorts` — Number of input and output ports
`2` (default) | positive scalar

This property is read-only.

Number of input and output ports of the delay lossy transmission line, returned as positive scalar.

`Terminals` — Terminals of delay lossy transmission line
`{'p1+' p2+' 'p1-' 'p2-'}` (default) | cell array

This property is read-only.

Terminals of the delay lossy transmission line, returned as a cell array.

Object Functions

sparameters	Calculate S-parameters for RF data, network, circuit, and matching network objects
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object
noisefigure	Calculate noise figure of transmission lines, series RLC, and shunt RLC circuits

Examples

Calculate Noise Figure of Lossy Transmission Line

Design a coaxial cable of `0.5` meters in length as a lossy transmission line with the characteristic impedance of `48` ohms.

```
dlossy = txlineDelayLossy("Z0",48,"LineLength",500e-3);
```

Calculate the noise figure at `1` GHz.

```
nf = noisefigure(dlossy,1e9)
```

```
nf = 0.0013
```

Algorithms

The delay lossy transmission line object calculates the S-parameters for the specified frequencies. This calculation is based on the delay line's line length, resistance, and time delay. The S-parameters are calculated using the equation given.

$S_{11} = 0$

$S_{12} = e^{-p}$

$S_{21} = e^{-p}$

$S_{22} = 0$

Here, $p = \alpha_a + (i \times \beta)$. α_a in p is the attenuation coefficient and β is the wave number.

The attenuation coefficient, α_a, is related to the loss, α, by

$$\alpha_a = \frac{(\text{linelength} \times \text{resistance})}{(2Z)}$$

and the wave number β is related to the time delay, `D`, by

$$\beta = 2 \times \pi \times f \times D$$

where `f` is the frequency range specified in the S-parameters input argument.

Version History

Introduced in R2021a

See Also

`txlineDelayLossless` | `txlineEquationBased` | `txlineCPW` | `txlineMicrostrip` | `txlineParallelPlate` | `txlineRLCGLine` | `txlineTwoWire` | `txlineCoaxial`

Topics

“Design Two-Stage Low Noise Amplifier Using Microstrip Transmission Line Matching Network”

txlineElectricalLength

Create electrical-length-based transmission line

Description

Use the `txlineElectricalLength` object to create an electrical-length-based transmission line. The `txlineElectricalLength` object is used in the Richards-Kuroda workflow.

Creation

Syntax

```
eltxline = txlineElectricalLength
eltxline = txlineElectricalLength(Name=Value)
```

Description

`eltxline = txlineElectricalLength` creates a default electrical-length-based transmission line object.

`eltxline = txlineElectricalLength(Name=Value)` sets "Properties" on page 1-314 of the electrical-length-based transmission line using one or more name-value arguments. For example, `txlineElectricalLength(Z0=75)` creates an electrical-length-based transmission line with an impedance of `75` ohms.

Properties

`Name` — Name of transmission line
`'ElectricalLength'` (default) | string scalar | character vector

Name of the electrical-length-based transmission line, specified as a string scalar or character vector.

Example: `Name='ELbased'`

`Z0` — Characteristic impedance
`50` (default) | positive scalar

Characteristic impedance of the electrical-length-based transmission line, specified as a positive scalar in ohms.

Example: `Z0=75`

`LineLength` — Electrical length
`pi/4` (default) | positive scalar

Electrical length of the electrical-length-based transmission line, specified as a positive scalar in radians.

Example: `LinneLength=pi/4`

`ReferenceFrequency` — Reference frequency
`1e9` (default) | positive scalar

Reference frequency at which electrical-length-based transmission line exhibits its electrical length, specified as a positive scalar in radians.

Example: `ReferenceFrequency=2e9`

`Termination` — Stub termination
`'NotApplicable'` (default) | `'Open'` | `'Short'`

Stub termination, specified as either `'NotApplicable'`, `'Open'`, or `'Short'`.

Example: `Termination='Short'`

Data Types: `char`

`StubMode` — Type of stub
`'NotAStub'` (default) | `'Series'` | `'Shunt'`

Type of stub, specified as either `'NotAStub'`, `'Series'`, or `'Shunt'`.

Example: `StubMode='Series'`

Data Types: `char`

`NumPorts` — Number of input and output ports
2 (default) | positive scalar

This property is read-only.

Number of input and output ports of the electrical-length-based transmission line, returned as positive scalar.

`Terminals` — Terminals of transmission line
`{'p1+' p2+' 'p1-' 'p2-'}` (default) | cell array

This property is read-only.

Terminals of the electrical-length-based transmission line, returned as a cell array.

Object Functions

sparameters	Calculate S-parameters for RF data, network, circuit, and matching network objects
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object
noisefigure	Calculate noise figure of transmission lines, series RLC, and shunt RLC circuits
realize	Realize circuit containing electrical-length-based transmission lines using microstrip transmission lines

Examples

Calculate Noise Figure and Group Delay of Electrical Length Transmission Line

Create an ideal transmission line with an electrical length of pi/8 radians at the reference frequency of 100 MHz.

```
el = txlineElectricalLength(LineLength=pi/8,ReferenceFrequency=100e6);
```

Calculate the noise figure at 10 MHz.

```
nf = noisefigure(el,10e6)
```

```
nf = 0
```

Calculate the group delay at 10 MHz.

```
gd = groupdelay(el,10e6)
```

```
gd = 6.2500e-10
```

Version History

Introduced in R2021b

See Also

`richards` | `kuroda` | `insertUnitElement`

Topics

“Richards-Kuroda Workflow for RF Filter Circuit”

phaseshift

Create phase-shift circuit

Description

Use a `phaseshift` object to create a phase-shift circuit. The `phaseshift` object is a two-port network commonly used in image rejection receivers and digital beam steering applications.

You can also use the `phaseshift` object to model a phase shift circuit in cascaded stages using an `rfbudget` object or the **RF Budget Analyzer** app.

Creation

Syntax

```
psh = phaseshift
psh = phaseshift(Name=Value)
```

Description

`psh = phaseshift` creates a `phaseshift` object with default values.

`psh = phaseshift(Name=Value)` sets "Properties" on page 1-317 of the `phaseshift` object using one or more name-value arguments. For example, `psh = phaseshift(phaseshift=80)` creates an 80 degree phase shift circuit. Properties you do not specify retain their default values.

Properties

`Name` — Name of phase shift circuit

`phaseshift` (default) | string scalar | character vector

Name of the phase shift circuit, specified as a string scalar or a character vector.

Example: `phaseshift(Name='phaseshiftObj')`

`Phaseshift` — Phase angle difference

`90` (default) | positive scalar

Phase angle difference between the input and the output signals expressed by the insertion phase angle, specified as a positive scalar in degrees.

Example: `phaseshift(phaseshift=80)`

`NumPorts` — Number of input and output ports

`2` (default) | positive scalar

This property is read-only.

Number of input and output ports of the phase shift circuit, specified as a positive scalar.

`Terminals` — Terminals of phase shift circuit

`{'p1+' 'p2+' 'p1-' 'p2-'}` (default) | cell array

This property is read-only.

Terminals of the phase shift circuit, specified as a cell array.

Object Functions

sparameters Calculate S-parameters for RF data, network, circuit, and matching network objects

Examples

Create Phase Shift Circuit

Create a 25 degree phase shift circuit for a radio receiver.

```
psh = phaseshift(PhaseShift = 25);
```

Calculate the S-parameters at 3 GHz.

```
spar = sparameters(psh,3e9)
```

```
spar =
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: 3.0000e+09
     Parameters: [2x2 double]
      Impedance: 50

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

Visualize Phase Shift Element in RF Budget Analyzer App

Create an amplifier with a gain of 4 dB.

```
a = amplifier(Gain = 4);
```

Create a modulator with an OIP3 value of 13 dBm.

```
m = modulator(OIP3 = 13);
```

Create an n-port element using `passive.s2p`.

```
n = nport('passive.s2p');
```

Create a 40 degree phase shift element.

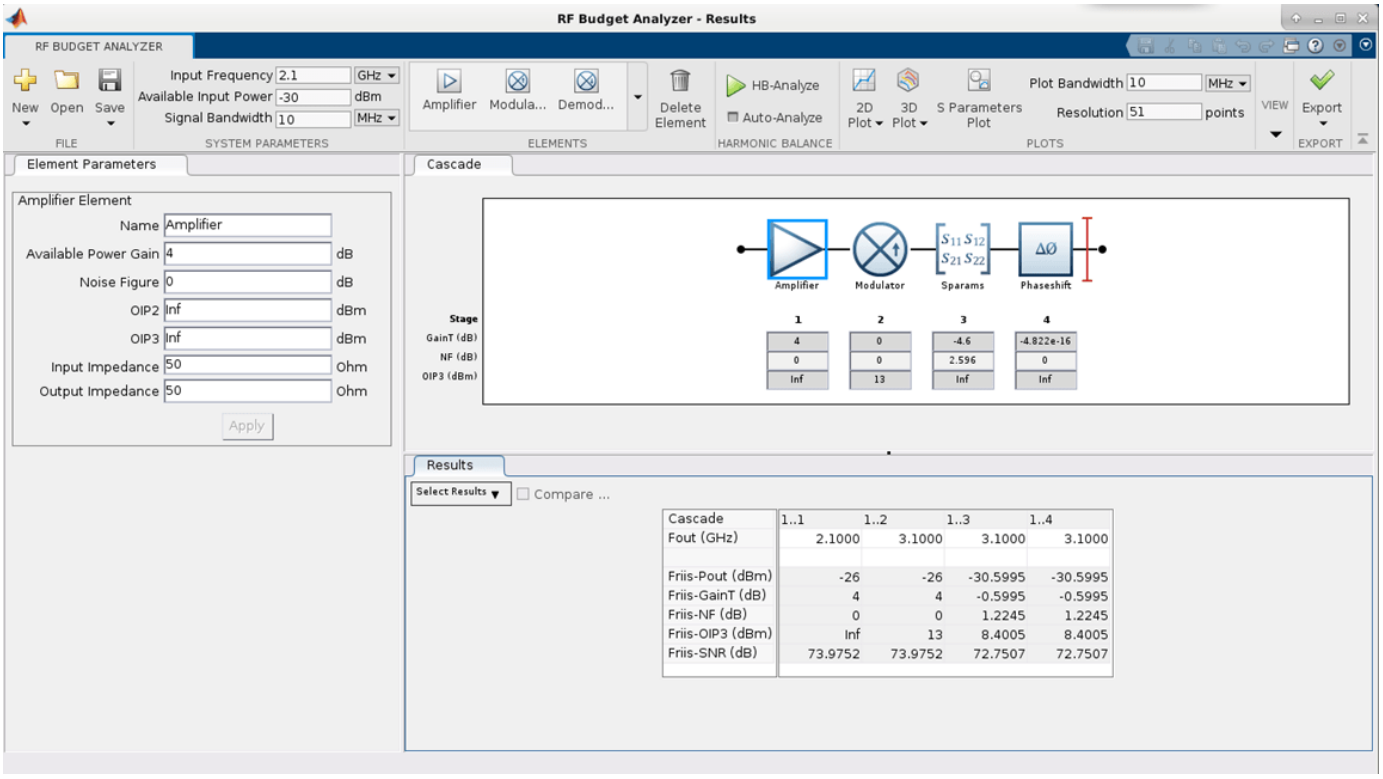
```
psh = phaseshift(PhaseShift = 40);
```

Calculate the RF budget of a series of RF elements at an input frequency of 2.1 GHz, an available input power of –30 dBm, and a bandwidth of 10 MHz.

```
b = rfbudget([a m n psh],2.1e9,-30,10e6);
```

Use the `show` command at the command line to visualize the RF budget chain in the **RF Budget Analyzer** app. You can also do further analysis on this chain using the app. For more information, see RF Budget Analyzer.

```
show(b)
```



Version History

Introduced in R2021b

See Also

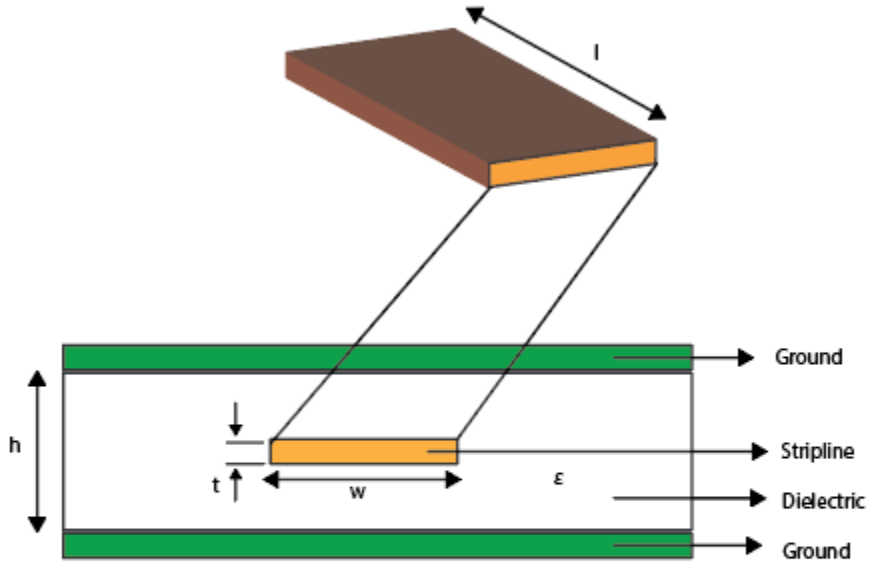
`rfantenna` | `seriesRLC` | `shuntRLC`

txlineStripline

Create stripline transmission line element

Description

Use the `txlineStripline` object to create a stripline transmission line element. The cross-section of a stripline transmission line is shown in the following figure. The physical characteristics of a stripline transmission line include strip width *w*, strip length *l*, strip thickness *t*, substrate height *h*, and relative permittivity constant *ε*. You can also use the `txlineStripline` object to model a delay lossless transmission line in an RF system using the `rfbudget` object or the **RF Budget Analyzer** app.



l — Stripline length
t — Stripline thickness
w — Stripline width
h — Dielectric thickness
ε — Relative permittivity constant

Creation

Syntax

```
txline = txlineStripline
txline = txlineStripline(Name=Value)
```

Description

`txline = txlineStripline` creates a stripline transmission line object.

`txline = txlineStripline(Name=Value)` sets properties using one or more name-value arguments. For example, `txline = txlineStripline(Width=0.0046)` creates a standard stripline transmission line with a width of 0.0046 meters. Properties you do not specify retain their default values.

Properties

`Name` — Name of stripline transmission line
`'Stripline'` (default) | string scalar | character vector

Name of the stripline transmission line, specified as a string scalar or a character vector.

Example: `Name='stripline1'`

`Width` — Physical width of stripline transmission line
`0.0027` (default) | positive scalar

Physical width of the stripline transmission line, specified as a positive scalar in meters.

Example: `Width=0.0008`

Data Types: `double`

`DielectricThickness` — Physical thickness of dielectric
`0.0032` (default) | positive scalar

Physical thickness of the dielectric, specified as a positive scalar in meters.

Example: `DielectricThickness=0.00835`

Data Types: `double`

`Thickness` — Physical thickness of stripline transmission line
`0.000001` (default) | positive scalar

Physical thickness of the stripline transmission line, specified as a positive scalar in meters.

Example: `Thickness=0.00002`

Data Types: `double`

`EpsilonR` — Relative permittivity of dielectric
`2.2` (default) | positive scalar

Relative permittivity of the dielectric, specified as a positive scalar.

Example: `EpsilonR=8.8`

Data Types: `double`

`LossTangent` — Loss angle tangent of dielectric
`0.001` (default) | nonnegative scalar

Loss angle tangent of the dielectric, specified as a nonegative scalar.

Example: `LossTangent=0.01`

Data Types: `double`

`SigmaCond` — Conductivity of conductor
`Inf` (default) | nonnegative scalar

Conductivity of the conductor, specified as a nonnegative scalar in Siemens per meter (S/m).

Example: `SigmaCond=2`

Data Types: `double`

`LineLength` — Physical length of stripline transmission line
`0.0100` (default) | positive scalar

Physical length of the stripline transmission line, specified as a positive scalar in meters.

Example: `LineLength=0.0200`

Data Types: `double`

`Termination` — Stub transmission line termination
`'NotApplicable'` (default) | `'Open'` | `'Short'`

Stub transmission line termination, specified as either `'NotApplicable'`, `'Open'`, or `'Short'`.

Example: `Termination='Short'`

Data Types: `char`

`StubMode` — Type of stub
`'NotAStub'` (default) | `'Series'` | `'Shunt'`

Type of stub, specified as either `'NotAStub'`, `'Series'`, or `'Shunt'`.

Example: `StubMode='Series'`

Data Types: `char`

`NumPorts` — Number of input and output ports
`2` (default) | positive scalar

This property is read-only.

Number of input and output ports, returned as a positive scalar.

Data Types: `double`

`Terminals` — Terminals of stripline transmission line
`{'p1+' p2+' 'p1-' 'p2-'}` (default) | cell array of strings

This property is read-only.

Terminals of the stripline transmission line, returned as a cell array of strings.

Data Types: `char` | `string`

Object Functions

sparameters	Calculate S-parameters for RF data, network, circuit, and matching network objects
groupdelay	Group delay of S-parameter object or RF filter object or RF Toolbox circuit object
noisefigure	Calculate noise figure of transmission lines, series RLC, and shunt RLC circuits
getZ0	Calculate characteristic impedance with and without dispersion for transmission line
circuit	Circuit object
clone	Create copy of existing circuit element or circuit object

Examples

Plot S-Parameters of Stripline Transmission Line

Create a 10 GHz stripline transmission line.

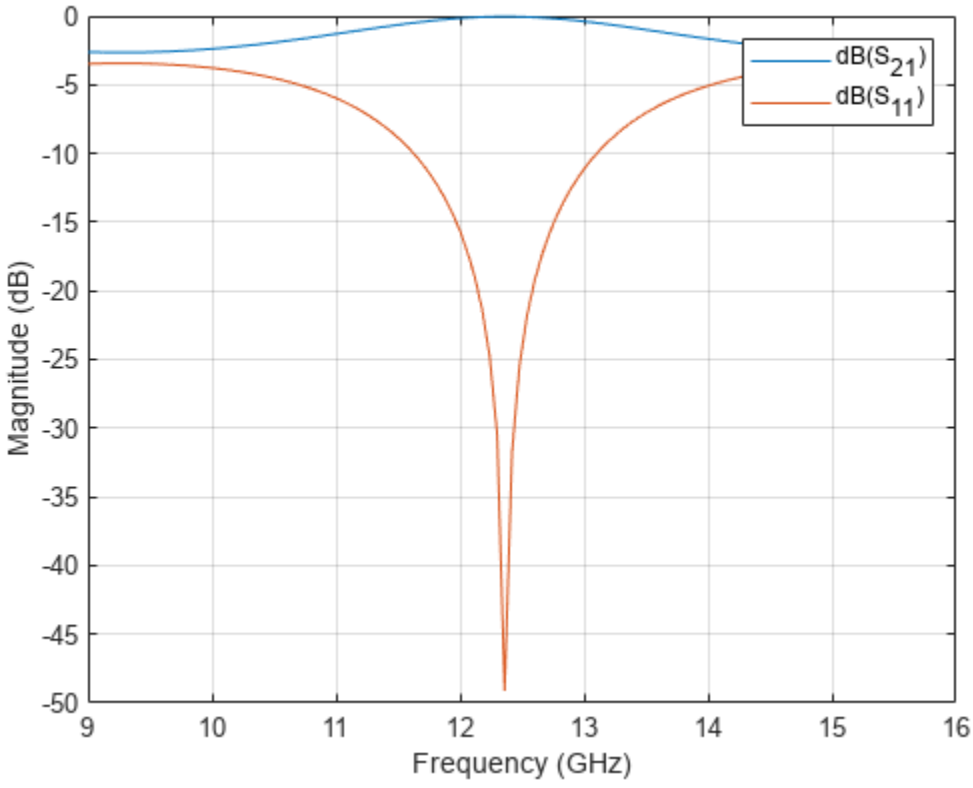
```
stl = txlineStripline(Width=0.08e-3,DielectricThickness=1.6e-3,LineLength=12.2777e-3,...
        Thickness=0.01e-3,EpsilonR=3.9,SigmaConductivity=5.88e7);
```

Calculate the S-parameters of the stripline transmission line.

```
freq = linspace(9e9,15e9,101);
spar1 = sparameters(stl,freq,50);
```

Plot the S21 and S11 of the stripline transmission line.

```
rfplot(spar1,[2 1],1)
```



Plot Group Delay of Stripline Transmission Line

Create a stripline transmission line object.

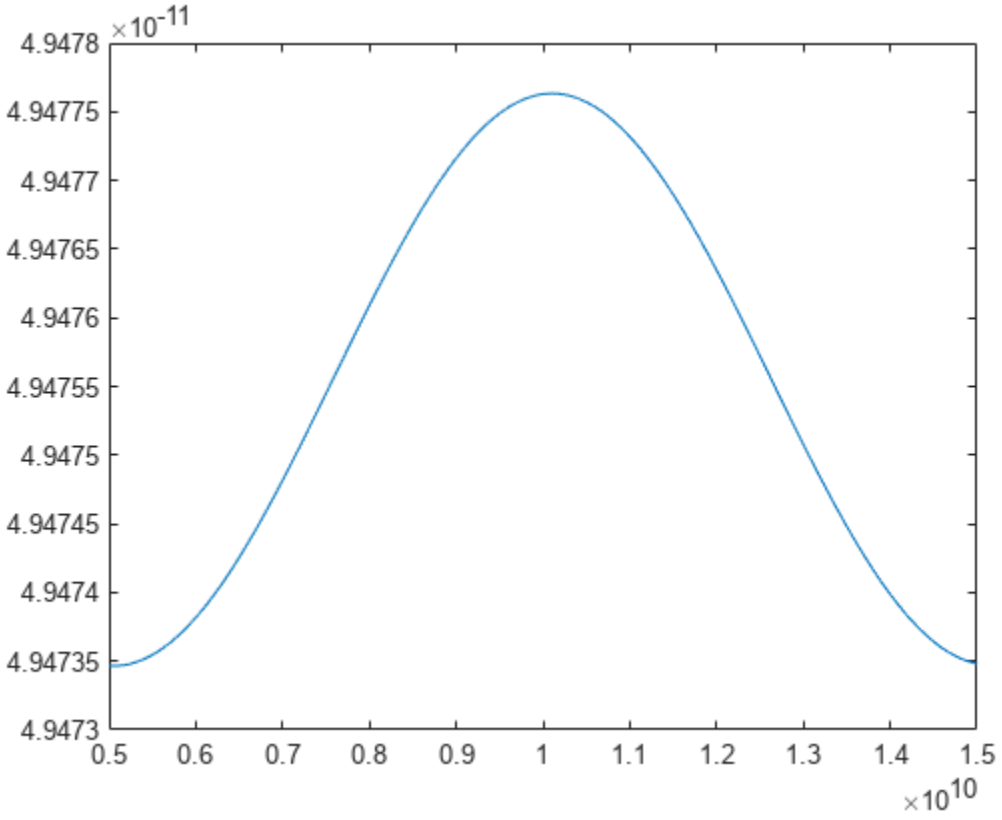
```
txline = txlineStripline;
```

Calculate the group delay of the stripline transmission line.

```
freq = linspace(5e9,15e9,101);
gd = groupdelay(txline,freq,Impedance=50);
```

Plot the group delay of the stripline transmission line.

```
plot(freq,gd)
```



Calculate the characteristic impedance of the stripline transmission line.

```
char = getZ0(txline,10e9)
```

```
char = 49.5426
```

Version History

Introduced in R2022a

Model stripline in RF Budget Analyzer app

To model `txlineStripline` object in **RF Budget Analyzer**, on the **Elements** pane, select **Transmission line** and set **Type** to `Stripline`.

See Also

`txlineCoaxial` | `txlineCPW` | `txlineMicrostrip` | `txlineParallelPlate` | `txlineRLCGLine` | `txlineTwoWire` | `txlineEquationBased` | `txlineDelayLossless` | `txlineDelayLossy` | `txlineElectricalLength`

mixerIMT

Create IMT mixer element

Description

Use the `mixerIMT` object to perform frequency translation defined in an intermodulation table (IMT) for a single-tone carrier mixed with a local oscillator (LO) signal. If you design an RF system using an `rfbudget` object or the **RF Budget Analyzer** app you can model an IMT mixer with the `mixerIMT` object and add it to the RF system. You can then export this element to RF Blockset or to an `rfsystem` System object for circuit envelope analysis.

Creation

Syntax

```
imt = mixerIMT
imt = mixerIMT(Name=Value)
```

Description

`imt = mixerIMT` creates an IMT mixer object with default property values.

`imt = mixerIMT(Name=Value)` sets Properties using one or more name-value arguments. For example, `imt = mixerIMT(LO=5e6)` sets the local oscillator frequency of an IMT mixer to 5 GHz. Properties not specified retain their default values.

Properties

`Name` — Name of IMT mixer
`'MixerIMT'` (default) | string scalar | character vector

Name of the IMT mixer, specified as a string scalar or character vector.

Example: `Name='imtmixer'`

Data Types: `char` | `string`

`ReferenceInputPower` — Reference input power
`-15` (default) | scalar

Reference input power, specified as a scalar in dBm.

Example: `'ReferenceInputPower=-10`

Data Types: `double`

`NominalOutputPower` — Nominal output power
`-5` (default) | scalar

Nominal output power, specified as a scalar in dBm.

Example: `NominalOutputPower=8`

Data Types: `double`

`NF` — Noise figure
`0` (default) | scalar

Noise figure, specified as a scalar in dBm.

Example: `NF=8`

Data Types: `double`

`LO` — Local oscillator frequency
`1e9` (default) | real finite positive scalar

Local oscillator frequency, specified as a real finite positive scalar in Hz.

Example: `LO=2e9`

Data Types: `double`

`ConverterType` — Conversion direction
`'Up'` (default) | `'Down'`

Conversion direction, specified as either `'Up'` or `'Down'`.

Example: `ConverterType='Down'`

Data Types: `string` | `char`

`Zin` — Input impedance
`50` (default) | positive scalar

Input impedance, specified as a positive scalar in ohms.

Example: `Zin=40`

Data Types: `double`

`Zout` — Output impedance
`50` (default) | positive scalar

Output impedance, specified as a positive scalar in ohms.

Example: `Zout=40`

Data Types: `double`

`IMT` — IMT spurs to plot
`[99 99 99; 99 0 99; 99 99 99]` (default) | real square matrix

IMT spurs to plot, specified as a real square matrix.

Example: `IMT=[99 99 80; 99 0 99; 99 80 99]`

`UseDataFile` — Use S2D file
`false or 0` (default) | `true or 1`

Use S2D file to design IMT mixer, specified as a numeric or logical `1 (true)` or `0 (false)`.

Example: `UseDataFile=1`

Data Types: `logical`

`FileName` — File name of S2D file

`[]` (default) | string scalar | character vector

File name of the S2D file, specified as a string scalar or a character vector.

Example: `FileName='samplespur1.s2d'`

Dependency

To enable this property, set `UseDataFile` to `1`.

Data Types: `string` | `char`

`NumPorts` — Number of input and output ports

2 (default) | positive scalar

This property is read-only.

Number of input and output ports, returned as a positive scalar.

Data Types: `double`

`Terminals` — Terminals of IMT mixer

`{'p1+' p2+' 'p1-' 'p2-'}` (default) | cell array of strings

This property is read-only.

Terminals of the IMT mixer, returned as a cell array of strings.

Data Types: `string`

Object Functions

sparameters	Calculate S-parameters for RF data, network, circuit, and matching network objects
rfplot	Plot mixer spurs of IMT mixer

Examples

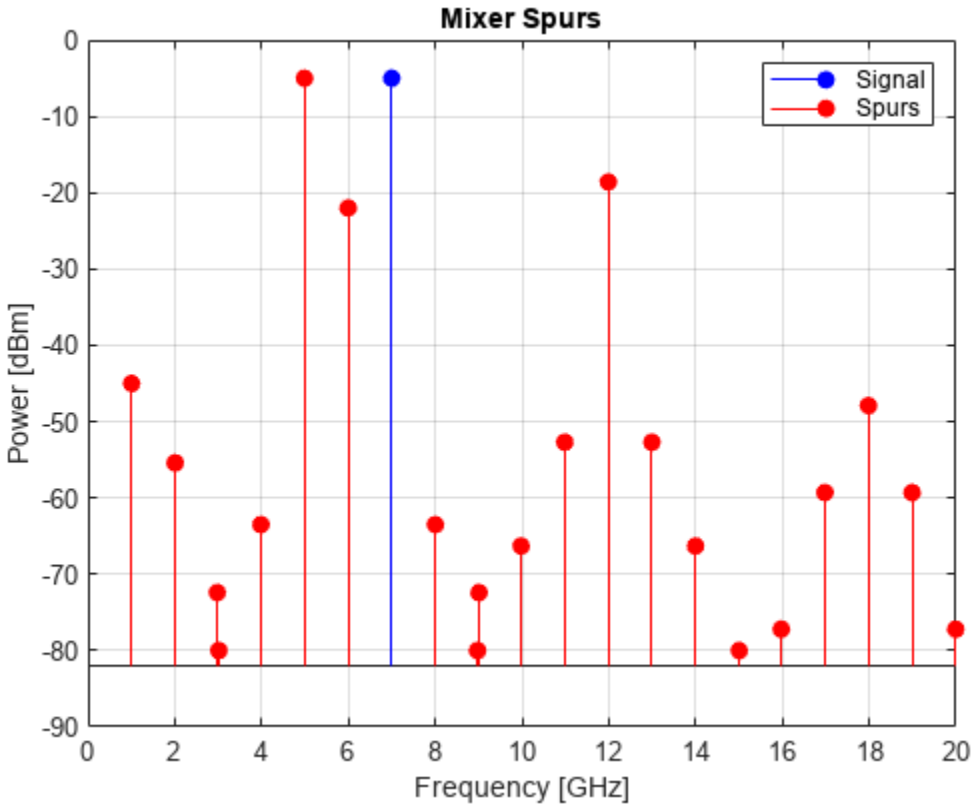
Plot Mixer Spur of IMT mixer

Create an IMT mixer with the LO frequency of 5995 MHz and with the IMT spurs.

```
imt = mixerIMT(LO=5995e6,IMT=...
     [99 17 13.6 42.9; 40 0 47.7 54.3; 50.4 58.5 61.3 72.2; 75 67.4 75 99]);
```

Plot the mixer spur at an input signal frequency of 1005 MHz.

```
rfplot(imt,1005e6)
```



Version History

Introduced in R2022a

See Also

`amplifier` | `modulator` | `rfelement`

Topics

“Visualize Mixer Spurs”

“Operations with RF Data Objects”

noiseParameters

Extract noise data from Touchstone file or build noise figure data

Description

Use the `noiseParameters` object to extract noise data from Touchstone file or build noise data from a noise figure data. You can also use this object to set the NoiseData property in the `amplifier` object.

Creation

Syntax

```
np = noiseParameters(filename)
np = noiseParameters(fmin,freq,z0)
```

Description

`np = noiseParameters(filename)` creates a noise parameters object `np` from the noise data in the two-port Touchstone file specified by `filename`.

`np = noiseParameters(fmin,freq,z0)` creates a noise parameters object `np` and sets the Fmin and Frequencies properties with the reference impedance `z0`.

Input Arguments

`filename` — Name of two-port Touchstone file
string scalar | character vector

Name of the two-port Touchstone file from which to extract noise data, specified as a string scalar or character vector.

Example: `np = noiseParameters('default.s2p')`

`z0` — Reference impedance
real scalar

Reference impedance, specified as a real scalar in ohms. `z0` must be equal to the reference impedance of the network parameters.

Example: `np = noiseParameters(fmin,freq,z0)`

Properties

`Frequencies` — Noise frequencies
column vector

Noise frequencies, specified as a column vector in Hz.

Example: `np.Frequencies=(2:2:22)*1e9`

`Fmin` — Minimum noise figure
column vector

Minimum noise figure, specified as a column vector in dB.

Example: `np.Fmin=2:10`

`GammaOpt` — Source reflection coefficient
complex column vector

Source reflection coefficient to realize the minimum noise figure, specified as a complex column vector. You cannot set this property when creating the object. You can modify the property once you have created the object using dot notation.

Example: `np.GammaOpt=[2+j*5; 7; 3+j*9]`

`Rn` — Effective noise resistance
column vector

Effective noise resistance normalized to the same resistance as the network parameters, specified as a column vector in ohms. You cannot set this property when creating the object. You can modify the property once you have created the object using dot notation.

Example: `np.Rn=[2:10]`

Object Functions

amplifier Create two-port amplifier element

Examples

Extract Noise Data from Touchstone File

Extract the noise data from a touchstone file.

```
np = noiseParameters('default.s2p')
```

```
np =
  noiseParameters with properties:

    Frequencies: [9x1 double]
           Fmin: [9x1 double]
       GammaOpt: [9x1 double]
             Rn: [9x1 double]
```

Add this noise data to an `amplifier` object.

```
a = amplifier(FileName='default.s2p',NoiseData=np)
```

```
a =
  amplifier: Amplifier element

               Name: 'Amplifier'
```

```
    UseNetworkData: 1
          FileName: 'default.s2p'
       NetworkData: [1x1 sparameters]
         NoiseData: [1x1 noiseParameters]
              OIP2: Inf
              OIP3: Inf
          NumPorts: 2
         Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Build Noise Parameters from Measured NF Data

Define a measured noise figure (NF), noise frequencies, and the reference impedance data.

```
NF = [4 3 2 2 2 2 2 2.5 2.5 3 3.5];
freqs = (2:2:22)*1e9;
z0 = 50;
```

Build the noise parameters from the measured NF data.

```
np = noiseParameters(NF,freqs,z0);
```

Add this noise data to an `amplifier` object.

```
a = amplifier(FileName='default.s2p',NoiseData=np)
```

```
a =
  amplifier: Amplifier element

              Name: 'Amplifier'
    UseNetworkData: 1
          FileName: ''
       NetworkData: [1x1 sparameters]
         NoiseData: [1x1 noiseParameters]
              OIP2: Inf
              OIP3: Inf
          NumPorts: 2
         Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Version History

Introduced in R2022a

See Also

`amplifier`

Functions

addEvaluationParameter

Adds performance goal for sort, pass, or fail matching network design

Syntax

```
mnobjupdated = addEvaluationParameter(mnobj,parameter,comparison,targetdb,
band,weight)
```

Description

`mnobjupdated = addEvaluationParameter(mnobj,parameter,comparison,targetdb, band,weight)` adds a performance goal to an existing matching network and returns an updated matching network object.

Examples

Matching Network From Dipole Antenna

Create a dipole antenna and create the S-parameters of the antenna. This example requires Antenna Toolbox.

```
d       = dipole('Length', 0.103, 'Width',0.0022);
freq    = linspace(0.5e9,2.5e9,1001);
sd      = sparameters(d, freq);
```

Alternatively, load S-Parameters from the MAT file

```
% load('sparams_dipole.mat')
```

Create a matching network from the S-parameters.

```
n = matchingnetwork('LoadImpedance',sd,'Components',3,...
    'LoadedQ',7,'CenterFrequency',2e9);
```

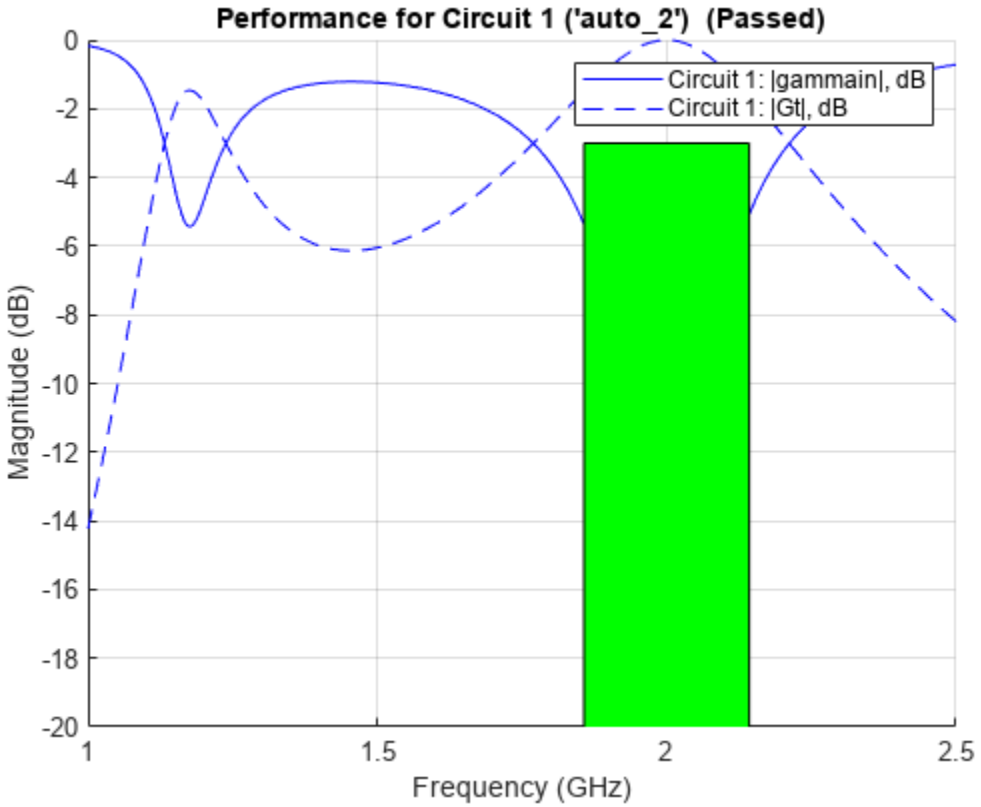
Get the evaluation parameters of the network.

```
t = getEvaluationParameters(n)
```

```
t=1×6 table
    Parameter    Comparison     Goal               Band                Weight       Source
    _________    __________    ______    ____________________________    ______    ____________

     {'Gt'}        {'>'}       {[-3]}    {[1.8571e+09 2.1429e+09]}    {[1]}     {'Automatic'}
```

Plot the reflection coefficient and transducer gain of the matching network circuit 1 , at a frequency range of 1 GHz to 2.5 GHz.

```
rfplot(n, (1e9:0.001e9:2.5e9),1);
```

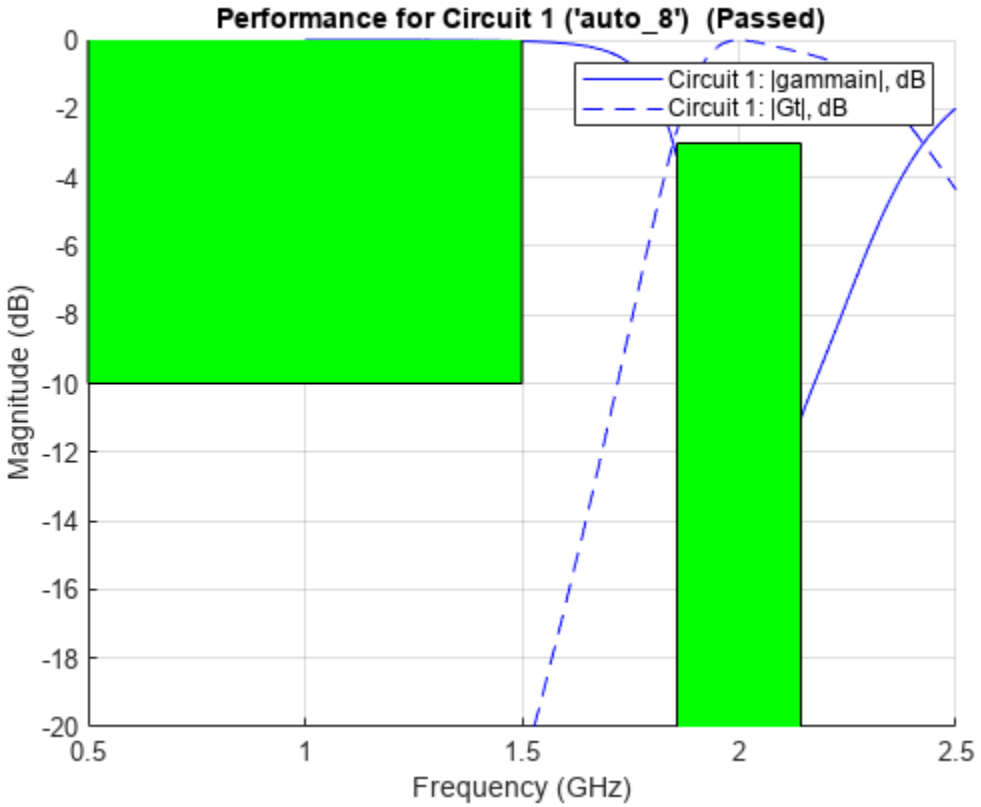


Add a new evaluation parameter to compare the transducer gain to have a cut-off of less than -10 dB. Use a frequency range of 0.5 GHz to 1.5 GHz. Plot the comparisons.

```
n = addEvaluationParameter(n, 'Gt', '<', -10, [0.5e9 1.5e9], 1);
t = getEvaluationParameters(n)
```

```
t=2×6 table
    Parameter    Comparison     Goal              Band                    Weight          Source
    _________    __________    _______    ____________________________    ______    _______________

     {'Gt'}         {'>'}       {[ -3]}    {[1.8571e+09 2.1429e+09]}      {[1]}     {'Automatic'
     {'Gt'}         {'<'}       {[-10]}    {[ 500000000 1.5000e+09]}      {[1]}     {'User-specifie
```

```
rfplot(n, (1e9:0.001e9:2.5e9),1);
```



Clear evaluation parameters.

```
n = clearEvaluationParameter(n,1);
t = getEvaluationParameters(n)
```

```
t=1×6 table
    Parameter    Comparison     Goal                Band                  Weight          Source
    _________    __________    _______    ____________________________    ______    ________________

     {'Gt'}        {'<'}       {[-10]}    {[500000000 1.5000e+09]}        {[1]}     {'User-specified
```

Input Arguments

`mnobj` — Matching network

`matchingnetwork` object

Matching network, specified as a `matchingnetwork` object.

Data Types: `char` | `string`

`parameter` — Evaluation parameter

`'gammain'` | `'Gt'`

Evaluation parameter to define targets for input reflection coefficients or transducer gain for matching networks when cascaded between source and load impedance, specified as `'gammain'` or `'Gt'`.

Data Types: `char` | `string`

`comparison` — Comparison to rank, pass, or fail matching networks
`'<'` | `'>'`

Comparison to rank, pass, or fail matching networks, specified as `'<'` or `'>'`.

Data Types: `char` | `string`

`targetdb` — Cut-off that determines particular performance goal
scalar

Cut-off that determines a particular performance goal, specified as a scalar in dB. The `targetdb` is shaded when you use the `rfplot` function. The shade is green when the matching network meets the performance goal. The shade is red when the matching network does not meet the performance goal.

Data Types: `double`

`band` — Frequency range in which performance goal or specifications are applied to matching network
vector

Frequency range in which the performance goal or the specifications are applied to matching network, specified as a vector with each element in Hz.

Data Types: `double`

`weight` — Weight factor of each performance goal
scalar

Weight factor of each performance goal when you specify more than one goal, specified as a scalar in the range of 0 to 1.

Data Types: `double`

Output Arguments

`mnobjupdated` — Matching network updated according to evaluation parameters
`matchingnetwork` object

Matching network updated according to evaluation parameters, returned as a `matchingnetwork` object.

Version History

Introduced in R2019a

See Also

`matchingnetwork` | `exportCircuits` | `circuitDescriptions` | `getEvaluationParameters` | `clearEvaluationParameter` | `rfplot` | `smithplot` | `sparameters`

circuitDescriptions

Tables describing each created matching network's topology and performance

Syntax

```
c = circuitDescriptions(mnobj)
[c,p] = circuitDescriptions(mnobj)
```

Description

`c = circuitDescriptions(mnobj)` returns a table, `c` describing each possible matching network in a row. The source is attached to the leftmost component on the table line, and the load is attached to the rightmost component on the table line.

`[c,p] = circuitDescriptions(mnobj)` returns a table, `c` describing each possible matching network in a row and a table, `p` with details about the circuit performance.

Examples

Description and Performance of Matching Network

Show the description and performance of each possible matching network.

```
matchnet = matchingnetwork;
[c,p] = circuitDescriptions(matchnet)
```

```
c=2×5 table
                circuitName    component1Type    component1Value    component2Type    component2
                ___________    ______________    _______________    ______________    __________

    Circuit 1     "auto_2"       "Shunt L"             Inf           "Series L"              0
    Circuit 2     "auto_1"       "Series L"              0           "Shunt L"             Inf


p=2×4 table
                circuitName    evaluationPassed    testsFailed     performanceScore
                ___________    ________________    ___________    ________________

    Circuit 1     "auto_2"         {["Yes"]}       {0x0 double}       {[3.0000]}
    Circuit 2     "auto_1"         {["Yes"]}       {0x0 double}       {[3.0000]}
```

Input Arguments

`mnobj` — Matching network

`matchingnetwork` object

Matching network, specified as a `matchingnetwork` object.

Data Types: `char` | `string`

Output Arguments

c — Description of each possible matching network
table

Description of each possible matching network, returned as a table.

p — Description of performance of each possible matching network
table

Description of the performance of each possible matching network, returned as a table. This table tells you if the matching network meets all performance goals, which performance goals have failed, and the overall performance metric used for ranking.

Version History

Introduced in R2019a

See Also

`matchingnetwork` | `addEvaluationParameter` | `getEvaluationParameters` | `clearEvaluationParameter` | `exportCircuits` | `rfplot` | `smithplot` | `sparameters`

clearEvaluationParameter

Delete one or more performance goals

Syntax

```
mnobjupdated = clearEvaluationParameter(mnobj,indices)
```

Description

`mnobjupdated = clearEvaluationParameter(mnobj,indices)` deletes one or more performance goals listed in `indices` and returns an updated matching network. Use `getEvaluationParameters` to view performance specifications and find each goal's index.

Examples

Matching Network From Dipole Antenna

Create a dipole antenna and create the S-parameters of the antenna. This example requires Antenna Toolbox.

```
d       = dipole('Length', 0.103, 'Width',0.0022);
freq    = linspace(0.5e9,2.5e9,1001);
sd      = sparameters(d, freq);
```

Alternatively, load S-Parameters from the MAT file

```
% load('sparams_dipole.mat')
```

Create a matching network from the S-parameters.

```
n = matchingnetwork('LoadImpedance',sd,'Components',3,...
    'LoadedQ',7,'CenterFrequency',2e9);
```

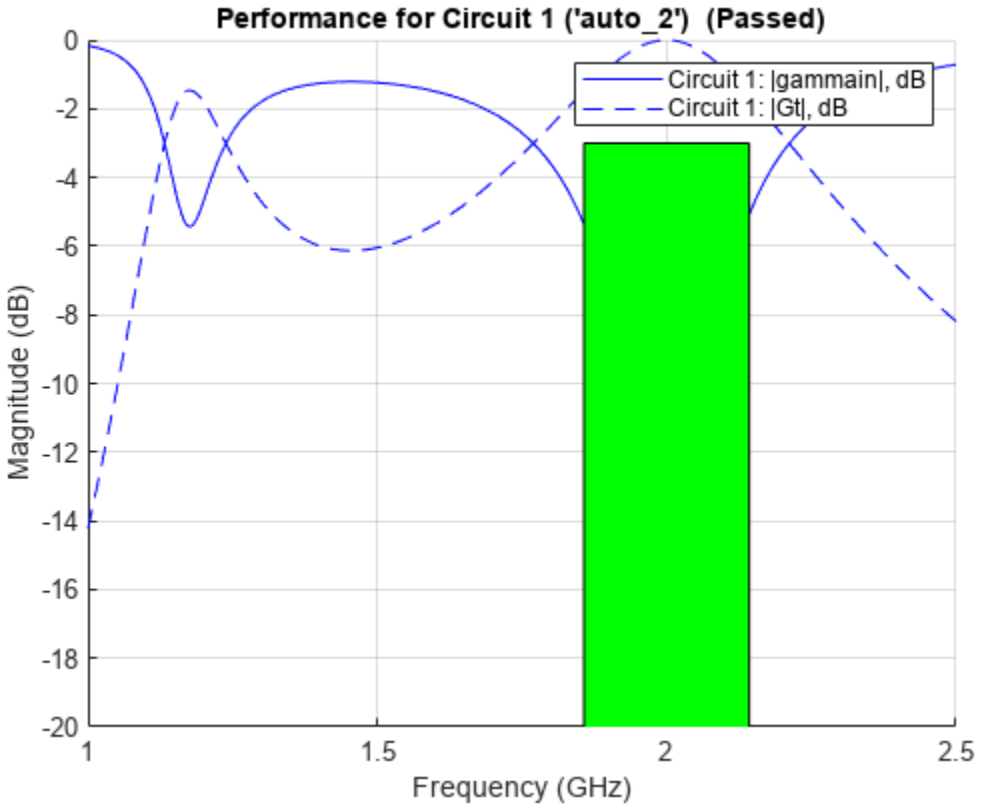
Get the evaluation parameters of the network.

```
t = getEvaluationParameters(n)
```

```
t=1×6 table
    Parameter    Comparison     Goal                Band                 Weight       Source
    _________    __________    ______    ____________________________    ______    ____________

     {'Gt'}        {'>'}       {[-3]}    {[1.8571e+09 2.1429e+09]}    {[1]}     {'Automatic'}
```

Plot the reflection coefficient and transducer gain of the matching network circuit 1 , at a frequency range of 1 GHz to 2.5 GHz.

```
rfplot(n, (1e9:0.001e9:2.5e9),1);
```

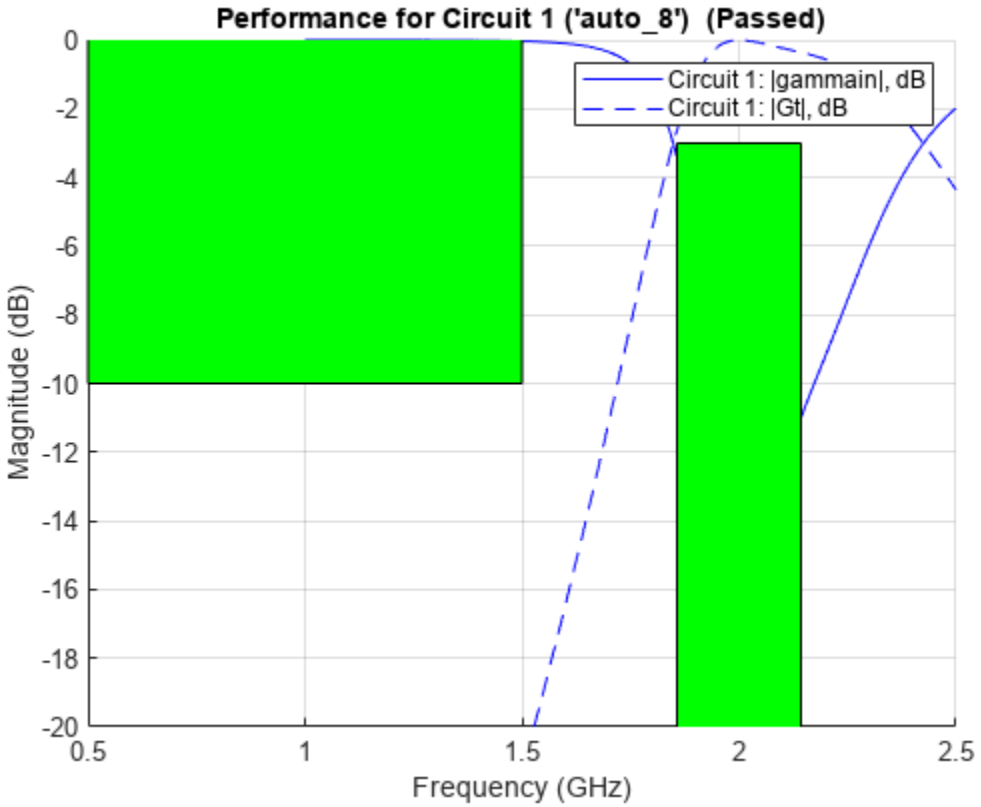


Add a new evaluation parameter to compare the transducer gain to have a cut-off of less than -10 dB. Use a frequency range of 0.5 GHz to 1.5 GHz. Plot the comparisons.

```
n = addEvaluationParameter(n, 'Gt', '<', -10, [0.5e9 1.5e9], 1);
t = getEvaluationParameters(n)
```

```
t=2×6 table
    Parameter    Comparison     Goal              Band                    Weight          Source
    _________    __________    _______    ____________________________    ______    _______________

     {'Gt'}         {'>'}       {[ -3]}    {[1.8571e+09 2.1429e+09]}      {[1]}     {'Automatic'
     {'Gt'}         {'<'}       {[-10]}    {[ 500000000 1.5000e+09]}      {[1]}     {'User-specifie
```

```
rfplot(n, (1e9:0.001e9:2.5e9),1);
```



Clear evaluation parameters.

```
n = clearEvaluationParameter(n,1);
t = getEvaluationParameters(n)
```

```
t=1×6 table
    Parameter    Comparison     Goal                Band                 Weight          Source
    _________    __________    _______    ____________________________    ______    _______________

     {'Gt'}        {'<'}       {[-10]}    {[500000000 1.5000e+09]}    {[1]}     {'User-specified
```

Input Arguments

`mnobj` — Matching network

`matchingnetwork` object

Matching network, specified as a `matchingnetwork` object.

Data Types: `char` | `string`

`indices` — Indices of matching network circuit

list

Indices of the matching network circuit, specified as a list.

Data Types: `char` | `string`

Output Arguments

`mnobjupdated` — Matching network updated according to evaluation parameters
`matchingnetwork` object

Matching network updated according to evaluation parameters, returned as a `matchingnetwork` object.

Version History

Introduced in R2019a

See Also

`matchingnetwork` | `addEvaluationParameter` | `circuitDescriptions` | `getEvaluationParameters` | `exportCircuits` | `rfplot` | `smithplot` | `sparameters`

getEvaluationParameters

Table of evaluation parameters currently used to rank and pass or fail matching network designs

Syntax

```
c = getEvaluationParameters(mnobj)
```

Description

`c = getEvaluationParameters(mnobj)` returns a table of evaluation parameters that are currently used to rank and pass or fail matching networks.

Examples

Matching Network Evaluation Parameters

Show the evaluation parameters of a default matching network.

```
matchnet = matchingnetwork;
c = getEvaluationParameters(matchnet)
```

```
c=1×6 table
    Parameter    Comparison     Goal               Band                 Weight       Source
    _________    __________    ______    ____________________________    ______    ____________

     {'Gt'}        {'>'}       {[-3]}    {[950000000 1.0500e+09]}    {[1]}     {'Automatic'}
```

Matching Network From Dipole Antenna

Create a dipole antenna and create the S-parameters of the antenna. This example requires Antenna Toolbox.

```
d       = dipole('Length', 0.103, 'Width',0.0022);
freq    = linspace(0.5e9,2.5e9,1001);
sd      = sparameters(d, freq);
```

Alternatively, load S-Parameters from the MAT file

```
% load('sparams_dipole.mat')
```

Create a matching network from the S-parameters.

```
n = matchingnetwork('LoadImpedance',sd,'Components',3,...
    'LoadedQ',7,'CenterFrequency',2e9);
```

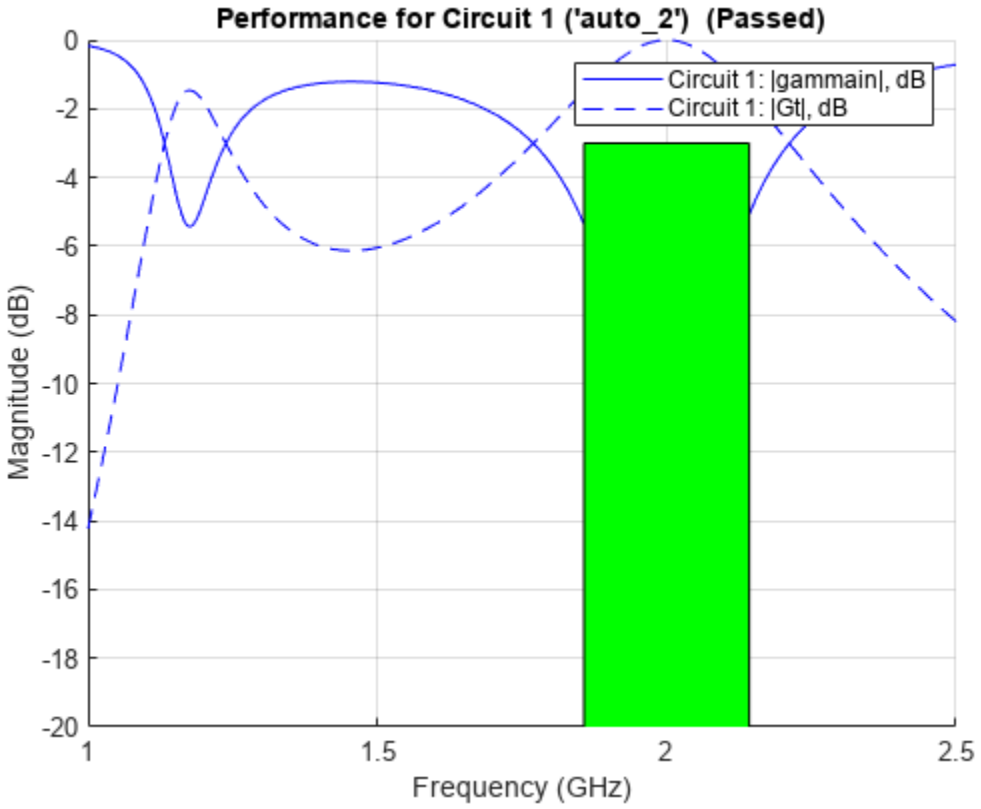
Get the evaluation parameters of the network.

```
t = getEvaluationParameters(n)
```

```
t=1×6 table
    Parameter    Comparison      Goal              Band                    Weight       Source
    _________    __________    _______    ____________________________    _______    ____________

     {'Gt'}        {'>'}       {[-3]}     {[1.8571e+09 2.1429e+09]}       {[1]}     {'Automatic'}
```

Plot the reflection coefficient and transducer gain of the matching network circuit 1 , at a frequency range of 1 GHz to 2.5 GHz.

```
rfplot(n, (1e9:0.001e9:2.5e9),1);
```

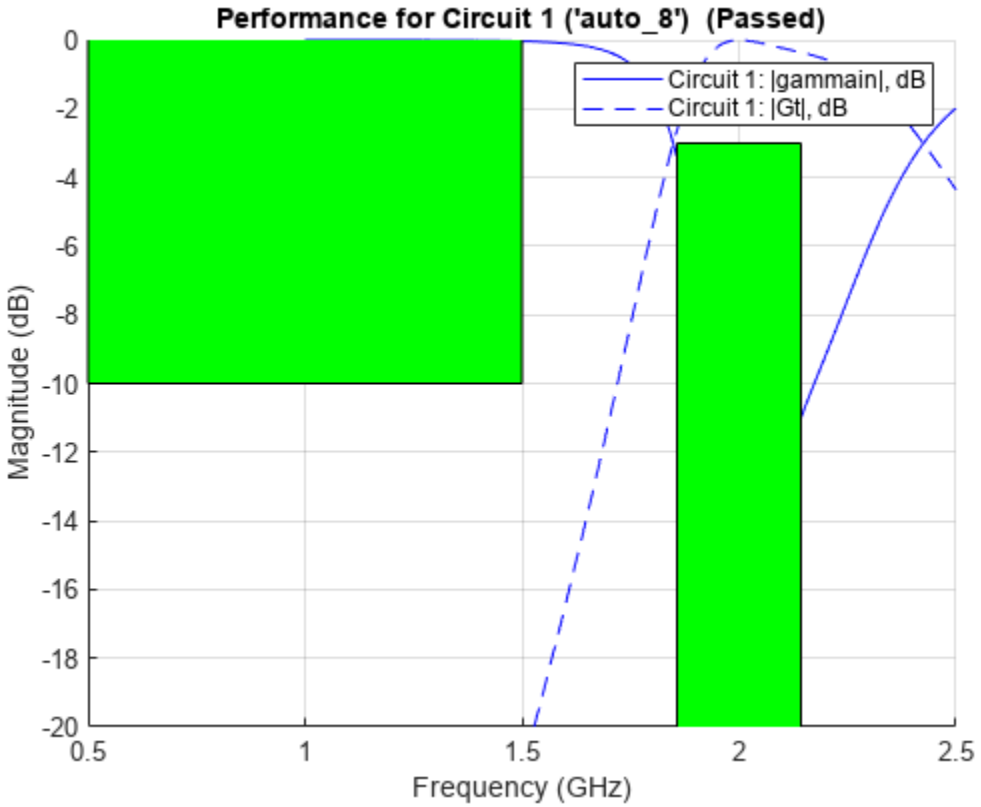


Add a new evaluation parameter to compare the transducer gain to have a cut-off of less than -10 dB. Use a frequency range of 0.5 GHz to 1.5 GHz. Plot the comparisons.

```
n = addEvaluationParameter(n, 'Gt', '<', -10, [0.5e9 1.5e9], 1);
t = getEvaluationParameters(n)
```

```
t=2×6 table
    Parameter    Comparison      Goal              Band                    Weight         Source
    _________    __________    _______    ____________________________    _______    _______________

     {'Gt'}        {'>'}       {[ -3]}    {[1.8571e+09 2.1429e+09]}       {[1]}     {'Automatic'
     {'Gt'}        {'<'}       {[-10]}    {[ 500000000 1.5000e+09]}       {[1]}     {'User-specifie
```

```
rfplot(n, (1e9:0.001e9:2.5e9),1);
```



Clear evaluation parameters.

```
n = clearEvaluationParameter(n,1);
t = getEvaluationParameters(n)
```

```
t=1×6 table
    Parameter    Comparison     Goal                Band                  Weight          Source
    _________    __________    _______    ____________________________    ______    _________________

     {'Gt'}        {'<'}       {[-10]}    {[500000000 1.5000e+09]}        {[1]}     {'User-specified
```

Input Arguments

`mnobj` — Matching network

`matchingnetwork` object

Matching network, specified as a `matchingnetwork` object.

Data Types: `char` | `string`

Output Arguments

`c` — Evaluation parameters currently used to rank and pass or fail matching networks

table

Evaluation parameters currently used to rank and pass or fail matching networks, returned as a table.

Version History

Introduced in R2019a

See Also

`matchingnetwork` | `addEvaluationParameter` | `circuitDescriptions` | `clearEvaluationParameter` | `exportCircuits` | `rfplot` | `smithplot` | `sparameters`

exportCircuits

Select and export generated matching networks as circuit objects from an existing matching network object

Syntax

```
cktout = exportCircuits(mnobj)
cktout = exportCircuits(mnobj,indexlist)
```

Description

`cktout = exportCircuits(mnobj)` exports the best matching network as a circuit object.

`cktout = exportCircuits(mnobj,indexlist)` exports only matching networks specified in the index list as an array of circuit objects.

Examples

Export a matching network circuit from the matchingnetwork object

Example shows how to export an circuit object from an matchingnetwork object

Create a default matchingnetwork object.

```
matchnet = matchingnetwork;
```

Export the best solution among the generated circuits (which is circuit #1 in the list):

```
cktout1 = exportCircuits(matchnet)
```

```
cktout1 =
  circuit: Circuit element

    ElementNames: {'L'  'L_1'}
        Elements: [1x2 inductor]
           Nodes: [1 2 3]
            Name: 'auto_2'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Note: To see the list of generated circuits use `c = circuitDescriptions(matchnet)`

Export circuit #2 from the generated matching network solutions using:

```
cktout2 = exportCircuits(matchnet,2)
```

```
cktout2 =
  circuit: Circuit element

    ElementNames: {'L'  'L_1'}
```

```
        Elements: [1x2 inductor]
           Nodes: [1 2 3]
            Name: 'auto_2'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Alternatively, use matchnet.Circuit(2).

Export Matching Network Circuits

This example shows how to export circuit objects from an `matchingnetwork` object

Design Matching Network

Create an `matchingnetwork` object with 3 components.

```
matchnet = matchingnetwork('Components',3);
```

Export circuits

Export circuits #3 and #4 from the list of generated circuit solutions. The circuits are exported as an array of circuit objects.

```
cktout = exportCircuits(matchnet,[3 4])
```

```
cktout =
  2x1 circuit array with properties:

    Name
    Terminals
    ParentNodes
    ParentPath
```

Input Arguments

`mnobj` — Matching network

`matchingnetwork` object

Matching network, specified as a `matchingnetwork` object.

Data Types: `char` | `string`

`indexlist` — Index list of matching networks to export as circuits

scalar | vector

Index list of matching networks to export as circuits, specified as a scalar or a vector.

Data Types: `double`

Note To export as `RF Circuit Objects` (`rfckt`) objects, type 'help exportCircuits'

Version History

Introduced in R2019a

See Also

`addEvaluationParameter` | `circuitDescriptions` | `getEvaluationParameters` | `clearEvaluationParameter` | `rfplot` | `smithplot` | `sparameters` | `matchingnetwork`

ispassive

Return true if `rationalfit` output is passive at all frequencies

Syntax

```
result = ispassive(fit)
[result,maxfreq,maxvalue] = ispassive(fit)
```

Description

`result = ispassive(fit)` determines if the rational fit input, `fit` is passive at all frequencies (0,Inf).

`[result,maxfreq,maxvalue] = ispassive(fit)` determines if the rational fit input, `fit` is passive at all frequencies (0,Inf).

Examples

Check and Plot Passivity of N-by-N Rationalfit

Read the file, `passive.s2p` and fit the 2-by-2 S-parameters.

```
S = sparameters('passive.s2p');
fit = rationalfit(S);
```

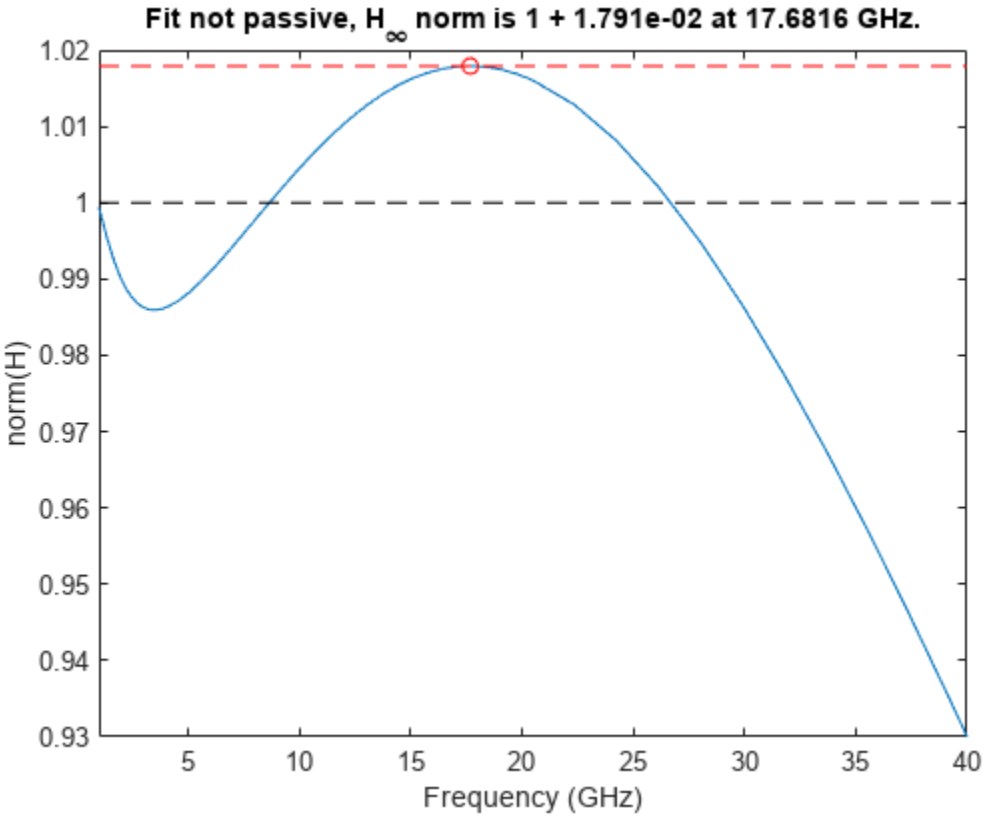
Test the passivity of the 2-by-2 fit.

```
ispassive(fit)
```

```
ans = logical
   0
```

Plot the passivity of the 2-by-2 fit.

```
figure
passivity(fit,[1e9 40e9])
```



Input Arguments

`fit` — Rational fit object
N-by-*N* array

Rational fit object, specified as an *N*-by-*N* array of `rfmodel.rational` objects as returned by the `rationalfit` function or a `rational` object.

Output Arguments

`result` — Result of passivity test
`true` | `false`

Result of passivity test of the `rfmodel.rational` object, returned as `true` or `false`. The result tells you if the `fit` can result in unstable simulations.

`maxfreq` — Frequency at which norm(H) is equal to maximum value
scalar

Frequency at which norm(H) is equal to maximum value, returned as a scalar in hertz.

`maxvalue` — Maximum norm(H) over frequencies (0,Inf)
scalar

Maximum norm(H) over frequencies (0,Inf), returned as a scalar.

Version History

Introduced in R2010a

See Also

`rationalfit` | `passivity` | `makepassive`

makepassive

Enforce passivity of `rationalfit` output or a rational object

Syntax

```
pfit = makepassive(fit,s)
pfit = makepassive(fit,s,'Display','on')
```

Description

`pfit = makepassive(fit,s)` produces a passive fit by modifying the input, `fit` while optimally matching the data of S-parameter input, `s`. `makepassive` function does modifies the residues of the `fit` to make it passive.

`pfit = makepassive(fit,s,'Display','on')` solves as above, but turns on the display of iteration information. The default for `'Display'` is `'off'`.

Examples

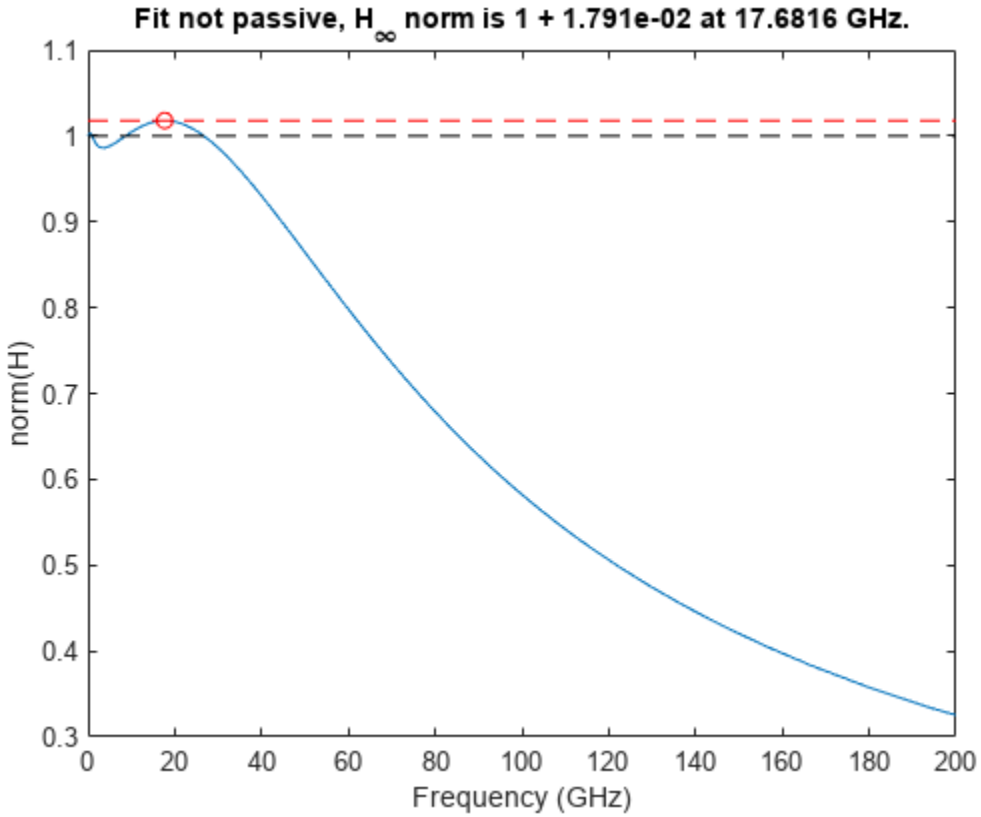
Make Passive Fit of S-Parameter Rationalfit

Read a file named `passive.s2p` and fit the 2x2 S-parameters.

```
S = sparameters('passive.s2p');
fit = rationalfit(S);
```

Plot the passivity of the 2x2 fit, noting the violations.

```
figure
passivity(fit)
```



Optimize residues to produce a passive fit still close to S.

```
 pfit = makepassive(fit,S)
```

```
pfit =
  2x2 rfmodel.rational array with properties:

    A
    C
    D
    Delay
    Name
```

To display iteration information:

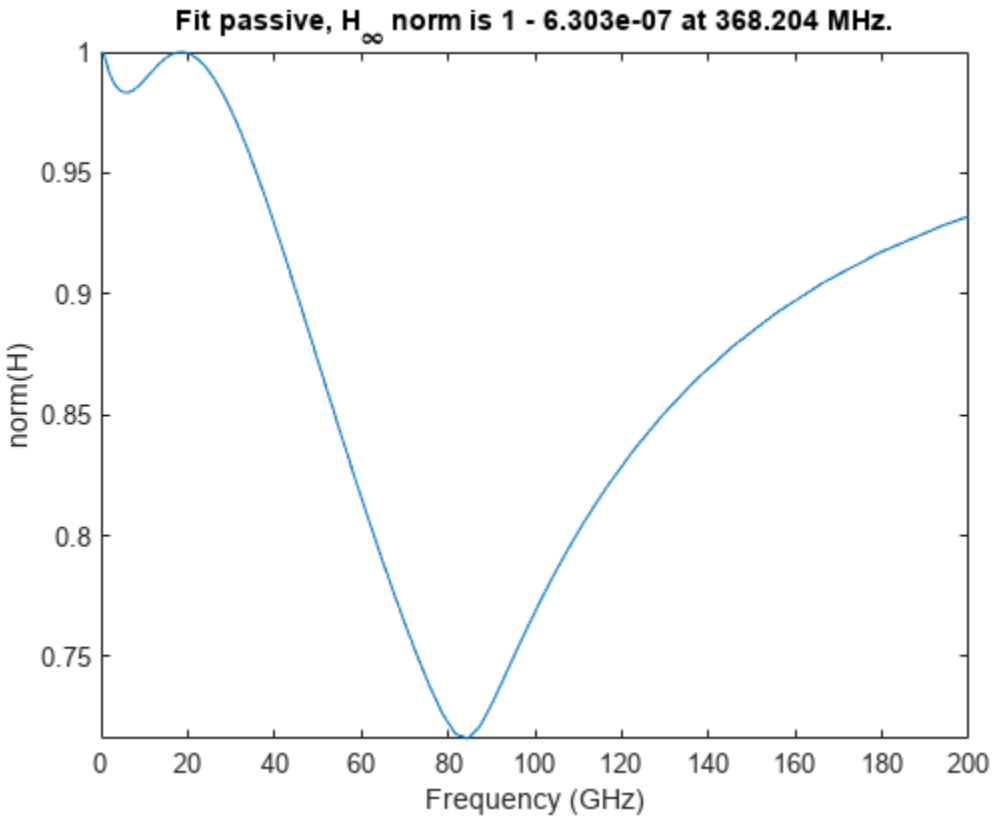
```
  pfit = makepassive(fit,S,'Display','on' )
```

```
ITER     H-INFTY NORM    FREQUENCY        ERRDB        CONSTRAINTS
0        1 + 1.791e-02    17.6816  GHz    -40.4702
1        1 + 2.878e-04    275.328  MHz    -40.9167    5
2        1 + 9.277e-05    365.606  MHz    -40.9092    7
3        1 - 6.303e-07    368.204  MHz    -40.9062    9

pfit =
  2x2 rfmodel.rational array with properties:

    A
    C
```

```
D
Delay
Name
```

Plot the passivity of the new fit.

```
figure
passivity(pfit)
```



Input Arguments

`fit` — `rfmodel.rational` or `rational` objects returned by rationalfit function or rational object

N-by-*N* array

N-by-*N* array, specified as a a `rfmodel.rational` objects returned by `rationalfit` or a `rational` object.

`s` — S-parameter object

network parameter object

S-parameters, specified as an RF Toolbox network parameter object. To create this type of object, use the `sparameters` function.

Output Arguments

`pfit` — `rfmodel.rational` objects
N-by-*N* array

`rfmodel.rational` objects, returned as *N*-by-*N* array.

Version History

Introduced in R2019a

See Also

`rationalfit` | `passivity` | `ispassive`

passivity

Plot passivity of *N*-by-*N* `rationalfit` function output

Syntax

```
passivity(fit)
passivity(fit,xlimits)
[maxfreq,maxvalue,freqs,ns] = passivity(fit)
```

Description

`passivity(fit)` plots the passivity of the input, `fit`, over a range of frequencies. Passivity is measured by computing the H-infinity norm of the fit. H-infinity norm is the maximum two-norm of the transfer function H over all the frequencies (0, Inf).

`passivity(fit,xlimits)` plots the passivity with X-axis limits of the plot.

`[maxfreq,maxvalue,freqs,ns] = passivity(fit)` returns the data that is used to generate the plot.

Examples

Check and Plot Passivity of N-by-N Rationalfit

Read the file, `passive.s2p` and fit the 2-by-2 S-parameters.

```
S = sparameters('passive.s2p');
fit = rationalfit(S);
```

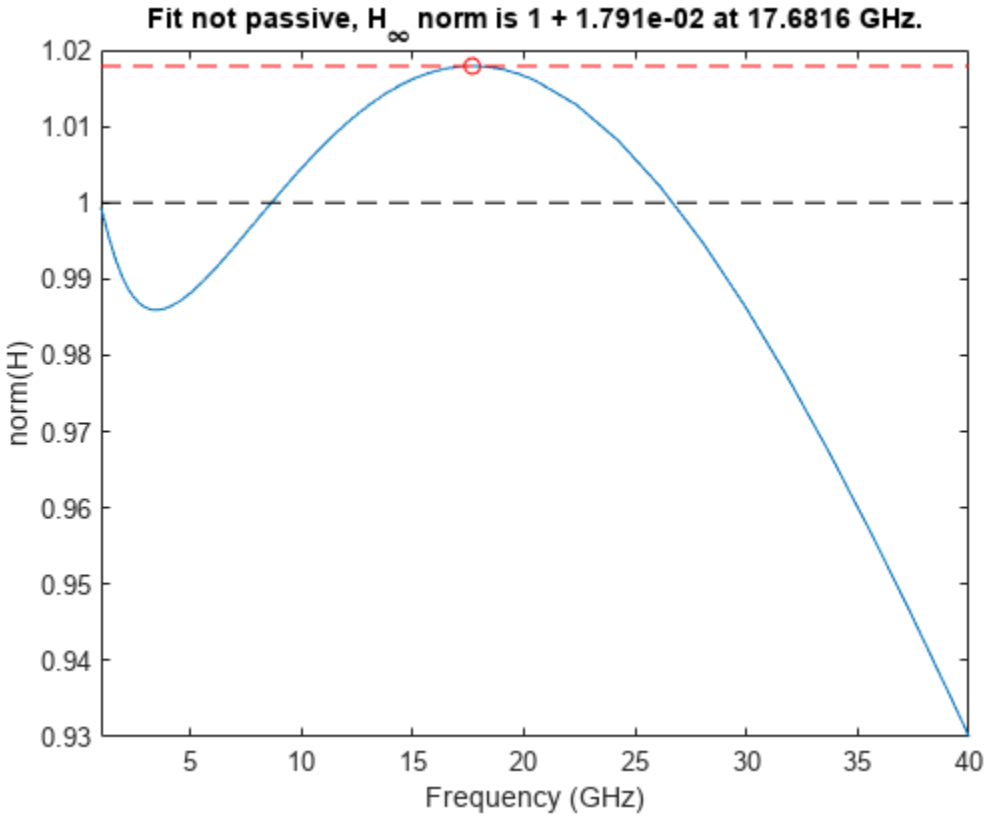
Test the passivity of the 2-by-2 fit.

```
ispassive(fit)
```

```
ans = logical
   0
```

Plot the passivity of the 2-by-2 fit.

```
figure
passivity(fit,[1e9 40e9])
```



Input Arguments

`fit` — Rational fit of S-parameters of passive elements

N-by-*N* array

Rational fit of S-parameters of passive elements, specified as an *N*-by-*N* array of `rfmodel.rational` objects as returned by the `rationalfit` function or a `rational` object with S-parameters as input.

Data Types: `double`

`xlimits` — X-axis limits of plot

1-by-2 vector

X-axis limits of the plot, specified as a 1-by-2 vector.

Data Types: `double`

Output Arguments

`maxvalue` — Maximum norm(H) over frequencies (0, Inf)

scalar

Maximum norm(H) over the frequencies (0, Inf), returned as a scalar.

Data Types: `double`

`maxfreq` — Frequency at which norm(H) is equal to maximum value
scalar

Frequency at which norm(H) is equal to maximum value, returned as a scalar in hertz.

Data Types: `double`

`ns` — Two-norm of transfer function H
column vector

Two-norm of the transfer function H, returned as a column vector.

Data Types: `double`

`freqs` — Frequency values
column vector

Frequency values, returned as a column vector.

Data Types: `double`

Version History

Introduced in R2019a

See Also

`ispassive` | `makepassive` | `rationalfit`

passivity

Plot passivity of S-parameters object

Syntax

```
passivity(s)
[ns,violationindex] = passivity(s)
```

Description

`passivity(s)` plots the passivity of S-parameter object, `s` over a range of frequencies specified in `s.Frequencies`. Passivity is measured by the two-norm of the frequency-dependent transfer function H(f).

`[ns,violationindex] = passivity(s)` plots the passivity and returns the data of the passivity plot.

Examples

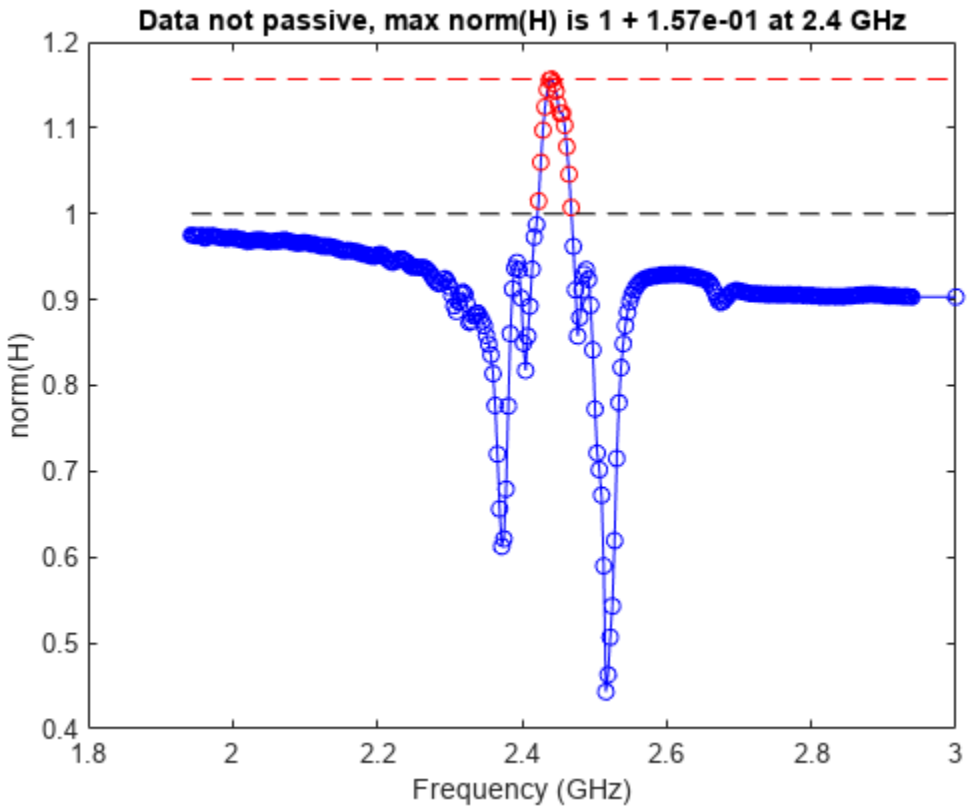
Plot Passivity of S-Parameters

Read the file, `sawfilter.s2p`.

```
S = sparameters('sawfilter.s2p');
```

Plot the passivity of the 2-by-2 S-parameters.

```
passivity(S)
```



Input Arguments

`s` — S-parameter object

s-parameter function object

S-parameters, specified as an RF Toolbox s-parameter function object. To create this type of object, use the `sparameters` function.

Output Arguments

`ns` — Two-norm transfer function H(f)

column vector

Two-norm transfer function H(f) at each frequency in `s.Frequencies`, returned as a column vector.

Data Types: `double`

`violationindex` — Frequencies indices where passivity is violated

column vector

Frequencies indices where passivity is violated (norm(H)>1), returned as a column vector.

Data Types: `double`

Version History

Introduced in R2019a

See Also

`ispassive` | `makepassive` | `rationalfit` | `sparameters`

makepassive

Make N-port S-parameters passive

Syntax

```
sparams_passive = makepassive(sparams)
```

Description

`sparams_passive = makepassive(sparams)` alters non-passive N-port S-parameters to make them passive. `makepassive` will error if the singular values at a frequency are too large. Reference impedance for S-parameters are assumed real and positive.

Examples

Make S-Parameters Passive

Convert `measured.s2p` to S-parameter object.

```
S = sparameters('measured.s2p');
```

Check if the S-parameter object is passive.

```
ispassive(S)
```

```
ans = logical
   0
```

Make the S-parameters data passive using `makepassive` function.

```
S_new = makepassive(S);
```

Check if the new S-parameter object is passive.

```
ispassive(S_new)
```

```
ans = logical
   1
```

Input Arguments

sparams — S-parameters

scalar S-parameters object | complex *N*-by-*N*-by-*K* array

S-parameters specified as one of the following:

- A scalar S-parameters object

- A complex *N*-by-*N*-by-*K* array for N-port S-parameters data.

Output Arguments

`sparams_passive` — Passive S-parameters
S-parameter object

Passive S-parameters, returned as an s-parameter object.

Note The `makepassive` function uses a purely mathematical method to calculate `sparams_passive`. As a result, the array `sparams_passive` does not represent the same network as `sparams`, unless `sparams` and `sparams_passive` are equal. The more closely `sparams` represents a passive network, the better the approximation `sparams_passive` is to that network. Therefore, `makepassive` generates the most realistic results when `sparams` is active only due to small numerical errors.

Version History

Introduced in R2010a

See Also

`sparameters` | `ispassive` | `rationalfit` | `passivity`

ispassive

Check passivity of N-port S-parameters

Syntax

```
[result, idx_nonpassive]= ispassive(sparams)
[ ___ ]= ispassive(sparams_data,'Impedance',z0)
```

Description

`[result, idx_nonpassive]= ispassive(sparams)` checks the passivity of S-parameters object or data. If the S-parameters are passive at every frequency, then the result is `true`. Otherwise, the result is `false`. It also optionally returns `idx_non_passive`, the indices of the non-passive S-parameters.

`[ ___ ]= ispassive(sparams_data,'Impedance',z0)` checks the passivity of N-port S-parameters data, that is referenced to the impedance value in the name-value pair, `'Impedance',z0`. The impedance can be in general complex.

Examples

Check Passivity of S-parameter Data

Read a Touchstone data file.

```
S = sparameters('measured.s2p');
```

Check the passivity of the S-parameters.

```
[passivevar,idx] = ispassive(S);
passivevar
```

```
passivevar = logical
   0
```

Get the nonpassive S-parameters.

```
if ~passivevar
    nonpassivevals = S.Parameters(:,:,idx);
end
```

Passivity of N-port S-parameter Data

Convert `passive.s2p` Touchstone file to an nport object.

```
nobj = nport('passive.s2p');
```

Convert the n-port object, `nobj` to s-parameter object.

```
sobj = sparameters(nobj)

sobj =
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [202x1 double]
     Parameters: [2x2x202 double]
      Impedance: 50

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

Find the passivity of n-port S-parameter data at impedance value, 60.

```
ispassive(sobj.Parameters,'Impedance',60)

ans = logical
   1
```

Input Arguments

`sparams` — S-parameters

scalar S-parameters object | complex *N*-by-*N*-by-*K* array

S-parameters, specified as one of the following:

- A scalar S-parameters object
- A complex *N*-by-*N*-by-*K* array for N-port S-parameters data.

`sparams_data` — S-parameter data referenced to z_0

N-by-*N*-by-*K* numeric matrix

S-parameter data referenced to z_0, specified as an *N*-by-*N*-by-*K* numeric matrix.

`z0` — Reference impedance

50 (default) | positive real scalar | vector

Reference impedance, specified as a positive real scalar.

Note `z0` must be a positive real scalar or vector. If `z0` is a vector, then the vector must be equal to the number of network parameter data points or frequency vector.

Output Arguments

`result` — Passivity of S-parameter data

logical scalar

Passivity of s-parameter data, returned as a logical scalar of `0` or `1`. If all the S-parameters are passive, then `ispassive` sets `flag` equal to `1` (`true`). Otherwise, `flag` is equal to `0` (`false`). If `flag` is `true`, `idx_non_passive` is empty.

`idx_nonpassive` — Indices that correspond to the frequencies
vector of numeric integers

Indices that correspond to the frequencies where the S-parameter is not passive, returned as vector of numeric integers.

Version History

Introduced in R2009b

See Also

`sparameters` | `passivity` | `rationalfit` | `makepassive`

rationalfit

Perform rationalfit on an S-parameters object

Syntax

```
[fit,errdb] = rationalfit(s,i,j)
[fit,errdb] = rationalfit(s)
```

Description

`[fit,errdb] = rationalfit(s,i,j)` uses the `rationalfit` function to construct a `fit`, an `rfmodel.rational` object fitting only the (i,j)th element of the S-parameter object `s`. This syntax is equivalent to `rationalfit(s.Frequencies,rfparam(s,i,j),...)`.

`[fit,errdb] = rationalfit(s)` uses the `rationalfit` function to construct a `fit`, an *N*-by-*N* of `rfmodel.rational` objects (sharing identical poles) fitting all the *N*-by-*N* elements of the S-parameter object `s`. You can directly pass this *N*-by-*N* `fit` to `freqresp`, `ispassive`, `passivity` functions. This syntax is equivalent to `rationalfit(s.Frequencies,s.Parameters)`.

Examples

Rationalfit of S-parameters

Read a file named `passive.s2p` and fit the 2-by-2 S-parameters.

```
S = sparameters('passive.s2p');
[fit,errdb] = rationalfit(S)
```

```
fit =
  2x2 rfmodel.rational array with properties:

    A
    C
    D
    Delay
    Name

errdb = -40.4702
```

Input Arguments

`s` — S-parameter object

network parameter object

S-parameters, specified as an RF Toolbox network parameter object. To create this type of object, use the `sparameters` function.

`i` — Row index

positive integer

Row index of data to plot, specified as a positive integer.

`j` — Column index
positive integer

Column index of data to plot, specified as a positive integer.

Output Arguments

`fit` — Rational function object
`rfmodel.rational` object

One or more rational function objects, returned as an *N*-by-*N* `rfmodel.rational` object. The number of dimensions in `data` determines the dimensionality of `h`.

`errdb` — Relative error
`-40` (default) | double

Relative error achieved, returned as a `double`, in dB.

Version History

Introduced before R2006a

See Also

`ispassive` | `makepassive` | `passivity` | `sparameters`

powergain

Calculate power gain from two-port S-parameters

Syntax

```
g = powergain(s_params,z0,zs,zl,'Gt')
g = powergain(s_params,z0,zs,'Ga')
g = powergain(s_params,z0,zl,'Gp')
g = powergain(s_params,'Gmag')
g = powergain(s_params,'Gmsg')

g = powergain(hs,zs,zl,'Gt')
g = powergain(hs,zs,'Ga')
g = powergain(hs,zl,'Gp')
g = powergain(hs,'Gmag')
g = powergain(hs,'Gmsg')
```

Description

`g = powergain(s_params,z0,zs,zl,'Gt')` calculates the transducer power gain of the 2-port network by:

$$G_t = \frac{P_L}{P_{\text{avs}}} = \frac{(1 - |\Gamma_S|^2)|S_{21}|^2(1 - |\Gamma_L|^2)}{|(1 - S_{11}\Gamma_S)(1 - S_{22}\Gamma_L) - S_{12}S_{21}\Gamma_S\Gamma_L|^2}$$

where,

- P_L is the output power and P_{avs} is the maximum input power.
- Γ_L and Γ_S are the reflection coefficients defined as:

$$\Gamma_S = \frac{Z_S - Z_0}{Z_S + Z_0}$$

$$\Gamma_L = \frac{Z_L - Z_0}{Z_L + Z_0}$$

`g = powergain(s_params,z0,zs,'Ga')` calculates the available power gain of the 2-port network by:

$$G_a = \frac{P_{\text{avn}}}{P_{\text{avs}}} = \frac{(1 - |\Gamma_S|^2)|S_{21}|^2}{|1 - S_{11}\Gamma_S|^2(1 - |\Gamma_{out}|^2)}$$

where

- P_{avn} is the available output power from the network.
- Γ_{out} is given by:

$$\Gamma_{\text{out}} = S_{22} + \frac{S_{12}S_{21}\Gamma_S}{1 - S_{11}\Gamma_S}$$

`g = powergain(s_params,z0,zl,'Gp')` calculates the operating power gain of the 2-port network by:

$$G_p = \frac{P_L}{P_{\text{in}}} = \frac{|S_{21}|^2(1-|\Gamma_L|^2)}{(1-|\Gamma_{\text{in}}|^2)|1-S_{22}\Gamma_L|^2}$$

where

- P_{in} is the input power.
- Γ_{in} is given by:

$$\Gamma_{\text{in}} = S_{11} + \frac{S_{12}S_{21}\Gamma_L}{1-S_{22}\Gamma_L}$$

`g = powergain(s_params,'Gmag')` calculates the maximum available power gain of the 2-port network. by:

$$G_{\text{mag}} = \frac{|S_{21}|}{|S_{12}|}\left(K - \sqrt{K^2-1}\right)$$

where K is the stability factor.

`g = powergain(s_params,'Gmsg')` calculates the maximum stable gain of the 2-port network by:

$$G_{msg} = \frac{|S_{21}|}{|S_{12}|}$$

`g = powergain(hs,zs,zl,'Gt')` calculates the transducer power gain of the network represented by the S-parameter object `hs`.

`g = powergain(hs,zs,'Ga')` calculates the available power gain of the network represented by the S-parameter object `hs`.

`g = powergain(hs,zl,'Gp')` calculates the operating power gain of the network represented by the S-parameter object `hs`.

`g = powergain(hs,'Gmag')` calculates the maximum available power gain of the network represented by the S-parameter object `hs`.

`g = powergain(hs,'Gmsg')` calculates the maximum stable gain of the network represented by the S-parameter object `hs`.

Examples

Power Gain of Two-Port Network

Calculate power gains for a sample 2-port network.

```
s11 = 0.61*exp(1j*165/180*pi);
s21 = 3.72*exp(1j*59/180*pi);
s12 = 0.05*exp(1j*42/180*pi);
s22 = 0.45*exp(1j*(-48/180)*pi);
```

```
sparam = [s11 s12; s21 s22];
z0 = 50;
zs = 10 + 1j*20;
zl = 30 - 1j*40;
```

Calculate the transducer power gain of the network

```
Gt = powergain(sparam,z0,zs,zl,'Gt')
```

```
Gt = 4.7066
```

Calculate the available power gain of the network

```
Ga = powergain(sparam,z0,zs,'Ga')
```

```
Ga = 11.4361
```

Note that, as expected, the available power gain is larger than the transducer power gain, `Gt`. The two become identical when `Gt` is measured with a matched load impedance:

```
zl_matched = gamma2z(gammaout(sparam, z0, zs)', z0);
Gt_zl_matched = powergain(sparam, z0, zs, zl_matched, 'Gt')
```

```
Gt_zl_matched = 11.4361
```

Calculate the operating power gain of the network

```
Gp = powergain(sparam,z0,zl,'Gp')
```

```
Gp = 10.5098
```

Note that, as expected, the operating power gain is larger than the transducer power gain, `Gt`. The two become identical when `Gt` is measured with a matched source impedance:

```
zs_matched = gamma2z(gammain(sparam, z0, zl)', z0);
Gt_zs_matched = powergain(sparam, z0, zs_matched, zl, 'Gt')
```

```
Gt_zs_matched = 10.5098
```

Calculate the maximum available power gain of the network

```
Gmag = powergain(sparam,'Gmag')
```

```
Gmag = 41.5032
```

Note that, as expected, the maximum available power gain is larger than the available power gain `Ga`, the transducer power gain, `Gt`, and the operating power gain, `Gp`. They all become identical when measured with simultaneously matched source and load impedances:

```
zs_matched_sim = gamma2z(gammams(sparam), z0);
zl_matched_sim = gamma2z(gammaout(sparam, z0, zs_matched_sim)', z0)
```

```
zl_matched_sim = 33.6758 + 91.4816i
```

That impedance can be also obtained directly using:

```
zl_matched_sim = gamma2z(gammaml(sparam), z0)
```

```
zl_matched_sim = 33.6758 + 91.4816i
```

```
Ga_matched_sim = powergain(sparam, z0, zs_matched_sim, 'Ga')
```

```
Ga_matched_sim = 41.5032
```

```
Gt_matched_sim = powergain(sparam, z0, zs_matched_sim, zl_matched_sim, 'Gt')
```

```
Gt_matched_sim = 41.5032
```

```
Gp_matched_sim = powergain(sparam, z0, zl_matched_sim, 'Gp')
```

```
Gp_matched_sim = 41.5032
```

When the scattering parameters represent a network that is *not* unconditionally stable, there is no set of source and load impedances that provide simultaneous matching. In this case, the maximum available power is infinite, but truly meaningless because the network is unstable.

To make the previously defined network conditionally stable, it is enough to increase the magnitude of the backward propagation scattering parameter, `s12`:

```
s12_cond_stable = 0.06*exp(1j*42/180*pi);
sparam_cond_stable = [s11 s12_cond_stable; s21 s22];
```

To verify that the network is conditionally stable, check that the stability factor, K, is smaller than 1:

```
K = stabilityk(sparam_cond_stable)
```

```
K = 0.9695
```

An attempt to calculate the maximum available gain of the network yields a NaN:

```
Gmag_cond_stable = powergain(sparam_cond_stable,'Gmag')
```

```
Gmag_cond_stable = NaN
```

Instead, the maximum stable gain, G_{msg}, should be used.

Calculate the maximum stable power gain of the network

```
Gmsg_cond_stable = powergain(sparam_cond_stable,'Gmsg')
```

```
Gmsg_cond_stable = 62.0000
```

The maximum stable power gain is only meaningful when the network is not unconditionally stable.

Input Arguments

`hs` — 2-port S-parameters

S-parameter object

2-port S-parameters, specified as an RF Toolbox S-parameter object.

`s_params` — 2-port S-parameters

array of complex numbers

2-port S-parameters, specified as a complex 2-by-2-by-N array.

`z0` — Reference impedance

50 (default) | positive scalar

Reference impedance in ohms, specified as a positive scalar. If the first input argument is an S-parameter object `hs`, the function uses `hs.Impedance` for the reference impedance.

`zl` — Load impedance
50 (default) | positive scalar

Load impedance in ohms, specified as a positive scalar.

`zs` — Source impedance
50 (default) | positive scalar

Source impedance in ohms, specified as a positive scalar.

Output Arguments

`g` — Power gain
vector

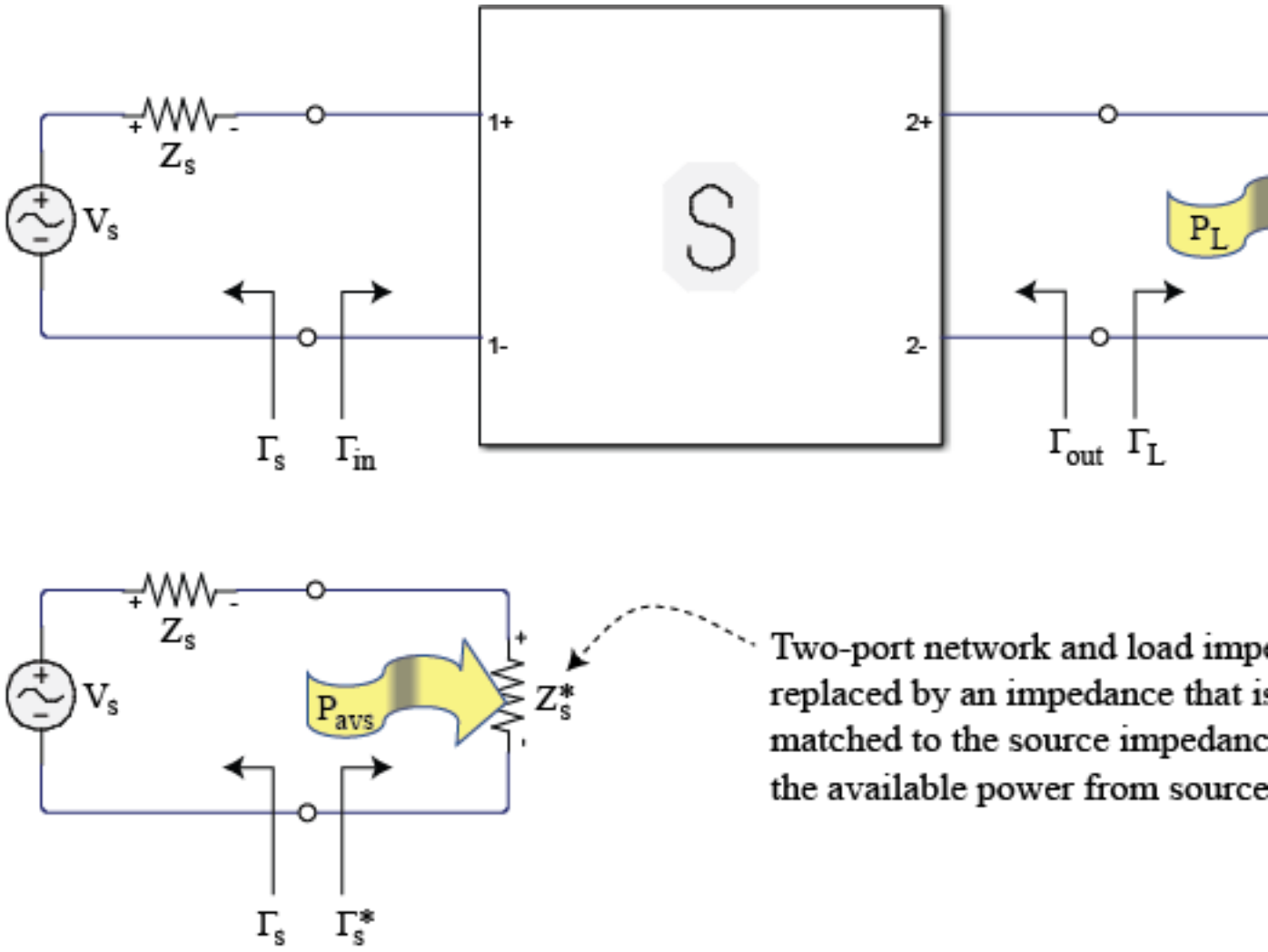
Unit less power gain values, returned as a vector. To obtain power gain in decibels, use `10*log10(g)`.

If the specified type of power gain is undefined for one or more of the specified S-parameter values in `s_params`, the `powergain` function returns `NaN`. As a result, `g` is either `NaN` or a vector that contains one or more `NaN` entries.

More About

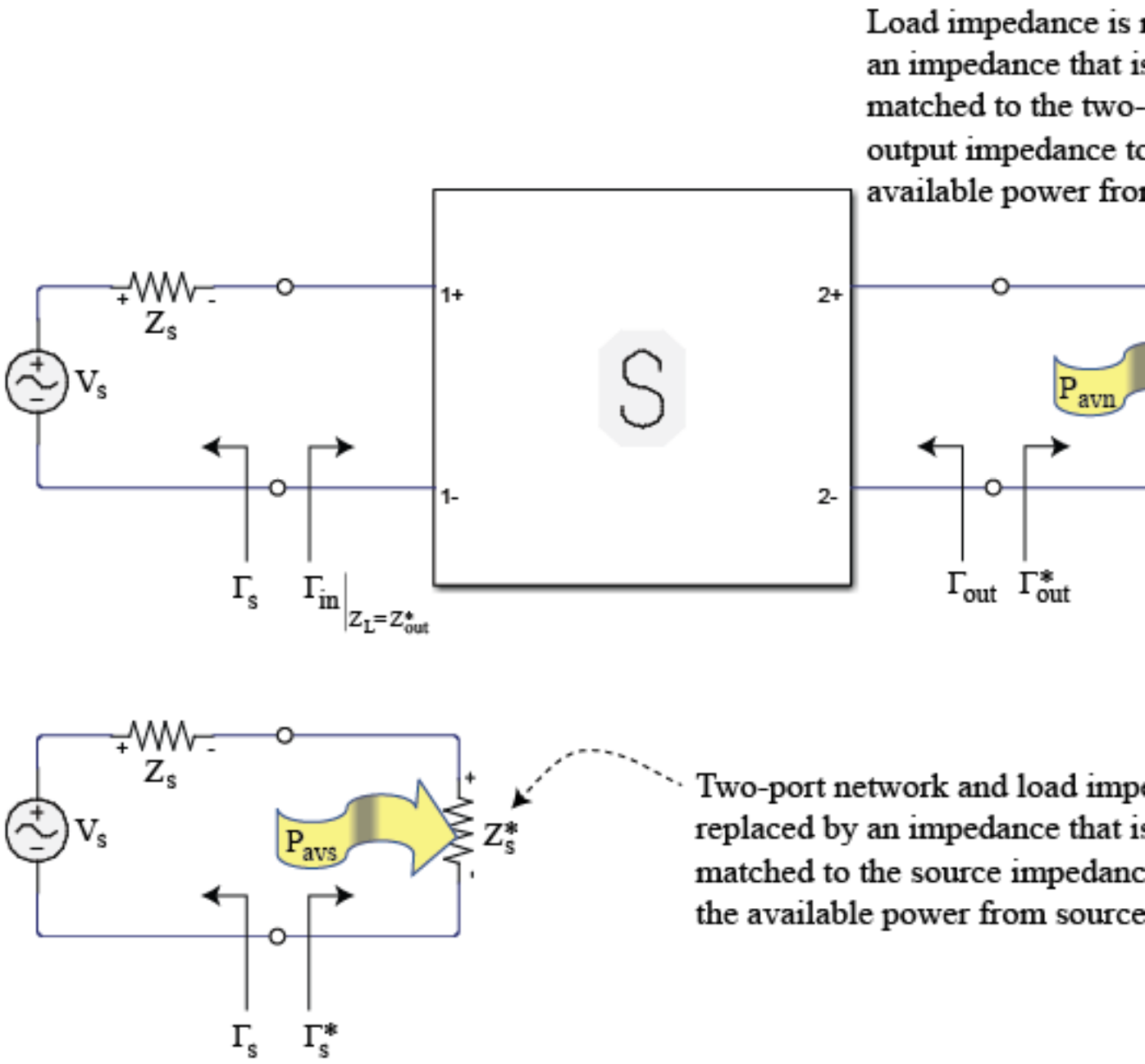
Transducer Power Gain

$G_t = P_L/P_{avs}$ is the ratio of power delivered to the load to the power available from the source. This depends on both Z_S and Z_L.



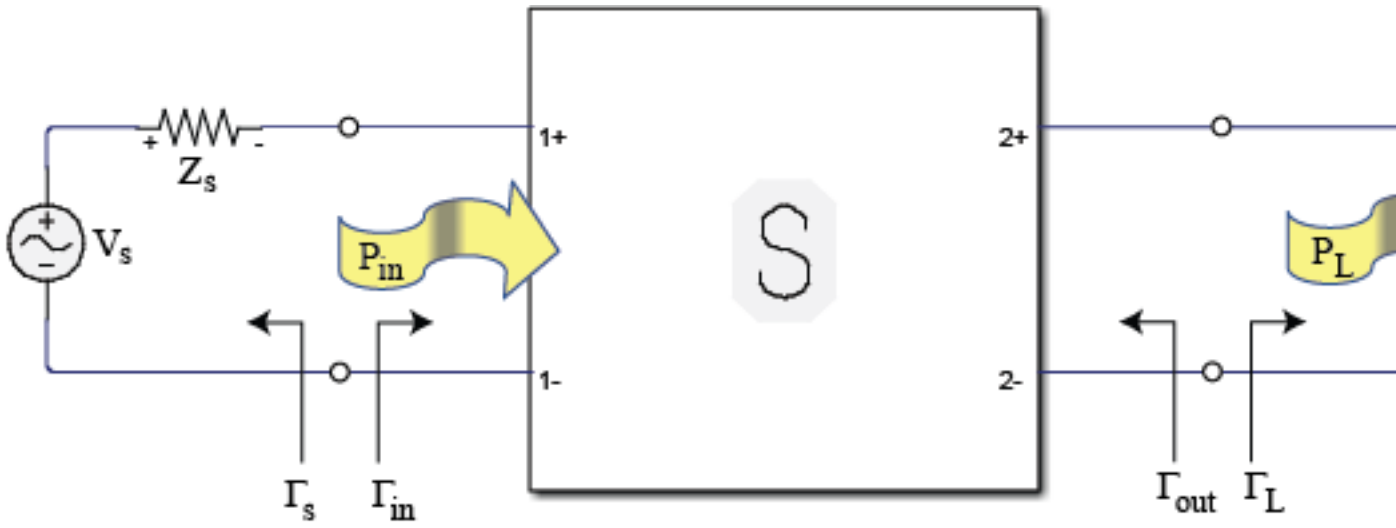
Available Power Gain

$G_a = P_{avn}/P_{avs}$ is the ratio of the power available from the two-port network to the power available from the source. Available gain is the transducer power gain when load impedance is equal to output impedance. Thus G_a depends only on Z_s.



Operating Power Gain

$G_p = P_L/P_{in}$ is the ratio of power dissipated in the load Z_L to the power delivered to the input of the two-port network. This gain is independent of Z_S, although some active circuits are strongly dependent on the input matching conditions.

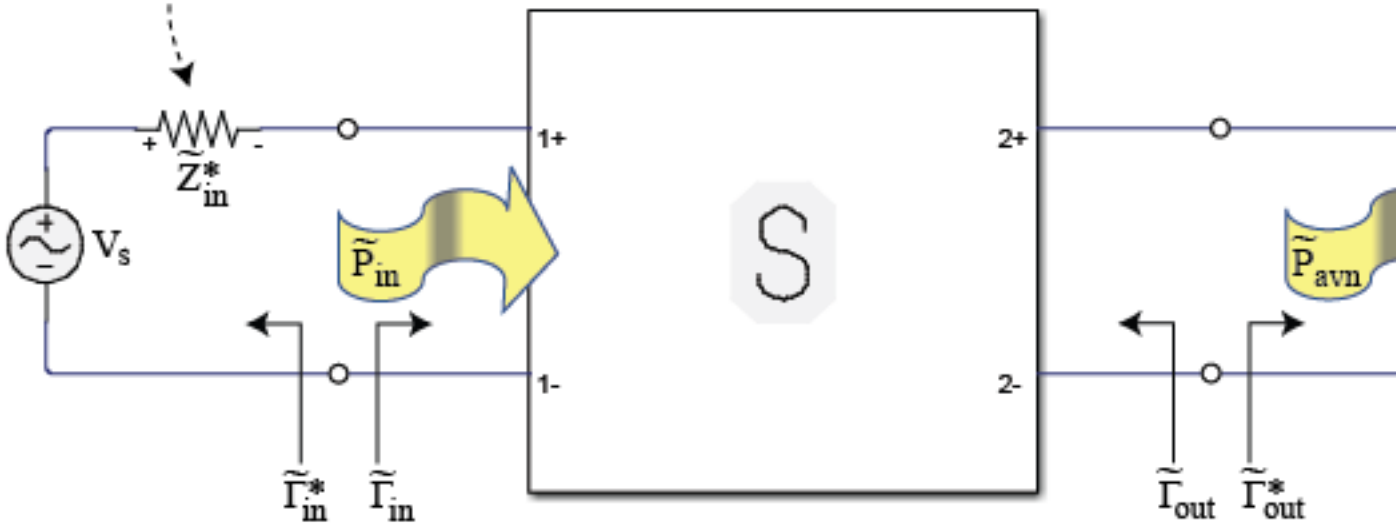


Maximum Available Power Gain and Maximum Stable Power Gain

Maximum available power gain, G_{mag} is $G_a$with matched input that is Z_S is equal to Z_{in}

In the case of conditionally stable two-port networks ($K<1$) where the maximum available power gain result is meaningless, the maximum stable power gain, G_{msg}, should be used.

Both the source and load impedances are replaced by impedances th
simultaneously conjugately-matched to the two-port network input a
output impedances, respectively, to measure the maximum available
gain of the system.



$$\tilde{Z}_{in} = Z_{in}\left(Z_L = \tilde{Z}_{out}^*\right) \iff \tilde{\Gamma}_{in} = \Gamma_{in}\left(Z_L = \tilde{Z}_{out}^*\right)$$

$$\tilde{Z}_{out} = Z_{out}\left(Z_s = \tilde{Z}_{in}^*\right) \iff \tilde{\Gamma}_{out} = \Gamma_{out}\left(Z_s = \tilde{Z}_{in}^*\right)$$

$$\tilde{P}_{in} = P_{in}\left(Z_s = \tilde{Z}_{in}^*, Z_L = \tilde{Z}_{out}^*\right) = P_{avs}\left(Z_s = \tilde{Z}_{in}^*\right)$$

$$\tilde{P}_{avn} = P_{avn}\left(Z_s = \tilde{Z}_{in}^*\right)$$

Version History

Introduced in R2007b

See Also

`gammaml` | `gammams` | `gammaout` | `gammain` | `z2gamma`

smithplot

Plot impedance transformation for selected matching network on smith chart

Syntax

```
smithplot(mnobj)
smithplot( ___ ,Name,Value)
```

Description

`smithplot(mnobj)` plots the impedance transformation from the source to load for the best matching network on a smith chart.

`smithplot( ___ ,Name,Value)` plots the impedance transformation from the source to load on a Smith chart with additional properties given by Name, Value pair . For instance `smithplot (MNOBJ, 'CircuitIndex',____, 'Z0', _____)` plots the impedance transformation for the selected matching network circuit at the characteristic impedance, `Z0` on a smith chart.

Properties not specified retain their default values.

Note For more name value pairs see, “Tips” on page 2-50.

Examples

Impedance Transformation of Matching Network

Create a new matchingnetwork and plot the impedance transformation of the second matching network circuit in the design.

```
f       = 2e9;          % Center frequency, Hz
Zin     = 150+75i;      % Source impedance, Ohms
Zl      = 75+15i;       % Load impedance, Ohms
Zo      = 75;           % characteristic impedance of system, Ohms
```

Design matching network object

```
n = matchingnetwork('CenterFrequency',f,'Bandwidth',1e8,           ...
  'LoadImpedance',Zl,'SourceImpedance',Zin,'Components',2);
```

View Circuit #2

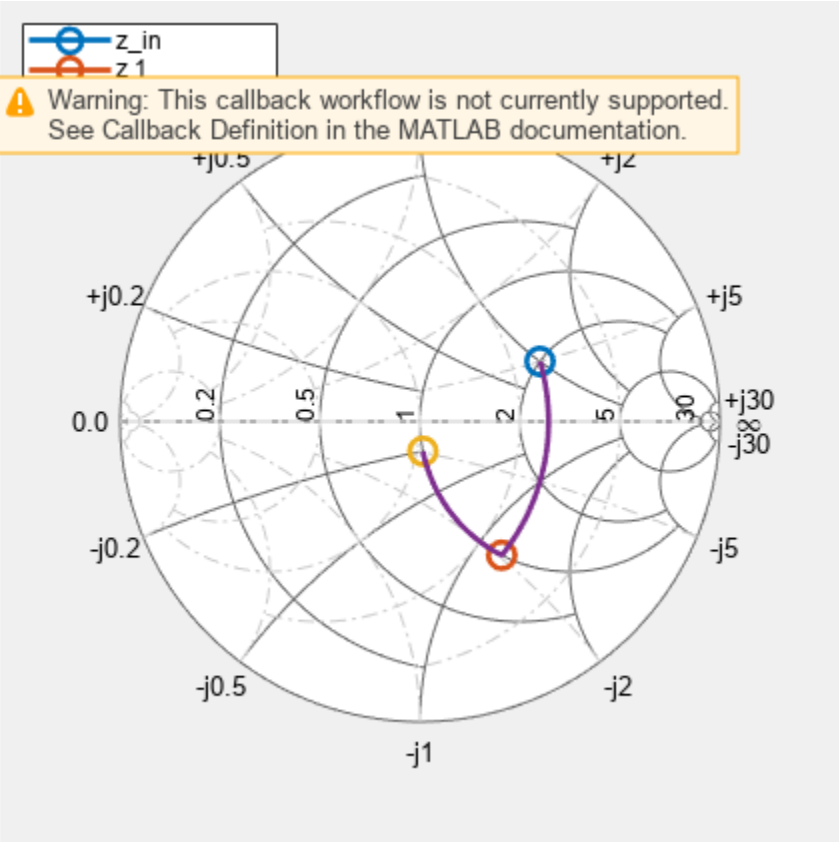
```
n.Circuit(2)
```

```
ans =
  circuit: Circuit element

    ElementNames: {'C'  'L'}
        Elements: [1x2 rf.internal.circuit.RLC]
           Nodes: [1 2 3]
```

```
        Name: 'auto_1'
    NumPorts: 2
   Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Impedance transformation of Circuit #2 on a smith chart

```
hs = smithplot(n, 'Z0', Zo, 'CircuitIndex',2);
```



Input Arguments

`mnobj` — Matching network

`matchingnetwork` object

Matching network, specified as a `matchingnetwork` object.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'CircuitIndex',1`

`CircuitIndex` — Selected matching network
1 (default) | real positive scalar

Selected matching network, specified as a positive scalar.

Data Types: `double`

`Z0` — Characteristic impedance
50 (default) | real positive scalar

Characteristic impedance, specified as scalar in ohms.

Data Types: `double`

Tips

- To list other property `Name,Value` pairs in `smithplot`, use `details(s)` where `s` is a `smithplot` object. You can use the properties to extract any data from the Smith chart.
- For a list of properties of `smithplot`, see .
- You can use the `smithplot` interactive menu to change the line and marker styles.

Version History

Introduced in R2019a

See Also

`add` | `replace` | `matchingnetwork` | `rfplot` | `sparameters`

rfplot

Plot input reflection coefficient and transducer gain of matching network

Syntax

```
rfplot(mnobj)
rfplot(mnobj,frequencylist)
rfplot(mnobj,frequencylist,circuitindices)
hline = rfplot(mnobj,frequencylist,circuitindices)
```

Description

`rfplot(mnobj)` plots the input reflection coefficient and transducer gain of the matching network when cascaded between source and load impedances. This plot compares the actual performance of the network against the performance goals.

`rfplot(mnobj,frequencylist)` plots using values calculated at each frequency in the `frequencylist`.

`rfplot(mnobj,frequencylist,circuitindices)` plots performances of matching networks specified in the `circuitindices`.

`hline = rfplot(mnobj,frequencylist,circuitindices)` returns a cell array of line handle vectors, one cell for each circuit plotted.

Examples

Matching Network From Dipole Antenna

Create a dipole antenna and create the S-parameters of the antenna. This example requires Antenna Toolbox.

```
d       = dipole('Length', 0.103, 'Width',0.0022);
freq    = linspace(0.5e9,2.5e9,1001);
sd      = sparameters(d, freq);
```

Alternatively, load S-Parameters from the MAT file

```
% load('sparams_dipole.mat')
```

Create a matching network from the S-parameters.

```
n = matchingnetwork('LoadImpedance',sd,'Components',3,...
    'LoadedQ',7,'CenterFrequency',2e9);
```

Get the evaluation parameters of the network.

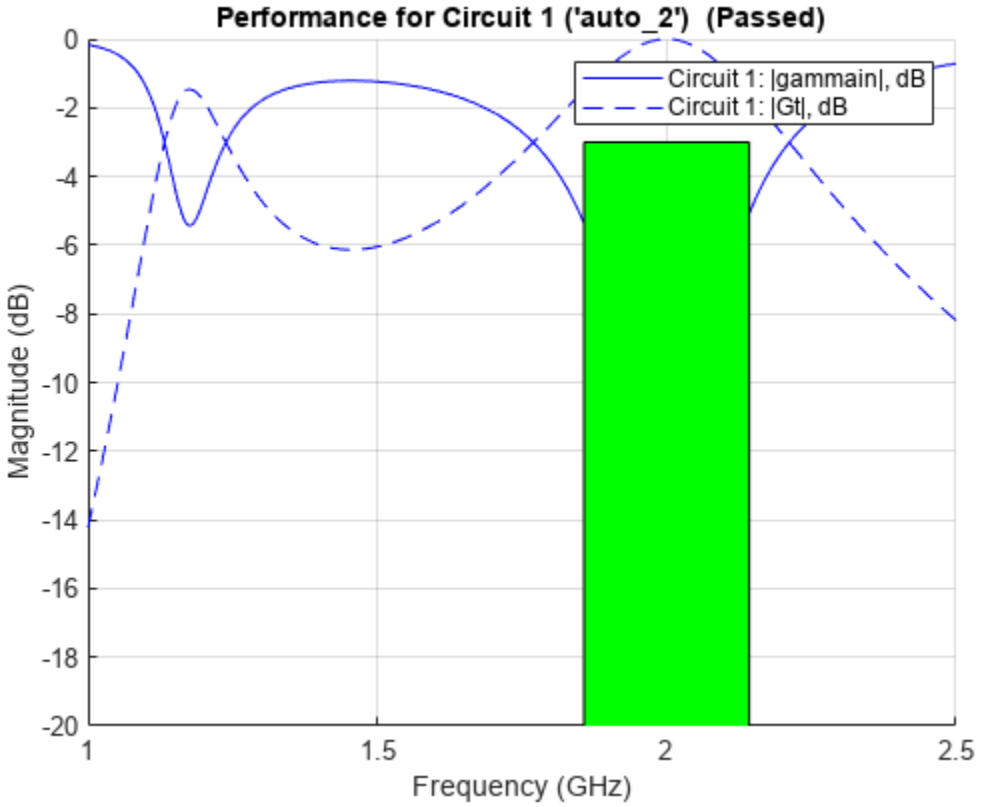
```
t = getEvaluationParameters(n)
```

```
t=1×6 table
    Parameter    Comparison     Goal              Band              Weight       Source
```

```
    ________    _________    _____    ____________________________    ______    _____________

    {'Gt'}      {'>'}        {[-3]}   {[1.8571e+09 2.1429e+09]}      {[1]}     {'Automatic'}
```

Plot the reflection coefficient and transducer gain of the matching network circuit 1 , at a frequency range of 1 GHz to 2.5 GHz.

```
rfplot(n, (1e9:0.001e9:2.5e9),1);
```

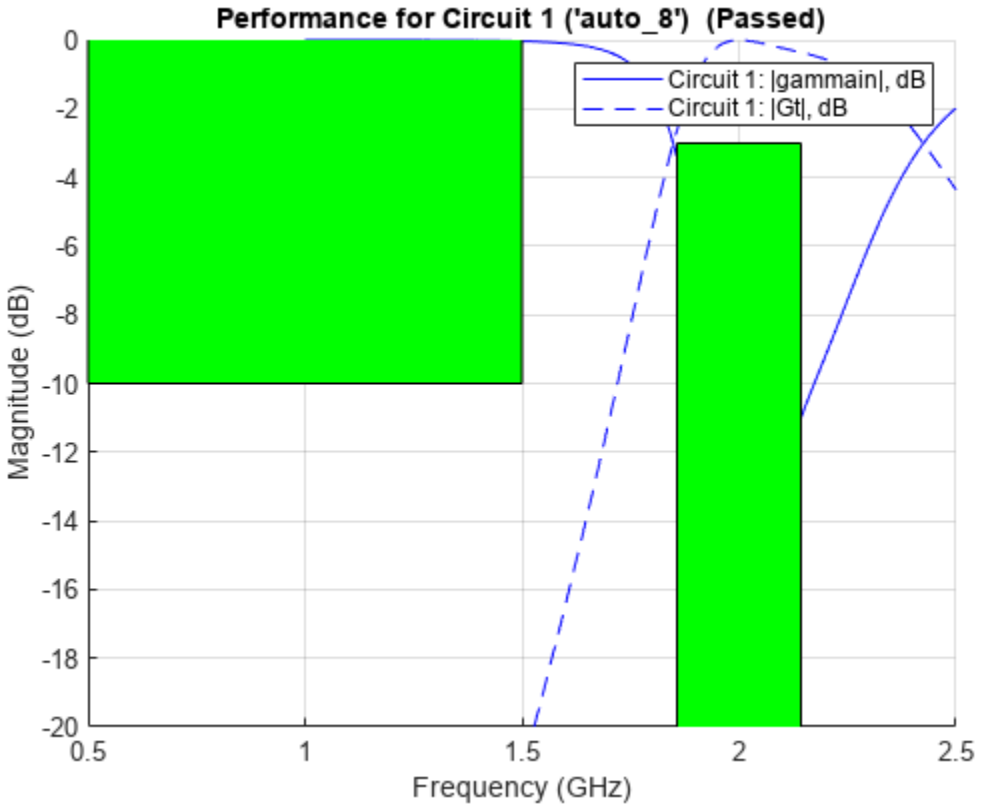


Add a new evaluation parameter to compare the transducer gain to have a cut-off of less than -10 dB. Use a frequency range of 0.5 GHz to 1.5 GHz. Plot the comparisons.

```
n = addEvaluationParameter(n, 'Gt', '<', -10, [0.5e9 1.5e9], 1);
t = getEvaluationParameters(n)
```

```
t=2×6 table
    Parameter    Comparison     Goal              Band                      Weight          Source
    _________    __________    _______    ____________________________    ______    _____________

     {'Gt'}        {'>'}       {[ -3]}    {[1.8571e+09 2.1429e+09]}      {[1]}     {'Automatic'
     {'Gt'}        {'<'}       {[-10]}    {[ 500000000 1.5000e+09]}      {[1]}     {'User-specifie
```

```
rfplot(n, (1e9:0.001e9:2.5e9),1);
```



Clear evaluation parameters.

```
n = clearEvaluationParameter(n,1);
t = getEvaluationParameters(n)
```

```
t=1×6 table
    Parameter    Comparison     Goal                Band                  Weight          Source
    _________    __________    _______    ____________________________    ______    _________________

     {'Gt'}        {'<'}       {[-10]}    {[500000000 1.5000e+09]}        {[1]}     {'User-specified
```

Input Arguments

`mnobj` — Matching network

`matchingnetwork` object

Matching network, specified as a `matchingnetwork` object.

Data Types: `char` | `string`

`frequencylist` — Frequency values at which to plot performance of matching network

vector

Frequency values at which to plot the performance of the matching network, specified as a vector with each element in Hz.

Data Types: `double`

`circuitindices` — Index of matching network
vector | scalar

Index of the matching network circuit, specified as a vector or scalar.

Data Types: `double`

Output Arguments

`hline` — Line
cell array

Line handle of circuit plotted, returned as a cell array with one cell for each circuit plotted.

Version History

Introduced in R2019a

See Also

`matchingnetwork` | `smithplot` | `sparameters`

generateSPICE

Generate SPICE file from `rationalfit` of S-parameters

Syntax

```
generateSPICE(fit,filename)
generateSPICE(fit,filename,zref)
```

Description

`generateSPICE(fit,filename)` generates a SPICE file from a rationalfit of S-parameters. Simulation program with integrated circuit emphasis (SPICE) is a general-purpose, open-source analog electronic circuit simulator.

`generateSPICE(fit,filename,zref)` generates a SPICE file using the reference impedance specified by `zref`.

Examples

Generate SPICE File of 2-by-2 S-parameters

Read a file named `passive.s2p` and fit the 2-by-2 S-parameters. Generate a SPICE file of these S-parameters.

```
S = sparameters('passive.s2p');
fit = rationalfit(S);
generateSPICE(fit,'passive.ckt')
```

The circuit is saved in your current folder.

Input Arguments

`fit` — Rationalfit of S-parameters

N-by-*N* array

Rationalfit of S-parameters, specified as an *N*-by-*N* array of `rfmodel.rational` objects as returned by the `rationalfit` function or a `rational` object with S-parameters as input.

Data Types: `double`

`filename` — Name of file for SPICE subcircuit

string | character vector

Name of file in which to store the SPICE subcircuit, specified as a string or a character vector. The SPICE subcircuit is constructed out of passive resistor (R) and capacitor (C) SPICE elements and controlled-source SPICE elements voltage-controlled voltage source (E), current-controlled current source (F), voltage-controlled current source (G), and current-controlled voltage source (H).

Data Types: `char` | `string`

`zref` — Reference impedance
`50` (default) | real scalar

Reference impedance, specified as a real scalar in ohms.

Data Types: `double`

Version History

Introduced in R2019b

See Also

`ispassive` | `makepassive` | `passivity` | `rationalfit`

setrfplot

Set axis type for `rfplot` in RF Toolbox

Syntax

```
setrfplot(axistype)
setrfplot(axistype,persist)
```

Description

`setrfplot(axistype)` applies or removes the use of engineering units on the X-axis of `rfplot`. By default, engineering units are always applied in the X-axis and persist across all MATLAB sessions.

`setrfplot(axistype,persist)` controls the persistence behavior of the units on the X-axis plot across MATLAB sessions.

Examples

Design Butterworth Filter and Determine Filter Order

This example shows how to design a low-pass Butterworth filter with passband frequency of 3 kHz, stopband frequency 7 kHz, passband attenuation of 2 dB, and stopband attenuation 60 dB. Display the filter order of such a designed filter and determine the passband frequency at 3.0103 dB. See [2] in `rffilter` object page.

Filter Parameters

```
Fp = 3e3;          % Passband frequency, Hz
Ap = 2;            % Passband attenuation, dB
Fs = 7e3;          % Stopband frequency, Hz
As = 60;           % Stopband attenuation, dB
```

Design Filter

```
r = rffilter("FilterType","Butterworth","ResponseType","Lowpass","Implementation","Transfer funct
    "PassbandAttenuation",Ap,"StopbandFrequency",Fs,"StopbandAttenuation",As);
```

Filter Order of Designed Filter

```
N = r.DesignData.FilterOrder;
sprintf('Calculated filter order is %d',N)
```

```
ans =
'Calculated filter order is 9'
```

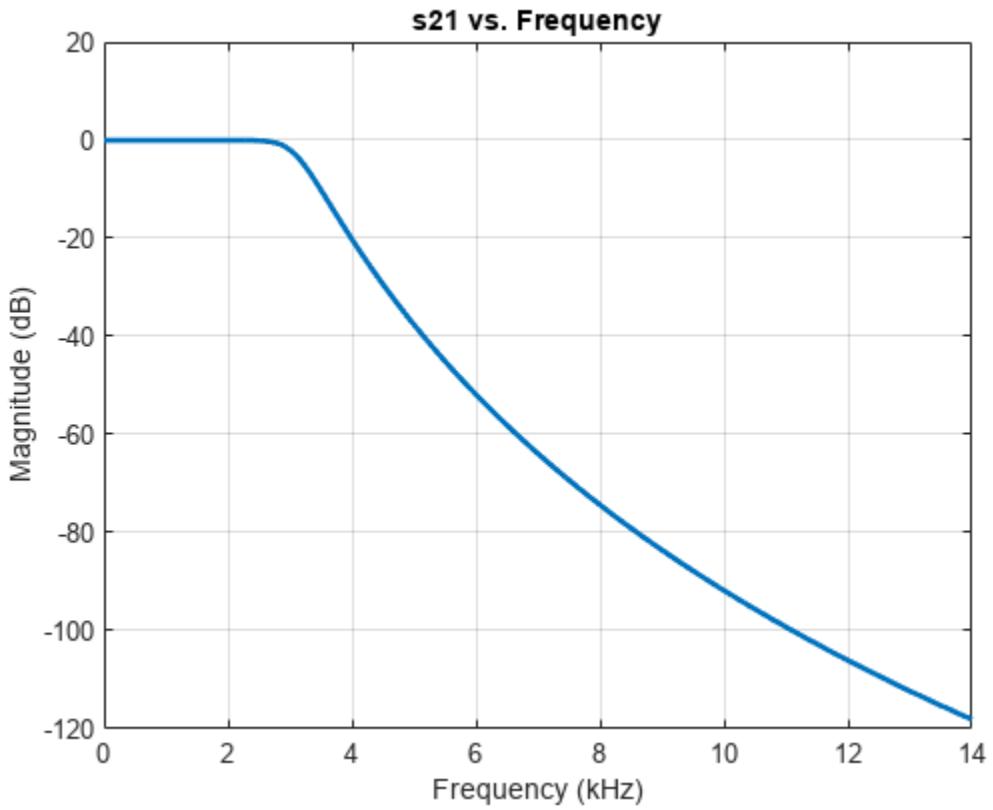
Frequency at 3.0103 dB

```
F_3dB = r.DesignData.PassbandFrequency/1e3;
sprintf('Frequency at 3.0103 dB is %d kHz',F_3dB)
```

```
ans =
'Frequency at 3.0103 dB is 3.090733e+00 kHz'
```

Visualize Magnitude Response

```
frequencies = linspace(0,2*Fs,1001);
rfplot(r, frequencies)
```



Note: To use `rfplot` and `plot` on the same figure use `setrfplot`. Type `'help setrfplot'` in command window for information.

Reference

1 Larry D. Paarmann, Design and Analysis of Analog Filters: A Signal Processing Perspective, Kluwer Academic Publishers

Input Arguments

`axistype` — Units on `rfplot` X-axis
`'engunits'` (default) | `'noengunits'`

Units on `rfplot` X-axis, specified as `'engunits'` or `'noengunits'`.

Data Types: `char` | `string`

`persist` — Units on `rfplot` X-axis across MATLAB sessions
`true` (default) | `false`

Units on `rfplot` X-axis across MATLAB sessions, specified as `true` or `false`.

Data Types: `logical`

Version History

Introduced in R2019b

See Also

`rfplot`

cascadesparams

Combine S-parameters to form cascade network

Syntax

```
s_params= cascadesparams(s1_params,s2_params,...,sk_params)
s_params= cascadesparams( ___ ,Kconn)

hs= cascadesparams(hs1,hs2,...,hsk)
```

Description

`s_params= cascadesparams(s1_params,s2_params,...,sk_params)` cascades the scattering parameters (S-parameters) of *K* input networks described by the S-parameters. Each input network must be a 2*N*-port network described by a 2*N*-by-2*N*-by-*M* array of S-parameters for *M* frequency points. All networks must have the same reference impedance.

Note The `cascadesparams` function uses ABCD-parameters. Alternatively, one can use S-parameters and ABCD-parameters (or T-parameters) to cascade S-parameters together by hand (assuming identical frequencies)

`s_params= cascadesparams( ___ ,Kconn)` creates the cascaded networks based on the number of cascaded connections between the networks specified by `Kconn`. Use this option with the input arguments in the previous syntax.

`hs= cascadesparams(hs1,hs2,...,hsk)` cascades *K* S-parameter objects to create a cascade network. The function checks that the impedance and frequencies of each object is equal and that the parameters of each object contain 2*N*-by-2*N*-by-*M* array of S-parameters for *M* frequency points.

Examples

Two-Port Cascaded Network

Assemble a 2-port cascaded network from two sets of 2-port S-parameters operating at 2 GHz and at 2.1 GHz.

Create two sets of 2-port S-parameters.

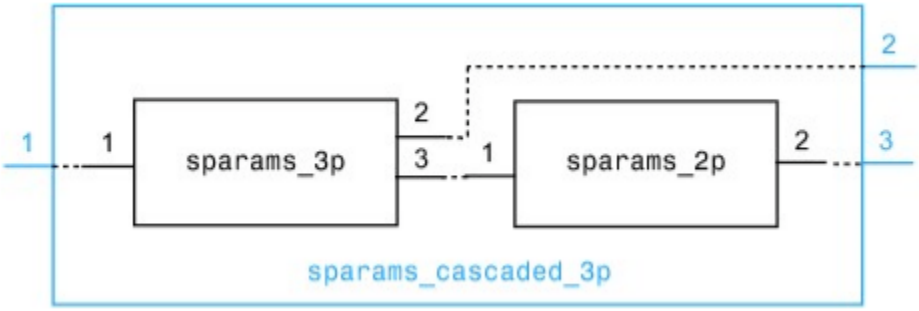
```
ckt1 = read(rfckt.amplifier,'default.s2p');
ckt2 = read(rfckt.passive,'passive.s2p');
freq = [2e9 2.1e9];
analyze(ckt1,freq);
analyze(ckt2,freq);
sparams_2p_1 = ckt1.AnalyzedResult.S_Parameters;
sparams_2p_2 = ckt2.AnalyzedResult.S_Parameters;
```

Cascade the S-parameters.

```
sparams_cascaded_2p = cascadesparams(sparams_2p_1,sparams_2p_2)

sparams_cascaded_2p =
sparams_cascaded_2p(:,:,1) =

  -0.4332 + 0.5779i   0.0081 - 0.0120i
   2.6434 + 1.2880i   0.5204 - 0.5918i


sparams_cascaded_2p(:,:,2) =

  -0.1271 + 0.3464i  -0.0004 - 0.0211i
   3.8700 - 0.6547i   0.4458 - 0.6250i
```

Three-Port Cascaded Network

Assemble a 3-port cascaded network from a set of 3-port S-parameters and a set of 2-port S-parameters.



Create one set of 3-port S-parameters and one set of 2-port S-parameters.

```
ckt1 = read(rfckt.passive,'default.s3p');
ckt2 = read(rfckt.amplifier,'default.s2p');
freq = [2e9 2.1e9];
a1 = analyze(ckt1,freq);
a2 = analyze(ckt2,freq);
sparams_3p = sparameters(a1);
sparams_2p = sparameters(a2);
```

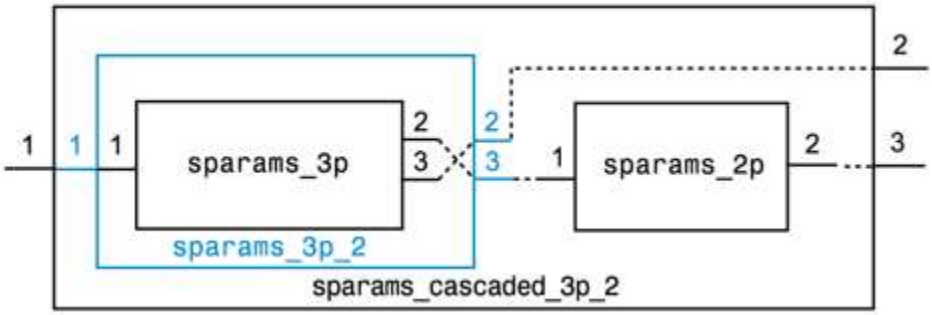
Cascade the two sets by connecting one port between them.

```
Kconn = 1;
sparams_cascaded_3p = cascadesparams(sparams_3p,sparams_2p,Kconn)

sparams_cascaded_3p =
  sparameters: S-parameters object

       NumPorts: 3
    Frequencies: [2x1 double]
     Parameters: [3x3x2 double]
      Impedance: 50.0000 + 0.0000i
```

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

Three-Port Cascaded Network from S-Parameters

Assemble a 3-port cascaded network from a set of 3-port S-parameters and a set of 2-port S-parameters, connecting the second port of the 3-port network to the first port of the 2-port.




```
ckt1 = read(rfckt.passive,'default.s3p');
ckt2 = read(rfckt.amplifier,'default.s2p');
freq = [2e9 2.1e9];
a1 = analyze(ckt1,freq);
a2 = analyze(ckt2,freq);
sparams_3p = sparameters(a1);
sparams_2p = sparameters(a2);
```

Reorder the second and third ports of the 3-port network

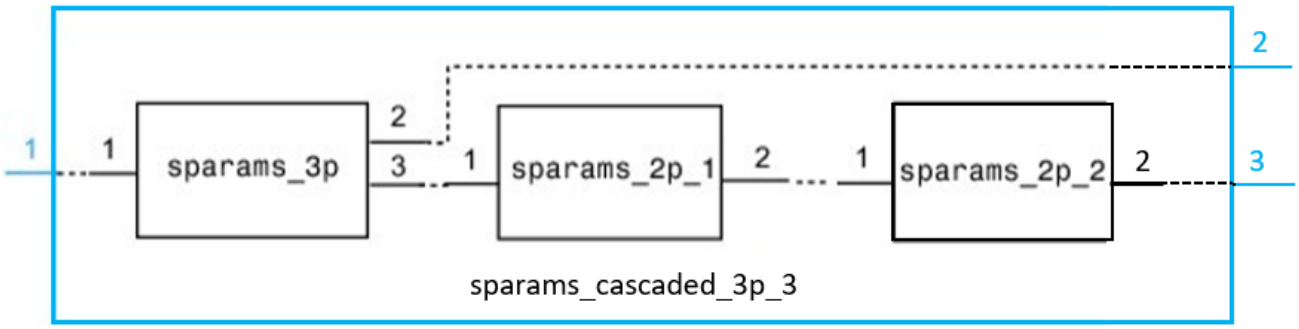
```
sparams_3p_2 = snp2smp(sparams_3p,[1 3 2]);
```

Cascade the two sets by connecting one port between them

```
Kconn = 1;
sparams_cascaded_3p_2 = cascadesparams(sparams_3p_2,...
     sparams_2p,Kconn)

sparams_cascaded_3p_2 =
  sparameters: S-parameters object

       NumPorts: 3
    Frequencies: [2x1 double]
     Parameters: [3x3x2 double]
      Impedance: 50.0000 + 0.0000i

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

Three-Port Cascade Network from Multiple S-Parameters

Assemble a 3-port cascaded network from a set of 3-port S-parameters and two sets of 2-port S-parameters.




```
ckt1 = read(rfckt.passive,'default.s3p');
ckt2 = read(rfckt.amplifier,'default.s2p');
ckt3 = read(rfckt.passive,'passive.s2p');
freq = [2e9 2.1e9];
a1 = analyze(ckt1,freq);
a2 = analyze(ckt2,freq);
a3 = analyze(ckt3,freq);
sparams_3p = sparameters(a1);
sparams_2p_1 = sparameters(a2);
sparams_2p_2 = sparameters(a3);
```

Connect one port between each set of adjacent networks.

```
Kconn = [1 1];
sparams_cascaded_3p_3 = cascadesparams(sparams_3p,...
     sparams_2p_1,sparams_2p_2,Kconn)
```

```
sparams_cascaded_3p_3 =
  sparameters: S-parameters object

       NumPorts: 3
    Frequencies: [2x1 double]
     Parameters: [3x3x2 double]
      Impedance: 50.0000 + 0.0000i

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

Complex Three-Port Cascaded Network

Assemble a 3-port cascaded network from a set of 3-port S-parameters and two sets of 2-port S-parameters, connecting the 3-port network to both 2-port networks.

```
ckt1 = read(rfckt.passive,'default.s3p');
ckt2 = read(rfckt.amplifier,'default.s2p');
```

```
ckt3 = read(rfckt.passive,'passive.s2p');
freq = [2e9 2.1e9];
analyze(ckt1,freq);
analyze(ckt2,freq);
analyze(ckt3,freq);
sparams_3p = ckt1.AnalyzedResult.S_Parameters;
sparams_2p_1 = ckt2.AnalyzedResult.S_Parameters;
sparams_2p_2 = ckt3.AnalyzedResult.S_Parameters;
```

Cascade sparams_3p and sparams_2p_1 by connecting one port between them.

```
Kconn = 1;
sparams_cascaded_3p = cascadesparams(sparams_3p, ...
  sparams_2p_1, Kconn);
```

Reorder the second and third ports of the 3-port network.

```
sparams_cascaded_3p_3 = snp2smp(sparams_cascaded_3p, ...
  50, [1 3 2]);
```

Cascade sparams_3p and sparams_2p_2 by connecting one port between them.

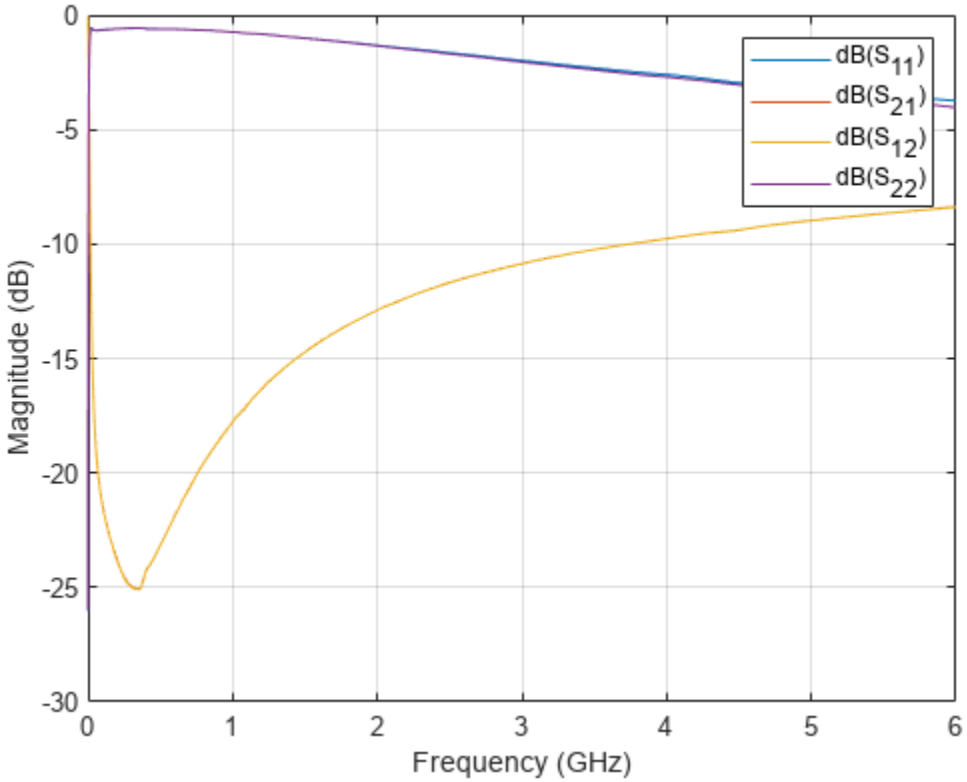
```
sparams_cascaded_3p_4 = cascadesparams(sparams_cascaded_3p_3, ...
  sparams_2p_2, Kconn)
```

```
sparams_cascaded_3p_4 =
sparams_cascaded_3p_4(:,:,1) =

   0.1724 - 0.9106i   0.0240 + 0.0134i   0.0104 + 0.0971i
  -3.7923 + 6.1234i  -0.7168 - 0.6498i  -0.5855 + 1.6475i
   0.1214 + 0.0866i   0.0069 + 0.0090i   0.6289 - 0.6145i


sparams_cascaded_3p_4(:,:,2) =

  -0.3014 - 0.6620i   0.0072 - 0.0255i  -0.0162 + 0.1620i
   6.3709 + 2.2809i  -0.5349 + 0.3637i   1.4106 + 0.2587i
   0.0254 + 0.1011i   0.0087 - 0.0075i   0.5477 - 0.6253i
```

Using T-Parameters

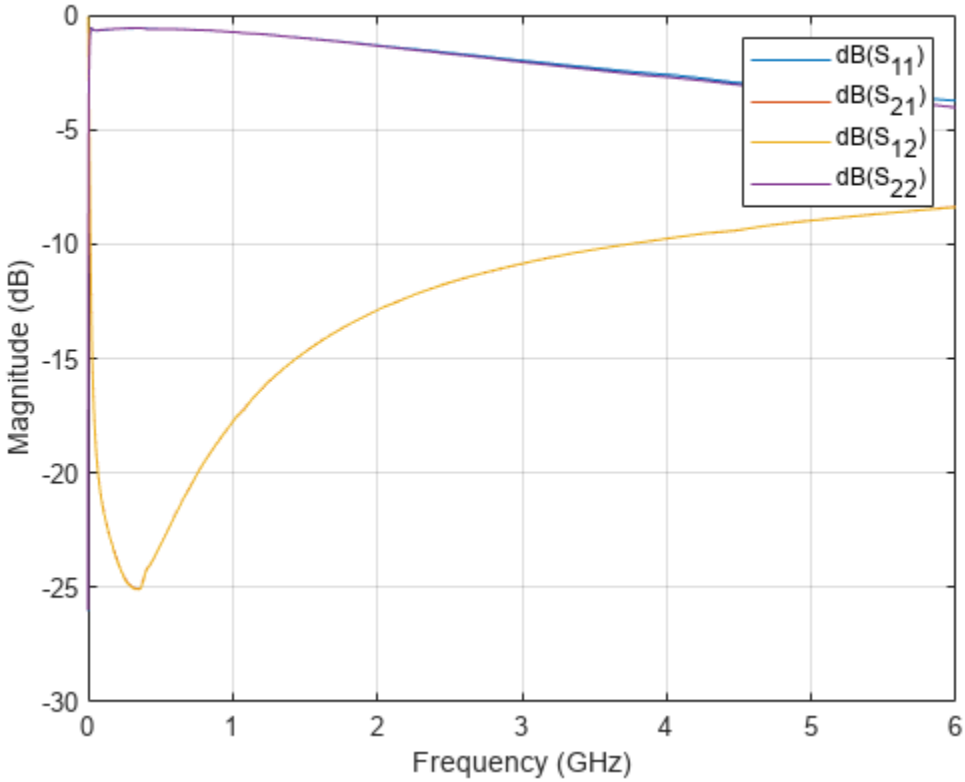
Compute cascaded S-parameters using T-parameters.

```
T = tparameters('passive.s2p');
freq = T.Frequencies;
for i = 1:length(freq)
    Tcascade(:,:,i) = T.Parameters(:,:,i)*T.Parameters(:,:,i);
end
Tcasc = tparameters(Tcascade,freq);
Scasc_T = sparameters(Tcasc);
rfplot(Scasc_T)
```



Validate results using `cascadesparams`.

```
S = sparameters(T);
Scasc = cascadesparams(S,S);
rfplot(Scasc)
```



Input Arguments

`s1_params` — S-parameters data
2*N*-by-2*N*-by-*M* array

S-parameters data, specified as a complex 2*N*-by-2*N*-by-*M* array.

`hs1` — S-parameter object
`sparameters` function object

S-parameter object, specified as a `sparameters` function object.

`Kconn` — Number of cascade connections
50 (default) | positive scalar or vector

Number of cascade connections, specified as a positive scalar or vector.

- If `Kconn` is a scalar, `cascadesparams` makes the same number of connections between each pair of consecutive networks.
- If `Kconn` is a vector, the `i`th element of `Kconn` specifies the number of connections between the `i`th and the `i+1`th networks.

More About

Port Ordering

The function assumes that the port ordering of the network and it is shown in the image.

Based on this ordering, the function connects ports `N + 1` through `2N` of the first network to ports `1` through `N` of the second network. Therefore, when you use this syntax:

- Each network has an even number of ports
- Every network in the cascade has the same number of ports.

To use this function for S-parameters with different port arrangements, use the `snp2smp` function to reorder the port indices before cascading the networks.

Kconn

`cascadesparams` always connects the last `Kconn(i)` ports of the `i`th network and the first `Kconn(i)` ports of the `i+1`th network. The ports of the entire cascaded network represent the unconnected ports of each individual network, taken in order from the first network to the `n`th network.

Also, when you specify `Kconn`:

- Each network can have either an even or odd number of ports.
- Every network in the cascade can have a different number of ports.

Version History

Introduced in R2008a

See Also

`deembedsparams` | `s2t` | `t2s` | `snp2smp` | `rfckt.cascade`

s2h

Convert S-parameters to hybrid h-parameters

Syntax

```
h_params = s2h(s_params,z0)
```

Description

`h_params = s2h(s_params,z0)` converts the scattering parameters to the hybrid parameters.

Examples

Convert S-Parameters to H-Parameters

Define a matrix of S-parameters.

```
s_11 = 0.61*exp(j*165/180*pi);
s_21 = 3.72*exp(j*59/180*pi);
s_12 = 0.05*exp(j*42/180*pi);
s_22 = 0.45*exp(j*(-48/180)*pi);
s_params = [s_11 s_12; s_21 s_22];
z0 = 50;
```

Convert S-parameters to h-parameters.

```
h_params = s2h(s_params,z0)
```

```
h_params = 2×2 complex

  15.3381 + 1.4019i   0.0260 + 0.0411i
  -0.9585 - 3.4902i   0.0106 + 0.0054i
```

Input Arguments

`s_params` — 2-port S-Parameters

2-by-2-by-*M* array of complex numbers

2-port S-parameters, specified as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port S-Parameters.

`z0` — Reference impedance

`50` (default) | positive real scalar

Reference impedance of 2-port S-parameters, specified as a positive real scalar in ohms.

Output Arguments

`h_params` — 2-port hybrid h-parameters
2-by-2-by-*M* array of complex numbers

2-port hybrid h-parameters, returned as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port hybrid h-parameters.

Version History

Introduced before R2006a

References

[1] Frickey, D. A. “Conversions between S, Z, Y, H, ABCD, and T Parameters Which Are Valid for Complex Source and Load Impedances.” *IEEE Transactions on Microwave Theory and Techniques*, vol. 42, no. 2, Feb. 1994, pp. 205–11. *DOI.org (Crossref)*, doi:10.1109/22.275248.

See Also

`h2s` | `s2abcd` | `s2s` | `s2sdd` | `s2smm` | `s2scd` | `s2sdc` | `s2rlgc` | `s2t` | `s2y` | `s2z` | `s2tf` | `smm2s` | `s2scc`

s2s

Convert S-parameters to S-parameters with different impedance

Syntax

```
s_params_new = s2s(s_params,z0)
s_params_new = s2s(s_params,z0,z0_new)
```

Description

`s_params_new = s2s(s_params,z0)` converts the scattering parameters with reference impedance `z0` to the scattering parameters with a default reference impedance of 50 ohms.

`s_params_new = s2s(s_params,z0,z0_new)` converts the scattering parameters with reference impedance `z0` into the scattering parameters with reference impedance `z0_new`.

Examples

S-Parameters to S-Parameters with Different Impedance

Define a matrix of S-parameters.

```
s_11 = 0.61*exp(1i*165/180*pi);
s_21 = 3.72*exp(1i*59/180*pi);
s_12 = 0.05*exp(1i*42/180*pi);
s_22 = 0.45*exp(1i*(-48/180)*pi);
s_params = [s_11 s_12; s_21 s_22];
z0 = 50;
z0_new = 40;
```

Convert to S-parameters with different impedance.

```
s_params_new = s2s(s_params,z0,z0_new)
```

```
s_params_new = 2×2 complex

  -0.5039 + 0.1563i   0.0373 + 0.0349i
   1.8929 + 3.2940i   0.4150 - 0.3286i
```

Input Arguments

`s_params` — *N*-port S-Parameters

N-by-*N*-by-*M* array of complex numbers

N-port S-Parameters, specified as an *N*-by-*N*-by-*M* array of complex numbers, where *M* represents the number of frequency points of *N*-port S-parameters.

`z0` — Reference impedance
`50` (default) | positive real scalar | vector

Reference impedance of *N*-port S-parameters, specified in scalar as ohms.

Note `z0` must be a positive real scalar or vector. If `z0` is a vector, then the vector must be equal to the number of network parameter data points or frequency vector.

`z0_new` — Reference impedance
`50` (default) | real scalar

Reference impedance of S-parameters, specified in scalar as ohms.

Output Arguments

`s_params_new` — *N*-port hybrid S-Parameters
N-by-*N*-by-*M* array of complex numbers

N-port hybrid S-parameters, returned as an *N*-by-*N*-by-*M* array of complex numbers, where *M* represents the number of frequency points of *N*-port hybrid S-parameters.

Alternatives

The `newref` function changes the reference impedance of S-parameters objects.

Version History

Introduced before R2006a

References

[1] Reveyrand, T. “Multiport Conversions between S, Z, Y, h, ABCD, and T Parameters.” *2018 International Workshop on Integrated Nonlinear Microwave and Millimetre-Wave Circuits (INMMIC)*, IEEE, 2018, pp. 1–3. *DOI.org (Crossref)*, doi:10.1109/INMMIC.2018.8430023.

See Also

`s2abcd` | `s2sdd` | `s2smm` | `s2scd` | `s2sdc` | `s2rlgc` | `s2t` | `s2y` | `s2z` | `s2tf` | `smm2s` | `s2scc` | `s2h`

s2z

Convert S-parameters to Z-parameters

Syntax

```
z_params = s2z(s_params,z0)
```

Description

`z_params = s2z(s_params,z0)` converts the scattering parameters to the impedance parameters.

Examples

Convert S-Parameters to Z-Parameters

Define a matrix of S-parameters.

```
s_11 = 0.61*exp(j*165/180*pi);
s_21 = 3.72*exp(j*59/180*pi);
s_12 = 0.05*exp(j*42/180*pi);
s_22 = 0.45*exp(j*(-48/180)*pi);
s_params = [s_11 s_12; s_21 s_22];
z0 = 50;
```

Convert S-parameters to Z-parameters.

```
z_params = s2z(s_params,z0)
```

```
z_params = 2×2 complex
10^2 ×

   0.1141 + 0.1567i   0.0352 + 0.0209i
   2.0461 + 2.2524i   0.7498 - 0.3803i
```

Input Arguments

`s_params` — *N*-port- S-Parameters

N-by-*N M* array of complex numbers

N-port- S-Parameters, specified as an *N*-by-*N M* array of complex numbers, where *M* represents the number of frequency points of *N*-port S-parameters.

`z0` — Reference impedance

`50` (default) | positive real scalar | vector

Reference impedance of an *N*-port S-parameters, specified as a positive real scalar in ohms.

Note `z0` can be a positive real scalar or vector. If `z0` is a vector, then the vector must be equal to the number of network parameter data points.

Output Arguments

`z_params` — *N*-port Z-parameters

N-by-*N*-by-*M* array of complex numbers

N-port Z-parameters, returned as an *N*-by-*N*-by-*M* array of complex numbers, where *M* represents the number of frequency points of *N*-port Z-parameters.

Version History

Introduced before R2006a

See Also

`s2abcd` | `s2h` | `s2s` | `s2sdd` | `s2smm` | `s2scd` | `s2sdc` | `s2scc` | `s2rlgc` | `s2t` | `s2y` | `s2tf` | `smm2s` | `snp2smp`

s2sdd

Convert 4-port, single-ended S-parameters to 2-port, differential-mode S-parameters (S_{dd})

Syntax

```
sdd_params = s2sdd(s_params)
sdd_params = s2sdd(s_params,option)
```

Description

`sdd_params = s2sdd(s_params)` converts the 2*N*-port, single-ended S-parameters to *N*-port, differential-mode S-parameters.

`sdd_params = s2sdd(s_params,option)` converts S-parameters based on the optional `option` argument, which indicates the port-ordering convention of the S-parameters.

Examples

Analyze Differential Mode S-Parameters

Create a `sparameters` object from a data file.

```
sparams = sparameters('default.s4p');
```

Create a data object to store the differential S-parameters.

```
diffSparams = rfdata.network;
diffSparams.Freq = sparams.Frequencies;
diffSparams.Data = s2sdd(sparams.Parameters);
diffSparams.Z0 = 2*sparams.Impedance;
```

Create a new circuit object with the data from the data object.

```
diffCkt = rfckt.passive;
diffCkt.NetworkData = diffSparams;
```

Analyze the new circuit object.

```
frequencyRange = diffCkt.NetworkData.Freq;
ZL = 50;
ZS = 50;
Z0 = diffSparams.Z0;
analyze(diffCkt,frequencyRange,ZL,ZS,Z0);
diffData = diffCkt.AnalyzedResult;
```

Write the differential S-parameters into a Touchstone data file.

```
write(diffCkt,'diffsparams.s2p')
```

```
ans = logical
   1
```

Single-ended S-Parameters to Differential-Mode S-Parameters

Convert network data to differential-mode S-Parameters using the default port ordering.

```
S = sparameters('default.s4p');
s4p = S.Parameters;
s_dd = s2sdd(s4p);
```

To display differential-mode S-Parameters at the first frequency, type the following command:

```
s_dd2 = s_dd(:,:,1)
```

```
s_dd2 = 2×2 complex

  -0.0124 - 0.0433i  -0.5434 - 0.6872i
  -0.5428 - 0.6900i  -0.0192 - 0.0504i
```

Input Arguments

`s_params` — 2*N*-port single-ended S-Parameters

complex 2*N*-by-2*N*-by-*M* array

2*N*-port single-ended S-Parameters, specified as a complex 2*N*-by-2*N*-by-*M* array, where *M* represents the number of frequency points of 2*N*-port single-ended S-Parameters.

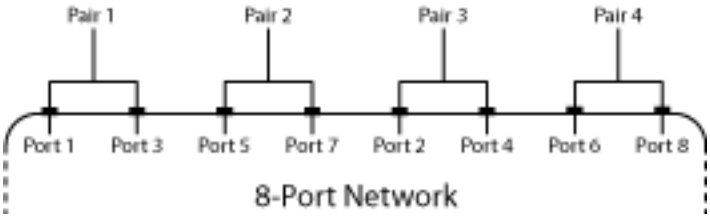
`option` — Port order

1 (default) | 2 | 3 | scalar

Port order, a scalar, specified as `1`, `2`, or `3`. Port order determines how the function orders the ports:

- `1` — `s2sdd` pairs the odd-numbered ports together first, followed by the even-numbered ports. For example, in a single-ended, 8-port network:
  - Ports 1 and 3 become differential-mode pair 1.
  - Ports 5 and 7 become differential-mode pair 2.
  - Ports 2 and 4 become differential-mode pair 3.
  - Ports 6 and 8 become differential-mode pair 4.

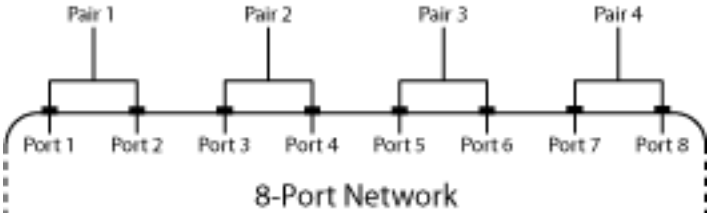
  The following figure illustrates this convention for an 8-port device.



- `2` — `s2sdd` pairs the input and output ports in ascending order. For example, in a single-ended, 8-port network:

- Ports 1 and 2 become differential-mode pair 1.
- Ports 3 and 4 become differential-mode pair 2.
- Ports 5 and 6 become differential-mode pair 3.
- Ports 7 and 8 become differential-mode pair 4.

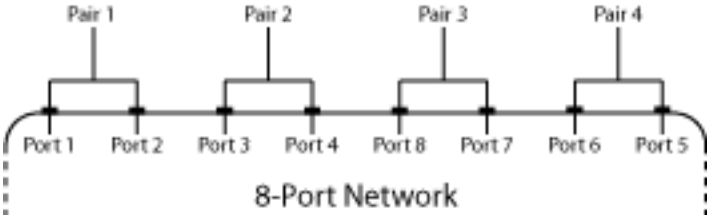
The following figure illustrates this convention for an 8-port device.



- `3` — `s2sdd` pairs the input ports in ascending order and the output ports in descending order. For example, in a single-ended, 8-port network:

  - Ports 1 and 2 become differential-mode pair 1.
  - Ports 3 and 4 become differential-mode pair 2.
  - Ports 8 and 7 become differential-mode pair 3.
  - Ports 6 and 5 become differential-mode pair 4.

  The following figure illustrates this convention for an 8-port device.



Output Arguments

`sdd_params` — *N*-port differential-mode S-Parameters
complex *N*-by-*N*-by-*M* array

N-port differential-mode S-Parameters, returned as a complex *N*-by-*N*-by-*M* array, where *M* represents the number of frequency points of an *N*-port, differential-mode S-parameters (S_{dd}).

Version History

Introduced in R2006a

References

[1] Fan, W., A. C. W. Lu, L. L. Wai, and B. K. Lok. "Mixed-Mode S-Parameter Characterization of Differential Structures." *Electronic Packaging Technology Conference*. pp. 533–537, 2003.

See Also

s2abcd | s2h | s2s | s2smm | s2scd | s2sdc | s2scc | s2rlgc | s2t | s2y | s2z | s2tf | smm2s | snp2smp

s2smm

Convert single-ended S-parameters to mixed-mode S-parameters

Syntax

```
[s_dd,s_dc,s_cd,s_cc] = s2smm(s_params_even,rfflag)
s_mm = s2smm(s_params_odd)
```

Description

`[s_dd,s_dc,s_cd,s_cc] = s2smm(s_params_even,rfflag)` converts single-ended S-parameters to mixed-mode form.

`s_mm = s2smm(s_params_odd)` converts single-ended odd S-parameters matrix to mixed-mode matrix. To create a mixed-mode matrix from `s_params_odd`, the single-ended input ports are paired sequentially (port1 with port 2, port 3 with port 4, etc.), and the last port is left single ended.

Examples

4-Port S-Parameters to 2-Port Mixed-Mode S-Parameters

Convert 4-port S-parameters to 2-port, mixed-mode S-Parameters.

```
S = sparameters('default.s4p');
s4p = S.Parameters;
[s_dd,s_dc,s_cd,s_cc] = s2smm(s4p);
```

Display the 2-port mixed-mode S-Parameters at the first frequency.

```
s_dd1 = s_dd(:,:,1)
```

```
s_dd1 = 2×2 complex

  -0.0124 - 0.0433i  -0.5434 - 0.6872i
  -0.5428 - 0.6900i  -0.0192 - 0.0504i
```

```
s_dc1 = s_dc(:,:,1)
```

```
s_dc1 = 2×2 complex

   0.0024 - 0.0035i  -0.0005 + 0.0019i
   0.0007 - 0.0012i   0.0023 - 0.0027i
```

```
s_cc1 = s_cc(:,:,1)
```

```
s_cc1 = 2×2 complex

   0.1799 - 0.1839i  -0.5300 - 0.6771i
  -0.5314 - 0.6800i   0.1756 - 0.1910i
```

```
s_cd = s_cd(:,:,1)

s_cd = 2×2 complex

   0.0015 - 0.0029i  -0.0005 + 0.0014i
   0.0003 - 0.0009i   0.0019 - 0.0027i
```

Input Arguments

`s_params_even` — S-parameters

2 *N*-by-2 *N*-by-*K* array

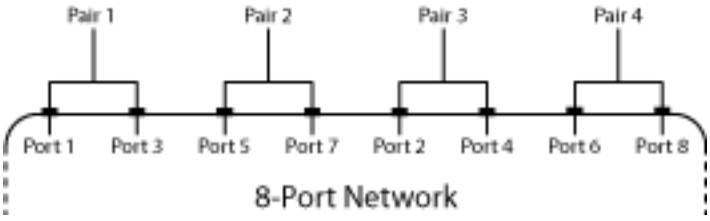
S-parameters, specified as a complex 2 *N*-by-2 *N*-by-K array, where *K* representing number of frequency points of 2 *N*-port S-Parameters. These parameters describe a device with an even number of ports.

`rfflag` — Port order

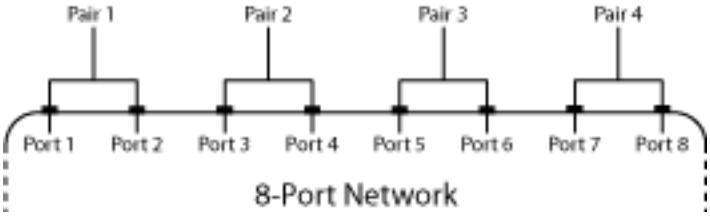
1 (default) | 2 | 3 | scalar

Port order, a scalar, specified as `1`, `2`, or `3`. Port order determines how the function orders the ports:

- `rfflag = 1` — `s2smm` Odd-numbered ports are followed by even-numbered ports: 1,3,5, ...,2*N*-4,2*N*-2,2*N*.

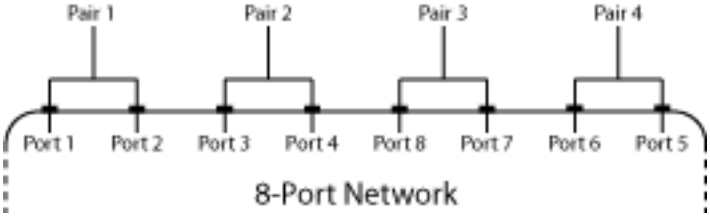


  - Ports 1 and 3 become mixed-mode pair 1.
  - Ports 5 and 7 become mixed-mode pair 2.
  - Ports 2 and 4 become mixed-mode pair 3.
  - Ports 6 and 8 become mixed-mode pair 4.
- `rfflag = 2` — Ports are paired in ascending or descending order: (1,2),.....,(2*N*-1,2*N*)



  - Ports 1 and 2 become mixed-mode pair 1.
  - Ports 3 and 4 become mixed-mode pair 2.
  - Ports 5 and 6 become mixed-mode pair 3.
  - Ports 7 and 8 become mixed-mode pair 4.

- `rfflag = 3` — Half of the ports are in ascending order and half of the ports are in descending order: 1,2,.....,*N*,2*N*,2*N*-1,....,*N*+1.

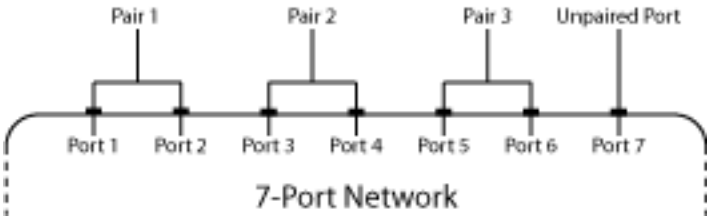


  - Ports 1 and 2 become mixed-mode pair 1.
  - Ports 3 and 4 become mixed-mode pair 2.
  - Ports 8 and 7 become mixed-mode pair 3.
  - Ports 6 and 5 become mixed-mode pair 4.

`s_params_odd` — *K* (2 *N*+1) port single-ended S-Parameters matrix
complex (2 *N*+1)-by-(2 *N*+1)-by-K array of

K (2 *N*+1) port single-ended S-Parameters matrix, specified as a complex (2 *N*+1)-by-(2 *N*+1)-by-K array. These parameters describe a device with an odd number of ports.

The port-ordering argument `option` is not available for (2*N* + 1)-by-(2*N* + 1)-by-*K* input arrays. In this case, the ports are always paired in ascending order, and the last port remains single-ended. For example, in a 7-port network:



- Ports 1 and 2 become mixed-mode pair 1.
- Ports 3 and 4 become mixed-mode pair 2.
- Ports 5 and 6 become mixed-mode pair 3.
- Ports 7 remains single ended.

Output Arguments

`s_dd` — Mixed-mode S-parameters
complex *N*-by-*N*-by-*K* array

Mixed-mode S-parameters, returned as complex *N*-by-*N*-by-*K* array, containing *K* matrices of differential-mode, 2*N*-port S-parameters (S_{dd}).

`s_dc` — Mixed-mode S-parameters
complex *N*-by-*N*-by-*K* array

Mixed-mode S-parameters, returned as complex *N*-by-*N*-by-*K* array, containing *K* matrices of differential-mode, 2*N*-port S-parameters (S_{dc}).

`s_cd` — Mixed-mode S-parameters
complex *N*-by-*N*-by-*K* array

Mixed-mode S-parameters, returned as complex *N*-by-*N*-by-*K* array, containing *K* matrices of differential-mode, 2*N*-port S-parameters (S_{cd}).

`s_cc` — Mixed-mode S-parameters
complex *N*-by-*N*-by-*K* array

Mixed-mode S-parameters, returned as complex *N*-by-*N*-by-*K* array, containing *K* matrices of differential-mode, 2*N*-port S-parameters (S_{cc}).

`s_mm` — Mixed-mode S-parameters
complex *N*-by-*N*-by-*K* array

Mixed-mode S-parameters, returned as complex *N*-by-*N*-by-*K* array, containing *K* matrices of differential-mode, 2*N*-port S-parameters (S_{mm}).

$$\begin{bmatrix} S_{dd,11} & \cdots & S_{dd,1N} & S_{dc,11} & \cdots & S_{dc,1N} & S_{ds,1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ S_{dd,N1} & \cdots & S_{dd,NN} & S_{dc,N1} & \cdots & S_{dc,NN} & S_{ds,N} \\ S_{cd,11} & \cdots & S_{cd,1N} & S_{cc,11} & \cdots & S_{cc,1N} & S_{cs,1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ S_{cd,N1} & \cdots & S_{cd,NN} & S_{cc,N1} & \cdots & S_{cc,NN} & S_{cs,N} \\ S_{sd,1} & \cdots & S_{sd,N} & S_{sc,1} & \cdots & S_{sc,N} & S_{ss} \end{bmatrix}$$

Version History

Introduced in R2009a

References

[1] Granberg, T.,*Handbook of Digital Techniques for High-Speed Design*. Upper Saddle River, NJ: Prentice Hall, 2004.

See Also

`s2abcd` | `s2h` | `s2s` | `s2sdd` | `s2scd` | `s2sdc` | `s2scc` | `s2rlgc` | `s2t` | `s2y` | `s2z` | `s2tf` | `smm2s` | `snp2smp`

s2rlgc

Convert S-parameters to RLGC transmission line parameters

Syntax

```
rlgc_params = s2rlgc(s_params,length,freq)
rlgc_params = s2rlgc( ___ ,z0)
```

Description

`rlgc_params = s2rlgc(s_params,length,freq)` transforms multi-port S-parameter data into RLGC transmission line parameters using a reference impedance of 50 Ω.

`rlgc_params = s2rlgc( ___ ,z0)` transforms multi-port S-parameter data into a frequency-domain representation of an RLGC transmission line using a characteristic impedance of `z0`. Use this option with the input arguments in the previous syntax.

Examples

Convert S-Parameters to RLGC Parameters

Define the S-parameters of a transmission line.

```
s_11 = 0.000249791883190134 - 9.42320545953709e-005i;
s_12 = 0.999250283783862 -  0.0002197701545247 34i;
s_21 = 0.999250283783863 -  0.0002197701545247 56i;
s_22 = 0.000249791883190079 - 9.42320545953931e-005i;
s_params = [s_11,s_12; s_21,s_22];
```

Specify the length, frequency of operation, and impedance of the transmission line.

```
length = 1e-3;
freq = 1e9;
z0 = 50;
```

Convert the S-parameters to RLGC parameters.

```
rlgc_params = s2rlgc(s_params,length,freq,z0)
```

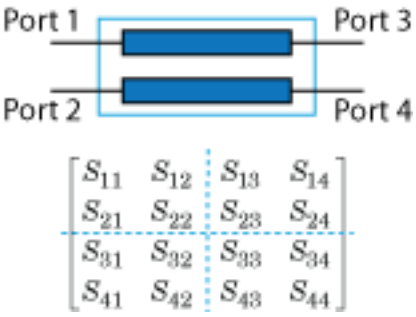
```
rlgc_params = struct with fields:
        R: 50.0000
        L: 1.0000e-09
        G: 0.0100
        C: 1.0000e-12
    alpha: 0.7265
     beta: 0.2594
       Zc: 63.7761 -14.1268i
```

Input Arguments

`s_params` — Scattering parameters of transmission line

2*N*-by-2*N*-by-*M* array

Specify a 2*N*-by-2*N*-by-*M* array of S-parameters to transform into RLGC transmission line parameters. The following figure describes the port ordering convention assumed by the function.



The function assumes that:

- Each 2*N*-by-2*N* matrix consists of *N* input terminals and *N* output terminals.
- The first *N* ports (1 through *N*) of the S-parameter matrix are input ports.
- The last *N* ports (*N* + 1 through 2*N*) are output ports.

To reorder ports before using this function, use the `snp2smp` function.

`length` — Length of transmission line

scalar

Specify the length of the transmission line in meters.

`freq` — Frequency

M-by-1 vector

Specify the vector of *M* frequencies over which the S-parameter array `s_params` is defined.

`z0` — Reference impedance

`50` (default) | positive real scalar

Reference impedance of *N*-port S-Parameters, specified as positive real scalar in ohms.

Output Arguments

`rlgc_params` — Transmission line parameters

N-by-*N*-by-*M* arrays

The output `rlgc_params` is structure whose fields are *N*-by-*N*-by-*M* arrays of transmission line parameters. Each of the *M* *N*-by-*N* matrices correspond to a frequency in the input vector `freq`.

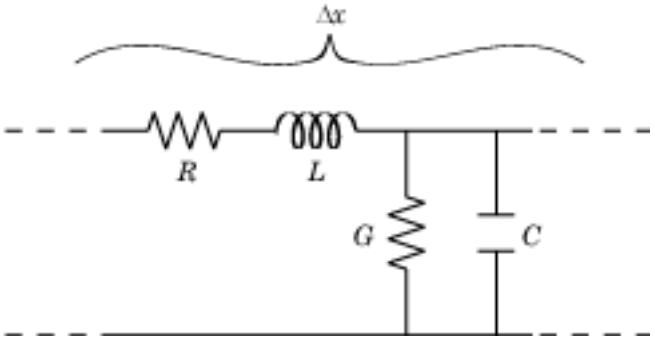
- `rlgc_params.R` is an array of distributed resistances, in units of Ω/m. The matrices are real symmetric, the diagonal terms are nonnegative, and the off-diagonal terms are nonnegative.
- `rlgc_params.L` is an array of distributed inductances, in units of H/m. The matrices are real symmetric, the diagonal terms are positive, and the off-diagonal terms are nonnegative.

- `rlgc_params.G` is an array of distributed conductances, in units of S/m. The matrices are real symmetric, the diagonal terms are nonnegative, and the off-diagonal terms are nonpositive.
- `rlgc_params.C` is an array of distributed capacitances, in units of F/m. The matrices are real symmetric, the diagonal terms are positive, and the off-diagonal terms are nonpositive.
- `rlgc_params.Zc` is an array of complex characteristic line impedances, in ohms.
- `rlgc_params.alpha` is an array of real attenuation coefficients, in units of Np/m.
- `rlgc_params.beta` is an array of real phase constants, in units of rad/m.

More About

RLCG Transmission Line Model

The following figure illustrates the RLGC transmission line model.



The representation consists of:

- The distributed resistance, *R*, of the conductors, represented by a series resistor.
- The distributed inductance, *L*, of the conductors, represented by a series inductor.
- The distributed conductance, *G*, between the two conductors, represented by a shunt resistor.
- The distributed capacitance, *C*, between the two conductors, represented by a shunt capacitor.

RLGC component units are all per unit length Δ*x*.

Tips

- If you have measured `s_params` for two different lengths of the transmission line, de-embed the S-parameters for the shorter length from the longer length before specifying the `s_params` input argument in the `s2rlgc` function.

Version History

Introduced in R2011b

References

[1] Degerstrom, M.J., Gilbert, B.K., and Daniel, E.S . "Accurate resistance, inductance, capacitance, and conductance (RLCG) from uniform transmission line measurements."*Electrical*

Performance of Electronic Packaging. IEEE-EPEP, 18th Conference, 27–29 October 2008, pp. 77–80.

[2] Sampath, M.K. "On addressing the practical issues in the extraction of RLGC parameters for lossy multi-conductor transmission lines using S-parameter models." *Electrical Performance of Electronic Packaging,*. IEEE-EPEP, 18th Conference, 27–29 October 2008, pp. 259–262.

[3] Eisenstadt, W. R., and Eo, Y. "S-parameter-based IC interconnect transmission line characterization," *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*. Vol. 15, No. 4, August 1992, pp. 483–490.

See Also

`s2abcd` | `s2s` | `s2sdd` | `s2smm` | `s2scd` | `s2sdc` | `s2rlgc` | `s2t` | `s2y` | `s2z` | `s2tf` | `smm2s` | `s2scc` | `rlgc2s` | `s2h`

s2t

Convert S-parameters to T-parameters

Syntax

```
t_params = s2t(s_params)
```

Description

`t_params = s2t(s_params)` converts the scattering parameters to the chain scattering parameters.

This function uses the following definition for T-parameters:

$$\begin{bmatrix} a_1 \\ b_1 \end{bmatrix} = \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix} \begin{bmatrix} b_2 \\ a_2 \end{bmatrix},$$

where:

- a_1 is the incident wave at the first port.
- b_1 is the reflected wave at the first port.
- a_2 is the incident wave at the second port.
- b_2 is the reflected wave at the second port.

Examples

Convert S-Parameters to T-Parameters

Define a matrix of S-parameters.

```
s11 = 0.61*exp(j*165/180*pi);
s21 = 3.72*exp(j*59/180*pi);
s12 = 0.05*exp(j*42/180*pi);
s22 = 0.45*exp(j*(-48/180)*pi);
s_params = [s11 s12; s21 s22];
```

Convert S-parameters to T-parameters.

```
t_params = s2t(s_params)
```

```
t_params = 2×2 complex

   0.1385 - 0.2304i   0.0354 + 0.1157i
  -0.0452 + 0.1576i  -0.0019 - 0.0291i
```

Input Arguments

`s_params` — 2-port- S-Parameters

2-by-2-by-*M* array of complex numbers

2-port S-parameters, specified as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port S-Parameters.

Output Arguments

`t_params` — 2-port T-Parameters

2-by-2-by-*M* array of complex numbers

2-port T-Parameters returned as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port T-parameters.

Version History

Introduced before R2006a

References

[1] Gonzalez, Guillermo. *Microwave Transistor Amplifiers: Analysis and Design*. 2nd edition. Prentice-Hall, 1997, p. 25.

See Also

`s2abcd` | `s2h` | `s2s` | `s2sdd` | `s2smm` | `s2scd` | `s2sdc` | `s2scc` | `s2rlgc` | `s2y` | `s2z` | `s2tf` | `smm2s` | `snp2smp`

s2y

Convert S-parameters to Y-parameters

Syntax

```
y_params = s2y(s_params,z0)
```

Description

`y_params = s2y(s_params,z0)` converts the scattering parameters to the admittance parameters.

Examples

Convert S-Parameters to Y-Parameters

Define the S-parameters and impedance.

```
s_11 = 0.61*exp(1i*165/180*pi);
s_21 = 3.72*exp(1i*59/180*pi);
s_12 = 0.05*exp(1i*42/180*pi);
s_22 = 0.45*exp(1i*(-48/180)*pi);
s_params = [s_11 s_12; s_21 s_22];
z0 = 50;
```

Convert the S-parameters to Y-parameters.

```
y_params = s2y(s_params,z0)
```

```
y_params = 2×2 complex

   0.0647 - 0.0059i  -0.0019 - 0.0025i
  -0.0826 - 0.2200i   0.0037 + 0.0145i
```

Input Arguments

`s_params` — *N*-port- S-Parameters

N-by-*N*-by-*M* array of complex numbers

N-port- S-Parameters, specified as an *N*-by-*N*-by-*M* array of a complex numbers, where *M* represents the number of frequency points of *N*-port S-parameters.

`z0` — Reference impedance

`50` (default) | positive real scalar

Reference impedance of *N*-port S-parameters, specified in positive real scalar as ohms.

Note `z0` must be a positive real scalar or vector. If `z0` is a vector, then the vector must be equal to the number of network parameter data points or frequency vector.

Output Arguments

`y_params` — *N*-port Y-Parameters

N-by-*N*-by-*M* array of complex numbers

N-port Y-Parameters, returned as an *N*-by-*N*-by-*M* array of complex numbers, where *M* represents the number of frequency points of *N*-port Y-parameters.

Version History

Introduced before R2006a

References

[1] Reveyrand, T. “Multiport Conversions between S, Z, Y, h, ABCD, and T Parameters.” *2018 International Workshop on Integrated Nonlinear Microwave and Millimetre-Wave Circuits (INMMIC)*, IEEE, 2018, pp. 1–3. *DOI.org (Crossref)*, doi:10.1109/INMMIC.2018.8430023.

See Also

`s2abcd` | `s2h` | `s2s` | `s2sdd` | `s2smm` | `s2scd` | `s2sdc` | `s2scc` | `s2rlgc` | `s2t` | `s2z` | `s2tf` | `smm2s` | `snp2smp`

s2tf

Convert S-parameters of 2-port network to voltage or power-wave transfer function

Syntax

```
tf = s2tf(s_params)
tf = s2tf(s_params,z0,zs,zl)
tf = s2tf(s_params,z0,zs,zl,option)

tf = s2tf(sparams_obj)
tf = s2tf(sparams_obj,zs,zl)
tf = s2tf(sparams_obj,zs,zl,option)
```

Description

`tf = s2tf(s_params)` converts the scattering parameters of a 2-port network to the voltage transfer function of the network.

`tf = s2tf(s_params,z0,zs,zl)` calculates the voltage transfer function using the reference impedance `z0`, source impedance `zs`, and load impedance `zl`.

`tf = s2tf(s_params,z0,zs,zl,option)` calculates the voltage or power-wave transfer function using the method specified by `option`.

`tf = s2tf(sparams_obj)` converts the 2-port S-parameter object, `hs`, into the voltage transfer function of the network.

`tf = s2tf(sparams_obj,zs,zl)` calculates the voltage transfer function using the source impedance `zs`, and load impedance `zl`.

`tf = s2tf(sparams_obj,zs,zl,option)` calculates the voltage or power-wave transfer function using the method specified by `option`.

Examples

S-Parameters to Voltage or Power Transfer Function

Calculate the voltage transfer function of an S-parameter array.

```
ckt = read(rfckt.passive,'passive.s2p');
sparams = ckt.NetworkData.Data;
tf = s2tf(sparams)
```

```
tf = 202×1 complex

   0.9964 - 0.0254i
   0.9960 - 0.0266i
   0.9956 - 0.0284i
   0.9961 - 0.0290i
   0.9960 - 0.0301i
```

```
   0.9953 - 0.0317i
   0.9953 - 0.0334i
   0.9952 - 0.0349i
   0.9949 - 0.0367i
   0.9946 - 0.0380i
      ⋮
```

Calculate Voltage Transfer Function of S-Parameters Object

Calculate the voltage transfer function of a S-parameters object using `s2tf` function.

```
sparams = sparameters('passive.s2p');
tf = s2tf(sparams)
```

```
tf = 202×1 complex

   0.9964 - 0.0254i
   0.9960 - 0.0266i
   0.9956 - 0.0284i
   0.9961 - 0.0290i
   0.9960 - 0.0301i
   0.9953 - 0.0317i
   0.9953 - 0.0334i
   0.9952 - 0.0349i
   0.9949 - 0.0367i
   0.9946 - 0.0380i
      ⋮
```

Input Arguments

`sparams_obj` — 2-port S-parameters

`sparameter` object

2-port S-parameters, specified as an RF Toolbox `sparameters` object.

`s_params` — Scattering parameters

2-by-2-by-*M* array (default)

Scattering parameters, specified as a complex 2-by-2-by-*M* array where *M* represents the number of frequency points of the S-parameters.

`z0` — Reference impedance

`50` (default) | positive scalar

Reference impedance of S-parameters, specified as a positive scalar in ohms.

`zs` — Source impedance

`50` (default) | positive scalar | vector

Source impedance of S-parameters, specified as a positive scalar or vector of length equal to the number of frequencies in ohms.

`zl` — Load impedance
`50` (default) | positive scalar | vector

Load impedance of S-parameters, specified as a positive scalar or vector of length equal to the number of frequencies in ohms.

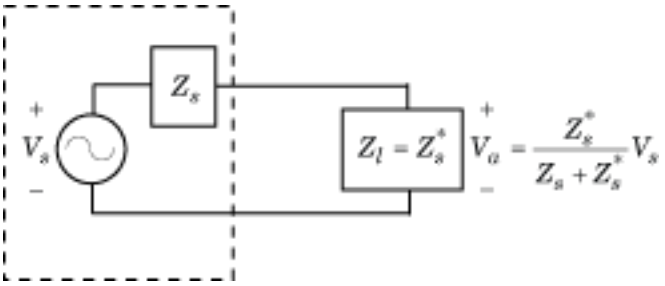
`option` — Transfer function type
1 (default) | integer

Transfer function type, specified as an integer equal to 1, 2, or 3.

- 1 — The transfer function is the gain from the incident voltage, V_a, to the output voltage for arbitrary source and load impedances:

  $$tf = \frac{V_l}{V_a}$$

  The following figure shows how to compute V_a from the source voltage V_s:

  For the S-parameters and impedance values, the transfer function is:

  $$tf = \frac{(Z_s + Z_s^*)}{Z_s^*} \frac{S_{21}(1 + \Gamma_l)(1 - \Gamma_s)}{2(1 - S_{22}\Gamma_l)(1 - \Gamma_{in}\Gamma_s)}$$

  where:

  $$\Gamma_l = \frac{Z_l - Z_o}{Z_l + Z_o}$$

  $$\Gamma_s = \frac{Z_s - Z_o}{Z_s + Z_o}$$

  $$\Gamma_{in} = S_{11} + \left(S_{12}S_{21}\frac{\Gamma_l}{(1 - S_{22}\Gamma_l)}\right)$$

  The following equation shows how the preceding transfer function is related to the transducer gain computed by the `powergain` function:

  $$G_T = |tf|^2 \frac{\mathrm{Re}(Z_l)}{|Z_l|^2} \frac{|Z_s|^2}{\mathrm{Re}(Z_s)}$$

  Notice that if Z_l and Z_S are real, $G_T = |tf|^2 \frac{Z_s}{Z_l}$.

- 2 — The transfer function is the gain from the source voltage to the output voltage for arbitrary source and load impedances:

$$tf = \frac{V_l}{V_s} = \frac{S_{21}(1+\Gamma_l)(1-\Gamma_s)}{2(1-S_{22}\Gamma_l)(1-\Gamma_{in}\Gamma_s)}$$

You can use this option to compute the transfer function $\frac{V_L}{V_{in}}$ by setting `zs` to `0`. This setting means that Γ_s = –1 and $V_{in} = V_s$.

- `3` — The transfer function is the power-wave gain from the incident power wave at the first port to the transmitted power wave at the second port:

$$tf = \frac{b_{p2}}{a_{p1}} = \frac{\sqrt{\mathrm{Re}(Z_l)\mathrm{Re}(Z_s)}}{Z_l}\frac{S_{21}(1+\Gamma_l)(1-\Gamma_s)}{(1-S_{22}\Gamma_l)(1-\Gamma_{in}\Gamma_s)}$$

Output Arguments

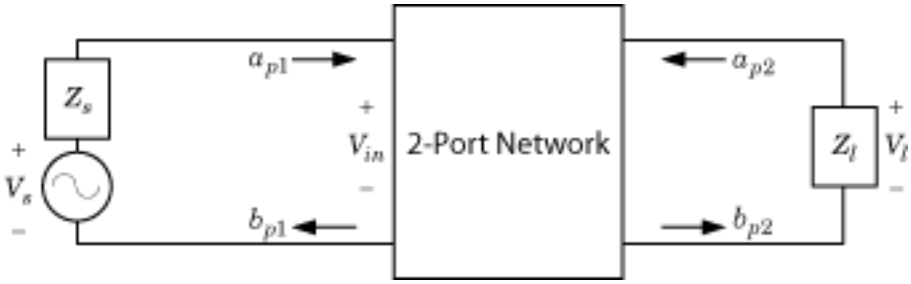
`tf` — Voltage transfer function

vector of doubles

Voltage transfer function, returned as a vector of doubles.

Algorithms

The following figure shows the setup for computing the transfer function, along with the impedances, voltages, and the power waves used to determine the gain.



The function uses the following voltages and power waves for calculations:

- V_l is the output voltage across the load impedance.
- V_s is the source voltage.
- V_{in} is the input voltage of the 2-port network.
- a_{p1} is the incident power wave, equal to $\frac{V_s}{2\sqrt{\mathrm{Re}(Z_s)}}$.
- b_{p2} is the transmitted power wave, equal to $\frac{\sqrt{\mathrm{Re}(Z_l)}}{Z_l}V_l$.

Version History

Introduced in R2006b

References

[1] Gonzalez, Guillermo. *Microwave Transistor Amplifiers: Analysis and Design*. 2nd ed, Prentice Hall, 1997.

See Also

`s2abcd` | `s2h` | `s2s` | `s2sdd` | `s2smm` | `s2scd` | `s2sdc` | `s2scc` | `s2rlgc` | `s2t` | `s2y` | `s2z` | `smm2s` | `snp2smp`

snp2smp

Convert and reorder single-ended N-port S-parameters to single-ended M-port S-parameters

Syntax

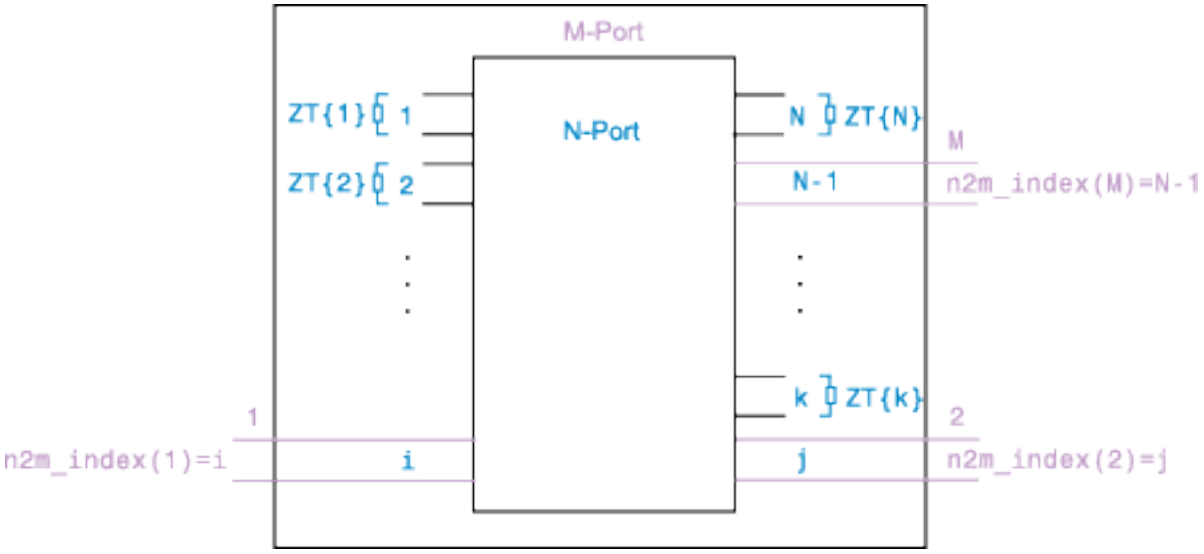
```
s_params_mp = snp2smp(s_params_np)
s_params_mp = snp2smp(s_params_np,Z0,n2m_index,ZT)
s_params_mp = snp2smp(s_obj,n2m_index,ZT)
```

Description

`s_params_mp = snp2smp(s_params_np)` convert and reorder the single-ended N-port S-parameters, `s_params_np`, into the single-ended M-port S-parameters, `s_params_mp`. *M* must be less than or equal to *N*.

`s_params_mp = snp2smp(s_params_np,Z0,n2m_index,ZT)` convert and reorder the S-parameter data using the optional arguments `Z0`, `n2m_index`, and `ZT` that control the conversion.

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



`s_params_mp = snp2smp(s_obj,n2m_index,ZT)` convert and reorder a S-parameters object, `s_obj`, into the single-ended M-port S-parameters, `s_params_mp`. *M* must be less than or equal to *N*.

Examples

Swap Ports of S-Parameters

Convert 3-port S-parameters to 3-port S-parameters with port indices swapped from `[1 2 3]` to `[2 3 1]` .

```
ckt = read(rfckt.passive,'default.s3p');
```

Default.s3p represents a real counterclockwise circulator.

```
s3p = ckt.NetworkData.Data;
Z0 = ckt.NetworkData.Z0;
s3p_new = snp2smp(s3p,Z0,[2 3 1]);
s3p_new = s3p_new(1:5)
```

```
s3p_new = 1×5 complex

   0.1431 - 0.7986i   0.0898 + 0.3177i  -0.0318 + 0.4208i  -0.0701 + 0.4278i   0.0503 - 0.8080i
```

3-Port to 2-Port S-Parameters

Convert 3-port S-parameters to 2-port S-parameters by terminating port 3 with an impedance of `Z0`.

```
ckt = read(rfckt.passive,'default.s3p');
s3p = ckt.NetworkData.Data;
Z0 = ckt.NetworkData.Z0;
s2p = snp2smp(s3p,Z0);
s2p_new = s2p(1:5)
```

```
s2p_new = 1×5 complex

  -0.0073 - 0.8086i   0.0869 + 0.3238i  -0.0318 + 0.4208i   0.1431 - 0.7986i  -0.0330 - 0.8060i
```

16-Port S-Parameters to 4-Port S-Parameters

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. Terminate the remaining 12 ports with an impedance of `Z0`.

```
S = sparameters('default.s16p');
s16p = S.Parameters;
Z0 = S.Impedance;
s4p = snp2smp(s16p,Z0,[1 16 2 15],Z0);
s4p = s4p(:,:,1)
```

```
s4p = 4×4 complex

   0.0857 - 0.1168i  -0.5372 - 0.6804i   0.0966 - 0.0706i   0.0067 + 0.0053i
  -0.5366 - 0.6860i   0.0803 - 0.1234i   0.0059 + 0.0048i   0.0977 - 0.0703i
   0.0957 - 0.0700i   0.0067 + 0.0048i   0.0818 - 0.1104i  -0.5362 - 0.6838i
   0.0055 + 0.0051i   0.0972 - 0.0703i  -0.5376 - 0.6840i   0.0761 - 0.1180i
```

16-Port S-Parameters to 4-Port S-Parameters Using Two Impedances

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 terminate port `4` with an impedance of `100` ohms and terminate the remaining `11` ports with an impedance of `50` ohms.

```
S = sparameters('default.s16p');
s16p = S.Parameters;
Z0 = S.Impedance;
ZT(1:16) = {50};
ZT{4} = 100;
s4p = snp2smp(s16p,Z0,[1 16 2 15],ZT);
s4p(:,:,1)
```

```
ans = 4×4 complex

   0.0857 - 0.1168i  -0.5372 - 0.6804i   0.0966 - 0.0706i   0.0067 + 0.0053i
  -0.5366 - 0.6860i   0.0803 - 0.1234i   0.0059 + 0.0048i   0.0977 - 0.0703i
   0.0957 - 0.0700i   0.0067 + 0.0048i   0.0818 - 0.1104i  -0.5362 - 0.6838i
   0.0055 + 0.0051i   0.0972 - 0.0703i  -0.5376 - 0.6840i   0.0761 - 0.1180i
```

Input Arguments

`s_params_np` — S-parameters

N-by-*N*-by-*K* array

S-parameters, specified as a *N*-by-*N*-by-*K* array , where *K* representing number of frequency points of a *N*-port S-parameters.

`s_obj` — S-parameter object

scalar handle objects

S-parameter object, specified as *N*-port scalar handle objects, which include numeric arrays of S-parameters.

`Z0` — Reference impedance

`50` (default) | positive real scalar

Reference impedance of *N*-port S-Parameters, specified as positive real scalar in ohms.

`n2m_index` — Mapping index

`[1, 2]` (default)

`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 the function converts to the `i`th port of `s_params_mp`. For example, the setting `[1, 2]` means that *M* is `2`, and the first two ports of the N-port S-parameters become the ports of the M-port parameters. The function terminates any additional ports with the impedances specified by `ZT`.

`ZT` — Termination Impedance

scalar | vector | cell array

Termination Impedance of the ports,`ZT`, specified as a scalar, vector, or cell array. If *M* is less than *N*, `snp2smp` terminates the *N*–*M* ports not listed in `n2m_index` using the values in `ZT`. If `ZT` is a scalar,

the function terminates all *N*–*M* ports not listed in `n2m_index` by the same impedance `ZT`. If `ZT` is a vector of length *K*, `ZT[i]` is the impedance that terminates all *N*–*M* ports of the `i`th frequency point not listed in `n2m_index`. If `ZT` is a cell array of length *N*, `ZT{j}` is the impedance that terminates the `j`th port of the N-port S-parameters. The function ignores impedances related to the ports listed in `n2m_index`. Each `ZT{j}` can be a scalar or a vector of length *K*.

Output Arguments

`s_params_mp` — Single-ended M-port S-parameters

M-by-*M*-by-*K* array | S-parameter object

Single-ended M-port S-parameters, returned as one of the following:

- If you provide `s_params_np` as an input, `s_params_mp` is returned as a *M*-by-*M*-by-*K* array representing *K* M-port S-parameters. where *M* representing number of frequency points of a single-ended *M*-port S-Parameters.
- If you provide `s_obj` as an input, `s_params_mp` is returned as a S-parameter object with following properties:
  - `NumPorts` — Number of ports, specified as an integer. The function calculates this value automatically when you create the object.
  - `Frequencies` — S-parameter frequencies, specified as a *K*-by-1 vector of positive real numbers sorted from smallest to largest. The function sets this property from the `filename` or `freq` input arguments.
  - `Parameters` — S-parameter data, specified as an *N*-by-*N*-by-*K* array of complex numbers. The function sets this property from the `filename` or `data` input arguments.
  - `Impedance` — Reference impedance in ohms, specified as a positive real scalar. The function sets this property from the `filename` or `Z0` input arguments. If no reference impedance is provided, the function uses a default value of `50`.

Version History

Introduced in R2007b

See Also

`s2abcd` | `s2h` | `s2s` | `s2sdd` | `s2smm` | `s2scd` | `s2sdc` | `s2scc` | `s2rlgc` | `s2t` | `s2y` | `s2z` | `s2tf` | `smm2s`

abcd2s

Convert ABCD-parameters to S-parameters

Syntax

```
s_params = abcd2s(abcd_params,z0)
```

Description

`s_params = abcd2s(abcd_params,z0)` converts the ABCD-parameters `abcd_params` into the scattering parameters `s_params`. `z0` is the reference impedance; its default is 50 ohms.

`s_params` is a complex 2*N*-by-2*N*-by-*M* array, where *M* representing number of frequency points of a 2*N*-port S-parameters.

For more information see, “RF Network Parameter Objects”.

Examples

Convert ABCD-Parameters to S-Parameters

Define a matrix of ABCD-parameters.

```
A =      0.999884396265344 +  0.000129274757618717i;
B =      0.314079483671772 +      2.51935878310427i;
C = -6.56176712108866e-007 + 6.67455405306704e-006i;
D =      0.999806365547959 +  0.000247230611054075i;
abcd_params = [A,B; C,D]
```

```
abcd_params = 2×2 complex

   0.9999 + 0.0001i   0.3141 + 2.5194i
  -0.0000 + 0.0000i   0.9998 + 0.0002i
```

Convert these ABCD parameters to S-parameters.

```
s_params = abcd2s(abcd_params)
```

```
s_params = 2×2 complex

   0.0038 + 0.0248i   0.9961 - 0.0250i
   0.9964 - 0.0254i   0.0037 + 0.0249i
```

Input Arguments

`abcd_params` — *N*-port- ABCD-Parameters

array of complex numbers

The `abcd_params` input is a complex 2*N*-by-2*N*-by-*M* array, where *M* representing number of frequency points of a 2*N*-port ABCD-parameters.

The function assumes that the ABCD-parameter matrices have distinct *A*, *B*, *C*, and *D* submatrices:

$$\begin{bmatrix}[A] & [B]\\ [C] & [D]\end{bmatrix}$$

`z0` — Reference impedance
`50` (default) | positive real scalar | vector

Reference impedance of *N*-port S-Parameters, specified as positive real scalar in ohms.

Note `z0` must be a positive real scalar or vector. If `z0` is a vector, then the vector must be equal to the number of network parameter data points or frequency vector.

Output Arguments

`s_params` — 2*N*-port hybrid S-Parameters
array of complex numbers

2*N*-port S-parameters, returned as a 2*N*-by-2*N*-by-*M* array of complex numbers, where *M* representing number of frequency points of a 2*N*-port S-Parameters.

Version History

Introduced before R2006a

References

[1] Pozar, David M. *Microwave Engineering*. 3rd ed, J. Wiley, 2005.

See Also

`abcd2y` | `abcd2z` | `abcd2h` | `s2abcd`

gamma2z

Convert reflection coefficient to impedance

Syntax

```
z = gamma2z(gamma)
z = gamma2z(gamma,z0)
```

Description

`z = gamma2z(gamma)` converts the reflection coefficient `gamma` to the impedance `z` using a reference impedance Z_0 of 50 ohms.

`z = gamma2z(gamma,z0)` converts the reflection coefficient `gamma` to the impedance `z` by:

- Computing the normalized impedance.
- Multiplying the normalized impedance by the reference impedance Z_0.

Examples

Impedance Calculation

Calculate impedance from given reference impedance and reflection coefficient values

```
z0 = 50;
gamma = 1/3;
z = gamma2z(gamma,z0)
```

```
z = 100.0000
```

Input Arguments

`gamma` — Reflection coefficient
scalar

Reflection coefficient specified as a scalar.

`z0` — Reference impedance
`50` (default)

Reference impedance, specified in scalar as ohms.

Note `z0` must be a positive real scalar or vector. If `z0` is a vector, then the vector must be equal to the number of network parameter data points or frequency vector.

Output Arguments

z — Impedance
array

Impedance obtained from reflection coefficient, returned as `z`

Algorithms

The following equation shows this conversion:

$$Z = Z_0 * \left(\frac{1 + \Gamma}{1 - \Gamma} \right)$$

Version History

Introduced in R2007a

References

[1] Ludwig, Reinhold, and Gene Bogdanov. *RF Circuit Design: Theory and Applications*. Prentice-Hall, 2009.

See Also

`gammain` | `gammaout` | `z2gamma`

s2scd

Convert 4-port, single-ended S-parameters to 2-port, cross-mode S-parameters (S_{cd})

Syntax

```
scd_params = s2scd(s_params)
scd_params = s2scd(s_params,option)
```

Description

`scd_params = s2scd(s_params)` converts the 2*N*-port single-ended S-parameters to *N*-portcross-mode S-parameters.

`scd_params = s2scd(s_params,option)` converts S-parameters based on the optional `option` argument, which indicates the port-ordering convention of the S-parameters.

Examples

Network Data to Cross-Mode S-Parameters

Convert network data to cross-mode S-Parameters using the default port ordering.

```
S = sparameters('default.s4p');
s4p = S.Parameters;
s_cd = s2scd(s4p);
```

Display the cross-mode S-Parameters at the first frequency.

```
s_cd_new = s_cd(:,:,1)
```

```
s_cd_new = 2×2 complex

   0.0015 - 0.0029i  -0.0005 + 0.0014i
   0.0003 - 0.0009i   0.0019 - 0.0027i
```

Input Arguments

`s_params` — 2*N*-port S-parameters

2*N*-by-2*N*-by-*M* array of complex numbers

2*N*-port S-parameters, specified as a 2*N*-by-2*N*-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2*N*-port S-parameters.

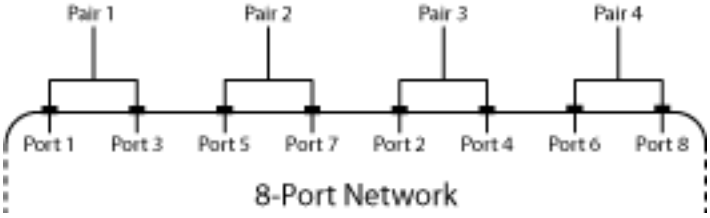
`option` — Port order

1 (default) | 2 | 3 | scalar

Port order, a scalar, specified as 1, 2, or 3. Port order determines how the function orders the ports:

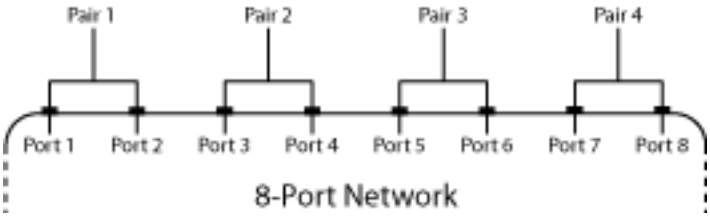
- 1 — `s2scd` pairs the odd-numbered ports together first, followed by the even-numbered ports. For example, in a single-ended, 8-port network:
  - Ports 1 and 3 become cross-mode pair 1.
  - Ports 5 and 7 become cross-mode pair 2.
  - Ports 2 and 4 become cross-mode pair 3.
  - Ports 6 and 8 become cross-mode pair 4.

  The following figure illustrates this convention for an 8-port device.



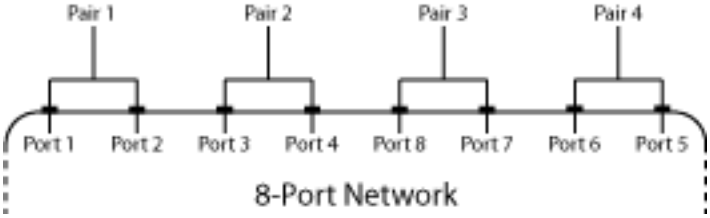
- 2 — `s2scd` pairs the input and output ports in ascending order. For example, in a single-ended, 8-port network:
  - Ports 1 and 2 become cross-mode pair 1.
  - Ports 3 and 4 become cross-mode pair 2.
  - Ports 5 and 6 become cross-mode pair 3.
  - Ports 7 and 8 become cross-mode pair 4.

  The following figure illustrates this convention for an 8-port device.



- 3 — `s2scd` pairs the input ports in ascending order and the output ports in descending order. For example, in a single-ended, 8-port network:
  - Ports 1 and 2 become cross-mode pair 1.
  - Ports 3 and 4 become cross-mode pair 2.
  - Ports 8 and 7 become cross-mode pair 3.
  - Ports 6 and 5 become cross-mode pair 4.

  The following figure illustrates this convention for an 8-port device.



Output Arguments

`scd_params` — *N*-port cross-mode S-Parameters
N-by-*N*-by-*M* array of complex numbers

N-port cross-mode S-Parameters, returned as a complex *N*-by-*N*-by-*M* array of complex numbers where *M* represents the number of frequency points of *N*-port cross-mode S-parameters.

Version History

Introduced in R2006a

References

[1] Fan, W., A. C. W. Lu, L. L. Wai, and B. K. Lok. "Mixed-Mode S-Parameter Characterization of Differential Structures."*Electronic Packaging Technology Conference*. pp. 533–537, 2003.

See Also

`s2abcd` | `s2h` | `s2s` | `s2sdd` | `s2smm` | `s2sdc` | `s2scc` | `s2rlgc` | `s2t` | `s2y` | `s2z` | `s2tf` | `smm2s` | `snp2smp`

s2abcd

Convert S-parameters to ABCD-parameters

Syntax

```
abcd_params = s2abcd(s_params,z0)
```

Description

`abcd_params = s2abcd(s_params,z0)` converts the scattering parameters to the ABCD-parameters.

Examples

Convert S-Parameters to ABCD-Parameters

Define a matrix of S-parameters.

```
s_11 = 0.61*exp(j*165/180*pi);
s_21 = 3.72*exp(j*59/180*pi);
s_12 = 0.05*exp(j*42/180*pi);
s_22 = 0.45*exp(j*(-48/180)*pi);
s_params = [s_11 s_12; s_21 s_22];
z0 = 50;
```

Convert S-parameters to ABCD-parameters.

```
abcd_params = s2abcd(s_params,z0)
```

```
abcd_params = 2×2 complex

   0.0633 + 0.0069i   1.4958 - 3.9839i
   0.0022 - 0.0024i   0.0732 - 0.2664i
```

Input Arguments

`s_params` — 2*N*-port S-Parameters

2*N*-by-2*N*-by-*M* array of complex numbers

2*N*-port S-parameters, specified as a 2*N*-by-2*N*-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2*N*-port S-Parameters.

`z0` — Reference impedance

`50` (default) | positive real scalar

Reference impedance of 2*N*-port S-parameters, specified as positive real scalar in ohms.

Note `z0` must be a positive real scalar or vector. If `z0` is a vector, then the vector must be equal to the number of network parameter data points or frequency vector.

Output Arguments

`abcd_params` — 2*N*-port ABCD parameters

2*N*-by-2*N*-by-*M* array of complex numbers

2*N*-port ABCD parameters, returned as a complex 2*N*-by-2*N*-by-*M* array, where *M* represents the number of frequency points of 2*N*-port ABCD-parameters. The output ABCD-parameters matrices have distinct *A*, *B*, *C*, and *D* submatrices:

$$\begin{bmatrix} [A] & [B] \\ [C] & [D] \end{bmatrix}$$

Version History

Introduced before R2006a

References

[1] Pozar, David M. *Microwave Engineering*. 3rd ed, J. Wiley, 2005.

See Also

`abcd2s`

rationalfit

Approximate data using stable rational function object

Syntax

```
fit = rationalfit(freq,data)
fit = rationalfit(freq,data,tol)
fit = rationalfit(___,Name,Value)
[fit,errdb] = rationalfit(___)

fit = rationalfit(s_obj,i,j)
```

Description

`fit = rationalfit(freq,data)` fits a rational function object of the form

$$F(s) = \sum_{k=1}^{n} \frac{C_k}{s - A_k} + D,\ s = j*2\pi f$$

to the complex vector `data` over the frequency values in the positive vector `freq`. The function returns a handle to the rational function object, `h`, with properties `A`, `C`, `D`, and `Delay`.

`fit = rationalfit(freq,data,tol)` fits a rational function object to complex data and constrains the error of the fit according to the optional input argument `tol`.

`fit = rationalfit(___,Name,Value)` fits a rational function object of the form

$$F(s) = \left(\sum_{k=1}^{n} \frac{C_k}{s - A_k} + D\right) e^{-s.Delay},\ \ s = j*2\pi f$$

with additional options specified by one or more `Name,Value` pair arguments. These arguments offer finer control over the performance and accuracy of the fitting algorithm.

`[fit,errdb] = rationalfit(___)` fits a rational function object to complex data and also returns `errdb`, which is the achieved error.

`fit = rationalfit(s_obj,i,j)` fits S_{ij} using `FREQ = s_obj.Frequencies` and `DATA = rfparam(s_obj,i,j)` for s-parameter object, `s_obj`.

Examples

Rational Function Approximation of S-parameter Data

Fit a rational function object to S-parameter data, and compare the results by plotting the object against the data.

Read the S-parameter data into an RF data object.

```
orig_data = read(rfdata.data,'passive.s2p');
freq = orig_data.Freq;
data = orig_data.S_Parameters(1,1,:);
```

Fit a rational function to the data using `rationalfit`.

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

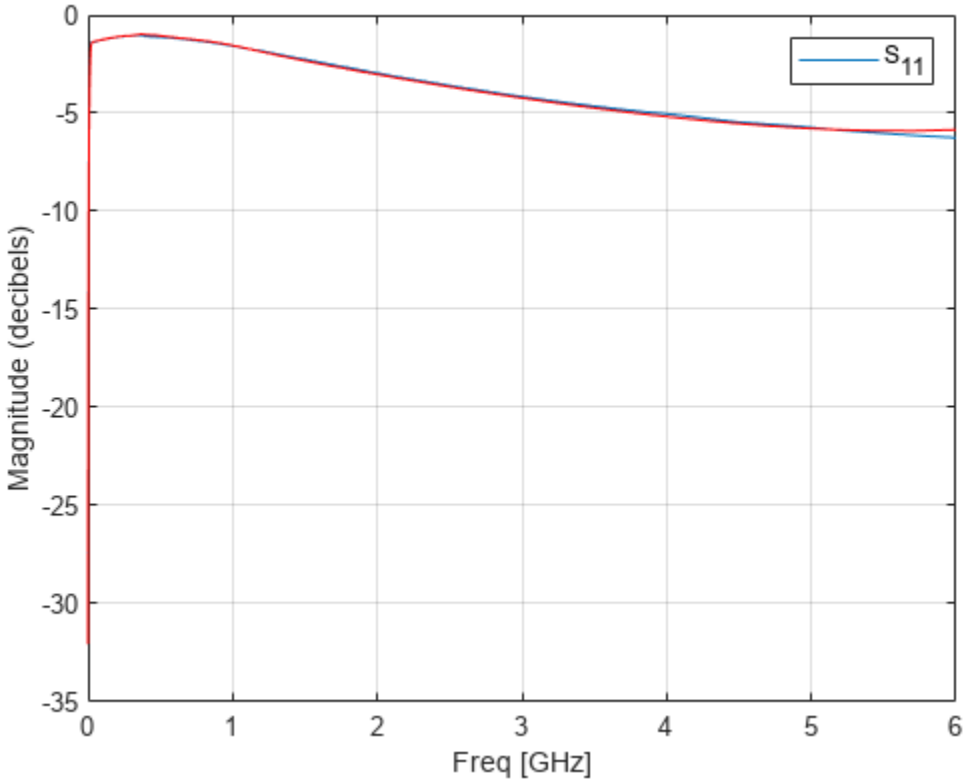
```
fit_data =
   rfmodel.rational with properties:

        A: [19x1 double]
        C: [19x1 double]
        D: 0
    Delay: 0
     Name: 'Rational Function'
```

Compute the frequency response of the rational function using `freqresp`.

```
[resp,freq] = freqresp(fit_data,freq);
```

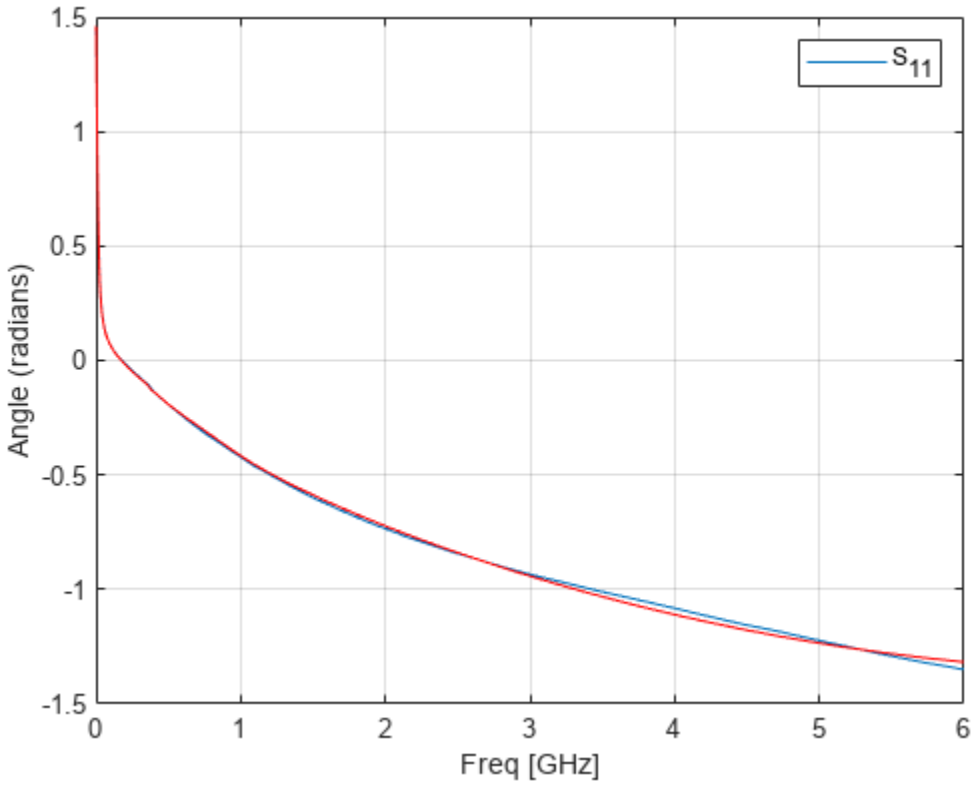
Plot the magnitude of the original data against the rational function approximation. S_{11} data appears in blue, and the rational function appears in red. Scaling the frequency values by `1e9` converts them to units of GHz.

```
figure
title('Rational fitting of S11 magnitude')
plot(orig_data,'S11','dB')
hold on
plot(freq/1e9,20*log10(abs(resp)),'r');
```



Plot the angle of the original data against the rational function approximation.

```
figure
title('Rational fitting of S11 angle')
plot(orig_data,'S11','Angle (radians)')
hold on
plot(freq/1e9,unwrap(angle(resp)),'r')
```



Rational Function Approximation of S-parameters

`rationalfit(freq,data)` also handles input 3D array of data (n x n x p), an input frequency array (p x 1), and returns a matrix (n x n) of rationalfit objects. Index into the matrix of rationalfit objects to access corresponding rationalfit information.

Use `rationalfit` on multiple datasets defined in a matrix.

```
orig_data = sparameters('defaultbandpass.s2p');
data = orig_data.Parameters;
freq = orig_data.Frequencies;
fit_data = rationalfit(freq, data)
```

```
fit_data =
  2x2 rfmodel.rational array with properties:

    A
    C
    D
    Delay
    Name
```

To access `rationalfit` data, use indexing on the rationalfit array. For example, to access the rational fit for the 1st element of the matrix, use:

```
S = fit_data(1, 1)

S =
   rfmodel.rational with properties:

        A: [12x1 double]
        C: [12x1 double]
        D: 0
    Delay: 0
     Name: 'Rational Function'
```

Fit S-Parameter Object

Use rational fit to fit an S-parameter object from the file 'passive.s2p'.

```
S = sparameters('passive.s2p');
fit = rationalfit(S,1,1,'TendsToZero',false)

fit =
   rfmodel.rational with properties:

        A: [5x1 double]
        C: [5x1 double]
        D: -0.4843
    Delay: 0
     Name: 'Rational Function'
```

Input Arguments

`freq` — Frequencies

vector of positive numbers

Frequencies over which the function fits a rational object, specified as a vector of length M.

`data` — Data to fit

N-by-N-by-M array of complex numbers (default) | vector of complex numbers

Data to fit, specified as an N-by-N-by-M array of complex numbers. The function fits N^2 rational functions to the data along the M (frequency) dimension.

`tol` — Error tolerance

-40 (default) | scalar

Error tolerance ε, specified as a scalar in units of dB. The error-fitting equation is

$$10^{\varepsilon/20} \geq \frac{\sqrt{\sum_{k=0}^{n} |W_k F_0\{f_k\} - F(s)|^2}}{\sqrt{\sum_{k=0}^{n} |W_k F_0\{f_k\}|^2}}$$

where

- ε is the specified value of `tol`.
- F_0 is the value of the original data (`data`) at the specified frequency f_k (`freq`).
- F is the value of the rational function at $s = j2\pi f$.
- W is the weighting of the data.

`rationalfit` computes the relative error as a vector containing the dependent values of the fit data. If the object does not fit the original data within the specified tolerance, a warning message appears.

`s_obj` — S-parameter object
network parameter object

S-parameters, specified as an RF Toolbox network parameter object. To create this type of object, use the `sparameters` function.

`i` — Row index
positive integer

Row index of data to plot, specified as a positive integer.

`j` — Column index
positive integer

Column index of data to plot, specified as a positive integer.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'DelayFactor',0.2`

`DelayFactor` — Delay factor
0 (default) | scalar from 0 to 1

Scaling factor that controls the amount of delay to fit to the data, specified as the comma-separated pair consisting of `'DelayFactor'` and a scalar between 0 and 1 inclusive. The `Delay` parameter, τ, of the rational function object is equal to the specified value of `'DelayFactor'` times an estimate of the group delay of the data. If the original data has delay, increasing this value might allow `rationalfit` to fit the data with a lower-order object.

`IterationLimit` — Maximum number of rationalfit iterations
[4,12] (default) | vector of positive integers

Maximum number of `rationalfit` iterations, specified as a vector of positive integers. Provide a two-element vector to specify minimum and maximum `[M1 M2]`. Increasing the limit extends the time that the algorithm takes to produce a fit, but it might produce more accurate results.

`NPoles` — Number of poles
`[0 48]` (default) | nonnegative integer | vector of two nonnegative integers

Number of poles A_k of the rational function, specified as the comma-separated pair consisting of `'NPoles'` and an integer n or range of possible values of n.

To help `rationalfit` produce an accurate fit, choose a maximum value of `npoles` greater than or equal to twice the number of peaks on a plot of the data in the frequency domain.

After completing a rational fit, the function removes coefficient sets whose residues (C_k) are zero. Thus, when you specify a range for `npoles`, the number of poles of the fit may be less than `npoles(1)`.

`TendsToZero` — Asymptotic behavior of fit
`true` (default) | `false`

Asymptotic behavior of the rational function as frequency approaches infinity, specified as the comma-separated pair consisting of `'TendsToZero'` and a logical value. When this argument is `true`, the resulting rational function variable D is zero, and the function tends to zero. A value of `false` allows a nonzero value for `D`.

`Tolerance` — Error tolerance
-40 (default) | scalar

Error tolerance ε, specified as the comma-separated pair consisting of `'Tolerance'` and a scalar in units of dB. The error-fitting equation is

$$10^{\varepsilon/20} \geq \frac{\sqrt{\sum_{k=0}^{n} \left| W_k F_0\{f_k\} - F(s) \right|^2}}{\sqrt{\sum_{k=0}^{n} \left| W_k F_0\{f_k\} \right|^2}}$$

where

- ε is the specified tolerance.
- F_0 is the value of the original data (`data`) at the specified frequency f_k (`freq`).
- F is the value of the rational function at $s = j2\pi f$.
- W is the weighting of the data.

If the object does not fit the original data within the specified tolerance, the function throws a warning.

`WaitBar` — Graphical wait bar
`false` (default) | `true`

Logical value that toggles display of the graphical wait bar during fitting, specified as the comma-separated pair consisting of `'WaitBar'` and either `true` or `false`. The `true` setting shows the graphical wait bar, and the `false` setting hides it. If you expect `rationalfit` to take a long time, and you want to monitor its progress, set `'WaitBar'` to `true`.

`Weight` — Weighting of data
`ones(size(`*`freq`*`))` (default) | vector of positive numbers

Weighting of the data at each frequency, specified as the comma-separated pair consisting of `'Weight'` and a vector of positive numbers or an array same as that of the data. Each entry in `weight` corresponds to a frequency in `freq`, so the length of `weight` must be equal to the length of

`freq`. Increasing the weight at a particular frequency improves the object fitting at that frequency. Specifying a weight of `0` at a particular frequency causes `rationalfit` to ignore the corresponding data point.

Output Arguments

`fit` — Rational function object

`rfmodel.rational` object

One or more rational function objects, returned as an *N*-by-*N* `rfmodel.rational` object. The number of dimensions in `data` determines the dimensionality of `h`.

`errdb` — Relative error

-40 (default) | double

Relative error achieved, returned as a `double`, in dB.

Tip

To see how well the object fits the original data, use the `freqresp` function to compute the frequency response of the object. Then, plot the original data and the frequency response of the rational function object. For more information, see the `freqresp` reference page or the above examples.

Version History

Introduced in R2006b

References

[1] Gustavsen.B and A.Semlyen, “Rational approximation of frequency domain responses by vector fitting,” *IEEE Trans. Power Delivery*, Vol. 14, No. 3, pp. 1052–1061, July 1999.

[2] Zeng.R and J. Sinsky, “Modified Rational Function Modeling Technique for High Speed Circuits,” *IEEE MTT-S Int. Microwave Symp. Dig.*, San Francisco, CA, June 11–16, 2006.

See Also

`timeresp` | `stepresp` | `freqresp` | `impulse` | `ispassive` | `makepassive` | `passivity`

s2sdc

Convert 4-port, single-ended S-parameters to 2-port, cross-mode S-parameters (S_{dc})

Syntax

```
sdc_params = s2sdc(s_params)
sdc_params = s2sdc(s_params,option)
```

Description

`sdc_params = s2sdc(s_params)` converts the 2*N*-port, single-ended S-parameters to *N*-port, cross-mode S-parameters.

`sdc_params = s2sdc(s_params,option)` converts S-parameters based on the optional `option` argument, which indicates the port-ordering convention of the S-parameters.

Examples

4-port Single-Ended S-Parameters to 2-port Cross-Mode S-Parameters

Convert network data to cross-mode S-Parameters using the default port ordering.

```
S = sparameters('default.s4p');
s4p = S.Parameters;
s_dc = s2sdc(s4p);
```

Display the 2-port cross-mode S-Parameters at the first frequency.

```
s_dc_new = s_dc(:,:,1)
```

```
s_dc_new = 2×2 complex

   0.0024 - 0.0035i  -0.0005 + 0.0019i
   0.0007 - 0.0012i   0.0023 - 0.0027i
```

Input Arguments

`s_params` — S-parameters

2*N*-by-2*N*-by-*M* array of complex numbers

S-parameters, specified as a complex 2*N*-by-2*N*-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2*N*-port S-parameters.

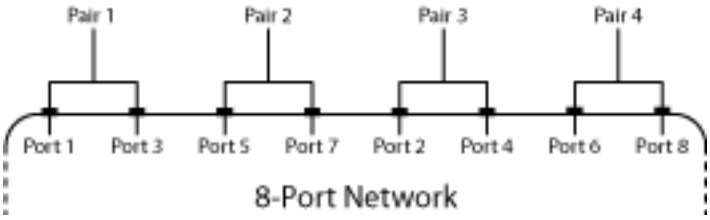
`option` — Port order

1 (default) | 2 | 3 | scalar

Port order, a scalar, specified as 1, 2, or 3. Port order determines how the function orders the ports:

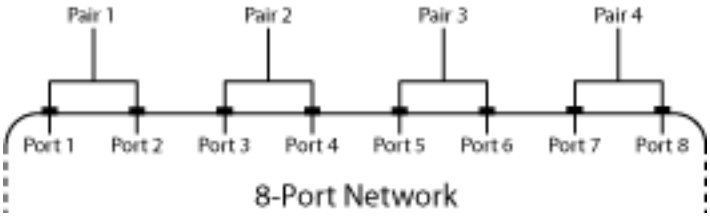
- 1 — `s2sdc` pairs the odd-numbered ports together first, followed by the even-numbered ports. For example, in a single-ended, 8-port network:
  - Ports 1 and 3 become cross-mode pair 1.
  - Ports 5 and 7 become cross-mode pair 2.
  - Ports 2 and 4 become cross-mode pair 3.
  - Ports 6 and 8 become cross-mode pair 4.

  The following figure illustrates this convention for an 8-port device.



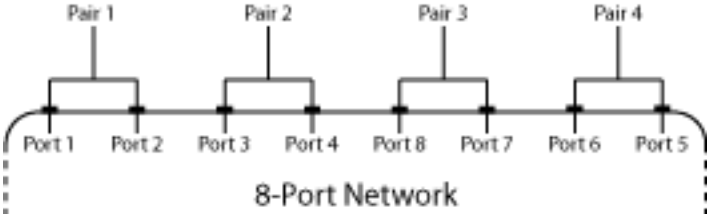
- 2 — `s2sdc` pairs the input and output ports in ascending order. For example, in a single-ended, 8-port network:
  - Ports 1 and 2 become cross-mode pair 1.
  - Ports 3 and 4 become cross-mode pair 2.
  - Ports 5 and 6 become cross-mode pair 3.
  - Ports 7 and 8 become cross-mode pair 4.

  The following figure illustrates this convention for an 8-port device.



- 3 — `s2sdc` pairs the input ports in ascending order and the output ports in descending order. For example, in a single-ended, 8-port network:
  - Ports 1 and 2 become cross-mode pair 1.
  - Ports 3 and 4 become cross-mode pair 2.
  - Ports 8 and 7 become cross-mode pair 3.
  - Ports 6 and 5 become cross-mode pair 4.

  The following figure illustrates this convention for an 8-port device.



Output Arguments

`sdc_params` — *N*-port cross-mode S-parameters
N-by-*N*-by-*M* array of complex numbers

N-port cross-mode S-parameters , returned as an *N*-by-*N*-by-*M* array of complex numbers, where *M* represents the number of frequency points of *N*-port cross-mode S-parameters (S_{dc}).

Version History

Introduced in R2006a

References

[1] Fan, W., et al. “Mixed-Mode S-Parameter Characterization of Differential Structures.” *Proceedings of the 5th Electronics Packaging Technology Conference (EPTC 2003)*, IEEE, 2003, pp. 533–37. *DOI.org (Crossref)*, doi:10.1109/EPTC.2003.1271579.

See Also

`s2abcd` | `s2h` | `s2s` | `s2sdd` | `s2smm` | `s2scd` | `s2scc` | `s2rlgc` | `s2t` | `s2y` | `s2z` | `s2tf` | `smm2s` | `snp2smp`

deembedsparams

De-embed 2N-port S-parameters

Syntax

```
s2_params = deembedsparams(s_params,s1_params,s3_params)

hs2 = deembedsparams(hs,hs1,hs3)
```

Description

`s2_params = deembedsparams(s_params,s1_params,s3_params)` de-embeds `s2_params` from cascaded S-parameters `s_params`, by removing the effects of `s1_params` and `s3_params`. `deembedsparams` assumes that you are using the port ordering shown here:

This function is ideal for situations in which the S-parameters of a `DUT` (device under test) must be de-embedded from S-parameters obtained through measurement.

`hs2 = deembedsparams(hs,hs1,hs3)` de-embeds S-parameter object, `hs2` from the chain `hs`.

Examples

De-embed S-Parameters of a DUT from a Cascaded 4-port Network

Read measured S-parameters of the cascaded network from `cascadedbackplanes.s4p`

```
S_measuredBJT = sparameters('cascadedbackplanes.s4p');
freq = S_measuredBJT.Frequencies;
```

Calculate the S-parameters of the left fixture of the network.

```
leftpad = circuit('left');
add(leftpad,[1 2],inductor(1e-9))
add(leftpad,[2 3],capacitor(100e-15))
setports(leftpad,[1 0],[3 0],[2 0],[3 0])
S_leftpad = sparameters(leftpad,freq)
```

```
S_leftpad =
  sparameters: S-parameters object

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

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

Calculate the S-parameters of the right fixture of the network.

```
rightpad = circuit('right');
add(rightpad,[1 3],capacitor(100e-15))
add(rightpad,[1 2],inductor(1e-9))
setports(rightpad,[1 0],[3 0],[2 0],[3 0])
S_rightpad = sparameters(rightpad,freq)
```

```
S_rightpad =
  sparameters: S-parameters object

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

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

De-embed the S-parameters of the DUT. The output is stored in `S-DUT` in MATLAB® workspace.

```
S_DUT = deembedsparams(S_measuredBJT,S_leftpad,S_rightpad)
```

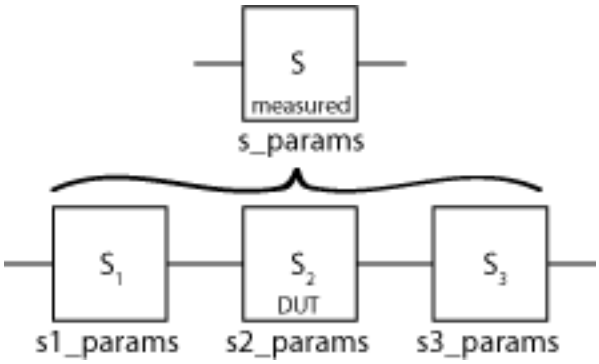
```
S_DUT =
  sparameters: S-parameters object

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

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

Input Arguments

`s_params,s1_params,s3_params` — S-parameter data

numeric arrays

S-parameter data, specified as `2Nx2NxK` arrays of `K` 2N-port S-parameters. `s_params` is the measured S-parameter array of the cascaded network. `s1_params` represents the first network of the cascade, and `s3_params` represents the third network. The function assumes that all networks in the cascade have the same reference impedance and are measured at the same frequencies. The function assumes the configuration of the cascade shown here:



Data Types: `double`

`hs,hs1,hs3` — S-parameter objects
scalar handle objects

S-parameter objects, specified as 2N-port scalar handle objects, which can include numeric arrays of S-parameters. The function checks that the `Frequencies` and `Impedance` properties are the same for all three inputs.

Data Types: `function_handle`

Output Arguments

`s2_params` — S-parameter data
numeric arrays

S-parameter data, returned as `2Nx2NxK` arrays of `K` 2N-port s-parameters, containing de-embedded S-parameters of the DUT (device under test).

Data Types: `double`

`hs2` — S-parameter objects
scalar handle object

S-parameter objects, returned as 2N-port scalar handle objects, containing de-embedded S-parameter objects of `DUT` (device under test).

Data Types: `function_handle`

Version History

Introduced before R2006a

See Also

`cascadesparams` | `rfckt.cascade`

Topics
“De-Embedding S-Parameters”

smm2s

Convert mixed-mode 2N-port S-parameters to single-ended 4N-port S-parameters

Syntax

```
s_params = smm2s(s_dd,s_dc,s_cd,s_cc)
s_params = smm2s(s_dd,s_dc,s_cd,s_cc,option)
```

Description

`s_params = smm2s(s_dd,s_dc,s_cd,s_cc)` converts mixed-mode, *N*-port S-parameters into single-ended, 2*N*-port S-parameters, `s_params`. `smm2s` maps the first half of the mixed-mode ports to the odd-numbered pairs of single-ended ports and maps the second half to the even-numbered pairs.

`s_params = smm2s(s_dd,s_dc,s_cd,s_cc,option)` converts the S-parameter data using the optional argument `option`. You can also reorder the ports in `s_params` using the `snp2smp` function.

Examples

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

Convert between mixed-mode and single-ended S-Parameters.

Create mixed-mode S-Parameters:

```
S = sparameters('default.s4p');
s4p = S.Parameters;
[sdd,scd,sdc,scc] = s2smm(s4p);
```

Convert them back to 4-port, single-ended S-Parameters.

```
s4p_converted_back = smm2s(sdd,scd,sdc,scc);
```

Display the single-ended S-Parameters at the first frequency.

```
s4p_converted_back_new = s4p_converted_back(:,:,1)
```

```
s4p_converted_back_new = 4×4 complex

   0.0857 - 0.1168i  -0.5372 - 0.6804i   0.0966 - 0.0706i   0.0067 + 0.0053i
  -0.5366 - 0.6860i   0.0803 - 0.1234i   0.0059 + 0.0048i   0.0977 - 0.0703i
   0.0957 - 0.0700i   0.0067 + 0.0048i   0.0818 - 0.1104i  -0.5362 - 0.6838i
   0.0055 + 0.0051i   0.0972 - 0.0703i  -0.5376 - 0.6840i   0.0761 - 0.1180i
```

Input Arguments

s_cc — S-parameters

array

S-parameters, specified as a complex *N*-by-*N*-by-*K* array containing *K* matrices of common-mode, *N*-port S-parameters (S_{cc}).

`s_cd` — S-parameters
array

S-parameters, specified as a complex *N*-by-*N*-by-*K* array containing *K* matrices of cross-mode, *N*-port S-parameters (S_{cd}).

`s_dc` — S-parameters
array

S-parameters, specified as a complex *N*-by-*N*-by-*K* array containing *K* matrices of cross-mode, *N*-port S-parameters (S_{dc}).

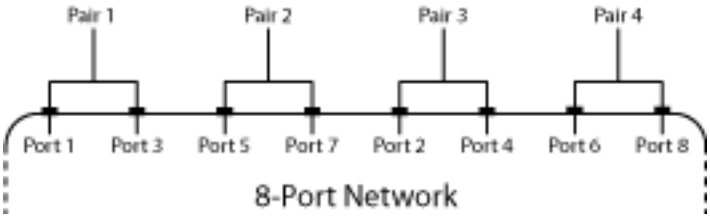
`s_dd` — S-parameters
array

S-parameters, specified as a complex *N*-by-*N*-by-*K* array containing *K* matrices of differential-mode, *N*-port S-parameters (S_{dd}).

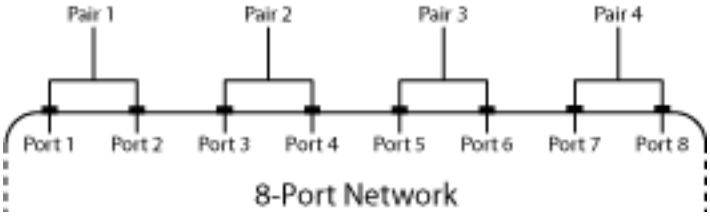
`option` — Port order
1 (default) | 2 | 3 | scalar

Port order, a scalar, specified as `1`, `2`,or `3`. Port order determines how the function orders the ports:

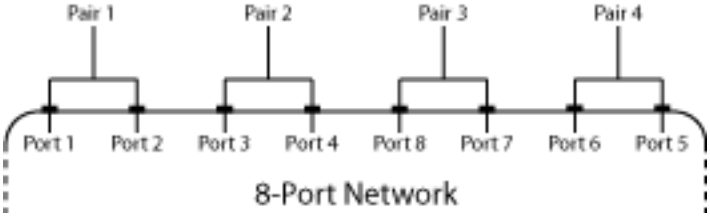
- `1` — `smm2s` maps the first half of the mixed-mode pairs to odd-numbered pairs of single-ended ports and maps the second half to even-numbered pairs. For example, in a mixed-mode, 4-port network:
  - Port 1 becomes single-ended ports 1 and 3.
  - Port 2 becomes single-ended ports 5 and 7.
  - Port 3 becomes single-ended ports 2 and 4.
  - Port 4 becomes single-ended ports 6 and 8.



- `2` — `smm2s` maps the first half of the mixed-mode pairs to single-ended ports in ascending numerical order, followed by the second half, also in ascending order. For example, in a mixed-mode, 4-port network:
  - Port 1 becomes single-ended ports 1 and 2.
  - Port 2 becomes single-ended ports 3 and 4.
  - Port 3 becomes single-ended ports 5 and 6.
  - Port 4 becomes single-ended ports 7 and 8.



- `3` — `smm2s` maps the first half of the mixed-mode pairs to single-ended ports in ascending numerical order. The function maps the second half to pairs of ports in descending order. For example, in a mixed-mode, 4-port network:
  - Port 1 becomes single-ended ports 1 and 2.
  - Port 2 becomes single-ended ports 3 and 4.
  - Port 3 becomes single-ended ports 8 and 7.
  - Port 4 becomes single-ended ports 6 and 5.



Output Arguments

`s_params` — S-parameters
array of complex numbers

S-parameters, returned as a 2*N*-by-2*N*-by-*K* array of complex numbers representing *K* single-ended, 2*N*-port S-parameters.

Version History

Introduced in R2009a

References

[1] Granberg, T.,.*Handbook of Digital Techniques for High-Speed Design*.Upper Saddle River, NJ: Prentice Hall, 2004.

See Also

`s2abcd` | `s2h` | `s2s` | `s2sdd` | `s2smm` | `s2scd` | `s2sdc` | `s2scc` | `s2rlgc` | `s2t` | `s2y` | `s2z` | `s2tf` | `snp2smp`

rlgc2s

Convert RLGC transmission line parameters to S-parameters

Syntax

```
s_params = rlgc2s(R,L,G,C,length,freq)
s_params = rlgc2s( ___ ,z0)
```

Description

`s_params = rlgc2s(R,L,G,C,length,freq)` transforms RLGC transmission line parameter data into S-parameters with a reference impedance of 50 Ω.

`s_params = rlgc2s( ___ ,z0)` transforms RLGC transmission line parameter data into S-parameters with a reference impedance of `z0`. Use this option with the input arguments in the previous syntax.

Examples

Convert RLGC Transmission Line Parameters to S-Parameters

Define the variables for a transmission line.

```
length = 1e-3;
freq = 1e9;
z0 = 50;
R = 50;
L = 1e-9;
G = .01;
C = 1e-12;
```

Calculate the s-parameters.

```
s_params = rlgc2s(R,L,G,C,length,freq,z0)
```

```
s_params = 2×2 complex

   0.0002 - 0.0001i   0.9993 - 0.0002i
   0.9993 - 0.0002i   0.0002 - 0.0001i
```

Input Arguments

R — Resistance matrix

N-by-*N*-by-*M* array

Resistance matrix, specified as an *N*-by-*N*-by-*M* array of distributed resistances, in units of Ω/m. The *N*-by-*N* matrices must be real symmetric, the diagonal terms must be nonnegative, and the off-diagonal terms must be nonnegative.

`L` — Inductance matrix
N-by-*N*-by-*M* array

Inductance matrix, specified as an *N*-by-*N*-by-*M* array of distributed inductances, in units of H/m. The *N*-by-*N* matrices must be real symmetric, the diagonal terms must be positive, and the off-diagonal terms must be nonnegative.

`G` — Conductance Matrix
N-by-*N*-by-*M* array

Conductance Matrix, specified as an *N*-by-*N*-by-*M* array of distributed conductances, in units of S/m. The *N*-by-*N* matrices must be real symmetric, the diagonal terms must be nonnegative, and the off-diagonal terms must be nonpositive.

`C` — Capacitance matrix
N-by-*N*-by-*M* array

Capacitance matrix, specified as an *N*-by-*N*-by-*M* array of distributed capacitances, in units of F/m. The matrices must be real symmetric, the diagonal terms must be positive, and the off-diagonal terms must be nonpositive.

`length` — Length of transmission line
scalar

Length of transmission line, specified as a scalar in meters.

`freq` — Frequency
M-by-1

Frequency, Specified as a vector of *M* frequencies over which the transmission line parameters are defined.

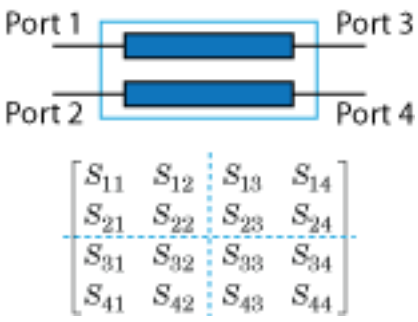
`z0` — Reference impedance
`50` (default) | positive real scalar

Reference impedance of *N*-port S-Parameters, specified as positive real scalar in ohms.

Output Arguments

`s_params` — S-parameters
2*N*-by-2*N*-by-*M* array

S-parameters, returned as a 2*N*-by-2*N*-by-*M* array of complex numbers. The following figure describes the port ordering convention of the output.



This port ordering convention assumes that:

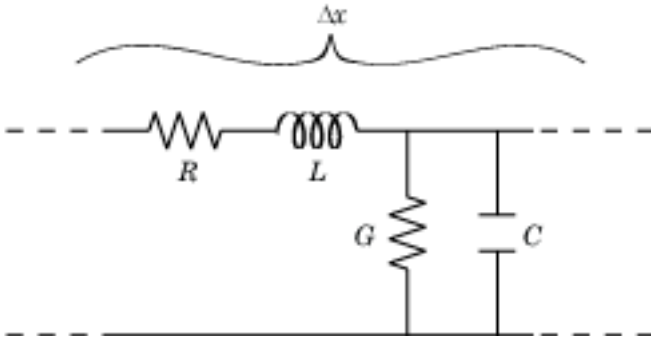
- Each 2*N*-by-2*N* matrix consists of *N* input terminals and *N* output terminals.
- The first *N* ports (1 through *N*) of the S-parameter matrix are input ports.
- The last *N* ports (*N* + 1 through 2*N*) are output ports.

To reorder ports after using this function, use the `snp2smp` function.

More About

RLCG Transmission Line Model

The following figure illustrates the RLGC transmission line model.



The representation consists of:

- The distributed resistance, *R*, of the conductors, represented by a series resistor.
- The distributed inductance, *L*, represented by a series inductor.
- The distributed conductance, *G*,
- The distributed capacitance, *C*, between the two conductors, represented by a shunt capacitor.

RLGC component units are all per unit length Δ*x*.

Version History

Introduced in R2011b

References

[1] Bhatti, A. A. “A Computer Based Method for Computing the N-Dimensional Generalized ABCD Parameter Matrices of N-Dimensional Systems with Distributed Parameters.” *[1990] Proceedings. The Twenty-Second Southeastern Symposium on System Theory*, IEEE Comput. Soc. Press, 1990, pp. 590–93. *DOI.org (Crossref)*, doi:10.1109/SSST.1990.138213.

See Also

`s2rlgc`

s2scc

Convert single-ended S-parameters to common-mode S-parameters (S_{cc})

Syntax

```
scc_params = s2scc(s_params)
scc_params = s2scc(s_params,option)
```

Description

`scc_params = s2scc(s_params)` converts the 2*N*-port single-ended S-parameters to *N*-port common-mode S-parameters.

`scc_params = s2scc(s_params,option)` converts S-parameters based on the port-ordering convention specified in `option` argument.

Examples

Network Data to Common-Mode S-Parameters

Convert network data to common-mode S-Parameters.

```
s_params = sparameters('default.s4p');
s4p = s_params.Parameters;
s_cc = s2scc(s4p);
```

To display common-mode S-Parameters at the first frequency, type the following command:

```
s_cc_new = s_cc(:,:,1)
```

```
s_cc_new = 2×2 complex

   0.1799 - 0.1839i  -0.5300 - 0.6771i
  -0.5314 - 0.6800i   0.1756 - 0.1910i
```

Input Arguments

`s_params` — 2*N*-port Single-ended S-parameters

2*N*-by-2*N*-by-*M* array of complex numbers

2*N*-port single ended S-parameters, specified as a 2*N*-by-2*N*-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2*N*-port single-ended S-Parameters.

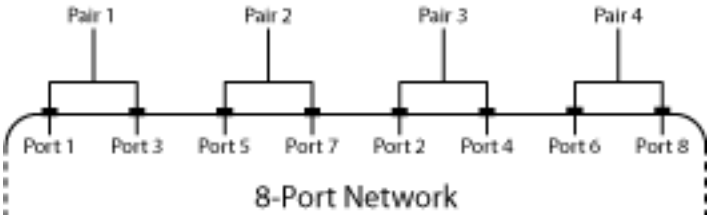
`option` — Port order

1 (default) | 2 | 3 | scalar

Port order, a scalar, specified as 1, 2, or 3. Port order determines how the function orders the ports:

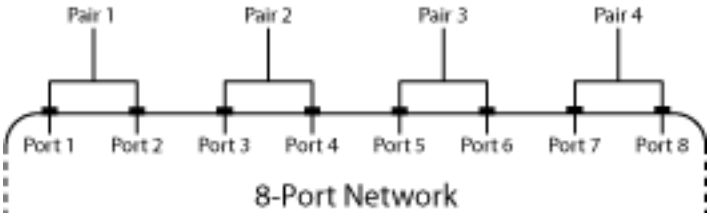
- 1 — `s2scc` pairs the odd-numbered ports together first, followed by the even-numbered ports. For example, in a single-ended, 8-port network:
  - Ports 1 and 3 become common-mode pair 1.
  - Ports 5 and 7 become common-mode pair 2.
  - Ports 2 and 4 become common-mode pair 3.
  - Ports 6 and 8 become common-mode pair 4.

  The following figure illustrates this convention for an 8-port device.



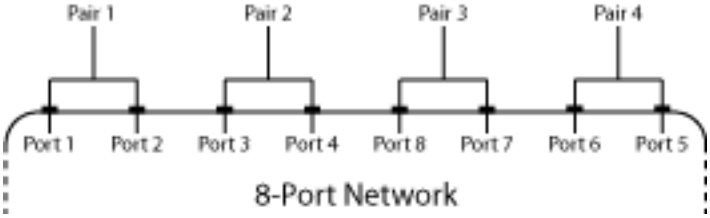
- 2 — `s2scc` pairs the input and output ports in ascending order. For example, in a single-ended, 8-port network:
  - Ports 1 and 2 become common-mode pair 1.
  - Ports 3 and 4 become common-mode pair 2.
  - Ports 5 and 6 become common-mode pair 3.
  - Ports 7 and 8 become common-mode pair 4.

  The following figure illustrates this convention for an 8-port device.



- 3 — `s2scc` pairs the input ports in ascending order and the output ports in descending order. For example, in a single-ended, 8-port network:
  - Ports 1 and 2 become common-mode pair 1.
  - Ports 3 and 4 become common-mode pair 2.
  - Ports 8 and 7 become common-mode pair 3.
  - Ports 6 and 5 become common-mode pair 4.

  The following figure illustrates this convention for an 8-port device.



Output Arguments

`scc_params` — *N*-port common-mode S-parameters
N-by-*N*-by-*M* array of complex numbers

N-port common-mode S-parameters , returned as an *N*-by-*N*-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2*N*-port common-mode S-parameters (S_{cc}).

Version History

Introduced in R2006a

References

[1] Fan, W., et al. “Mixed-Mode S-Parameter Characterization of Differential Structures.” *Proceedings of the 5th Electronics Packaging Technology Conference (EPTC 2003)*, IEEE, 2003, pp. 533–37. *DOI.org (Crossref)*, doi:10.1109/EPTC.2003.1271579.

See Also

`s2abcd` | `s2h` | `s2s` | `s2sdd` | `s2smm` | `s2scd` | `s2sdc` | `s2rlgc` | `s2t` | `s2y` | `s2z` | `s2tf` | `smm2s`

zpk

Compute zeros, poles, and gain of rational object

Syntax

```
[z,p,k,dcgain] = zpk(fit)
```

Description

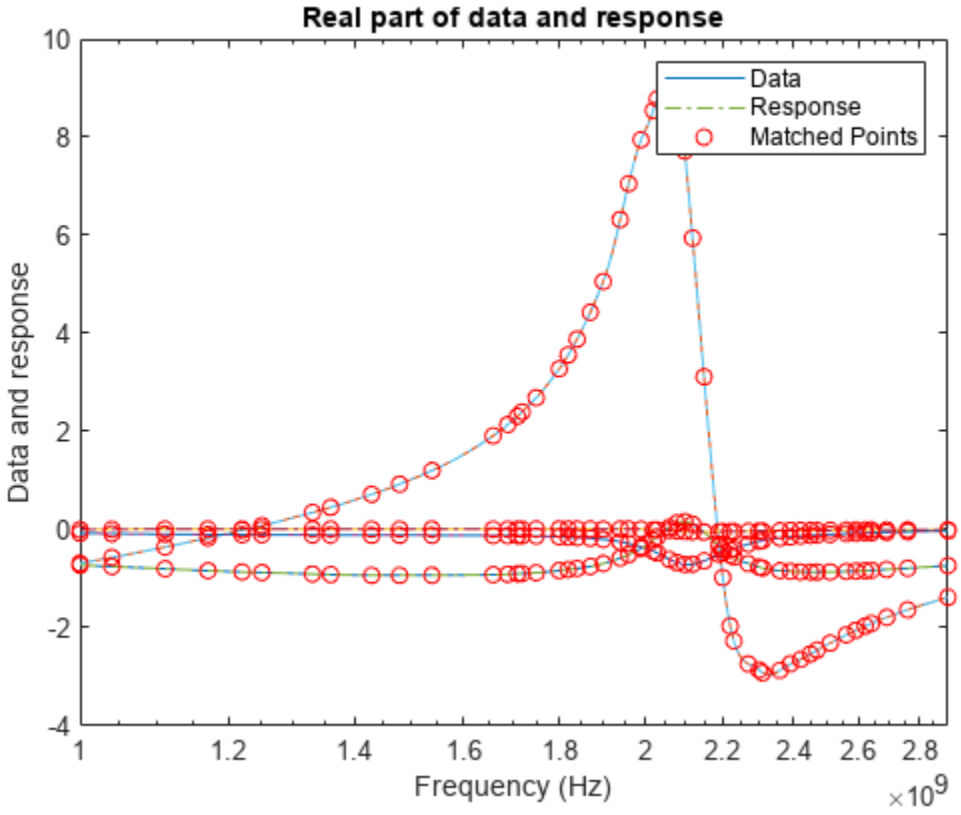
`[z,p,k,dcgain] = zpk(fit)` returns the zeros, poles, gain, and DC gain of a rational object.

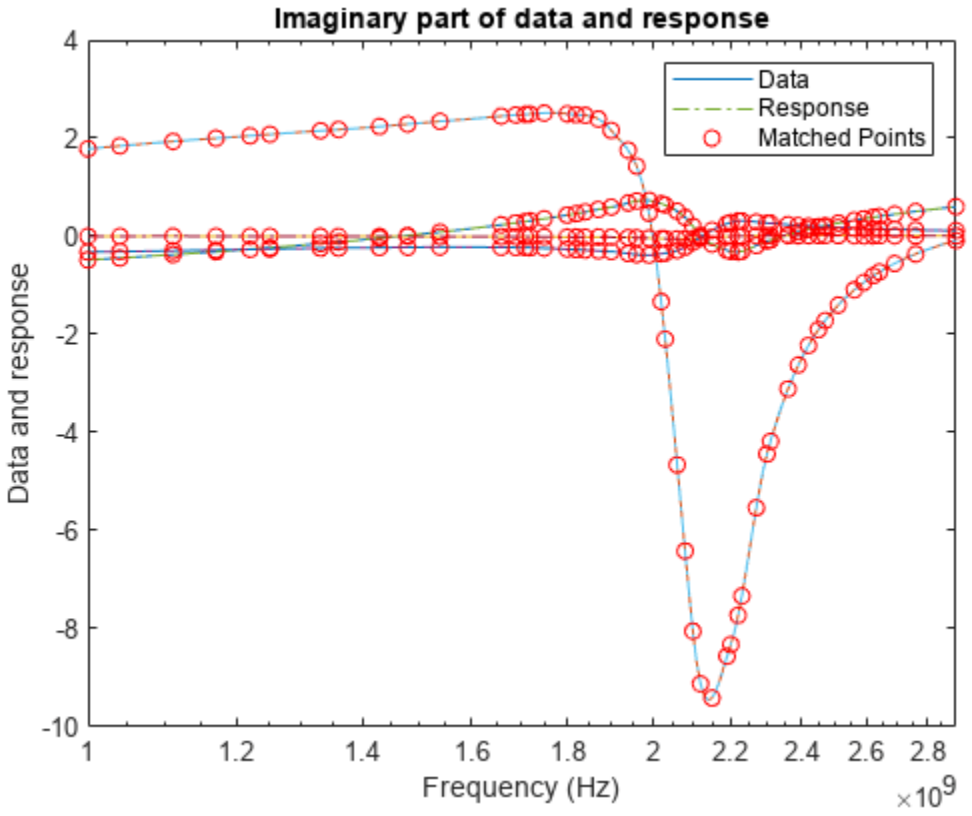
Examples

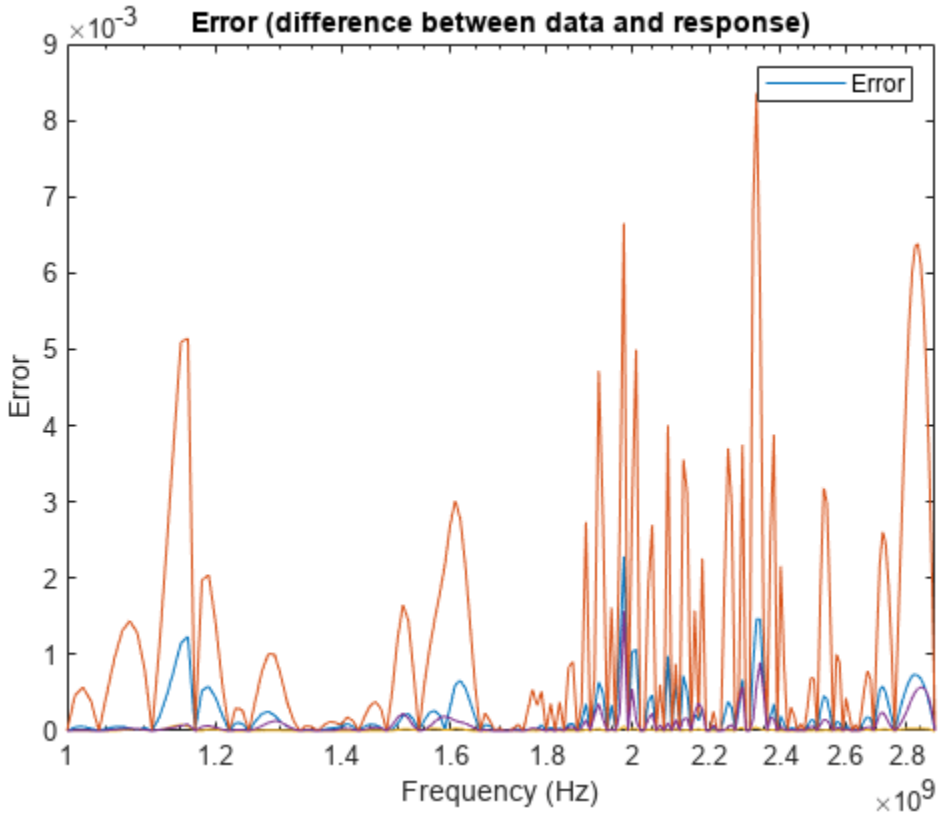
Zeros, Poles, Gain, and DC Gain of Fit

Create an S-Parameters object from the file named `default.s2p`. Perform rational fitting of the S-Parameters.

```
S = sparameters('default.s2p');
fit = rational(S,Display='plot')
```




Imaginary part of data and response
Data
Response
Matched Points
Data and response
Frequency (Hz)
×10^9
4
2
0
-2
-4
-6
-8
-10
1
1.2
1.4
1.6
1.8
2
2.2
2.4
2.6
2.8




```
fit =
  rational with properties:

      NumPorts: 2
      NumPoles: 52
         Poles: [52x1 double]
      Residues: [2x2x52 double]
    DirectTerm: [2x2 double]
         ErrDB: -22.6872
```

Calculate the zeros, poles, gain, and DC gain of the rational object.

```
[z,p,k,dcgain] = zpk(fit)
```

```
z=2×2 cell array
    {51x1 double}    {51x1 double}
    {51x1 double}    {51x1 double}


p=2×2 cell array
    {52x1 double}    {52x1 double}
    {52x1 double}    {52x1 double}


k = 2×2
10^10 ×
```

```
    1.0544   -0.0194
    0.9158    0.0377


dcgain = 2×2

    0.1289   -0.0838
   -0.1209    0.7649
```

Input Arguments

`fit` — Rational fit
`rational` object | `rfmodel.rational` object

Rational fit, specified as a `rational` or `rfmodel.rational` object.

Output Arguments

`z` — Zeroes of fit
1-D array of doubles | 3-D array of doubles

Zeroes of the fit, returned as a 1-D array of doubles or a 3-D array of doubles.

`p` — Poles of fit
1-D array of doubles | 3-D array of doubles

Poles of the fit, returned as a 1-D array of doubles or a 3-D array of doubles.

`k` — Gain of fit
2-D array of doubles

Gain of the fit, returned as a 2-D array of doubles. `k` is the coefficient of the rational function when poles and zeros are expressed as monic polynomials in `S`.

`dcgain` — DC gain of fit
2-D array of doubles

DC gain of the fit, returned as 2-D array of doubles for zero frequency response.

Version History

Introduced in R2020a

See Also

`rationalfit` | `ispassive` | `makepassive` | `passivity`

zpk

Converts rffilter to zero-pole-gain representation

Syntax

```
[z,p,k] = zpk(filter)
```

Description

`[z,p,k] = zpk(filter)` returns zero-pole-gain representation of S-parameters, S_{ij} contained in z{i,j}, p, and k{i,j} of the filter. This method only works for the `'Transfer function'` implementation of `rffilter` object.

Examples

Generate Zeroes, Poles, and Gain of Chebyshev filter

Generate the zpk of a high-pass fourth-order Chebyshev filter for cut-off frequency of 1 rad/sec.

Create the rffilter object.

```
filtobj = rffilter('FilterType',"Chebyshev",'ResponseType','Highpass', ...
    'FilterOrder',4,'Implementation',"Transfer function",                ...
    "PassbandFrequency",1/(2*pi),"Zin",50,"Zout",50);
```

Use `zpk` function to generate the zeroes, poles, and gain.

```
[zeros,poles,gain] = zpk(filtobj);
zeros{1,1}
```

```
ans = 4×1 complex

   0.0000 + 1.0824i
   0.0000 - 1.0824i
   0.0000 + 2.6131i
   0.0000 - 2.6131i
```

```
poles
```

```
poles = 4×1 complex

  -0.0941 + 1.0482i
  -0.0941 - 1.0482i
  -1.0482 + 2.0022i
  -1.0482 - 2.0022i
```

```
gain{1,1}
```

```
ans = 0.1250
```

Input Arguments

`filter` — RF filter
`rffilter` object

RF filter, specified as an `rffilter` object.

Output Arguments

z — Zeroes of filter
2-by-2 cell array

Zeroes of the filter, returned as a 2-by-2 cell array. Each cell contains zeros corresponding to its S-parameter.

p — Poles of filter
1-D array of doubles | 2-D array of doubles

Poles of the filter, returned as a 1-D array of doubles or a 2-D array of doubles.

k — Gain of filter
2-by-2 cell array

Gain of the filter, returned as a 2-by-2 cell array. k{i,j} corresponds to the gain of the S_{ij} S-parameter.

Version History

Introduced in R2019b

See Also

`rffilter` | `tf`

tf

Converts rffilter to transfer function

Syntax

```
[numerator,denominator] = tf(filter)
```

Description

`[numerator,denominator] = tf(filter)` returns numerators of S-parameters, S_{ij} contained in num(i,j) and the denominator of the filter object. This method only works for the `'Transfer function'` implementation of `rffilter` object.

Examples

Generate Transfer Function of Butterworth Filter

Generate the transfer function of a low-pass fourth-order Butterworth filter for cut-off frequency of 1 rad/sec.

Create the rffilter object.

```
filtobj = rffilter('FilterType',"Butterworth",'ResponseType','Lowpass', ...
    'FilterOrder',4,'Implementation',"Transfer function",               ...
    "PassbandFrequency",1/(2*pi),"Zin",50,"Zout",50);
```

Use tf function to generate the transfer function.

```
[numerator,denominator] = tf(filtobj);
```

The numerator of the transfer function for S21 is:

```
numerator{2,1}
```

```
ans = 1×5

     0     0     0     0     1
```

The denominator of the transfer function for S21 is:

```
denominator
```

```
denominator = 1×5

    1.0000    2.6131    3.4142    2.6131    1.0000
```

The corresponding polynomials in factored form is:

```
filtobj.DesignData.Numerator21
```

```
ans = 2×3

     0     0     1
     0     0     1
```

```
filtobj.DesignData.Denominator
```

```
ans = 2×3

    1.0000    0.7654    1.0000
    1.0000    1.8478    1.0000
```

Input Arguments

`filter` — RF filter
`rffilter` object

RF filter, specified as an `rffilter` object.

Output Arguments

`numerator` — Numerators of S-parameters
cell array

Numerators of S-parameters, returned as a cell array of S_{ij} contained in num{i,j}.

`denominator` — Denominator of coefficients
row vector

Denominator of coefficients, returned as a row vector.

Version History

Introduced in R2019b

See Also

`zpk` | `rffilter`

abcd2h

Convert ABCD-parameters to hybrid h-parameters

Syntax

```
h_params = abcd2h(abcd_params)
```

Description

`h_params = abcd2h(abcd_params)` converts the ABCD-parameters to the hybrid parameters. For more information see, “RF Network Parameter Objects”.

Examples

Convert ABCD-Parameters to h-Parameters

Define a matrix for ABCD-parameters.

```
A =      0.999884396265344 +  0.000129274757618717i;
B =      0.314079483671772 +      2.51935878310427i;
C = -6.56176712108866e-007 + 6.67455405306704e-006i;
D =      0.999806365547959 +  0.000247230611054075i;
abcd_params = [A,B; C,D];
```

Convert ABCD-parameters to h-parameters.

```
h_params = abcd2h(abcd_params)
```

```
h_params = 2×2 complex

   0.3148 + 2.5198i   0.9999 + 0.0001i
  -1.0002 + 0.0002i  -0.0000 + 0.0000i
```

Input Arguments

abcd_params — 2-port- ABCD-Parameters

2-by-2-by-*M* array of complex numbers

2-port- ABCD-Parameters, specified as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port ABCD-parameters.

The function assumes that the ABCD-parameter matrices have distinct *A*, *B*, *C*, and *D* submatrices:

$$\begin{bmatrix}[A] & [B]\\ [C] & [D]\end{bmatrix}$$

Output Arguments

`h_params` — 2-port hybrid h-parameters
2-by-2-by-*M* array of complex numbers

2-port hybrid h-parameters, returned as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port hybrid h-parameters.

Version History

Introduced before R2006a

See Also

`abcd2s` | `abcd2y` | `abcd2z` | `z2abcd` | `h2abcd`

abcd2y

Convert ABCD-parameters to Y-parameters

Syntax

```
y_params = abcd2y(abcd_params)
```

Description

`y_params = abcd2y(abcd_params)` converts the ABCD-parameters to the admittance parameters.

For more information see, “RF Network Parameter Objects”.

Examples

Convert ABCD-Parameters to Y-Parameters

Define a matrix of ABCD parameters.

```
A =      0.999884396265344 +  0.000129274757618717i;
B =      0.314079483671772 +      2.51935878310427i;
C = -6.56176712108866e-007 + 6.67455405306704e-006i;
D =      0.999806365547959 +  0.000247230611054075i;
abcd_params = [A,B; C,D];
```

Convert ABCD-parameters to Y-parameters.

```
y_params = abcd2y(abcd_params)
```

```
y_params = 2×2 complex

   0.0488 - 0.3908i  -0.0489 + 0.3907i
  -0.0487 + 0.3909i   0.0488 - 0.3908i
```

Input Arguments

`abcd_params` — 2*N*-port- ABCD-Parameters

2*N*-by-2*N*-by-*M* array of complex numbers

2*N*-port- ABCD-Parameters, specified as a 2*N*-by-2*N*-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2*N*-port ABCD-parameters.

The function assumes that the ABCD-parameter matrices have distinct *A*, *B*, *C*, and *D* submatrices:

$$\begin{bmatrix}[A] & [B]\\ [C] & [D]\end{bmatrix}$$

Output Arguments

`y_params` — 2*N*-port Y-Parameters
2*N*-by-2*N*-by-*M* array of complex numbers

2*N*-port Y-Parameters, returned as a 2*N*-by-2*N*-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2 *N*-port Y-parameters.

Version History

Introduced before R2006a

See Also

`s2y` | `abcd2h` | `abcd2s` | `abcd2z` | `h2y` | `y2abcd` | `z2y`

abcd2z

Convert ABCD-parameters to Z-parameters

Syntax

```
z_params = abcd2z(abcd_params)
```

Description

`z_params = abcd2z(abcd_params)` converts the ABCD-parameters to the impedance parameters.

Examples

Convert ABCD-Parameters to Z-Parameters

Define a matrix for ABCD-parameters.

```
A =      0.999884396265344 +  0.000129274757618717i;
B =      0.314079483671772 +      2.51935878310427i;
C = -6.56176712108866e-007 + 6.67455405306704e-006i;
D =      0.999806365547959 +  0.000247230611054075i;
abcd_params = [A,B; C,D];
```

Convert ABCD-parameters to Z-parameters.

```
z_params = abcd2z(abcd_params)
```

```
z_params = 2×2 complex
10^5 ×

  -0.1457 - 1.4837i  -0.1453 - 1.4835i
  -0.1459 - 1.4839i  -0.1455 - 1.4836i
```

Input Arguments

abcd_params — 2*N*-port- ABCD-Parameters

2*N*-by-2*N*-by-*M* array of complex numbers

2*N*-port- ABCD-Parameters, specified as a 2*N*-by-2*N*-by-*M* array of complex numbers, where *M* representing number of frequency points of 2*N*-port ABCD-parameters.

The function assumes that the ABCD-parameter matrices have distinct *A*, *B*, *C*, and *D* submatrices:

$$\begin{bmatrix}[A] & [B]\\ [C] & [D]\end{bmatrix}$$

Output Arguments

`z_params` — 2*N*-port Z-parameters
2*N*-by-2*N*-by-*M* array of complex numbers

2*N*-port Z-parameters, returned as a 2*N*-by-2*N*-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2*N*-port Z-parameters.

Version History

Introduced before R2006a

See Also

`abcd2h` | `abcd2s` | `abcd2y` | `h2y` | `y2abcd` | `z2abcd`

z2abcd

Convert Z-parameters to ABCD-parameters

Syntax

```
abcd_params = z2abcd(z_params)
```

Description

`abcd_params = z2abcd(z_params)` converts the Z-parameters to ABCD-parameters.

Examples

Convert Z-Parameters to ABCD-Parameters

Define a matrix of Z-parameters.

```
Z11 = -14567.2412789287 - 148373.315116592i;
Z12 = -14588.1106171651 - 148388.583516562i;
Z21 = -14528.0522132692 - 148350.705757767i;
Z22 = -14548.5996561832 - 148363.457002006i;
z_params = [Z11,Z12; Z21,Z22];
```

Convert Z-parameters to ABCD-parameters.

```
abcd_params = z2abcd(z_params)
```

```
abcd_params = 2×2 complex

   1.0002 - 0.0002i   0.3151 + 2.5200i
  -0.0000 + 0.0000i   1.0001 - 0.0001i
```

Input Arguments

`z_params` — 2*N*-port Z-parameters

2*N*-by-2*N*-by-*M* array of complex numbers

2*N*-port Z-parameters, specified as a 2*N*-by-2*N*-by-*M* array of complex numbers, where *M* represents the number of frequency points of *N*-port Z-parameters.

Output Arguments

`abcd_params` — 2*N*-port- ABCD-Parameters

2*N*-by-2*N*-by-*M* array of complex numbers

2*N*-port- ABCD-Parameters, specified as a 2*N*-by-2*N*-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2*N*-port ABCD-parameters.

The function assumes that the ABCD-parameter matrices have distinct A, B, C, and D submatrices:

$$\begin{bmatrix}[A] & [B]\\ [C] & [D]\end{bmatrix}$$

Version History

Introduced before R2006a

See Also

`z2h` | `z2s` | `z2y` | `abcd2z`

z2h

Convert Z-parameters to hybrid h-parameters

Syntax

```
h_params = z2h(z_params)
```

Description

`h_params = z2h(z_params)` converts the Z-parameters to hybrid h-parameters.

Examples

Convert Z-Parameters to Hybrid h-Parameters

Define a matrix of Z-parameters.

```
Z11 = -14567.2412789287 - 148373.315116592i;
Z12 = -14588.1106171651 - 148388.583516562i;
Z21 = -14528.0522132692 - 148350.705757767i;
Z22 = -14548.5996561832 - 148363.457002006i;
z_params = [Z11,Z12; Z21,Z22];
```

Convert the Z-parameters to hybrid h-parameters.

```
h_params = z2h(z_params)
```

```
h_params = 2×2 complex

   0.3148 + 2.5198i   1.0002 - 0.0002i
  -0.9999 - 0.0001i  -0.0000 + 0.0000i
```

Input Arguments

z_params — 2-port Z-parameters

2-by-2-by-*M* array of complex numbers

2-port Z-parameters, specified as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port Z-parameters.

Output Arguments

h_params — 2-port hybrid h-parameters

2-by-2-by-*M* array of complex numbers

2-port hybrid h-parameters, returned as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port hybrid h-parameters.

Version History

Introduced before R2006a

See Also

z2abcd | z2s | z2y | h2z

z2s

Convert Z-parameters to S-parameters

Syntax

```
s_params = z2s(z_params,z0)
```

Description

`s_params = z2s(z_params,z0)` converts the Z-parameters to the S-parameters.

Examples

Convert Z-Parameters to S-Parameters

Define a matrix of Z-parameters.

```
Z11 = -14567.2412789287 - 148373.315116592i;
Z12 = -14588.1106171651 - 148388.583516562i;
Z21 = -14528.0522132692 - 148350.705757767i;
Z22 = -14548.5996561832 - 148363.457002006i;
z_params = [Z11,Z12; Z21,Z22];
```

Convert Z-parameters to S-parameters.

```
s_params = z2s(z_params)
```

```
s_params = 2×2 complex

   0.0038 + 0.0248i   0.9964 - 0.0254i
   0.9961 - 0.0250i   0.0037 + 0.0249i
```

Input Arguments

`z_params` — *N*-port Z-parameters

N-by-*N*-by-*M* array of complex numbers

N-port Z-parameters, specified as an *N*-by-*N*-by-*M* array of complex numbers, where *M* represents the number of frequency points of *N*-port Z-parameters.

`z0` — Reference impedance

`50` (default) | positive real scalar | vector

Reference impedance of *N*-port S-parameters, specified as a positive real scalar in ohms.

Note `z0` must be a positive real scalar or vector. If `z0` is a vector, then the vector must be equal to the number of network parameter data points or frequency vector.

Output Arguments

`s_params` — *N*-port hybrid S-Parameters
N-by-*N*-by-*M* array of complex numbers

N-port S-parameters, returned as a *N*-by-*N*-by-*M* array of complex numbers, where *M* representing number of frequency points of *N*-port S-parameters.

Version History

Introduced before R2006a

See Also

`z2abcd` | `z2h` | `z2y` | `s2z`

z2y

Convert Z-parameters to Y-parameters

Syntax

```
y_params = z2y(z_params)
```

Description

`y_params = z2y(z_params)` converts Z-parameters to Y-parameters.

Examples

Convert Z-Parameters to Y-Parameters

Define a matrix of Z-parameters.

```
Z11 = -14567.2412789287 - 148373.315116592i;
Z12 = -14588.1106171651 - 148388.583516562i;
Z21 = -14528.0522132692 - 148350.705757767i;
Z22 = -14548.5996561832 - 148363.457002006i;
z_params = [Z11,Z12; Z21,Z22];
```

Convert the Z-parameters to Y-parameters.

```
y_params = z2y(z_params)
```

```
y_params = 2×2 complex

   0.0488 - 0.3908i  -0.0487 + 0.3909i
  -0.0489 + 0.3907i   0.0488 - 0.3908i
```

Input Arguments

z_params — Impedance parameters

N-by-*N*-by-*M* complex array

Impedance parameters, specified as an *N*-by-*N*-by-*M* complex array, where *M* represents the number of frequency points of an *N*-port Z-parameter.

Output Arguments

y_params — Admittance parameters

N-by-*N*-by-*M* complex array

Admittance parameters, returned as an *N*-by-*N*-by-*M* complex array, where *M* represents the number of frequency points of an *N*-port Y-parameter.

Version History

Introduced before R2006a

See Also

z2abcd | z2h | z2s | y2z

y2s

Convert Y-parameters to S-parameters

Syntax

```
s_params = y2s(y_params,z0)
```

Description

`s_params = y2s(y_params,z0)` converts Y-parameters to S-parameters.

Examples

Convert Y-Parameters to S-Parameters

Define a matrix of Y-parameters.

```
Y11 =  0.0488133074245012 - 0.390764155450191i;
Y12 = -0.0488588365420561 + 0.390719345880018i;
Y21 = -0.0487261119282660 + 0.390851884427087i;
Y22 =  0.0487710062903760 - 0.390800401433241i;
y_params = [Y11,Y12; Y21,Y22];
```

Convert Y-parameters to S-parameters.

```
s_params = y2s(y_params)
```

```
s_params = 2×2 complex

   0.0038 + 0.0248i   0.9961 - 0.0250i
   0.9964 - 0.0254i   0.0037 + 0.0249i
```

Input Arguments

`y_params` — *N*-port Y-Parameters

N-by-*N*-by-*M* array

N-port Y-Parameters, specified as an *N*-by-*N*-by-*M* array, where *M* represents the number of frequency points of *N*-port Y-parameters.

`z0` — Reference impedance

`50` (default) | positive real scalar

Reference impedance of *N*-port S-parameters, specified as a positive real scalar in ohms.

Note `z0` must be a positive real scalar or vector. If `z0` is a vector, then the vector must be equal to the number of network parameter data points or frequency vector.

Output Arguments

`s_params` — *N*-port- S-Parameters
complex *N*-by-*N M* array

N-port- S-parameters, returned as a complex *N*-by-*N M* array, where *M* represents the number of frequency points of *N*-port S-parameters.

Version History

Introduced before R2006a

See Also

`y2abcd` | `y2z` | `y2h` | `s2y`

y2z

Convert Y-parameters to Z-parameters

Syntax

```
z_params = y2z(y_params)
```

Description

`z_params = y2z(y_params)` converts the Y-parameters to Z-parameters.

Examples

Convert Y-parameters to Z-parameters

Define a matrix of Y-parameters.

```
Y11 =  0.0488133074245012 - 0.390764155450191i;
Y12 = -0.0488588365420561 + 0.390719345880018i;
Y21 = -0.0487261119282660 + 0.390851884427087i;
Y22 =  0.0487710062903760 - 0.390800401433241i;
y_params = [Y11,Y12; Y21,Y22];
```

Convert the Y-parameters to Z-parameters.

```
z_params = y2z(y_params)
```

```
z_params = 2×2 complex
10^5 ×

  -0.1457 - 1.4837i  -0.1453 - 1.4835i
  -0.1459 - 1.4839i  -0.1455 - 1.4836i
```

Input Arguments

y_params — Admittance parameters

N-by-*N*-by-*M* complex array

Admittance parameters, specified as an *N*-by-*N*-by-*M* complex array, where *M* represents the number of frequency points of *N*-port Y-parameters.

Output Arguments

z_params — Impedance parameters

N-by-*N*-by-*M* complex array

Impedance parameters, returned as an *N*-by-*N*-by-*M* complex array, where *M* represents the number of frequency points of *N*-port Z-parameters.

Version History

Introduced before R2006a

See Also

y2abcd | y2s | y2h | z2y

h2g

Convert hybrid h-parameters to g-parameters

Syntax

```
g_params = h2g(h_params)
```

Description

`g_params = h2g(h_params)` converts the hybrid parameters to the hybrid g-parameters.

Examples

Convert h-Parameters to g-Parameters

Define a matrix of h-parameters.

```
h_11 =      0.314441556185771 +      2.519609410005 98i;
h_12 =      0.999823389146385 -  0.000246785162909241i;
h_21 =     -1.000115600382660 -  0.000129304649930592i;
h_22 = -6.5538951551 2306e-007 + 6.67541048071651e-006i;
h_params = [h_11,h_12; h_21,h_22];
```

Convert h-parameters to g-parameters.

```
g_params = h2g(h_params)
```

```
g_params = 2×2 complex

  -0.0000 + 0.0000i  -0.9999 + 0.0001i
   1.0002 + 0.0002i   0.3142 + 2.5198i
```

Input Arguments

h_params — 2-port hybrid h-parameters

2-by-2-by-*M* array of complex numbers

2-port hybrid h-parameters, specified as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port hybrid h-parameters.

Output Arguments

g_params — 2-port inverse hybrid or g-parameters

2-by-2-by-*M* array of complex numbers

2-port inverse hybrid or g-parameters, returned as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port inverse hybrid or g-parameters.

Version History

Introduced before R2006a

See Also

g2h | h2abcd | h2s | h2y | h2z

h2s

Convert hybrid h-parameters to S-parameters

Syntax

```
s_params = h2s(h_params,z0)
```

Description

`s_params = h2s(h_params,z0)` converts the hybrid parameters to the scattering parameters.

Examples

Convert H-Parameters to S-Parameters

Define a matrix of h-parameters.

```
h_11 =      0.314441556185771 +      2.51960941000598i;
h_12 =      0.999823389146385 -  0.000246785162909241i;
h_21 =     -1.000115600382660 -  0.000129304649930592i;
h_22 = -6.55389515512306e-007 + 6.67541048071651e-006i;
h_params = [h_11,h_12; h_21,h_22];
```

Convert h-parameters to S-parameters.

```
s_params = h2s(h_params)
```

```
s_params = 2×2 complex

   0.0037 + 0.0248i   0.9961 - 0.0254i
   0.9964 - 0.0250i   0.0038 + 0.0249i
```

Input Arguments

`h_params` — 2-port hybrid h-parameters

2-by-2-by-*M* array of complex numbers

2-port hybrid h-parameters, specified as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port hybrid h-parameters.

`z0` — Reference impedance

`50` (default) | positive real scalar

Reference impedance of 2-port S-parameters, specified as a positive real scalar in ohms.

Note `z0` can be a positive real scalar or vector. If `z0` is a vector, then the vector must be equal to the number of network parameter data points.

Output Arguments

`s_params` — 2-port hybrid S-Parameters
2-by-2-by-*M* array of complex numbers

2-port S-parameters, returned as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port S-Parameters.

Version History

Introduced before R2006a

See Also

`abcd2s` | `h2abcd` | `h2y` | `h2z` | `y2s` | `y2z`

h2y

Convert hybrid h-parameters to Y-parameters

Syntax

```
y_params = h2y(h_params)
```

Description

`y_params = h2y(h_params)` converts the hybrid parameters to the admittance parameters.

Examples

Convert h-Parameters to Y-Parameters

Define a matrix of h-parameters.

```
h_11 =      0.314441556185771 +      2.519609410005 98i;
h_12 =      0.999823389146385 -  0.000246785162909241i;
h_21 =     -1.000115600382660 -  0.000129304649930592i;
h_22 = -6.5538951551 2306e-007 + 6.67541048071651e-006i;
h_params = [h_11,h_12; h_21,h_22];
```

Convert h-parameters to Y-parameters.

```
y_params = h2y(h_params)
```

```
y_params = 2×2 complex

   0.0488 - 0.3908i  -0.0487 + 0.3907i
  -0.0488 + 0.3908i   0.0487 - 0.3908i
```

Input Arguments

h_params — 2-port hybrid h-parameters

2-by-2-by-*M* array of complex numbers

2-port hybrid h-parameters, specified as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port hybrid h-parameters.

Output Arguments

y_params — 2-port Y-Parameters

2-by-2-by-*M* array of complex numbers

2-port Y-Parameters, returned as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port Y-parameters.

Version History

Introduced before R2006a

See Also

h2g | h2s | h2z | h2abcd | y2h

h2z

Convert hybrid h-parameters to Z-parameters

Syntax

```
z_params = h2z(h_params)
```

Description

`z_params = h2z(h_params)` converts the hybrid parameters to the impedance parameters.

Examples

Convert Hybrid h-Parameters to Z-Parameters

Define a matrix of h-parameters.

```
h_11 =      0.314441556185771 +      2.51960941000598i;
h_12 =      0.999823389146385 -  0.000246785162909241i;
h_21 =     -1.000115600382660 -  0.000129304649930592i;
h_22 = -6.5538951551230 6e-007 + 6.67541048071651e-006i;
h_params = [h_11,h_12; h_21,h_22];
```

Convert hybrid h-parameters to Z-parameters.

```
z_params = h2z(h_params)
```

```
z_params = 2×2 complex
10^5 ×

  -0.1458 - 1.4836i  -0.1460 - 1.4834i
  -0.1455 - 1.4839i  -0.1457 - 1.4837i
```

Input Arguments

h_params — 2-port hybrid h-parameters

2-by-2-by-*M* array of complex numbers

2-port hybrid h-parameters, specified as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port hybrid h-parameters.

Output Arguments

z_params — 2-port Z-parameters

2-by-2-by-*M* array of complex numbers

2-port Z-parameters, returned as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port Z-parameters.

Alternatives

You can also use network parameter objects to perform network parameter conversions. For more information, see “RF Network Parameter Objects”.

Version History

Introduced before R2006a

See Also

`h2g` | `h2s` | `h2y` | `h2abcd` | `z2h`

h2abcd

Convert hybrid h-parameters to ABCD-parameters

Syntax

```
abcd_params = h2abcd(h_params)
```

Description

`abcd_params = h2abcd(h_params)` converts the hybrid parameters to the ABCD-parameters.

Examples

Convert H-Parameters to ABCD-Parameters

Define a matrix of h-parameters.

```
h_11 =      0.314441556185771 +      2.51960941000598i;
h_12 =      0.999823389146385 -  0.000246785162909241i;
h_21 =     -1.000115600382660 -  0.000129304649930592i;
h_22 = -6.55389515512306e-007 + 6.67541048071651e-006i;
h_params = [h_11,h_12; h_21,h_22];
```

Convert h-parameters ABCD-parameters.

```
abcd_params = h2abcd(h_params)
```

```
abcd_params = 2×2 complex

   0.9998 - 0.0002i   0.3147 + 2.5193i
  -0.0000 + 0.0000i   0.9999 - 0.0001i
```

Input Arguments

h_params — 2-port hybrid h-parameters

2-by-2-by-*M* array of complex numbers

2-port hybrid h-parameters, specified as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port hybrid h-parameters.

Output Arguments

abcd_params — 2-port ABCD parameters

2-by-2-by-*M* array of complex numbers

2-port ABCD parameters, returned as a complex 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port ABCD-parameters. The output ABCD-parameters matrices have distinct *A*, *B*, *C*, and *D* submatrices:

$$\begin{bmatrix} [A] & [B] \\ [C] & [D] \end{bmatrix}$$

Version History

Introduced before R2006a

See Also

abcd2h | h2s | h2y | h2z | s2abcd | y2abcd | z2abcd

g2h

Convert g-parameters to hybrid h-parameters

Syntax

```
h_params = g2h(g_params)
```

Description

`h_params = g2h(g_params)` converts the g-parameters to the hybrid h-parameters.

Examples

Convert g-Parameters to h-Parameters

Define a matrix of g-parameters.

```
g_11 = -6.5538951551230 6e-007 + 6.67541048071651e-006i;
g_12 =     -0.999823389146385 -  0.000246785162909241i;
g_21 =       1.00011560038266 -  0.000129304649930592i;
g_22 =      0.314441556185771 +      2.51960941000598i;
g_params = [g_11,g_12; g_21,g_22];
```

Convert g-parameters to h-parameters.

```
h_params = g2h(g_params)
```

```
h_params = 2×2 complex

   0.3148 + 2.5198i   0.9999 + 0.0001i
  -1.0002 + 0.0002i  -0.0000 + 0.0000i
```

Input Arguments

g_params — 2-port inverse hybrid or g-parameters

2-by-2-by-*M* array of complex numbers

2-port inverse hybrid or g-parameters, specified as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port inverse hybrid or g-parameters.

Output Arguments

h_params — 2-port hybrid h-parameters

2-by-2-by-*M* array of complex numbers

2-port hybrid h-parameters, returned as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port hybrid h-parameters.

Version History

Introduced before R2006a

See Also

h2g

t2s

Convert T-parameters to S-parameters

Syntax

```
s_params = t2s(t_params)
```

Description

`s_params = t2s(t_params)` converts the chain scattering parameters to the scattering parameters.

Examples

Convert T-Parameters to S-Parameters

Define a matrix of T-parameters.

```
t11 =  0.138451095405929   - 0.230421317393041i;
t21 = -0.0451985986689165  + 0.157626245839348i;
t12 =  0.0353675449261375  + 0.115682026931012i;
t22 = -0.00194567217559662 - 0.0291212122613417i;
t_params = [t11 t12; t21 t22];
```

Convert T-parameters to S-parameters.

```
s_params = t2s(t_params)
```

```
s_params = 2×2 complex

  -0.5892 + 0.1579i   0.0372 + 0.0335i
   1.9159 + 3.1887i   0.3011 - 0.3344i
```

Input Arguments

`t_params` — 2-port T-Parameters

2-by-2-by-*M* array

2-port T-Parameters, specified as a complex 2-by-2-by-*M* array, where *M* represents the number of frequency points of 2-port T-parameters.

This function defines the T-parameters as

$$\begin{bmatrix} a_1 \\ b_1 \end{bmatrix} = \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix} \begin{bmatrix} b_2 \\ a_2 \end{bmatrix},$$

where:

- a_1 is the incident wave at the first port.
- b_1 is the reflected wave at the first port.
- a_2 is the incident wave at the second port.
- b_2 is the reflected wave at the second port.

Output Arguments

`s_params` — 2-port- S-Parameters

2-by-2-by-*M* array of complex numbers

2-port S-parameters, returned as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port S-Parameters.

Version History

Introduced before R2006a

References

[1] Gonzalez, Guillermo, *Microwave Transistor Amplifiers: Analysis and Design*, 2nd edition. Prentice-Hall, 1997, p. 25.

See Also

`s2t`

y2h

Convert Y-parameters to hybrid h-parameters

Syntax

```
h_params = y2h(y_params)
```

Description

`h_params = y2h(y_params)` converts the Y-parameters to hybrid h-parameters.

Examples

Convert Y-Parameters to h-Parameters

Define a matrix of Y-parameters.

```
Y11 =  0.0488133074245012 - 0.390764155450191i;
Y12 = -0.0488588365420561 + 0.390719345880018i;
Y21 = -0.0487261119282660 + 0.390851884427087i;
Y22 =  0.0487710062903760 - 0.390800401433241i;
y_params = [Y11,Y12; Y21,Y22];
```

Convert Y-parameters to h-parameters.

```
h_params = y2h(y_params)
```

```
h_params = 2×2 complex

   0.3148 + 2.5198i   0.9999 + 0.0001i
  -1.0002 + 0.0002i  -0.0000 + 0.0000i
```

Input Arguments

y_params — 2-port Y-Parameters

2-by-2-by-*M* array of complex numbers

2-port Y-Parameters, specified as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port Y-parameters.

Output Arguments

h_params — 2-port hybrid h-parameters

2-by-2-by-*M* array of complex numbers

2-port hybrid h-parameters, returned as a 2-by-2-by-*M* array of complex numbers, where *M* represents the number of frequency points of 2-port hybrid h-parameters.

Version History

Introduced before R2006a

See Also

y2abcd | y2s | y2z | h2y

y2abcd

Convert Y-parameters to ABCD-parameters

Syntax

```
abcd_params = y2abcd(y_params)
```

Description

`abcd_params = y2abcd(y_params)` converts Y-parameters to ABCD-parameters.

Examples

Convert Y-Parameters to ABCD-Parameters

Define a matrix of Y-parameters.

```
Y11 =  0.0488133074245012 - 0.390764155450191i;
Y12 = -0.0488588365420561 + 0.390719345880018i;
Y21 = -0.0487261119282660 + 0.390851884427087i;
Y22 =  0.0487710062903760 - 0.390800401433241i;
y_params = [Y11,Y12; Y21,Y22];
```

Convert Y-parameters to ABCD-parameters

```
abcd_params = y2abcd(y_params)
```

```
abcd_params = 2×2 complex

   0.9999 + 0.0001i   0.3141 + 2.5194i
  -0.0000 + 0.0000i   0.9998 + 0.0002i
```

Input Arguments

y_params — 2*N*-port Y-Parameters

2*N*-by-2*N*-by-*M* array

2*N*-port Y-Parameters, specified as an 2*N*-by-2*N*-by-*M* array, where *M* represents the number of frequency points of 2*N*-port Y-parameters.

Output Arguments

abcd_params — 2*N*-port- ABCD-parameters

complex 2*N*-by-2*N*-by-*M* array

2*N*-port- ABCD-parameters, specified as a complex 2*N*-by-2*N*-by-*M* array, where *M* represents the number of frequency points of 2*N*-port ABCD-parameters.

The function assumes that the ABCD-parameter matrices have distinct A, B, C, and D submatrices:

$$\begin{bmatrix}[A] & [B]\\ [C] & [D]\end{bmatrix}$$

Version History

Introduced before R2006a

See Also

y2s | y2z | y2h | abcd2y

copy

Copy circuit or data object

Syntax

```
h2 = copy(h)
```

Description

`h2 = copy(h)` returns a copy of the circuit, data, or network parameter object `h`.

Note The syntax `h2 = h` copies only the object handle and does not create an object.

Examples

Copy Circuit Object

Copy a circuit object to represent the cascaded amplifier.

```
FirstCkt = rfckt.txline;
SecondCkt = rfckt.amplifier;
ThirdCkt = copy(FirstCkt);
```

Create a cascaded circuit object.

```
CascadedCkt = rfckt.cascade('Ckts', {FirstCkt,SecondCkt,ThirdCkt})
```

```
CascadedCkt =
   rfckt.cascade with properties:

              Ckts: {1x3 cell}
             nPort: 2
    AnalyzedResult: []
              Name: 'Cascaded Network'
```

Input arguments

h — RF object handle

`circuit object | data object | network parameter object`

RF object, specified as a circuit, data, or network parameter object.

Output arguments

h2 — RF object handle

`circuit object | data object | network parameter object`

RF object handle, returned as a circuit, data, or network parameter object.

Alternatives

The syntax `h2 = h` copies only the object handle and does not create an object.

Version History

Introduced before R2006a

See Also

`analyze`

Topics

“Write S2P Touchstone Files”

gammaml

Calculate load reflection coefficient of two-port network

Syntax

```
coefficient = gammaml(s_params)
coefficient = gammaml(hs)
```

Description

`coefficient = gammaml(s_params)` calculates the load reflection coefficient of a two-port network required for simultaneous conjugate match.

`coefficient = gammaml(hs)` calculates the load reflection coefficient of the two-port network represented by the S-parameter object `hs`.

Examples

Load Reflection Coefficient Calculation

Calculate the load reflection coefficient using network data from a file

```
ckt = read(rfckt.amplifier,'default.s2p');
s_params = ckt.NetworkData.Data;
coefficient = gammaml(s_params);
```

Calculate Load Reflection Coefficient of S-Parameters Object

Define S-parameters object specified from a file.

```
s_params = sparameters('default.s2p');
```

Calculate the load reflection coefficient using the `gammaml` function.

```
coefficient = gammaml(s_params)
```

```
coefficient = 191×1 complex

  -0.0741 + 0.3216i
  -0.0751 + 0.3292i
  -0.0763 + 0.3365i
  -0.0776 + 0.3435i
  -0.0791 + 0.3502i
  -0.0807 + 0.3564i
  -0.0825 + 0.3619i
  -0.0843 + 0.3668i
  -0.0862 + 0.3709i
  -0.0882 + 0.3741i
```

Input Arguments

`s_params` — Two-port S-parameters

complex 2-by-2-by-*M* array

Two-port S-parameters, specified as a complex 2-by-2-by-*M* array. *M* is the number of two-port S-parameters.

Data Types: `double`

`hs` — Two-port network

S-parameter object

Two-port network, specified as an S-parameter object.

Data Types: `function_handle`

Output Arguments

`coefficient` — Load reflection coefficient

M element complex vector

Load reflection coefficient, returned as a *M* element complex vector.

Algorithms

The function calculates `coefficient` using the equation

$$\Gamma_{ML} = \frac{B_2 \pm \sqrt{B_2{}^2 - 4|C_2{}^2|}}{2C_2}$$

where

$$B_2 = 1 - |S_{11}{}^2| + |S_{22}{}^2| - \left|\Delta^2\right|$$
$$C_2 = S_{22} - \Delta \cdot S_{11}^*$$
$$\Delta = S_{11}S_{22} - S_{12}S_{21}$$

Version History

Introduced before R2006a

See Also

`gammams`

gammams

Calculate source reflection coefficient of two-port network

Syntax

```
coefficient = gammams(s_params)
coefficient = gammams(hs)
```

Description

`coefficient = gammams(s_params)` calculates the source reflection coefficient of a two-port network required for simultaneous conjugate match.

`coefficient = gammams(hs)` calculates the source reflection coefficient of the two-port network represented by the S-parameter object `hs`.

Examples

Source Reflection Coefficient Calculation

Calculate the source reflection coefficient using network data from a file.

```
ckt = read(rfckt.amplifier,'default.s2p');
s_params = ckt.NetworkData.Data;
coefficient = gammams(s_params)
```

```
coefficient = 191×1 complex

  -0.7247 + 0.4813i
  -0.7324 + 0.4723i
  -0.7401 + 0.4632i
  -0.7478 + 0.4541i
  -0.7554 + 0.4449i
  -0.7630 + 0.4357i
  -0.7704 + 0.4264i
  -0.7778 + 0.4170i
  -0.7850 + 0.4075i
  -0.7921 + 0.3980i
      ⋮
```

Source Reflection Coefficient Calculation of S-Parameters Object

Define a S-parameters object specified from a file.

```
hs = sparameters('default.s2p');
```

Calculate the source reflection coefficient using `gammams` function.

```
coefficient = gammams(hs)

coefficient = 191×1 complex

  -0.7247 + 0.4813i
  -0.7324 + 0.4723i
  -0.7401 + 0.4632i
  -0.7478 + 0.4541i
  -0.7554 + 0.4449i
  -0.7630 + 0.4357i
  -0.7704 + 0.4264i
  -0.7778 + 0.4170i
  -0.7850 + 0.4075i
  -0.7921 + 0.3980i
      ⋮
```

Input Arguments

`s_params` — Two-port S-parameters

complex 2-by-2-by-*M* array

Two-port S-parameters, specified as a complex 2-by-2-by-*M* array. *M* is the number of two-port S-parameters.

Data Types: `double`

`hs` — Two-port network

S-parameter object

Two-port network, specified as an S-parameter object.

Data Types: `function_handle`

Output Arguments

`coefficient` — Source reflection coefficient

M element complex vector

Source reflection coefficient, returned as a *M* element complex vector.

Algorithms

The function calculates `coefficient` using the equation

$$\Gamma_{MS} = \frac{B_1 \pm \sqrt{B_1{}^2 - 4|C_1{}^2|}}{2C_1}$$

where

$$B_1 = 1 + |S_{11}{}^2| - |S_{22}{}^2| - \left|\Delta^2\right|$$

$$C_1 = S_{11} - \Delta \cdot S_{22}^*$$

$$\Delta = S_{11}S_{22} - S_{12}S_{21}$$

Version History

Introduced before R2006a

See Also

gammaout

Calculate output reflection coefficient of two-port network

Syntax

```
coefficient = gammaout(s_params,z0,zs)
coefficient = gammaout(hs,zs)
```

Description

`coefficient = gammaout(s_params,z0,zs)` calculates the output reflection coefficient of a two-port network. `z0` is the reference impedance Z_0; its default value is 50 ohms. `zs` is the source impedance Z_s; its default value is also 50 ohms. `coefficient` is an M-element complex vector.

`coefficient = gammaout(hs,zs)` calculates the output reflection coefficient of the two-port network represented by the S-parameter object `hs`.

Examples

Output Reflection Coefficient Calculation using Network Data

Calculate the output reflection coefficient using network data from a file.

```
ckt = read(rfckt.amplifier,'default.s2p');
s_params = ckt.NetworkData.Data;
z0 = ckt.NetworkData.Z0;
zs = 100;
coefficient = gammaout(s_params,z0,zs)
```

```
coefficient = 191×1 complex

  -0.0741 - 0.3216i
  -0.0765 - 0.3184i
  -0.0787 - 0.3152i
  -0.0809 - 0.3121i
  -0.0829 - 0.3090i
  -0.0848 - 0.3059i
  -0.0867 - 0.3029i
  -0.0884 - 0.3000i
  -0.0900 - 0.2971i
  -0.0915 - 0.2943i
      ⋮
```

Output Reflection Coefficient Calculation of S-Parameters Object

Define a S-parameters object from a file.

```
s_params = sparameters('default.s2p');
```

Specify the source impedance.

```
zs = 100;
```

Calculate the output reflection coefficient using the `gammaout` function. .

```
coefficient = gammaout(s_params,zs)
```

```
coefficient = 191×1 complex

  -0.0741 - 0.3216i
  -0.0765 - 0.3184i
  -0.0787 - 0.3152i
  -0.0809 - 0.3121i
  -0.0829 - 0.3090i
  -0.0848 - 0.3059i
  -0.0867 - 0.3029i
  -0.0884 - 0.3000i
  -0.0900 - 0.2971i
  -0.0915 - 0.2943i
      ⋮
```

Input Arguments

`s_params` — Two-port S-parameters

complex 2-by-2-by-*M* array

Two-port S-parameters, specified as a complex 2-by-2-by-*M* array. *M* is the number of two-port S-parameters.

Data Types: `double`

`z0` — Reference impedance

`50` (default) | positive scalar

Reference impedance, specified as a positive scalar.

Data Types: `double`

`zs` — Source impedance

`50` (default) | positive scalar

Source impedance, specified as a positive scalar.

Data Types: `double`

`hs` — Two-port network

S-parameter object

Two-port network, specified as an S-parameter object.

Data Types: `function_handle`

Output Arguments

`coefficient` — Output reflection coefficient
M element complex vector

Output reflection coefficient, returned as a *M* element complex vector.

Algorithms

The function calculates `coefficient` using the equation

$$\Gamma_{out} = S_{22} + \frac{S_{12}S_{21}\Gamma_S}{1 - S_{11}\Gamma_S}$$

where

$$\Gamma_S = \frac{Z_s - Z_0}{Z_s + Z_0}$$

Version History

Introduced before R2006a

See Also

`gammain`

getdata

Data object containing analyzed result of specified circuit object

Syntax

```
hd = getdata(h)
```

Description

`hd = getdata(h)` returns a handle, `hd`, to the `rfdata.data` object containing the analysis data, if any, for circuit (`rfckt`) object `h`. If there is no analysis data, `getdata` displays an error message.

Examples

Analyze Result Using getdata Function

Create a circuit object, `rfckt.amplifier` object.

```
h = rfckt.amplifier;
```

Use the `getdata` function to create a data object containing analyzed result of specified circuit object.

```
hd = getdata(h)
```

```
hd =
   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: 'Linear'
             Name: 'Data object'
```

Input Arguments

h — Circuit object

`rfckt` object

Circuit object, specified as an `rfckt` object.

Data Types: `double`

Output Arguments

hd — Analysis data
`rfdata.data`

Analysis data, returned as a `rfdata.data` object.

Version History

Introduced before R2006a

See Also

z2gamma

Convert impedance to reflection coefficient

Syntax

```
gamma = z2gamma(z)
gamma = z2gamma(z,z0)
```

Description

`gamma = z2gamma(z)` converts the impedance `z` to the reflection coefficient `gamma` using a reference impedance of 50 ohms.

`gamma = z2gamma(z,z0)` converts the impedance `z` to the reflection coefficient `gamma` using a reference impedance of `z0` ohms.

Examples

Impedance to Reflection Coefficient

Convert an impedance of 100 ohms into a reflection coefficient, using a 50-ohm reference impedance

```
z = 100;
gamma = z2gamma(z)
```

```
gamma = 0.3333
```

Input Arguments

z — Impedance value

positive scalar

Impedance value, specified as a positive scalar.

Data Types: `double`

z0 — Reference impedance

`50` (default) | positive scalar

Reference impedance, specified as a positive scalar.

Data Types: `double`

Output Arguments

gamma — Reflection coefficient

M element complex vector

Reflection coefficient, returned as a *M* element complex vector.

Algorithms

`z2gamma` calculates the coefficient using the equation

$$\Gamma = \frac{Z - Z_0}{Z + Z_0}$$

Version History

Introduced in R2008a

See Also

`z2abcd` | `z2h` | `z2s` | `z2y`

vswr

VSWR at given reflection coefficient Γ

Syntax

```
ratio = vswr(gamma)
```

Description

`ratio = vswr(gamma)` calculates the voltage standing-wave ratio $VSWR$ at the given reflection coefficientΓ as

$$VSWR = \frac{1 + |\Gamma|}{1 - |\Gamma|}$$

Examples

VSWR from Reflection Coefficient

Calculate the VSWR for a given reflection coefficient.

```
gamma = 1/3;
ratio = vswr(gamma)
```

```
ratio = 2.0000
```

Input Arguments

`gamma` — Reflection coefficient

M element complex vector

Reflection coefficient, returned as a *M* element complex vector.

Data Types: `double`

Output Arguments

`ratio` — Voltage standing wave ratio

real vector

Voltage standing wave ratio, returned as a real vector.

Version History

Introduced before R2006a

See Also

gamma2z | gammain | gammaout

stabilityk

Stability factor *K* of two-port network

Syntax

```
[k,b1,b2,delta] = stabilityk(s_params)
[k,b1,b2,delta] = stabilityk(hs)
```

Description

`[k,b1,b2,delta] = stabilityk(s_params)` calculates and returns the stability factor, `k`, and the conditions `b1`, `b2`, and `delta` for the two-port network.

`[k,b1,b2,delta] = stabilityk(hs)` calculates and returns the stability factor and stability conditions for the two-port network represented by the S-parameter object `hs`.

Examples

Stability of Network

Examine the stability of network data from a file by first calculating the stability factor.

```
S = sparameters('passive.s2p');
s_params = S.Parameters;
[k,b1,b2,delta] = stabilityk(s_params);
```

Check stability criteria.

```
stability_index = (k>1)&(abs(delta)<1);
is_stable = all(stability_index)
```

```
is_stable = logical
   1
```

List frequencies with unstable S-parameters.

```
freq = S.Frequencies;
freq_unstable = freq(~stability_index)
```

```
freq_unstable =

  0x1 empty double column vector
```

Examine Stability of Network Using S-Parameters Object

Create a S-parameters object from a specified file. .

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

Calculate the stability using `stabilityk` function.

```
[k,b1,b2,delta] = stabilityk(s_params);
```

Check the stability criteria.

```
stability_index = (k>1)&(abs(delta)<1);
is_stable = all(stability_index)
```

```
is_stable = logical
   1
```

List frequencies with unstable S-parameters.

```
freq = s_params.Frequencies;
freq_unstable = freq(~stability_index)
```

```
freq_unstable =

  0x1 empty double column vector
```

Input Arguments

`s_params` — Two-port S-parameters

complex 2-by-2-by-*M* array

Two-port S-parameters, specified as a complex 2-by-2-by-*M* array. *M* is the number of two-port S-parameters.

Data Types: `double`

`hs` — Two-port network

S-parameter object

Two-port network, specified as an S-parameter object.

Data Types: `function_handle`

Algorithms

Necessary and sufficient conditions for stability are `k>1` and `abs(delta)<1`. `stabilityk` calculates the outputs using the equations

$$K = \frac{1 - |S_{11}|^2 - |S_{22}|^2 + |\Delta|^2}{2|S_{12}S_{21}|}$$

$$B_1 = 1 + |S_{11}|^2 - |S_{22}|^2 - |\Delta|^2$$

$$B_2 = 1 - |S_{11}|^2 + |S_{22}|^2 - |\Delta|^2$$

where:

- S_{11}, S_{12}, S_{21}, and S_{22} are S-parameters from the input argument `s_params`.
- Δ is a vector whose members are the determinants of the *M* 2-port S-parameter matrices:

$$\Delta = S_{11}S_{22} - S_{12}S_{21}$$

The function performs these calculations element-wise for each of the M S-parameter matrices in `s_params`.

Version History

Introduced before R2006a

References

[1] Gonzalez, Guillermo. Microwave Transistor Amplifiers: Analysis and Design. 2nd ed. Upper Saddle River, N.J: Prentice Hall, 1997.

See Also

`stabilitymu`

stabilitymu

Stability factor μ of two-port network

Syntax

```
[mu,muprime] = stabilitymu(s_params)
[mu,muprime] = stabilitymu(hs)
```

Description

`[mu,muprime] = stabilitymu(s_params)` calculates and returns the stability factor, μ, and μ' for the two-port S-parameters

`[mu,muprime] = stabilitymu(hs)` calculates and returns the stability factor for the two-port network represented by the S-parameter object `hs`.

Examples

Stability Factor of Two-Port Network

Calculate the stability factor of network data from a file.

```
S = sparameters('passive.s2p');
s_params = S.Parameters;
[mu,muprime] = stabilitymu(s_params);
```

Check stability criteria.

```
stability_index = (mu>1)|(muprime>1);
is_stable = all(stability_index)
```

```
is_stable = logical
   1
```

List frequencies with unstable S-parameters.

```
freq = S.Frequencies;
freq_unstable = freq(~stability_index)
```

```
freq_unstable =

  0x1 empty double column vector
```

Examine Stability Factor Using S-Parameter Object

Create a `sparameters` object from the specified file.

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

Calculate the stability factor using `stabilitymu` function.

```
[mu,muprime] = stabilitymu(s_params);
```

Check the stability criteria.

```
stability_index = (mu>1)|(muprime>1);
is_stable = all(stability_index)
```

```
is_stable = logical
   1
```

List frequencies with unstable S-parameters.

```
freq = s_params.Frequencies;
freq_unstable = freq(~stability_index)
```

```
freq_unstable =

  0x1 empty double column vector
```

Input Arguments

`s_params` — Two-port S-parameters

complex 2-by-2-by-*M* array

Two-port S-parameters, specified as a complex 2-by-2-by-*M* array. *M* is the number of two-port S-parameters.

Data Types: `double`

`hs` — Two-port network

S-parameter object

Two-port network, specified as an S-parameter object.

Data Types: `function_handle`

Output Arguments

`mu` — Minimum distance between center of unit Smith chart and unstable region in load plane

vector

Minimum distance between the center of the unit Smith chart and the unstable region in the load plane, returned as vector equal to the number of frequency or data points.

`muprime` — Minimum distance between center of unit Smith chart and unstable region in source plane

vector

Minimum distance between the center of the unit Smith chart and the unstable region in the source plane, returned as a vector equal to the number of frequency or data points.

Algorithms

`stabilitymu` calculates the stability factors using the equations

$$\mu = \frac{1 - |S_{11}|^2}{|S_{22} - S_{11}^* \Delta| + |S_{21} S_{12}|}$$

$$\mu' = \frac{1 - |S_{22}|^2}{|S_{11} - S_{22}^* \Delta| + |S_{21} S_{12}|}$$

where:

- S_{11}, S_{12}, S_{21}, and S_{22} are S-parameters, from the input argument `s_params`.
- Δ is a vector whose members are the determinants of the M 2-port S-parameter matrices:

  $$\Delta = S_{11} S_{22} - S_{12} S_{21}$$

- S^* is the complex conjugate of the corresponding S-parameter.

The function performs these calculations element-wise for each of the M S-parameter matrices in `s_params`.

Version History

Introduced before R2006a

References

[1] Edwards, M.L., and J.H. Sinsky. "A New Criterion for Linear 2-Port Stability Using a Single Geometrically Derived Parameter." *IEEE Transactions on Microwave Theory and Techniques 40*, no. 12 (December 1992): 2303–11. https://doi.org/10.1109/22.179894.

See Also

`stabilityk`

rfchain.setstage

Update RF chain stage

Syntax

```
setstage(obj,idx,g,nf,oip3val,'Name',nm)

setstage(obj,g,nf,'IIP3',ip3val,'Name',nm)

setstage(_____,Name,Value)
```

Description

`setstage(obj,idx,g,nf,oip3val,'Name',nm)` updates gain, noise figure, output-referred third-order intercept values of a stage. Use the index, `idx` of the RF chain object to specify the stage you want to update. At a time, you can change the name of only one stage.

`setstage(obj,g,nf,'IIP3',ip3val,'Name',nm)` updates the input-referred third-order intercept value of a stage.

`setstage(_____,Name,Value)` updates the values of a stage using the name-value pair arguments.

Examples

Change Noise Figure Of RF Chain Stage

Create an RF chain object.

```
g = [11 -3];
nf = [25 3];
o3 = [30 Inf];
nm = {'amp1','filt1'};
rfch = rfchain(g,nf,o3,'Name',nm);
```

Change the noise figure of **filt1** to 20 dB.

```
setstage(rfch,2,'NoiseFigure',20)
```

View results on a worksheet.

```
worksheet(rfch)
```

	amp1	filt1
Stage Gain	11	-3
Stage Noise Figure	25	20
Stage OIP3	30	Inf
Stage IIP3	19	Inf
Cascaded Gain	11	8
Cascaded Noise Figure	25	25.1067
Cascaded OIP3	30	27
Cascaded IIP3	19	19

Input Arguments

`obj` — RF chain object

`rfchain` object

RF chain object, specified as an `rfchain` object.

`idx` — Number of a stage

integer | vector of integers

Number of a stage, specified as an integer or vector of integers.

Example: 2

Data Types: `double`

`g` — Gain

0 (default) | scalar | vectors of same length

Gain of a stage, specified as a scalar or vectors of same length.

Example: -3

Data Types: `double`

`nf` — Noise figure

0 (default) | scalar | vectors of same length

Noise figure of a stage, specified as a scalar or vectors of same length.

Example: 20

Data Types: `double`

`oip3val` — Output-referred third-order intercept

inf (default) | scalar | vectors of same length

Output-referred third-order intercept of a stage, specified as a scalar or vectors of same length.

Example: 30

Data Types: `double`

Name-Value Pair Arguments

Optional comma-separated pairs of `Name,Value` pair arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`).

Example: setstage(ch,2,'NoiseFigure',20)

`Gain` — Gain
0 (default) | scalar | vectors of same length

Gain of a stage, specified as a comma-separated pair consisting of `'Gain'` and a scalar or vectors of same length. This pair updates the gain of a stage specified by `idx`.

Example: 10

Data Types: `double`

`NoiseFigure` — Noise figure
0 (default) | scalar | vectors of same length

Noise figure of a stage, specified as a comma-separated pair consisting of `'NoiseFigure'` and a scalar or vectors of same length. This pair updates the noise figure of a stage specified by `idx`.

Example: 30

Data Types: `double`

`OIP3` — Output-referred third-order intercept
inf (default) | scalar | vectors of same length

Output-referred third-order intercept of a stage, specified as a comma-separated pair consisting of `'OIP3'` and a scalar or vectors of same length. This pair updates the output-referred third-order intercept of a stage specified by `idx`.

Example: 30

Data Types: `double`

`IIP3` — Input-referred third-order intercept
inf (default) | scalar | vectors of same length

Input-referred third-order intercept of a stage, specified as a comma-separated pair consisting of `'IIP3'` and a scalar or vectors of same length. This pair updates the input-referred third-order intercept of a stage specified by `idx`.

Example: 30

Data Types: `double`

`Name` — Name of stage
character vector

Name of a stage, specified as a comma-separated pair consisting of `'Name'` and a character vector. This pair updates the name of the stage specified by `idx`.

Example: amp1

Version History

Introduced in R2014b

See Also

`rfchain.plot` | `rfchain.worksheet` | `rfchain.plot` | `rfchain.cumoip3` | `rfchain.cumiip3` | `rfchain.cumgain`

rfchain.cumgain

Cascaded gain of the RF chain object

Syntax

```
g = cumgain(obj)
```

Description

`g = cumgain(obj)` returns the cascaded gain for each stage of the RF chain object `obj`.

Examples

Calculate Cascaded Gain

Assign stage-by-stage values of gain, noise figure, OIP3 and stage names.

```
g = [11 -3 7];
nf = [25 3 5];
o3 = [30 Inf 10];
nm = {'amp1','filt1','lna1'};
```

Create an RF chain object.

```
rfch = rfchain(g,nf,o3,'Name',nm);
```

```
%Calculate cascaded gain.
gain = cumgain(rfch)
```

```
gain = 1×3

    11     8    15
```

Input Arguments

obj — RF chain object

`rfchain` object

RF chain object, specified as an `rfchain` object.

Output Arguments

g — Cascaded gain

vector

Cascaded gain of the RF chain object, returned as a vector. The vector length is equal to the number of stages in the RF chain object.

Version History

Introduced in R2014b

See Also

`rfchain.setstage` | `rfchain.worksheet` | `rfchain.plot` | `rfchain.cumiip3` | `rfchain.cumoip3` | `rfchain.cumnoisefig`

rfchain.cumnoisefig

Cascaded noise figure of the RF chain object

Syntax

```
nf = cumnoisefig(obj)
```

Description

`nf = cumnoisefig(obj)` returns the cascaded noise figure for each stage for RF chain object `obj`. The syntax first calculates the noise factor and then the noise figure. The formulae used are:

$$noisefactor(total) = noisefactor(1) + (noisefactor(2) - 1)/g1 + (noisefactor(3) - 1)/g1 + g2 + ...$$

$$noisefigure = 10*\log10(noisefactor)$$

Examples

Calculate Cascaded Noise Figure

Assign stage-by-stage values of gain, noise figure, OIP3 and stage names.

```
g = [11 -3 7];
nf = [25 3 5];
o3 = [30 Inf 10];
nm = {'amp1','filt1','lna1'};
```

Create an RF chain object.

```
rfch = rfchain(g,nf,o3,'Name',nm);
```

Calculate cascaded noise figure.

```
noisefig = cumnoisefig(rfch)
```

```
noisefig = 1×3

   25.0000   25.0011   25.0058
```

Input Arguments

`obj` — RF chain object

`rfchain` object

RF chain object, specified as an `rfchain` object.

Output Arguments

`nf` — Cascaded noise figure

vector

Cascaded noise figure for RF chain object, returned as a vector. The vector length is equal to the number of stages in the RF chain object.

Version History

Introduced in R2014b

See Also

`rfchain.setstage` | `rfchain.worksheet` | `rfchain.plot` | `rfchain.cumoip3` | `rfchain.cumiip3` | `rfchain.cumgain`

rfchain.cumoip3

Cascaded output-referred third-order intercept of the RF chain object

Syntax

```
oip3val = cumoip3(obj)
```

Description

`oip3val = cumoip3(obj)` returns the cascaded output-referred third-order intercept for each stage of the RF chain object `obj`. The oip3 is calculated using the formula:

$$oip3lin = iip3lin * gainlin$$

where all values are linear

Examples

Calculate Cascaded OIP3

Assign stage-by-stage values of gain, noise figure, OIP3 and stage names.

```
g = [11 -3 7];
nf = [25 3 5];
o3 = [30 Inf 10];
nm = {'amp1','filt1','lna1'};
```

Create an RF chain object.

```
rfch = rfchain(g,nf,o3,'Name',nm);
```

Calculate cascaded oip3 value.

```
oip3val = cumoip3(rfch)
```

```
oip3val = 1×3

   30.0000   27.0000    9.9827
```

Input Arguments

`obj` — RF chain object

`rfchain` object

RF chain object, specified as an `rfchain` object.

Output Arguments

`oip3val` — Cascaded output-referred third-order intercept
vectors

Cascaded output-referred third-order intercept for RF chain object, returned as vectors. The vector length is equal to the number of stages in the RF chain object.

Version History

Introduced in R2014b

See Also

`rfchain.setstage` | `rfchain.worksheet` | `rfchain.plot` | `rfchain.cumnoisefig` | `rfchain.cumiip3` | `rfchain.cumgain`

rfchain.cumiip3

Cascaded noise figure of the RF chain object

Syntax

```
ip3val = cumiip3(obj)
```

Description

`ip3val = cumiip3(obj)` returns the cascaded input-referred third-order intercept for each stage of the RF chain object `obj`. The input-referred third-order intercept is calculated using the formula:

$1/iip3lin(total) = 1/iip3lin(1) + g1/iip3lin(2) + (g1 * g2)/iiplin(3) + ...$

where, $iip3lin$ = iip3 (linear values)

Examples

Calculate Cascaded IIP3

Assign stage-by-stage values of gain, noise figure, IIP3 and stage names.

```
g = [11 -3 7];
nf = [25 3 5];
i3 = [19 Inf 3];
nm = {'amp1','filt1','lna1'};
```

Create an RF chain object.

```
rfch = rfchain(g,nf,'IIP3',i3,'Name',nm);
```

Calculate cascaded iip3 value.

```
iip3val = cumiip3(rfch)
```

```
iip3val = 1×3

   19.0000   19.0000   -5.0173
```

Input Arguments

`obj` — RF chain object

`rfchain` object

RF chain object, specified as an `rfchain` object.

Output Arguments

`ip3val` — Cascaded input-referred third-order intercept
vectors

Cascaded input-referred third-order intercept for RF chain object, returned as vectors. The vector length is equal to the number of stages in the RF chain object.

Version History

Introduced in R2014b

See Also

`rfchain.setstage` | `rfchain.worksheet` | `rfchain.plot` | `rfchain.cumoip3` | `rfchain.cumnoisefig` | `rfchain.cumgain`

rfchain.plot

Plot RF chain cascaded analysis results

Syntax

```
plot(obj)
```

```
h = plot(obj)
```

Description

`plot(obj)` displays a plot of the cascaded gain, noise figure, OIP3 and IIP3 values of the RF chain object `obj`.

`h = plot(obj)` returns a column vector of line series handles, where h contains one handle per plotted line.

Examples

Plot Results of RF Chain Object

Assign stage-by-stage values of gain, noise figure, OIP3 and stage names.

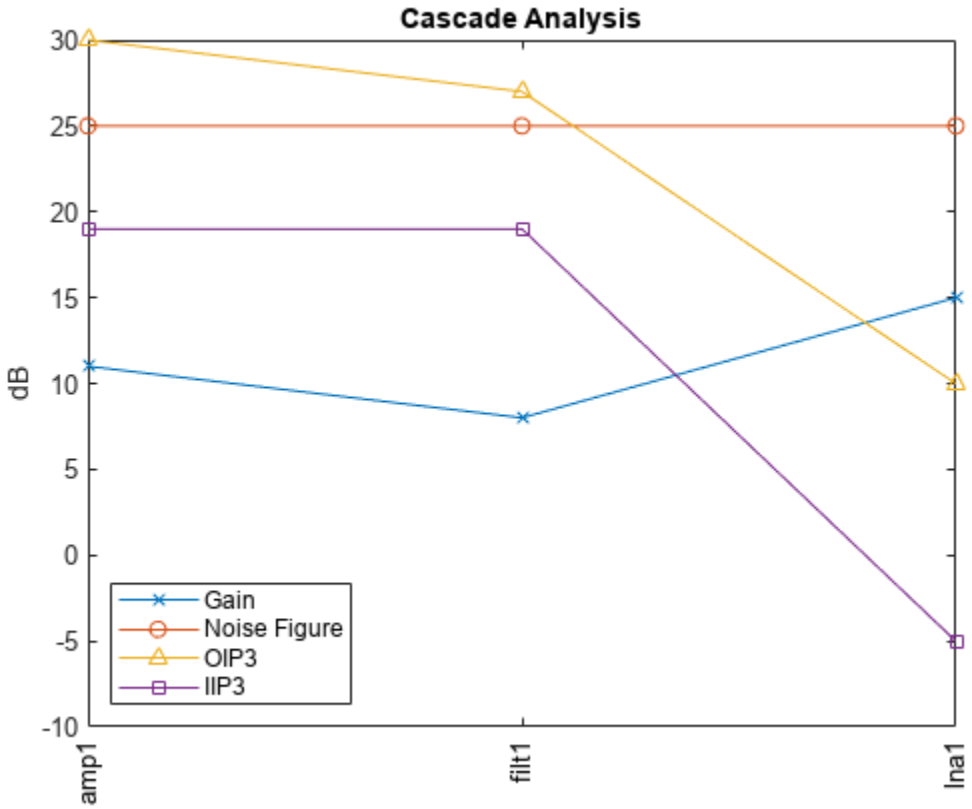
```
g = [11 -3 7];
nf = [25 3 5];
oip3 = [30 Inf 10];
nm = {'amp1','filt1','lna1'};
```

Create an RF chain object.

```
rfch = rfchain(g,nf,oip3,'Name',nm);
```

Plot the results.

```
plot(rfch)
```



Input Arguments

obj — RF chain object

`rfchain` object

RF chain object, specified as an `rfchain` object.

Output Arguments

h — line series handle

column vector

Line series handle, returned as a column vector, that contains one handle per plotted line.

Version History

Introduced in R2014b

See Also

`rfchain.setstage` | `rfchain.worksheet` | `rfchain.cumoip3` | `rfchain.cumoip3` | `rfchain.cumiip3` | `rfchain.cumgain`

rfchain.worksheet

RF chain cascaded analysis table

Syntax

```
worksheet(obj)
```

```
fig = worksheet(obj)
```

Description

`worksheet(obj)` displays a table of values for the gain, noise figure, OIP3, and IIP3 of the RF chain object `obj`. The table contains both the original input values and the calculated cascade values.

`fig = worksheet(obj)` returns a figure handle of the table.

Examples

Create RF Chain Adding Stage-By-Stage Values

Assign three stage-by-stage values of gain, noise figure, OIP3 and stage names.

```
g = [11 -3 7];
nf = [25 3 5];
o3 = [30 Inf 10];
nm = {'amp1','filt1','lna1'};
```

Create an RF chain object.

```
rfch = rfchain(g,nf,o3,'Name',nm);
```

View results in a worksheet.

```
worksheet(rfch)
```

	amp1	filt1	lna1
Stage Gain	11	-3	7
Stage Noise Figure	25	3	5
Stage OIP3	30	Inf	10
Stage IIP3	19	Inf	3
Cascaded Gain	11	8	15
Cascaded Noise Figure	25	25.0011	25.0058
Cascaded OIP3	30	27	9.9827
Cascaded IIP3	19	19	-5.0173

Input Arguments

`obj` — RF chain object
`rfchain` object

RF chain object, specified as an `rfchain` object.

Output Arguments

`fig` — figure handle
scalar handle object

Figure handle of the table, returned as a scalar handle object, that contains the properties of the RF chain object.

Version History

Introduced in R2014b

See Also

`rfchain.setstage` | `rfchain.plot` | `rfchain.cumoip3` | `rfchain.cumiip3` | `rfchain.cumgain`

smithchart

(To be removed) Plot complex vector of a reflection coefficient on Smith chart

Note `smithchart` is not recommended. Use `smithplot` instead. For more information, see "Compatibility Considerations".

Syntax

```
[lineseries,hsm] = smithchart(gamma)

hsm = smithchart
```

Description

`[lineseries,hsm] = smithchart(gamma)` plots the complex vector of a reflection coefficient `gamma` on a Smith chart. `hsm` is the handle of the Smith chart object. `lineseries` is a column vector of handles to `lineseries` objects, one handle per plotted line.

To plot network parameters from a circuit (`rfckt`) or data (`rfdata`) object on a Smith chart, use the `smith` function.

`hsm = smithchart` draws a blank Smith chart and returns the handle, `hsm`, of the Smith chart object.

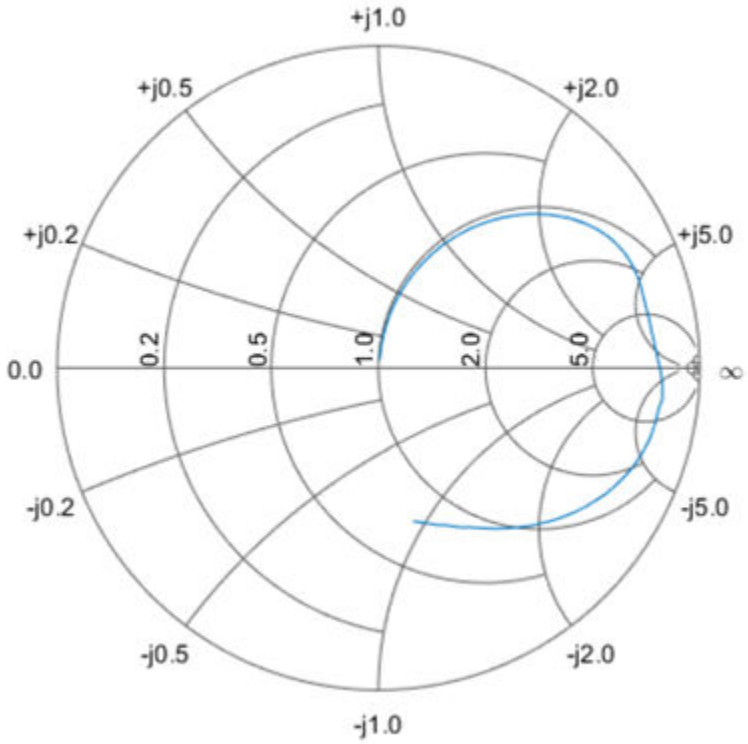
Examples

Plot Reflection and Impedance Data on Smith Chart

Input `passive.s2p` to a `sparameter` object and plot reflection, S11 on a Smith chart.

```
S = sparameters('passive.s2p');
s11 = rfparam(S,1,1);
figure
smithchart(s11)
```

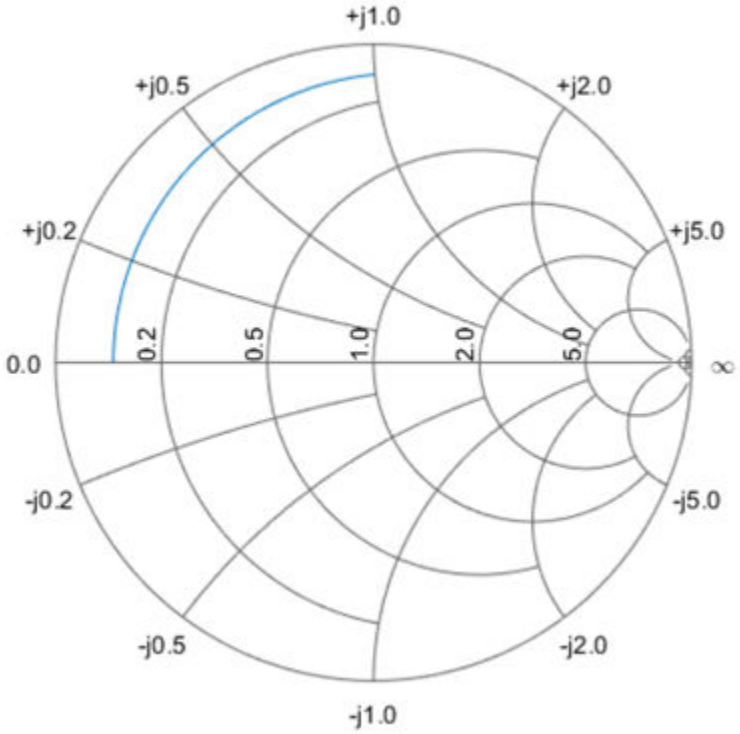
```
Warning: SMITHCHART will be removed in a future release. Use SMITHPLOT instead.
```



Convert impedance data to reflection coefficient and plot the data on a Smith chart.

```
z = 0.1*50 + 1j*(0:0.1:50);
gamma = z2gamma(z);
figure
smithchart(gamma)
```

```
Warning: SMITHCHART will be removed in a future release. Use SMITHPLOT instead.
```



Input arguments

gamma — reflection coefficient
complex vector

Reflection coefficient, specified as a complex vector.

Data Types: `double`

Output Arguments

`lineseries` — line properties handle
column vector

Line properties handle, returned as a column vector, changes the properties of the plotted lines by changing the Chart Line. There is one handle per plotted line.

`hsm` — Smith chart handle
scalar handle

Smith chart handle, specified as a scalar handle.

This table lists all properties you can specify for `smithchart` objects along with units, valid values, and descriptions of their use.

Property Name	Description	Units and Values
`Color`	Line color for a Z or Y Smith chart. For a ZY Smith chart, the Z line color.	Default is `[0.4 0.4 0.4]` (dark gray). For more information, see “Color Specification” on page 2-217.
`LabelColor`	Color of the line labels.	Default is `[0 0 0]` (black). For more information, see “Color Specification” on page 2-217.
`LabelSize`	Size of the line labels.	`FontSize`. Default is `10`.
`LabelVisible`	Visibility of the line labels.	`'on'` (default) or `'off'`
`LineType`	Line spec for a Z or Y Smith chart. For a ZY Smith chart, the Z line spec.	Default is `'-'` (solid line).
`LineWidth`	Line width for a Z or Y Smith chart. For a ZY Smith chart, the Z line width.	Number of points. Default is `0.5`.
`SubColor`	The Y line color for a ZY Smith chart.	Default is `[0.8 0.8 0.8]` (medium gray). For more information, see “Color Specification” on page 2-217.
`SubLineType`	The Y line spec for a ZY Smith chart.	Default is `':'` (dotted line). For more information, see “Line Specification” on page 2-216.
`SubLineWidth`	The Y line width for a ZY Smith chart.	Number of points. Default is `0.5`.
`Type`	Type of Smith chart.	`'z'` (default), `'y'`, or `'zy'`
`Value`	Two-row matrix. Row 1 specifies the values of the constant resistance and reactance lines in the chart. For the constant resistance and reactance lines, each element in Row 2 specifies the value at which the corresponding line in Row 1 ends.	2-by-n matrix. Default is `[0.2000 0.5000 1.0000 2.0000 5.0000; 1.0000 2.0000 5.0000 5.0000 30.0000]`

More About

Line Specification

Use the table provided to set Line style of the Smith Chart

Line Style	Description	Resulting Line
`'-'`	Solid line	————
`'--'`	Dashed line	- — — — —
`':'`	Dotted line	.....
`'-.'`	Dash-dotted line	—·—·—·—·-

Color Specification

Use the table provided to set color of the Smith Chart.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
'red'	'r'	[1 0 0]	'#FF0000'	
'green'	'g'	[0 1 0]	'#00FF00'	
'blue'	'b'	[0 0 1]	'#0000FF'	
'cyan'	'c'	[0 1 1]	'#00FFFF'	
'magenta'	'm'	[1 0 1]	'#FF00FF'	
'yellow'	'y'	[1 1 0]	'#FFFF00'	
'black'	'k'	[0 0 0]	'#000000'	
'white'	'w'	[1 1 1]	'#FFFFFF'	

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	'#0072BD'	
[0.8500 0.3250 0.0980]	'#D95319'	
[0.9290 0.6940 0.1250]	'#EDB120'	
[0.4940 0.1840 0.5560]	'#7E2F8E'	
[0.4660 0.6740 0.1880]	'#77AC30'	
[0.3010 0.7450 0.9330]	'#4DBEEE'	
[0.6350 0.0780 0.1840]	'#A2142F'	

Version History

Introduced before R2006a

`smithchart` function will be removed

Warns starting in R2022a

`smithchart` is not recommended. Use `smithplot` instead.

To update your code, change instances of the function name `smithchart` to `smithplot` if you have not specified any output arguments. If you have specified output arguments in your code, you must use the new syntaxes of the `smithplot` function to plot your data.

Unlike the `smithchart` function, `smithplot` supports interactive context menus, axes, and marker capabilities.

See Also

`smithplot`

gammain

Calculate input reflection coefficient of two-port network

Syntax

```
coefficient = gammain(s_params,z0,zl)
coefficient = gammain(hs,zl)
```

Description

`coefficient = gammain(s_params,z0,zl)` calculates the input reflection coefficient of a two-port network. `z0` is the reference impedance Z_0; its default value is 50 ohms. `zl` is the load impedance Z_l; its default value is also 50 ohms. `coefficient` is an M-element complex vector.

`coefficient = gammain(hs,zl)` calculates the input reflection coefficient of the two-port network represented by the S-parameter object `hs`.

Examples

Input Reflection Coefficient Calculation

Calculate the input reflection coefficients at each index of an S-parameter array.

```
ckt = read(rfckt.amplifier,'default.s2p');
s_params = ckt.NetworkData.Data;
z0 = ckt.NetworkData.Z0;
zl = 100;
coefficient = gammain(s_params,z0,zl)
```

```
coefficient = 191×1 complex

  -0.7247 - 0.4813i
  -0.7323 - 0.4707i
  -0.7397 - 0.4601i
  -0.7470 - 0.4495i
  -0.7542 - 0.4389i
  -0.7612 - 0.4284i
  -0.7682 - 0.4179i
  -0.7750 - 0.4075i
  -0.7817 - 0.3972i
  -0.7883 - 0.3870i
      ⋮
```

Input Reflection Coefficient Calculation of S-Parameters Object

Define a S-parameters object from a file.

```
s_params = sparameters('default.s2p');
```

Specify the load impedance.

```
zl = 100;
```

Calculate the input reflection coefficients at each index of a `sparameters` object.

```
coefficient = gammain(s_params,zl)
```

```
coefficient = 191×1 complex

  -0.7247 - 0.4813i
  -0.7323 - 0.4707i
  -0.7397 - 0.4601i
  -0.7470 - 0.4495i
  -0.7542 - 0.4389i
  -0.7612 - 0.4284i
  -0.7682 - 0.4179i
  -0.7750 - 0.4075i
  -0.7817 - 0.3972i
  -0.7883 - 0.3870i
      ⋮
```

Input Arguments

`s_params` — Two-port S-parameters

complex 2-by-2-by-*M* array

Two-port S-parameters, specified as a complex 2-by-2-by-*M* array. *M* is the number of two-port S-parameters.

Data Types: `double`

`z0` — Reference impedance

`50` (default) | positive scalar

Reference impedance, specified as a positive scalar.

Data Types: `double`

`zl` — Load impedance

`50` (default) | positive scalar

Load impedance, specified as a positive scalar.

Data Types: `double`

`hs` — Two-port network

S-parameter object

Two-port network, specified as an S-parameter object.

Data Types: `function_handle`

Output Arguments

`coefficient` — Input reflection coefficient
M element complex vector

Input reflection coefficient, returned as a *M* element complex vector.

Algorithms

`gammain` uses the equation:

$$\Gamma_{in} = S_{11} + \frac{(S_{12}S_{21})\Gamma_L}{1 - S_{22}\Gamma_L}$$

where

$$\Gamma_L = \frac{Z_l - Z_0}{Z_l + Z_0}$$

Version History

Introduced before R2006a

See Also

`gammaout`

noisefigure

Calculate noise figure of transmission lines, series RLC, and shunt RLC circuits

Syntax

```
nf = noisefigure(transmissionline,frequency,zs)
nf = noisefigure(RLCckt,frequency,zs)
```

Description

`nf = noisefigure(transmissionline,frequency,zs)` calculate the noise figure of transmission lines using the given frequency and source impedance.

`nf = noisefigure(RLCckt,frequency,zs)` calculate the noise figure of series and shunt RLC circuits using the given frequency and source impedance. `RLCckt` is either `seriesRLC` or `shuntRLC` objects.

Examples

Group Delay and Noise Figure of Two-Wire Transmission Line

Create a two-wire transmission line using these specifications:

- Radius - 0.5 mm
- Dielectric - air
- Thickness of dielectric or separation - 1.088 mm
- Permittivity or `EpsilonR` - 1.0054

```
twowiretxline = txlineTwoWire('Radius',0.5e-3,'EpsilonR',1.0054,'Separation',1.088e-3);
```

Calculate the noise figure and the group delay of the transmission line at 2.5 GHz.

```
nf = noisefigure(twowiretxline,2.5e9)
```

```
nf = 0
```

```
gd = groupdelay(twowiretxline,2.5e9)
```

```
gd = 3.3446e-11
```

Input Arguments

`transmissionline` — Transmission line

`txline` objects

Transmission line, specified as any one of these objects:

- `txlineCoaxial`

- `txlineCPW`
- `txlineMicrostrip`
- `txlineParallelPlate`
- `txlineRLCGLine`
- `txlineTwoWire`
- `txlineEquationBased`
- `txlineDelayLossless`
- `txlineDelayLossy`
- `txlineElectricalLength`
- `txlineStripline`

`RLCckt` — Series or shunt RLC circuit objects
`seriesRLC` object | `shuntRLC` object

Series or shunt RLC circuit objects, specified as a `seriesRLC` or `shuntRLC` objects.

`frequency` — Frequency to calculate noise figure
positive scalar

Frequency to calculate noise figure, specified as a positive scalar in hertz.

`zs` — Source impedance
`50` (default) | positive real scalar

Source impedance, specified as a positive real scalar in ohms.

Version History

Introduced in R2020b

See Also

`sparameters` | `groupdelay` | `getZ0`

getZ0

Calculate characteristic impedance with and without dispersion for transmission line

Syntax

```
z0_f = getZ0(txline,freq)
[z0,eff_ep] = getZ0(txline)
[z0_f,eff_epf] = getZ0(txline,freq)
```

Description

`z0_f = getZ0(txline,freq)` returns the characteristic impedance with dispersion for a transmission line at the specified frequency.

`[z0,eff_ep] = getZ0(txline)` returns the characteristic impedance and effective epsilon without dispersion for a microstrip or coplanar waveguide (CPW) transmission line.

`[z0_f,eff_epf] = getZ0(txline,freq)`returns the characteristic impedance and effective epsilon with dispersion for a microstrip or CPW transmission line at the specified frequency.

Examples

Calculate Characteristic Impedance with Dispersion for Coaxial Transmission Line

Create a coaxial transmission line.

```
txline = txlineCoaxial('OuterRadius',1.47e-3,'InnerRadius',0.45e-3,...
            'EpsilonR',3.4,'SigmaCond',5.8e7);
```

Calculate the characteristic impedance with dispersion for the coaxial transmission line at at the frequency of 6 GHz.

```
z0_f = getZ0(txline,6e9)
```

```
z0_f = 38.4927 - 0.0201i
```

Calculate Effective Epsilon with Dispersion for Microstrip Transmission Line

Create a microstrip transmission line.

```
txline_morcpw = txlineMicrostrip('Width',4.78e-3,'Height',1.57e-3,...
    'LineLength',12.2777e-3,'Thickness',0.003e-3,'EpsilonR',2.2,'SigmaCond',5.88e7);
```

Calculate the characteristic impedance and effective epsilon with dispersion for the microstrip transmission line for a frequency range of 6–7 GHz.

```
[z0_f,eff_ep_f] = getZ0(txline_morcpw,6e9:0.2e9:7e9)
```

```
z0_f = 6×1

   52.4097
   52.4822
   52.5546
   52.6270
   52.6994
   52.7716


eff_ep_f = 6×1

    1.9106
    1.9123
    1.9139
    1.9156
    1.9173
    1.9189
```

Calculate Characteristic Impedance and Effective Epsilon of CPW Transmission Line

Create a coplanar waveguide (CPW) transmission line.

```
txline_morcpw = txlineCPW('ConductorWidth',45e-6,'SlotWidth',50e-6,'Height',525e-6,...
    'Thickness',1e-6,'EpsilonR',2.5,'SigmaCond',3.33e7);
```

Calculate the characteristic impedance and effective epsilon of the CPW transmission line.

```
[z0,eff_ep] = getZ0(txline_morcpw)
```

```
z0 = 116.8647
```

```
eff_ep = 1.7303
```

Input Arguments

`txline` — Transmission line

transmission line object

Transmission line, specified as one of the following transmission line objects:

- `txlineCoaxial`
- `txlineCPW`
- `txlineMicrostrip`
- `txlineParallelPlate`
- `txlineRLCGLine`
- `txlineTwoWire`
- `txlineStripline`

`freq` — Design frequency

positive scalar | vector

Design frequency, specified as a positive scalar or vector in hertz.

Output Arguments

`z0` — Characteristic impedance without effect of dispersion
complex scalar

Characteristic impedance without the effect of dispersion, returned as a complex scalar.

Data Types: `double`

`z0_f` — Characteristic impedance with effect of dispersion
vector

Characteristic impedance with the effect of dispersion for a microstrip or CPW transmission line, returned as a vector.

Data Types: `double`

`eff_ep` — Effective epsilon without effect of dispersion
scalar

Effective epsilon without the effect of dispersion for a microstrip or CPW transmission line, returned as a scalar.

Data Types: `double`

`eff_epf` — Effective epsilon with effect of dispersion
scalar | vector

Effective epsilon with the effect of dispersion for a microstrip or CPW transmission line, returned as a scalar or vector.

Data Types: `double`

Version History

Introduced in R2020b

See Also

`sparameters` | `groupdelay` | `noisefigure`

timeresp

Time response for rational objects

Syntax

```
[y,t,xf] = timeresp(h,inputsignal,ts)
[y,t,xf] = timeresp(h,inputsignal,ts,xo)
```

Description

`[y,t,xf] = timeresp(h,inputsignal,ts)` computes the time response `y`, sample time `t` of `y`, and final states of the system `xf`. The function calculates the time response for a `rational`, `rfmodel.rational`, or `rationalfit` object `h`, using the time-varying input signal `inputsignal` at the specified sample time `ts`.

`[y,t,xf] = timeresp(h,inputsignal,ts,xo)` computes the time response using `xo` as the initial states of a system.

Examples

Time Response of Rational Object

Compute Time Response with Single Input Signal

Define an input signal.

```
SampleTime = 2e-11;
OverSamplingFactor = 25;
TotalSampleNumber = 2^12;
InputSignal = sign(randn(1, ...
              ceil(TotalSampleNumber/OverSamplingFactor)));
InputSignal = repmat(InputSignal, [OverSamplingFactor, 1]);
InputSignal = InputSignal(:);
```

Create a `rational` function object.

```
S = sparameters('default.s2p');
s21 = rfparam(S,2,1);
datafreq = S.Frequencies;
fit_data = rationalfit(datafreq,s21,'Tolerance',-32);
```

Compute the time response with initial states set to `0`.

```
[y,t,xf]=timeresp(fit_data,InputSignal,SampleTime);
```

Compute Time Response with Segmented Input Signals

Segment the input signal into two parts.

Compute the time response of the first half of the input signal segment.

```
[y2(1:2000,1),t21,xf2]=timeresp(fit_data,InputSignal(1:2000),SampleTime);
```

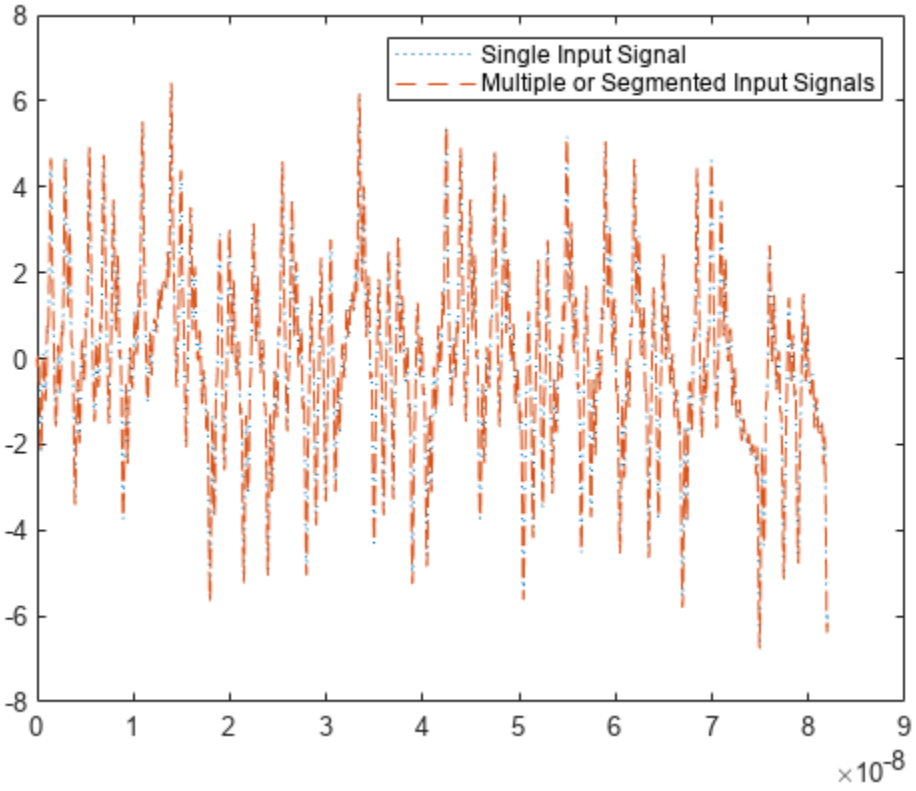
Use the final states of the first half of the input signal segment `xf2` as the initial states to compute the time response of the second half of the input signal segment.

```
[y2(2001:4100,1),t22,xf2]=timeresp(fit_data,InputSignal(2001:4100),SampleTime,xf2);
```

Plot Time Responses

Plot the time responses of the single and segmented input signals.

```
plot(t,y,':',t,y2,'--')
legend({'Single Input Signal','Multiple or Segmented Input Signals'})
```



Input Arguments

`inputsignal` — Time-varying input signal
real finite vector

Time-varying input signal, specified as a real finite vector.

Data Types: `double`

h — Rational function object
`rationalfit` object | `rfmodel.rational` object | `rational` object

Rational function object, specified as a `rationalfit`, `rfmodel.rational`, or a `rational` object.

Data Types: `double`
Complex Number Support: Yes

`ts` — Sample time of input signal
positive scalar integer

Sample time of the input signal, specified as a positive scalar integer in seconds.

Data Types: `double`

`xo` — Initial states of system
`0` (default) | real or complex column vector

Initial states of the system, specified as a real or complex column vector of length L, where L is equal to `length(h.ComplexPoles/2)+length(h.RealPoles)`.

Data Types: `double`

Output Arguments

`y` — Time-varying output signal
real finite vector

Time-varying output signal, returned as a real finite vector.

Data Types: `double`

`t` — Sample time of output signal
nonnegative vector

Sample time of the output signal, returned as a nonnegative vector of time values with discrete step size `ts` corresponding to the data in `y` in seconds.

Data Types: `double`

`xf` — Final states of system
real or complex column vector

Final states of the system that the function uses in computing the last value of the `y`, returned as a real or complex column vector of length L, where L is equal to `length(h.ComplexPoles/2)+length(h.RealPoles)`.

Data Types: `double`

More About

Output Signal Equation

`timeresp` function uses a variant of this equation to calculate the output signal.

$$Y(n) = sum(C .* X(n - Delay/ts)) + D*U(n - Delay/ts)$$

where

$$X(n+1) = F*X(n) + G*U(n)$$
$$X(1) = 0$$
$$F = \exp(A*ts)$$
$$G = (F - 1) ./ A$$

`A` — Complex vector of poles of the rational function

`C` — Complex vector of residues of the rational function

`D` — Scalar value specifying direct feedthrough

`Delay` — Delay time properties of the rational function object `h`. For more information, see `rationalfit` function.

Version History

Introduced in R2007a

Compute time response of rational objects with initial states

To compute the time response of rational objects with the initial states, use `xo` as one of the input arguments to this function.

See Also

`freqresp` | `pwlresp` | `rationalfit` | `makepassive` | `writeva` | `ispassive` | `rfmodel.rational` | `rational`

stepresp

Step-signal response for rational object and `rationalfit` function object

Syntax

```
[outputsignal,tout] = stepresp(h,ts,n,trise)
```

Description

`[outputsignal,tout] = stepresp(h,ts,n,trise)` computes the time domain response of a rational function object, `h`, to a step signal based on the number of samples, `n` and the rise time, `trise`.

Examples

Calculate Step Response

Calculate the step response of a rational function object from the file `passive.s2p`. Read `passive.s2p`.

```
S = sparameters('passive.s2p');
freq = S.Frequencies;
```

Get S11 and convert to a TDR transfer function.

```
s11 = rfparam(S,1,1);
Vin = 1;
tdrfreqdata = Vin*(s11+1)/2;
```

Fit to a rational function object.

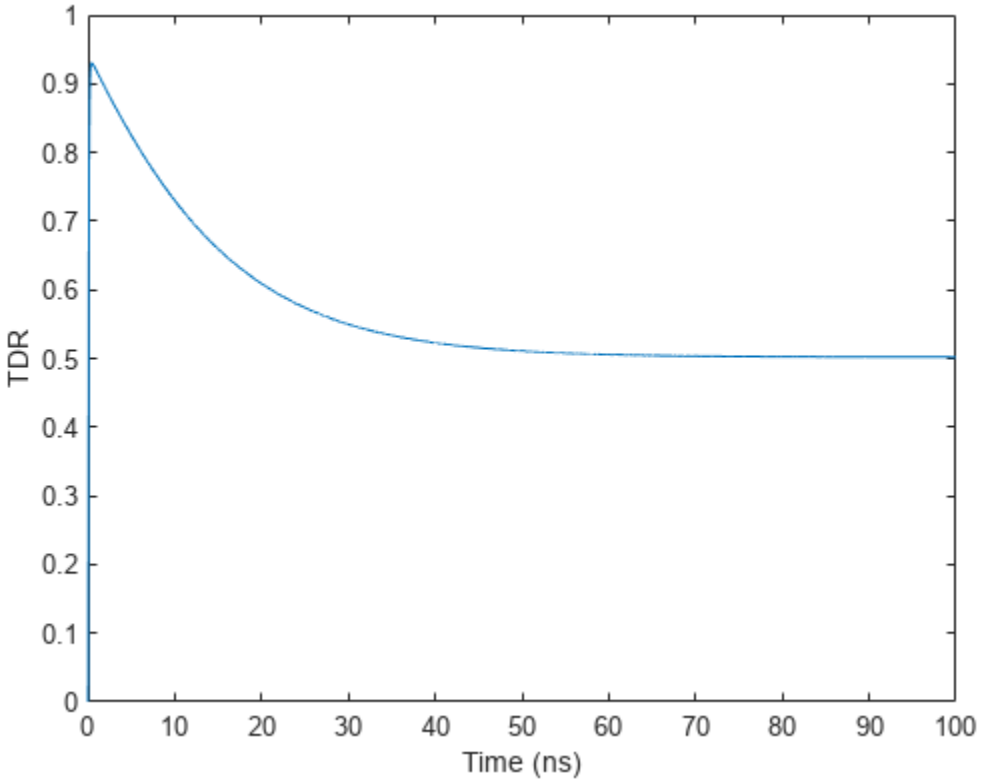
```
tdrfit = rationalfit(freq,tdrfreqdata);
```

Define parameters for a step signal. Define parameters for a step signal

```
Ts = 1.0e-11;
N = 10000;
Trise = 1.0e-10;
```

Calculate the step response for TDR and plot it

```
[tdr,t1] = stepresp(tdrfit,Ts,N,Trise);
figure
plot(t1*1e9,tdr)
ylabel('TDR')
xlabel('Time (ns)')
```



Input Arguments

h — Rational function object

`rationalfit` object | `rational` object

Rational function object, specified as a `rationalfit` or `rational` object.

Data Types: `double`
Complex Number Support: Yes

ts — Sample time of input signal

positive scalar integer

Sample time of the input signal, specified as a positive scalar integer in seconds.

Data Types: `double`

n — Number of samples

positive scalar integer

Number of samples, specified as a positive scalar integer.

Data Types: `double`

trise — Time taken for step signal to reach maximum value

positive scalar integer

Time taken for step signal to reach maximum value, specified as a positive scalar integer in seconds.

Data Types: `double`

Output Arguments

`outputsignal` — Computed step response

real finite vector

Computed step response, returned as a real finite vector.

Data Types: `double`

`tout` — Sample time of output signal

nonnegative vector

Sample time of the output signal, returned as a nonnegative vector of time values with discrete step size `ts` corresponding to the data in `outputsignal` in seconds.

Data Types: `double`

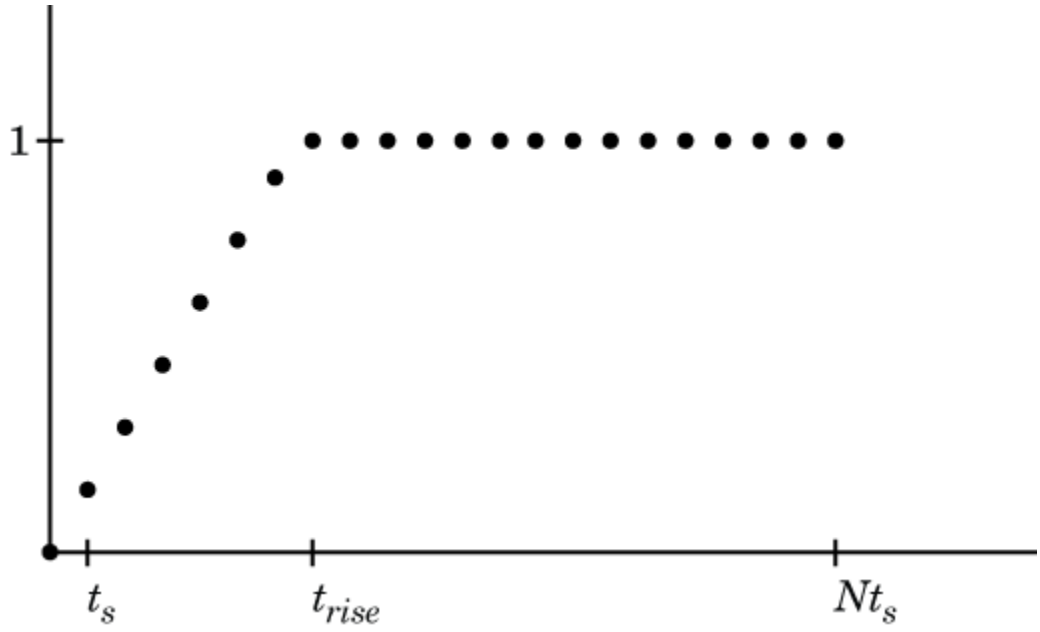
More About

Step Signal Equation

RF Toolbox uses the following equation to for the step signal:

$$\begin{cases} U(kt_s) = kt_s/t_{rise}, & 0 \le k < (t_{rise}/t_s) \\ U(kt_s) = 1, & (t_{rise}/t_s) \le k \le N \end{cases}$$

The following figure illustrates the construction of this signal.



Version History

Introduced in R2010a

See Also

`freqresp` | `pwlresp` | `rationalfit` | `makepassive` | `writeva` | `ispassive` | `rfmodel.rational`

impulse

Impulse response for rational function object

Syntax

```
[response,tout] = impulse(h,ts,n)
```

Description

`[response,tout] = impulse(h,ts,n)` computes the impulse response of a rational function object, `h`, over a time period specified by `ts` and the number of samples `n`.

Note While you can compute the output response for a rational function object by computing the impulse response of the object and then convolving that response with the input signal, this approach is not recommended. Instead, you should use the `timeresp` method to perform this computation because it generally gives a more accurate output signal for a given input signal.

Examples

Impulse Response of Data Stored In File

Create a `sparameters` object from a file and use `rfparam` to extract the S_{21}parameters.

```
S = sparameters('passive.s2p');
S21 = rfparam(S,2,1);
```

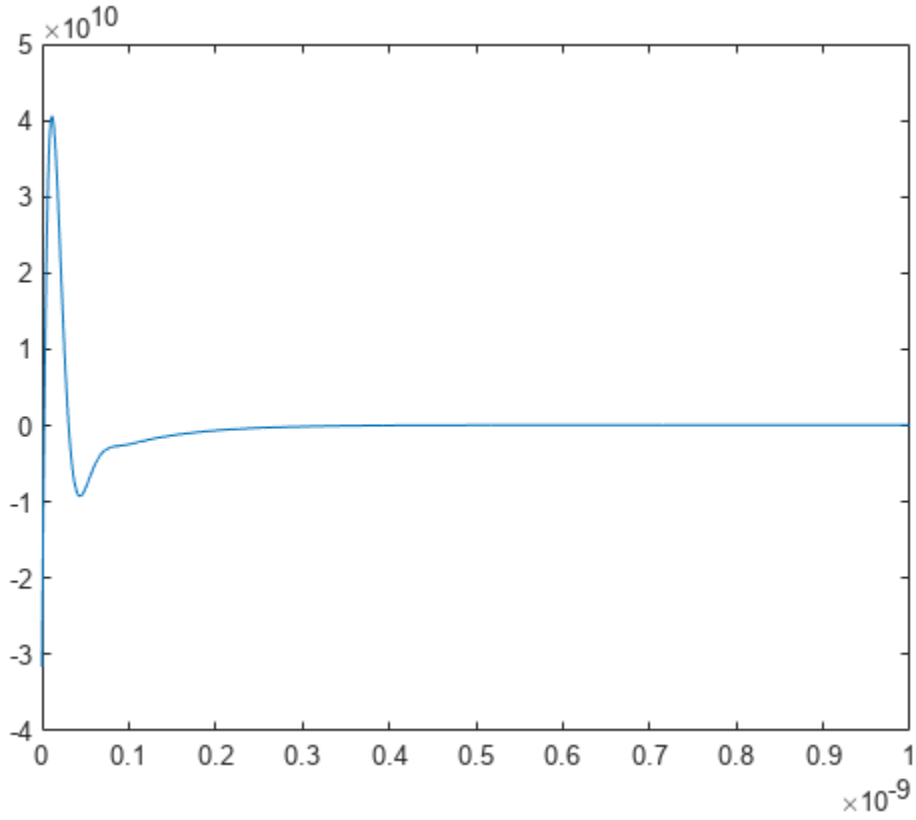
Fit a rational function object to the S_{21} data by using `rationalfit`.

```
freq = S.Frequencies;
fit_data = rationalfit(freq,S21)
```

```
fit_data =
   rfmodel.rational with properties:

        A: [6x1 double]
        C: [6x1 double]
        D: 0
    Delay: 0
     Name: 'Rational Function'
```

Calculate the impulse response using the `impulse` method and plot the results.

```
[resp,t] = impulse(fit_data,1e-12,1e3);
plot(t,resp);
```



Impulse Response of S-Parameters Object

Perform rational fitting on a S-parameters object to create a rational function.

```
S = sparameters('passive.s2p');
fit = rational(S);
```

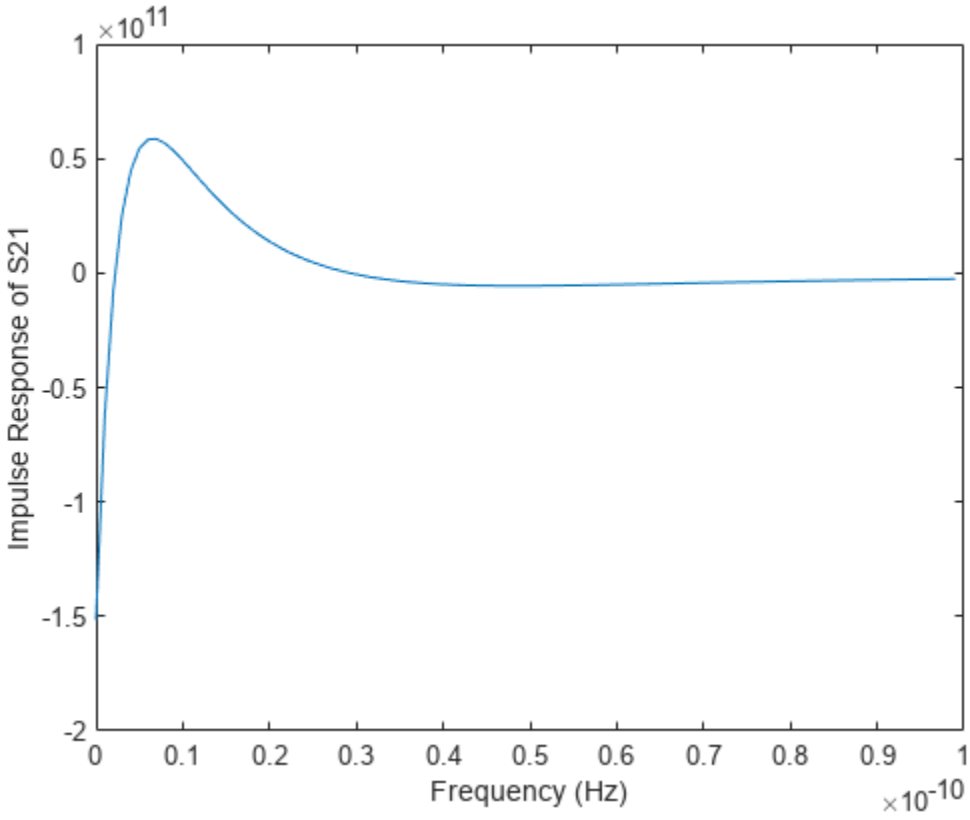
Compute the impulse response using the `impulse` method.

```
[resp,t] = impulse(fit,1e-12,1e2)
```

```
resp=2×2 cell array
    {100x1 double}    {100x1 double}
    {100x1 double}    {100x1 double}


t=2×2 cell array
    {100x1 double}    {100x1 double}
    {100x1 double}    {100x1 double}
```

Plot the results of the S21 data.

```
plot(t{2,1},resp{2,1})
xlabel('Frequency (Hz)');
ylabel('Impulse Response of S21');
```



Input Arguments

h — Rational function object

`rationalfit` object | `rational` object

Rational function object, specified as a `rationalfit` or a `rational` object.

Data Types: `double`
Complex Number Support: Yes

ts — Sample time of computed impulse response

positive scalar integer

Sample time of the computed impulse response, specified as a positive scalar integer in seconds.

Data Types: `double`

n — Number of samples

positive scalar integer

Number of samples in the impulse response, specified as a positive scalar integer.

Data Types: `double`

Output Arguments

`response` — Computed impulse response

real vector

Computed impulse response, returned as an *n* element real vector of impulse response values.

`tout` — Sample time of output signal

nonnegative vector

Sample time of the output signal, returned as a nonnegative vector of time values with discrete step size `ts` corresponding to the data in `response` in seconds.

Data Types: `double`

More About

Impulse Response And Time Samples Equation

RF Toolbox uses the following equation to for the impulse response:

$$resp = \sum_{k=1}^{M} C_k e^{A_k(t - Delay)} u(t - Delay) + D\delta(t - Delay)$$

where

- `A`, `C`, `D`, and `Delay` are properties of the rational function object, `h`.
- `M` is the number of poles in the rational function object.

The vector of time samples of the impulse response, `t`, is computed from the inputs as `t = [0,ts,2*ts,...,(n-1)*ts]`

Version History

Introduced in R2006b

See Also

`freqresp` | `pwlresp` | `rationalfit` | `makepassive` | `writeva` | `ispassive` | `rfmodel.rational`

save_system

Save RF Blockset model created using `rfsystem`

Syntax

```
filename = save_system(rfs)
filename = save_system(rfs,newrfs)
```

Description

`filename = save_system(rfs)` saves the RF Blockset model created using the `rfsystem` System object `rfs` .

`filename = save_system(rfs,newrfs)` saves the model to a new file, `newrfs`.

Examples

Open, Save, and Close RF Blockset Model

Create a fifth-order bandpass RF filter and an amplifier with the gain of 3 dB.

```
f1 = rffilter('ResponseType','Bandpass','FilterOrder',5,'PassbandFrequency',[4.85 5.15]*1e9);
a1 = amplifier('Gain',3,'NF',1.53,'OIP3',35);
```

Create an `rfbudget` object using the elements at an input frequency of 5 GHz, an available input power of `-30` dBm, and a bandwidth of `200` MHz.

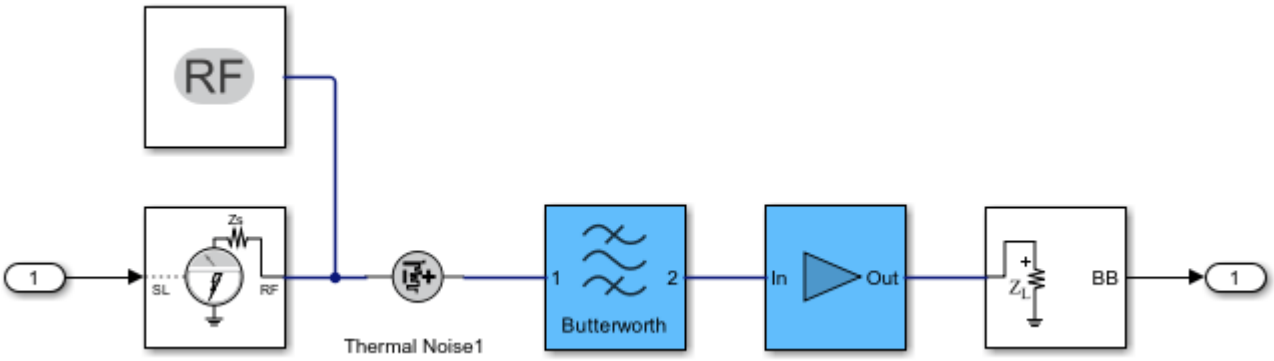
```
rfb = rfbudget([f1 a1],5e9,-30,200e6);
```

Create an RF system using the `rfbudget` object and name the RF Blockset model.

```
rfs = rfsystem(rfb,'ModelName','RFSystem');
```

Open the RF Blockset model.

```
open_system(rfs)
```



Save and close the RF Blockset model.

```
save_system(rfs);
close_system(rfs)
```

Input Arguments

`rfs` — RF system

`rfsystem` object

RF system, specified as an `rfsystem` object

`newrfs` — File to save to

character vector | string scalar

File to save to, specified as a character vector or string scalar. You can specify a model name in the current folder or the full path name, with or without an extension.

Example: `save_system(rfs,'mysystem2')`

Output Arguments

`filename` — Name of saved file

character vector | cell array of character vectors

Name of the saved file, returned as a character vector or a cell array of character vectors.

Tips

- To save a smaller RF system use the `release` function before `save_system(rfs)`.

Version History

Introduced in R2021a

See Also

`rfsystem` | `close_system` | `open_system` | `hide_system`

open_system

Open RF Blockset model created using `rfsystem`

Syntax

```
open_system(rfs)
```

Description

`open_system(rfs)` opens the RF Blockset model created using the `rfsystem` System object `rfs`.

Examples

Open, Save, and Close RF Blockset Model

Create a fifth-order bandpass RF filter and an amplifier with the gain of 3 dB.

```
f1 = rffilter('ResponseType','Bandpass','FilterOrder',5,'PassbandFrequency',[4.85 5.15]*1e9);
a1 = amplifier('Gain',3,'NF',1.53,'OIP3',35);
```

Create an `rfbudget` object using the elements at an input frequency of 5 GHz, an available input power of `-30` dBm, and a bandwidth of `200` MHz.

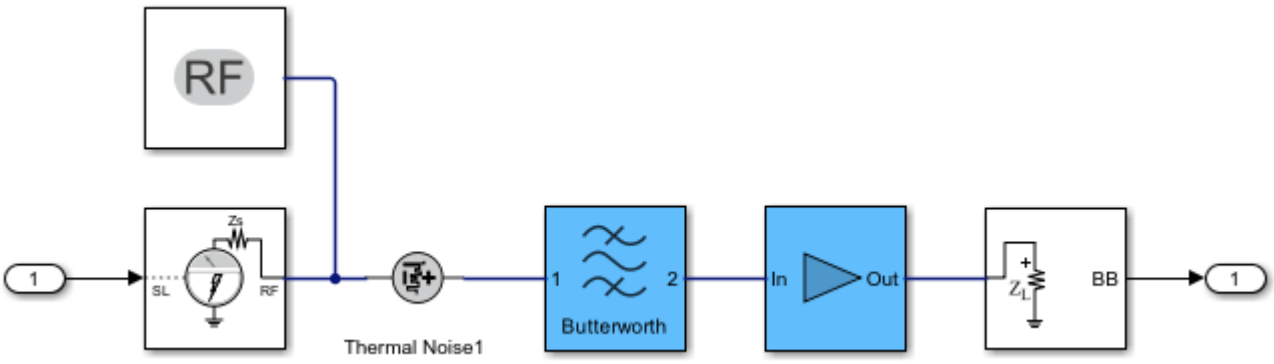
```
rfb = rfbudget([f1 a1],5e9,-30,200e6);
```

Create an RF system using the `rfbudget` object and name the RF Blockset model.

```
rfs = rfsystem(rfb,'ModelName','RFSystem');
```

Open the RF Blockset model.

```
open_system(rfs)
```



Save and close the RF Blockset model.

```
save_system(rfs);
close_system(rfs)
```

Input Arguments

`rfs` — RF system

`rfsystem` object

RF system, specified as an `rfsystem` object

Version History

Introduced in R2021a

See Also

`rfsystem` | `save_system` | `close_system` | `hide_system`

hide_system

Hide RF Blockset model window created using `rfsystem`

Syntax

```
hide_system(rfs)
```

Description

`hide_system(rfs)` hides the RF Blockset model window created using `rfsystem` System object `rfs`.

Examples

Hide RF Blockset Model Created

Create a fifth-order bandpass RF filter and an amplifier with the gain of 3 dB.

```
f1 = rffilter('ResponseType','Bandpass','FilterOrder',5,'PassbandFrequency',[4.85 5.15]*1e9);
a1 = amplifier('Gain',3,'NF',1.53,'OIP3',35);
```

Create an `rfbudget` object using the elements at an input frequency of 5 GHz, an available input power of `-30` dBm, and a bandwidth of `200` MHz.

```
rfb = rfbudget([f1 a1],5e9,-30,200e6);
```

Create an RF system using the `rfbudget` object.

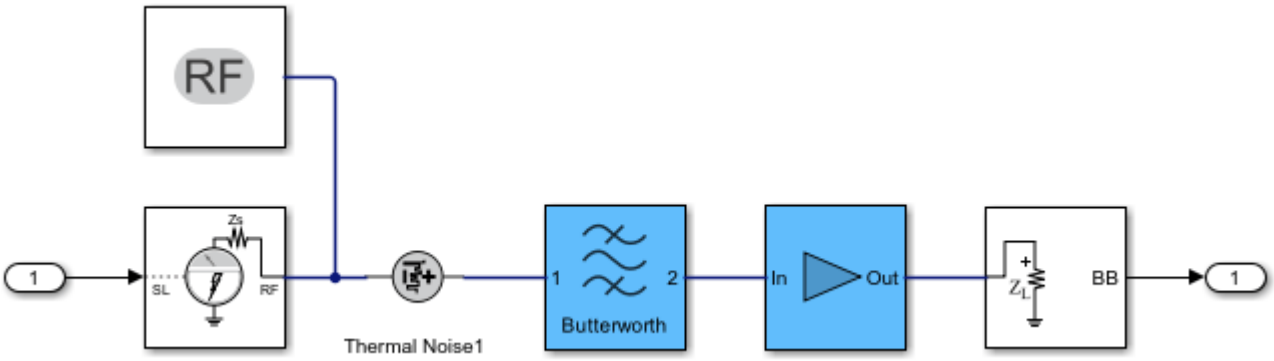
```
rfs = rfsystem(rfb,'ModelName','myRFmodel');
```

Release system resources and turn off fast restart.

```
release(rfs)
```

Open an RF Blockset model of the designed RF system using the `open_system` object function.

```
open_system(rfs)
```



Hide the RF Blockset model.

```
hide_system(rfs)
```

Input Arguments

`rfs` — RF system
`rfsystem` object

RF system, specified as an `rfsystem` object

Version History

Introduced in R2021a

See Also

`rfsystem` | `open_system` | `save_system` | `close_system`

close_system

Close RF Blockset model window created using `rfsystem`

Syntax

```
close_system(rfs)
close_system(rfs,saveflag)
```

Description

`close_system(rfs)` closes the RF Blockset model window created using `rfsystem` System object `rfs`.

Note You must save your system before using the `close_system` function.

`close_system(rfs,saveflag)` lets you specify whether to save the RF Blockset model with its current name or to close it without saving.

Examples

Open, Save, and Close RF Blockset Model

Create a fifth-order bandpass RF filter and an amplifier with the gain of 3 dB.

```
f1 = rffilter('ResponseType','Bandpass','FilterOrder',5,'PassbandFrequency',[4.85 5.15]*1e9);
a1 = amplifier('Gain',3,'NF',1.53,'OIP3',35);
```

Create an `rfbudget` object using the elements at an input frequency of 5 GHz, an available input power of `-30` dBm, and a bandwidth of `200` MHz.

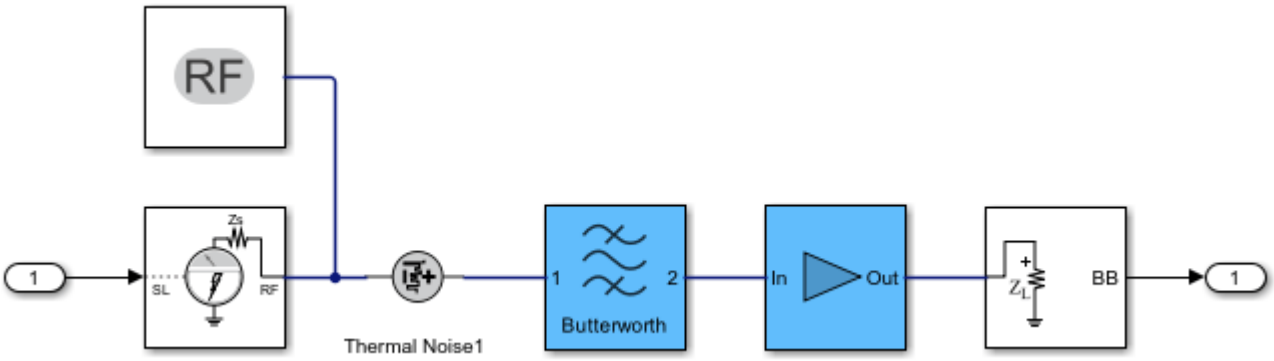
```
rfb = rfbudget([f1 a1],5e9,-30,200e6);
```

Create an RF system using the `rfbudget` object and name the RF Blockset model.

```
rfs = rfsystem(rfb,'ModelName','RFSystem');
```

Open the RF Blockset model.

```
open_system(rfs)
```



Save and close the RF Blockset model.

```
save_system(rfs);
close_system(rfs)
```

Input Arguments

`rfs` — RF system

`rfsystem` object

RF system, specified as an `rfsystem` object.

`saveflag` — RF system

`0` (default) | `1`

Option to save model using the current file name, specified as `0` to close without saving or `1` to save and then close.

Version History

Introduced in R2021a

See Also

`rfsystem` | `open_system` | `save_system` | `hide_system`

richards

Convert lumped element circuit to distributed element circuit using Richards' transformation

Syntax

```
cktOut = richards(cktIn,opFreq)

txOut = richards(LorCobj,opFreq)
[txOut,nodes] = richards(LorCobj,opFreq)
[txOut,nodes] = richards( ___ ,stubmode=stubType)
```

Description

`cktOut = richards(cktIn,opFreq)` applies Richards' transformation on the circuit `cktIn` and returns the circuit object `cktOut` at the given reference frequency `opFreq`. In the `cktOut` all capacitors and inductors are replaced by electrical-length-based transmission line objects `txlineElectricalLength`.

Note You can apply Richard's transformation only to circuits where all negative terminals of the ports share the same node.

`txOut = richards(LorCobj,opFreq)` convert a capacitor or inductor `LorCobj` into an electrical-length-based transmission line object `txOut` at `opFreq`.

`[txOut,nodes] = richards(LorCobj,opFreq)` also returns a vector of suggested nodes `nodes` to connect `txOut` if `LorCobj` is connected to a circuit.

`[txOut,nodes] = richards( ___ ,stubmode=stubType)` specify the stub type of the output transmission line by using a name-value argument.

Examples

Apply Richards' Transformation to RF Filter

Create 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;
cktIn = rffilter("FilterType","Chebyshev","ResponseType","Lowpass","Implementation","LC Pi","Pass
    Fp,"PassbandAttenuation",Ap,"FilterOrder",Ord);
opFreq = 1e9;
```

Convert the lumped elements of the RF filter to a distributed element using Richards' transformation.

```
cktOut = richards(cktIn,opFreq)
```

```
cktOut =
  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-'}
```

Apply Richards' Transformation to Inductor

Create an inductor with the inductance of 5 nH.

```
LorCobj= inductor(5e-9);
```

Create a circuit.

```
ckt = circuit('new_circuit1');
```

Add a resistor and the inductor you created earlier to the circuit.

```
add(ckt,[1 2],LorCobj);
add(ckt,[2 3],resistor(100));
```

Set the ports and display the results.

```
setports(ckt,[1 0],[3 0])
disp(ckt)
```

```
  circuit: Circuit element

    ElementNames: {'L'  'R'}
        Elements: [1x2 rf.internal.circuit.RLC]
           Nodes: [0 1 2 3]
            Name: 'new_circuit1'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Apply Richards' transformation to the inductor at 1 GHz and display nodes to connect the transmission line.

```
[txOut,nodes] = richards(LorCobj,1e9)
```

```
txOut =
  txlineElectricalLength: ElectricalLength element

                    Name: 'L_tx'
                      Z0: 31.4159
              LineLength: 0.7854
    ReferenceFrequency: 1.0000e+09
           Termination: 'Short'
              StubMode: 'Series'
              NumPorts: 2
             Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

```
nodes = 1×4

     1     2     0     0
```

The nodes in this example represent the nodes at which a 2-port transmission line representing the series L. The returned node value is set to `-1` when a ground node cannot be determined from the circuit.

Input Arguments

`cktIn` — Input RF circuit
`circuit` object | `lcladder` object | `rffilter` object | `matchingnetwork` object

Input RF circuit, specified as a `circuit`, `lcladder`, `rffilter`, or `matchingnetwork` object.

`LorCobj` — Inductor or capacitor
`inductor` object | `capacitor` object

Inductor or capacitor, specified as an `inductor` or `capacitor` object.

`opFreq` — Operating frequency
positive scalar

Operating frequency at which the Richards' transformation is applied, specified as a positive scalar.

`stubType` — Stub type
`'Series'` (default) | `'Shunt'`

Stub type of `txOut`, specified as a `'Series'` or `'Shunt'`.

Output Arguments

`cktOut` — Output circuit
`circuit` object

Output circuit, returned as a `circuit` object.

`txOut` — Electrical-length-based transmission line
`txlineElectricalLength` object

Electrical-length-based transmission line, returned as a `txlineElectricalLength` object.

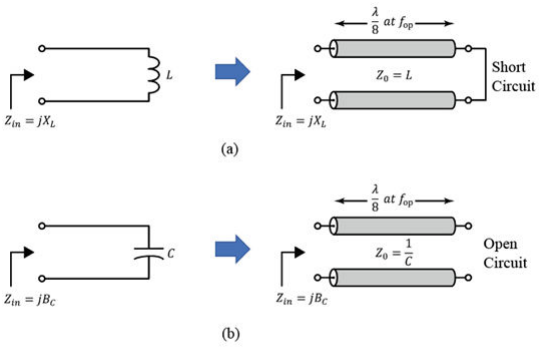
`nodes` — Nodes to connect `txOut`
vector

Nodes to connect `txOut`, returned as a vector.

Algorithms

Richards' Transformation

This figure shows how Richards' transformation converts a circuit with capacitors and inductors into an abstract transmission line model [1].



Richards' transformation (a) inductor to short stub (b) capacitor to open stub

Version History

Introduced in R2021b

References

[1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.

See Also

Functions

`kuroda` | `insertUnitElement` | `realize`

Objects

`txlineElectricalLength`

Topics

“Richards-Kuroda Workflow for RF Filter Circuit”

kuroda

Apply Kuroda's transformation based on Kuroda's identities

Syntax

```
outObj = kuroda(inObj,EL1,EL2)
outObj = kuroda(inObj,EL1,EL2,EL3)
```

Description

`outObj = kuroda(inObj,EL1,EL2)` applies the suitable Kuroda's identity to the two elements `EL1` and `EL2` in the circuit `inObj`.

`outObj = kuroda(inObj,EL1,EL2,EL3)` applies the suitable Kuroda's identity to the of three elements `EL1`, `EL2`, and `EL3`.

Note You can apply Kuroda's identities only to circuits where all negative terminals of the ports share the same node.

Examples

Apply Kuroda's Transformation to Two Elements

Create a lowpass pi LC ladder object.

```
L = 3.18e-8;
C = [6.37e-12 6.37e-12];
lpp = lcladder('lowpasspi',L,C);
```

Apply Richards' transformation to the LC ladder object.

```
r = richards(lpp,1e9)
```

```
r =
  circuit: Circuit element

    ElementNames: {'C_tx'  'L_tx'  'C_1_tx'}
        Elements: [1x3 txlineElectricalLength]
           Nodes: [0 1 2 3 4]
            Name: 'unnamed'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Insert a unit element at port 1 of the circuit at the operating frequency of 1 GHz and characteristic impedance of 50 ohms.

```
UE = insertUnitElement(r,'C_tx',1,1e9,50)
```

```
UE =
  circuit: Circuit element
```

```
    ElementNames: {'C_tx_p1_elem_UE'  'C_tx'  'L_tx'  'C_1_tx'}
        Elements: [1x4 txlineElectricalLength]
           Nodes: [0 1 2 3 4 5]
            Name: 'unnamed'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Apply Kuroda's transformation to the two elements `C_tx_p1_elem_UE` and `C_tx`.

```
ku = kuroda(UE,'C_tx_p1_elem_UE','C_tx')
```

```
ku =
  circuit: Circuit element

    ElementNames: {1x4 cell}
        Elements: [1x4 txlineElectricalLength]
           Nodes: [0 1 2 3 4 5]
            Name: 'unnamed'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Input Arguments

`inObj` — RF circuit

`circuit` object

RF circuit, specified as a `circuit` object. The `circuit` object can have a `txlineElectricalLength` object as one of its elements.

`EL1` — First element

`txlineElectricalLength` object | scalar

First element in `inObj`, specified as a `txlineElectricalLength` object or a scalar. When you specify the value as scalar, the value refers to an index of element in the circuit. This element must be sequentially connected to the second element specified in the Kuroda's transformation.

`EL2` — Second Element

`txlineElectricalLength` object | scalar

Second element in `inObj`, specified as a `txlineElectricalLength` object or a scalar. When you specify the value as scalar, the value refers to an index of element in the circuit. This element must be sequentially connected to the first element specified in the Kuroda's transformation.

`EL3` — Third element

`nport` object | scalar

Third element in `inObj`, specified as an `nport` object or a scalar. When you specify the value as an `nport` object the value refers to an ideal transformer and when you specify the value as a scalar, the value refers to an index of element in the circuit.

An ideal transformer is implemented by using a 2-port `nport` element with S-parameter data corresponding to an `1:N` or `N:1` ideal transformer. That is, the transformer must be passive, lossless,

frequency independent, and with S-parameter data conforming to `S12 = S21`, and `S12 = N x (1+S11)`, where `N` is the number of turns in a transformer.

This element must be sequentially connected to the first two elements specified in the Kuroda's transformation.

Output Arguments

`outObj` — Circuit element

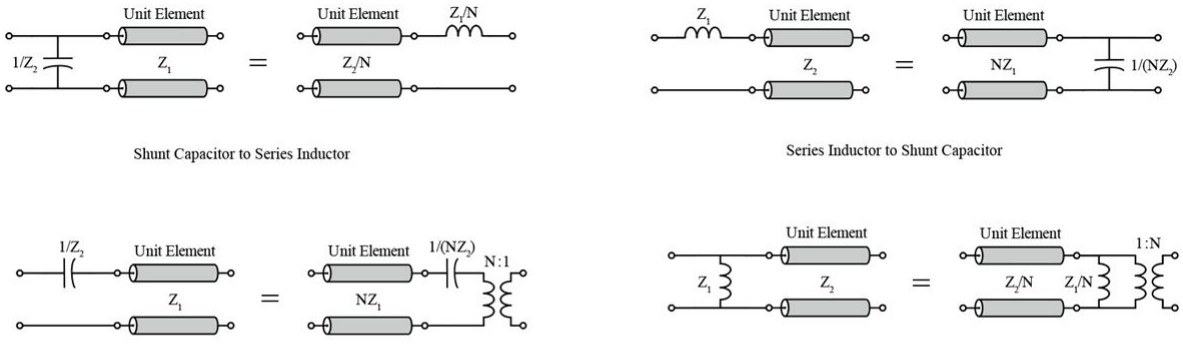
`circuit` object

Circuit element, returned as a `circuit` object.

Algorithms

Kuroda's Transformation

This figure shows how you can apply Kuroda's transformation or identities to a shunt capacitor, shunt inductor, series capacitor,or series inductor [1].



Kuroda's Identites (N = 1+Z2/Z1)

Version History

Introduced in R2021b

References

[1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.

See Also

Functions

`richards` | `insertUnitElement` | `realize`

Objects
`txlineElectricalLength`

Topics
“Richards-Kuroda Workflow for RF Filter Circuit”

insertUnitElement

Insert unit element into circuit object

Syntax

```
unitEle = insertUnitElement(cktIn,cktElem,elePort,opFreq,Z0)
```

Description

`unitEle = insertUnitElement(cktIn,cktElem,elePort,opFreq,Z0)` inserts a new circuit with unit element `unitEle` into a circuit object `cktIn` at a given reference frequency `opFreq` and characteristic impedance `Z0`. The `cktElem` and `elePort` together determine the position of insertion of the unit element.

Examples

Insert Unit Elements on Circuit

Create a lowpass LC-Pi Chebyshev filter with a passband frequency of 1 GHz, a passband attenuation of 0.5 dB, and a filter order of 5.

```
Fp = 1e9;
Ap = 0.5;
Ord = 5;
r = rffilter(FilterType="Chebyshev",ResponseType="Lowpass",Implementation="LC Pi",PassbandFreque
    Fp,PassbandAttenuation=Ap,FilterOrder=Ord);
```

Convert the lumped elements of the RF filter to a distributed element using Richards' transformation.

```
ri = richards(r,1e9)
```

```
ri =
  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-'}
```

Insert a unit element at the edges of the circuit `ri` and the first circuit element `C_tx` at port `1`, which operates at 1 GHz and has a characteristic impedance of 50 ohms.

```
unitEle = insertUnitElement(ri,'C_tx',1,1e9,50)
```

```
unitEle =
  circuit: Circuit element

    ElementNames: {1x6 cell}
```

```
        Elements: [1x6 txlineElectricalLength]
           Nodes: [0 1 2 3 4 5 6 7]
            Name: 'unnamed'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Insert a unit element at the edges of the circuit `unitEle` and the last circuit element `C_2_tx` at port `2`, which operates at 1 GHz and has a characteristic impedance of 50 ohms.

```
unitEle = insertUnitElement(unitEle,'C_2_tx',2,1e9,50)
```

```
unitEle =
  circuit: Circuit element

    ElementNames: {1x7 cell}
        Elements: [1x7 txlineElectricalLength]
           Nodes: [0 1 2 3 4 5 6 7 8]
            Name: 'unnamed'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Apply Kuroda's transformation on the group of two elements.

```
outObj =  kuroda(unitEle,'C_tx_p1_elem_UE','C_tx')
```

```
outObj =
  circuit: Circuit element

    ElementNames: {1x7 cell}
        Elements: [1x7 txlineElectricalLength]
           Nodes: [0 1 2 3 4 5 6 7 8]
            Name: 'unnamed'
        NumPorts: 2
       Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Input Arguments

`cktIn` — RF circuit object

`circuit` object

RF circuit object, specified as a `circuit` object. Before inserting the unit element you must first apply Richards' transformation.

`cktElem` — Circuit element

index in circuit | element names

Circuit element, specified as an index in the circuit or element names. The function inserts the unit element in a port of the circuit element you specify.

`elePort` — Port number

positive scalar

Port number of the circuit element for inserting the unit element, specified as a positive scalar.

`opFreq` — Operating frequency
positive scalar

Operating frequency at which the unit element is defined with “LineLength” on page 1-0 `= pi/4` radians, specified as a positive scalar.

`Z0` — Characteristic impedance
positive scalar

Characteristic impedance of the unit element, specified as a positive scalar.

Output Arguments

`unitEle` — Unit element
`cktIn` object

Unit element, returned as a `cktIn` object. The output is a circuit similar to `cktIn`, but with a unit element inserted.

The unit element is placed at the port `elePort` of the element of the circuit, `cktElem`. If `cktElem` is empty, `0` or `numel(INOBJ.Elements)+1`, the unit element is inserted at a port of the circuit `cktIn` itself.

Version History

Introduced in R2021b

See Also

`richards` | `kuroda` | `txlineElectricalLength`

Topics
“Richards-Kuroda Workflow for RF Filter Circuit”

realize

Realize circuit containing electrical-length-based transmission lines using microstrip transmission lines

Syntax

```
outObj = realize(inObj,implObj)
```

Description

`outObj = realize(inObj,implObj)` realizes a circuit containing the electrical-length-based transmission lines `inObj` using a microstrip transmission line `implObj`.

Note To realize a circuit containing electrical-length based transmission lines, you must have an RF PCB Toolbox license.

Examples

Realize Electrical-Length Based Transmission Lines

Realize a circuit containing electrical-length based transmission lines into a microstrip transmission line on an FR4 board with copper cladding.

```
inObj = txlineElectricalLength;
implObj = txlineMicrostrip('Height',0.0015748,'EpsilonR',4.6,...
    'LossTangent',0.026,'SigmaCond',59600000,'Thickness',3.5560e-05);
outObj = realize(inObj,implObj)
```

```
outObj =
  txlineMicrostrip: Microstrip element

           Name: 'txlMs_of_ElectricalLength'
          Width: 0.0029
         Height: 0.0016
      Thickness: 3.5560e-05
       EpsilonR: 4.6000
    LossTangent: 0.0260
      SigmaCond: 59600000
     LineLength: 0.0202
    Termination: 'NotApplicable'
       StubMode: 'NotAStub'
       NumPorts: 2
      Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Input Arguments

`inObj` — Input object
`txlineElectricalLength` object | `circuit` object

Input object, specified as a `txlineElectricalLength` object or a `circuit` object with at least one `txlineElectricalLength` object in its element.

`implObj` — Implementation object
`txlineMicrostrip` object

Implementation object, specified as a `txlineMicrostrip` object.

Output Arguments

`outObj` — Output object
`txlineMicrostrip` object

Output object, returned as a `txlineMicrostrip` object.

Version History

Introduced in R2021b

See Also

`richards` | `kuroda` | `insertUnitElement` | `txlineElectricalLength` | `txlineMicrostrip`

Topics
“Richards-Kuroda Workflow for RF Filter Circuit”

abcd

Construct state-space matrices from `rational` object

Syntax

```
[A,B,C,D,sharedpoles] = abcd(fit)
```

Description

`[A,B,C,D,sharedpoles] = abcd(fit)` constructs real diagonal form `A`, `B`, `C`, and `D` state-space matrices from input `fit`, a `rational` object. This function also indicates whether or not each element of `fit` has the same poles using `sharedpoles` output argument.

Examples

Construct State-Space Matrices from `rational` Object

Create S-Parameters from the file named `passive.s2p`.

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

Perform rational fitting of the S-parameters.

```
fit = rational(S);
```

Construct state-space matrices from rational object, `fit`.

```
[A,B,C,D,sharedpoles] = abcd(fit)
```

```
A =
   1.0e+11 *

   (1,1)      -0.0008
   (2,1)      -0.0011
   (1,2)       0.0011
   (2,2)      -0.0008
   (3,3)      -4.0560
   (4,4)      -0.8359
   (5,5)      -0.5389
   (6,6)      -0.1148
   (7,7)      -0.0173
   (8,8)      -0.0008
   (9,9)      -0.0008
  (10,9)      -0.0011
   (9,10)      0.0011
  (10,10)     -0.0008
  (11,11)     -4.0560
  (12,12)     -0.8359
  (13,13)     -0.5389
  (14,14)     -0.1148
  (15,15)     -0.0173
```

```
  (16,16)     -0.0008


B =
   (1,1)       1.4142
   (3,1)       1.0000
   (4,1)       1.0000
   (5,1)       1.0000
   (6,1)       1.0000
   (7,1)       1.0000
   (8,1)       1.0000
   (9,2)       1.4142
  (11,2)       1.0000
  (12,2)       1.0000
  (13,2)       1.0000
  (14,2)       1.0000
  (15,2)       1.0000
  (16,2)       1.0000


C = 2×16
10^11 ×

    0.0001    0.0001   -2.8008    0.6205    0.0907    0.0831   -0.0015   -0.0006   -0.0001   -0.0
   -0.0001   -0.0001   -3.1236    2.4641   -0.7827   -0.0750    0.0010    0.0007    0.0001    0.0


D = 2×2

     0     0
     0     0


sharedpoles = logical
   1
```

Input Arguments

`fit` — Rational fit

`rational` object | `rfmodel.rational` object

Rational fit, specified as a `rational` or `rfmodel.rational` object.

Output Arguments

`A,B,C,D` — State-space matrices

array of scalar numbers

State-space matrices, returned as an array of scalar numbers. The size of the state-space matrix elements are

- `A` — *M*-by-*M* array.
- `B` — *M*-by-`NumPorts` array.

- `C` — `NumPorts`-by-*M* array.
- `D` — `NumPorts`-by-`NumPorts` array.

  where, *M* is `NumPoles` × `NumPorts` and `NumPoles` and `NumPorts` are number of poles and ports, respectively, derived from `fit`.

`sharedpoles` — Shared poles
`1` | `0`

Shared poles indicating whether or not each element of `fit` has the same poles, returned as `1` or `0` of data type `logical`.

Data Types: `logical`

Version History
Introduced in R2020a

See Also
`zpk` | `pwlresp`

load_system

Load RF Blockset model to memory

Syntax

```
load_system(rfs)
```

Description

`load_system(rfs)` loads the RF Blockset model associated with the RF system `rfs` to memory without opening the model in Simulink® Editor. You must create the RF system using the `rfsystem` System object™

Examples

Load RF Blockset Model into Memory

Create a fifth-order bandpass RF filter and an amplifier with the gain of 3 dB.

```
f1 = rffilter('ResponseType','Bandpass','FilterOrder',5,'PassbandFrequency',[4.85 5.15]*1e9);
a1 = amplifier('Gain',3,'NF',1.53,'OIP3',35);
```

Create an `rfbudget` object using the two elements at an input frequency of 5 GHz, an available input power of – 30 dBm, and a bandwidth of 200 MHz.

```
rfb = rfbudget([f1 a1],5e9,-30,200e6);
```

Create an RF system using `rfsystem` System object.

```
rfs = rfsystem(rfb);
```

Open the RF Blockset model.

```
load_system(rfs)
```

Input Arguments

`rfs` — RF system

`rfsystem` object

RF system, specified as an `rfsystem` object.

Version History

Introduced in R2022a

See Also

`rfsystem` | `open_system` | `save_system` | `close_system` | `hide_system`

rfplot

Plot mixer spurs of IMT mixer

Syntax

```
rfplot(imt,frequency)
rfplot(ax,imt,frequency)
```

Description

`rfplot(imt,frequency)` plots the mixer spurs response of the IMT mixer object `imt` at the frequency specified in `frequency`. The function plots the mixer spurs on the current axes, with the frequency in Hz on the X-axis and the input power in dBm on the Y-axis.

`rfplot(ax,imt,frequency)` plots the mixer spurs on the axes specified in `ax` instead of the current axes. Return the current axes using the `gca` function.

Examples

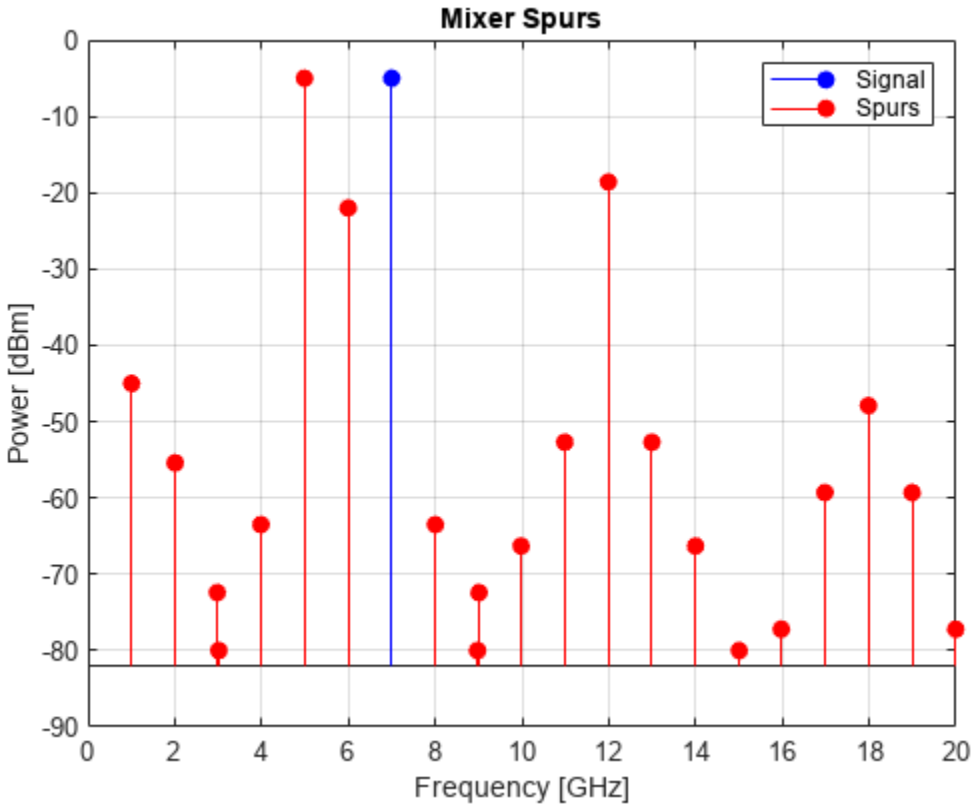
Plot Mixer Spur of IMT mixer

Create an IMT mixer with the LO frequency of 5995 MHz and with the IMT spurs.

```
imt = mixerIMT(LO=5995e6,IMT=...
     [99 17 13.6 42.9; 40 0 47.7 54.3; 50.4 58.5 61.3 72.2; 75 67.4 75 99]);
```

Plot the mixer spur at an input signal frequency of 1005 MHz.

```
rfplot(imt,1005e6)
```



Plot IMT Spurs in Tabs

Create an IMT mixer with 4-by-4 IMT spur data.

```
imt = mixerIMT(LO=5995e6,IMT=...
     [99 17 11.6 42.9; 40 0 44.7 54.3; 51.4 58.5 61.3 72.2; 75 67.4 73 99]);
```

Create an IMT mixer with 3-by-3 IMT spur data.

```
imt2 = mixerIMT(LO=5995e6,IMT=...
     [99 99 80; 99 0 99; 99 80 99]);
```

Type these commands at the command line to create a `rfplot` figure with two tabs. Add axes to each tab by specifying the parent container for each axis. Plot the IMT mixer with the 4-by-4 IMT spur data in the first tab and the IMT mixer with the 3-by-3 IMT spur data in the second tab.

```
figure
tab1 = uitab('Title','4-by-4 IMT spur data');
ax3 = axes(tab1);
rfplot(ax3,imt,frequency=1005e6, powerinput=-5)

tab2 = uitab('Title','3-by-3 IMT spur data');
ax4 = axes(tab2);
rfplot(ax4,imt2,frequency=1005e6, powerinput=-5)
```

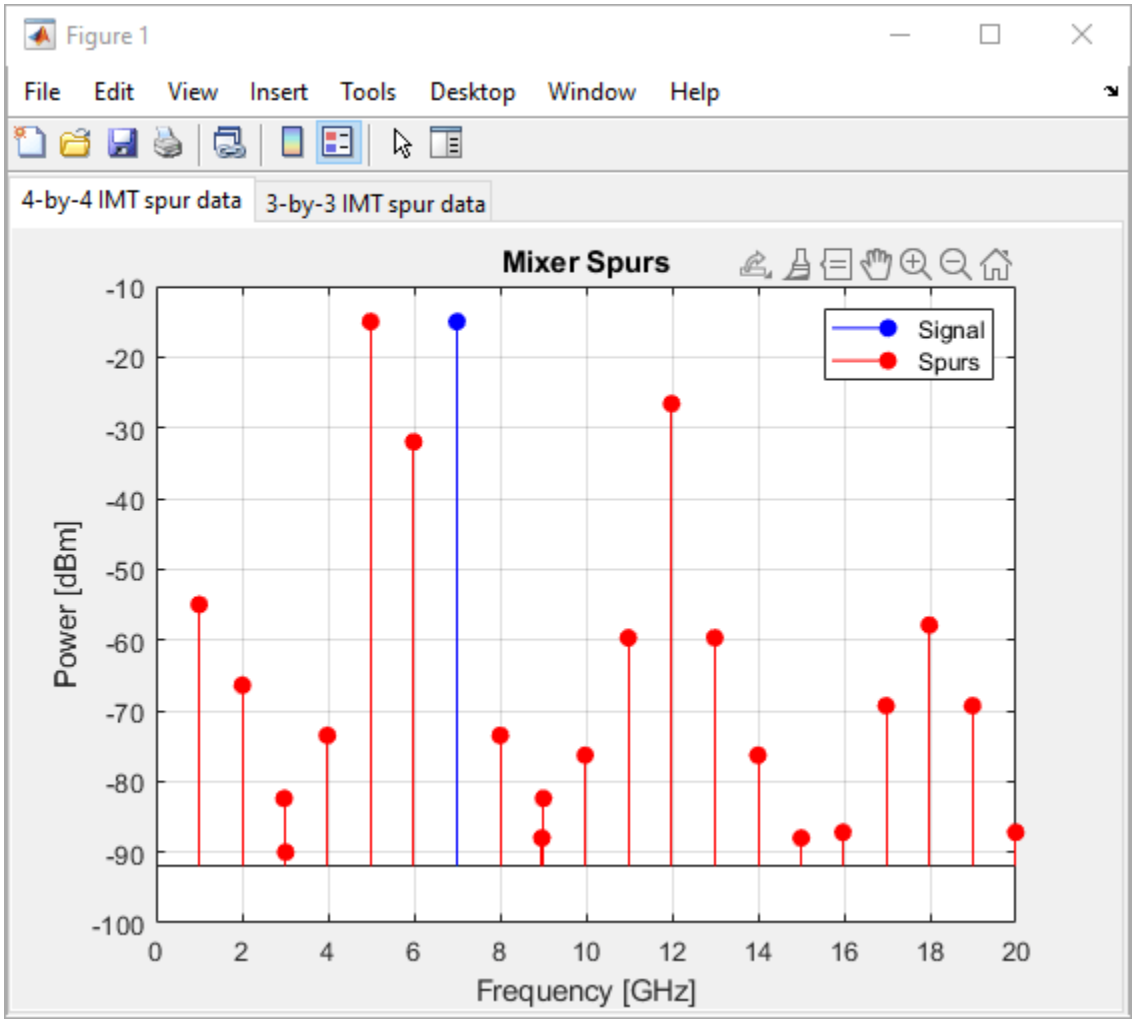
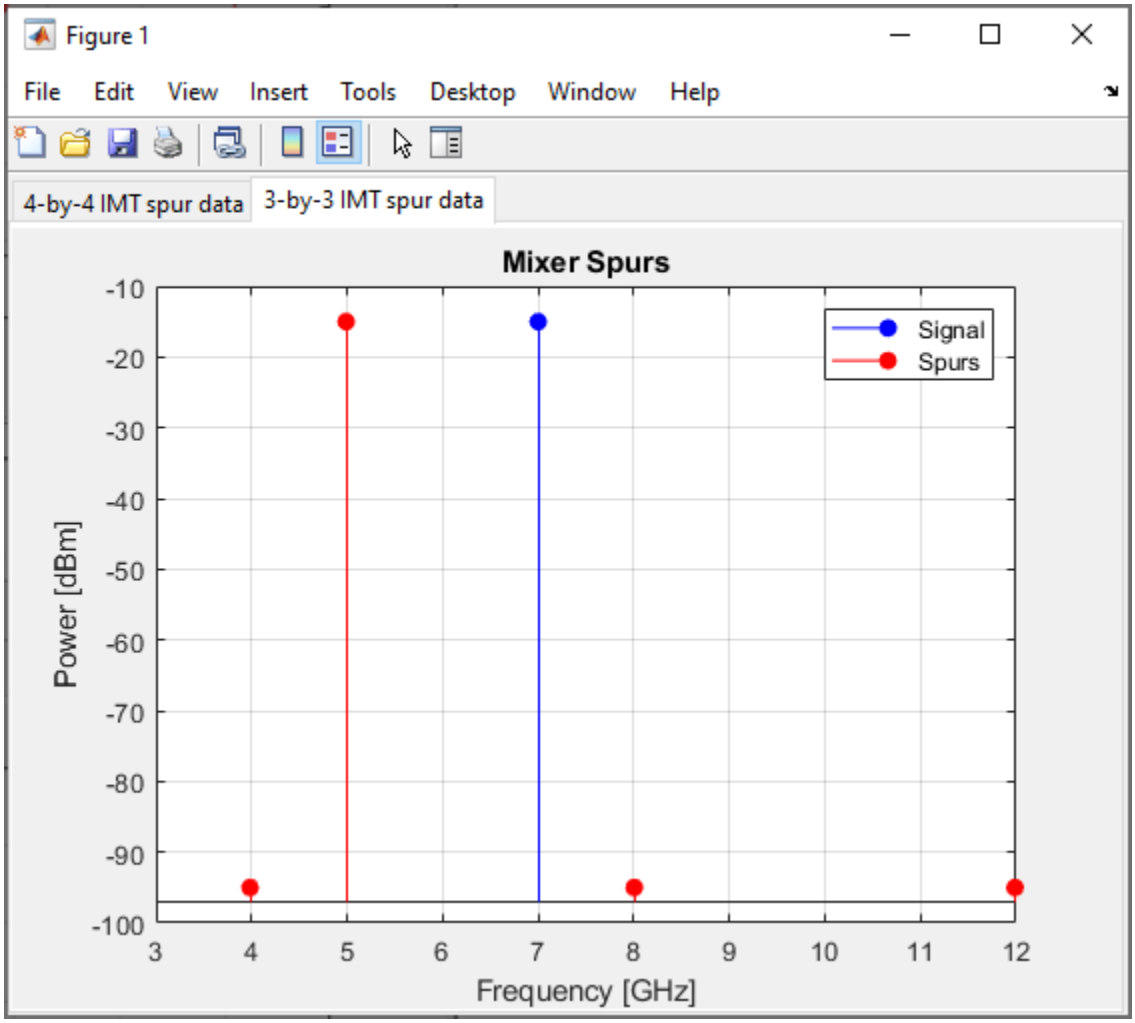


Figure 1
File Edit View Insert Tools Desktop Window Help
4-by-4 IMT spur data
3-by-3 IMT spur data
Mixer Spurs
Signal
Spurs
Power [dBm]
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
0 2 4 6 8 10 12 14 16 18 20
Frequency [GHz]



Input Arguments

`imt` — IMT mixer object

`mixerIMT` object

IMT mixer object, specified as a `mixerIMT` object.

Data Types: `object`

`frequency` — Input frequency of IMT mixer object

positive scalar

Input frequency of the IMT mixer object, specified as a positive scalar in Hz.

Data Types: `double`

`ax` — Axes to plot mixer spurs

`axex` object

Axes to plot mixer spurs, specified as an `axes` object.

Tips

In addition to plotting the mixer spurs, you can also use the `rfplot` function to:

- Plot the `S-parameter` data.
- Plot the `cumulative RF budget` result versus the cascade input frequency.
- Plot the `input reflection coefficient and transducer gain` of a matching network.

Version History

Introduced in R2022a

See Also

`mixerIMT`

delete

Delete circuit object and decouple its elements

Syntax

```
delete(obj)
```

Description

`delete(obj)` deletes the circuit object `obj` and decouples its elements. You can use the decoupled elements in other circuit objects.

Examples

Delete Circuit Object

Create a circuit.

```
hckt = circuit('new_circuit');
```

Add a capacitor to the circuit.

```
hC = capacitor(1e-12);
add(hckt,[1 2],hC)
```

Display the parent path.

```
hC.ParentPath
```

```
ans =
'new_circuit'
```

Delete the circuit object and display the parent path.

```
delete(hckt)
hC.ParentPath
```

```
ans =

  0x0 empty char array
```

Reuse the capacitor `hc`.

```
hckt = circuit('circuit_1');
add(hckt,[1 2],hC)
```

Input Arguments

obj — RF circuit

`circuit` object

RF circuit, specified as a `circuit` object.

Version History

Introduced in R2022a

See Also

Methods — Alphabetical List

addMixer

Add an additional mixer/RF specification

Syntax

`addMixer(hif,newimt,newrfcf,newrfbw,newmixtype,newifbw)`

Description

`addMixer(hif,newimt,newrfcf,newrfbw,newmixtype,newifbw)` adds a mixer to a multiband transmitter or receiver object `hif` as part of an intermediate-frequency (IF) planning analysis workflow.

Examples

Add Two Mixers to System

Set up the object

```
h = OpenIF('IFLocation','MixerOutput');
```

Add two mixers to the system

```
IMT1 = [99 0 21 17 26; 11 0 29 29 63; ...
        60 48 70 65 41; 90 89 74 68 87; 99 99 95 99 99];
addMixer(h,IMT1,2400e6,100e6,'low',50e6)

IMT2 = [99 0 9 12 15; 20 0 26 31 48; ...
        55 70 51 70 53; 85 90 60 70 94; 96 95 94 93 92];
addMixer(h,IMT2,3700e6,150e6,'high',50e6)
```

Input Arguments

`hif` — OpenIF object

object handle

OpenIF object, specified as an object handle,

`newimt` — Intermodulation table

matrix

Intermodulation table, specified as a matrix of size 2-by-2 or greater with each element unit in dB. Values in the matrix are intermodulation levels. Positive values represent greater attenuation.

Columns of the matrix represent integer multiples of the local oscillator (LO) of the mixer, where column one is `0*LO`, column 2 is `1*LO`, etc. Rows of the matrix represent multipliers for the input frequency to the mixer.

Example: `[99 0 21 17 26; 11 0 29 29 63; ... 60 48 70 65 41; 90 89 74 68 87; 99 99 95 99 99];`

Data Types: `double`

`newrfcf` — RF center frequency
scalar

RF center frequency, specified as a scalar in Hz.

Example: `2400e6`

Data Types: `double`

`newrfbw` — RF bandwidth
scalar

RF bandwidth, specified as a scalar in Hz.

Example: `100e6`

Data Types: `double`

`newifbw` — IF bandwidth
scalar

IF bandwidth, specified as a scalar in Hz.

Example: `50e6`

Data Types: `double`

`newmixtype` — Mixer type
`'sum'` | `'diff'` | `'low'` | `'high'`

Mixer type, specified as `'sum'`, `'diff'`, `'low'`, `'high'`. If the `IFLocation` property in `OpenIF` object is set to `'MixerInput'`, then the mixer type is `'sum'` or `'diff'`. If the `IFLocation` property in `OpenIF` object is set to `'MixerOutput'`, then the mixer type is `'low'` or `'high'`

Example: `'high'`

Data Types: `char`

Version History

Introduced in R2011b

See Also

`OpenIF`

Topics
“Finding Free IF Bandwidths”

analyze

Package: `rfckt`

Analyze RFCKT object in frequency domain

Syntax

```
analyze(rfcktobject,frequency)
analyze(rfcktobject,frequency,zl,zs,zo,aperture)
analyze(rfcktobject,frequency,condition,value)
```

Description

`analyze(rfcktobject,frequency)` calculates the following rfckt data at the specified frequency values:

- Circuit network 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

`analyze(rfcktobject,frequency,zl,zs,zo,aperture)` calculates the circuit data specified frequency values with optional arguments such as load impedance, source impedance, reference impedance and aperture.

`analyze(rfcktobject,frequency,condition,value)` calculates the circuit data at the specified frequency values and operating conditions for the `circuitdata` object. For more information to set conditions and values, see `setop` function.

Note When you specify condition/value pairs, the `analyze` method changes the object's values to match your specification.

Examples

Analyze Network Object

Create and analyze a two-wire network object.

```
 tx1=rfckt.twowire('Radius',7.5e-4);
 analyze(tx1,1.9e9)
```

```
ans =
   rfckt.twowire with properties:

            Radius: 7.5000e-04
        Separation: 0.0016
               MuR: 1
          EpsilonR: 2.3000
       LossTangent: 0
         SigmaCond: Inf
        LineLength: 0.0100
          StubMode: 'NotAStub'
       Termination: 'NotApplicable'
              nPort: 2
    AnalyzedResult: [1x1 rfdata.data]
              Name: 'Two-Wire Transmission Line'
```

Analyze RF Amplifier at Different Reference Impedances

This example shows how to analyze an RF amplifier at different reference impedances.

Assign the load and the source impedances.

```
zl = 50 - 50*1i;
zs = 200 + 50*1i;
```

Create two amplifier circuits with the same Touchstone® file.

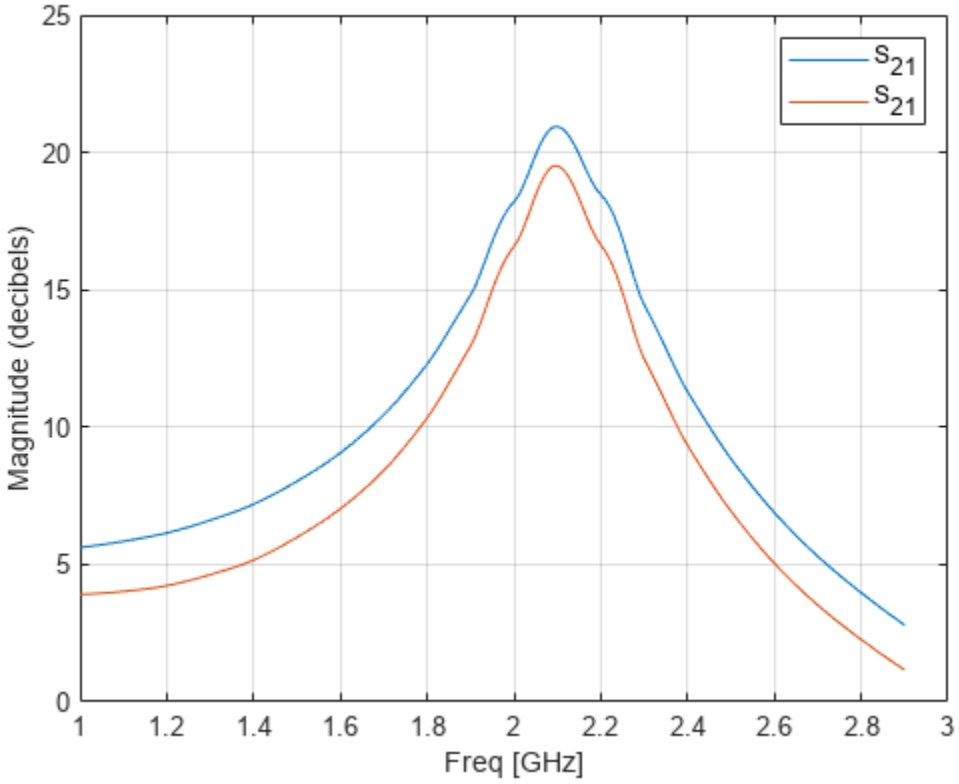
```
circuit50 = read(rfckt.amplifier,'default.s2p');
circuit75 = read(rfckt.amplifier,'default.s2p');
```

Analyze the amplifier circuits at two different reference impedance, `50` and `75` ohms.

```
analyzed_circuit50 = analyze(circuit50, circuit50.NetworkData.Freq, zl, zs,50);
analyzed_circuit75 = analyze(circuit75, circuit50.NetworkData.Freq, zl, zs,75);
```

Plot S21 for the two amplifier circuits.

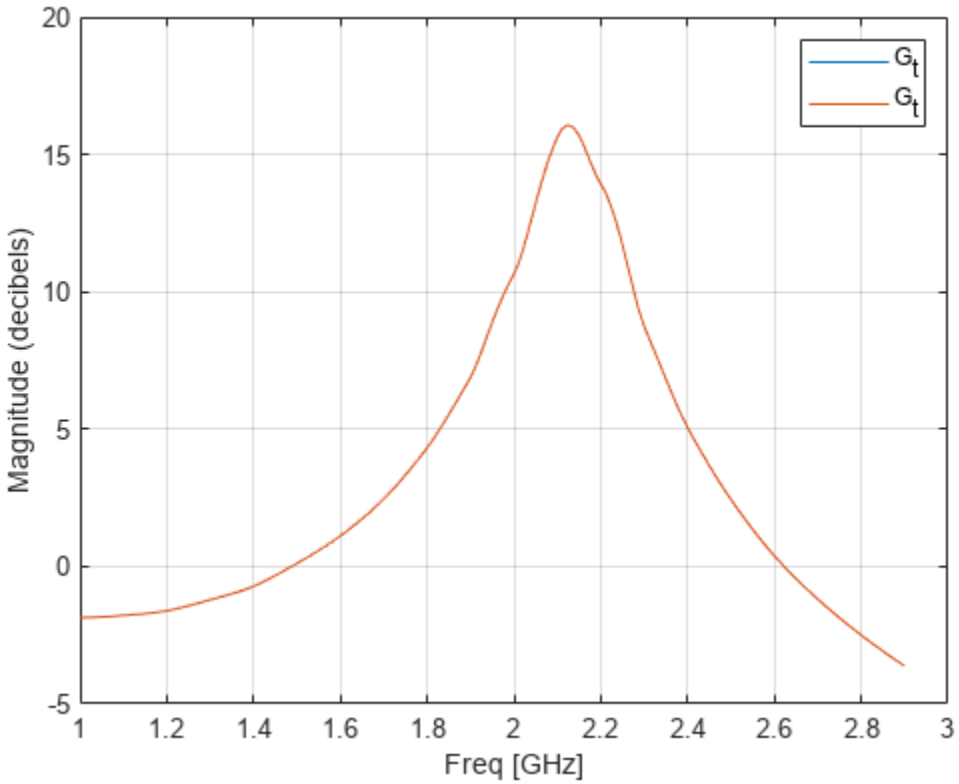
```
figure(30);
plot(analyzed_circuit50,'S21')
hold on;
plot(analyzed_circuit75,'S21')
```



Note that in this example two amplifier circuits derived from a same touchstone file at two different reference impedances produce two different S21 plot. This is because, the S-parameters are only dependent on the reference impedance, `z0`, and not on the source impedance, `zs` or the load impedance, `zl`.

The analyze function stores `zs` and `zl` in the amplifier and these impedances are used when a `zs` and `zl` dependent parameter is calculated. For example, plot transducer gain, `Gt` of the two amplifier circuits.

```
figure(20);
plot(analyzed_circuit50,'Gt')
hold on;
plot(analyzed_circuit75,'Gt')
```



Note that `Gt` is dependent on `zs`, `zl`, and `z0`. Hence for the two amplifier circuits derived from a same touchstone file with same `zs` and `zl` at `z0` of `50` and `75` ohms yields the same `Gt`.

Input Arguments

`rfcktobject` — RFCKT object

circuit object (default)

RFCKT object to analyze, specified as a object handle.

Example: `amp = rfckt.amplifier;analyze(amp,frequency)` Analyzes the `rfckt.amplifier` object with handle `amp` at the specified frequency.

Data Types: `char` | `string`

`frequency` — Simulation frequency

vector

Simulation frequencies, specified as a vector in hertz.

Example: `1.9e9`

Data Types: `double`

`zl` — Load impedance

`50` (default) | scalar

Load impedance, specified as a scalar in ohms.

Example: `40`

Data Types: `double`

`zs` — Source impedance
`50` (default) | scalar

Source impedance, specified as a scalar in ohms.

Example: `40`

Data Types: `double`

`zo` — Reference impedance of S-parameters
`50` (default) | real positive scalar | real positive vector

Reference impedance of S-parameters, specified as a real positive scalar or real positive vector in ohms. The length of this vector must be same as of “frequency” on page 3-0 argument.

Example: `40`

Data Types: `double`

`aperture` — Value to determine two closely spaced frequencies at each simulation frequency
positive scalar (default) | vector

Value to determine two closely spaced frequencies at each simulation frequency for the calculation of group delay, specified as a positive scalar or a vector of same length as simulation frequencies. If the `aperture` is not specified, it will be determines based on the simulation frequencies.

Example: `40`

Data Types: `double`

Version History

Introduced before R2006a

See Also

`calculate` | `circle` | `extract` | `listformat` | `listparam` | `loglog` | `plot` | `plotyy` | `getop` | `polar` | `semilogx` | `semilogy` | `smith` | `write` | `getz0` | `read` | `restore` | `getop`

calculate

Calculate specified parameters for rfckt objects or rfdata objects

Syntax

```
[data,parameters,frequency] = calculate(rfdataobject,
parameter1,...,parameterN,format)
[data,parameters,frequency] = calculate(rfcktobject,
parameter1,...,parameterN,format)
```

Description

`[data,parameters,frequency] = calculate(rfdataobject, parameter1,...,parameterN,format)` calculates the required parameters of the rfdata.data object, `rfdataobject` and returns them in a cell array, `data`.

`[data,parameters,frequency] = calculate(rfcktobject, parameter1,...,parameterN,format)` calculates the required parameters of the rfckt object, `rfcktobject` and returns them in a cell array, `data`.

Examples

Calculate S-Parameters of Transmission Line

Analyze a general transmission line of impedance, 50 ohms, phase velocity of 299792458 m/s, and line length of 0.01 meters for frequencies 1.0 GHz to 3.0 GHz.

```
trl = rfckt.txline;
f = 1e9:1.0e7:3e9;
analyze(trl,f)
```

```
ans =
   rfckt.txline with properties:

        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: [1x1 rfdata.data]
              Name: 'Transmission Line'
```

Calculate the S11 and S22 parameters in dB.

```
[data,params,freq] = calculate(trl,'S11','S22','dB')
```

```
data=1×2 cell array
    {201x1 double}    {201x1 double}


params = 1x2 cell
    {'S_{11}'}    {'S_{22}'}


freq = 201×1
10^9 ×

    1.0000
    1.0100
    1.0200
    1.0300
    1.0400
    1.0500
    1.0600
    1.0700
    1.0800
    1.0900
      ⋮
```

Input Arguments

`rfdataobject` — RF data

`rfdata.data` object (default)

RF data, specified as a handle of an `rfdata.data` object.

Example: `rfdataobject = rfdata.data;[data,parameter,frequency] = calculate[rfdataobject]` calculates and returns the cell array of data for the rfdata.data object, `rfdataobject`.

Data Types: `char` | `string`

`rfcktobject` — RFCKT element

rfckt object (default)

RFCKT element, specified as a handle of an rfckt object.

Example: `rfcktobject = rfckt.amplifier;[data,parameter,frequency] = calculate[rfcktobject]` calculates and returns the cell array of data for the rfckt amplifier object, `rfcktobject`.

Data Types: `char` | `string`

`parameter1,...,parameterN` — Parameters of an rfckt object or rfdata.data object

character vector (default) | string

Parameters of an rfckt object or rfdata.data object, specified as a character vector or string. Use the `listparameter` function to list the parameters of the specified rfckt object or rfdata.data object.

Example: `rfcktobject = rfckt.amplifier;listparam(rfcktobject);`You can use any of the parameter from the output of `listparam` in the calculate function.

Data Types: `char` | `string`

`format` — Format of output data
character vector (default) | string

Format of output data, specified as a character vector or string. Use the `listformat` function to list the valid formats of the parameter values of the specified rfckt object or rfdata.data object.

Example: `rfcktobject = rfckt.amplifier;listformat(rfcktobject,parameter);`Lists the format of the specified `parameter` of the `rfcktobject`. You can then use this format value in the `calculate` function.

Example: Specify format as `Real` to compute the real part of the selected parameter. Specify format as `none` to return the parameters values unchanged.

Data Types: `char` | `string`

Output Arguments

`data` — Data of the rfckt element or rfdata.data object
n-element cell array

Data of the rfckt element or rfdata.data object, returned as an n-element cell array.

`parameters` — Name of the parameters in `data` output
n-element cell array

Name of the parameters in `data` output, returned as an n-element cell array.

`frequency` — Frequencies
vector

Frequencies at which the parameters are known, returned as vector.

Version History

Introduced before R2006a

See Also

`analyze` | `extract` | `getz0` | `listformat` | `listparam` | `loglog` | `plot` | `plotyy` | `polar` | `read` | `restore`

extract

Extract specified network parameters from rfckt object or data object

Syntax

```
[outmatrix, frequencies] = extract(rfcktobj,outtype,z0)
[outmatrix, frequencies] = extract(rfdataobj,outtype,z0)
```

Description

`[outmatrix, frequencies] = extract(rfcktobj,outtype,z0)` extracts the network parameters of an rfckt object, `rfcktobj` and returns them in an `outmatrix`.

`[outmatrix, frequencies] = extract(rfdataobj,outtype,z0)` extracts the network parameters of a data object, `rfdataobj` and returns them in an `outmatrix`.

Examples

Extract Network Parameters of RFCKT Object

Extract the ABCD-parameters for an rfckt.amplifier object read from `default.s2p`.

```
amp = read(rfckt.amplifier,'default.s2p');
[outmatrix,freq] = extract(amp,'ABCD_parameters');
```

Input Arguments

`rfcktobj` — RFCKT object

object handle

RFCKT object, specified as an object handle.

Example: `amp = rfckt.amplifier;[outmatrix,freq] = extract(amp,'ABCD_parameters');`. Extracts the ABCD-parameters of an RFCKT amplifier object.

Data Types: `char` | `string`

`rfdataobj` — Data object

object handle

Data object, specified as an object handle.

Data Types: `char` | `string`

`outtype` — Type of network parameters to extract

`'S-Parameters'` | `'Y-Parameters'` | `'Z-Parameters'` | `'H-Parameters'` | `'G-Parameters'` | `'T-Parameters'` | `'ABCD-Parameters'`

Type of network parameters to extract, specified as `'S-Parameters'`, `'Y-Parameters'`, `'Z-Parameters'`, `'H-Parameters'`, `'G-Parameters'`, `'T-Parameters'`, and `'ABCD-Parameters'`.

Example: `amp = rfckt.amplifier;[outmatrix,freq] = extract(amp,'ABCD_parameters');`. Extracts the ABCD-parameters of an RFCKT amplifier object.

Data Types: `char` | `string`

`z0` — Reference impedance when `outtype` is `'S-Parameters'`
`50` | positive scalar integer

Reference impedance when `outtype` is `'S-Parameters'`, specified as a positive scalar integer.

Example: `40`

Data Types: `double`

Output Arguments

`outmatrix` — Network parameters extracted from an rfckt or data object
2-by-2-by-*N* matrix

Network parameters extracted from an rfckt or data object, returned as a 2-by-2-by-*N* matrix.

`frequencies` — Network parameter frequencies
vector

Network parameter frequencies, returned as a vector.

Version History

Introduced before R2006a

See Also

`analyze` | `calculate` | `getz0` | `listformat` | `listparam` | `loglog` | `plot` | `plotyy` | `polar` | `read` | `restore`

freqresp

Frequency response of rational object and `rationalfit` function object

Syntax

```
[response, outputfreq] = freqresp(fit,inputfreq)
```

Description

`[response, outputfreq] = freqresp(fit,inputfreq)` calculates the frequency response, `response` of the `fit` of a `rationalfit` function object or a `rational` object at the specified input frequencies, `inputfreq`.

Examples

Frequency Response of Data Stored In File

Create a `sparameters` object from a file and use `rfparam` to extract the S_{21}parameters.

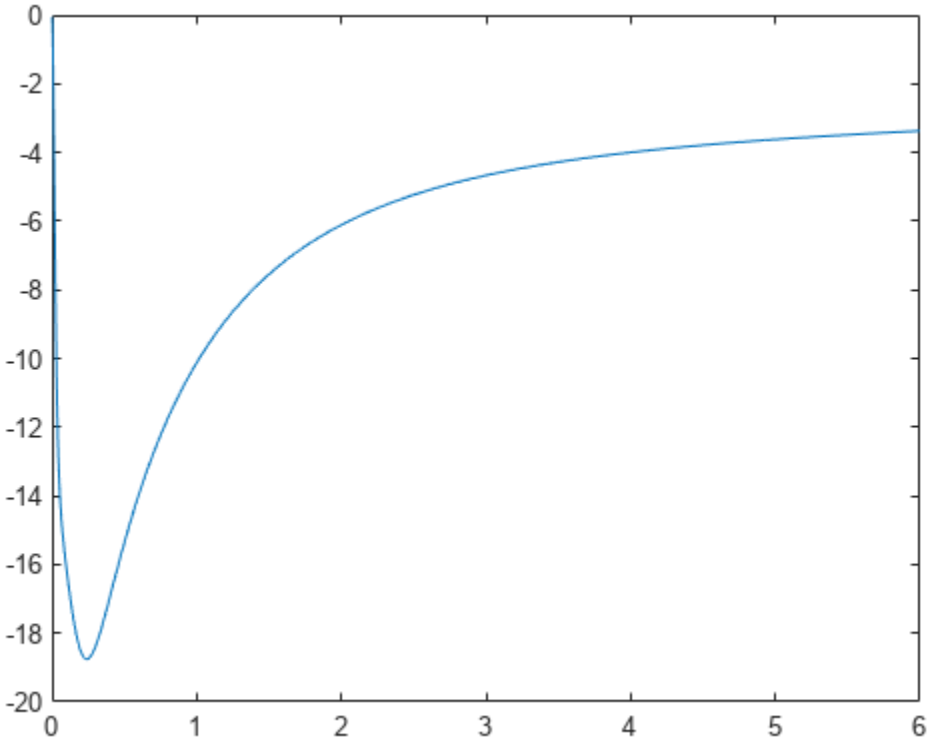
```
S = sparameters('passive.s2p');
S21 = rfparam(S,2,1);
```

Fit a rational function to the data by using `rationalfit`.

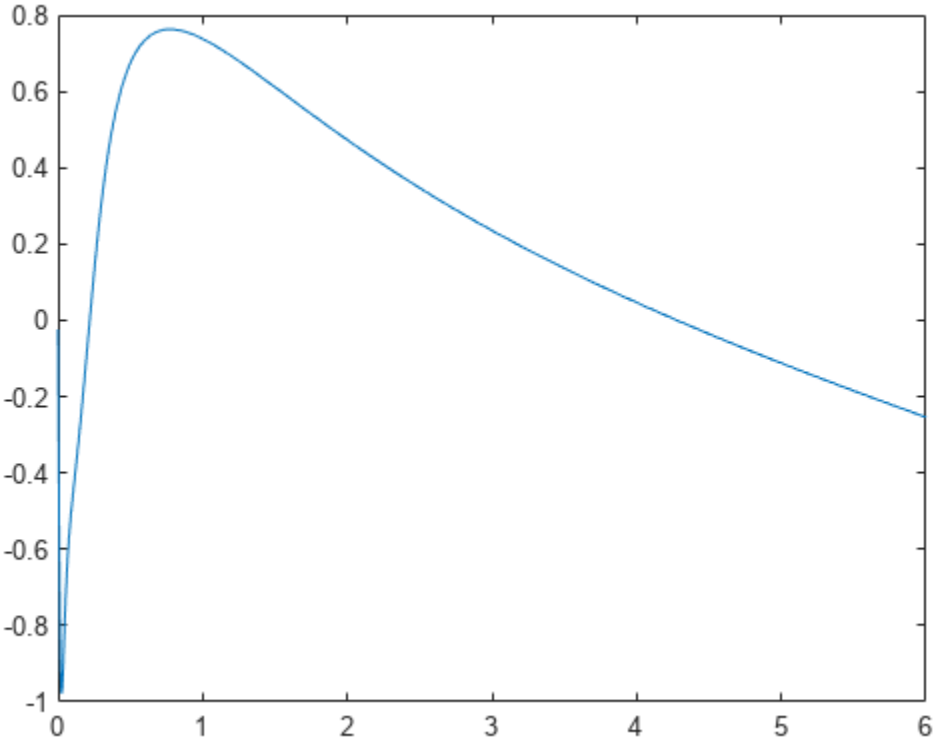
```
freq = S.Frequencies;
fit_data = rationalfit(freq,S21);
```

Compute the frequency response by using the `freqresp` method and plot the magnitude and angle of the frequency response.

```
[resp,freq] = freqresp(fit_data,freq);
plot(freq/1e9,20*log10(abs(resp)))
```




```
plot(freq/1e9,unwrap(angle(resp)))
```



Frequency Response of S-Parameters Object

Perform rational fitting on a S-parameters object by using `rational` object.

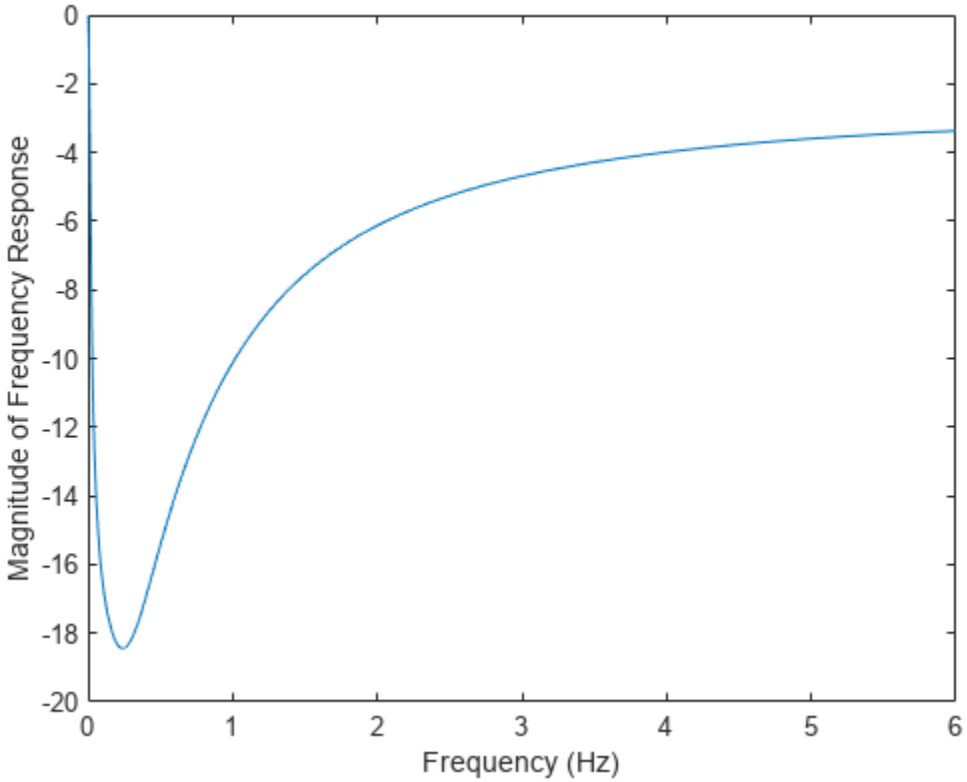
```
S = sparameters('passive.s2p');
fit_data = rational(S);
```

Compute the frequency response by using the `freqresp` function.

```
freq = S.Frequencies;
[resp,freq] = freqresp(fit_data,freq);
```

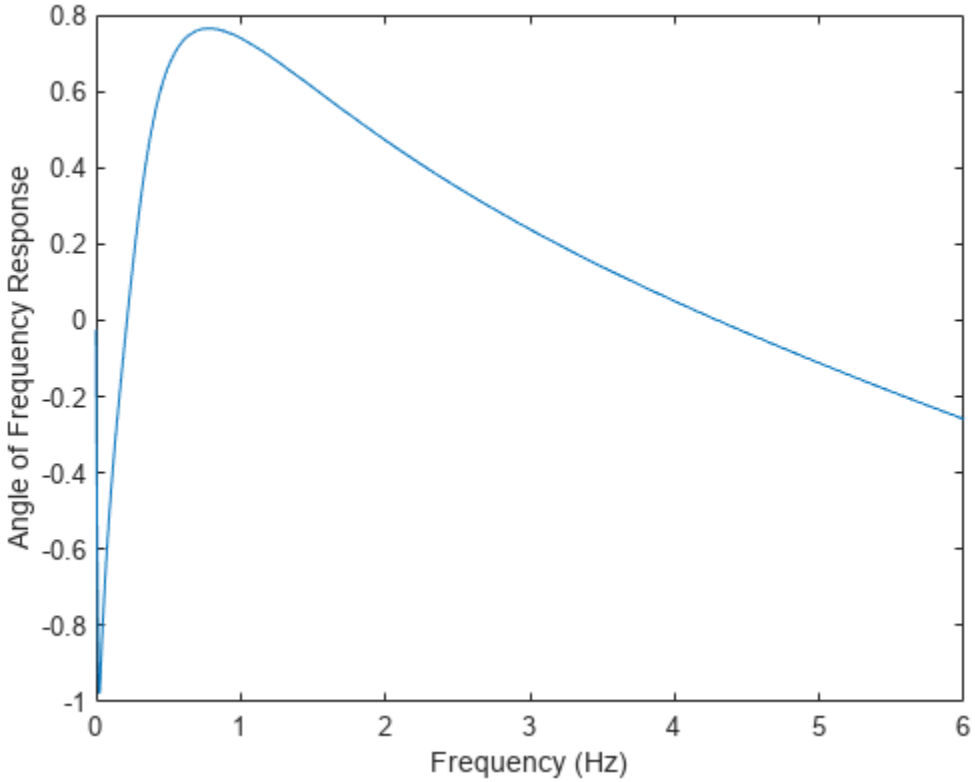
Plot the magnitude of the frequency response of the S_{21}data.

```
plot(freq/1e9,20*log10(abs(squeeze(resp(2,1,:)))))
xlabel('Frequency (Hz)');
ylabel('Magnitude of Frequency Response');
```



Plot the angle of the frequency response of the S_{21}data.

```
plot(freq/1e9,unwrap(angle(squeeze(resp(2,1,:)))))
xlabel('Frequency (Hz)');
ylabel('Angle of Frequency Response');
```



Input Arguments

`fit` — Rational fit object

`rfmodel.rational` object | `rational` object | *M*-by-*N* array

Rational fit object, specified as a `rfmodel.rational` or `rational` object. The size of this object is *M*-by-*N* array.

`inputfreq` — Input frequency

real positive vector

Input frequency to compute and plot the frequency response, specified as a vector of positive frequencies in Hz.

Data Types: `double`

Output Arguments

`response` — Computed frequency response

vector

Computed frequency response of each *M*-by-*N* fit, specified as a vector.

Data Types: `double`

`outputfreq` — Frequency values same as input frequencies
real positive vector

Frequency values same as input frequencies, returned as a real positive vector.

Data Types: `double`

Version History

Introduced before R2006a

See Also

`rfmodel.rational` | `rationalfit` | `timeresp` | `pwlresp`

Topics

“Convert Scattering Parameter to Impulse Response for SerDes System” (SerDes Toolbox)

getop

Display operating conditions

Syntax

```
getop(hobj)
```

Description

`getop(hobj)` displays the selected operating conditions for the circuit or data object, `hobj`.

Examples

Display Operating Conditions of Circuit Object

Display the operating conditions of a circuit.

```
ckt1 = read(rfckt.amplifier,'default.p2d');
getop(ckt1)
```

```
ans = 1x2 cell
    {'Bias'}    {'1.5'}
```

Input Arguments

hobj — Circuit or data object

object handle

Circuit or data object, specified as an object handle.

Data Types: `char` | `string`

Version History

Introduced before R2006a

See Also

`setop`

getz0

Calculate characteristic impedance of RFCKT transmission line object

Syntax

```
z0 = getz0(txline)
```

Description

`z0 = getz0(txline)` returns the characteristic impedance `z0`, of a transmission line object `txline`.

Examples

Get Z0 of Network Object

Create and analyze a two-wire network object.

```
 tx1=rfckt.twowire('Radius',7.5e-4)
```

```
tx1 =
   rfckt.twowire with properties:

            Radius: 7.5000e-04
        Separation: 0.0016
               MuR: 1
          EpsilonR: 2.3000
       LossTangent: 0
         SigmaCond: Inf
        LineLength: 0.0100
          StubMode: 'NotAStub'
       Termination: 'NotApplicable'
              nPort: 2
    AnalyzedResult: []
              Name: 'Two-Wire Transmission Line'
```

```
 analyze(tx1,1.9e9)
```

```
ans =
   rfckt.twowire with properties:

            Radius: 7.5000e-04
        Separation: 0.0016
               MuR: 1
          EpsilonR: 2.3000
       LossTangent: 0
         SigmaCond: Inf
        LineLength: 0.0100
          StubMode: 'NotAStub'
       Termination: 'NotApplicable'
```

```
           nPort: 2
    AnalyzedResult: [1x1 rfdata.data]
            Name: 'Two-Wire Transmission Line'
```

Find the Z0 of the two-wire object.

```
z0 = getz0(tx1)
```

```
z0 = 31.4212
```

Input Arguments

`txline` — Transmission line

`rfckt.txline` object (default)

Transmission lines object to analyze, specified as a `rfckt.txline` handle.

Example: `txline = rfckt.txline;getz0(txline)`. Calculates the characteristic impedance of the transmission line object with handle `txline`.

Data Types: `char` | `string`

Output Arguments

`z0` — Characteristic impedance of transmission lines

complex scalar

Characteristic impedance of the transmission line, returned as a complex scalar.

Data Types: `double`

Version History

Introduced before R2006a

See Also

`analyze` | `calculate` | `extract` | `listformat` | `listparam` | `loglog` | `plot` | `plotyy` | `polar` | `restore` | `semilogx` | `semilogy` | `smith` | `write`

plotyy

Plot parameters of RF circuit or RF data on X-Y plane with two Y-axes

Syntax

```
plotyy(h,circuitPara)
plotyy(h,circuitPara,format1,format2)
plotyy(h,circuitPara1_1,format1,circuitPara2_1,format2)
plotyy( ___ ,xAxisPara,xAxisFmt,opCon,opVal)
plotyy( ___ ,Name,Value)
[ax,hlines1,hlines2] = plotyy( ___ )
```

Description

`plotyy(h,circuitPara)` plots the RFCKT or RF Data object parameters on the X-Y plane with two Y-axes. You can specify multiple RFCKT or RF Data object in this syntax.

Note For all circuit objects except for those that contain data from a data file, you must perform frequency domain analysis using the `analyze` function before using `plotyy`.

`plotyy(h,circuitPara,format1,format2)` plots the RFCKT or RF Data object parameters with primary and secondary formats.

“Determining Parameter Formats” table lists primary and secondary formats for the all RFCKT or RF data object parameters. You can also use `listformat` to get a list of valid parameters for circuit or data objects.

`plotyy(h,circuitPara1_1,format1,circuitPara2_1,format2)` plots RF data as follows:

- Plot `circuitPara1_1` using `format1` in the left Y-axis.
- Plot `circuitPara2_1` using `format2` in the right Y-axis.

`plotyy( ___ ,xAxisPara,xAxisFmt,opCon,opVal)` plots the circuit parameters `circuitPara` on a X-Y plane with the variables `xAxisPara`, their corresponding format `xAxisFmt`, operating conditions `opCon`, and operating values `opVal` for the RFCKT or RF data object `h`.

`plotyy( ___ ,Name,Value)` plots the RFCKT or RF data object parameters under specified name-value pair operating conditions for the RF object.

`[ax,hlines1,hlines2] = plotyy( ___ )` returns the handles of the axes and two line series objects.

Examples

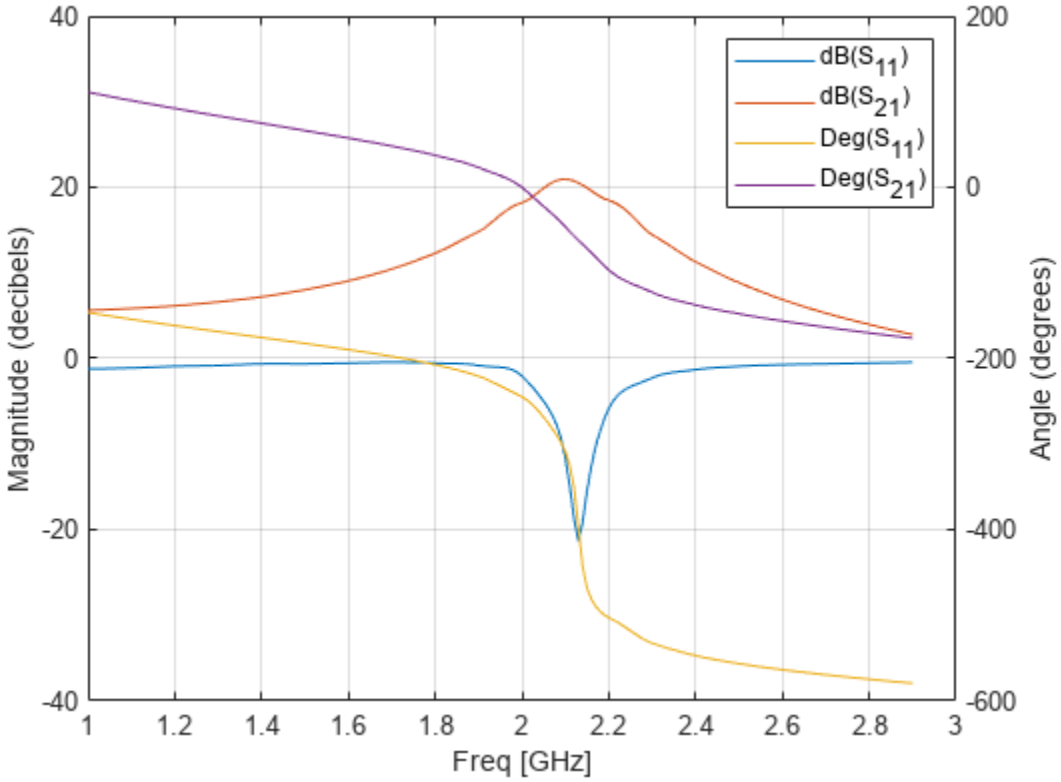
Determining Formats For Multiple Parameters

Create rfckt amplifier object.

```
amp = rfckt.amplifier;
```

Plot `S11` and `S21` parameters of the amplifier on two `y`-axis.

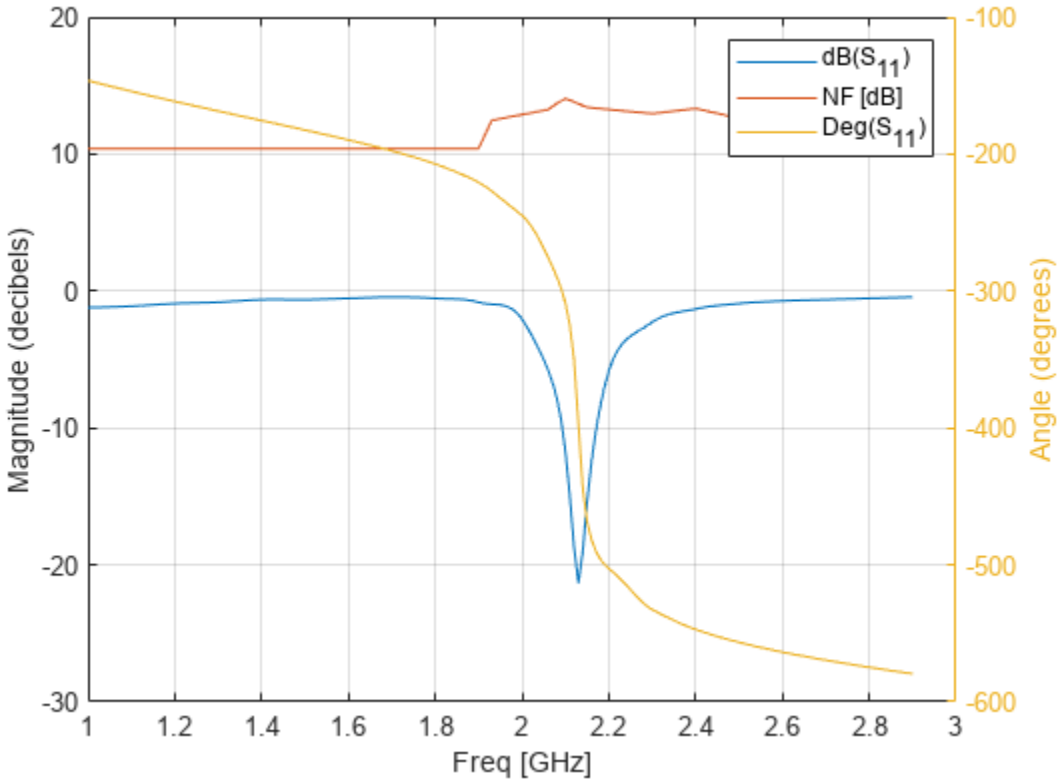
```
plotyy(amp,'S11','S21')
```



The primary and secondary formats for both S11 and S21 are `Magnitude(decibels)` and `Angle(degrees)`. respectively. So, the `plotyy` uses this primary-secondary format pair to create the plot.

Plot the `S11` and `NF` (noise figure) parameters of the amplifier on two `y`-axis.

```
plotyy(amp,'S11','NF')
```



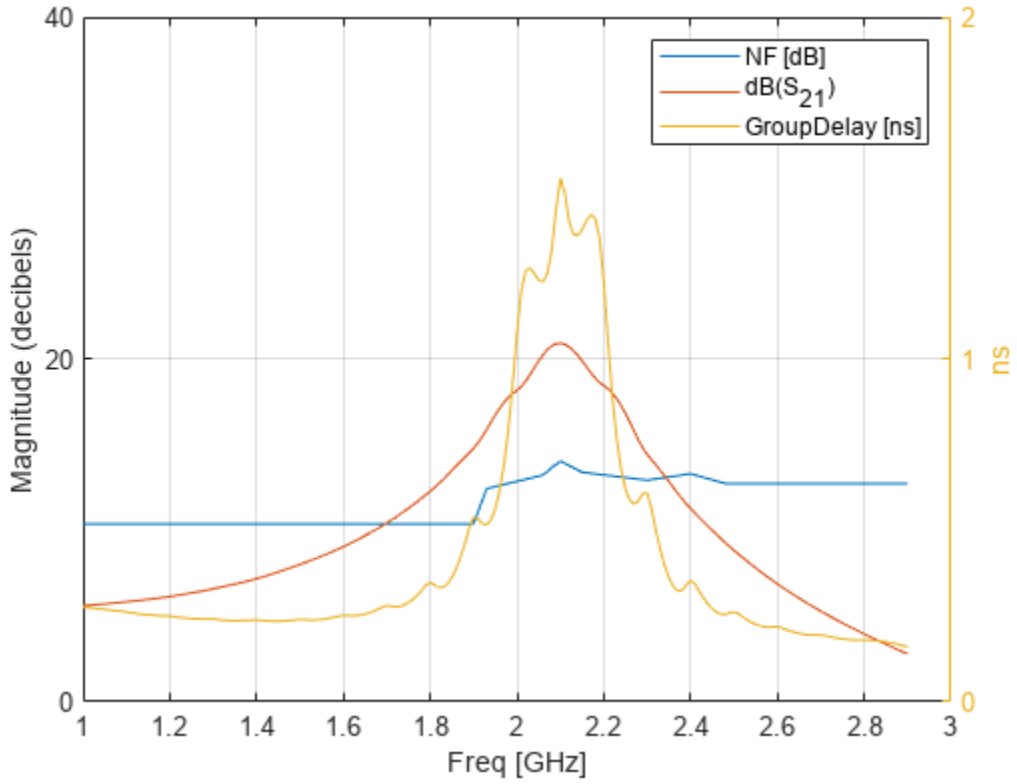
The primary and secondary formats for `S11` are `Magnitude (decibels)` and `Angle (degrees)`, respectively.

- `Magnitude (decibels)` is a valid format for both `S11` and `NF`
- `Angle (Degrees)` is a valid format for `S11`.

These formats both meet the preceding criteria, so the function uses this primary-secondary format pair to create the plot.

Plot the `NF`, `S21` and `GroupDelay` parameters of `amp` on two *y*-axis.

```
plotyy(amp, 'NF', 'S21', 'GroupDelay')
```



The primary and secondary formats for `S21` are `Magnitude (decibels)` and `Angle (Degrees)`, respectively. Both `NF` and `GroupDelay` have only a primary format.

- `Magnitude (decibels)` is the primary format for `NF`.
- `ns` is the primary format for `GroupDelay`.

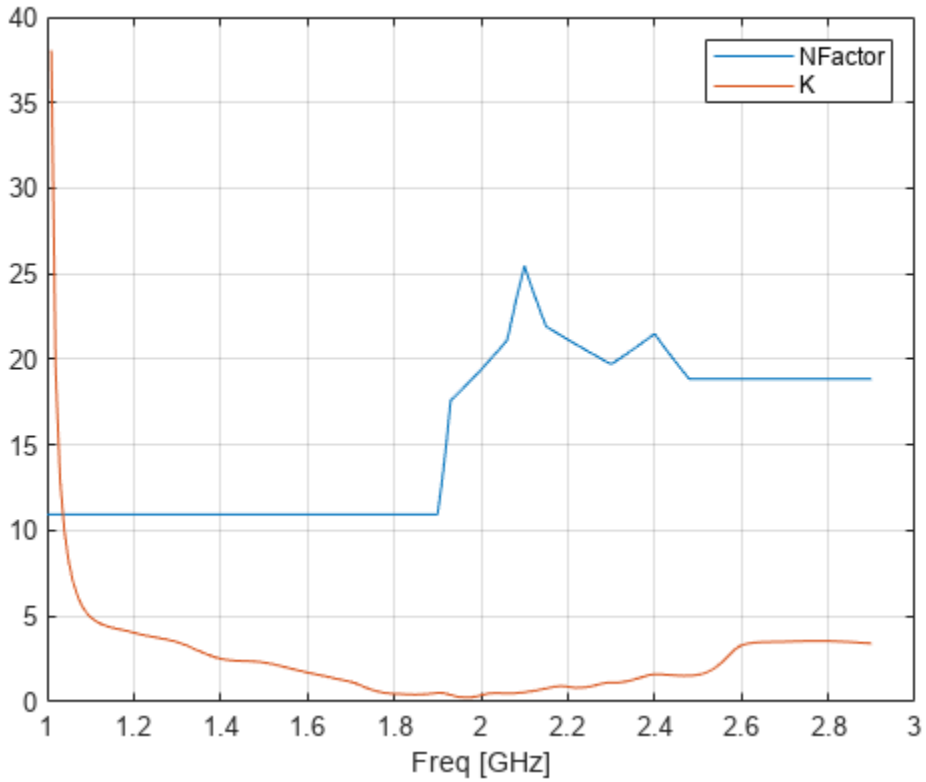
There is no primary-secondary format pair that meets the preceding criteria, so `plotyy` tries to find a pair of primary formats that meet the criteria. `plotyy` creates the plot using:

- `Magnitude (decibels)` for the left `y`-axis. This format is valid for both `NF` and `S21`.
- `ns` for the right `y`-axis. This format is valid for `GroupDelay` .

These formats meet the criteria.

Plot the `NFactor` and `K` parameters of `amp` on two *y*-axis.

```
plotyy(amp, 'NFactor', 'K')
```



Both `NFactor` and `K` have only a primary format, `None`, so `plotyy` calls the `plot` command to create a plot with a single *y*-axis whose format is `None`.

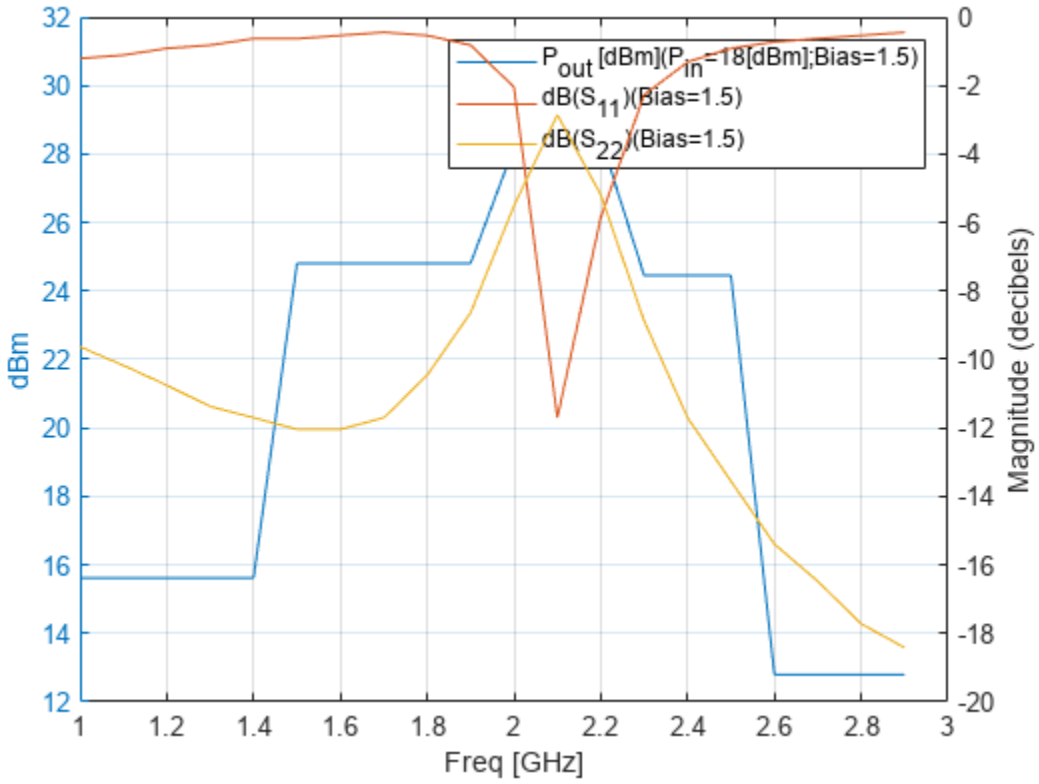
Plot Pout, S11, and S22 of RF Amplifier

Create an amplifier object from the specified P2D file.

```
ckt1 = read(rfckt.amplifier,'default.p2d');
```

Plot the Pout, S11, and S22 of the amplifier with frequency is plotted in `GHz` and bias is set to `1.5` v.

```
plotyy(ckt1,'Pout','dBm','S11','S22','Magnitude (decibels)', 'Freq','GHz','bias',1.5,'Pin',18)
```



Input Arguments

h — Circuit or data object

`rfckt` object | `rfdata` object

Circuit or data object, specified as `rfckt` object or `rfdata` object. For complete list of RFCKT and RF data objects, see “RF Circuit Objects” and “RF Data Objects”.

Example: `amp = rfckt.amplifier; plotyy(amp,'S11',S12')` Plots the parameters S11 and S12 of the `rfckt.amplifier` on the left and right Y-axis.

circuitPara — Valid parameters of RFCKT or RF data object

character vector | string

Valid parameters of RFCKT or RF data objects, specified as a character vector or string.

Type `listparam(h)` to get a list of valid parameters for a circuit object, `h`. Type `listformat(h,circuitPara)` to see the legitimate formats for a specified `parameter`. The first listed format is the default for the specified parameter.

Example: `amp = rfckt.amplifier; plotyy(amp,'S11',S12')` Plots the parameters, S11 and S12 of the `rfckt.amplifier` on the left and right Y-axis.

xAxisPara — Independent variables to plot with circuit parameters

`Pin` (default) | `Freq` | `AM`

Independent variables to plot with the circuit parameters, `circuitPara`, values specified in the table given. This table shows the `circuitPara` and their corresponding `xAxisPara` values. The function uses the default values listed in the table if you do not specify `xAxisPara`.

circuitPara Value	**xAxisPara Value**
`Pout`, `Phase`, `LS11`, `LS12`, `LS21`, `LS22`	`Pin` (default), `Freq`
`S11`, `S12`, `S21`, `S22`, `NF`, `IIP3`, `OIP3`, `GroupDelay`, `VSWRIn`, `VSWROut`, `GammaIn`, `GammaOut`, `FMIN`, `GammaOPT`, `RN`, `TF1`, `TF2`, `Gt`, `Ga`, `Gp`, `Gmag`, `Gmsg`, `GammaMS`, `GammaML`, `K`, `Delta`, `Mu`, `MuPrime`	`Freq`
`AM/AM`, `AM/PM`	`AM`

Derive `xAxisPara` for the RFCKT or RF data object `h` using the `listparam(h)` command.

`xAxisFmt` — `xAxisPara` format

`dBm` (default) | character vector | string scalar

`xAxisPara` format, specified as a character vector or string scalar. You do not need to specify `xAxisFmt` when `xAxisPara` is an operating condition.

This table shows the `xAxisPara` and their corresponding `xAxisFmt` values. The function uses the default values listed in the table if you do not specify `xAxisFmt` .

xAxisPara Value	**xAxisFmt Value**
`Pin`	`dBm` (default), `mW`, `W`, `dBW`
`Freq`	`THz`, `GHz`, `MHz`, `KHz`, `Hz` By default, `xAxisFmt` is chosen to provide the best scaling for the given `xAxisPara` values.
`AM`	`Magnitude (decibels)` (default), `Magnitude (linear)`

Derive `xAxisFmt` for the RFCKT or RF data object `h` using the `listformat(h,'xAxisPara')` command.

Example: `plotyy(h,'Pout','Pin','mW')` plots data on a X-Y plane for circuit object, `h`, with `xAxisPara` set to `'Pin'` and `xAxisFmt` set to `'mW'`.

`opCon` — Operating conditions

string scalar | character vector

Operating conditions derived from a P2D or S2D file, specified as a string scalar or a character vector.

For some circuit parameters, you can specify a set of frequency or input power values at which the function plots the specified parameter.

For example:

- When plotting large-signal S-parameters as a function of input power, you can specify frequency points of interest using `opCon` and `opVal` arguments.

- When plotting large-signal S-parameters as a function of frequency, you can specify input power levels of interest using `opCon` and `opVal` arguments.
- When plotting parameters as a function of an operating condition, you can specify both frequency and input power values using `opCon` and `opVal` arguments.

Enter the `getop(h)` command at the command line to get the operating conditions for the RF circuit object `h`.

`opVal` — Value of operating conditions
scalar

Value of the operating conditions in the `opCon` argument, specified as a scalar.

Example: `plotyy(h,'Pout','Pin','mW','bias',1.5)` plots the data on a X-Y plane for circuit object, `h`, with `opCon` set to `'bias'` and `value` set to `1.5`.

`format1` — Parameter format to plot on left Y-axis
character vector | string scalar

Parameter format to plot on left Y-axis, specified as a character vector or string.

`format2` — Parameter format to plot on right Y-axis
character vector | string scalar

Parameter format to plot on right Y-axis, specified as a character vector or string.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` pair arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `plotyy(h,'Pout','Pin','Freq',2.1e9)`

`Freq` — Frequency value
positive scalar

Frequency value at which to plot the specified parameters, specified as a positive scalar in Hz.

`Pin` — Input power level
positive scalar

Input power level at which to plot the specified parameters, specified as a positive scalar in dBm.

Output Arguments

`ax` — Axes handles
column vector of function handles

Axes handles, returned as a column vector of function handles.

`hlines1` — `lineseries` object for left y-axis
column vector of object handles

`lineseries` object for left y-axis, returned as a column vector of object handles.

`hlines2` — `lineseries` object for right y-axis
column vector of object handles

`lineseries` object for right y-axis, returned as a column vector of object handles.

Tips

- Use the Property Editor (`propertyeditor`) or the MATLAB `set` function to change Chart Line.

Version History

Introduced in R2007a

See Also

`analyze` | `calculate` | `listformat` | `listparam` | `loglog` | `plot` | `polar` | `semilogx` | `semilogy`

writeva

Generate Verilog-A description of `rational` object

Syntax

```
status = writeva(h,filename,innets,outnets,discipline,printformat,
filestoinclude)
```

Description

`status = writeva(h,filename,innets,outnets,discipline,printformat, filestoinclude)` writes a Verilog-A module that describes a 1-port rational object, `h` to a file specified using `filename` argument.

The function, `writeva` implements the object in Verilog-A using Laplace transform S-domain filters and returns a `status` of `True`, if the write operation is successful, and `False`, if the write operation is unsuccessful.

Examples

Write Verilog-A Description of Rational Object

Create a S-parameters object from the S-parameter data and frequency.

```
S = sparameters(0.5,2.4e9);
```

Perform rational fitting of the S-parameters.

```
h = rational(S);
```

Write Verilog-A description of an `rational` object.

```
status = writeva(h,'oneportS', ...
    'line_in', 'line_out', 'electrical', '%12.10e', 'disciplines.vams')
```

```
status = logical
   1
```

Input Arguments

h — Rational function object

1-port `rational` object

Rational function object, specified as a 1-port `rational` object.

filename — Verilog-A file name

`.va` (default) | character | string scalar

Verilog-A file name to which to write the module, specified as character vector or string scalar.

The `filename` can be specified with or without a path name and extension. The default extension, `.va`, is added automatically if `filename` does not end in this extension. The module name that is used in the file is the part of the `filename` argument that remains when the path name and extension are removed.

`innets` — Name of each of module's input nets
`'in'` (default) | character vector | string scalar

Name of each of the module's input nets, specified as a character vector or string scalar.

`outnets` — Name of each of module's output nets
`'out'` (default) | character vector | string scalar

Name of each of the module's output nets, specified as a character vector or string scalar.

`printformat` — Precision of Verilog-A module parameters
`'%15.10e'` (default) | character vector | string scalar

Precision of the following Verilog-A module parameters using the C language conversion specifications, specified as a character vector or string scalar. The C language conversion specifications includes:

- The numerator and denominator coefficients of the Verilog-A filter.
- The module's delay value and constant offset (or direct feedthrough), which are taken directly from the rational function object.

For more information on how to specify `printformat`, see the Format specification for `fprintf`.

`discipline` — Predefined Verilog-A discipline of nets
`'electrical'` (default) | character vector | string scalar

Predefined Verilog-A discipline of the nets, specified as a character vector or string scalar. The `discipline` argument defines attributes and characteristics associated with the nets.

`filestoinclude` — List of header files to include in module using Verilog-A `'include'` statements
`'include discipline.vams'` (default) | character vector | string scalar

List of header files to include in the module using Verilog-A `'include'` statements, specified as a character vector or string scalar.

Note The `write` function only accepts a single `rationalfit` object. It does not work with an array/ matrix of `rationalfit` objects,

---

Output Arguments

`status` — Status of write operation
`True` | `False` | logical vector

Status of write operation, returned as a logical vector. The argument, `status` returns `True`, if the Verilog-A write operation is successful, and `False` if the Verilog-A write operation is unsuccessful.

Version History

Introduced before R2006a

`writeva` function will support only one-port `rational` object

Behavior changed in R2022a

Starting in R2022a, the `writeva` function will support only one-port `rational` object.

When you open a model created before R2022a that contains the `writeva` function using a two-port `rational` object, the software throws an error asking you to update your code with a one-port `rational` object in the `writeva` function.

See Also

`rfmodel.rational` | `rationalfit` | `timeresp` | `stepresp` | `freqresp`

Topics

“Export Verilog-A module from Rational Function”

newref

Change reference impedance of S-parameters

Syntax

```
hs2 = newref(hs,Z0)
```

Description

`hs2 = newref(hs,Z0)` creates an S-parameter object, `hs2`, by converting the S-parameters in `hs` to the specified reference impedance, `Z0`.

Examples

Change Reference Impedance of S-parameters

Create an S-parameters object from data in the file, `default.s2p`.

```
hs = sparameters('default.s2p');
```

Change the reference impedance to 40 ohms.

```
hs2 = newref(hs,40)
```

```
hs2 =
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [191x1 double]
     Parameters: [2x2x191 double]
      Impedance: 40

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

Input Arguments

`hs` — S-parameters

network parameter object

S-parameters, specified as an RF Toolbox network parameter object. To create this type of object, use the `sparameters` function.

`Z0` — Reference impedance

real positive scalar

Characteristic impedance, in ohms, specified as a real positive scalar.

Output Arguments

hs2 — S-parameters
network parameter object

S-parameters with reference impedance `Z0`, returned as an RF Toolbox network parameter object.

Version History

Introduced in R2012b

See Also

`sparameters`

rfinterp1

Interpolate network parameter data at new frequencies

Syntax

```
objnew = rfinterp1(objold,newfreq)
objnew = rfinterp1( ___ ,'extrap')
```

Description

`objnew = rfinterp1(objold,newfreq)` interpolates the network parameter data in `objold` at the specified frequencies, `newfreq`, storing the results in `objnew`. `rfinterp1` uses the MATLAB function `interp1` to interpolate each individual `(i,j)` parameter of `objold` to the new frequencies.

Note If any value of the specified frequency is outside of the range specified by `objold.Frequencies`, then `rfinterp1` function inserts `NaNs` into `objnew` for those frequency values.

`objnew = rfinterp1( ___ ,'extrap')` also interpolates the network data, but if any of the frequency values you specify in `newfreq` are above the final frequency of the network parameter object, `objold.Frequencies(end)`, then the function extrapolates flat using the final value of the network parameter data `objold.Parameters(:,:,end)`. If any of the frequency values you specify in `newfreq` are below the first frequency of the network parameter object, `objold.Frequencies(1)`, then the function extrapolates linearly between `conj(objold.Parameters(:,:,1))` and `objold.Parameters(:,:,1)`. This ensures that the network parameter data of the new object, `objnew.Parameters` approach real values as in `newfreq` approach `0`.

Examples

Interpolate S-parameter data

Read the data from the file `default.s2p` into an S-parameter object.

```
hnet = sparameters('default.s2p');
```

Interpolate the data at a specified set of frequencies.

```
freq = [1.2:0.2:2.8]*1e9;
hnet2 = rfinterp1(hnet,freq)
```

```
hnet2 =
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [9x1 double]
     Parameters: [2x2x9 double]
      Impedance: 50
```

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

Input Arguments

`objold` — Data to interpolate

network parameter object

Data to interpolate, specified as an RF Toolbox network parameter object. `objold` must be one of the following types of network parameter objects: S-parameters, t-parameters, Y-parameters, Z-parameters, h-parameters, g-parameters, or ABCD-parameters.

`newfreq` — Interpolation frequencies

vector of positive numbers

Interpolation frequencies, specified as a vector of positive numbers ordered from smallest to largest.

Output Arguments

`objnew` — Interpolated data

network parameter object

Interpolated data, returned as an RF Toolbox network parameter object of the same type as `objnew`.

Algorithms

The function uses the MATLAB function `interp1` to perform the interpolation operation. Overall performance is similar to the RF Toolbox `analyze` function. However, behaviors of the two functions differ when `freq` contains frequencies outside the range of the original data:

- `analyze` performs a zeroth-order extrapolation for out-of-range data points.
- `rfinterp1` inserts `NaN` values for out-of-range data points.

Version History

Introduced in R2012b

See Also

`analyze` | `interp1`

rfparam

Extract vector of network parameters

Syntax

```
n_ij = rfparam(hnet,i,j)
abcd_vector = rfparam(habcd,abcdflag)
```

Description

`n_ij = rfparam(hnet,i,j)` extracts the network parameter vector (*i*,*j*) from the network parameter object, `hnet`.

`abcd_vector = rfparam(habcd,abcdflag)` extracts the *A*, *B*, *C*, or *D* vector from ABCD-parameter object, `habcd`.

Examples

Create Data Vector From S-Parameter Object

Read in the file default.s2p into an sparameters object and get the S21 value.

```
S = sparameters('default.s2p');
s21 = rfparam(S,2,1)
```

```
s21 = 191×1 complex

  -0.6857 + 1.7827i
  -0.6560 + 1.7980i
  -0.6262 + 1.8131i
  -0.5963 + 1.8278i
  -0.5664 + 1.8422i
  -0.5363 + 1.8563i
  -0.5062 + 1.8700i
  -0.4760 + 1.8835i
  -0.4457 + 1.8966i
  -0.4152 + 1.9094i
      ⋮
```

Input Arguments

abcdflag — ABCD-parameter index

`'A'|'B'|'C'|'D'`

Flag that determines which ABCD parameters the function extracts, specified as `'A'`, `'B'`, `'C'`, or `'D'`.

habcd — 2-port ABCD parameters
ABCD parameter object

2-port ABCD parameters, specified as an RF Toolbox ABCD parameter object. When you specify `abcdflag`, you must also specify an ABCD parameter object.

hnet — Network parameters
network parameter object

Network parameters, specified as an RF Toolbox network parameter object.

i — Row index
positive integer

Row index of data to extract, specified as a positive integer.

j — Column index
positive integer

Column index of data to extract, specified as a positive integer.

Output Arguments

n_ij — Network parameters (*i*, *j*)
vector

Network parameters (*i*, *j*), returned as a vector. The `i` and `j` input arguments determine which parameters the function returns.

Example: `S_21 = rfparam(hs,2,1)`

abcd_vector — *A*, *B*, *C*, or *D*- parameters
vector

A, *B*, *C*, or *D*- parameters, returned as a vector. The `abcdflag` input argument determines which parameters the function returns. The function supports only 2-port ABCD parameters; thus, the output is always a vector.

Example: `a_vector = rfparam(habcd,'A');`

Version History

Introduced before R2006a

See Also

`zparameters` | `sparameters` | `rfinterp1` | `rfplot`

rfplot

Plot S-parameter data

Syntax

```
rfplot(s_obj)
rfplot(s_obj,i,j)
rfplot(s_obj,[i1:in],[j1:jn])
rfplot(s_obj,{[i1 j1];...;[in jn]})
rfplot( ___ ,LineSpec)
rfplot( ___ ,plotflag)
rfplot(s_obj,'diag')
rfplot(s_obj,part)
rfplot(s_obj,part,k)
rfplot(ax, ___ )
hline = rfplot( ___ )
[hline,haxes] = rfplot(filter,frequencies)
```

Description

`rfplot(s_obj)` plots the magnitude in decibels versus frequency of all S-parameters (S_{11}, S_{12} ... S_{NN}) on the current axes.

`rfplot(s_obj,i,j)` plots the magnitude of S_{ij} in decibels on the current axis.

`rfplot(s_obj,[i1:in],[j1:jn])` plots the magnitude of multiple S-parameters in decibels on the current axis.

`rfplot(s_obj,{[i1 j1];...;[in jn]})` plots the magnitude of specific S-parameters in decibels on the current axis.

`rfplot( ___ ,LineSpec)` plots S-parameters using the line parameters specified in `LineSpec`.

`rfplot( ___ ,plotflag)` plots S-parameters according to the type specified in `plotflag`.

`rfplot(s_obj,'diag')` plots the magnitude of S_{ii} reflection coefficients or the diagonal elements of the S-parameter matrix `'diag'` on the current axis.

`rfplot(s_obj,part)` plots the upper or lower triangular portion of the S-parameters matrix on the current axis.

`rfplot(s_obj,part,k)` plots the elements on, above, or below the `k`th diagonal of the S-parameters matrix. For more information, see the `tril` and `triu` functions.

`rfplot(ax, ___ )` plots the S-parameters on the axes specified in `ax` instead of the current axes. Specify `ax` as the first input argument followed by any of the input argument combinations in the previous syntaxes. Return the current axes using the `gca` function.

`hline = rfplot( ___ )` plots the S-parameters and returns a column vector of line handles in `hline`.

`[hline,haxes] = rfplot(filter,frequencies)` plots the magnitude response of the S-parameters of the RF filter.

Examples

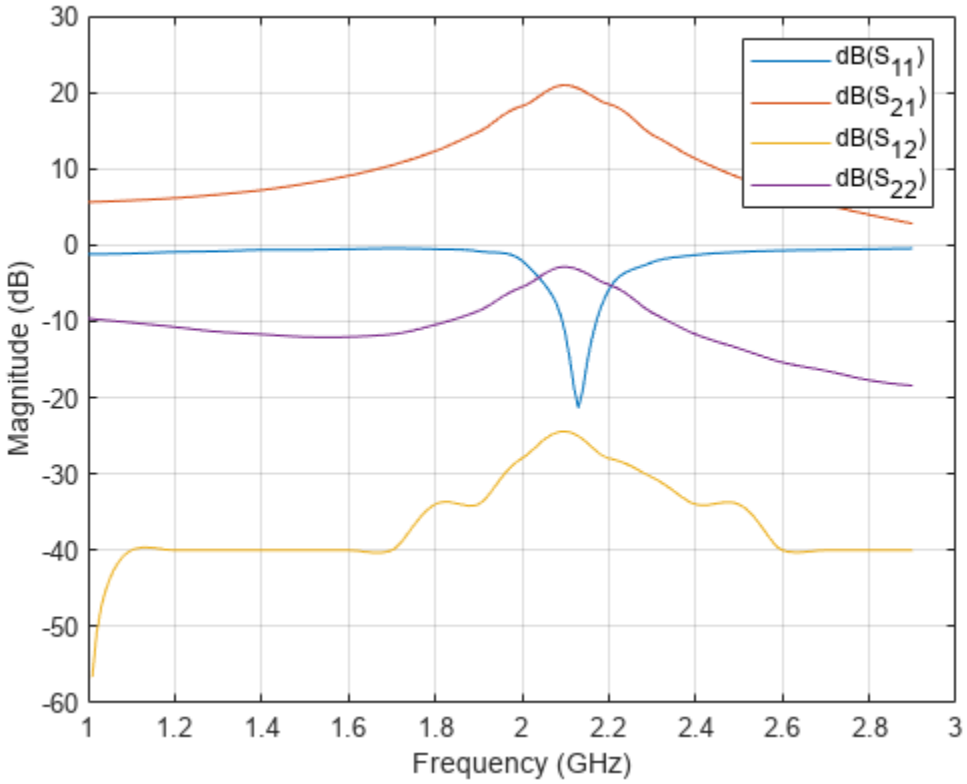
Plot S-Parameter Data

Use the `sparameters` function to create a set S-parameters.

```
hs = sparameters('default.s2p');
```

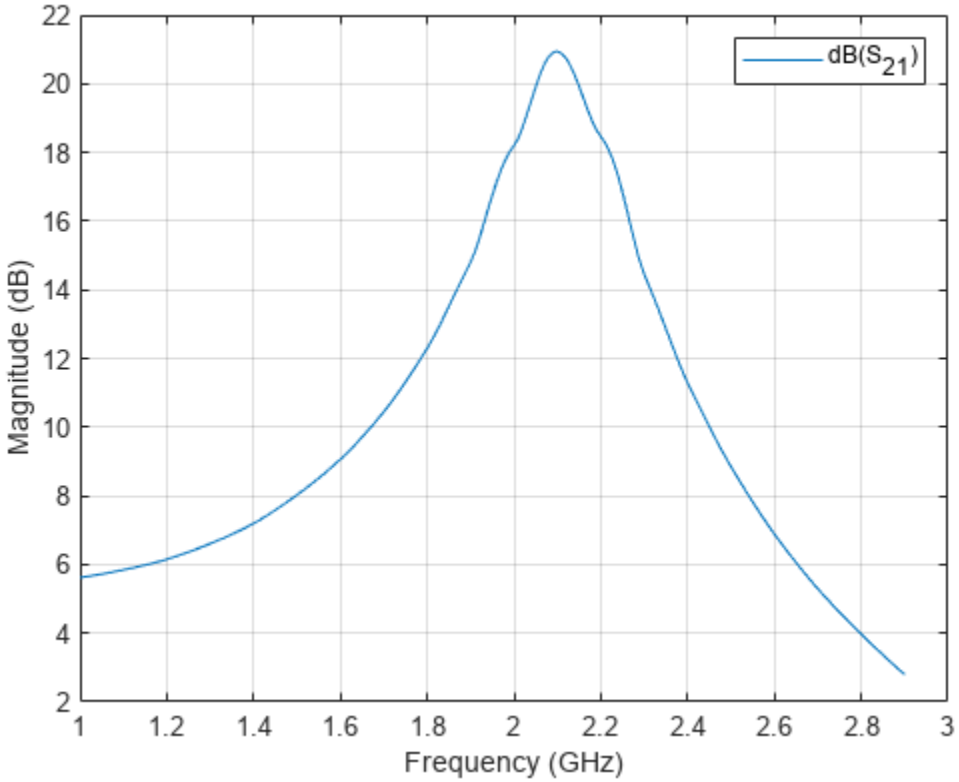
Plot all the S-parameters.

```
rfplot(hs)
```



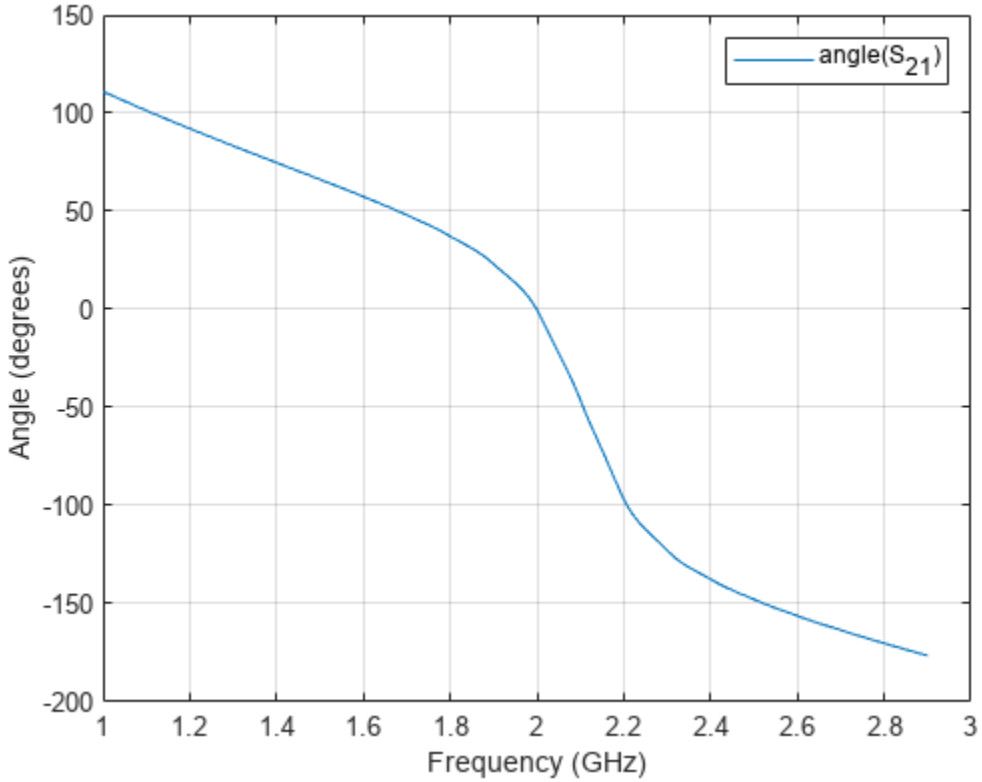
Plot S21.

```
rfplot(hs,2,1)
```



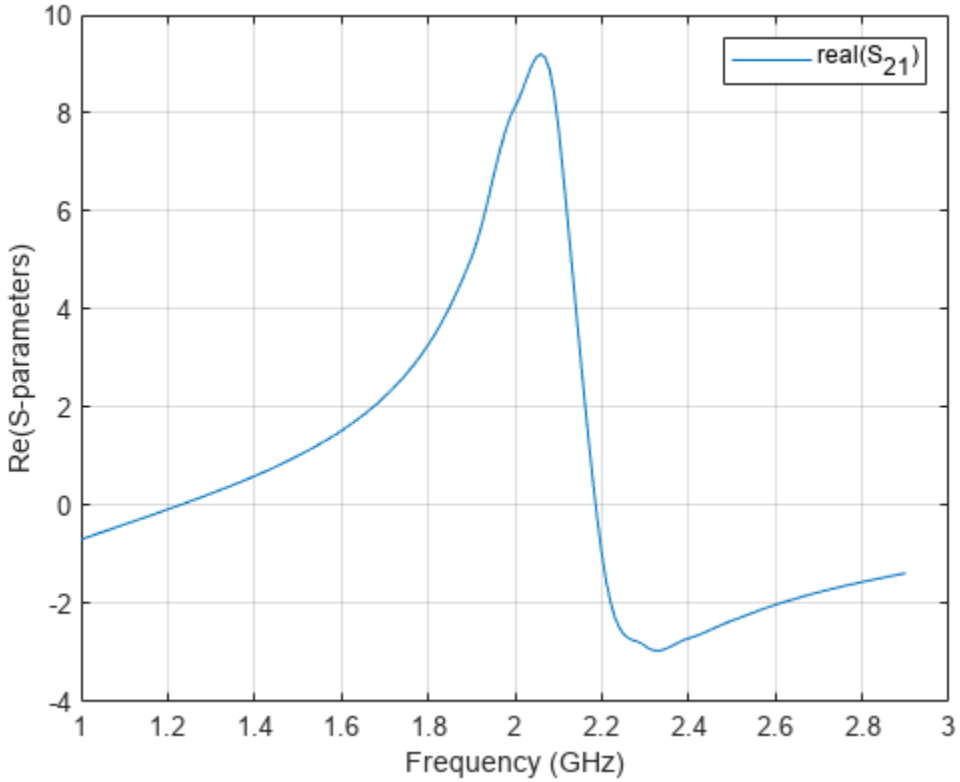
Plot the angle of S21 in degrees.

```
rfplot(hs,2,1,'angle')
```



Plot the real part of S21.

```
rfplot(hs,2,1,'real')
```



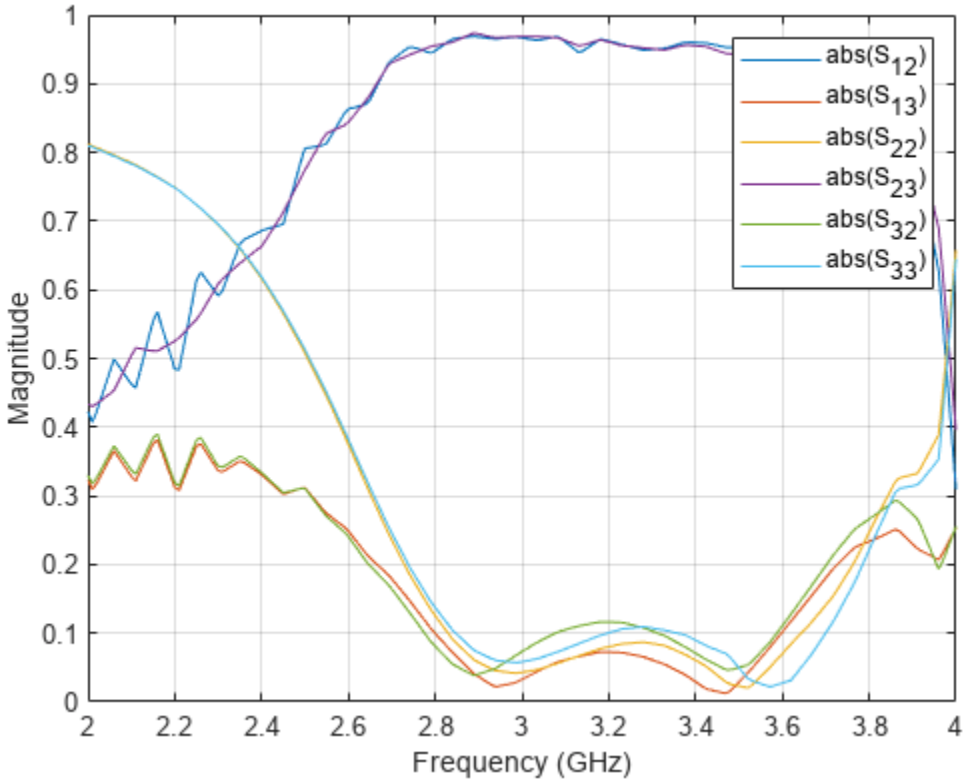
Plot S-Parameters in Specified Range

Create an S-parameter object from a three-port Touchstone file.

```
sobj = sparameters('default.s3p');
```

Plot S12, S13, S22, S23, S32, and S33.

```
rfplot(sobj,[1:3],[2:3],'abs')
```



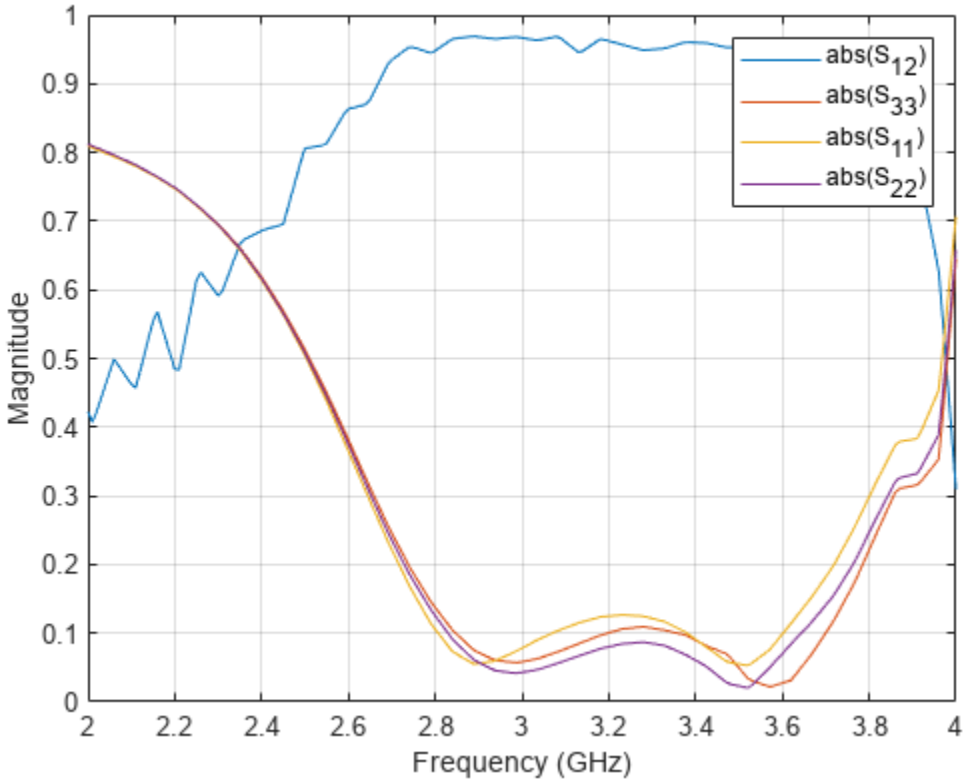
Plot Specific S-Parameters

Create an S-parameter object from a three-port Touchstone file.

```
sobj = sparameters('default.s3p');
```

Plot S12, S33, S11, and S22.

```
rfplot(sobj,{[1 2]; [3 3]; [1 1]; [2 2]},'abs')
```



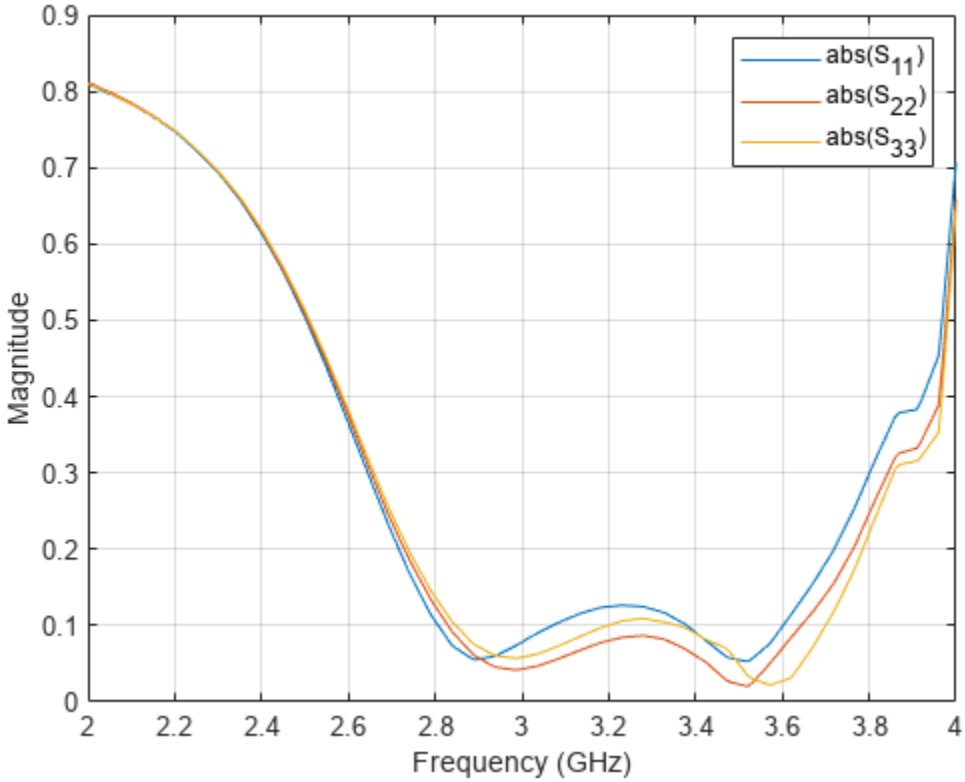
Plot Reflection and Transmission Coefficients

Create an S-parameter object from a three-port Touchstone file.

```
sobj = sparameters('default.s3p');
```

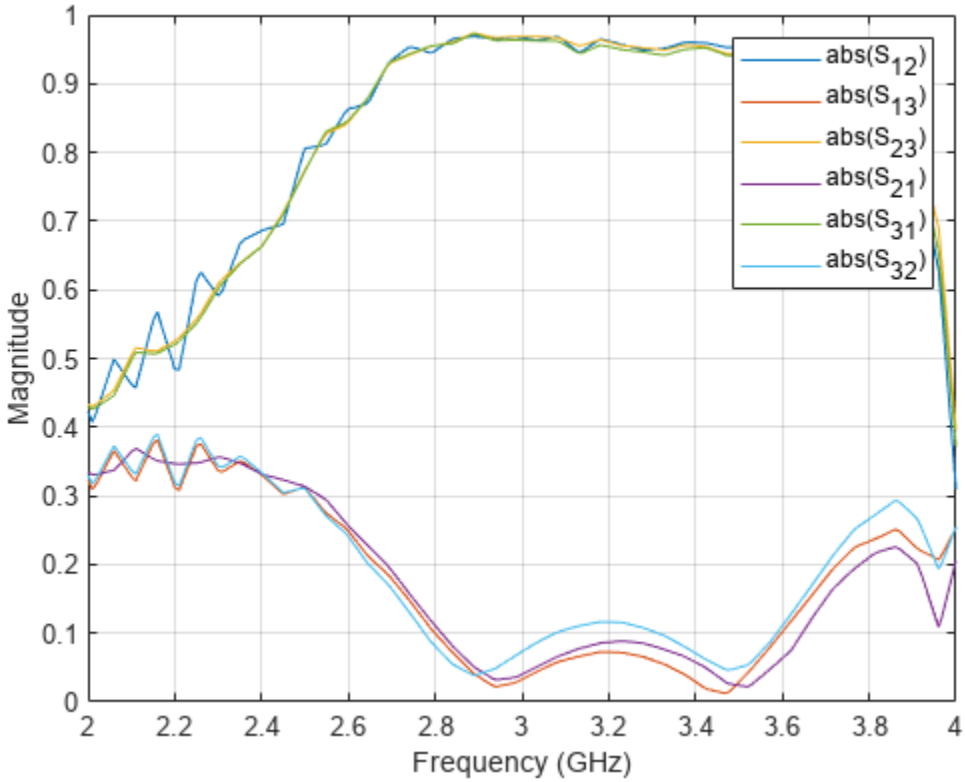
Plot the reflection coefficients of the S-parameters.

```
rfplot(sobj,'diag','abs')
```



Plot the transmission coefficients of the S-parameters.

```
rfplot(sobj,'triu',1,'abs')
hold on
rfplot(sobj,'tril',-1,'abs')
```



Input Arguments

`s_obj` — S-parameters

network parameter object

S-parameters, specified as RF Toolbox network parameter object. To create this type of object, use the `sparameters` function.

`i` — Row index

scalar | vector | cell array

Row index of the data to plot, specified as a scalar, vector, or cell array.

Type of Plot	How to Specify Indices
Single parameter	Specify `i` and `j` as scalars. `rfplot(s_obj,[1,2])`

Set of parameters	Specify `i` and `j` as vectors.
	`rfplot(s_obj,[1:3],[2:3])`
	`rfplot(s_obj,[1,2],[2,3])`

Type of Plot	How to Specify Indices
Specific parameters	Specify a cell array of `i` and `j` scalars. `rfplot(s_obj, {[1 2];[2 3]})`

`j` — Column index

scalar | vector | cell array

Column index of the data to plot, specified as a scalar, vector, or cell array.

Type of Plot	How to Specify Indices
Single parameter	Specify `i` and `j` as scalars. `rfplot(s_obj,[1,2])`

Set of parameters	Specify `i` and `j` as vectors.
	`rfplot(s_obj,[1:3],[2:3])`
	`rfplot(s_obj,[1,2],[2,3])`

Specific parameters	Specify a cell array of `i` and `j` scalars. `rfplot(s_obj, {[1 2];[2 3]})`

`LineSpec` — Line style, marker, and color

character vector | string

Line style, marker, and color, specified as a character vector or a string containing symbols. The symbols can appear in any order. You do not need to specify all three characteristics (line style, marker, and color). For example, if you omit the line style and specify the marker, then the plot shows only the marker and no line.

Example: `'--or'` creates a dashed line in red with circular markers

Line Style	Description	Resulting Line
`'-'`	Solid line	————
`'--'`	Dashed line	- — — — —
`':'`	Dotted line	.....
`'-.'`	Dash-dotted line	—·—·—·—·

Marker	Description	Resulting Marker
`'o'`	Circle	○
`'+'`	Plus sign	+
`'*'`	Asterisk	✳

Marker	Description	Resulting Marker
`'.'`	Point	
`'x'`	Cross	
`'_'`	Horizontal line	
`'|'`	Vertical line	
`'s'`	Square	
`'d'`	Diamond	
`'^'`	Upward-pointing triangle	
`'v'`	Downward-pointing triangle	
`'>'`	Right-pointing triangle	
`'<'`	Left-pointing triangle	
`'p'`	Pentagram	
`'h'`	Hexagram	

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

`plotflag` — Plot types
`'db'` (default) | `'real'` | `'imag'` | `'abs'` | `'angle'`

Plot types, specified as either `'db'`, `'real'`, `'imag'`, `'abs'`, or `'angle'`.

Example: `'angle'`

`filter` — RF filter
`rffilter` object | `lcladder` object

RF filter, specified as an `rffilter` object or an `lcladder` object.

`frequencies` — Frequencies to plot magnitude response
vector

Frequencies to plot magnitude response, specified as a vector.

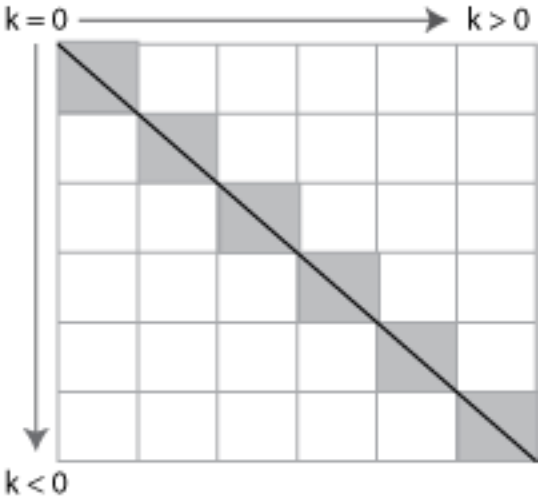
`part` — Portion of S-parameters matrix
`'triu'` | `'tril'`

Portion of the S-parameters matrix, specified as `'triu'` or `'tril'`. Specify `triu` to plot the "Upper Triangular" on page 3-53 portion of the matrix and `tril` to plot the "Lower Triangular" on page 3-53 portion.

`k` — Diagonals to include
`0` (default) | scalar

Diagonals to include, specified as a scalar.

- `k = 0` specifies the main diagonal.
- `k > 0` specifies a diagonal above the main diagonal.
- `k < 0` specifies a diagonal below the main diagonal.



`ax` — Axes object
`axes` object | `uiaxes` object

Axes object, specified as an `axes` or a `uiaxes` object.

Output Arguments

`hline` — Line
line handle

Line containing the S-parameter plot, returned as a line handle.

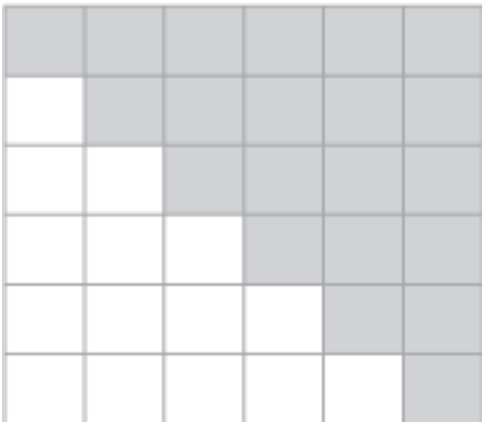
`haxes` — Axes
axes handle

Axes of the `rfplot`, returned as an axes handle.

More About

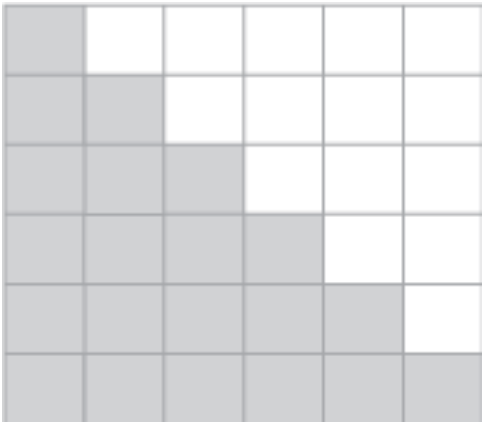
Upper Triangular

The upper triangular portion of a matrix includes the main diagonal and all elements above it. The shaded elements in this graphic depict the upper triangular portion of a 6-by-6 matrix.



Lower Triangular

The lower triangular portion of a matrix includes the main diagonal and all elements below it. The shaded elements in this graphic depict the lower triangular portion of a 6-by-6 matrix.



Version History

Introduced before R2006a

Plot S-parameter data with multiple indices

Use the `'diag'`, `part`, and `k` input arguments in the `rfplot` function to plot S-parameter data with multiple indices.

See Also

`smith` | `rfparam` | `rfinterp1` | `sparameters` | `sparameters` | `setrfplot`

sparameters

Calculate S-parameters for RF data, network, circuit, and matching network objects

Syntax

```
sobj = sparameters(filename)

sobj = sparameters(data,freq)
sobj = sparameters(data,freq,Z0)

sobj = sparameters(rfobj,freq)
sobj = sparameters(rfobj,freq,Z0)

sobj = sparameters(netparamobj)
sobj = sparameters(netparamobj,Z0)

sobj = sparameters(rfdataorckt)

sobj = sparameters(mnobj)
sobj = sparameters(mnobj,freq)
sobj = sparameters(mnobj,freq,Z0)
sobj = sparameters( ___ ,circuitindices)
```

Description

`sobj = sparameters(filename)` creates an S-parameter object `sobj` by importing data from the Touchstone file specified by `filename`.

`sobj = sparameters(data,freq)` creates an S-parameter object from the S-parameter data, `data`, and frequencies, `freq`.

`sobj = sparameters(data,freq,Z0)` creates an S-parameter object from the S-parameter data, `data`, and frequencies, `freq`, with a given reference impedance `Z0`.

`sobj = sparameters(rfobj,freq)` calculates the S-parameters of a RF object such as a filter object, circuit object, transmission line object, series RLC object, shunt RLC object, attenuator object or RF antenna object with the default reference impedance.

`sobj = sparameters(rfobj,freq,Z0)` calculates the S-parameters of a RF object such as a filter object, circuit object or transmission line object with a given reference impedance `Z0`.

`sobj = sparameters(netparamobj)` converts the network parameter object, `netparamobj`, to S-parameter object with the default reference impedance.

`sobj = sparameters(netparamobj,Z0)` converts the network parameter object, `netparamobj`, to S-parameter object with a given reference impedance, `Z0`.

`sobj = sparameters(rfdataorckt)` extracts network data from `rfdataobj` or `rfcktobj` and converts it into S-parameter object.

`sobj = sparameters(mnobj)` returns the S-parameters of the best created matching network, evaluated at a frequency list constructed from source and load impedance.

`sobj = sparameters(mnobj,freq)` returns the S-parameters of the best created matching network at each specified frequency.

`sobj = sparameters(mnobj,freq,Z0)` returns the S-parameters of the best created matching network at each specified frequency and characteristic impedance, `Z0`.

`sobj = sparameters( ___ ,circuitindices)` returns an array of S-parameter objects, one object for each circuit indicated in `circuitindices`. Use this option with any of the input argument combinations in the previous syntaxes.

Examples

Extract and Plot the S-Parameters of File

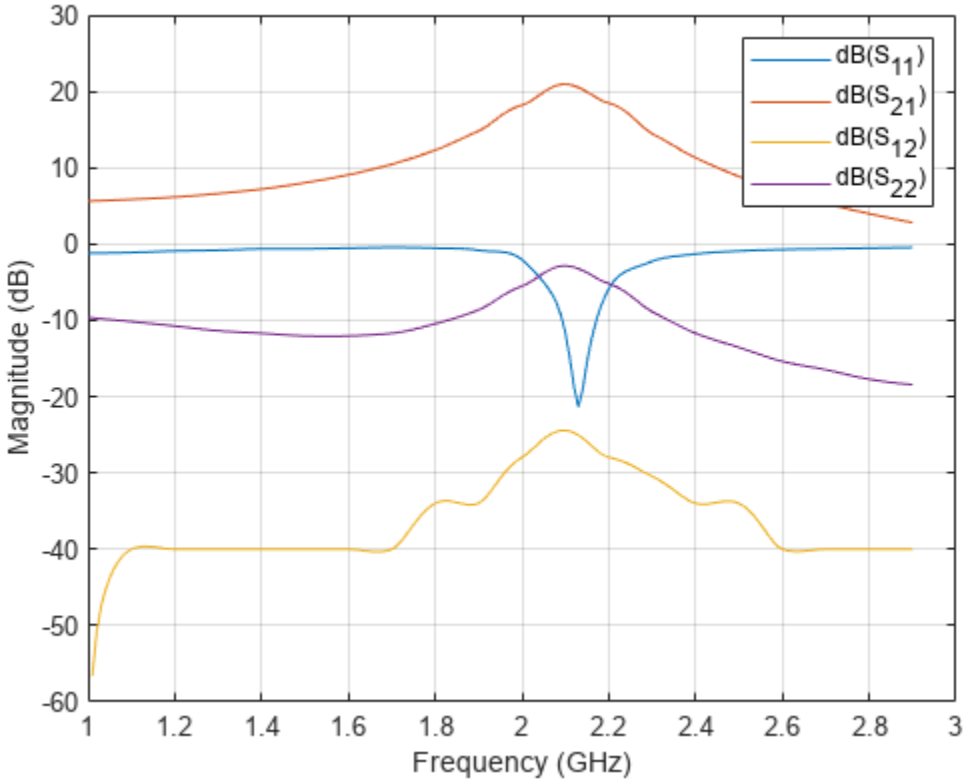
Extract S-parameters from file `default.s2p` and plot it.

```
S = sparameters('default.s2p');
disp(S)
```

```
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [191x1 double]
     Parameters: [2x2x191 double]
      Impedance: 50

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

Plot the extracted S-parameters data.

```
rfplot(S)
```



Calculate the S-Parameters of Circuit Object

Create a resistor element R50 and add it to a circuit object `example2` . Calculate the S-parameters of `example2` .

```
hR1 = resistor(50,'R50');
hckt1 = circuit('example2');
add(hckt1,[1 2],hR1)
setports (hckt1, [1 0],[2 0])
freq = linspace (1e3,2e3,100);
S = sparameters(hckt1,freq,100);
disp(S)
```

```
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [100x1 double]
     Parameters: [2x2x100 double]
      Impedance: 100

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

Convert Y-Parameters to S-Parameters

Extract Y-parameters from file default.s2p. Convert the resulting Y-parameters to S-parameters.

```
Y1 = yparameters('default.s2p');
S1 = sparameters(Y1,100);
disp(Y1)
```

```
  yparameters: Y-parameters object

       NumPorts: 2
    Frequencies: [191x1 double]
     Parameters: [2x2x191 double]

  rfparam(obj,i,j) returns Y-parameter Yij
```

```
disp(S1)
```

```
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [191x1 double]
     Parameters: [2x2x191 double]
      Impedance: 100

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

Convert RF Data Object to S-parameters

```
file = 'default.s2p';
h = read(rfdata.data, file);
S = sparameters(h)
```

```
S =
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [191x1 double]
     Parameters: [2x2x191 double]
      Impedance: 50.0000 + 0.0000i

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

Calculate the S-Parameters of a Matching Network Circuit

This example shows how to calculate the S-Parameters for a newly created matching network for the auto-generated circuit #2 with a reference impedance of 100 Ohm.

```
n       = matchingnetwork('LoadImpedance',100,'Components',3);
freq    = linspace(n.CenterFrequency-n.Bandwidth/2,n.CenterFrequency+n.Bandwidth/2);
RefZ0   = 100;
ckt_no  = 2;
s       = sparameters(n,freq,RefZ0,ckt_no)
```

```
s =
  sparameters: S-parameters object

       NumPorts: 2
    Frequencies: [100x1 double]
     Parameters: [2x2x100 double]
      Impedance: 100

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

S-parameters of RLCG Transmission Line

Create an RLCG transmission line using these specifications:

- Resistor : 100 ohms
- Capacitor : 1 pF

```
rlcglinetxline = txlineRLCGLine(R=100,C=1e-12)
```

```
rlcglinetxline =
  txlineRLCGLine: RLCGLine element

           Name: 'RLCGLine'
      Frequency: 1.0000e+09
              R: 100
              L: 0
              C: 1.0000e-12
              G: 0
        IntpType: 'Linear'
     LineLength: 0.0100
    Termination: 'NotApplicable'
       StubMode: 'NotAStub'
       NumPorts: 2
      Terminals: {'p1+'  'p2+'  'p1-'  'p2-'}
```

Calculate the S-parameters of the transmission line at 1 GHz.

```
sparam = sparameters(rlcglinetxline,1e9);
```

Input Arguments

`data` — S-parameter data

array of complex numbers

S-parameter data, specified as an array of complex numbers, of size *N*-by-*N*-by-*K*.

`rfobj` — RF object

circuit object | `rffilter` object | transmission line object | `seriesRLC` object | `shuntRLC` object | `attenuator` object | `rfantenna` object | `mixerIMT` object

RF object, specified as one of the following:

Circuit object	`circuit`
RF Filter object	`rffilter` and `lcladder`.
Transmission line objects	• `txlineCoaxial` • `txlineCPW` • `txlineMicrostrip` • `txlineParallelPlate` • `txlineRLCGLine` • `txlineTwoWire` • `txlineEquationBased` • `txlineDelayLossless` • `txlineDelayLossy` • `txlineElectricalLength` • `txlineStripline`
Series and Shunt RLC objects	`seriesRLC`, and `shuntRLC`
Attenuator object	`attenuator`
RF antenna object	`rfantenna`
Phase shift object	`phaseshift`
IMT mixer object	`mixerIMT`

`netparamobj` — Network parameter object
network parameter object

Network parameter object. The network parameter objects are of the type: `sparameters`, `yparameters`, `zparameters`, `gparameters`, `hparameters`, `abcdparameters`, and `tparameters`.

Example: `S1 = sparameters(Y1,100)` . `Y1` is a parameter object. This example converts Y-parameters to S-parameters at `100` ohms.

`filename` — Touchstone data file
character vector | string scalar

Touchstone data file, specified as a character vector, that contains network parameter data. `filename` can be the name of a file on the MATLAB path or the full path to a file.

Example: `sobj = sparameters('defaultbandpass.s2p');`

`freq` — S-parameter frequencies
vector of positive real numbers

S-parameter frequencies, specified as a vector of positive real numbers, sorted from smallest to largest.

`Z0` — Reference impedance
`50` (default) | positive real scalar

Reference impedance in ohms, specified as a positive real scalar. You cannot specify `Z0` if you are importing data from a file. The argument Z0 is optional and is stored in the `Impedance` property.

`rfdataorckt` — RF data or circuit object
RF data object | RF network object

RF data or circuit object. Specify `rfdataobj` as either `rfdata.data`, or `rfdata.network` object or specify `rfcktobj` as any analyzed `rfckt` type object, such as `rfckt.amplifier`, `rkckt.cascade` object.

`mnobj` — Matching network
`matchingnetwork` object

Matching network, specified as a `matchingnetwork` object.

Data Types: `char` | `string`

`circuitindices` — Index of matching network
scalar

Index of the matching network circuit, specified as a scalar.

Data Types: `double`

Output Arguments

`sobj` — S-parameter data
S-parameter object

S-parameter data, returned as an object. `disp(sobj)` returns the properties of the object:

- `NumPorts` — Number of ports, specified as an integer. The function calculates this value automatically when you create the object.
- `Frequencies` — S-parameter frequencies, specified as a *K*-by-1 vector of positive real numbers sorted from smallest to largest. The function sets this property from the `filename` or `freq` input arguments.
- `Parameters` — S-parameter data, specified as an *N*-by-*N*-by-*K* array of complex numbers. The function sets this property from the `filename` or `data` input arguments.
- `Impedance` — Reference impedance in ohms, specified as a positive real scalar. The function sets this property from the `filename` or `Z0` input arguments. If no reference impedance is provided, the function uses a default value of `50`.

Version History

Introduced in R2012a

Calculate S-parameters of stripline and IMT mixer elements

Specify a `txlineStripline` or `mixerIMT` object in `rfobj` to calculate a S-parameters of stripline or IMT mixer elements, respectively.

See Also

`yparameters` | `zparameters` | `gparameters` | `hparameters` | `abcdparameters` | `tparameters` | `rfparam` | `rfinterp1` | `rfplot` | `circuit`

Topics
“Bisect S-Parameters of Cascaded Probes”

abcdparameters

Create ABCD parameter object

Syntax

```
habcd = abcdparameters(filename)

habcd = abcdparameters(hnet)
habcd = abcdparameters(data,freq)

habcd = abcdparameters(rftbxobj)
```

Description

`habcd = abcdparameters(filename)` creates an ABCD parameter object `habcd` by importing data from the Touchstone file specified by `filename`. All data is stored in real/imag format.

`habcd = abcdparameters(hnet)` creates an ABCD parameter object from the RF Toolbox network parameter object `hnet`.

`habcd = abcdparameters(data,freq)` creates an ABCD parameter object from the ABCD parameter data, `data`, and frequencies, `freq`.

`habcd = abcdparameters(rftbxobj)` extracts network data from `rftbxobj` and converts it into ABCD-parameter data.

Examples

Read a File as ABCD-parameters and Extract A

Read the file `default.s2p` as abcd-parameters.

```
abcd = abcdparameters('default.s2p')
```

```
abcd =
  abcdparameters: ABCD-parameters object

       NumPorts: 2
    Frequencies: [191x1 double]
     Parameters: [2x2x191 double]

  rfparam(obj,specifier) returns specified ABCD-parameter 'A', 'B', 'C', or 'D'
```

Extract parameter A.

```
A = rfparam(abcd,'A')
```

```
A = 191×1 complex

  -0.1470 - 0.0698i
```

```
-0.1421 - 0.0698i
-0.1373 - 0.0696i
-0.1325 - 0.0694i
-0.1277 - 0.0691i
-0.1231 - 0.0688i
-0.1185 - 0.0683i
-0.1140 - 0.0678i
-0.1097 - 0.0672i
-0.1054 - 0.0666i
    ⋮
```

Input Arguments

`data` — ABCD parameter data

array of complex numbers

ABCD parameter data, specified as an array of complex numbers, of size 2*N*-by-2*N*-by-*K*. The function uses this input argument to set the value of the `Parameters` property of `habcd`.

`filename` — Touchstone data file

character vector

Touchstone data file, specified as a character vector, that contains network parameter data. `filename` can be the name of a file on the MATLAB path or the full path to a file.

Example: `habcd = abcdparameters('defaultbandpass.s2p');`

`freq` — ABCD parameter frequencies

vector of positive numbers

ABCD parameter frequencies, specified as a vector of positive real numbers, sorted from smallest to largest. The function uses this input argument to set the value of the `Frequencies` property of `habcd`.

`hnet` — Network parameter data

scalar handle

Network parameter data, specified as a scalar handle. If `hnet` is an ABCD parameter object, then `habcd` is a deep copy of `hnet`. Otherwise, the function performs a network parameter conversion to create `habcd`. When converting network parameters, the same restrictions apply as those for RF Toolbox network parameter data conversion functions:

- ABCD parameter objects support 2*N*-port data.
- Hybrid-g parameter objects support 2-port data.
- Hybrid parameter objects support 2-port data.
- S-parameter objects support *N*-port data.
- Y-parameter objects support *N*-port data.
- Z-parameter objects support *N*-port data.
- T-parameter objects support 2-port data.

`rftbxobj` — network object

scalar handle

Network object, specified as a scalar handle. Specify `rftbxobj` as one of the following types: `rfdata.data`, `rfdata.network`, and any analyzed `rfckt` type.

Output Arguments

habcd — ABCD parameter data

scalar handle

ABCD parameter data, returned as a scalar handle. `disp(habcd)` returns the properties of the object:

- `NumPorts` — Number of ports, specified as an integer. The function calculates this value automatically when you create the object.
- `Frequencies` — ABCD parameter frequencies, specified as a *K*- by- 1 vector of positive real numbers sorted from smallest to largest. The function sets this property from the `filename` or `freq` input arguments.
- `Parameters` — ABCD parameter data, specified as a 2*N*-by-2*N*-by-*K* array of complex numbers. The function sets this property from the `filename` or `data` input arguments.

Version History

Introduced in R2012b

See Also

`sparameters` | `yparameters` | `zparameters` | `gparameters` | `hparameters` | `tparameters`

gparameters

Create hybrid-g parameter object

Syntax

```
hg = gparameters(filename)

hg = gparameters(hnet)
hg = gparameters(data,freq)

hg = gparameters(rftbxobj)
```

Description

`hg = gparameters(filename)` creates a hybrid-g parameter object `hg` by importing data from the Touchstone file specified by `filename`. All data is stored in real/imag format.

`hg = gparameters(hnet)` creates a hybrid-g parameter object from the RF Toolbox network parameter object `hnet`.

`hg = gparameters(data,freq)` creates a hybrid-g parameter object from the g-parameter data, `data`, and frequencies, `freq`.

`hg = gparameters(rftbxobj)` extracts network data from `rftbxobj` and converts it into G-parameter data.

Examples

Extract G11

Read the file `default.s2p` as g-parameters and extract G11.

```
g = gparameters('default.s2p')
```

```
g =
  gparameters: g-parameters object

       NumPorts: 2
    Frequencies: [191x1 double]
     Parameters: [2x2x191 double]

  rfparam(obj,i,j) returns g-parameter gij
```

```
g11 = rfparam(g,1,1);
```

Input Arguments

`data` — Hybrid-g parameter data

array of complex numbers

Hybrid-g parameter data, specified as an array of complex numbers, of size 2-by-2-by-*K*. The function uses this input argument to set the value of the `Parameters` property of `hg`.

`filename` — Touchstone data file

character vector

Touchstone data file, specified as a character vector, that contains network parameter data. `filename` can be the name of a file on the MATLAB path or the full path to a file.

Example: `hg = gparameters('defaultbandpass.s2p');`

`freq` — Hybrid-g parameter frequencies

vector of positive scalars

Hybrid-g parameter frequencies, specified as a vector of positive real numbers, sorted from smallest to largest. The function uses this input argument to set the value of the `Frequencies` property of `hg`.

`hnet` — Network parameter data

scalar handle

Network parameter data, specified as a scalar handle. If `hnet` is a hybrid-g parameter object, then `hg` is a deep copy of `hnet`. Otherwise, the function performs a network parameter conversion to create `hg`. When converting network parameters, the same restrictions apply as those for RF Toolbox network parameter data conversion functions:

- ABCD parameter objects support 2*N*-port data.
- Hybrid-g parameter objects support 2-port data.
- Hybrid parameter objects support 2-port data.
- S-parameter objects support *N*-port data.
- Y-parameter objects support *N*-port data.
- Z-parameter objects support *N*-port data.
- T-parameter objects support 2-port data.

`rftbxobj` — network object

scalar handle

Network object, specified as a scalar handle. Specify `rftbxobj` as one of the following types: `rfdata.data`, `rfdata.network`, and any analyzed `rfckt` type.

Output Arguments

`hg` — Hybrid-g parameter data

scalar handle

Hybrid-g parameter data, returned as a scalar handle. `disp(hg)` returns the properties of the object:

- `Frequencies` — Hybrid-g parameter frequencies, specified as a *K*-by-1 vector of positive real numbers sorted from smallest to largest. The function sets this property from the `filename` or `freq` input arguments.
- `Parameters` — Hybrid-g parameter data, specified as an 2-by-2-by-*K* array of complex numbers. The function sets this property from the `filename` or `data` input arguments.

Version History

Introduced in R2012b

See Also

`sparameters` | `yparameters` | `zparameters` | `hparameters` | `abcdparameters` | `tparameters`

hparameters

Create hybrid parameter object

Syntax

```
hh = hparameters(filename)

hh = hparameters(hnet)
hh = hparameters(data,freq)

hh = hparameters(rftbxobj)
```

Description

`hh = hparameters(filename)` creates a hybrid parameter object `hh` by importing data from the Touchstone file specified by `filename`. All data is stored in real/imag format.

`hh = hparameters(hnet)` creates a hybrid parameter object from the RF Toolbox network parameter object `hnet`.

`hh = hparameters(data,freq)` creates a hybrid parameter object from the hybrid parameter data, `data`, and frequencies, `freq`.

`hh = hparameters(rftbxobj)` extracts network data from `rftbxobj` and converts it into H-parameter data.

Examples

Extract H11

Read the file `default.s2p` as h-parameters and extract H11.

```
 h = hparameters('default.s2p')
```

```
h =
  hparameters: h-parameters object

       NumPorts: 2
    Frequencies: [191x1 double]
     Parameters: [2x2x191 double]

  rfparam(obj,i,j) returns h-parameter hij
```

```
  h11 = rfparam(h,1,1);
```

Input Arguments

`data` — Hybrid parameter data

array of complex numbers

Hybrid parameter data, specified as array of complex numbers, of size 2-by-2-by-*K*. The function uses this input argument to set the value of the `Parameters` property of `hh`.

`filename` — Touchstone data file

character vector

Touchstone data file, specified as a character vector, that contains network parameter data. `filename` can be the name of a file on the MATLAB path or the full path to a file.

Example: `hh = hparameters('defaultbandpass.s2p');`

`freq` — Hybrid parameter frequencies

vector of positive numbers

Hybrid parameter frequencies, specified as a vector of positive real numbers, sorted from smallest to largest. The function uses this input argument to set the value of the `Frequencies` property of `hh`.

`hnet` — Network parameter data

scalar handle

Network parameter data, specified as a scalar handle. If `hnet` is a hybrid parameter object, then `hh` is a deep copy of `hnet`. Otherwise, the function performs a network parameter conversion to create `hh`. When converting network parameters, the same restrictions apply as those for RF Toolbox network parameter data conversion functions:

- ABCD parameter objects support 2*N*-port data.
- Hybrid-g parameter objects support 2-port data.
- Hybrid parameter objects support 2-port data.
- S-parameter objects support *N*-port data.
- Y-parameter objects support *N*-port data.
- Z-parameter objects support *N*-port data.
- T-parameter objects support 2-port data.

`rftbxobj` — network object

scalar handle

Network object, specified as scalar handle. Specify `rftbxobj` as one of the following types: `rfdata.data`, `rfdata.network`, and any analyzed `rfckt` type.

Output Arguments

`hh` — Hybrid parameter data

scalar handle

Hybrid parameter data, returned as a scalar handle. `disp(hh)` returns the properties of the object:

- `Frequencies` — Hybrid parameter frequencies, specified as a *K*-by-1 vector of positive real numbers sorted from smallest to largest. The function sets this property from the `filename` or `freq` input arguments.
- `Parameters` — Hybrid parameter data, specified as a 2-by-2-by-*K* array of complex numbers. The function sets this property from the `filename` or `data` input arguments.

Version History

Introduced in R2012b

See Also

`sparameters` | `yparameters` | `zparameters` | `gparameters` | `abcdparameters` | `tparameters`

yparameters

Create Y-parameter object

Syntax

```
hy = yparameters(filename)

hy = yparameters(hnet)
hy = yparameters(data,freq)

hy = yparameters(rftbxobj)
```

Description

`hy = yparameters(filename)` creates a Y-parameter object `hy` by importing data from the Touchstone file specified by `filename`. All data is stored in real/imag format.

`hy = yparameters(hnet)` creates a Y-parameter object from the RF Toolbox network parameter object `hnet`.

`hy = yparameters(data,freq)` creates a Y-parameter object from the Y-parameter data, `data`, and frequencies, `freq`.

`hy = yparameters(rftbxobj)` extracts network data from `rftbxobj` and converts it into y-parameter data.

Examples

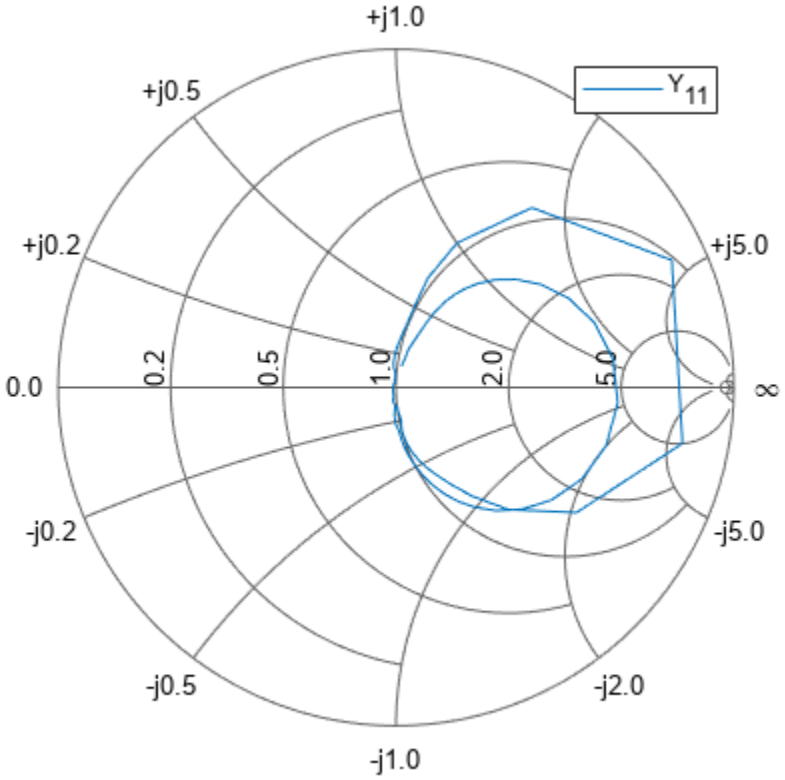
Plot Y-Parameters on Smith Chart

Extract Y-parameters from `default.s2p` and plot on a smith chart.

```
Y = yparameters('default.s2p')
```

```
Y =
  yparameters: Y-parameters object

       NumPorts: 2
    Frequencies: [191x1 double]
     Parameters: [2x2x191 double]

  rfparam(obj,i,j) returns Y-parameter Yij
```

```
figure;
smith(Y,1,1)
```



Input Arguments

`data` — Y-parameter data

array of complex numbers

Y-parameter data, specified as an array of complex numbers, of size *N*-by-*N*-by-*K*. The function uses this input argument to set the value of the `Parameters` property of `hy`.

`filename` — Touchstone data file

character vector

Touchstone data file, specified as a character vector, that contains network parameter data. `filename` can be the name of a file on the MATLAB path or the full path to a file.

Example: `hy = yparameters('defaultbandpass.s2p');`

`freq` — Y-parameter frequencies

vector of positive numbers

Y-parameter frequencies, specified as a vector of positive real numbers, sorted from smallest to largest. The function uses this input argument to set the value of the `Frequencies` property of `hy`.

`hnet` — Network parameter data

scalar handle

Network parameter data, specified as a scalar handle. If `hnet` is a Y-parameter object, then `hy` is a deep copy of `hnet`. Otherwise, the function performs a network parameter conversion to create `hy`. When converting network parameters, the same restrictions apply as those for RF Toolbox network parameter data conversion functions:

- ABCD parameter objects support 2*N*-port data.
- Hybrid-g parameter objects support 2-port data.
- Hybrid parameter objects support 2-port data.
- S-parameter objects support *N*-port data.
- Y-parameter objects support *N*-port data.
- Z-parameter objects support *N*-port data.
- T-parameter objects support 2-port data.

`rftbxobj` — network object
scalar

Network object, specified as scalar handle. Specify `rftbxobj` as one of the following types: `rfdata.data`, `rfdata.network`, and any analyzed `rfckt` type.

Output Arguments

`hy` — Y-parameter data
scalar handle

Y-parameter data, returned as a scalar handle. `disp(hy)` returns the properties of the object:

- `NumPorts` — Number of ports, specified as an integer. The function calculates this value automatically when you create the object.
- `Frequencies` — Y-parameter frequencies, specified as a *K*-by-1 vector of positive real numbers sorted from smallest to largest. The function sets this property from the `filename` or `freq` input arguments.
- `Parameters` — Y-parameter data, specified as an *N*-by-*N*-by-*K* array of complex numbers. The function sets this property from the `filename` or `data` input arguments.

Version History

Introduced in R2012b

See Also

`sparameters` | `zparameters` | `gparameters` | `hparameters` | `abcdparameters` | `tparameters`

zparameters

Create Z-parameter object

Syntax

```
hz = zparameters(filename)

hz = zparameters(hnet)
hz = zparameters(data,freq)

hz = zparameters(rftbxobj)
```

Description

`hz = zparameters(filename)` creates a Z-parameter object `hz` by importing data from the Touchstone file specified by `filename`. All data is stored in real/imag format.

`hz = zparameters(hnet)` creates a Z-parameter object from the RF Toolbox network parameter object `hnet`.

`hz = zparameters(data,freq)` creates a Z-parameter object from the Z-parameter data, `data`, and frequencies, `freq`.

`hz = zparameters(rftbxobj)` extracts network data from `rftbxobj` and converts it into z-parameter data.

Examples

Extract and Plot Imaginary Part of Z11

Read the file `default.s2p` as z-parameters and extract Z11.

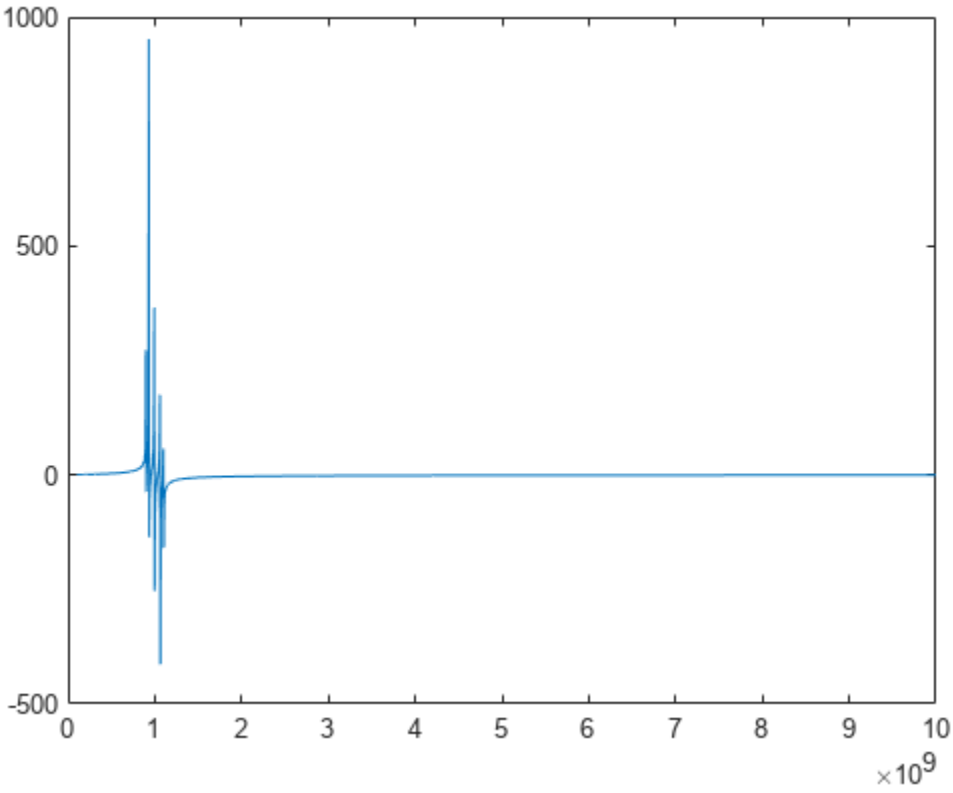
```
Z = zparameters('defaultbandpass.s2p')
```

```
Z =
  zparameters: Z-parameters object

       NumPorts: 2
    Frequencies: [1000x1 double]
     Parameters: [2x2x1000 double]

  rfparam(obj,i,j) returns Z-parameter Zij
```

```
z11 = rfparam(Z,1,1);
```

Plot imaginary part of Z11.

```
 plot(Z.Frequencies, imag(z11))
```



Input Arguments

`data` — Z-parameter data

array of complex numbers

Z-parameter data, specified as an array of complex numbers, of size *N*-by-*N*-by-*K*. The function uses this input argument to set the value of the `Parameters` property of `hz`.

`filename` — Touchstone data file that contains network parameter data

character vector

Touchstone data file, specified as a character vector. `filename` can be the name of a file on the MATLAB path or the full path to a file.

Example: `hz = zparameters('defaultbandpass.s2p');`

`freq` — Z-parameter frequencies

vector of positive numbers

Z-parameter frequencies, specified as a vector of positive real numbers, sorted from smallest to largest. The function uses this input argument to set the value of the `Frequencies` property of `hz`.

`hnet` — Network parameter data

scalar handle

Network parameter data, specified as a scalar handle. If `hnet` is a Z-parameter object, then `hz` is a deep copy of `hnet`. Otherwise, the function performs a network parameter conversion to create `hz`. When converting network parameters, the same restrictions apply as those for RF Toolbox network parameter data conversion functions:

- ABCD parameter objects support 2*N*-port data.
- Hybrid-g parameter objects support 2-port data.
- Hybrid parameter objects support 2-port data.
- S-parameter objects support *N*-port data.
- Y-parameter objects support *N*-port data.
- Z-parameter objects support *N*-port data.
- T-parameter objects support 2-port data.

`rftbxobj` — network object
scalar

Network object, specified as scalar handle. Specify `rftbxobj` as one of the following types: `rfdata.data`, `rfdata.network`, and any analyzed `rfckt` type.

Output Arguments

`hz` — Z-parameter object
scalar handle

Z-parameter data, returned as a scalar handle. `disp(hz)` returns the properties of the object:

- `NumPorts` — Number of ports, specified as an integer. The function calculates this value automatically when you create the object.
- `Frequencies` — Z-parameter frequencies, specified as a *K*-by-1 vector of positive real numbers sorted from smallest to largest. The function sets this property from the `filename` or `freq` input arguments.
- `Parameters` — Z-parameter data, specified as an *N*-by-*N*-by-*K* array of complex numbers. The function sets this property from the `filename` or `data` input arguments.

Version History

Introduced in R2012b

See Also

`sparameters` | `yparameters` | `gparameters` | `hparameters` | `abcdparameters` | `tparameters`

tparameters

Create T-parameter object

Syntax

```
tobj = tparameters(filename)
tobj = tparameters(tobj_old,z0)

tobj = tparameters(rftbx_obj)
tobj = tparameters(hnet, z0)
tobj = tparameters(paramdata,freq,z0)
```

Description

`tobj = tparameters(filename)` creates a T-parameter object, `ht` by importing data from the Touchstone file specified by `filename`. All data is stored in real/imag format.

`tobj = tparameters(tobj_old,z0)` converts a T-parameter data in `tobj_old` to the new impedance z_0. z_0 is optional, and if not provided, `tparam_data` is copied instead of converted.

`tobj = tparameters(rftbx_obj)` extracts S-parameter network data from `rfdata.network` object, an `rfdata.data` object, or any analyzed network object, and then converts the data to T-parameter data.

`tobj = tparameters(hnet, z0)` converts the network parameter data in `hnet` into T-parameter data.

`tobj = tparameters(paramdata,freq,z0)` creates T-parameter object directly from the specified data, `paramdata` using specified frequency and impedance.

Examples

Convert File to T-Parameters

Read S-parameter data from a Touchstone file and convert the data to T-parameters

```
T1 = tparameters('passive.s2p');
disp(T1)
```

```
  tparameters: T-parameters object

       NumPorts: 2
    Frequencies: [202x1 double]
     Parameters: [2x2x202 double]
      Impedance: 50

  rfparam(obj,i,j) returns T-parameter Tij
```

Change Impedance of T-Parameters

Change the impedance of T-parameters to 100 ohms.

```
T1 = tparameters('passive.s2p');
disp(T1)
```

```
  tparameters: T-parameters object

       NumPorts: 2
    Frequencies: [202x1 double]
     Parameters: [2x2x202 double]
      Impedance: 50

  rfparam(obj,i,j) returns T-parameter Tij
```

```
T2 = tparameters(T1,100);
disp(T2)
```

```
  tparameters: T-parameters object

       NumPorts: 2
    Frequencies: [202x1 double]
     Parameters: [2x2x202 double]
      Impedance: 100

  rfparam(obj,i,j) returns T-parameter Tij
```

Input Arguments

`tobj_old` — T-parameter object

scalar handle

T-parameter object, specified as a scalar handle.

`paramdata` — Input T-parameter data

2-by-2-by-*K* array of complex numbers

Input T-parameter data, specified as 2-by-2-by-*K* array of complex numbers. The function uses this input argument to set the value of the `Parameters` property of `ht`.

`filename` — Touchstone data file

character vector

Touchstone data file, specified as a character vector. `filename` can be the name of a file on the MATLAB path or the full path to a file.

Example: `ht = tparameters('defaultbandpass.s2p');`

`freq` — T-parameter frequencies

vector of positive real numbers

T-parameter frequencies, specified as a vector of positive real numbers. The frequencies are sorted from smallest to largest. The function uses this input argument to set the value of the `Frequencies` property of `ht`.

`z0` — T-parameter impedance
50 (default) | scalar

T-parameter impedance, specified as a scalar. z_0 is optional and is stored in the `Impedance`.

`hnet` — Network parameter data
scalar handle

Network parameter data, specified as a scalar handle. If `hnet` is a T-parameter object, then `tobj` is a deep copy of `hnet`. Otherwise, the function performs a network parameter conversion to create `tobj`. Specify `hnet` as one of the following types: `sparameters`, `yparameters`, `gparameters`, `hparameters`, `zparameters`, or `abcdparameters`.

`rftbx_obj` — network object
scalar handle

Network object, specified as a scalar handle. You can specify `rftbxobj` as one of the following types: `rfdata.data` object, `rfdata.network` object, or as any analyzed `rfckt` type.

Output Arguments

`tobj` — T-parameter object
scalar handle

T-parameter data, returned as a scalar handle. `disp(ht)` returns the properties of the object:

- `NumPorts` — Number of ports, specified as an integer. The function calculates this value automatically when you create the object.
- `Parameters` — T-parameter data, specified as a 2-by-2-by-*K* array of complex numbers. The 2x2 T-parameter data is specified for each frequency in the “Frequencies” property. The function sets this property from the `filename` or `paramdata` input arguments.
- `Impedance` — Characteristic impedance used to measure the T-Parameters, specified as a numeric positive real scalar.
- `Frequencies` — T-parameter frequencies, specified as a *K*-by-1 vector of positive real numbers sorted from smallest to largest. The function sets this property from the `filename` or `freq` input arguments.

Version History

Introduced in R2015a

See Also

`sparameters` | `yparameters` | `zparameters` | `gparameters` | `hparameters` | `abcdparameters`

add

Insert circuit element or circuit object into circuit

Syntax

```
add(cktobj,cktnodes,elem)
add(cktobj,cktnodes,elem,termorder)
elem_out = add(_____)
```

Description

`add(cktobj,cktnodes,elem)` inserts a circuit element `elem` into a circuit object `cktobj`. The terminals of the `elem` are attached to the nodes specified in `cktnodes`. If `elem` is a Touchstone file name or s-parameters object, an `nport` object is created from input and added to `cktobj`.

`add(cktobj,cktnodes,elem,termorder)` the terminals specified in `termorder` are attached to circuit nodes specified in `cktnodes`.

`elem_out = add(_____)` returns the inserted circuit element `elem_out` as an output. Specify any of the input argument combinations in the previous syntaxes.

Examples

Add Element to Circuit

Create a resistor, and add it to a circuit.

```
hR1 = resistor(50);
hckt1 = circuit('new_circuit1');
add(hckt1,[1 2],hR1)
disp(hR1)
```

```
  resistor: Resistor element

     Resistance: 50
           Name: 'R'
      Terminals: {'p'  'n'}
    ParentNodes: [1 2]
     ParentPath: 'new_circuit1'
```

```
disp(hckt1)
```

```
  circuit: Circuit element

    ElementNames: {'R'}
        Elements: [1x1 resistor]
           Nodes: [1 2]
            Name: 'new_circuit1'
```

Add Element to Specific Nodes of Circuit

Create a capacitor.

```
hC2 = capacitor(1e-10)
```

```
hC2 =
  capacitor: Capacitor element

    Capacitance: 1.0000e-10
           Name: 'C'
      Terminals: {'p'  'n'}
```

```
disp(hC2)
```

```
  capacitor: Capacitor element

    Capacitance: 1.0000e-10
           Name: 'C'
      Terminals: {'p'  'n'}
```

Connect terminal **n** of the capacitor to node 3 and terminal **p** of the capacitor to node 4.

```
hckt2 = circuit('new_circuit2');
add(hckt2,[3 4],hC2,{'n' 'p'})
disp(hckt2)
```

```
  circuit: Circuit element

    ElementNames: {'C'}
        Elements: [1x1 capacitor]
           Nodes: [3 4]
            Name: 'new_circuit2'
```

Create and Insert Element in Circuit

Create a circuit.

```
hckt3 = circuit('new_circuit3')
```

```
hckt3 =
  circuit: Circuit element

    ElementNames: {}
           Nodes: []
            Name: 'new_circuit3'
```

Insert an inductor into the circuit using the add function.

```
hL3 = add(hckt3,[100 200],inductor(1e-9));
```

Display the circuit element.

```
disp(hckt3)
```

```
  circuit: Circuit element

    ElementNames: {'L'}
        Elements: [1x1 inductor]
           Nodes: [100 200]
            Name: 'new_circuit3'
```

Display the inserted circuit element.

```
disp(hL3)
```

```
  inductor: Inductor element

     Inductance: 1.0000e-09
           Name: 'L'
      Terminals: {'p'  'n'}
    ParentNodes: [100 200]
     ParentPath: 'new_circuit3'
```

Add Two Circuits Together

Create circuit 1 and set the terminals using the **setterminals** functions.

```
hckt1 = circuit('circuit_new1');
add(hckt1,[1 2], resistor(100));
setterminals(hckt1, [1 2]);
disp(hckt1);
```

```
  circuit: Circuit element

    ElementNames: {'R'}
        Elements: [1x1 resistor]
           Nodes: [1 2]
            Name: 'circuit_new1'
       Terminals: {'t1'  't2'}
```

Create circuit 2 and set the terminals.

```
hckt2 = circuit('circuit_new2');
add(hckt2, [3 4], capacitor(1.5e-9));
setterminals(hckt2, [3 4]);
disp(hckt2);
```

```
  circuit: Circuit element

    ElementNames: {'C'}
        Elements: [1x1 capacitor]
           Nodes: [3 4]
            Name: 'circuit_new2'
       Terminals: {'t1'  't2'}
```

Add the two circuits.

```
add(hckt1, [2 4], hckt2);
disp(hckt2)
```

```
  circuit: Circuit element

    ElementNames: {'C'}
        Elements: [1x1 capacitor]
           Nodes: [3 4]
            Name: 'circuit_new2'
       Terminals: {'t1'  't2'}
     ParentNodes: [2 4]
      ParentPath: 'circuit_new1'

disp(hckt1)

  circuit: Circuit element

    ElementNames: {'R'  'circuit_new2'}
        Elements: [1x2 rf.internal.circuit.Element]
           Nodes: [1 2 4]
            Name: 'circuit_new1'
       Terminals: {'t1'  't2'}
```

Input Arguments

`cktobj` — Circuit object

scalar handle object

Circuit object into which the circuit element is inserted, specified as scalar handle object. This circuit object can be a new circuit or a nport object, or an already existing circuit.

`cktnodes` — Circuit nodes

vector of integers

Circuit nodes of the circuit object, specified as vector of integers. The function uses this input argument to attach the new element to the circuit.

`elem` — Circuit elements

scalar handle objects

Circuit elements that are inserted into the circuit object, specified as scalar handle objects. The element can be a resistor, capacitor, inductor, Touchstone file name, s-parameter object, or an entire circuit.

`termorder` — Element terminals

cell vector

Element terminals, which are the cell vectors found in `Terminals` property of `elem`. These input arguments are specified as scalar handle objects. .

Output Arguments

`elem_out` — Circuit elements

scalar handle objects

Circuit elements, which are returned as scalar handle objects, after using the add function. The function uses any or all of the input arguments to create these circuit elements.

Version History

Introduced in R2013b

See Also

`sparameters` | `setports` | `setterminals` | `clone`

setports

Set ports of circuit object

Syntax

```
setports(cktobj,nodepairs)
setports(cktobj,nodepairs,portnames)
```

Description

`setports(cktobj,nodepairs)` defines the node_pairs in an N-port `cktobj` using `nodepairs` argument. This syntax then assigns the ports default names. It also defines the terminals of a `cktobj`, taking the terminal names from the port names. If any of the node pairs do not exist, `setports` creates it.

`setports(cktobj,nodepairs,portnames)` defines the node_pairs in an N-port `cktobj` as ports using `nodepairs` argument. After defining the ports, this syntax names them using `portnames`. The length of the `portnames` must equal to the number of node_pairs in the circuit.

Examples

Create 1-Port Circuit with Default Names

Create a 1-port circuit using setports.

```
hckt1 = circuit('new_circuit1');
add(hckt1,[1 2],resistor(50))
setports(hckt1,[1 2])
disp(hckt1)
```

```
  circuit: Circuit element

    ElementNames: {'R'}
        Elements: [1x1 resistor]
           Nodes: [1 2]
            Name: 'new_circuit1'
        NumPorts: 1
       Terminals: {'p1+'  'p1-'}
```

Create 2-Port Circuit and Assign Port Names

Create a circuit and define two ports. Name the ports **in** and **out**.

```
hckt2 = circuit('example_circuit2');
add(hckt2,[2 3],resistor(50))
add(hckt2,[3 1],capacitor(1e-9))
setports(hckt2,[2 1],[3 1],{'in' 'out'})
disp(hckt2)
```

```
circuit: Circuit element

  ElementNames: {'R'  'C'}
      Elements: [1x2 rf.internal.circuit.RLC]
         Nodes: [1 2 3]
          Name: 'example_circuit2'
      NumPorts: 2
     Terminals: {'in+'  'out+'  'in-'  'out-'}
```

Input Arguments

`cktobj` — Circuit Object

scalar handle object

Circuit object for which the ports are defined, specified as scalar handle objects.

`nodepairs` — Node pairs

vector of integers

Node pairs of the circuit object, specified as vector of integers. The function uses this input argument to define the ports.

`portnames` — Port names

character vector

Names to name the ports defined for the circuit object, specified as character vector.

Version History

Introduced in R2013b

See Also

`sparameters` | `clone` | `setterminals`

setterminals

Set terminals of circuit object

Syntax

```
setterminals(cktobj,cktnodes)
setterminals(cktobj,cktnodes,termnames)
```

Description

`setterminals(cktobj,cktnodes)` defines the nodes in a `cktobj` as terminals using `cktnodes`. It then gives the terminals default names.

`setterminals(cktobj,cktnodes,termnames)` defines the nodes in a `cktobj` as terminals `cktnodes`. It then names the terminals using `termnames`. `cktnodes` and `termnames` must be same length.

Examples

Create a Circuit and Define Its Nodes as Terminals

Create a circuit names **new_circuit1**.

```
hckt1 = circuit('new_circuit1');
```

Add a resistor and capacitor to the circuit.

```
add(hckt1,[1 2],resistor(50));
add(hckt1,[2 3],capacitor(1e-9));
```

Set the terminals of the circuit.

```
setterminals(hckt1,[1 3])
disp(hckt1)
```

```
  circuit: Circuit element

    ElementNames: {'R'  'C'}
        Elements: [1x2 rf.internal.circuit.RLC]
           Nodes: [1 2 3]
            Name: 'new_circuit1'
       Terminals: {'t1'  't2'}
```

Create a Circuit and Define Its Nodes as Terminals Using Names

Create a circuit and add three resistors to it.

```
hckt2 = circuit('example_circuit2');
add(hckt2,[1 2],resistor(50));
```

```
add(hckt2,[1 3],resistor(50));
add(hckt2,[1 4],resistor(50));
```

Set terminals of the circuit by using (a, b, c) as **termnames**.

```
setterminals(hckt2,[2 3 4],{'a' 'b' 'c'})
disp(hckt2)
```

```
  circuit: Circuit element

    ElementNames: {'R'  'R_1'  'R_2'}
        Elements: [1x3 resistor]
           Nodes: [1 2 3 4]
            Name: 'example_circuit2'
       Terminals: {'a'  'b'  'c'}
```

Add Two Circuits Together

Create circuit 1 and set the terminals using the **setterminals** functions.

```
hckt1 = circuit('circuit_new1');
add(hckt1,[1 2], resistor(100));
setterminals(hckt1, [1 2]);
disp(hckt1);
```

```
  circuit: Circuit element

    ElementNames: {'R'}
        Elements: [1x1 resistor]
           Nodes: [1 2]
            Name: 'circuit_new1'
       Terminals: {'t1'  't2'}
```

Create circuit 2 and set the terminals.

```
hckt2 = circuit('circuit_new2');
add(hckt2, [3 4], capacitor(1.5e-9));
setterminals(hckt2, [3 4]);
disp(hckt2);
```

```
  circuit: Circuit element

    ElementNames: {'C'}
        Elements: [1x1 capacitor]
           Nodes: [3 4]
            Name: 'circuit_new2'
       Terminals: {'t1'  't2'}
```

Add the two circuits.

```
add(hckt1, [2 4], hckt2);
disp(hckt2)
```

```
  circuit: Circuit element

    ElementNames: {'C'}
```

```
        Elements: [1x1 capacitor]
           Nodes: [3 4]
            Name: 'circuit_new2'
       Terminals: {'t1'  't2'}
     ParentNodes: [2 4]
      ParentPath: 'circuit_new1'
```

```
disp(hckt1)
```

```
  circuit: Circuit element

    ElementNames: {'R'  'circuit_new2'}
        Elements: [1x2 rf.internal.circuit.Element]
           Nodes: [1 2 4]
            Name: 'circuit_new1'
       Terminals: {'t1'  't2'}
```

Input Arguments

`cktobj` — Circuit object

scalar handle object

Circuit object for which the terminals are defined, specified as a scalar handle object.

`cktnodes` — Circuit nodes

vector of integers

Circuit nodes, used by the function to define the terminals of the circuit, specified as a vector of integers.

`termnames` — Names

character vector

Names, used to identify the terminals defined for the circuit object, specified as a character vector.

Version History

Introduced in R2013b

See Also

`sparameters` | `clone` | `setports`

clone

Create copy of existing circuit element or circuit object

Syntax

```
outelem = clone(inelem)
outckt = clone(inckt)
```

Description

`outelem = clone(inelem)` creates a circuit element, `outelem`, with identical properties as `inelem`. The clone does not copy information about the parent circuit such as `ParentNodes` and `ParentPath`.

`outckt = clone(inckt)` creates a circuit object, `outckt`, identical to `inckt`. Circuit elements in the `inckt` are cloned recursively and added to the same nodes in the `outckt`. The ports or terminals in the `outckt` are defined same as `inckt`.

Examples

Create an Element and Clone It

Create a resistor element.

```
hR1 = resistor(50);
disp (hR1)
```

```
  resistor: Resistor element

    Resistance: 50
          Name: 'R'
     Terminals: {'p'  'n'}
```

Clone resistor **hR1**.

```
hR2 = clone(hR1);
disp (hR2)
```

```
  resistor: Resistor element

    Resistance: 50
          Name: 'R'
     Terminals: {'p'  'n'}
```

Create an Circuit and Clone it

Create a circuit object. Add a resistor and capacitor to it.

```
hckt1 = circuit('circuit1');
hC1= add(hckt1,[1 2],capacitor(3e-9));
hR1 = add(hckt1,[2 3],resistor(100));
disp(hckt1)
```

```
  circuit: Circuit element

    ElementNames: {'C'  'R'}
        Elements: [1x2 rf.internal.circuit.RLC]
           Nodes: [1 2 3]
            Name: 'circuit1'
```

Clone the circuit object.

```
hckt2 = clone(hckt1);
disp (hckt2)
```

```
  circuit: Circuit element

    ElementNames: {'C'  'R'}
        Elements: [1x2 rf.internal.circuit.RLC]
           Nodes: [1 2 3]
            Name: 'circuit1'
```

Input Arguments

`inelem` — Circuit element

scalar handle object

Circuit element to be cloned, specified as scalar handle object. The circuit element can be a resistor, capacitor, or inductor.

`inckt` — Circuit object

scalar handle object

Circuit object to be cloned, specified as scalar handle object.

Output Arguments

`outelem` — Circuit element

scalar handle object

Cloned circuit element, returned as scalar handle object. The circuit element can be a resistor, capacitor, or inductor.

`outckt` — Circuit object

scalar handle object

Cloned circuit object, returned as scalar handle object.

Version History

Introduced in R2013b

See Also

`sparameters` | `setterminals` | `setports`

rfwrite

Write RF network data to Touchstone® file

Syntax

```
rfwrite(data,freq,filename)
rfwrite(netobj,filename)
rfwrite(_____,Name,Value)
```

Description

`rfwrite(data,freq,filename)` creates a Touchstone data

file, `filename`. `rfwrite` touchstone files output 16 digits.

Note RF Toolbox does not support Touchstone 2.0 files.

`rfwrite(netobj,filename)` creates a Touchstone file from a network parameter object, `netobj`.

`rfwrite(_____,Name,Value)` creates a Touchstone file using the options specified in the name-value pair arguments for the file specified in `filename`. For example, `rfwrite(S150,'passive150.s2p','FrequencyUnit','MHz')` writes Touchstone file `passive150.s2p` in MHz using the S-parameters stored in the variable, `S150`.

Examples

Write Touchstone File Using Data and Frequency Values

Write a new Touchstone file from file `default.s2p` using data and frequency values. The output is stored in `defaultnew.s2p`.

```
S50 = sparameters('default.s2p');
data = S50.Parameters;
freq = S50.Frequencies;
rfwrite(data, freq, 'defaultnew.s2p')
```

Write Touchstone File Using Network Object Parameters

Convert an existing Touchstone file `passive.s2p` to S-parameters with a new resistance value.

```
S50 = sparameters('passive.s2p');
S100 = newref(S50,100);
```

Write a Touchstone file `passive100.s2p` using the new S-parameters.

```
rfwrite(S100, 'passive100.s2p');
```

Write Touchstone File Using Name-Value Pair Arguments

Convert an existing Touchstone file `passive.s2p` to S-parameters with a new resistance value.

```
S50 = sparameters('passive.s2p');
S150 = newref(S50,150);
```

Write a Touchstone file `passive150.s2p` in `MHz` using the new S-parameters.

```
rfwrite(S150, 'passive150.s2p','FrequencyUnit', 'MHz');
```

Write Touchstone File Using Y-Parameters

Convert an existing Touchstone file `passive.s2p` to Y-parameters.

```
Y50 = yparameters('passive.s2p');
```

Write a Touchstone file `passive.y2p` in `MHz` using the new Y-parameters.

```
rfwrite(Y50, 'passive.y2p','FrequencyUnit', 'MHz');
```

Input Arguments

`data` — Number of ports and frequencies
matrix

Number of ports and frequencies, specified as an `N-by-N-by-K` matrix, to create Touchstone file. `N` is the number of ports of data to be written. `K` is the number of frequencies.

Data Types: `double`
Complex Number Support: Yes

`freq` — Value of frequencies
numeric vector

Value of frequencies, specified as a numeric vector of length `K`, represents the value of frequencies in Hz.

Data Types: `double`

`filename` — Name of Touchstone file
character vector | string scalar

Name of a Touchstone file, specified as a character vector.

Example: default.s2p

Data Types: `char` | `string`

`netobj` — Network parameter object
scalar

Network parameter object, specified as a scalar, to create Touchstone file. The `netobj` can be any one of the following types S-parameters, Y-parameters, Z-parameters, h-parameters, g-parameters, or ABCD-parameters.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `rfwrite(S150,'passive150.s2p','FrequencyUnit','MHz')`

`FrequencyUnit` — Scaling unit for frequency values

`'GHz'` (default) | `'MHz'` | `'KHz'` | `'Hz'`

Scaling unit for frequency value, specified as a comma-separated pair consisting of `'Frequency Unit'` and any one of the values shown in value summary.

Example: `'FrequencyUnit','MHz'`

Data Types: `double`

`Parameter` — Network parameter type

`'S'` (default) | `'Y'` | `'Z'` | `'h'` | `'g'`

Network parameter type, specified as a comma-separated pair consisting of `'Parameter'` and any one of the values shown in value summary. This pair determines the parameter type the data has to be converted into in the Touchstone file.

Example: `'Parameter','Z'`

Data Types: `double`

`Format` — File storage format

`'MA'` (default) | `'DB'` | `'RI '`

File storage format, specified as a comma-separated pair consisting of `'Format'` and any one of the values shown in value summary. This pair determines the format to store the Touchstone file.

Example: `'Format', 'MA'`

`ReferenceResistance` — Resistance

`50` (default) | positive scalar (Ohm)

Reference resistance, specified as a comma-separated pair consisting of `'ReferenceResistance'` and a positive scalar.

Example: `'ReferenceResistance',100`

Data Types: `double`

`ForceOverwrite` — Flag to suppress the warning message while overwriting existing file

`false` (default) | `true`

Flag to suppress the warning message while overwriting an existing file, specified as a comma-separated pair consisting of `'ForceOverwrite'` and a logical value. Set `'ForceOverwrite'` to true, to overwrite the `filename` without a warning message.

Example: `'ForceOverwrite',true`

Data Types: `double`

Version History

Introduced in R2014a

See Also

`write` | `show` | `report` | `sparameters`

Topics

“Write S2P Touchstone Files”

groupdelay

Group delay of S-parameter object or RF filter object or RF Toolbox circuit object

Syntax

```
gd = groupdelay(sparamobj)
gd = groupdelay(sparamobj,i,j)

gd = groupdelay(rfobj,freq)
gd = groupdelay(rfobj,freq,Name,Value)
```

Description

`gd = groupdelay(sparamobj)` calculates the group delay of an S-parameter object at the frequencies specified in the S-parameter object file. `sparamobj` can be an S-parameters object or an `nport` object.

`gd = groupdelay(sparamobj,i,j)` calculates the group delay of a specific S_{ij}. If i, j are not specified, the group delay is calculated for S_{21} for two-port objects and S_{11} for non-two-port objects.

`gd = groupdelay(rfobj,freq)` calculates the group delay of an RF Toolbox network object, `rfobj`, at the specified frequencies.

`gd = groupdelay(rfobj,freq,Name,Value)` calculates the group delay using additional options specified by one or more name-value pair arguments.

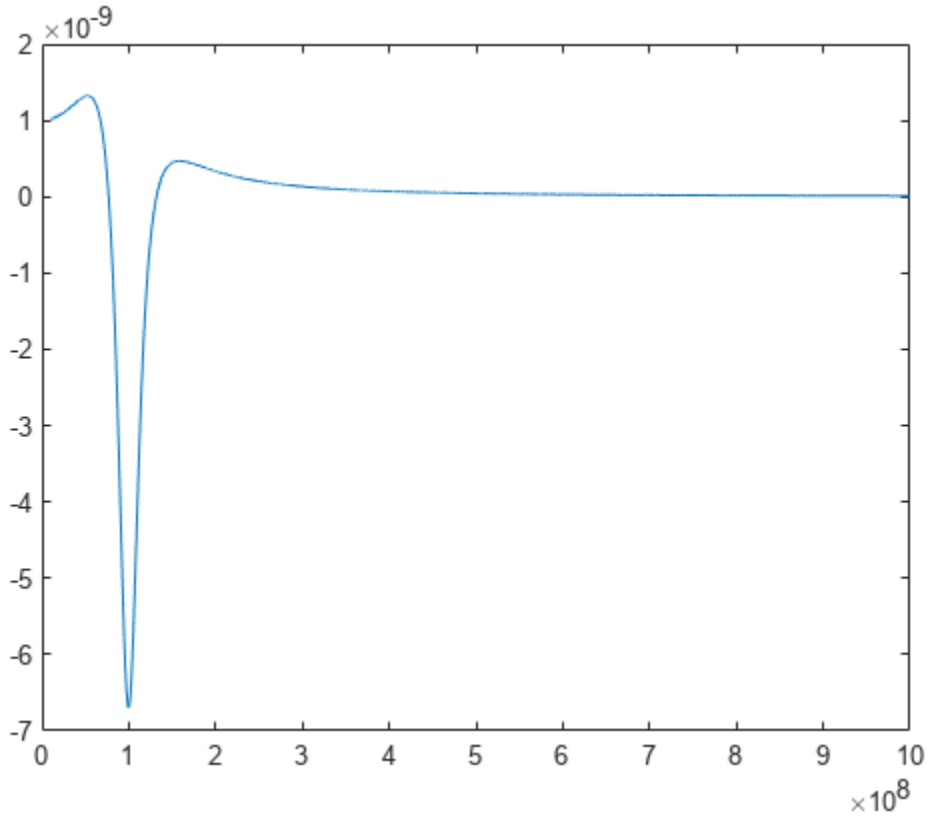
Example: `gd = groupdelay (filter, frequency, 'Aperture', 50)`

Examples

Group Delay of RLC Notch Filter

Calculate and plot the group delay of an RLC notch filter at the frequency range of `10–1000` MHz.

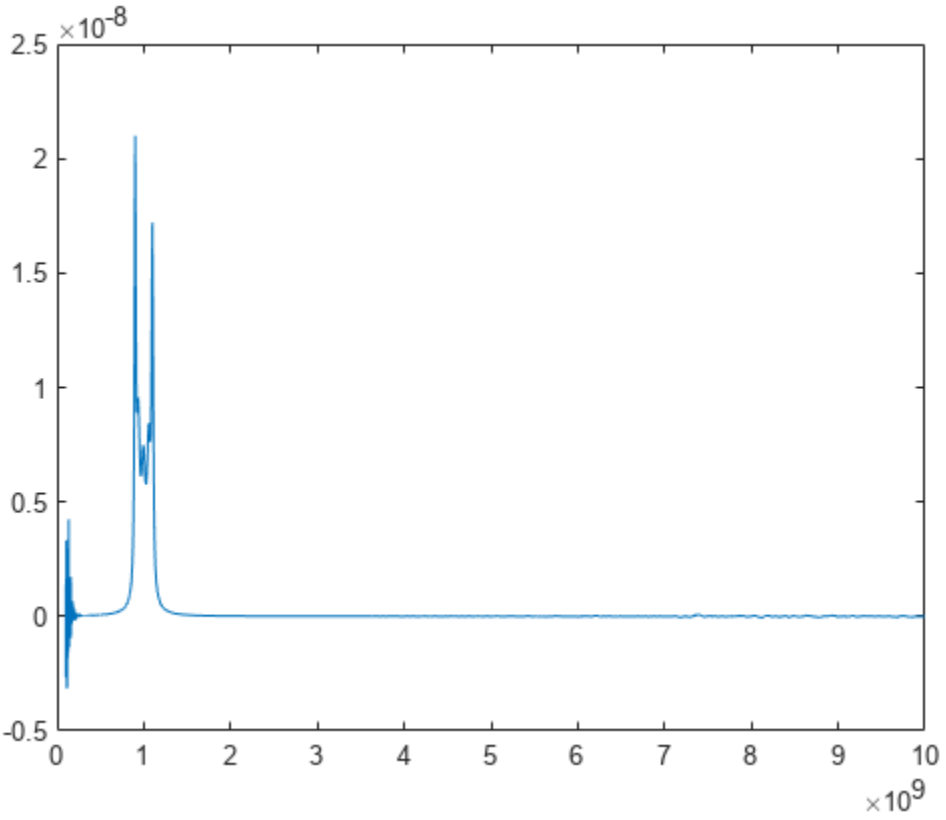
```
filt = circuit('notch');
add(filt,[1 2],resistor(200));
add(filt,[1 2],inductor(100e-9));
add(filt,[1 2],capacitor(25e-12));
setports(filt,[1 0],[2 0]);
freq = 10e6:10e4:1000e6;
gd = groupdelay(filt,freq);
figure
plot(freq,gd)
```



Group Delay of S-Parameter Data File

Find and plot the group delay from the specified Touchstone® file.

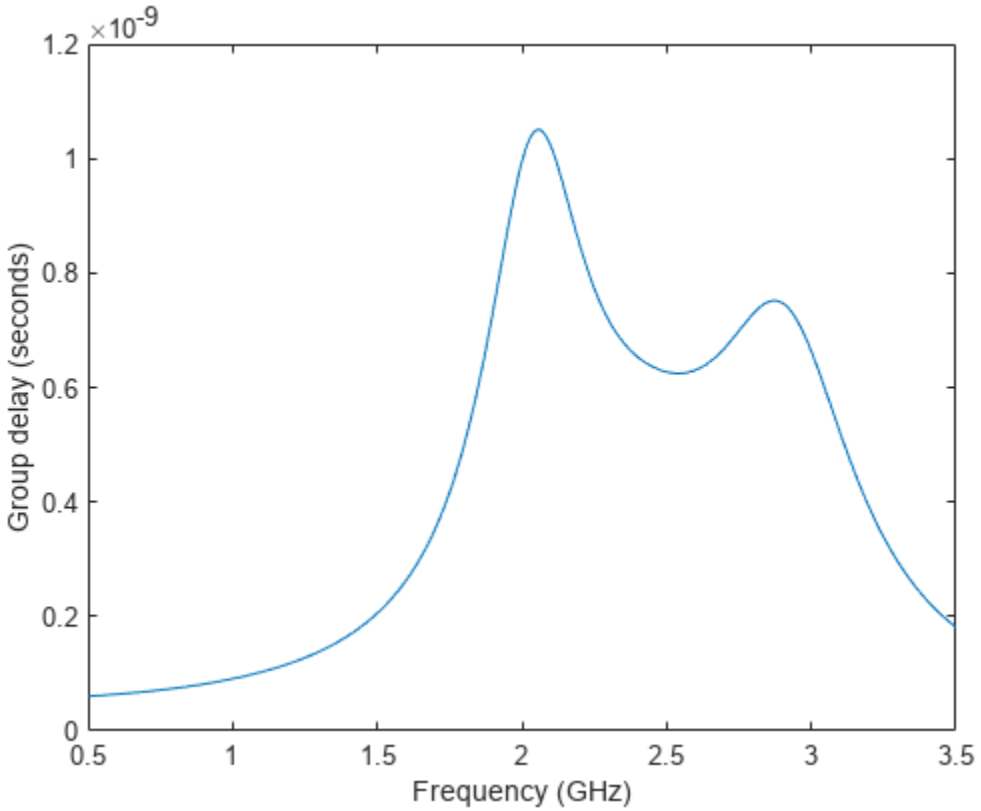
```
S = sparameters('defaultbandpass.s2p');
freq = S.Frequencies;
gd = groupdelay(S,freq);
figure
plot(freq,gd)
```



Group Delay of RF Filter Object

Calculate and plot the group delay of an RF Filter object at the frequency range of `0.5 - 3.5` GHz.

```
r = rffilter('FilterType','Butterworth','ResponseType',"Bandpass");
freq = linspace(0.5e9,3.5e9,1001);
gd  = groupdelay(r,freq);
plot(freq/1e9,gd)
xlabel('Frequency (GHz)');
ylabel('Group delay (seconds)');
```



Group Delay and Noise Figure of Two-Wire Transmission Line

Create a two-wire transmission line using these specifications:

- Radius - 0.5 mm
- Dielectric - air
- Thickness of dielectric or separation - 1.088 mm
- Permittivity or `EpsilonR` - 1.0054

```
twowiretxline = txlineTwoWire('Radius',0.5e-3,'EpsilonR',1.0054,'Separation',1.088e-3);
```

Calculate the noise figure and the group delay of the transmission line at 2.5 GHz.

```
nf = noisefigure(twowiretxline,2.5e9)
```

```
nf = 0
```

```
gd = groupdelay(twowiretxline,2.5e9)
```

```
gd = 3.3446e-11
```

Input Arguments

`sparamobj` — S-parameter object

object

S-parameter object. The function uses the data in the object to calculate the group delay.

Example: `sparamobj = sparameters('defaultbandpass.s2p')`

`rfobj` — RF object

circuit object | `rffilter` object | transmission line object | `seriesRLC` object | `shuntRLC` object | RF network objects

RF object, specified as one of the following:

Circuit object	`circuit`
RF Filter object	`rffilter` and `lcladder`.
Transmission line objects	• `txlineCoaxial` • `txlineCPW` • `txlineMicrostrip` • `txlineParallelPlate` • `txlineRLCGLine` • `txlineTwoWire` • `txlineEquationBased` • `txlineDelayLossless` • `txlineDelayLossy` • `txlineElectricalLength` • `txlineStripline`
Series and Shunt RLC objects	`seriesRLC`, and `shuntRLC`
RF Network objects	For complete list of RF network objects see, “RF Network Parameter Objects”.

`freq` — Frequencies at which group delay is calculated

vector of positive real numbers

Frequencies at which group delay is calculated, specified as a vector of positive real numbers.

`i,j` — Port numbers of S-parameter object or RF object

scalar integers

Port numbers of S-parameter object or RF object, specified as scalar integers.

Example: `S12`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `gd = groupdelay (filter, frequency, 'Aperture', 50)`

`Aperture` — Width of two frequency points
`freq*sqrt(eps)` (default) | real, positive, numeric scalar or vector

Width of two frequency points, specified as the comma-separated pair consisting of `'Aperture'` and a real, positive, numerics scalar or vector.

Example: `'Aperture',50`

Data Types: `double`

`Impedance` — Impedance of S-parameters
real, positive, scalar

Impedance of S-parameters, specified as the comma-separated pair consisting of `'Impedance'` and a real positive numeric scalar. The default impedance values for different objects are:

- `50` — LC ladder and circuit objects
- `obj.impedance` — S-parameter objects
- `obj.networkdata.impedance` — N-port objects

Example: `50`

Data Types: `double`

Output Arguments

`gd` — Group delay
numeric scalar

Group delay, returned as a numeric scalar in seconds.

Version History

Introduced in R2015b

See Also

`circuit` | `lcladder` | `nport` | `sparameters`

computeBudget

Compute results of RF budget object

Syntax

```
computeBudget(rfobj)
```

Description

`computeBudget(rfobj)` computes the result of an RF budget object. You can use this method only when the AutoUpdate property of the RF budget object is set to `false`.

Examples

Compute Results of RF Budget Object

Create a modulator with output-referred second-order intercept set to 20 and available power gain set to 3.

```
m = modulator(OIP2=20,Gain=3,ImageReject=false,ChannelSelect=false);
```

Create an amplifier with 10 dB gain.

```
a = amplifier(Gain=10);
```

Create an RF budget object specifying the input frequency of the signal, power applied at cascade, and signal bandwidth.

```
b1 = rfbudget([m a],2.1e9,-30,100e6,AutoUpdate=false)
```

```
b1 =
  rfbudget with properties:

              Elements: [1x2 rf.internal.rfbudget.RFElement]
        InputFrequency: 2.1 GHz
    AvailableInputPower: -30 dBm
        SignalBandwidth: 100 MHz
                Solver: Friis
            AutoUpdate: false
```

Use the `computeBudget` function to compute the results of an RF budget object.

```
computeBudget(b1)
```

Display the RF budget results.

```
b1
```

```
b1 =
  rfbudget with properties:

              Elements: [1x2 rf.internal.rfbudget.RFElement]
```

```
      InputFrequency: 2.1 GHz
 AvailableInputPower: -30 dBm
     SignalBandwidth: 100 MHz
              Solver: Friis
          AutoUpdate: false

Analysis Results
     OutputFrequency: (GHz) [  3.1    3.1]
         OutputPower: (dBm) [  -27    -17]
      TransducerGain: (dB)  [    3     13]
                  NF: (dB)  [    0      0]
                IIP2: (dBm) []
                OIP2: (dBm) []
                IIP3: (dBm) [  Inf    Inf]
                OIP3: (dBm) [  Inf    Inf]
                 SNR: (dB)  [63.98  63.98]
```

Input Arguments

`rfobj` — RF budget object

`rfbudget` object

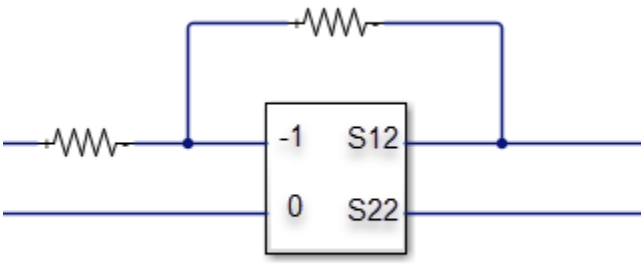
RF budget object, specified as a `rfbudget` object.

Algorithms

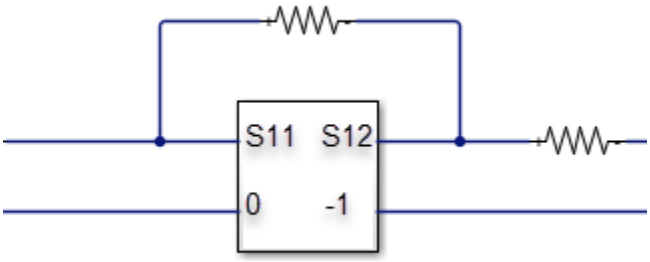
S_{21} = 0

If S_{21} of an element is zero, you make the following modifications to that element:

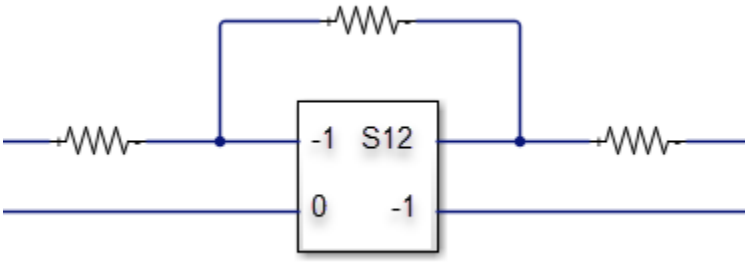
- S_{21} = 0 and S_{11} = -1



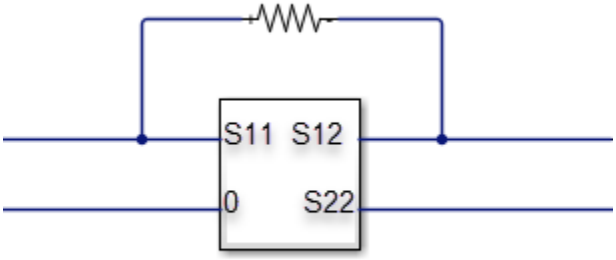
- S_{21} = 0 and S_{22} = -1



- S_{21} = 0, S_{22} = -1, and S = -1



- S_{21} = 0



Version History

Introduced in R2017a

See Also

`rfbudget` | `show` | `exportScript` | `exportRFBlockset` | `exportTestbench` | `rfplot` | `smithplot` | `polar`

exportScript

Export MATLAB code that generates RF budget object

Syntax

```
exportScript(rfobj)
```

Description

`exportScript(rfobj)` exports the MATLAB command-line code that generates an RF budget object. The script opens in an `Untitled*` window in the MATLAB editor.

Input Arguments

`rfobj` — RF budget object

`rfbudget` object

RF budget object, specified as a `rfbudget` object.

Examples

Export RF Budget Analysis to MATLAB Script

Create an RF budget object.

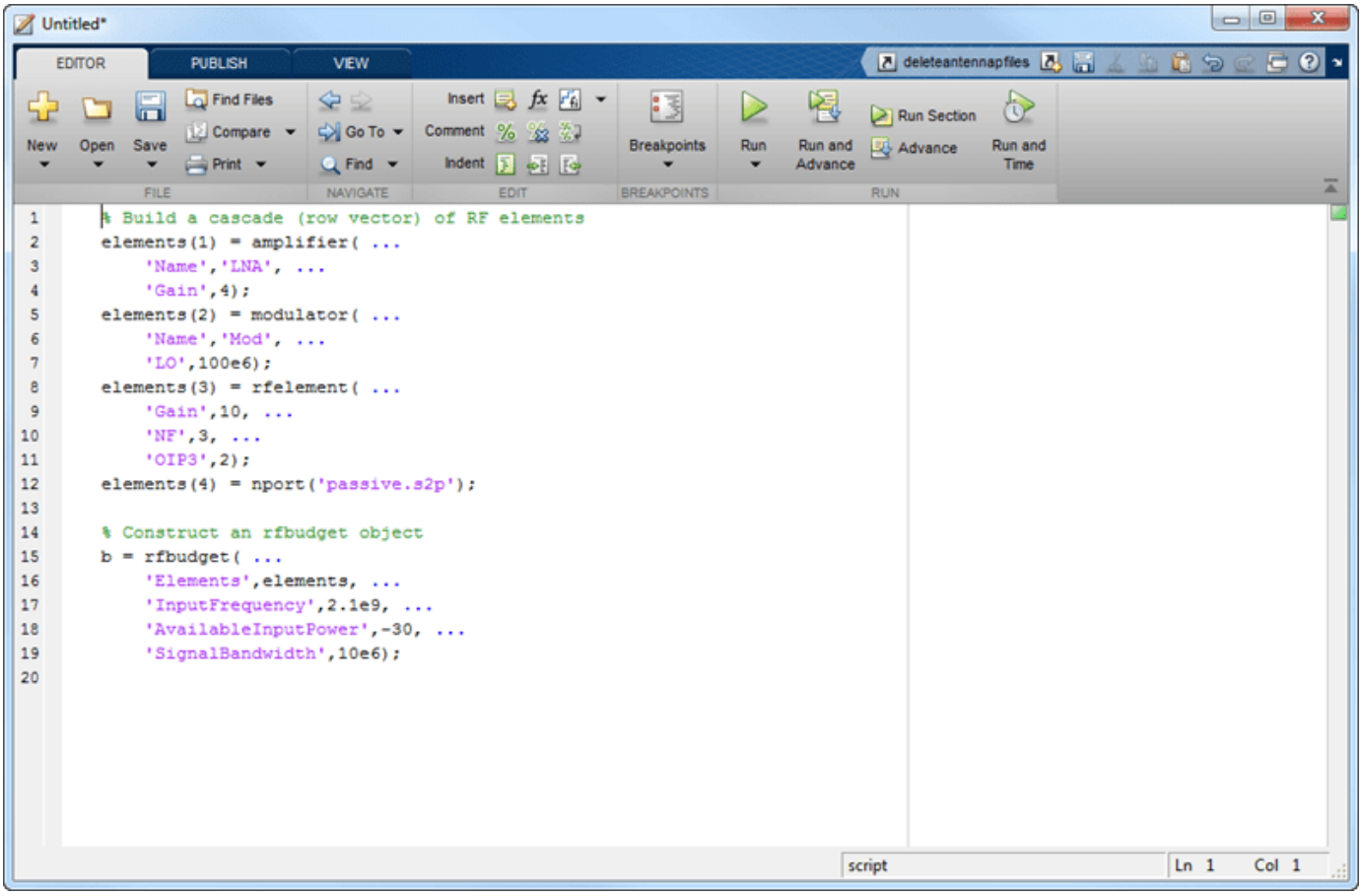
```
a = amplifier('Name','LNA','Gain',4);
m = modulator('ConverterType','Up','LO',100e6,'Name','Mod');
r = rfelement('Gain',10,'NF',3,'OIP3',2);
n = nport('passive.s2p');
```

Calculate the RF budget analysis.

```
b = rfbudget([a m r n],2.1e9,-30,10e6);
```

Export the analysis to a MATLAB script.

exportScript(b)



Version History

Introduced in R2017a

See Also

`rfbudget` | `show` | `computeBudget` | `exportRFBlockset` | `exportTestbench` | `rfplot` | `smithplot` | `polar`

exportRFBlockset

Create RF Blockset model from RF budget object

Syntax

```
exportRFBlockset(rfobj)
sys = exportRFBlockset(rfobj)
```

Description

`exportRFBlockset(rfobj)` creates an RF Blockset from the RF budget object, and opens the system.

`sys = exportRFBlockset(rfobj)` creates an RF Blockset model, and returns the system name.

Examples

RF Blockset Model From RF Budget Object

Create an amplifier with a gain of 4 dB.

```
a = amplifier(Gain=4);
```

Create a modulator with an OIP3 of 13 dBm.

```
m = modulator(OIP3=13);
```

Create an RF element with a gain of 10 dB.

```
r = rfelement(Gain=10);
```

Calculate the RF budget of a series of RF elements at an input frequency of 2.1 GHz, an available input power of –30 dBm, and a bandwidth of 10 MHz.

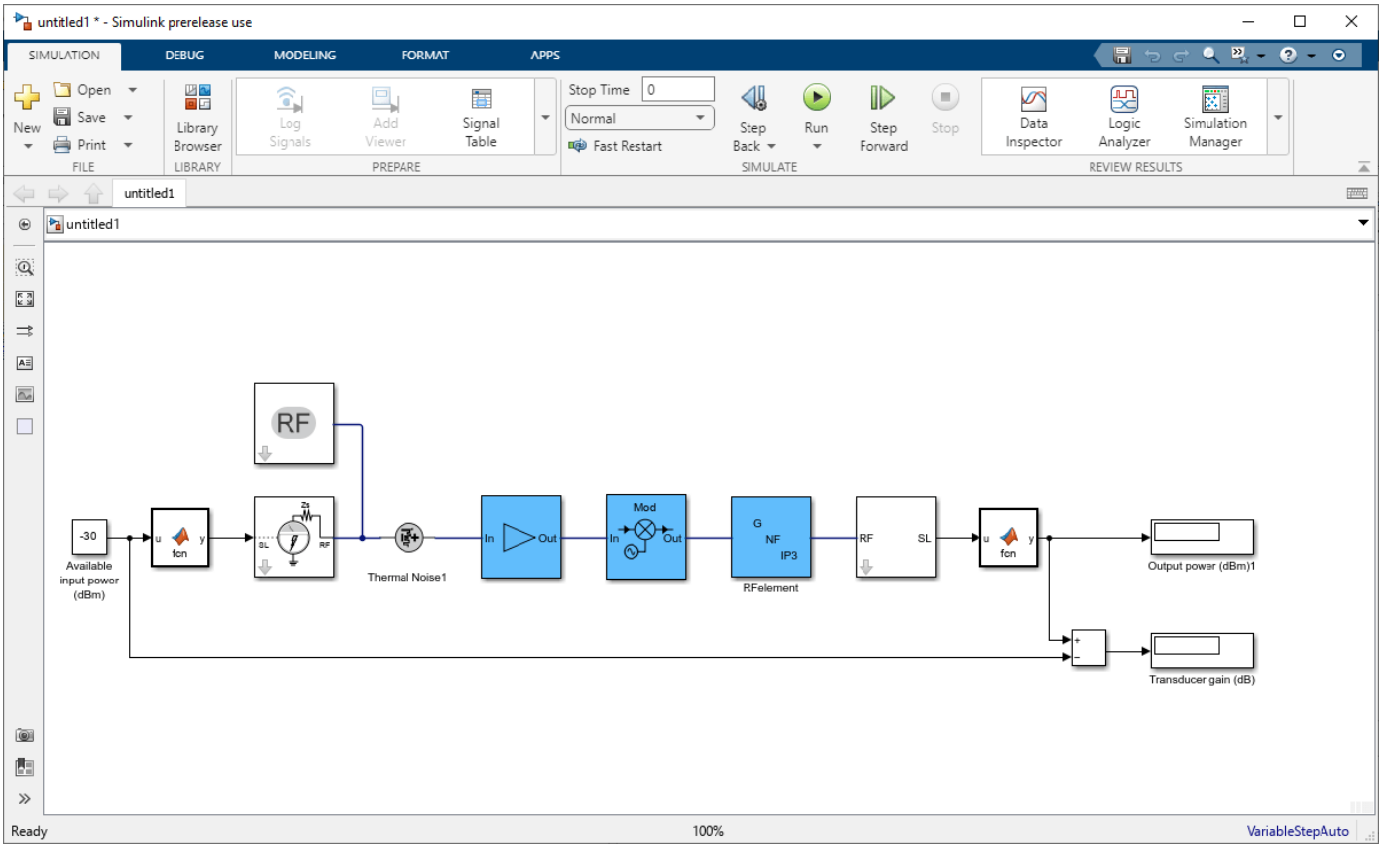
```
rfobj = rfbudget([a m r],2.1e9,-30,10e6)
```

```
rfobj =
  rfbudget with properties:

              Elements: [1x3 rf.internal.rfbudget.RFElement]
        InputFrequency: 2.1 GHz
    AvailableInputPower: -30 dBm
        SignalBandwidth:  10 MHz
                Solver: Friis
            AutoUpdate: true

   Analysis Results
        OutputFrequency: (GHz) [  2.1    3.1    3.1]
            OutputPower: (dBm) [  -26    -26    -16]
         TransducerGain: (dB)  [    4      4     14]
                     NF: (dB)  [    0      0      0]
```

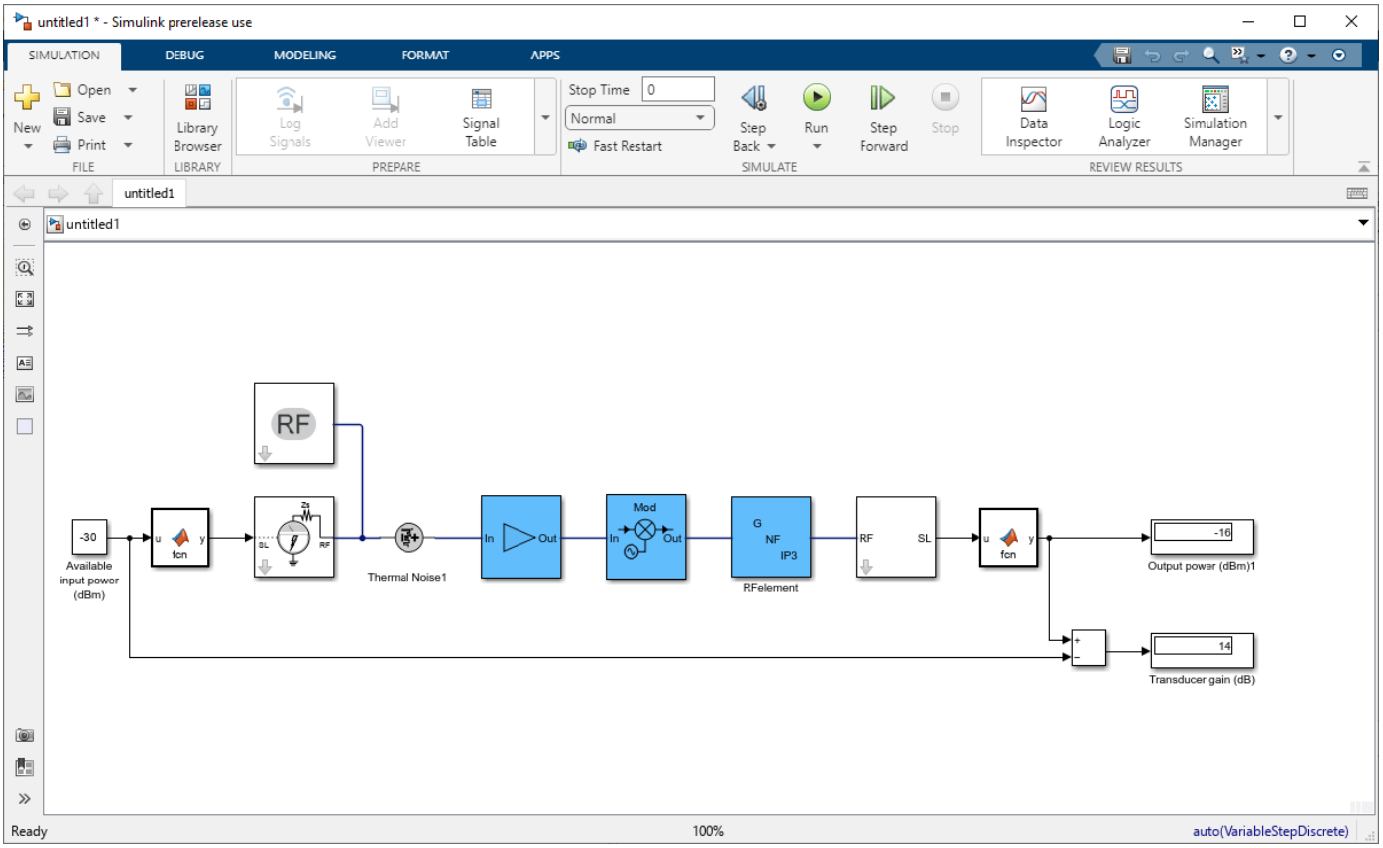
```
IIP2: (dBm) []
OIP2: (dBm) []
IIP3: (dBm) [  Inf      9      9]
OIP3: (dBm) [  Inf     13     23]
 SNR: (dB)  [73.98  73.98  73.98]
```

Create RF Blockset™ model from the RF budget object.

```
exportRFBlockset(rfobj)
```



Select **Run** from the **Simulate** section to run your RF Blockset model.



Input Arguments

`rfobj` — RF budget object

`rfbudget` object

RF budget object, specified as a `rfbudget` object.

Version History

Introduced in R2017a

See Also

`rfbudget` | `show` | `computeBudget` | `exportScript` | `exportTestbench` | `rfplot` | `smithplot` | `polar`

exportTestbench

Create measurement testbench from RF budget object

Syntax

```
exportTestbench(rfobj)
sys = exportTestbench(rfobj)
```

Description

`exportTestbench(rfobj)` creates an RF Blockset model from the RF budget object, and opens a measurement testbench system.

Note This function requires DSP System Toolbox™.

`sys = exportTestbench(rfobj)` creates an RF Blockset model, and returns the measurement testbench system.

Examples

Measurement Testbench From RF Budget Object

Create an amplifier with a gain of 4 dB.

```
a = amplifier(Gain=4);
```

Create a modulator with an OIP3 of 13 dBm.

```
m = modulator(OIP3=13);
```

Create an RF element with a gain of 10 dB.

```
r = rfelement(Gain=10);
```

Calculate the RF budget of a series of RF elements at an input frequency of 2.1 GHz, an available input power of –30 dBm, and a bandwidth of 10 MHz.

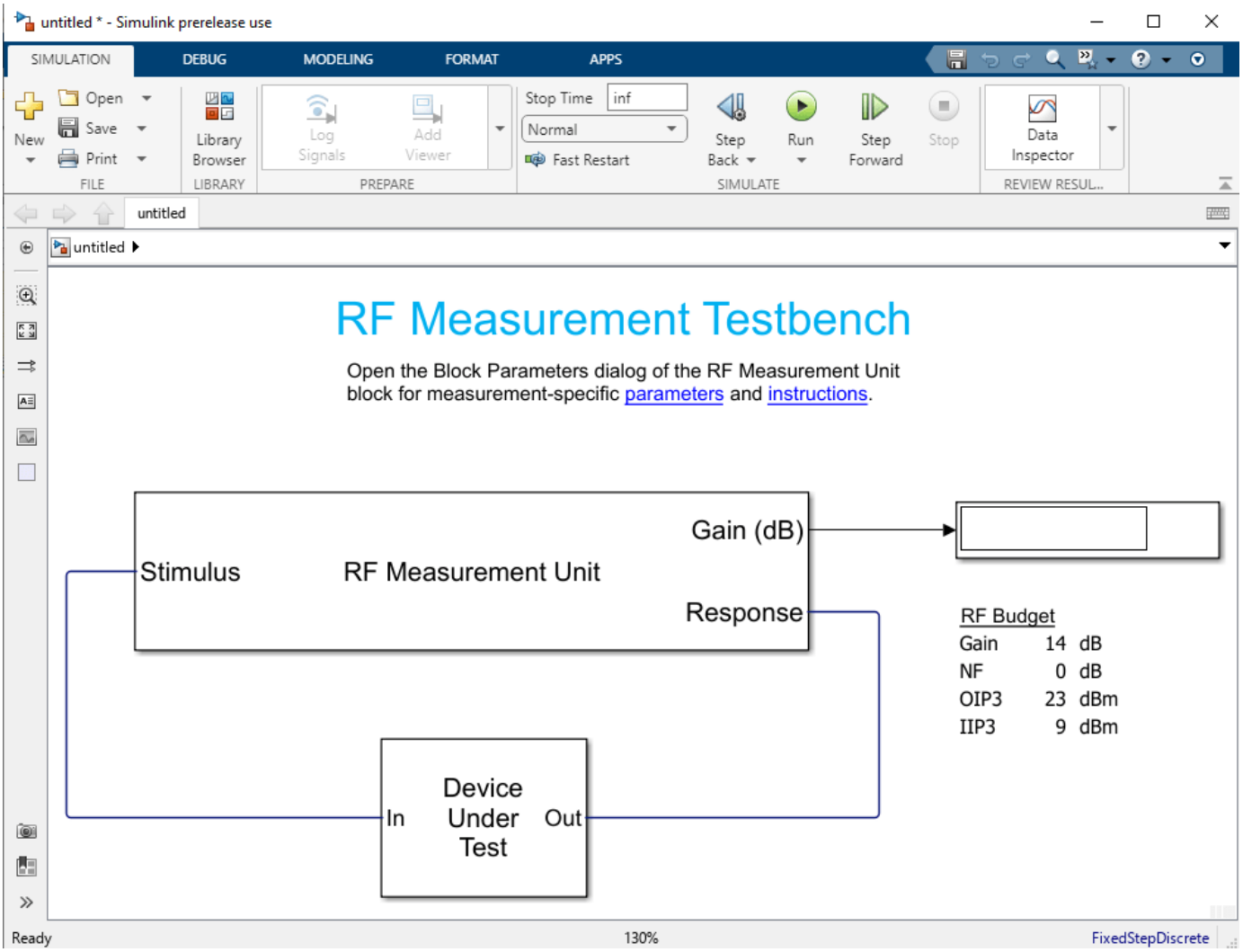
```
rfobj = rfbudget([a m r],2.1e9,-30,10e6)
```

```
rfobj =
  rfbudget with properties:

               Elements: [1x3 rf.internal.rfbudget.RFElement]
         InputFrequency: 2.1 GHz
    AvailableInputPower: -30 dBm
        SignalBandwidth:  10 MHz
                 Solver: Friis
             AutoUpdate: true
```

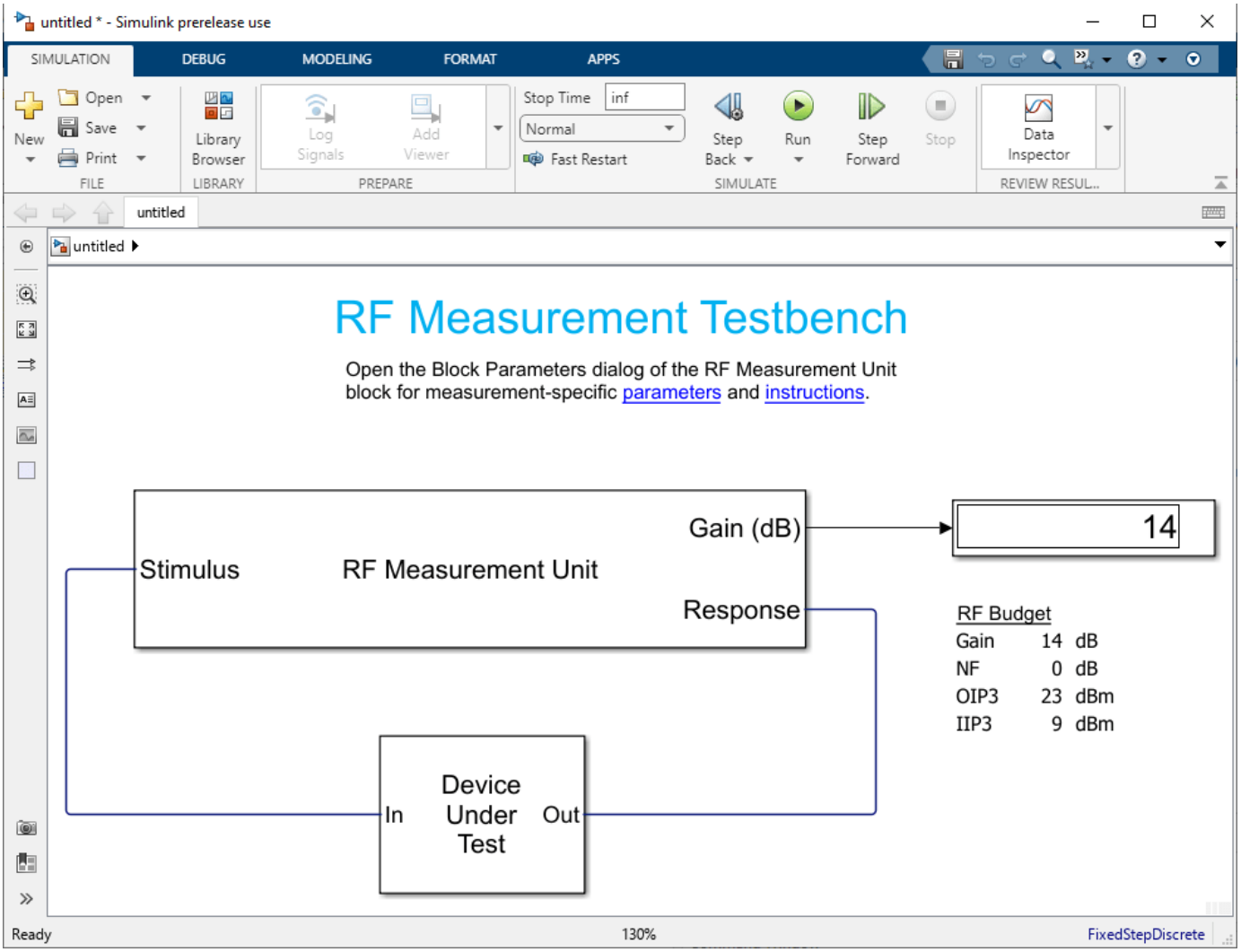
```
Analysis Results
     OutputFrequency: (GHz) [  2.1    3.1    3.1]
         OutputPower: (dBm) [  -26    -26    -16]
      TransducerGain: (dB)  [    4      4     14]
                  NF: (dB)  [    0      0      0]
               IIP2: (dBm) []
               OIP2: (dBm) []
               IIP3: (dBm) [  Inf      9      9]
               OIP3: (dBm) [  Inf     13     23]
                SNR: (dB)  [73.98  73.98  73.98]
```

Create the measurement testbench from the RF budget object.

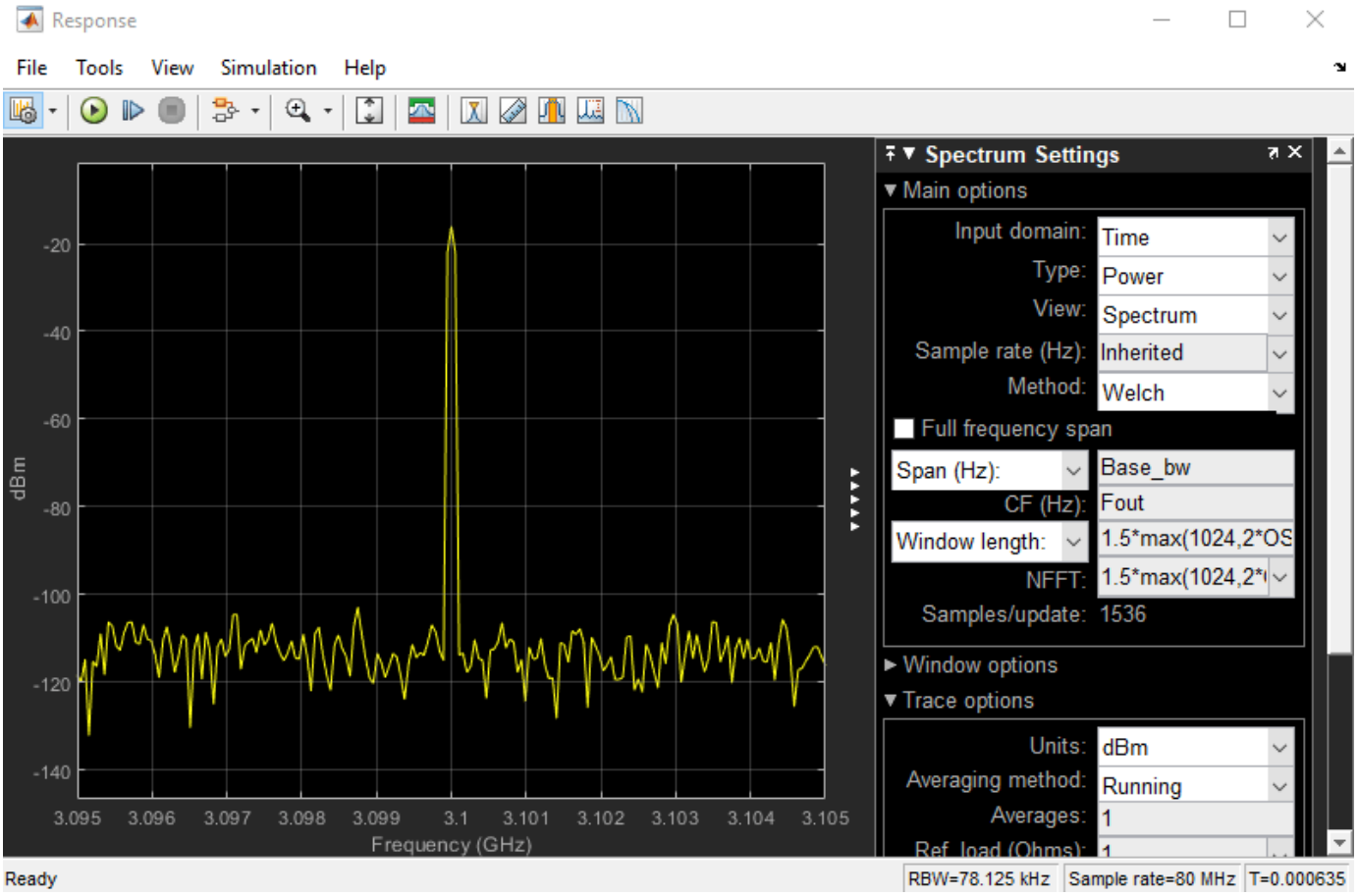
```
exportTestbench(rfobj)
```



Select **Run** to simulate your testbench.



The response will be displayed when you run the testbench.



Input Arguments

`rfobj` — RF budget object

`rfbudget` object

RF budget object, specified as a `rfbudget` object.

Version History

Introduced in R2017a

See Also

`rfbudget` | `show` | `computeBudget` | `exportScript` | `exportRFBlockset` | `rfplot` | `smithplot` | `polar`

show

Display RF budget object in RF Budget Analyzer app

Syntax

```
show(rfobj)
```

Description

`show(rfobj)` opens an **RF Budget Analyzer** app to display a clone of the `rfbudget` object.

Examples

Display RF Budget Analysis in RF Budget Analyzer App

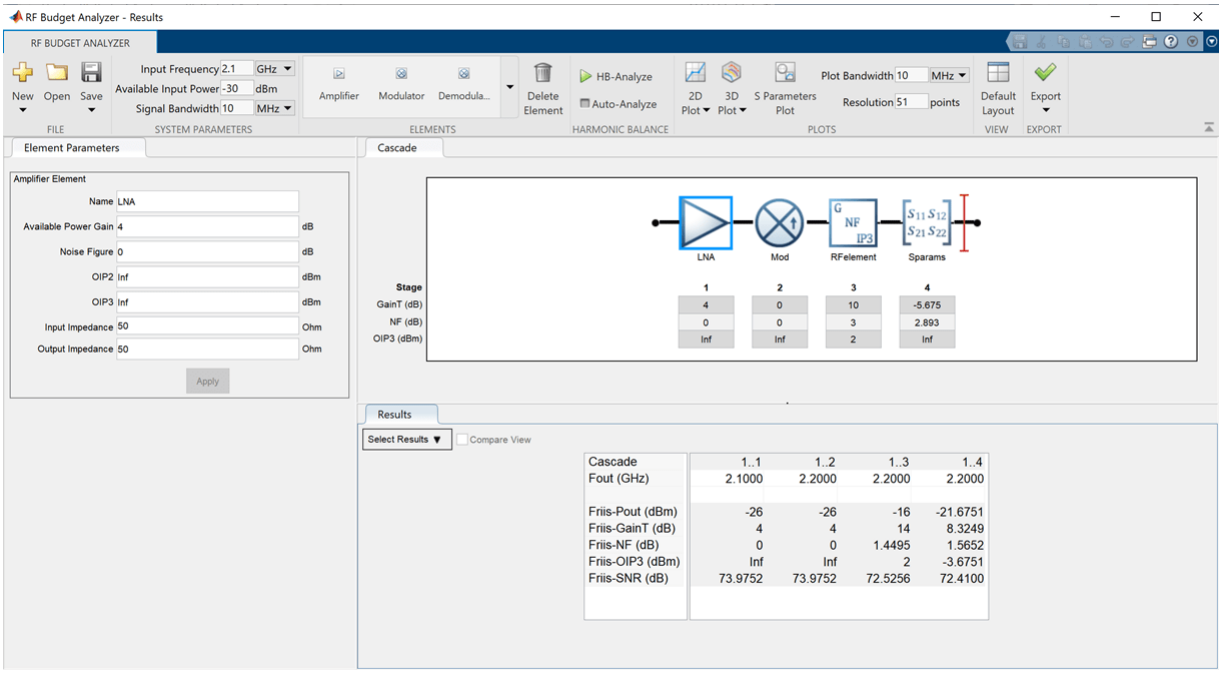
Create `amplifier`, `modulator`, `rfelement`, and `nport` objects for RF budget analysis.

```
a = amplifier('Name','LNA','Gain',4);
m = modulator('ConverterType','Up','LO',100e6,'Name','Mod');
r = rfelement('Gain',10,'NF',3,'OIP3',2);
n = nport('passive.s2p');
```

Compute RF budget results for chain of 2-port elements at an input frequency of `2.1` GHz, an available input power of `-30` dBm, and a bandwidth of `10` MHz.

```
b = rfbudget([a m r n],2.1e9,-30,10e6);
```

To display the RF budget results for exploration in the **RF Budget Analyzer** app, type `show(b)` in command line.



Input Arguments

`rfobj` — RF budget object

`rfbudget` object

RF budget object, specified as a `rfbudget` object.

Version History

Introduced in R2017a

See Also

`rfbudget` | `computeBudget` | `exportScript` | `exportRFBlockset` | `exportTestbench` | `rfplot` | `smithplot` | `polar`

rfplot

Plot cumulative RF budget result versus cascade input frequency

Syntax

```
rfplot(rfobj)
rfplot(rfobj,rfpara)
rfplot(rfobj,m,n)
rfplot(ax,___)
```

Description

`rfplot(rfobj)` plots the magnitude response of `S-Parameters`, S_{21} for the cascaded budget object, `rfobj`.

`rfplot(rfobj,rfpara)` plots the RF budget result specified by RF parameters `rfpara` versus a range of input frequencies. The input frequencies are applied to the cascade of elements in the RF budget object, `rfobj`.

Cumulative (that is, terminated subcascade) results are automatically computed to show the variation of the RF budget result through the entire design.

`rfplot(rfobj,m,n)` plots the magnitude response of `S-Parameters`, S_{mn} (S_{11}, S_{12}, S_{21} , or S_{22}) for the cascaded budget object, `rfobj`.

`rfplot(ax,___)` plots the cumulative RF budget result on the axes specified in `ax` instead of the current axes. Specify `ax` as the first input argument followed by any of the input argument combinations in the previous syntaxes. Return the current axes using the `gca` function.

Examples

Plot Cumulative Output Power and Gain of RF System

Create an RF system.

Create an RF bandpass filter using the Touchstone® file `RFBudget_RF`.

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

Create an amplifier with a gain of 11.53 dB, a noise figure (NF) of 1.53 dB, and an output third-order intercept (OIP3) of 35 dBm.

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

Create a demodulator with a gain of –6 dB, a NF of 4 dB, and an OIP3 of 50 dBm.

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

Create an IF bandpass filter using the Touchstone file `RFBudget_IF`.

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

Create an amplifier with a gain of 30 dB, a NF of 8 dB, and an OIP3 of 37 dBm.

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

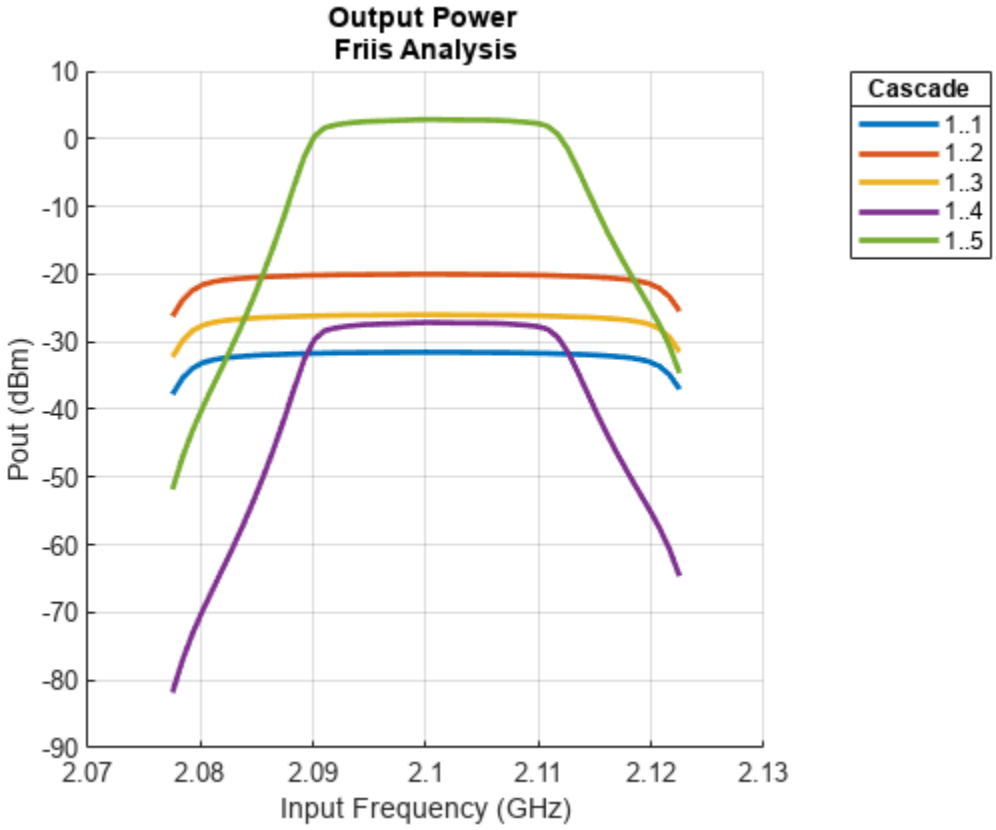
Calculate the RF budget of the system using an input frequency of 2.1 GHz, an input power of –30 dBm, and a bandwidth of 45 MHz.

```
b = rfbudget([f1 a1 d f2 a2],2.1e9,-30,45e6)
```

```
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]
```

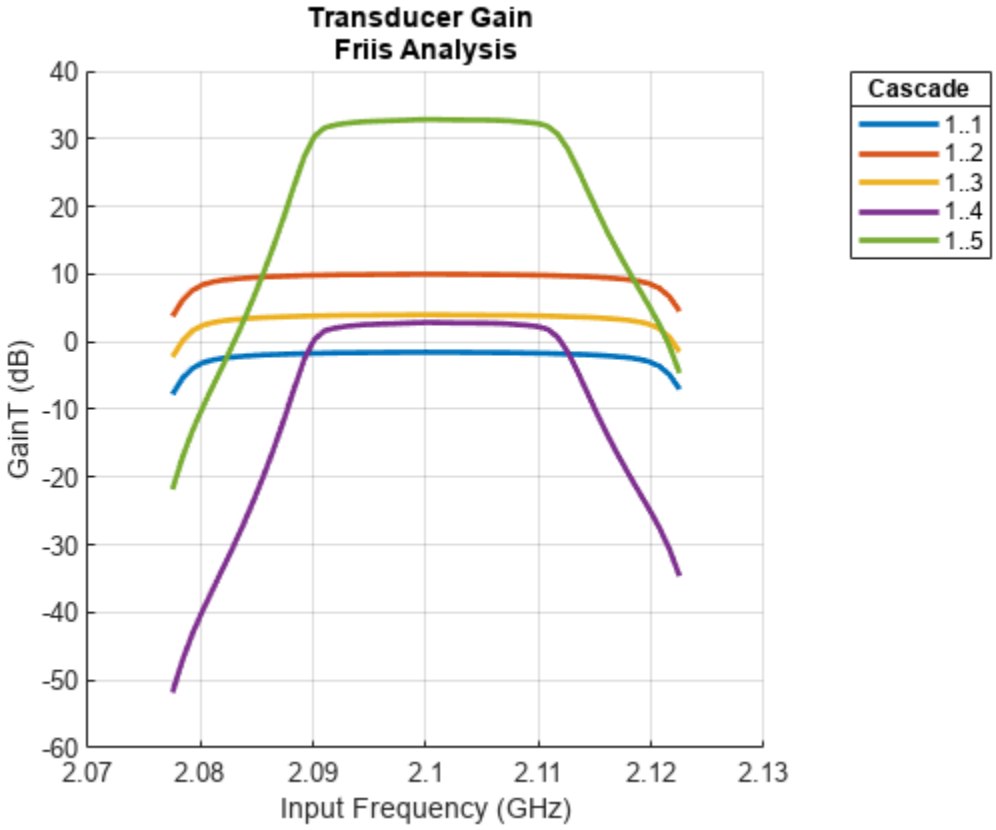
Plot the available output power.

```
rfplot(b,'Pout')
view(90,0)
```



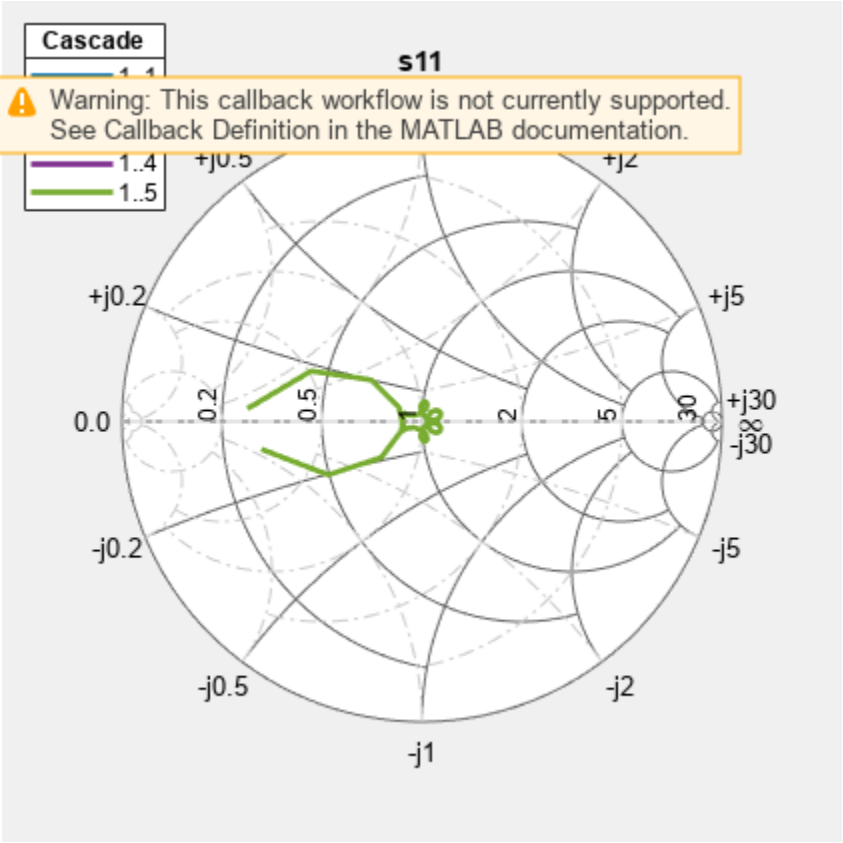
Plot the transducer gain.

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

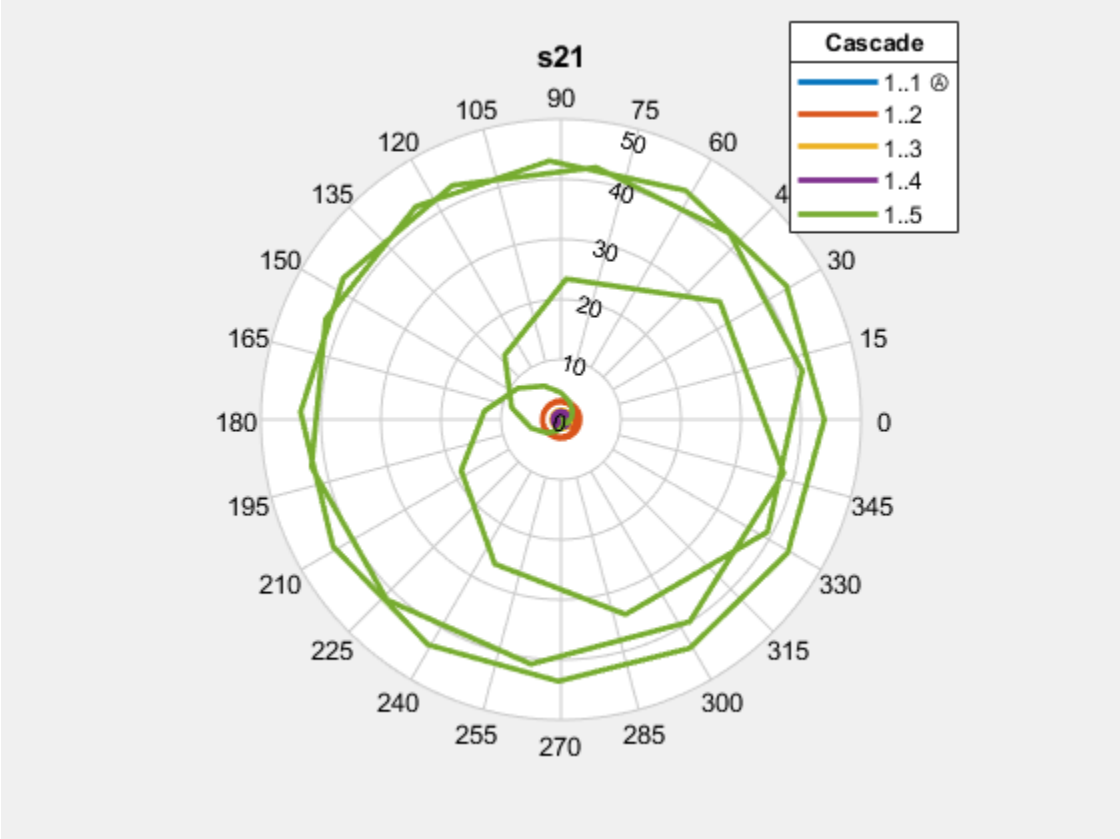


Plot S-parameters of an RF system on a Smith Chart and a Polar plot.

```
s = smithplot(b,1,1,'GridType','ZY');
```


Cascade
1..4
1..5
s11
Warning: This callback workflow is not currently supported.
See Callback Definition in the MATLAB documentation.
+j0.5
+j2
+j0.2
+j5
0.0
+j30
-j30
-j0.2
-j5
-j0.5
-j2
-j1
0.2
0.5
1
2
5
30
∞

```
p = polar(b,2,1);
```



Plot Phase and Group Delay of RF System

Create an amplifier with a gain of 4 dB.

```
a = amplifier(Gain=4);
```

Create a modulator with an OIP3 of 13 dBm.

```
m = modulator(OIP3=13);
```

Create an N-port element using `passive.s2p`.

```
n = nport('passive.s2p');
```

Create an RF element with a gain of 10 dB.

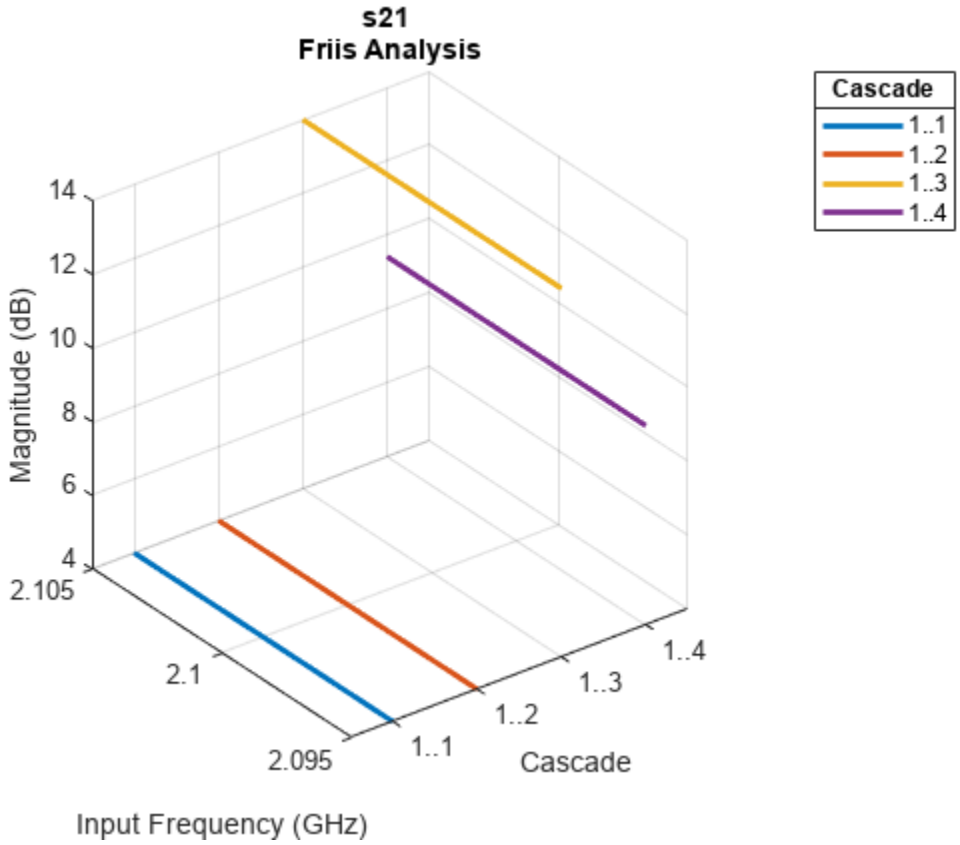
```
r = rfelement(Gain=10);
```

Calculate the RF budget of a series of RF elements at an input frequency of 2.1 GHz, an available input power of –30 dB, and a bandwidth of 10 MHz.

```
b = rfbudget([a m r n],2.1e9,-30,10e6);
```

Show the analysis in the RF plot.

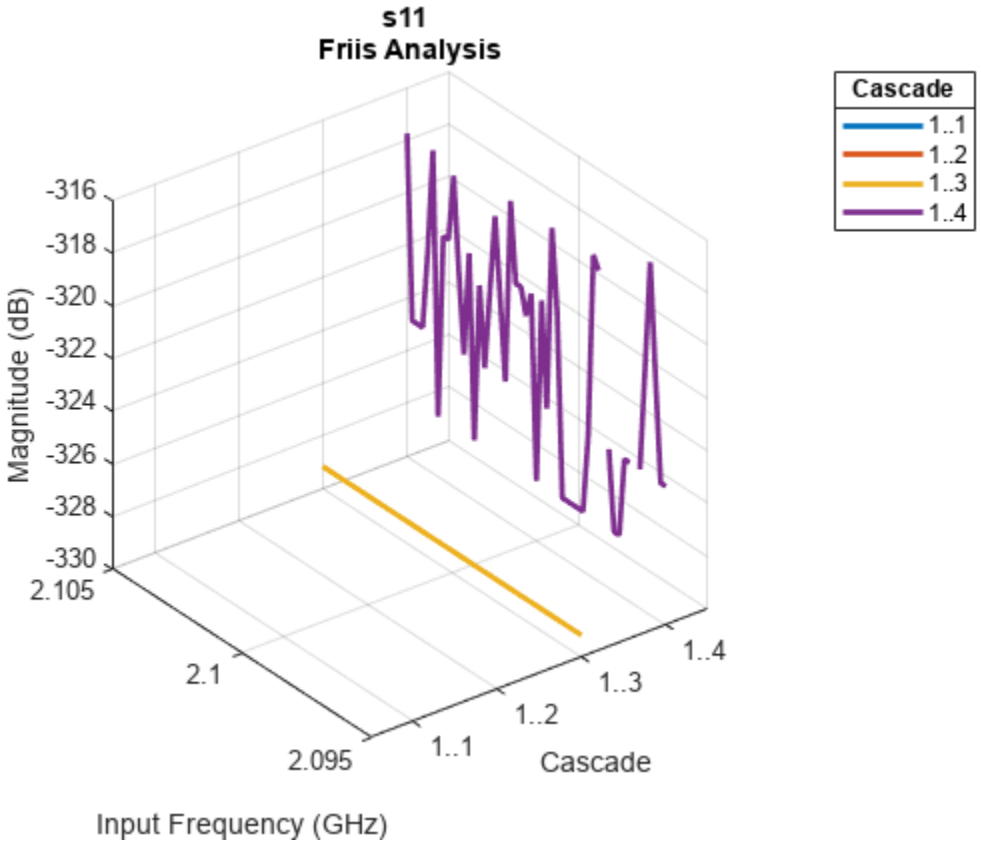
```
rfplot(b)
```



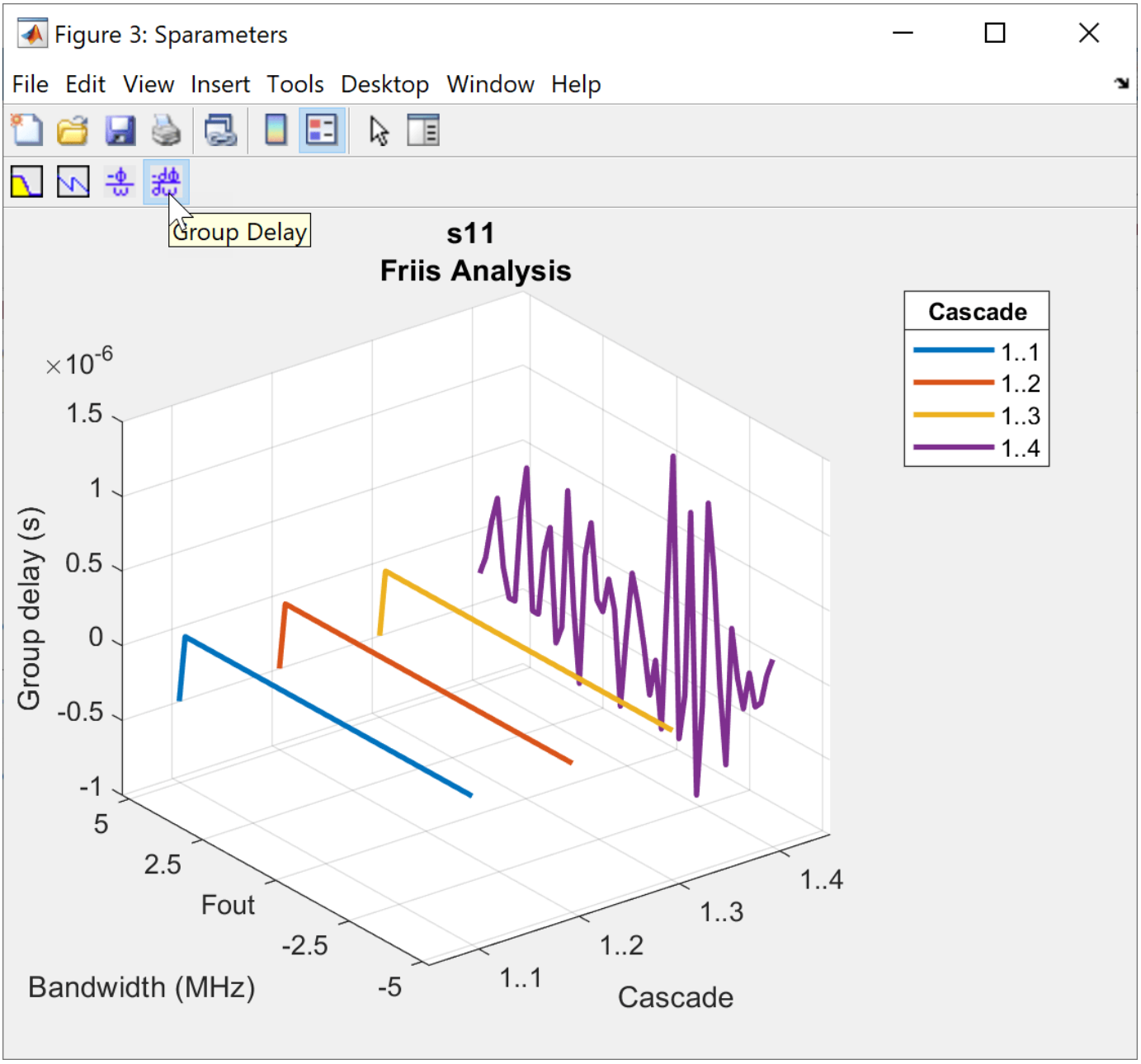
Group Delay

To plot the group delay, first plot the S11 data for the RF System.

```
rfplot(b,1,1)
```

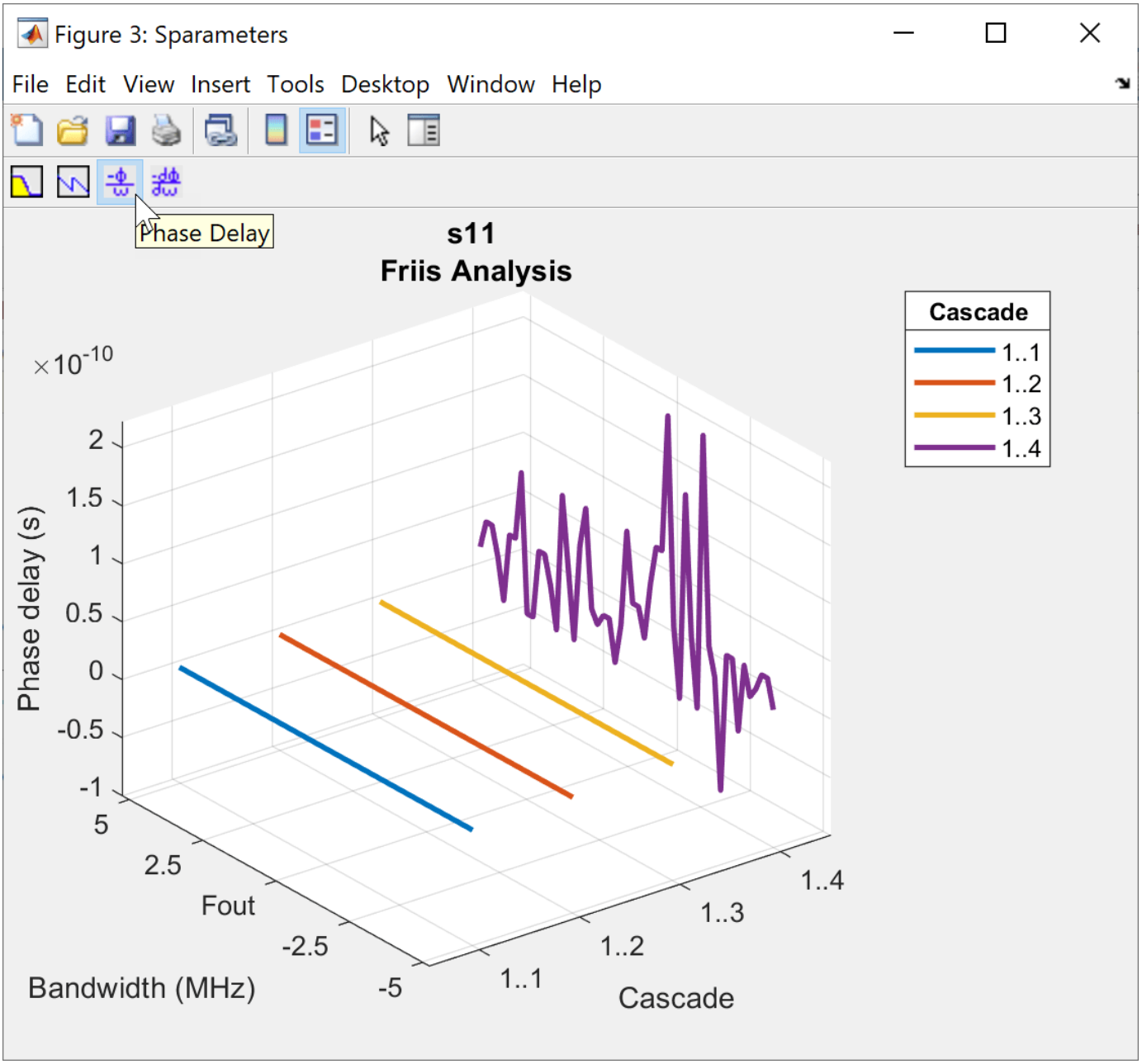


Use the `Group Delay` option on the plot graph to plot the group delay of the RF system.



Phase Delay

Use the `Phase Delay` option on the plot graph to plot the phase delay of the RF System.



Input Arguments

`rfobj` — RF budget object
`rfbudget` object

RF budget object, specified as a `rfbudget` object.

Example: `rfplot(rfobj,'Pout')`

`rfpara` — RF parameters
`'Pout'` | `'GainT'` | `'NF'` | `'OIP3'` | `'IIP3'` | `'SNR'`

RF parameters, specified as one of the following:

- `'Pout'` - Available output power (dBm)
- `'GainT'` - Transducer gain (dB)
- `'NF'` - Noise Figure (dB)
- `'OIP3'` - Output Third-Order Intercept (dBm)
- `'IIP3'` - Input Third-Order Intercept (dBm)
- `'SNR'` - Signal-to-Noise Ratio (dB)
- `'Sparameters'` - S - Parameters S_{21} magnitude response (dB)

Example: `rfplot(rfobj,'Pout')` where `'Pout'` is the available output power of an RF system obtained from the RF budget analysis.

ax — Axes object
`axes` object | `uiaxes` object

Axes object, specified as an `axes` or a `uiaxes` object.

Version History

Introduced in R2017b

See Also

`rfbudget` | `show` | `computeBudget` | `exportScript` | `exportRFBlockset` | `exportTestbench` | `rfplot` | `smithplot` | `polar`

getSpurFreeZoneData

Return frequency data related to the spur-free zones in multiband transmitter or receiver frequency space

Syntax

```
allfrequencyzones = getSpurFreeZoneData(hif)
```

Description

`allfrequencyzones = getSpurFreeZoneData(hif)` returns frequency data related to the spur-free zones in multiband transmitter or receiver frequency space. Each zone is a range of IF center frequencies. An IF centered in this range does not generate interference in any transmission or reception bands.

Examples

Spur-Free zones

Calculate spur-free zones.

```
h = OpenIF('IFLocation','MixerOutput');
```

Add two mixers to the system.

```
IMT1 = [99 0 21 17 26; 11 0 29 29 63; ...
       60 48 70 65 41; 90 89 74 68 87; 99 99 95 99 99];
addMixer(h,IMT1,2400e6,100e6,'low',50e6)
IMT2 = [99 0 9 12 15; 20 0 26 31 48; ...
       55 70 51 70 53; 85 90 60 70 94; 96 95 94 93 92];
addMixer(h,IMT2,3700e6,150e6,'high',50e6)
```

Check for spur-free zones.

```
sfzfreqs = getSpurFreeZoneData(h)
```

```
sfzfreqs = 7×2
10^9 ×

    0.2500    0.4300
    0.5300    0.5563
    0.6438    0.7167
    1.0375    1.1125
    1.3417    1.4100
    1.4700    1.5333
    2.0750    2.3000
```

Input Arguments

`hif` — OpenIF object

object handle

OpenIF object, specified as an object handle.

Output Arguments

`allfrequencyzones` — Spur-free zones

K-by-2 matrix

Spur-free zones of a defined network, returned as a *K*-by-2 matrix. *K* is the number of spur free zones. The two columns in the matrix contain the start and stop frequencies of each spur-free zone. The first column contains the start frequencies and the second column contains the stop frequencies.

Alternative Functionality

- The `report` method displays mixer configurations, intermodulation tables, and spur-free zone information at the command line.
- The `show` method generates an interactive spur graph that shows spurious regions and spur-free zones.

Version History

Introduced in R2011b

See Also

`getSpurData`

getSpurData

Return frequency data related to the spurs in multiband transmitter or receiver frequency space

Syntax

```
allfrequencies = getSpurData(hif)
[allfrequencies,dBs,mixers,mns = getSpurData
```

Description

`allfrequencies = getSpurData(hif)` Return frequency data related to the spurs in multiband transmitter or receiver frequency space. Each spur is a range of frequencies.

`[allfrequencies,dBs,mixers,mns = getSpurData` returns relevant data for all spurs calculated by `OpenIF` object.

Examples

Spur Data

Setup the object.

```
 h = OpenIF('IFLocation','MixerOutput');
```

Add two mixers to the system

```
IMT1 = [99 0 21 17 26; 11 0 29 29 63; ...
        60 48 70 65 41; 90 89 74 68 87; 99 99 95 99 99];
addMixer(h,IMT1,2400e6,100e6,'low',50e6)
IMT2 = [99 0 9 12 15; 20 0 26 31 48; ...
        55 70 51 70 53; 85 90 60 70 94; 96 95 94 93 92];
addMixer(h,IMT2,3700e6,150e6,'high',50e6)
```

Get spur free data.

```
[allspurs,dBs,mixers,mn] = getSpurData(h)
```

```
allspurs = 33×2
10^9 ×

    1.9000    1.9400
    1.4100    1.4700
    0.9200    1.0000
    1.5333    1.6667
    0.4300    0.5300
    0.7167    0.8833
    1.7812    1.8188
    1.1687    1.2312
    2.3375    2.3500
    0.5563    0.6438
      ⋮
```

```
dBs = 33×1

    26
    63
    41
    41
    87
    87
    17
    29
    29
    65
     ⋮

mixers = 33×1

     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     ⋮

mn = 33×2

    -4     0
    -4     1
    -4     2
    -4     2
    -4     3
    -4     3
    -3     0
    -3     1
    -3     1
    -3     2
     ⋮
```

Input Arguments

`hif` — OpenIF object

object handle

OpenIF object, specified as an object handle.

Output Arguments

`allfrequencies` — Start and stop frequencies of spur data

K-by-2 matrix

Start and stop frequencies of spur data, returned as a *K*-by-2 matrix or *K*-by-1 matrix. *K* is the number of spurs. The two columns in the matrix contain the start and stop frequencies of each spur-free zone. The first column contains the start frequencies and the second column contains the stop frequencies.

`dBs` — Decibel carpet value (dBc) of spur data
K-by-1 matrix

Decibel carpet value (dBc) value of spur data, returned as a *K*-by-1 matrix. Each K is a dBc value (relative to the output) of that spur.

`mixers` — Mixer that caused the spur
K-by-1 matrix

Mixer that caused the spur, returned as a *K*-by-1 matrix. Each K is the mixer that caused the spur.

`mns` — *M* and *N* values used to calculate the spur
K-by-2 matrix

M and *N* values used to calculate the spur, returned as a *K*-by-2 matrix.

Version History

Introduced in R2011b

See Also

`getSpurFreeZoneData`

report

Summarize IF planning results in command window

Syntax

```
report(hif)
```

Description

`report(hif)` returns the summary of IF planning results in command window. The summary contains:

- The IF location.
- The properties of each mixer, including RF center frequencies, bandwidths, mixing type, and intermodulation tables.

  The spur-free zones.

Examples

Values in OpenIF

Set up the object

```
h = OpenIF('IFLocation','MixerOutput');
```

Add two mixers to the system

```
IMT1 = [99 0 21 17 26; 11 0 29 29 63; ...
       60 48 70 65 41; 90 89 74 68 87; 99 99 95 99 99];
addMixer(h,IMT1,2400e6,100e6,'low',50e6)
IMT2 = [99 0 9 12 15; 20 0 26 31 48; ...
       55 70 51 70 53; 85 90 60 70 94; 96 95 94 93 92];
addMixer(h,IMT2,3700e6,150e6,'high',50e6)
```

Check for spur-free zones

```
report(h)
```

```
     Intermediate Frequency (IF) Planner
     IF Location: MixerOutput

     -- MIXER 1 --
     RF Center Frequency: 2.4 GHz
     RF Bandwidth: 100 MHz
     IF Bandwidth: 50 MHz
     MixerType: low
     Intermodulation Table:   99   0  21  17  26
                              11   0  29  29  63
                              60  48  70  65  41
                              90  89  74  68  87
```

```
                          99  99  95  99  99

-- MIXER 2 --
RF Center Frequency: 3.7 GHz
RF Bandwidth: 150 MHz
IF Bandwidth: 50 MHz
MixerType: high
Intermodulation Table:   99   0   9  12  15
                         20   0  26  31  48
                         55  70  51  70  53
                         85  90  60  70  94
                         96  95  94  93  92

Spur-Free Zones:
 250.00 -  430.00 MHz
 530.00 -  556.25 MHz
 643.75 -  716.67 MHz
   1.04 -    1.11 GHz
   1.34 -    1.41 GHz
   1.47 -    1.53 GHz
   2.08 -    2.30 GHz
```

Input Arguments

`hif` — OpenIF object

object handle

OpenIF object, specified as an object handle.

Version History

Introduced in R2011b

See Also

`OpenIF`

show

Graphical summary of all relevant spurs and spur-free zones

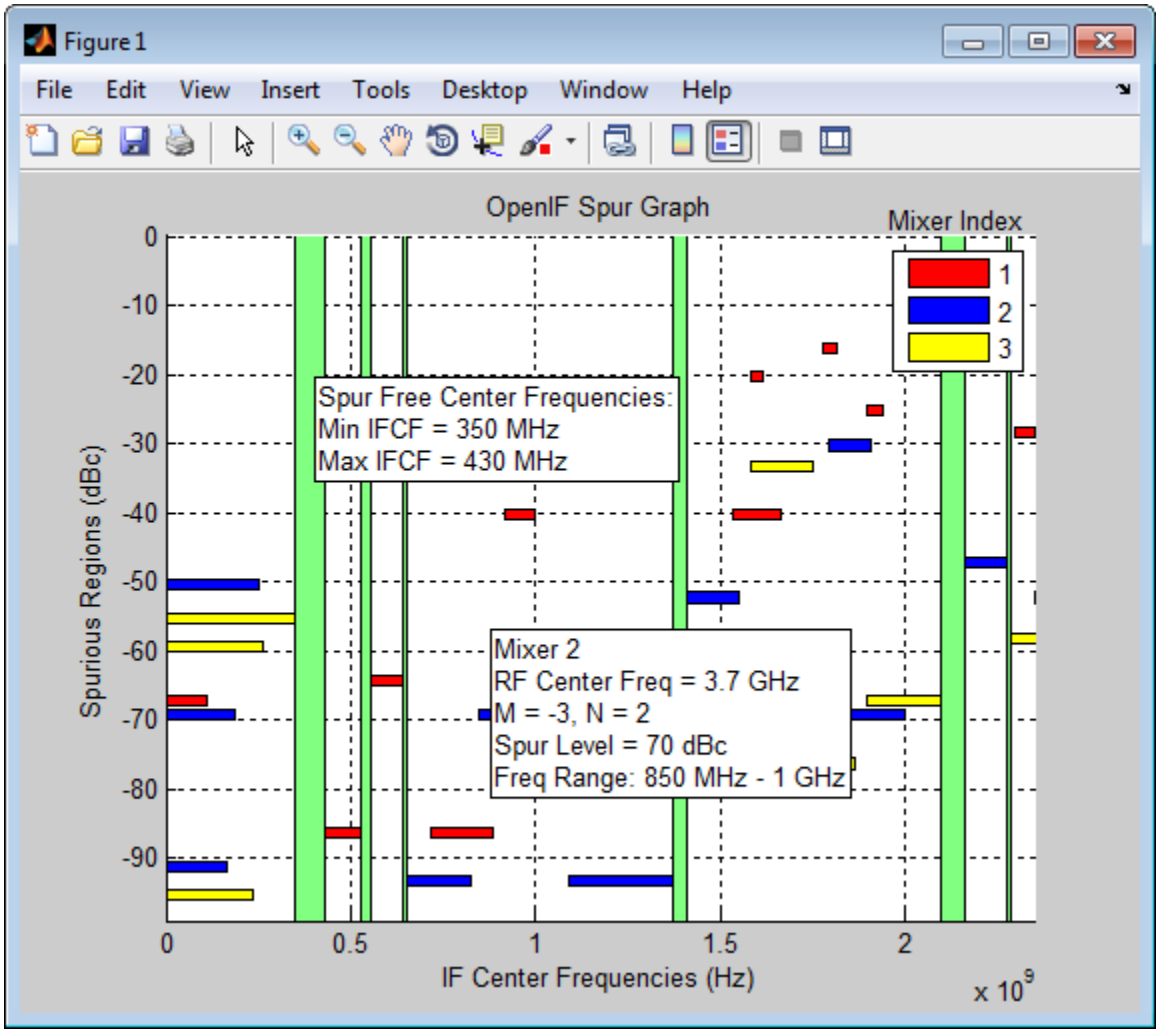
Syntax

```
show(hif)
```

Description

`show(hif)` produces a spur graph of the `OpenIF` object `hif`. The spur graph contains:

- Vertical green bands, representing spur-free zones.
- Horizontal colored bands, representing spurious regions.



Spur-free zones are ranges of possible IF center frequencies that are free from intermodulation distortion. Depending on the configuration of the mixers in `hif`, spur-free zones may not appear. Clicking a spur-free zone produces a tool tip, which displays information about the spur-free zone:

- **Min IFCF** — The minimum possible IF center frequency f_{IF} for the corresponding spur-free zone.
- **Max IFCF** — The maximum IF center frequency f_{IF} for the corresponding spur-free zone.

Spurious regions contain intermodulation products from at least one mixer. The color of a spur on the spur graph indicates which mixer generates the spur, according to the legend on the spur graph. Clicking a spurious region produces a tool tip, which displays information about the spur:

- **RF Center Freq** — The RF center frequency f_{RF} of the mixer that generates the spur
- **M**, **N** — The coefficients in the equation $|Mf_{RF} - N(f_{RF} \pm f_{IF})|$ (down-conversion) or the equation $|Mf_{IF} + N(f_{RF} \pm f_{IF})|$. Injection type of the receiver determines the sign in the equations. These coefficients refer to the particular mixing product that generates the spurious region.
- **Spur Level** — The difference in magnitude between a signal at 0 dBc and the spur. If you set `hif.SpurLevel` to a number greater than this value, then `hif` does not report the region as spurious.
- **Freq Range** — The frequency range of the spurious region. Choosing an IF center frequency in this range causes interference with the intermodulation product corresponding to the spur.

Examples

Show Spur-Free Zones

Set up the object

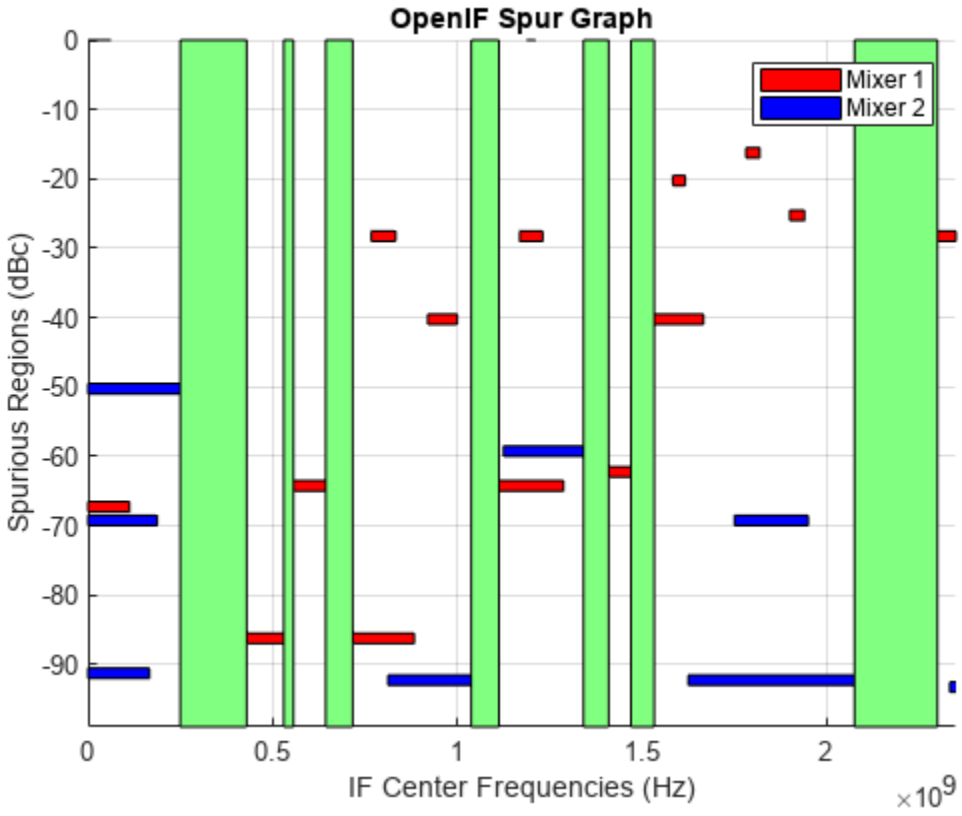
```
h = OpenIF('IFLocation','MixerOutput');
```

Add two mixers to the system

```
IMT1 = [99 0 21 17 26; 11 0 29 29 63; ...
       60 48 70 65 41; 90 89 74 68 87; 99 99 95 99 99];
addMixer(h,IMT1,2400e6,100e6,'low',50e6)

IMT2 = [99 0 9 12 15; 20 0 26 31 48; ...
       55 70 51 70 53; 85 90 60 70 94; 96 95 94 93 92];
addMixer(h,IMT2,3700e6,150e6,'high',50e6)
```

Check for spur-free zones

```
show(h)
```



Input Arguments

`hif` — OpenIF object

object handle

OpenIF object, specified as an object handle.

Version History

Introduced in R2011b

See Also

circle

Draw circles on Smith Chart

Syntax

```
[hsm_out] = circle(rfcktobject,freq,type1,value1,....,typen,valuen,hsm1_out)
[hlines,hsm] = circle(rfcktobject,freq,type1,value1,....,typen,valuen,
hsm_out)
```

Description

`[hsm_out] = circle(rfcktobject,freq,type1,value1,....,typen,valuen,hsm1_out)` draws the specified circles on a Smith chart created using the `smithplot` function. The syntax returns an existing `smithplot` handle.

`[hlines,hsm] = circle(rfcktobject,freq,type1,value1,....,typen,valuen,
hsm_out)` draws the specified circles on a Smith chart. This syntax returns vector handles of line objects and handles of the Smith chart.

Examples

Draw Circles on Smith Chart Created using `smithplot` Function

Create an amplifier object from `default.s2p`.

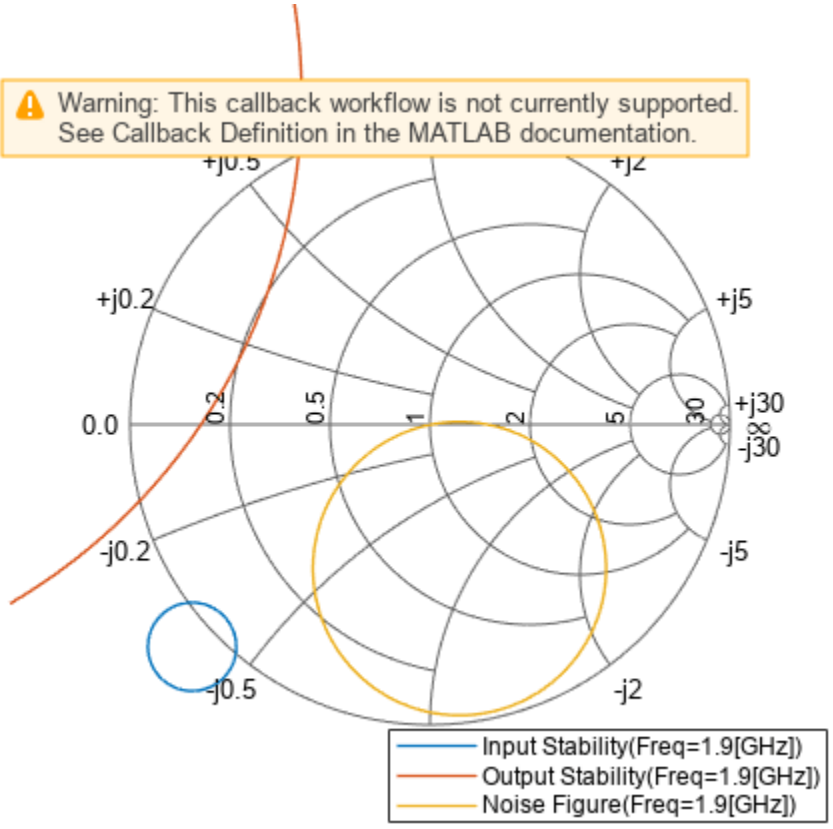
```
amp = read(rfckt.amplifier,'default.s2p');
```

Plot the noise figure of the amplifier 1.9 GHz using a Smith chart created using `smithplot` function

```
fc = 1.9e9;
h = smithplot
```

```
h =
  smithplot with properties:

         Data: []
    Frequency: []

   Show all properties, methods
```

```
circle(amp,fc,'Stab','In','Stab','Out','NF',10.396,h);
legend('Location','SouthEast')
```



Input Arguments

`rfcktobject` — RF Toolbox rfckt object

object handle

RF Toolbox rfckt object, specified as an object handle,

`freq` — Single frequency point of interest

scalar

Single frequency point of interest, specified as a scalar in Hz.

Data Types: `double`

`type1,value1,....,typen,valuen` — Type value pairs specifying circles to plots

character vector | string scalar

Type value pairs specifying circles to plots, specified as a character vector or a string scalar.

The following table lists the supported circle type options:

type	Definition
`'Ga'`	Constant available power gain circle
`'Gp'`	Constant operating power gain circle

type	Definition
'Stab'	Stability circle
'NF'	Constant noise figure circle
'R'	Constant resistance circle
'X'	Constant reactance circle
'G'	Constant conductance circle
'B'	Constant susceptance circle
'Gamma'	Constant reflection magnitude circle

The following table lists the circle value options:

value	Definition
'Ga'	Scalar or vector of gains in dB
'Gp'	Scalar or vector of gains in dB
'Stab'	'in' or 'source' for input/source stability circle; 'out' or 'load' for output/load stability circle
'NF'	Scalar or vector of noise figures in dB
'R'	Scalar or vector of normalized resistance
'X'	Scalar or vector of normalized reactance
'G'	Scalar or vector of normalized conductance
'B'	Scalar or vector of normalized susceptance
'Gamma'	Scalar or vector of non-negative reflection magnitude

Data Types: char | string

hsm1_out — Existing Smith plot handle
object handle

Existing Smith chart handle created using smithplot function, specified as an object handle. You can obtain the object handle using hsm1 = smithplot('gco').

hsm — Existing Smith chart handle
object handle

Existing Smith chart handle, specified as an object handle.

Output Arguments

hlines — Line objects for circle specifications
vector of line handle

Line objects for circle specifications, returned as a vector of line handles.

hsm — Smith chart
object handle

Smith chart, returned as an object handle.

`hsm_out` — Smith chart created using `smithplot` function
object handle

Smith chart created using `smithplot` function, returned as an object handle.

Version History

Introduced in R2007b

See Also

`smithplot`

Topics
“Designing Matching Networks for Low Noise Amplifiers”

semilogy

Plot RF circuit object parameters using log scale for *y*-axis

Syntax

```
semilogy(h,circuitPara)
semilogy(h,circuitPara,dataFormat)
semilogy( ___ ,xAxisPara,xAxisFmt)
semilogy( ___ ,opCon,opVal)
semilogy( ___ ,Name,Value)
lineseries = semilogy( ___ )
```

Description

`semilogy(h,circuitPara)` plots the circuit parameter `circuitPara` from the RFCKT or RF data object `h` using a logarithmic scale for the *y*-axis. You can specify multiple circuit parameters in this syntax.

Note For all circuit objects except those that contain data from a data file, you must perform a frequency domain analysis with the `analyze` method before calling `semilogy`.

`semilogy(h,circuitPara,dataFormat)` plots the data of the RFCKT or RF data object using a logarithmic scale for the *y*-axis with the specified data format.

`semilogy( ___ ,xAxisPara,xAxisFmt)` plots the circuit parameters `circuitPara` using a logarithmic scale for the *y*-axis along with the variables `xAxisPara` and their corresponding format `xAxisFmt`. Specify `xAxisPara` and `xAxisFmt` arguments after any of the input argument combinations in the previous syntaxes.

`semilogy( ___ ,opCon,opVal)` plots the circuit parameters using a logarithmic scale for the *y*-axis with operating conditions `opCon` and operating values `opVal` for the circuit object `h`.

Derive operating conditions for the RFCKT or RF data object `h` using the `getop(h)` command

`semilogy( ___ ,Name,Value)` plots the data of a RFCKT or RF data object with name-value arguments.

`lineseries = semilogy( ___ )` returns the line series property object `lineseries`. This output is the same as the output returned by the MATLAB `semilogy` function.

Examples

Plot S11 and S22 of Amplifier

Create an amplifier object from `default.s2p` file.

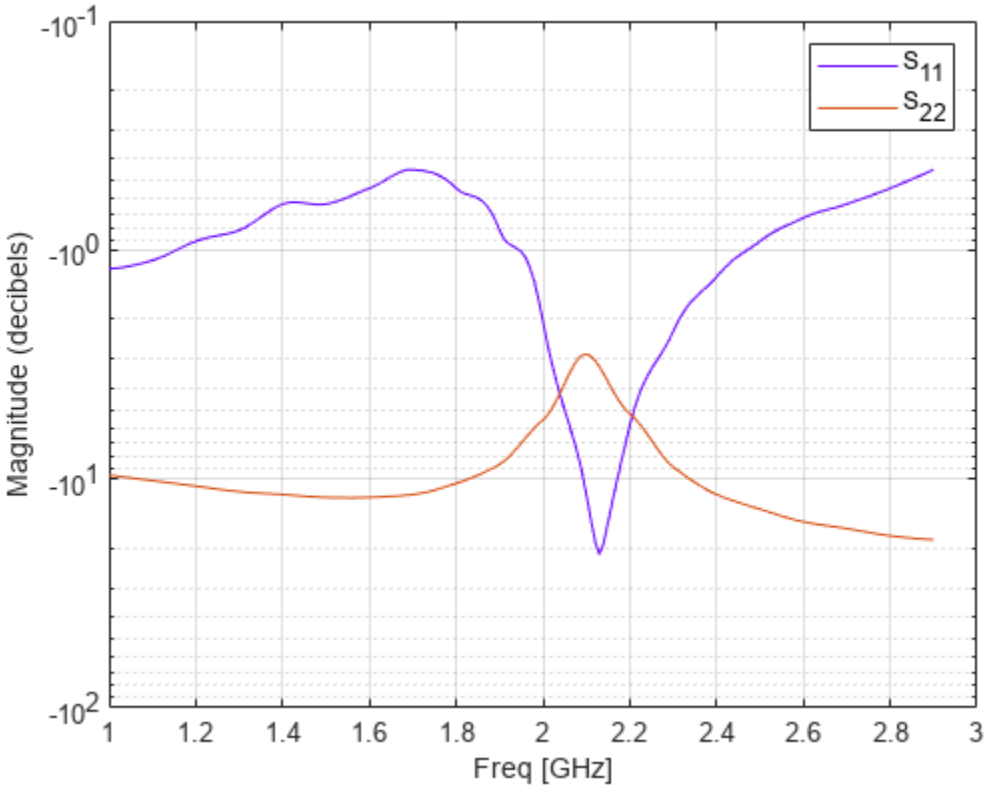
```
h = read(rfckt.amplifier,'default.s2p');
```

Plot S11 and S22 using log scale on x-axis.

```
lineseries = semilogy(h,'S11','S22');
```

Change the color of the S11 data.

```
lineseries(1).Color = [0.4 0 1];
```



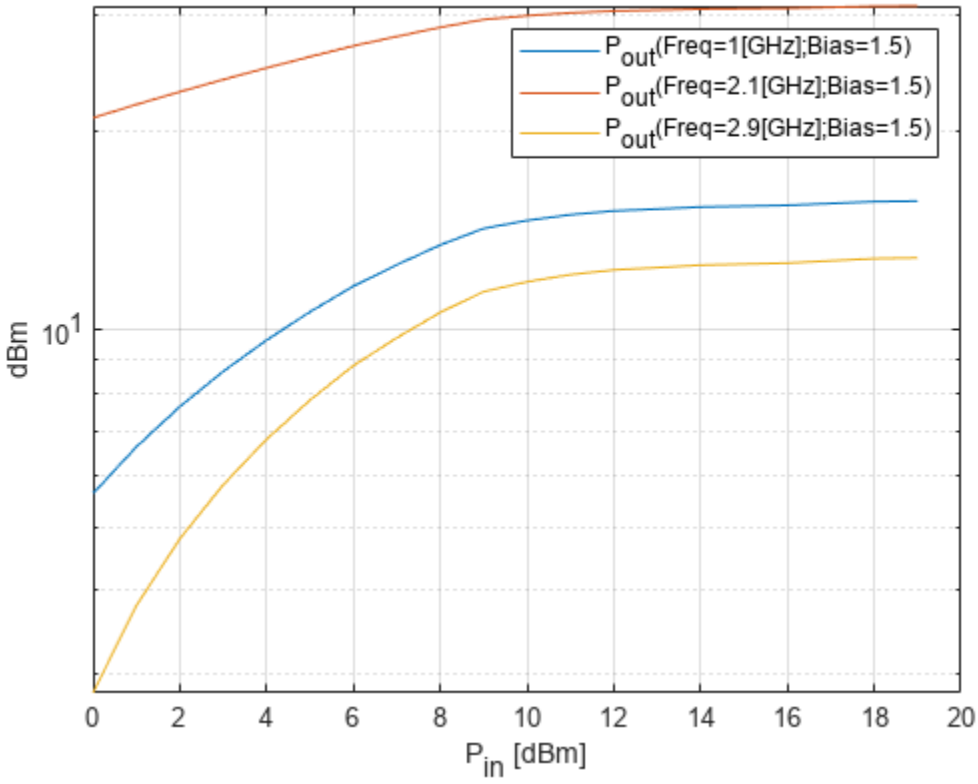
Plot Pout vs. Pin of RFCKT Amplifier

Create an RCKT amplifier object from the specified P2D file type.

```
h = read(rfckt.amplifier,'default.p2d');
```

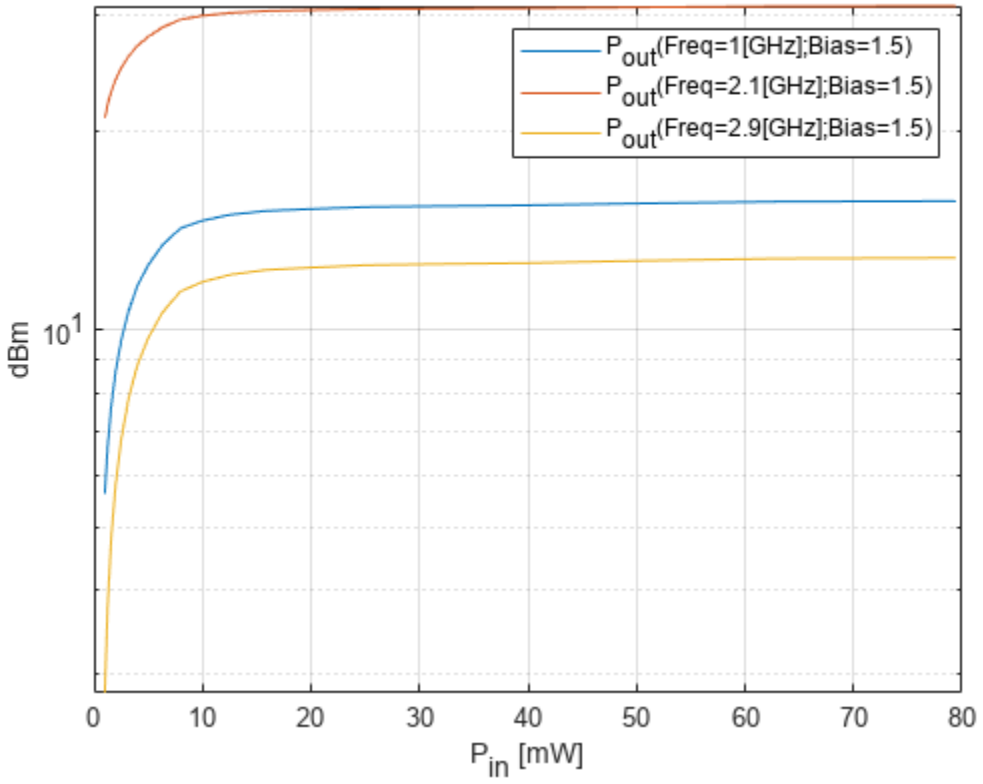
Plot the output power of the amplifier.

```
semilogy(h,'Pout')
```



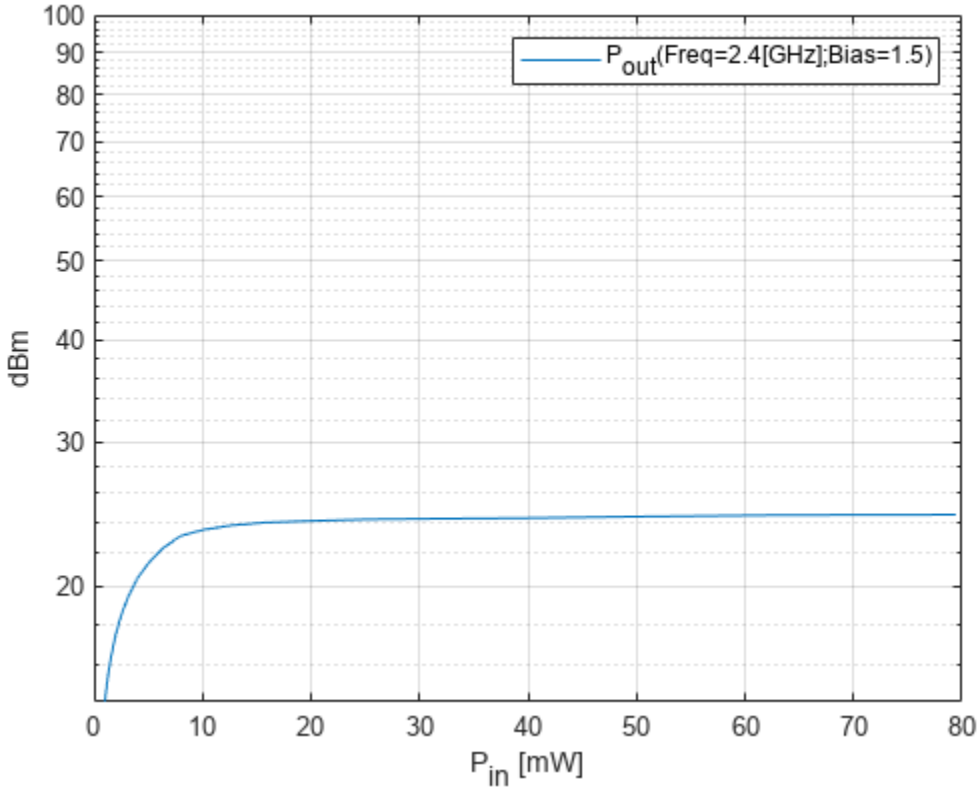
Plot the output power of the amplifier and set the format for `Pin` in milliwatts.

```
semilogy(h,'Pout','Pin','mW')
```



Plot the output power of the amplifier at `2.4` GHz.

```
semilogy(h,'Pout','Pin','mW','bias',1.5,'Freq',2.4e9)
```



Input Arguments

h — RFCKT or RF data object

`rfckt` or `rfdata` object

RFCKT or RF data object, specified as a `rfckt` or `rfdata` object.

For complete list of RFCKT and RF data objects, see “RF Circuit Objects” and “RF Data Objects”.

dataFormat — Format of data

character vector | string scalar | `'Magnitude (decibels)'` | `'Magnitude (linear)'` | `'Angle (degrees)'` | `'dBm'`

Format of the data to be plotted, specified as character vector or string scalar. Type `listformat(h,circuitPara)` command to see the available formats for a specified parameter.

Example: `lineseries = semilogy(h,'Pout','dBm')`

circuitPara — Valid RFCKT or RF data object parameter

character vector | string scalar

Valid RFCKT or data object parameter, specified as a character vector or string scalar.

Use `listparam(h)` for a list of valid parameters for the circuit or data object `h`.

xAxisPara — X-axis variable to plot with circuit parameters
`Pin` (default) | `Freq` | `AM` | character vector | string scalar

X-axis variable to plot with the circuit parameters, `circuitPara`, specified as a character vector or string scalar.

This table shows the commonly used `circuitPara` and their corresponding `xAxisPara` values. The function uses the default values listed in the table if you do not specify `xAxisPara` .

circuitPara Value	xAxisPara Value
`Pout`, `Phase`, `LS11`, `LS12`, `LS21`, `LS22`	`Pin` (default), `Freq`
`S11`, `S12`, `S21`, `S22`, `NF`, `IIP3`, `OIP3`, `GroupDelay`, `VSWRIn`, `VSWROut`, `GammaIn`, `GammaOut`, `FMIN`, `GammaOPT`, `RN`, `TF1`, `TF2`, `Gt`, `Ga`, `Gp`, `Gmag`, `Gmsg`, `GammaMS`, `GammaML`, `K`, `Delta`, `Mu`, `MuPrime`	`Freq`
`AM/AM`, `AM/PM`	`AM`

xAxisFmt — xAxisPara format
`dBm` (default) | character vector | string scalar

`xAxisPara` format, specified as a character vector or string scalar. You do not need to specify `xAxisFmt` when `xAxisPara` is an operating condition.

This table shows the commonly used `xAxisPara` and their corresponding `xAxisFmt`. The function uses the default values listed in the table if you do not specify `xAxisFmt` .

xAxisPara Value	xAxisFmt Value
`Pin`	`dBm` (default), `mW`, `W`, `dBW`
`Freq`	`THz`, `GHz`, `MHz`, `KHz`, `Hz` By default, `xAxisFmt` is chosen to provide the best scaling for the given `xAxisPara` values.
`AM`	`Magnitude (decibels)` (default), `Magnitude (linear)`

Example: `semilogy(h,'Pout','Pin','mW')` plots data using a logarithmic scale for the *y*-axis for circuit object, `h`, with `xAxisPara` set to `'Pin'` and `xAxisFmt` set to `'mW'`.

opCon — Operating conditions
string scalar | character vector

Operating conditions derived from a P2D or S2D file, specified as a string scalar or a character vector.

For some circuit parameters, you can specify a set of frequency or input power values at which the function plots the specified parameter.

For example:

- When plotting large-signal S-parameters as a function of input power, you can specify frequency points of interest using `opCon` and `opVal` arguments.

- When plotting large-signal S-parameters as a function of frequency, you can specify input power levels of interest using `opCon` and `opVal` arguments.
- When plotting parameters as a function of an operating condition, you can specify both frequency and input power values using `opCon` and `opVal` arguments.

Enter the `getop(h)` command at the command line to get the operating conditions for the RF circuit object `h`.

`opVal` — Value of operating conditions
scalar

Value of the operating conditions specified using the `opCon` argument, specified as a scalar.

Example: `semilogy(h,'Pout','Pin','mW','bias',1.5)` plots the data using a logarithmic scale for the *y*-axis for circuit object, `h`, with `opCon` set to `'bias'` and `value` set to `1.5`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `semilogy(h,'Pout','Pin','mW','bias',1.5,'Freq',2.4)`

`Freq` — Frequency value
positive scalar

Frequency value used to plot the data using a logarithmic scale for the *y*-axis, specified as the comma-separated pair consisting of `'Freq'` and a positive scalar in Hz.

`Pin` — Input power level
scalar

Input power level used to plot the data using a logarithmic scale for the *y*-axis, specified as the comma-separated pair consisting of `'Pin'` and a scalar in dBm.

Output Arguments

`lineseries` — Lineseries object
column vector of object handles

Lineseries object, returned as a column vector of object handles.

Tips

- Use the Property Editor (`propertyeditor`) or the MATLAB `set` function to change Chart Line.

Version History

Introduced in R2007a

See Also

smithplot | analyze | extract | listformat | listparam | loglog | plot | plotyy | polar | smith | semilogx | read

polar

Plot specified object parameters on polar coordinates

Syntax

```
p = polar(budgetobj,i,j)
lineseries = polar(cktobj,'parameter1',...,'parametern')
lineseries = polar( ___ ,x-axis parameter,x-axis format,
'condition1',value1,...,'conditionm',valuem,'freq',freq,'pin',pin)
```

Description

`p = polar(budgetobj,i,j)` plots the (*i*,*j*)th s-parameter on polar plot for an `rfbudget` object. `p` is a polar plot function object. For more information about properties of `p`, see Polar Properties .

`lineseries = polar(cktobj,'parameter1',...,'parametern')` plots the parameters `parameter1`, `...`, `parametern` on polar plot for a circuit object `cktobj`.

The `polar` function returns a column vector of handles to `lineseries` objects, one handle per element.

`lineseries = polar(___ ,x-axis parameter,x-axis format,
'condition1',value1,...,'conditionm',valuem,'freq',freq,'pin',pin)` plots the specified parameters at the specified operating conditions on polar plot for a circuit object, `cktobj`. Use this option with the input arguments in the previous syntax.

Note

- For all circuit objects except those that contain data from a data file, you must use the `analyze` method to perform a frequency domain analysis before calling `polar`.
- Use the function `polarpattern`, or the MATLAB function `polarplot` to plot parameters that are not part of a `rfckt` or `rfbudget` object, but are specified as vector data.

Examples

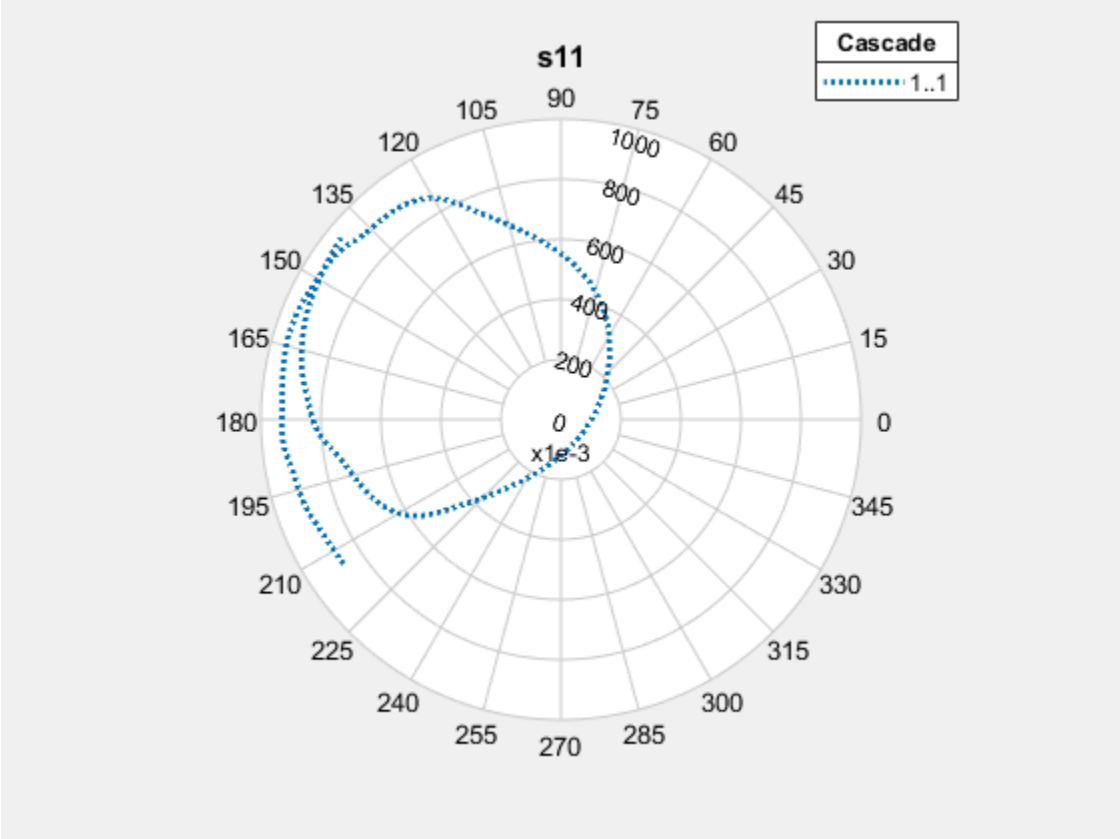
Plot S-Parameters of RF Budget Object on Polar Plot

Create an RF budget object from `default.s2p`.

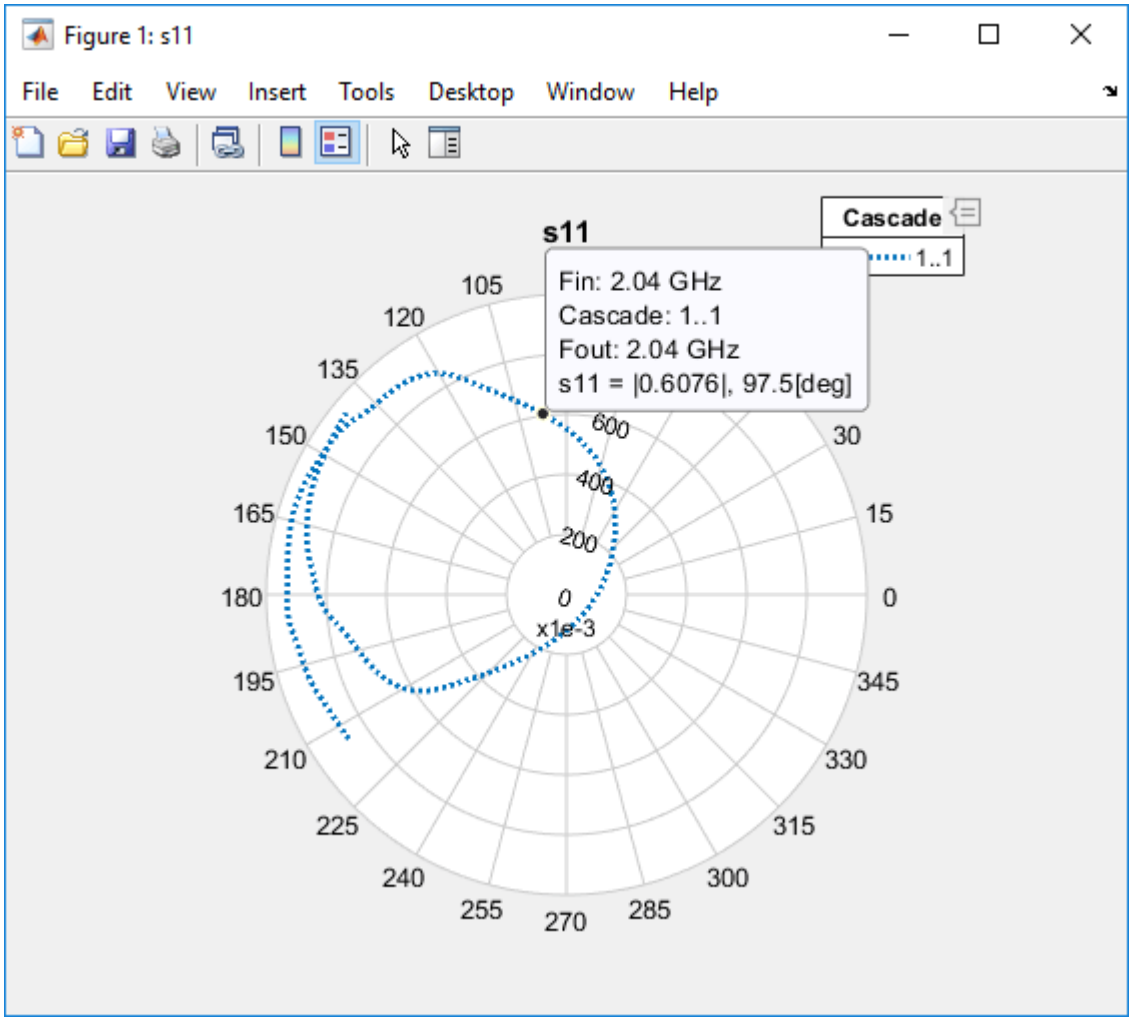
```
Sa = nport('default.s2p');
rfobj = rfbudget(Sa,Sa.NetworkData.Frequencies,-30,10);
```

Plot S11 on polar plot.

```
p = polar(rfobj,1,1);
p.LineStyle = ':';
```



In the newly opened figure window, click **View** > **Figure Toolbar**, then hover over the dataset to see the parameters specific to a particular point. You can right click to interact with the plot.



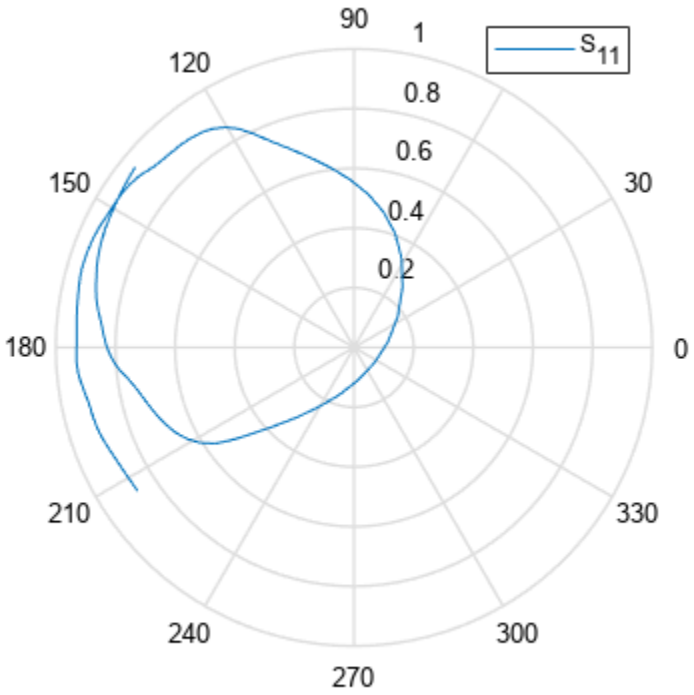
Plot Circuit Parameters of Network Object on Polar Plot

Create an amplifier object from |default.s2p|.

```
amp = read(rfckt.amplifier,'default.s2p');
```

Plot S11 on polar plot.

```
lineseries = polar(amp,'S11')
```




```
lineseries =
  Line (S_{11}) with properties:

            Color: [0 0.4470 0.7410]
        LineStyle: '-'
        LineWidth: 0.5000
           Marker: 'none'
       MarkerSize: 6
  MarkerFaceColor: 'none'
            XData: [-0.7247 -0.7318 -0.7388 -0.7457 -0.7525 -0.7593 ... ]
            YData: [-0.4813 -0.4715 -0.4616 -0.4517 -0.4419 -0.4320 ... ]

  Show all properties
```

Input Arguments

`budgetobj` — RF budget object
`rfbudget` object

RF budget object, specified as a `rfbudget` object.

`cktobj` — Circuit object
object handle

Circuit object (`rfckt`) object, specified as an object handle.

To get a list of valid parameters for `cktobj`, type `listparam(cktobj)`.

`x-axis parameter` — Independent variable along *x*-axis
character vector | string

The independent variable along the *x*-axis to plot the specified parameters along the *y*-axis, specified as a character vector or string. Several `x-axis parameter` values are available for all objects. When you import `rfckt.amplifier`, `rfckt.mixer`, or `rfdata.data` object specifications from a `.p2d` or `.s2d` file, you can also specify any operating conditions from the file that have numeric values, such as bias.

The following table shows the most commonly available parameters and the corresponding `x-axis parameter` values. The default settings listed in the table are used if `x-axis parameter` is not specified.

Parameter Name	**`x-axis parameter` Values**
`Pout`, `Phase`, `LS11`, `LS12`, `LS21`, `LS22`	`Pin` (default), `Freq`
`S11`, `S12`, `S21`, `S22`, `NF`, `OIP3`, `VSWRIn`, `VSWROut`, `GammaIn`, `GammaOut`, `FMIN`, `GammaOPT`, `RN`	`Freq`
`AM/AM`, `AM/PM`	`AM`

`x-axis format` — Format used for specific *x*-axis parameter
character vector | string

The format used for the specific `x-axis parameter`, specified as a character vector or string. No `x-axis format` specification is needed when `x-axis parameter` is an operating condition.

The following table shows the `x-axis format` values that are available for the `x-axis parameter` values listed in the preceding table, along with the default settings that are used if `x-axis format` is not specified.

`x-axis parameter` Values	**`x-axis format` Values**
`Pin`	`dBm` (default), `mW`, `W`, `dBW`
`Freq`	`THz`, `GHz`, `MHz`, `KHz`, `Hz` By default, `x-axis format` is chosen to provide the best scaling for the given `x-axis parameter` values.
`AM`	`Magnitude (decibels)` (default), `Magnitude (linear)`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `lineseries = polar(h,'Freq',2.1e9)`

`'condition1',value1,...,'conditionm',valuem` — Optional condition-value pairs
scalar

Optional condition-value pairs at which to plot the specified parameters, specified as a series of `'condition',value` pairs separated by commas. These pairs are usually operating conditions from a `.p2d` or `.s2d` file. For some parameters, you can specify a set of frequency or input power values at which to plot the specified parameter.

For example:

- When plotting large-signal S-parameters as a function of input power, you can specify frequency points of interest using condition-value pairs.
- When plotting large-signal S-parameters as a function of frequency, you can specify input power levels of interest using condition-value pairs.
- When plotting parameters as a function of an operating condition, you can specify both frequency and input power values using condition-value pairs.

`freq` — Optional frequency value
scalar

The optional frequency value, in Hz, at which to plot the specified parameters. `Freq` is specified as the comma-separated pair of `'Freq'`, and a scalar value.

`pin` — Optional input power level
scalar

The optional input power value, in dBm, at which to plot the specified parameters. `pin` is specified as the comma-separated pair of `'pin'`, and a scalar value.

Output Arguments

`p` — Polar plot function object
object handle

Polar plot function object, returned as object handle

For more information about properties of `p`, see Polar Properties.

`lineseries` — `lineseries` object
column vector of object handles.

`lineseries` object, returned as a column vector of object handles.

Tips

- If you do not specify any operating conditions as arguments to the `polar` method, then the method plots the parameter values based on the currently selected operating condition.
- If you specify one or more operating conditions, the `polar` method plots the parameter values based on those operating conditions.
- When you use an operating condition for the `x-axis parameter` input argument, the method plots the parameters for all operating condition values.
- Use the Property Editor (`propertyeditor`) or the MATLAB `set` function to change the Chart Line.
- The reference pages for MATLAB functions such as `figure`, `axes`, and `text` list available properties and provide links to detailed descriptions.

Version History

Introduced before R2006a

See Also

Polar Properties | `analyze` | `calculate` | `extract` | `plotyy` | `listformat` | `listparam` | `loglog` | `plot` | `polarpattern` | `smithplot`

Topics

“Visualizing RF Budget Analysis over Bandwidth”

set

Set `rffilter` object property values

Syntax

```
set(rfobj,'propertyname',propertyvalue)
set(rfobj,pn,pv)
set(rfobj,a)
a1 = set(rfobj,propertyname)
a2 = set(rfobj)
```

Description

`set(rfobj,'propertyname',propertyvalue)` sets the `propertyname` and the `propertyvalue` for the RF filter object with object handle `rfobj`.

`set(rfobj,pn,pv)` sets the properties given in the cell array `pn` to the values given in the cell array `pv` for the RF filter object with object handle `rfobj`.

`set(rfobj,a)` sets each rf filter properties names with values contained in the structure `a`.

`a1 = set(rfobj,propertyname)` returns and displays the possible values for the `propertyname`.

`a2 = set(rfobj)` returns and displays the names of the user-settable properties and their possible values for the RF filter object with object handle `rfobj`.

Input Arguments

`rfobj` — RF Filter object
object handle

RF filter object, specified as a `rffilter` object handle.

`propertyname` — User-settable property name
character vector | string

User-settable filter property name, specified as a character vector or string.

Example: `set(rfobj,'FilterType','Butterworth')`. Sets the `'FilterType'` property of RF filter, `rfobj`, to `'Butterworth'`.

Data Types: `char` | `string`

`propertyvalue` — Property value
scalar

Property value, specified as a scalar.

Example: `set(rfobj,'FilterType','Butterworth')`. Sets the `'FilterType'` property of RF filter, `rfobj`, to `'Butterworth'`.

Data Types: `double` | `char` | `string`

a — List of object property names and property values
structure

List of object property names and property values, specified as a structure.

Data Types: `struct`

pn — User-settable property names
1-by-*N* cell array

User-settable property names, specified as a 1-by-*N* cell array.

Example: `set(rfobj,{'FilterType','Implementation'},{'Butterworth','LC Pi'})`. Sets the `'FilterType'` and `'Implementation'` properties of RF filter, `rfobj`, to `'Butterworth'` and `'LC Pi'` respectively.

Data Types: `char` | `string`

pv — Property values
1-by-*N* cell array

Property values, specified as a 1-by-*N* cell array.

Example: `set(rfobj,{'FilterType','Implementation'},{'Butterworth','LC Pi'})`. Sets the `'FilterType'` and `'Implementation'` properties of RF filter, `rfobj`, to `'Butterworth'` and `'LC Pi'` respectively.

Data Types: `double` | `char` | `string`

Output Arguments

a1 — Possible values of specified property name
cell array of possible value strings | empty cell array

Possible values of the specified property name, returned as a cell array of possible value strings or an empty cell array if the property does not have a finite set of possible string values.

a2 — User-settable property names and their values
structure of property names and arrays of their possible values

User-settable property names and their values, returned as a structure of property names and arrays of their possible values. Values can be cell array of possible value strings or an empty cell array if the property does not have a finite set of possible string values.

Version History

Introduced in R2018b

See Also

`rffilter`

table

Display specified RF object parameters in Variable Editor

Syntax

```
table(h,param1,format1,..., paramn,formatn)
table(h,'budget',param1,format1,...,paramn,formatn)
```

Description

`table(h,param1,format1,..., paramn,formatn)` displays the specified parameters *`param1`* through *`paramn`*, with units *`format1`* through *`formatn`*, in the Variable Editor. The input `h` is a object handle to an `rfckt` object.

This function creates a structure in the MATLAB workspace and constructs the name of the structure from the names of the object and parameters you provide. Specify the parameters and formats in pairs. If you do not specify a format, the method uses the default format for that parameter.

`table(h,'budget',param1,format1,...,paramn,formatn)` specified budget parameters of an `rfckt.cascade` object `h`.

Examples

Use Table to Display Link Budget of RF Cascade

Construct a cascaded RF Circuit object.

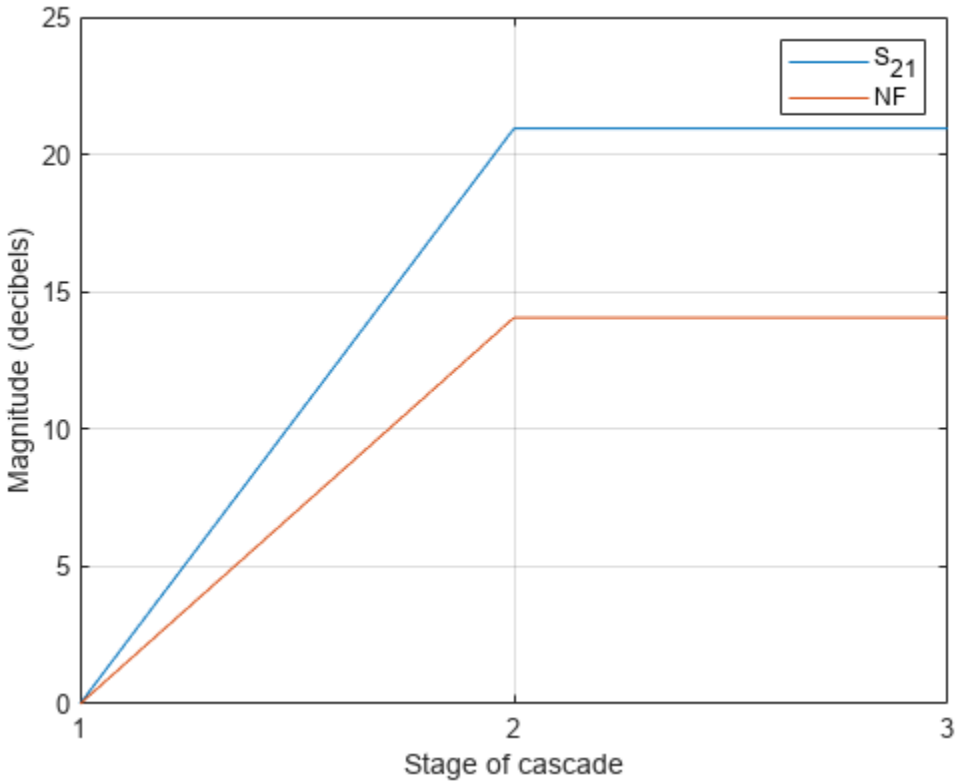
```
Cascaded_Ckt = rfckt.cascade('Ckts', ...
               {rfckt.txline('LineLength', .001), ...
               rfckt.amplifier, rfckt.txline( ...
               'LineLength', 0.025, 'PV', 2.0e8)});
```

Analyze the RF cascade in frequency domain at 2.1 GHz.

```
freq = 2.1e9;
analyze(Cascaded_Ckt,freq);
```

Plot the budget S21 and noise figure.

```
plot(Cascaded_Ckt,'budget','S21','NF');
```



Type this command at the MATLAB® command line to display the budget S21 and noise figure in a table.

```
table(Cascaded_Ckt,'budget','S21','NF')
```

Variables - Cascaded_Ckt_budget_S21_NF

Cascaded_Ckt_budget_S21_NF

2x7 cell

	1	2	3	4	5	6	7	8
1	'Freq [GHz]'	'Stage 1: S2...	'Stage 2: S2...	'Stage 3: S2...	'Stage 1: NF...	'Stage 2: NF...	'Stage 3: NF...	
2	2.1000	0	20.9455	20.9455	9.6433e-16	14.0612	14.0612	
3								
4								
5								
6								
7								
8								
9								
10								
11								

Input Arguments

h — RF network object
object handle

RF network object, specified as object handle.

Data Types: `char` | `string`

'budget' — Flag to get rfckt.cascade object
char | string

Flag to get the `rfckt.cascade` object, specified as a character vector or string.

param1,format1,..., paramn,formatn — RF network parameter
char | string

RF network parameter object, specified as a character vector or string.

To list valid parameters and parameter formats for `h`, use the `listparam` and `listformat` methods.

Version History

Introduced in R2010b

See Also

`analyze` | `calculate` | `circle` | `extract` | `listformat` | `listparam` | `loglog` | `plot` | `plotyy` | `getop` | `polar` | `semilogx` | `semilogy` | `smith` | `write` | `getz0` | `read` | `restore` | `getop` | `groupdelay`

listformat

List valid formats for specified circuit object parameter

Syntax

```
listformat(h,'parameter')
```

Description

`listformat(h,'parameter')` lists the allowable formats for the specified network parameter. The first listed format is the default format for the specified parameter.

In these lists, `'Abs'` and `'Mag'` are the same as `'Magnitude (linear)'`, and `'Angle'` is the same as `'Angle (degrees)'`.

When you plot phase information as a function of frequency, RF Toolbox software unwraps the phase data using the MATLAB `unwrap` function. The resulting plot is only meaningful if the phase data varies smoothly as a function of frequency, as described in the `unwrap` reference page. If your data does not meet this requirement, you must obtain data on a finer frequency grid.

Use the `listparam` method to get the valid parameters of a circuit object.

Note Before calling `listformat` function, you must use the `analyze` function to perform a frequency domain analysis for the circuit object.

Examples

List Format of Network Parameter

List the available formats of analysis of a transmission line.

```
trl = rfckt.txline;
f = [1e9:1.0e7:3e9];
analyze(trl,f);
listformat(trl,'S11')
```

```
ans = 11x1 cell
    {'dB'                   }
    {'Magnitude (decibels)'}
    {'Abs'                  }
    {'Mag'                  }
    {'Magnitude (linear)'   }
    {'Angle'                }
    {'Angle (degrees)'      }
    {'Angle (radians)'      }
    {'Real'                 }
    {'Imag'                 }
    {'Imaginary'            }
```

Input Arguments

h — RF circuit object
object handle

RF circuit object, specified as an object handle.

'parameter' — RF network parameter
character vector | string

RF network parameter object, specified as a character vector or string.

Version History

Introduced before R2006a

See Also

`analyze` | `calculate` | `circle` | `extract` | `listparam` | `loglog` | `plot` | `plotyy` | `getop` | `polar` | `semilogx` | `semilogy` | `smith` | `write` | `getz0` | `read` | `restore` | `getop` | `groupdelay`

write

Write RF data from circuit or data object to file

Syntax

```
status = write(data,filename,dataformat,funit,printformat,freqformat)
```

Description

`status = write(data,filename,dataformat,funit,printformat,freqformat)` writes information from `data` to the specified file. The `write` function returns `True` if the operation is successful and returns `False` otherwise.

Note The method only writes property values from `data` that the specified output file supports. For example, Touchstone files, which have the `.snp`, `.ynp`, `.znp`, or `.hnp` extension, do not support noise figure or output third-order intercept point. Consequently, the `write` method does not write these property values to these such files.

Examples

Write Data to Touchstone® File

Read the data stored in the file `default.s2p`

```
orig_data=read(rfdata.data,'default.s2p')
```

```
orig_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: 'Linear'
            Name: 'Data object'
```

Analyze the data stored in the specified file for a set of frequency values.

```
freq = [1:.1:2]*1e9;
analyze(orig_data,freq);
```

Use the `write` function to store the results in a file called `test.s2p`.

```
write(orig_data,'test.s2p')
```

```
ans = logical
   1
```

Input Arguments

`data` — Object that contains sufficient information to write specified files

RF circuit object | RF data object

Object that contains sufficient information to write the specified files, specified as an RF circuit or data object.

`filename` — File name

`.snp` (default) | character vector | string scalar

File name of a `.snp, .ynp, .znp, .hnp, or .amp` file extensions, where n is the number of ports, specified as a character vector, or string scalar.

`dataformat` — Data format to be written

`'DB'` | `'MA'` | `'RI'` | character vector | string scalar

Data format to be written, specified as a character vector or string scalar. It must be one of the case-insensitive values specified in the following table.

Format	Description
`'DB'`	Data is given in (dB-magnitude, angle) pairs with angle in degrees.
`'MA'`	Data is given in (magnitude, angle) pairs with angle in degrees.
`'RI'`	Data is given in (real, imaginary) pairs (default).

`funit` — Frequency units of data

`GHz` | `MHz` | `KHz` | `Hz`

Frequency units of the data, specified in `'GHz'`, `'MHz'`, `'KHz'`, or `'Hz'`.

Note If you do not specify `funit`, its value is taken from the object `data`. All values are case-insensitive

`printformat` — Represents precision of network and noise parameters

`%22.10f` (default) | formatting operators

Represents precision of the network and noise parameters, specified using formatting operators . For example, if you set the `printformat` to `%22.10f`, this means that the method writes the data using fixed-point notation with a precision of 10 digits. The minimum positive value that the `write` function can express by default is `1e-10`. For greater precision, specify a different `printformat`. See the `formatspec` argument in the `fprintf` function for more information.

`freqformat` — Represents precision of frequency

`%-22.10f` (default) | formatting operators

Represents precision of the frequency, specified using formatting operators. See `fprintf` for more information.

Version History

Introduced before R2006a

References

[1] EIA/IBIS Open Forum, “ Touchstone File Format Specification,” Rev. 1.1, 2002 (https://ibis.org/connector/touchstone_spec11.pdf)

See Also

`analyze` | `calculate` | `circle` | `extract` | `listformat` | `listparam` | `loglog` | `plot` | `plotyy` | `getop` | `polar` | `semilogx` | `semilogy` | `smith` | `write` | `getz0` | `read` | `restore` | `getop` | `groupdelay`

read

Read RF data from file to new or existing circuit or data object

Syntax

```
h = read(rfcktobj,filename)
```

Description

`h = read(rfcktobj,filename)` creates an RF circuit object `h`, reads the RF data from the specified file, and stores it in `h`

Example: `h = read(rfckt.passive,filename)` creates an `rfckt.passive` object `h`, reads the RF data from the specified file, and stores it in `h`.

Examples

Import Data

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

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

```
ckt_obj =
   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'
```

Input Arguments

`rfcktobj` — Object that contains sufficient information to read files

`rfckt.datafile` | `rfckt.passive` | `rfckt.amplifier` | `rfckt.mixer` | `rfdata.data` | RF circuit object

Object that contains sufficient information to read the files, specified as an RF circuit object. You can read data from an `.snp`, `.ynp`, `.znp`, `.hnp`, `.gnp`, or `.amp` file extension, where `n` is the number of ports.

Example: `h = read(rfckt.amplifier,filename)` creates an `rfckt.amplifier` object `h`, reads the RF data from the specified file, and stores it in `h`.

Note If `h` is an `rfckt.amplifier`, `rfckt.mixer`, or `rfdata.data` object, you can also read data from `.p2d` and `.s2d` files.

`filename` — File name

`.snp` (default) | `.ynp` | `.znp` | `.hnp` | `.gnp` | `.amp` | character vector

File name of an `.snp`, `.ynp`, `.znp`, `.hnp`, `.gnp`, or `.amp` file extension, where `n` is the number of ports, specified as a character vector.

For an example of how to use RF Toolbox software to read data from a `.s2d` file, see "Visualize Mixer Spurs".

Example: `h = read(rfckt.mixer,filename)` creates an `rfckt.mixer` object `h`, reads the RF data from the specified file, and stores it in `h`.

Note

- For all files, the file name must include the file extension.
- If `h` is an `rfckt.amplifier`, `rfckt.mixer`, or `rfdata.data` object, you can also read data from `.p2d` and `.s2d` files.

Output Arguments

`h` — Object that contains information from the files read

RF circuit object

Object that contains information from the files read, returned as an RF circuit object.

Example: `h = read(rfckt.datafile,filename)`creates an `rfckt.datafile` object `h`, reads the RF data from the specified file, and stores it in `h`.

Data Types: `char` | `string`

Version History

Introduced before R2006a

References

[1] EIA/IBIS Open Forum, " Touchstone File Format Specification," Rev. 1.1, 2002 (https://ibis.org/connector/touchstone_spec11.pdf)

See Also

`analyze` | `calculate` | `circle` | `extract` | `listformat` | `listparam` | `loglog` | `plot` | `plotyy` | `getop` | `polar` | `semilogx` | `semilogy` | `smith` | `write` | `getz0` | `read` | `restore` | `getop` | `groupdelay`

restore

Restore data to original frequencies

Syntax

```
h = restore(h)
```

Description

`h = restore(h)` restores data in `h` to the original frequencies of network data for plotting.

Examples

Restore Data of Circuit Object

Create an amplifier object from the specified file and restore data.

```
amp = read(rfckt.amplifier,'default.s2p');
restore(amp)
```

```
ans =
   rfckt.amplifier with properties:

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

Input Arguments

h — Object that contains data to be restored

`rfckt.datafile` | `rfckt.passive` | `rfckt.amplifier` | `rfckt.mixer` | RF circuit object

Object that contains the data to be restored, specified as an RF circuit object. Here, the object `h` can be `rfckt.passiverfckt.amplifier`, `rfckt.mixer`, or `rfckt.datafile`.

Output Arguments

h — Object that holds restored data

`rfckt.datafile` | `rfckt.passive` | `rfckt.amplifier` | `rfckt.mixer` | RF circuit object

Object that holds the restored data, returned as an RF circuit object. Here, the object `h` can be `rfckt.passive` `rfckt.amplifier`, `rfckt.mixer`, or `rfckt.datafile`.

Version History

Introduced before R2006a

See Also

analyze | calculate | circle | extract | listformat | listparam | loglog | plot | plotyy | getop | polar | semilogx | semilogy | smith | write | getz0 | read | restore | getop | groupdelay

setop

Set operating conditions

Syntax

```
setop(h)
setop(h,'Condition1')
setop(h,'Condition1',value1,'Condition2',value2,...)
```

Description

`setop(h)` lists the available values for all operating conditions of the object `h`. Operating conditions only apply to objects you import from a `.p2d` or `.s2d` file. To import these types of data into an object, use the `read` method. Operating conditions are not listed with other properties of an object.

`setop(h,'Condition1')` lists the available values for the specified operating condition '*Condition1*'.

`setop(h,'Condition1',value1,'Condition2',value2,...)` changes the operating conditions of the circuit or data object, `h`, to those specified by the condition/value pairs. Conditions you do not specify retain their original values. The method ignores any conditions that are not applicable to the specified object. Ignoring these conditions lets you apply the same set of operating conditions to an entire network where different conditions exist for different components.

Note When you set the operating conditions for a network that contains several objects, the software does not issue an error or warning if the specified conditions cannot be applied to all objects. For some networks, this lack of error or warning lets you call the `setop` method once to apply the same set of operating conditions to any objects where operating conditions are applicable. However, you may want to specify a network that contains one or more of the following:

- Several objects with different sets of operating conditions.
- Several objects with the same set of operating conditions that are configured differently.

To specify operating conditions one of these types of networks, use a separate call to the `setop` method for each object.

Examples

List Operating Conditions of Network Object

List the operating conditions of an RF amplifier object.

```
ckt1 = read(rfckt.amplifier,'default.p2d');
setop(ckt1)
```

```
Operating conditions set 1:
    {'Bias'}    {'1.5'}
```

Analyze RF Amplifier Object Under Specific Operating Conditions

Analyze an RF amplifier under specific operating conditions set using the function `setop`.

```
ckt1 = read(rfckt.amplifier, 'default.p2d');
freq = ckt1.AnalyzedResult.Freq;
setop(ckt1, 'Bias', '1.5');
analyze(ckt1, freq)
```

```
ans =
   rfckt.amplifier with properties:

         NoiseData: [1x1 rfdata.noise]
      NonlinearData: [1x1 rfdata.p2d]
           IntpType: 'Linear'
        NetworkData: [1x1 rfdata.network]
              nPort: 2
     AnalyzedResult: [1x1 rfdata.data]
               Name: 'Amplifier'
```

Input Arguments

h — Object used to set operating conditions

RF Circuit object

Object used to set the operating conditions, specified as an RF circuit object.

Version History

Introduced in R2007a

See Also

`analyze` | `calculate` | `circle` | `extract` | `listformat` | `listparam` | `loglog` | `plot` | `plotyy` | `getop` | `polar` | `semilogx` | `semilogy` | `smith` | `write` | `getz0` | `read` | `restore` | `getop` | `groupdelay`

listparam

List valid parameters for specified circuit object

Syntax

```
listparam(h)
```

Description

`listparam(h)` lists the valid parameters for the specified circuit object `h`.

Note Before calling `listparam` function, you must use the `analyze` function to perform a frequency domain analysis for the circuit object.

Examples

List Parameters of Network Object

List the available parameters of analysis of a transmission line.

```
trl = rfckt.txline;
f = [1e9:1.0e7:3e9];
analyze(trl,f);
listparam(trl)
```

```
ans = 28x1 cell
    {'S11'       }
    {'S12'       }
    {'S21'       }
    {'S22'       }
    {'GroupDelay'}
    {'GammaIn'   }
    {'GammaOut'  }
    {'VSWRIn'    }
    {'VSWROut'   }
    {'OIP3'      }
    {'IIP3'      }
    {'NF'        }
    {'NFactor'   }
    {'NTemp'     }
    {'TF1'       }
    {'TF2'       }
    {'TF3'       }
    {'Gt'        }
    {'Ga'        }
    {'Gp'        }
    {'Gmag'      }
    {'Gmsg'      }
    {'GammaMS'   }
    {'GammaML'   }
```

```
{'K'         }
{'Delta'     }
{'Mu'        }
{'MuPrime'   }
```

Input Arguments

h — RF circuit or data object

object handle

RF circuit or data object, specified as an object handle.

Several parameters are available for all objects. When you import `rfckt.amplifier`, `rfckt.mixer`, or `rfdata.data` object specifications from a `.p2d` or `.s2d` file, the list of valid parameters also includes any operating conditions from the file that have numeric values, such as bias.

This table describes the most commonly available parameters.

Parameter	Description
`S11`, `S12`, `S21`, `S22` `LS11`, `LS12`, `LS21`, `LS22` (Amplifier and mixer objects with multiple operating conditions only)	S-parameters
`GroupDelay`	Group delay
`GammaIn`, `GammaOut`	Input and output reflection coefficients
`VSWRIn`, `VSWROut`	Input and output voltage standing-wave ratio
`IIP3`, `OIP3` (Amplifier and mixer objects only)	Third-order intercept point
`NF`	Noise figure
`TF1`	Ratio of the load voltage to the output voltage of the source when the input port is conjugate matched
`TF2`	Ratio of load voltage to the source voltage
• `Gt` • `Ga` • `Gp` • `Gmag` • `Gmsg`	• Transducer power gain • Available power gain • Operating power gain • Maximum available power gain • Maximum stable gain
`GammaMS`, `GammaML`	Source and load reflection coefficients for simultaneous conjugate match
`K`, `Mu`, `MuPrime`	Stability factor
`Delta`	Stability condition

Version History

Introduced before R2006a

See Also

analyze | calculate | circle | extract | listformat | loglog | plot | plotyy | getop | polar | semilogx | semilogy | smith | write | getz0 | read | restore | getop | groupdelay

loglog

Plot specified circuit object parameters using log-log scale

Syntax

```
loglog(h,circuitPara)
loglog(h,circuitPara,dataFormat)
loglog( ___ ,xAxisPara,xAxisFmt)
loglog( ___ ,opCon,opVal)
loglog( ___ ,Name,Value)
lineseries = loglog( ___ )
```

Description

`loglog(h,circuitPara)` plots the circuit parameter `circuitPara` from the RFCKT or RF data object `h` using a log-log scale. You can specify multiple circuit parameters in this syntax.

Note For all circuit objects except those that contain data from a data file, you must perform a frequency domain analysis with the `analyze` method before calling `loglog`.

`loglog(h,circuitPara,dataFormat)` plots the data of the RFCKT or RF data object using a log-log scale with the specified data format.

`loglog( ___ ,xAxisPara,xAxisFmt)` plots the circuit parameters `circuitPara` using a log-log scale along with the variables `xAxisPara` and their corresponding format `xAxisFmt`. Specify `xAxisPara` and `xAxisFmt` arguments after any of the input argument combinations in the previous syntaxes.

`loglog( ___ ,opCon,opVal)` plots the circuit parameters using a log-log scale with operating conditions `opCon` and operating values `opVal` for the circuit object `h`.

Derive operating conditions for the RFCKT or RF data object `h` using the `getop(h)` command

`loglog( ___ ,Name,Value)` plots the data of a RFCKT or RF data object with name-value arguments.

`lineseries = loglog( ___ )` returns the line series property object `lineseries`. This output is the same as the output returned by the MATLAB `loglog` function.

Examples

Plot S11 of Two-Wire Transmission Line

Create and analyze a two-wire transmission line RFCKT object.

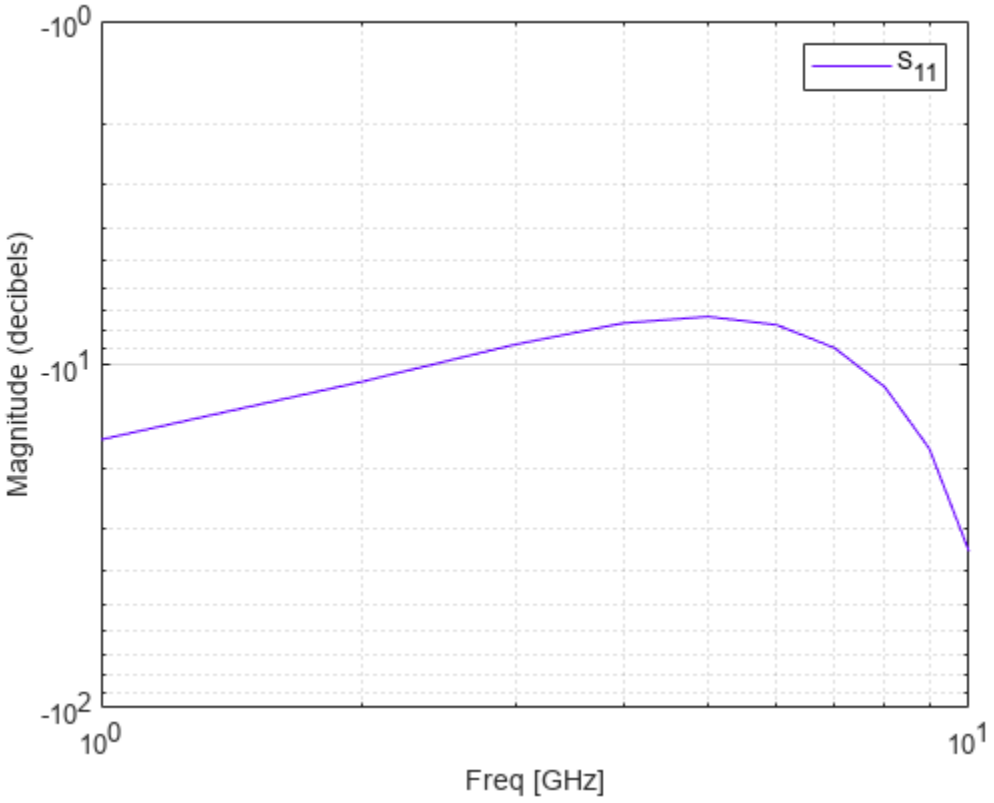
```
h = rfckt.twowire('Radius',7.5e-4);
freq = linspace(1e9,10e9,10);
analyze(h,freq);
```

Plot S11 using the log-log scale.

```
lineseries = loglog(h,'S11');
```

Change the color of the S11 data.

```
lineseries(1).Color = [0.4 0 1];
```



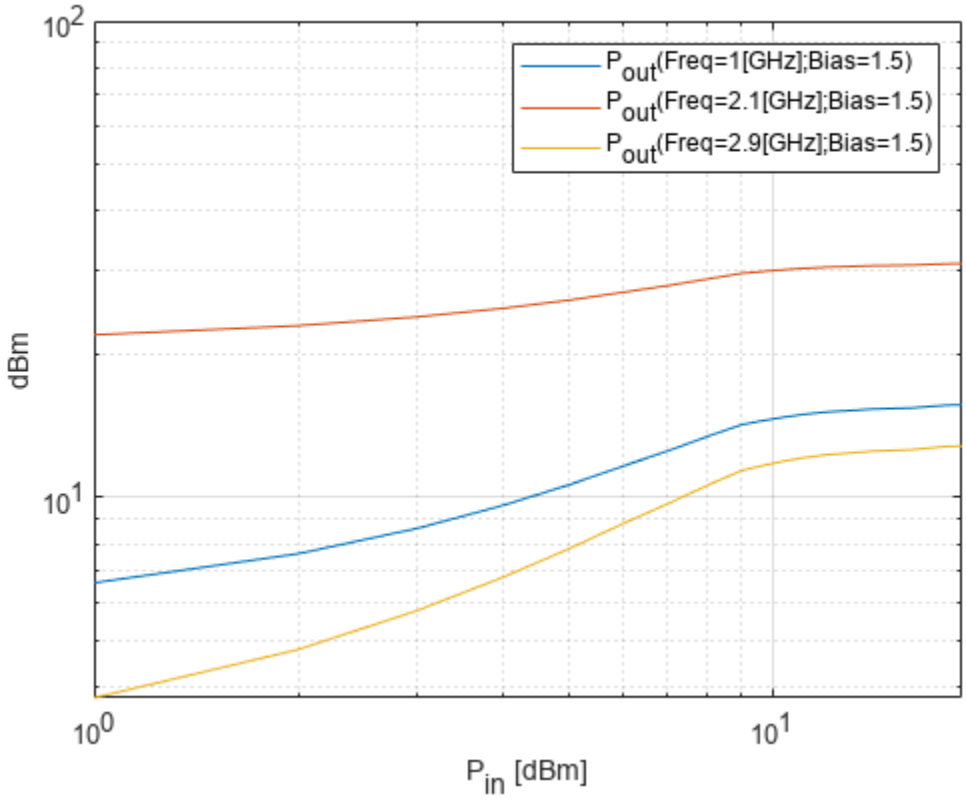
Plot Pout vs. Pin of Amplifier Using `loglog` Function

Create an amplifier object from the specified P2D file type.

```
h = read(rfckt.amplifier,'default.p2d');
```

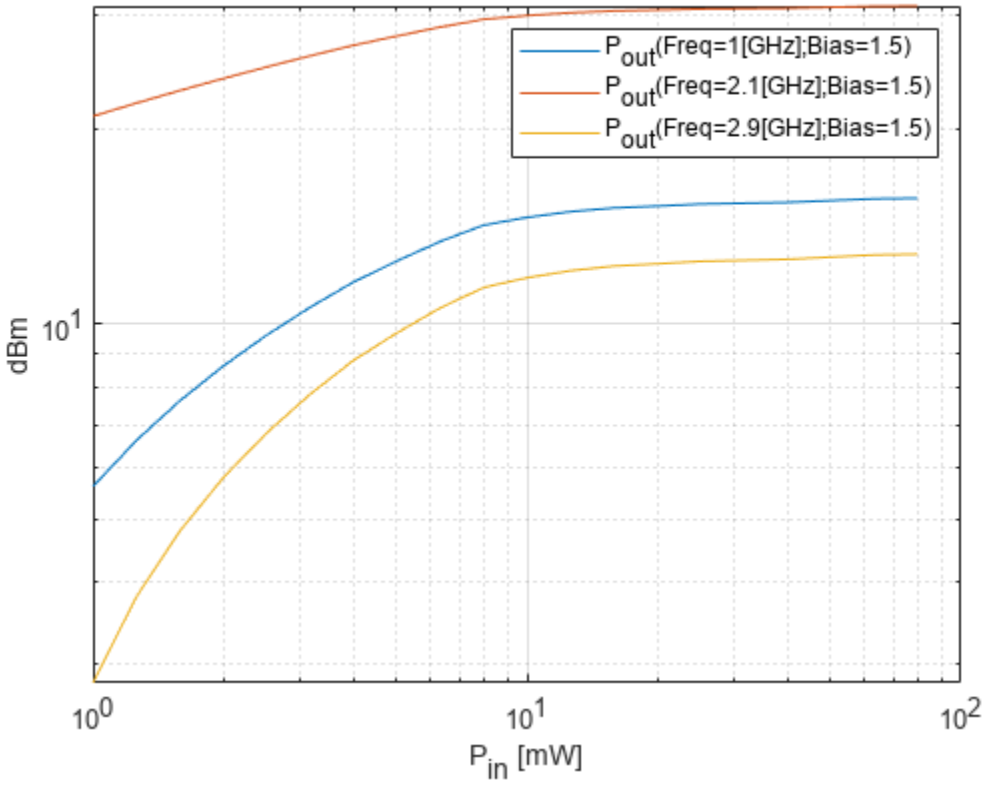
Plot the output power of the amplifier.

```
loglog(h,'Pout')
```



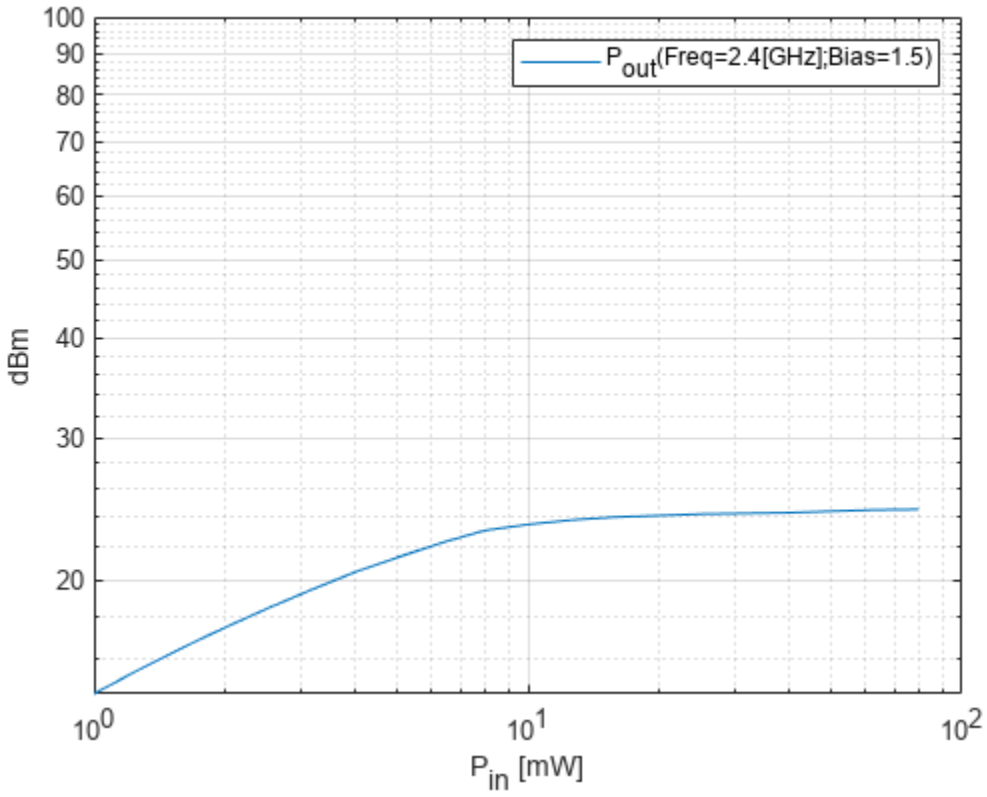
Plot the output power of the amplifier and set the format for `Pin` in milliwatts.

```
loglog(h,'Pout','Pin','mW')
```



Plot the output power of the amplifier at `2.4` GHz.

```
loglog(h,'Pout','Pin','mW','bias',1.5,'Freq',2.4e9)
```



Input Arguments

h — RFCKT or RF data object

`rfckt` or `rfdata` object

RFCKT or RF data object, specified as a `rfckt` or `rfdata` object.

For complete list of RFCKT and RF data objects, see “RF Circuit Objects” and “RF Data Objects”.

dataFormat — Format of data

character vector | string scalar | `'Magnitude (decibels)'` | `'Magnitude (linear)'` | `'Angle (degrees)'` | `'dBm'`

Format of the data to be plotted, specified as character vector or string scalar. Type `listformat(h,circuitPara)` command to see the available formats for a specified parameter.

Example: `lineseries = semilogy(h,'Pout','dBm')`

circuitPara — Valid RFCKT or RF data object parameter

character vector | string scalar

Valid RFCKT or data object parameter, specified as a character vector or string scalar.

Use `listparam(h)` for a list of valid parameters for the circuit or data object `h`.

xAxisPara — X-axis variable to plot with circuit parameters
`Pin` (default) | `Freq` | `AM` | character vector | string scalar

X-axis variable to plot with the circuit parameters, `circuitPara`, specified as a character vector or string scalar.

This table shows the commonly used `circuitPara` and their corresponding `xAxisPara` values. The function uses the default values listed in the table if you do not specify `xAxisPara` .

circuitPara Value	xAxisPara Value
`Pout`, `Phase`, `LS11`, `LS12`, `LS21`, `LS22`	`Pin` (default), `Freq`
`S11`, `S12`, `S21`, `S22`, `NF`, `IIP3`, `OIP3`, `GroupDelay`, `VSWRIn`, `VSWROut`, `GammaIn`, `GammaOut`, `FMIN`, `GammaOPT`, `RN`, `TF1`, `TF2`, `Gt`, `Ga`, `Gp`, `Gmag`, `Gmsg`, `GammaMS`, `GammaML`, `K`, `Delta`, `Mu`, `MuPrime`	`Freq`
`AM/AM`, `AM/PM`	`AM`

xAxisFmt — xAxisPara format
`dBm` (default) | character vector | string scalar

`xAxisPara` format, specified as a character vector or string scalar. You do not need to specify `xAxisFmt` when `xAxisPara` is an operating condition.

This table shows the commonly used `xAxisPara` and their corresponding `xAxisFmt`. The function uses the default values listed in the table if you do not specify `xAxisFmt` .

xAxisPara Value	xAxisFmt Value
`Pin`	`dBm` (default), `mW`, `W`, `dBW`
`Freq`	`THz`, `GHz`, `MHz`, `KHz`, `Hz` By default, `xAxisFmt` is chosen to provide the best scaling for the given `xAxisPara` values.
`AM`	`Magnitude (decibels)` (default), `Magnitude (linear)`

Example: `loglog(h,'Pout','Pin','mW')` plots data using a log-log scale for circuit object, `h`, with `xAxisPara` set to `'Pin'` and `xAxisFmt` set to `'mW'`.

opCon — Operating conditions
string scalar | character vector

Operating conditions derived from a P2D or S2D file, specified as a string scalar or a character vector.

For some circuit parameters, you can specify a set of frequency or input power values at which the function plots the specified parameter.

For example:

- When plotting large-signal S-parameters as a function of input power, you can specify frequency points of interest using `opCon` and `opVal` arguments.

- When plotting large-signal S-parameters as a function of frequency, you can specify input power levels of interest using `opCon` and `opVal` arguments.
- When plotting parameters as a function of an operating condition, you can specify both frequency and input power values using `opCon` and `opVal` arguments.

Enter the `getop(h)` command at the command line to get the operating conditions for the RF circuit object `h`.

`opVal` — Value of operating conditions
scalar

Value of the operating conditions specified using the `opCon` argument, specified as a scalar.

Example: `loglog(h,'Pout','Pin','mW','bias',1.5)` plots the data using a log-log scale for circuit object, `h`, with `opCon` set to `'bias'` and `value` set to `1.5`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `loglog(h,'Pout','Pin','mW','bias',1.5,'Freq',2.4)`

`Freq` — Frequency value
positive scalar

Frequency value used to plot the data using a log-log scale, specified as the comma-separated pair consisting of `'Freq'` and a positive scalar in Hz.

`Pin` — Input power level
scalar

Input power level used to plot the data using a log-log scale, specified as the comma-separated pair consisting of `'Pin'` and a scalar in dBm.

Output Arguments

`lineseries` — Lineseries object
column vector of object handles

Lineseries object, returned as a column vector of object handles.

Tips

- Use the Property Editor (`propertyeditor`) or the MATLAB `set` function to change Chart Line. The reference pages for MATLAB functions such as `figure`, `axes`, and `text` also list available properties and provide links to more complete property descriptions.

Note Use the MATLAB `loglog` function to create a log-log scale plot of parameters that are specified as vector data and are not part of a circuit (`rfckt`) object or data (`rfdata`) object.

- If `h` has multiple operating conditions, such as from a `.p2d` or `.s2d` file, the `loglog` function operates as follows:
  - If you do not specify any operating conditions as arguments to the `loglog` method, then the method plots the parameter values based on the currently selected operating condition.
  - If you specify one or more operating conditions, the `loglog` method plots the parameter values based on those operating conditions.
  - When you use an operating condition for the `xparameter` input argument, the method plots the parameters for all operating condition values.

Version History

Introduced in R2007a

See Also

`smithplot` | `analyze` | `extract` | `listformat` | `listparam` | `semilogy` | `plot` | `plotyy` | `polar` | `smith` | `semilogx` | `read`

plot

Plot circuit object parameters on X-Y plane

Syntax

```
plot(h,circuitPara)
plot(h,circuitPara,xAxisPara,xAxisFmt)
plot(h,circuitPara,xAxisPara,xAxisFmt,opCon,opVal)
plot( ___ ,Name,Value)
plot(h,'budget', ___ )
plot(h,'mixerspur',k,pin,fin)
plot( ___ ,format)
lineseries = plot( ___ )
```

Description

`plot(h,circuitPara)` plots the circuit parameters specified in `circuitPara` from the RFCKT or RF data object `h` on the X-Y plane in the default format. You can specify one or more circuit parameters in `circuitPara`.

`plot(h,circuitPara,xAxisPara,xAxisFmt)` plots the circuit parameters on the X-Y plane along with the variables `xAxisPara` and their corresponding format `xAxisFmt`.

Derive `xAxisPara` and `xAxisFmt` for the RFCKT or RF data object `h` using the `listparam(h)` and `listformat(h,'xAxisPara')` commands, respectively.

`plot(h,circuitPara,xAxisPara,xAxisFmt,opCon,opVal)` plots the circuit parameters on the X-Y plane with operating conditions `opCon` and operating values `opVal`.

Derive operating conditions for the RFCKT or RF data object `h` using the `getop(h)` command.

`plot( ___ ,Name,Value)` plots the circuit parameters with name-value arguments. Specify the name-value argument after any of the input argument combinations in the previous syntaxes.

`plot(h,'budget', ___ )` plots budget data on the X-Y plane from the `rfckt.cascade` object `h`. Specify any of the input argument combinations in the previous syntaxes after `'budget'` .

`plot(h,'mixerspur',k,pin,fin)` plots the spur power of the `rfckt.mixer` object or `rfckt.cascade` object `h` that contains one or more mixers. For more information on plotting mixer spur power, see the “Visualize Mixer Spurs” example.

Note For all circuit objects except those that contain data from a data file, you must perform a frequency domain analysis with the `analyze` function before calling `plot`.

`plot( ___ ,format)` plots the data in the specified `format`.

`lineseries = plot( ___ )` returns a column vector of handles to `lineseries` objects with one handle per line. This output is the same as the output returned by the MATLAB `plot` function.

Examples

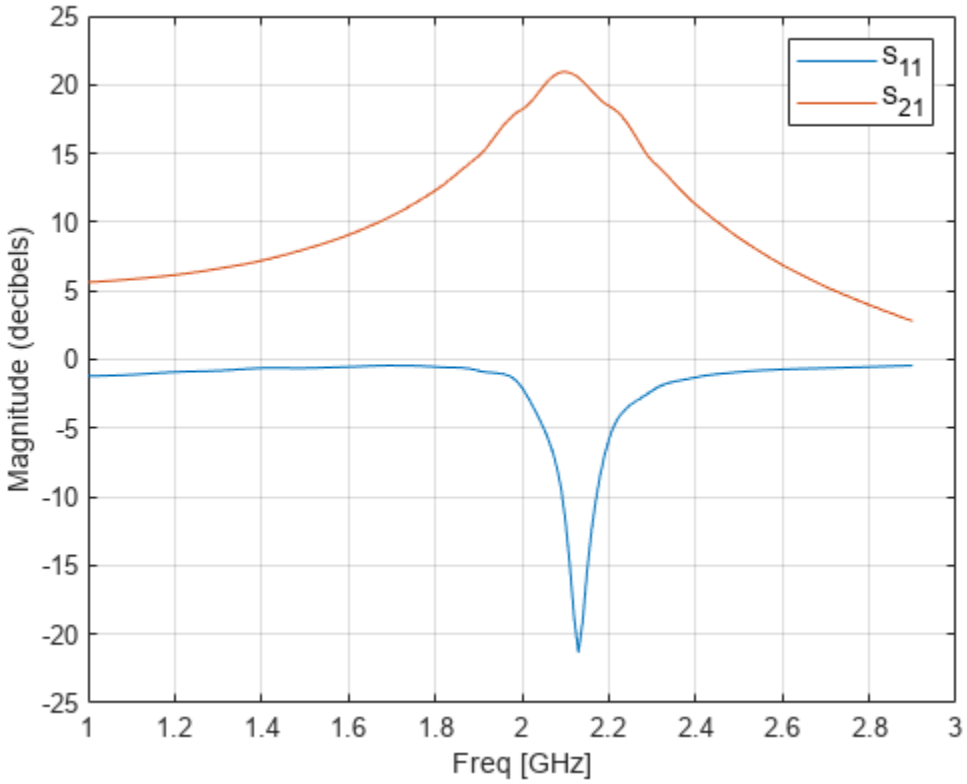
Plot Circuit Parameters of RFCKT Object on X-Y Plane

Create an RFCKT amplifier object.

```
amp = rfckt.amplifier;
```

Plot the S11 and S12 parameters on the X-Y plane.

```
plot(amp,'S11','S21')
```



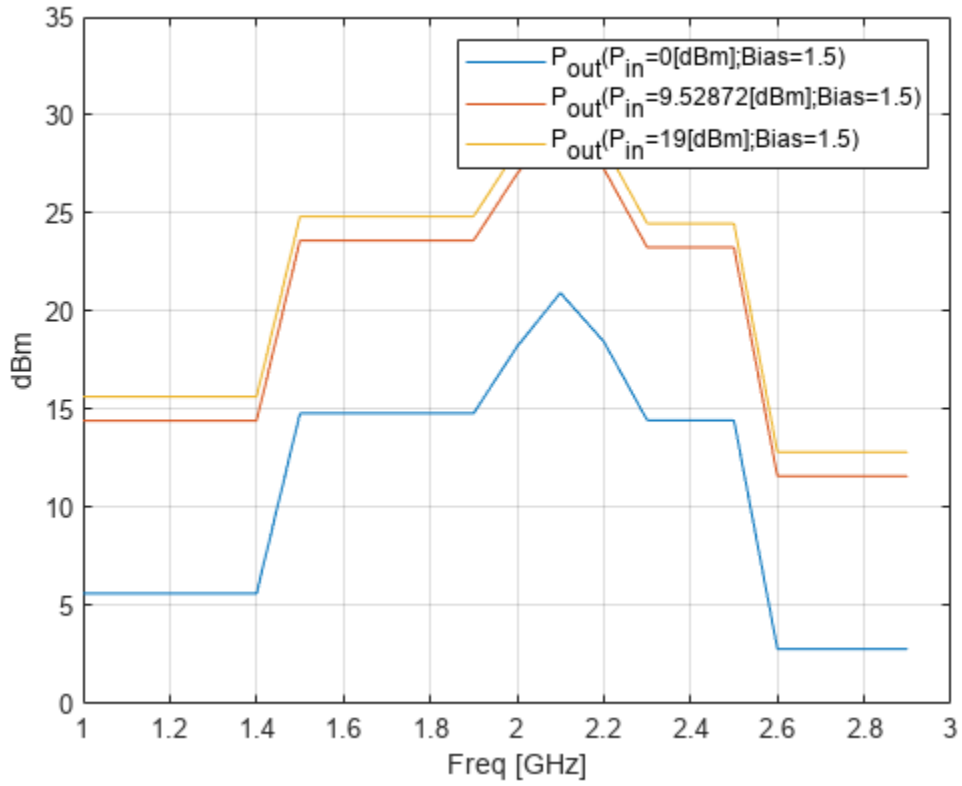
Plot Output Power of Amplifier in X-Y Plane

Create an amplifier object from a P2D file.

```
ckt1 = read(rfckt.amplifier,'default.p2d');
```

Plot the output power of the amplifier in the X-Y plane.

```
plot(ckt1,'Pout','Freq','GHz')
```



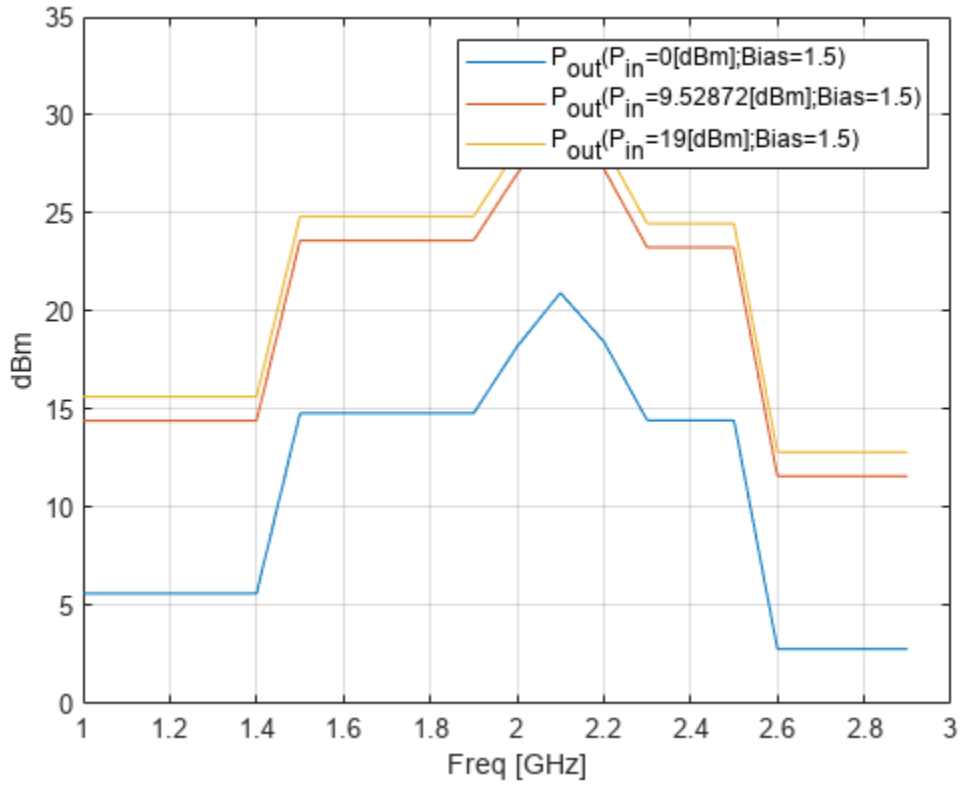
Plot Output of Amplifier with Biasing Condition

Create an amplifier object from a P2D file.

```
ckt1 = read(rfckt.amplifier,'default.p2d');
```

Plot the output power of the amplifier with the bias set to `1.5` v.

```
plot(ckt1,'Pout','Freq','GHz','bias',1.5)
```



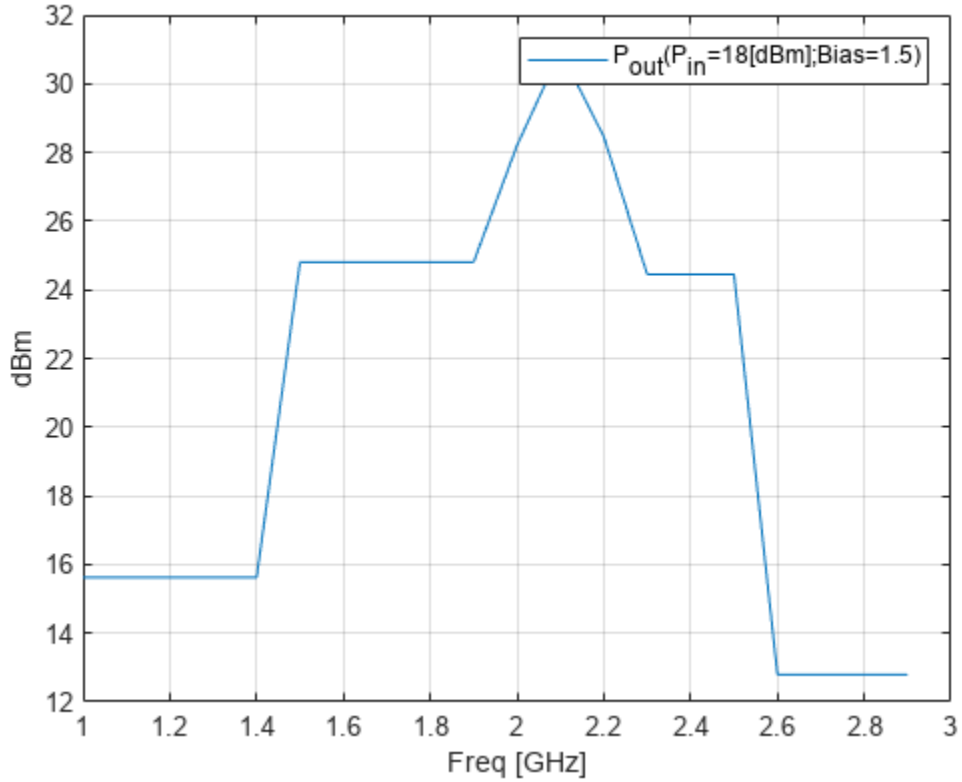
Plot Output Power of Amplifier at Specified Input Power

Create an amplifier object from a P2D file.

```
ckt1 = read(rfckt.amplifier, 'default.p2d');
```

Plot the output power of the amplifier when the input power is 18 dBm.

```
plot(ckt1,'Pout','Freq','GHz','bias',1.5,'Pin',18)
```



Plot Budget Data on X-Y Plane

Create two amplifiers by specifying an `rfdata.network` object as an input to an `rfckt.amplifier` object.

```
ai1 = rfckt.amplifier('NetworkData', ...
    rfdata.network('Type','S','Freq',2.1e9,'Data',[0,0;3.98,0]), ...
    'NoiseData',2,'NonlinearData',35);
ai2 = rfckt.amplifier('NetworkData', ...
    rfdata.network('Type','S','Freq',2.1e9,'Data',[0,0;31.66,0]), ...
    'NoiseData',8,'NonlinearData',37);
```

Create a microstrip transmission line with an `rfckt.microstrip` object.

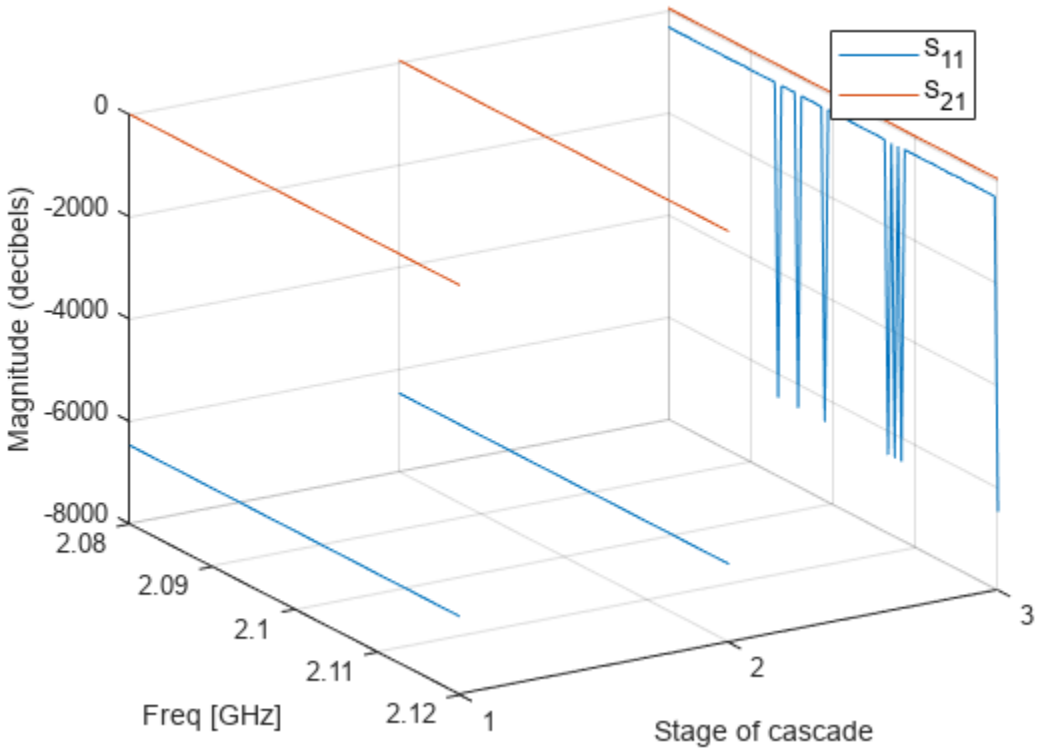
```
tx1 = rfckt.microstrip('Thickness',0.0075e-6);
```

Cascade the circuit using an `rfckt.cascade` object.

```
c = rfckt.cascade('Ckts',{ai1 ai2 tx1});
```

Analyze the cascaded circuit and plot the 3-D S11 and S21 plot.

```
analyze(c,linspace(2.08e9,2.12e9,100));
plot(c,'budget','S11','S21','Freq','GHz')
```



Plot S11 of Microstrip Transmission Line

Create an `rfckt.microstrip` object.

```
tx1 = rfckt.microstrip;
```

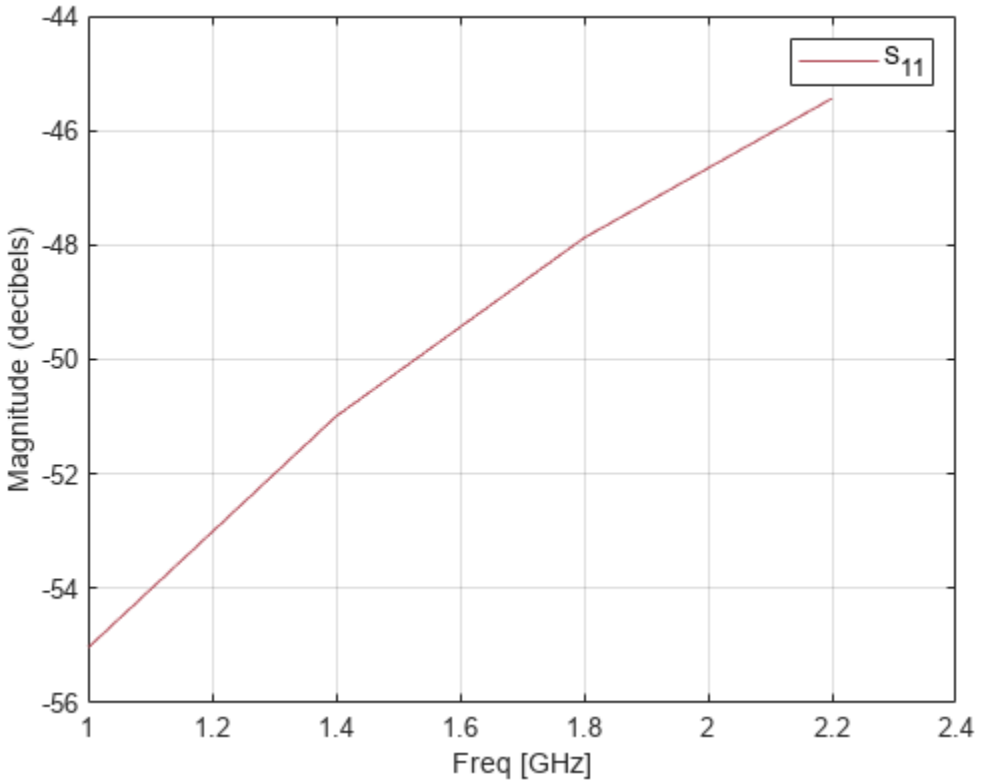
Analyze and plot the S11 parameter of the microstrip transmission line from 1– 2.4 GHz

```
analyze(tx1,1e9:0.4e9:2.4e9);
lineseries = plot(tx1,'S11')
```

```
lineseries =
  Line (S_{11}) with properties:

              Color: [0 0.4470 0.7410]
          LineStyle: '-'
          LineWidth: 0.5000
             Marker: 'none'
         MarkerSize: 6
    MarkerFaceColor: 'none'
              XData: [1 1.4000 1.8000 2.2000]
              YData: [-55.0369 -50.9945 -47.8763 -45.4391]

  Show all properties
```

Change the color of the S11 plot.

```
set(lineseries,'Color',[0.7 0.3 0.35])
```



Input Arguments

h — RF circuit or data object

RFCKT object | RF Data object

RF circuit or data object, specified as an RFCKT or RF data object. For a complete list of RFCKT and RF data objects, see “RF Circuit Objects” and “RF Data Objects”, respectively.

Data Types: `char` | `string`

circuitPara — Valid circuit parameters

character vector | string scalar

Valid circuit parameters of the RFCKT or RF data object, specified as a character vector or string scalar.

Use `listparam(h)` for a list of valid parameters for the circuit or data object `h`. You can also use `listformat(h,parameter)` to see the valid formats for a specific parameter. For more information, see `listparam`.

xAxisPara — Independent variable to plot with circuit parameters

`Pin` (default) | `Freq` | `AM`

Independent variable to plot with the circuit parameters `circuitPara`, values specified in the table given. This table shows the `circuitPara` and their corresponding `xAxisPara` values. The function uses the default values listed in the table if you do not specify `xAxisPara`.

circuitPara Value	xAxisPara Value
`Pout`, `Phase`, `LS11`, `LS12`, `LS21`, `LS22`	`Pin` (default), `Freq`
`S11`, `S12`, `S21`, `S22`, `NF`, `IIP3`, `OIP3`, `GroupDelay`, `VSWRIn`, `VSWROut`, `GammaIn`, `GammaOut`, `FMIN`, `GammaOPT`, `RN`, `TF1`, `TF2`, `Gt`, `Ga`, `Gp`, `Gmag`, `Gmsg`, `GammaMS`, `GammaML`, `K`, `Delta`, `Mu`, `MuPrime`	`Freq`
`AM/AM`, `AM/PM`	`AM`

xAxisFmt — xAxisPara format
`dBm` (default) | character vector | string scalar

`xAxisPara` format, specified as a character vector or string scalar. You do not need to specify `xAxisFmt` when `xAxisPara` is an operating condition.

This table shows the commonly used `xAxisPara` values and their corresponding `xAxisFmt` values. The function uses the default values listed in the table if you do not specify `xAxisFmt` .

xAxisPara Value	xAxisFmt Value
`Pin`	`dBm` (default), `mW`, `W`, `dBW`
`Freq`	`THz`, `GHz`, `MHz`, `KHz`, `Hz` If you do not specify `xAxisFmt`, the function chooses the `xAxisFmt` value that provides the best scaling for the given `xAxisPara` value.
`AM`	`Magnitude (decibels)` (default), `Magnitude (linear)`

Example: `plot(h,'Pout','Pin','mW')` plots data on the X-Y plane for the circuit object `h` with `xAxisPara` set to `'Pin'` and `xAxisFmt` set to `'mW'`.

opCon — Operating conditions
string scalar | character vector

Operating conditions derived from a P2D or S2D file, specified as a string scalar or a character vector.

For some circuit parameters, you can specify a set of frequency or input power values at which the function plots the specified parameter.

For example:

- When plotting large-signal S-parameters as a function of input power, you can specify frequency points of interest using `opCon` and `opVal` arguments.
- When plotting large-signal S-parameters as a function of frequency, you can specify input power levels of interest using `opCon` and `opVal` arguments.

- When plotting parameters as a function of an operating condition, you can specify both frequency and input power values using `opCon` and `opVal` arguments.

Enter the `getop(h)` command at the command line to get the operating conditions for the RF circuit object `h`.

`opVal` — Value of operating conditions
scalar

Value of the operating conditions in the `opCon` argument, specified as a scalar.

Example: `plot(h,'Pout','Pin','mW','bias',1.5)` plots the data on the X-Y plane for circuit object `h` with `opCon` set to `'bias'` and `value` set to `1.5`.

`'budget'` — Budget parameter of `rfckt.cascade` object
budget parameter object

Budget parameter of an `rfckt.cascade` object, specified as a budget parameter object handle.

The following table summarizes the parameters and formats that are available for a budget plot.

Parameter	Format

S_{11}, S_{12}, S_{21}, S_{22}, S_{ij}	`Magnitude (decibels)`
	`Magnitude (linear)`
	`Angle (degrees)`
	`Real`
	`Imaginary`

`OIP3`	`dBm`
	`dBW`
	`W`
	`mW`

`NF`	`Magnitude (decibels)` `Magnitude (linear)`

`format` — Format of data
`'Real'` | `'None'` | string scalar | character vector | ...

Format of data, specified as a string scalar or character vector. The format determines if RF Toolbox converts the parameter values to a new set of units or operates on the components of complex parameter values. For example:

- Specify `format` as `'Real'` to plot the real part of the circuit parameter.
- Specify `format` as `'None'` to plot the unchanged parameter values.

Use the `listformat` function to get a list of the valid formats for a particular parameter.

`k` — Index of the circuit object
`'all'` (default) | integer

Index of the circuit object to plot spur power, specified as an integer or `'all'`. This value creates a budget plot of the spur power for `h`. Use `0` to plot the power at the input of `h`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `plot(h,'Pout','Pin','Freq',2.1e9)`

`Freq` — Frequency
positive scalar

Frequency, specified as a positive scalar in Hz.

`Pin` — Input power level
positive scalar

Input power level, specified as a positive scalar in dBm.

`fin` — Input frequency value
positive scalar

Input frequency value, specified as a positive scalar in Hz to plot the spur power. The default value of `fin` varies based on `h`.

- If `h` is an `rfckt.mixer` object, the default value of `fin` is the input frequency at which the magnitude of the S_{21} parameter of the mixer, in decibels, is highest.
- If `h` is an `rfckt.cascde` object, the default value of `fin` is the input frequency at which the magnitude of the S_{21} parameter of the first mixer in the cascade is highest.

Note When you create a spur plot for an object, the previous input frequency value is used for subsequent plots until you specify a different value.

Output Arguments

`lineseries` — `lineseries` object
column vector of object handles

`lineseries` object, returned as a column vector of object handles.

Tips

- Use the Property Editor (`propertyeditor`) or the MATLAB `set` function to change Chart Line. For more information and complete description of object properties, see `figure`, `axes`, and `text`.
- Use the MATLAB `plot` function to plot network parameters specified as vector data and not as part of a circuit (`rfckt`) object or data (`rfdata`) object.

Alternatives

`rfplot`The function creates magnitude-frequency plots for RF Toolbox S-parameter objects.

Version History

Introduced before R2006a

See Also

analyze | calculate | circle | extract | listformat | listparam | loglog | plot | plotyy | getop | polar | semilogx | semilogy | smith | write | getz0 | read | restore | getop | groupdelay

smith

Plot circuit object parameters on Smith chart

Syntax

```
smith(hnet,i,j)
lineseries = smith(hnet,i,j)

smith(h,circuitPara)
smith(h,circuitPara,xAxisPara,xAxisFmt)
smith(h,circuitPara,xAxisPara,xAxisFmt,opCon,opVal)
smith( ___ ,Name,Value)
smith( ___ ,gridType)
[lineseries,hsm] = smith( ___ )
```

Description

RF Network Object

`smith(hnet,i,j)` plots the $(i, j)^{th}$ parameter of the network object `hnet` on a Smith® Chart.

`lineseries = smith(hnet,i,j)` returns the line series property object `lineseries`. The `lineseries` object can be used to set the properties of the data in the Smith plot.

RFCKT or RF Data Objects

`smith(h,circuitPara)` plots the circuit parameter `circuitPara` from the RFCKT or RF data object `h` on a Smith chart. You can specify multiple circuit parameters in this syntax.

`smith(h,circuitPara,xAxisPara,xAxisFmt)` plots the circuit parameters `circuitPara` on a Smith chart along with the variables `xAxisPara` and their corresponding format `xAxisFmt`.

Derive `xAxisPara` and `xAxisFmt` for the RFCKT or RF data object `h` using the `listparam(h)` and `listformat(h,'xAxisPara')` commands, respectively.

`smith(h,circuitPara,xAxisPara,xAxisFmt,opCon,opVal)` plots the circuit parameters on a Smith chart with operating conditions `opCon` and operating values `opVal` for the circuit object `h`.

Derive operating conditions for the RFCKT or RF data object `h` using the `getop(h)` command

`smith( ___ ,Name,Value)` plots the data of a RFCKT or RF data object with name-value arguments. Specify name-value argument after any of the input argument combinations in the previous syntaxes.

`smith( ___ ,gridType)` plots the data of the RFCKT or RF data object on a Smith chart with the specified grid type.

`[lineseries,hsm] = smith( ___ )` returns the line series property object `lineseries` and Smith chart property object `hsm`.

Examples

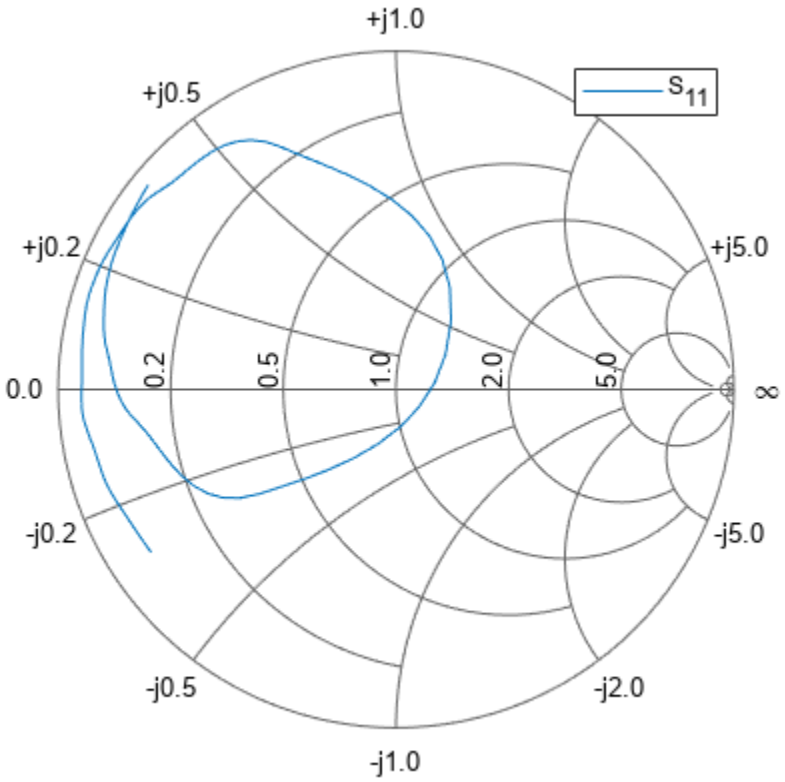
Plot S-Parameters on Smith Plot

Create an S-Parameters object from the specified file.

```
S = sparameters('default.s2p');
```

Plot the input reflection coefficient S11 on a Smith chart.

```
smith(S,1,1)
```



Change Color of Data Line of RF Filter in Smith Chart

Create an S-parameters object from the specified Touchstone® file of an RF Filter.

```
S = sparameters('RFBudget_RF.s2p');
```

Plot the input reflection coefficient S11 on a Smith chart.

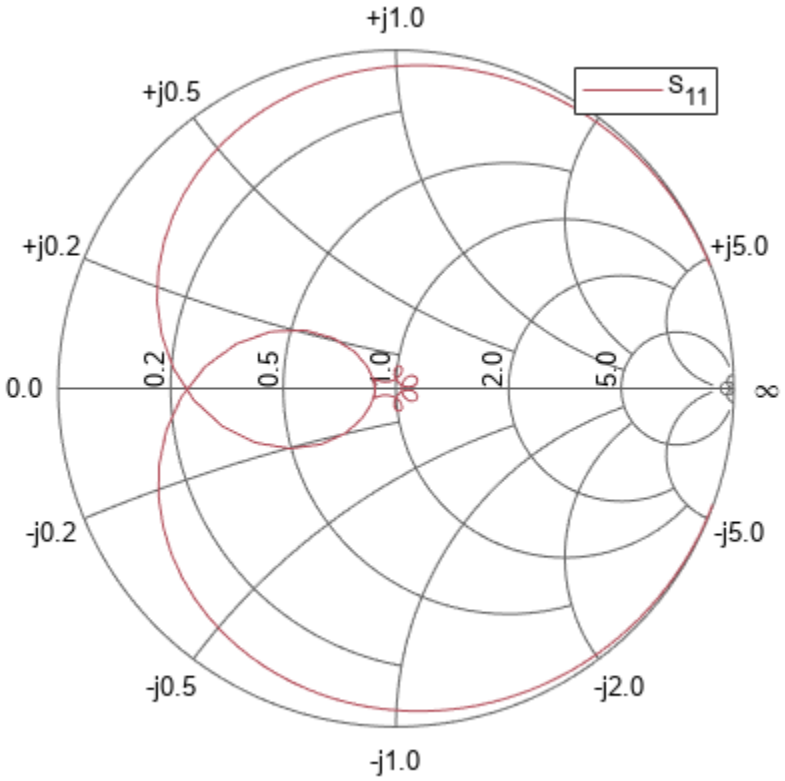
```
lineseries = smith(S,1,1)
```

```
lineseries =
  Line (S_{11}) with properties:

              Color: [0 0.4470 0.7410]
          LineStyle: '-'
```

```
          LineWidth: 0.5000
             Marker: 'none'
         MarkerSize: 6
    MarkerFaceColor: 'none'
              XData: [0.9369 0.9364 0.9360 0.9355 0.9351 0.9346 0.9341 ... ]
              YData: [-0.3435 -0.3447 -0.3458 -0.3470 -0.3482 -0.3494 ... ]

  Show all properties
```

Change the color of the S11 data line in the Smith chart.

```
lineseries.Color = [0.7 0.3 0.35]
```




```
lineseries =
  Line (S_{11}) with properties:

              Color: [0.7000 0.3000 0.3500]
          LineStyle: '-'
          LineWidth: 0.5000
             Marker: 'none'
         MarkerSize: 6
    MarkerFaceColor: 'none'
              XData: [0.9369 0.9364 0.9360 0.9355 0.9351 0.9346 0.9341 ... ]
              YData: [-0.3435 -0.3447 -0.3458 -0.3470 -0.3482 -0.3494 ... ]

  Show all properties
```

Plot Input and Output Reflection Coefficient in Smith Chart

Import network parameters, noise data, and power data from the `default.amp` file into the amplifier object `h`.

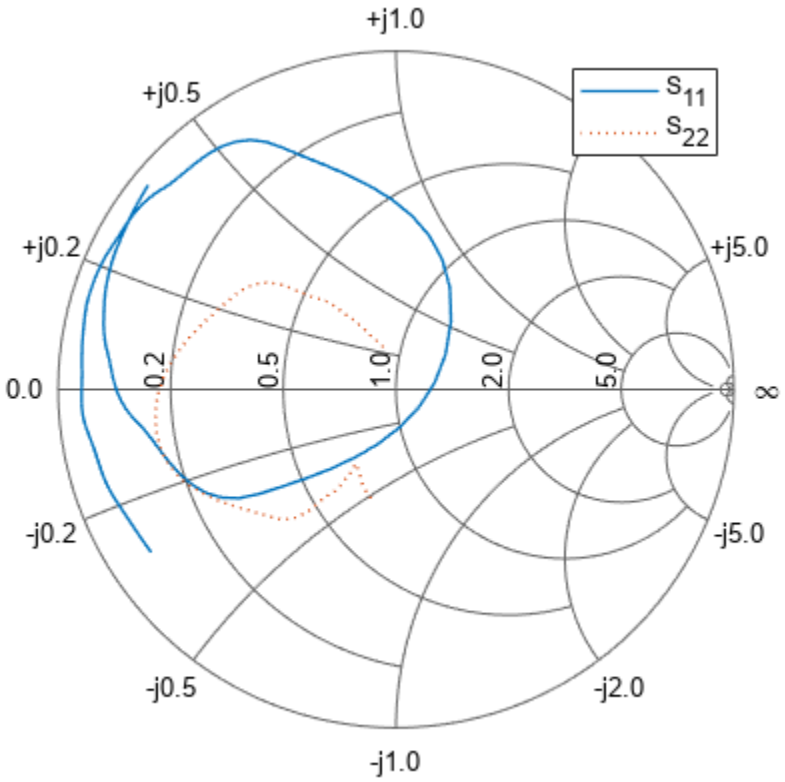
```
h = read(rfckt.amplifier,'default.amp');
```

Set the interpolation method of the amplifier object, `h`, to `cubic`.

```
h.IntpType = 'cubic';
```

Plot the S11 and S22 parameters of the amplifier object `h` on a Z Smith chart.

```
lineseries = smith(h,'S11','S22');
lineseries(1).LineStyle = '-';
lineseries(1).LineWidth = 1;
lineseries(2).LineStyle = ':';
lineseries(2).LineWidth = 1;
```



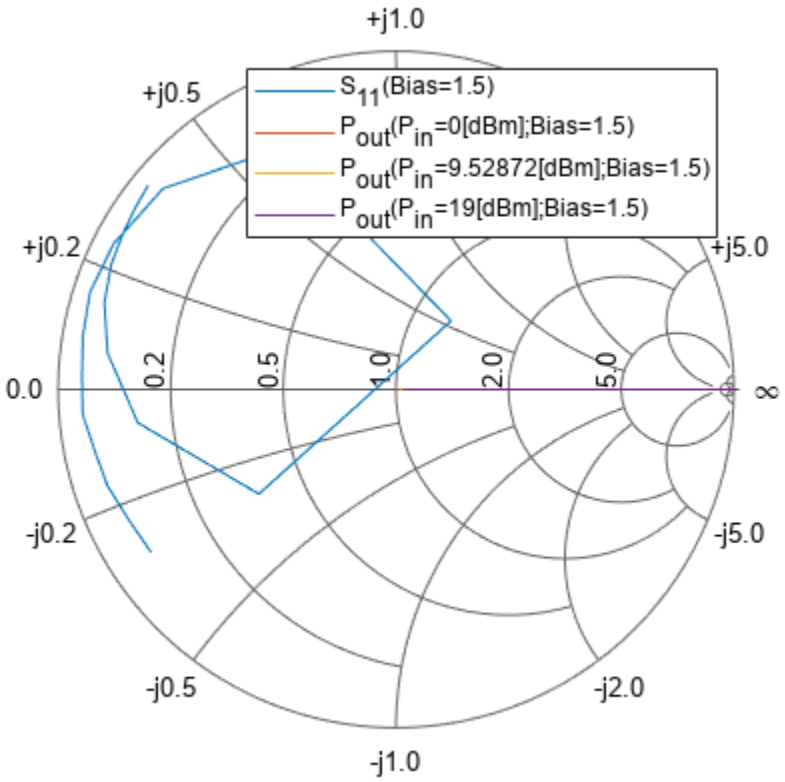
Plot Output Power of Amplifier

Create an amplifier object from the specified P2D file.

```
ckt1 = read(rfckt.amplifier, 'default.p2d');
```

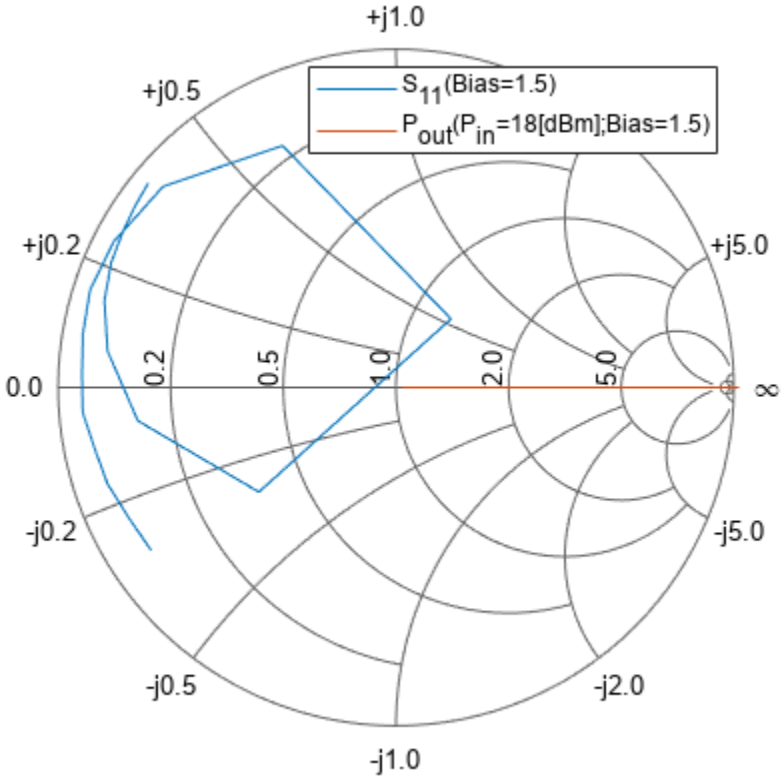
Plot the input reflection coefficient and output power of the amplifier.

```
smith(ckt1,'S11','Pout','Freq','GHz');
```



Plot the output power of the amplifier when the input power is at 18 dBm.

```
smith(ckt1,'S11','Pout','Freq','GHz','bias',1.5,'Pin',18);
```



Plot Amplifier Data on Smith Chart

Create an amplifier object from the specified Touchstone® file.

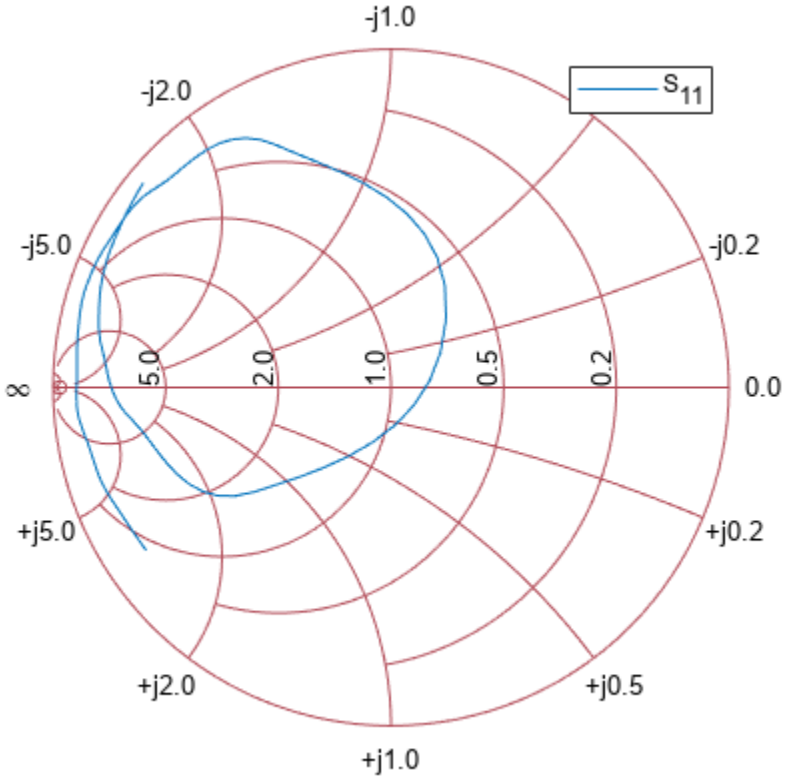
```
amp = read(rfckt.amplifier,'default.s2p');
```

Set the circuit parameters as `S11` and grid type as `y` and plot the amplifier data on the Smith chart.

```
[lineseries,hsm] = smith(amp,'S11','y');
```

Change the color of the Smith chart.

```
set(hsm,'Color',[0.7 0.3 0.35]);
```



Input Arguments

hnet — RF network parameter object
`abcdparameters` | `sparameters` | `yparameters` | `zparamters` | `gparameters` | `hparamters` | `tparameters`

RF network parameter object, specified as a one of these objects:

- `abcdparameters`
- `sparameters`
- `yparameters`
- `zparameters`
- `gparameters`
- `hparameters`
- `tparameters`

Note Use the `smithplot` function to plot network parameters that are not part of an RFCKT or RF data object but are specified as vector data.

`i,j` — $(i, j)^{th}$ parameter of network object
positive integers

$(i, j)^{th}$ parameter of the network object, `hnet`, specified as positive integers.

- When `hnet` is a hybrid or hybrid-g parameter object, specify *i* and *j* in the range `[1,2]` .
- When `hnet` is an ABCD, S, Y, or Z-parameters object, specify *i* and *j* such that they are less than or equal to number of ports in `hnet`.

Example: `smith(hnet,2,1)`

h — RFCKT or RF data object
`rfckt` or `rfdata` object

RFCKT or RF data object, specified as a `rfckt` or `rfdata` object.

For complete list of RFCKT and RF data objects, see “RF Circuit Objects” and “RF Data Objects”.

`circuitPara` — Valid RFCKT or RF data object parameter
character vector | string scalar

Valid RFCKT or data object parameter, specified as a character vector or string scalar.

Use `listparam(h)` for a list of valid parameters for the circuit or data object `h`. You can also use `listformat(h,parameter)` to see the valid formats for a specific parameter.

`xAxisPara` — Independent variables to plot with circuit parameters
`Pin` (default) | `Freq` | `AM` | character vector | string scalar

Independent variables to plot with the circuit parameters, `circuitPara`, specified as a character vector or string scalar.

This table shows the commonly used `circuitPara` and their corresponding `xAxisPara` values. The function uses the default values listed in the table if you do not specify `xAxisPara` .

`circuitPara` Value	`xAxisPara` Value
`Pout`, `Phase`, `LS11`, `LS12`, `LS21`, `LS22`	`Pin` (default), `Freq`
`S11`, `S12`, `S21`, `S22`, `NF`, `IIP3`, `OIP3`, `GroupDelay`, `VSWRIn`, `VSWROut`, `GammaIn`, `GammaOut`, `FMIN`, `GammaOPT`, `RN`, `TF1`, `TF2`, `Gt`, `Ga`, `Gp`, `Gmag`, `Gmsg`, `GammaMS`, `GammaML`, `K`, `Delta`, `Mu`, `MuPrime`	`Freq`
`AM/AM`, `AM/PM`	`AM`

`xAxisFmt` — `xAxisPara` format
`dBm` (default) | character vector | string scalar

`xAxisPara` format, specified as a character vector or string scalar. You do not need to specify `xAxisFmt` when `xAxisPara` is an operating condition.

This table shows the commonly used `xAxisPara` and their corresponding `xAxisFmt`. The function uses the default values listed in the table if you do not specify `xAxisFmt` .

`xAxisPara` Value	`xAxisFmt` Value
`Pin`	`dBm` (default), `mW`, `W`, `dBW`

xAxisPara Value	xAxisFmt Value
`Freq`	`THz`, `GHz`, `MHz`, `KHz`, `Hz` By default, `xAxisFmt` is chosen to provide the best scaling for the given `xAxisPara` values.
`AM`	`Magnitude (decibels)` (default), `Magnitude (linear)`

Example: `smith(h,'Pout','Pin','mW')` plots data on a Smith chart for circuit object, `h`, with `xAxisPara` set to `'Pin'` and `xAxisFmt` set to `'mW'`.

`opCon` — Operating conditions
string scalar | character vector

Operating conditions derived from a P2D or S2D file, specified as a string scalar or a character vector.

For some circuit parameters, you can specify a set of frequency or input power values at which the function plots the specified parameter.

For example:

- When plotting large-signal S-parameters as a function of input power, you can specify frequency points of interest using `opCon` and `opVal` arguments.
- When plotting large-signal S-parameters as a function of frequency, you can specify input power levels of interest using `opCon` and `opVal` arguments.
- When plotting parameters as a function of an operating condition, you can specify both frequency and input power values using `opCon` and `opVal` arguments.

Enter the `getop(h)` command at the command line to get the operating conditions for the RF circuit object `h`.

`opVal` — Value of operating conditions
scalar

Value of the operating conditions specified using the `opCon` argument, specified as a scalar.

Example: `smith(h,'Pout','Pin','mW','bias',1.5)` plots the data on a Smith chart for circuit object, `h`, with `opCon` set to `'bias'` and `value` set to `1.5`.

`gridType` — Smith chart grid type
`'z'` (default) | `'y'` | `'zy'` | character vector | string scalar

Smith chart grid type, specified as a character vector or string scalar.

Example: `smith(h,'Pout','Pin','mW','bias',1.5,'Freq',2.4,'y')` plots the data on a Smith chart for circuit object, `h`, with the Smith chart grid type set to the admittance grid.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `smith(h,'Pout','Pin','mW','bias',1.5,'Freq',2.4)`

`Freq` — Frequency value
positive scalar

Frequency value used to plot the Smith chart, specified as the comma-separated pair consisting of `'Freq'` and a positive scalar in Hz.

`Pin` — Input power level
scalar

Input power level used to plot the Smith chart, specified as the comma-separated pair consisting of `'Pin'` and a scalar in dBm.

`fin` — Input frequency value to plot spur power
positive scalar

Input frequency value used to plot the spur power in the Smith chart, specified as the comma-separated pair consisting of `'fin'` and a positive scalar in Hz.

- When the input object to the function is an `rfckt.mixer` object, the default value of `fin` is the input frequency at which the magnitude in decibels of the S_{21} parameter of the mixer is the highest.
- When the input object to the function is an `rfckt.cascde` object, the default value of `fin` is the input frequency at which the magnitude of the S_{21} parameter of the first mixer in the cascade is highest.

Output Arguments

`lineseries` — Line series property object
column vector

Line series property object, returned as a column vector.

`hsm` — Smith chart properties
`rfchart.smith` object

Smith chart properties, returned as an `rfchart.smith` object.

More About

Change Properties of Smith Chart

The `smith` function returns the Smith chart properties object, `hsm`.

The `smith` function plots the Smith chart using the default property values. Use `set(hsm,'PropertyName',PropertyValue)` to change the property values of the chart and use `get(hsm)` to get the property values.

This table lists all the properties you can specify for a Smith chart object along with their descriptions and associated values. Use the table below to change the properties of the chart.

Name	Description	Units, Values
`Color`	Line color for a Z or Y Smith chart and the color of the Z line for a ZY Smith chart	Default value is `[0.4 0.4 0.4]` (dark gray). For more information, see “Color Specification” on page 3-207.
`LabelColor`	Color of the line labels	Default value is `[0 0 0]` (black). For more information, see “Color Specification” on page 3-207.
`LabelSize`	Size of the line labels	`FontSize`. Default value is `10`.
`LabelVisible`	Visibility of the line labels	`'on'` (default) or `'off'`
`LineType`	Line spec for a Z or Y Smith chart. For a ZY Smith chart, the Z line spec	Default value is `'-'` (solid line). For more information, see “Line Specification” on page 3-207.
`LineWidth`	Line width for a Z or Y Smith chart. For a ZY Smith chart, width of the Z line	Number of points. Default value is `0.5`.
`SubColor`	The Y line color for a ZY Smith chart	Default value is `[0.8,0.8,0.8]` (medium gray). For more information, see “Color Specification” on page 3-207.
`SubLineType`	The Y line spec for a ZY Smith chart	Default value is `':'` (dotted line). For more information, see “Line Specification” on page 3-207.
`SubLineWidth`	The Y line width for a ZY Smith chart	Number of points. Default value is `0.5`.
`Type`	Type of Smith chart	`'z'` (default), `'y'`, or `'zy'`
`Value`	Two-row matrix. First row specifies the values of the constant resistance and reactance lines that appear on the chart. For constant resistance or reactance lines, each element in second row specifies the value of the constant reactance or resistance line at which the corresponding line specified in first row ends	2-by-n matrix. Default is `[0.2000 0.5000 1.0000 2.0000 5.0000; 1.0000 2.0000 5.0000 5.0000 30.0000]`

Change Properties of Plotted Lines

The `smith` function returns a `lineseries` object as a column vector of handles to `lineseries` objects, one object per plotted line. For more information, see Chart Line.

Line Specification

Use the table provided to set Line style of the Smith Chart

Line Style	Description	Resulting Line
'-'	Solid line	
'--'	Dashed line	
':'	Dotted line	
'-.'	Dash-dotted line	

Color Specification

Use the table provided to set color of the Smith Chart.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
'red'	'r'	[1 0 0]	'#FF0000'	
'green'	'g'	[0 1 0]	'#00FF00'	
'blue'	'b'	[0 0 1]	'#0000FF'	
'cyan'	'c'	[0 1 1]	'#00FFFF'	
'magenta'	'm'	[1 0 1]	'#FF00FF'	
'yellow'	'y'	[1 1 0]	'#FFFF00'	
'black'	'k'	[0 0 0]	'#000000'	
'white'	'w'	[1 1 1]	'#FFFFFF'	

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	'#0072BD'	
[0.8500 0.3250 0.0980]	'#D95319'	
[0.9290 0.6940 0.1250]	'#EDB120'	
[0.4940 0.1840 0.5560]	'#7E2F8E'	
[0.4660 0.6740 0.1880]	'#77AC30'	
[0.3010 0.7450 0.9330]	'#4DBEEE'	
[0.6350 0.0780 0.1840]	'#A2142F'	

Tips

- Type `listparam(h)` to get a list of valid parameters for the circuit object `h`.

Note For all circuit objects, except those that contain data from a data file, you must use the `analyze` function to perform a frequency domain analysis before calling `smith`.

Version History

Introduced before R2006a

See Also

`smithplot` | `analyze` | `extract` | `listformat` | `listparam` | `loglog` | `plot` | `plotyy` | `polar` | `semilogx` | `semilogy` | `read`

semilogx

Plot RF circuit object parameters using log scale for *x*-axis

Syntax

```
semilogx(h,circuitPara)
semilogx(h,circuitPara,dataFormat)
semilogx(___,xAxisPara,xAxisFmt)
semilogx(___,opCon,opVal)
semilogx(___,Name,Value)
lineseries = semilogx(___)
```

Description

`semilogx(h,circuitPara)` plots the circuit parameter `circuitPara` from the RFCKT or RF data object `h` using a logarithmic scale for the *x*-axis. You can specify multiple circuit parameters in this syntax.

Note For all circuit objects except those that contain data from a data file, you must perform a frequency domain analysis with the `analyze` method before calling `semilogx`.

`semilogx(h,circuitPara,dataFormat)` plots the data of the RFCKT or RF data object using a logarithmic scale for the *x*-axis with the specified data format.

`semilogx(___,xAxisPara,xAxisFmt)` plots the circuit parameters `circuitPara` using a logarithmic scale for the *x*-axis along with the variables `xAxisPara` and their corresponding format `xAxisFmt`. Specify `xAxisPara` and `xAxisFmt` arguments after any of the input argument combinations in the previous syntaxes.

`semilogx(___,opCon,opVal)` plots the circuit parameters using a logarithmic scale for the *x*-axis with operating conditions `opCon` and operating values `opVal` for the circuit object `h`.

Derive operating conditions for the RFCKT or RF data object `h` using the `getop(h)` command

`semilogx(___,Name,Value)` plots the data of a RFCKT or RF data object with name-value arguments. Specify name-value argument after any of the input argument combinations in the previous syntaxes.

`lineseries = semilogx(___)` returns the line series property object `lineseries`. This output is the same as the output returned by the MATLAB `semilogx` function.

Examples

Plot S11 and S21 of Amplifier

Create an amplifier object from `default.s2p` file.

```
h = read(rfckt.amplifier,'default.s2p');
```

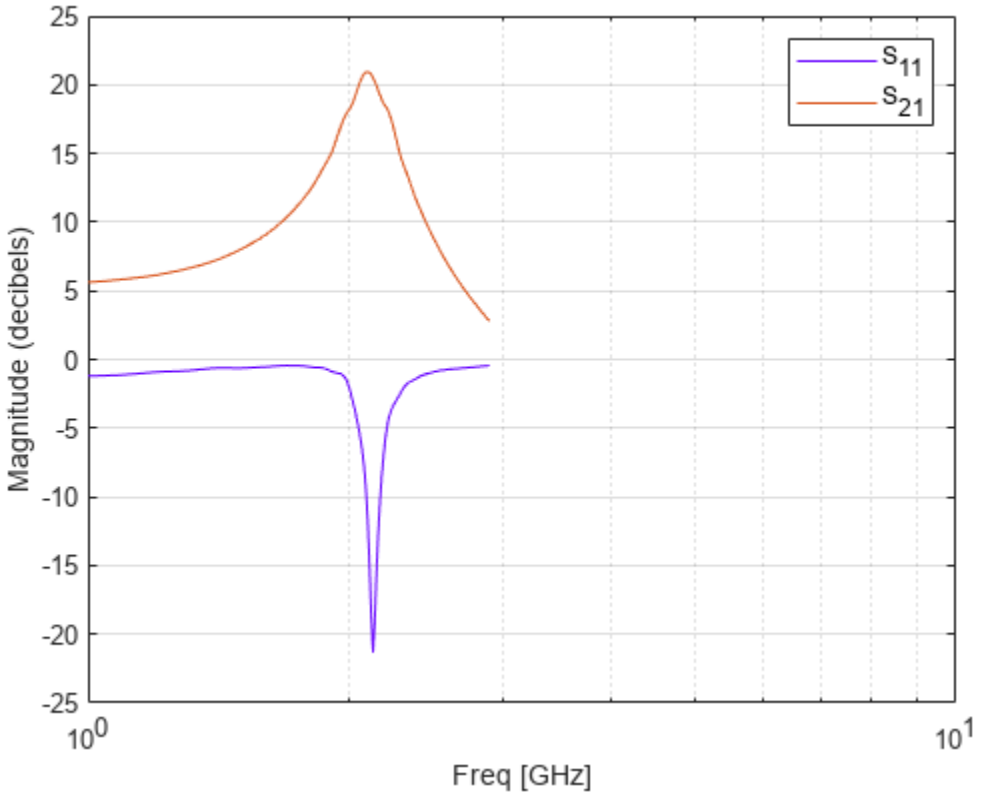
Plot S11 and S21 using log scale on x-axis.

```
lineseries = semilogx(h,'S11','S21')
```

```
lineseries =
  2x1 Line array:

  Line    (S_{11})
  Line    (S_{21})
```

Change the color of the S11 data.

```
lineseries(1).Color = [0.4 0 1];
```



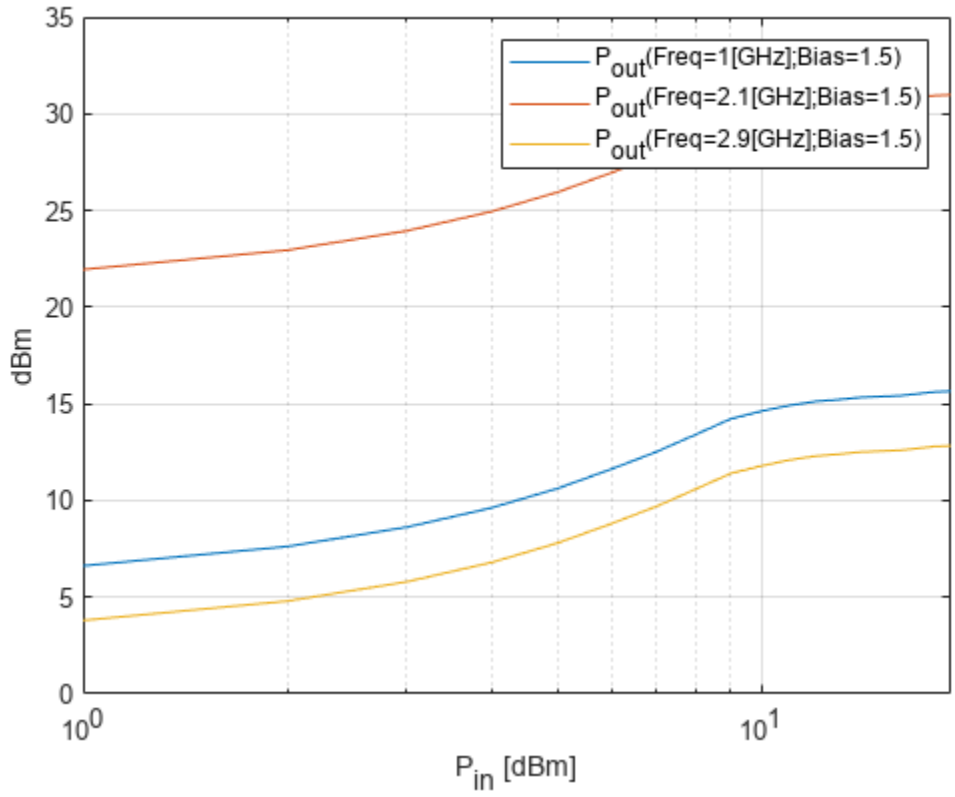
Plot Pout vs. Pin of Amplifier

Create an RFCKT amplifier object from the specified P2D file type.

```
h = read(rfckt.amplifier,'default.p2d');
```

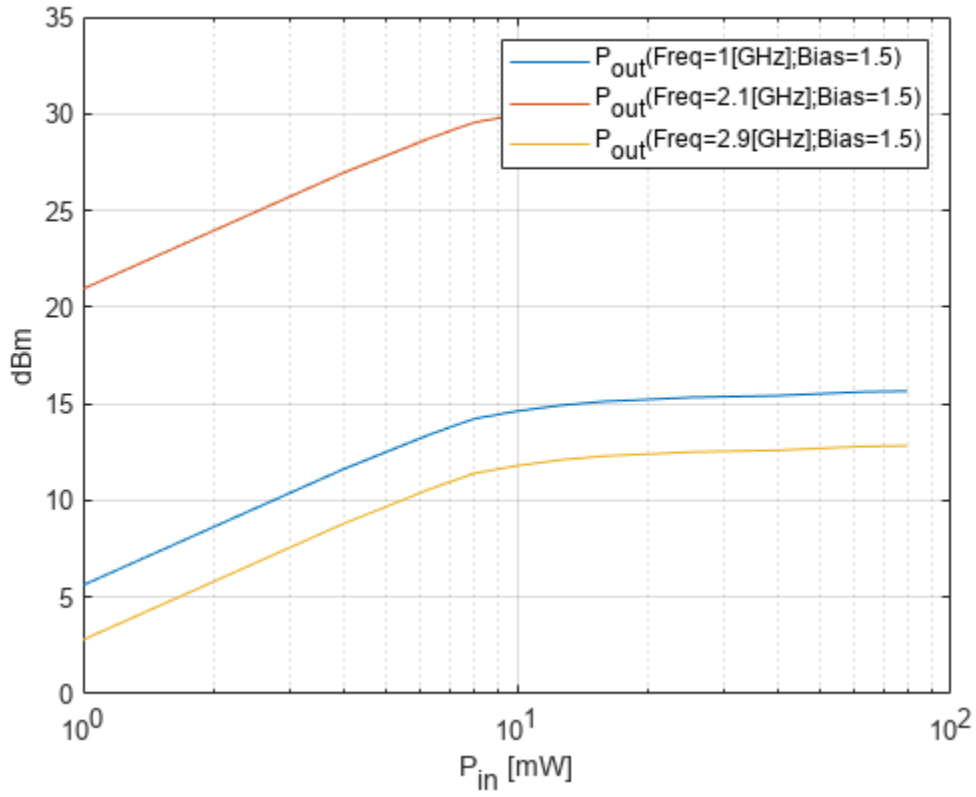
Plot the output power of the amplifier.

```
semilogx(h,'Pout')
```



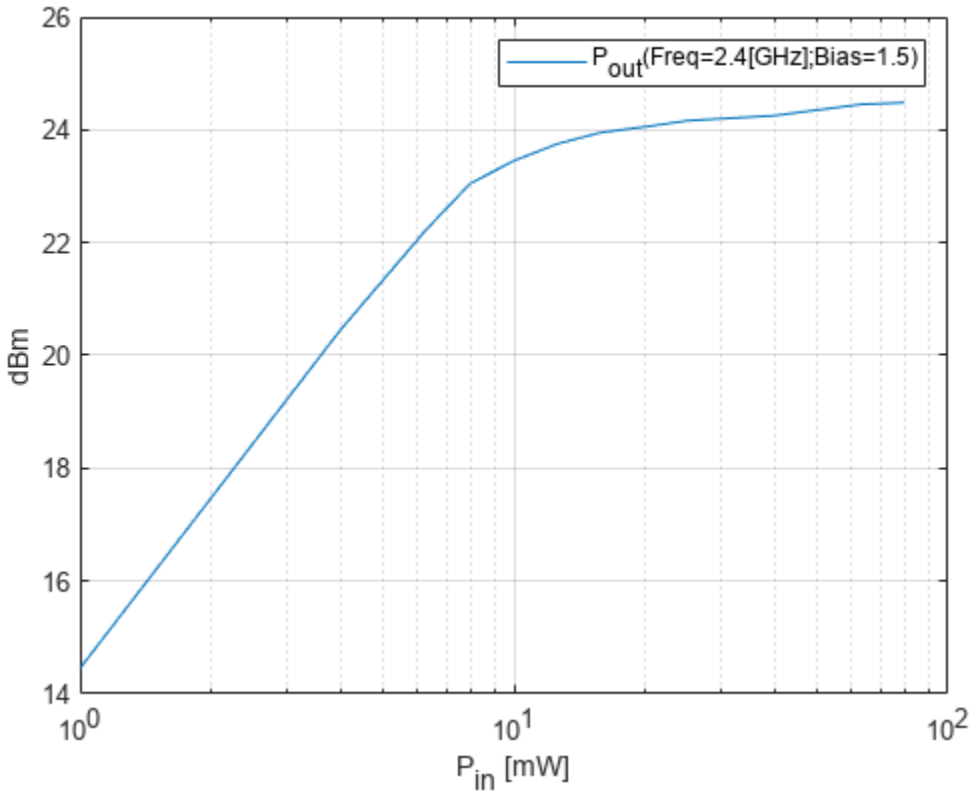
Plot the output power of the amplifier and set the format for `Pin` in milliwatts.

```
semilogx(h,'Pout','Pin','mW')
```



Plot the output power of the amplifier at `2.4` GHz.

```
semilogx(h,'Pout','Pin','mW','bias',1.5,'Freq',2.4e9)
```



Input Arguments

h — RFCKT or RF data object

`rfckt` or `rfdata` object

RFCKT or RF data object, specified as a `rfckt` or `rfdata` object.

For complete list of RFCKT and RF data objects, see “RF Circuit Objects” and “RF Data Objects”.

dataFormat — Format of data

character vector | string scalar | `'Magnitude (decibels)'` | `'Magnitude (linear)'` | `'Angle (degrees)'` | `'dBm'`

Format of the data to be plotted, specified as character vector or string scalar. Type `listformat(h,circuitPara)` command to see the available formats for a specified parameter.

Example: `lineseries = semilogy(h,'Pout','dBm')`

circuitPara — Valid RFCKT or RF data object parameter

character vector | string scalar

Valid RFCKT or data object parameter, specified as a character vector or string scalar.

Use `listparam(h)` for a list of valid parameters for the circuit or data object `h`.

xAxisPara — X-axis variable to plot with circuit parameters
`Pin` (default) | `Freq` | `AM` | character vector | string scalar

X-axis variable to plot with the circuit parameters, `circuitPara`, specified as a character vector or string scalar.

This table shows the commonly used `circuitPara` and their corresponding `xAxisPara` values. The function uses the default values listed in the table if you do not specify `xAxisPara` .

circuitPara Value	xAxisPara Value
`Pout`, `Phase`, `LS11`, `LS12`, `LS21`, `LS22`	`Pin` (default), `Freq`
`S11`, `S12`, `S21`, `S22`, `NF`, `IIP3`, `OIP3`, `GroupDelay`, `VSWRIn`, `VSWROut`, `GammaIn`, `GammaOut`, `FMIN`, `GammaOPT`, `RN`, `TF1`, `TF2`, `Gt`, `Ga`, `Gp`, `Gmag`, `Gmsg`, `GammaMS`, `GammaML`, `K`, `Delta`, `Mu`, `MuPrime`	`Freq`
`AM/AM`, `AM/PM`	`AM`

xAxisFmt — xAxisPara format
`dBm` (default) | character vector | string scalar

`xAxisPara` format, specified as a character vector or string scalar. You do not need to specify `xAxisFmt` when `xAxisPara` is an operating condition.

This table shows the commonly used `xAxisPara` and their corresponding `xAxisFmt`. The function uses the default values listed in the table if you do not specify `xAxisFmt` .

xAxisPara Value	xAxisFmt Value
`Pin`	`dBm` (default), `mW`, `W`, `dBW`
`Freq`	`THz`, `GHz`, `MHz`, `KHz`, `Hz` By default, `xAxisFmt` is chosen to provide the best scaling for the given `xAxisPara` values.
`AM`	`Magnitude (decibels)` (default), `Magnitude (linear)`

Example: `semilogx(h,'Pout','Pin','mW')` plots data using a logarithmic scale for the *x*-axis for circuit object, `h`, with `xAxisPara` set to `'Pin'` and `xAxisFmt` set to `'mW'`.

opCon — Operating conditions
string scalar | character vector

Operating conditions derived from a P2D or S2D file, specified as a string scalar or a character vector.

For some circuit parameters, you can specify a set of frequency or input power values at which the function plots the specified parameter.

For example:

- When plotting large-signal S-parameters as a function of input power, you can specify frequency points of interest using `opCon` and `opVal` arguments.

- When plotting large-signal S-parameters as a function of frequency, you can specify input power levels of interest using `opCon` and `opVal` arguments.
- When plotting parameters as a function of an operating condition, you can specify both frequency and input power values using `opCon` and `opVal` arguments.

Enter the `getop(h)` command at the command line to get the operating conditions for the RF circuit object `h`.

`opVal` — Value of operating conditions
scalar

Value of the operating conditions specified using the `opCon` argument, specified as a scalar.

Example: `semilogx(h,'Pout','Pin','mW','bias',1.5)` plots the data using a logarithmic scale for the *x*-axis for circuit object, `h`, with `opCon` set to `'bias'` and `value` set to `1.5`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `semilogx(h,'Pout','Pin','mW','bias',1.5,'Freq',2.4)`

`Freq` — Frequency value
positive scalar

Frequency value used to plot the data using a logarithmic scale for the *x*-axis, specified as the comma-separated pair consisting of `'Freq'` and a positive scalar in Hz.

`Pin` — Input power level
scalar

Input power level used to plot the data using a logarithmic scale for the *x*-axis, specified as the comma-separated pair consisting of `'Pin'` and a scalar in dBm.

Output Arguments

`lineseries` — Lineseries object
column vector of object handles

Lineseries object, returned as a column vector of object handles.

Tips

- Use the Property Editor (`propertyeditor`) or the MATLAB `set` function to change Chart Line.

Note Use the MATLAB `semilogx` function to create a semi-log scale plot of network parameters that are specified as vector data and are not part of a circuit (`rfckt`) object or data (`rfdata`) object.

Version History

Introduced in R2007a

See Also

`smithplot` | `analyze` | `extract` | `listformat` | `listparam` | `loglog` | `plot` | `plotyy` | `polar` | `smith` | `semilogy` | `read`

pwlresp

Calculate time response of piecewise linear input signal

Syntax

```
[tran,t] = pwlresp(h,signalTime,signalValue,tsim)
[tran,t] = pwlresp(h,signalTime,signalValue,tsim,tper)
[tran,t] = pwlresp(h,signalTime,signalValue,tsim,tper,flag)
```

Description

`[tran,t] = pwlresp(h,signalTime,signalValue,tsim)` computes the time response, `tran`, and time vector, `t`, of a piecewise linear input signal. The time response is calculated for a `rational` or `rfmodel.rational` object, `h`, using the signal parameters, `signalTime` and `signalVlaue`, over the simulation time, `tsim`.

`[tran,t] = pwlresp(h,signalTime,signalValue,tsim,tper)` computes the time response for the period of the input signal, `tper`.

`[tran,t] = pwlresp(h,signalTime,signalValue,tsim,tper,flag)` computes the time response across a set of time points that speed up the time response computation.

Note Use the `flag` input argument only for periodic signals. When you use the `flag` input, the first element of `tsim` vector should be `0`.

Examples

Calculate Time Response of Piecewise Linear Input Signal

Calculate the transient output waveform when a piecewise linear input signal is applied to a system described using a `rational` object.

Perform Rational Fit

Read the specified S2P data file.

```
S = sparameters('passive.s2p');
freq = S.Frequencies;
```

Convert the S-parameters of the two-port network to a transfer function and fit to a `rational` object.

```
tf_data = s2tf(S);
h = rational(freq,tf_data);
```

Define Parameters of Periodic Input Signal

Define the parameters of a periodic input signal over its first period, `tper`.

```
signalTime = [0,0.1,0.6,0.7,1.5]*1e-9;
signalValue = [0,5,5,0,0];
tper = 1.5e-9;
```

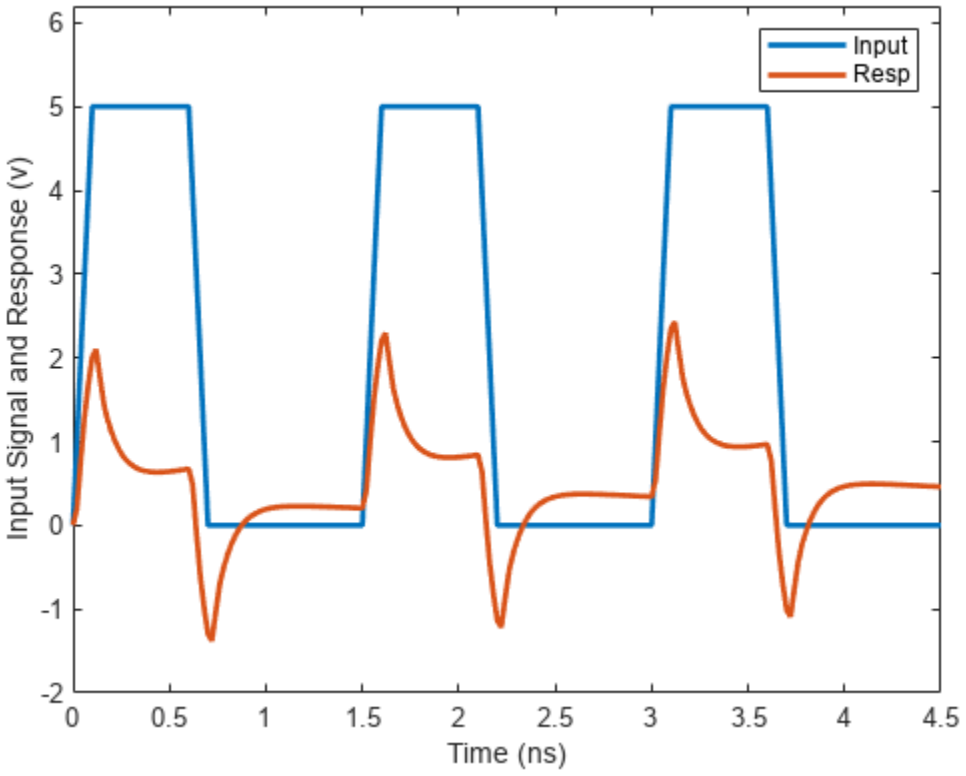
Calculate and Plot Transient Response

Calculate the transient response over three time periods for the simulation time interval, `ts`.

```
ts = 2e-11;
tsim = 0:ts:3*tper;
[tran,t] = pwlresp(h,signalTime,signalValue,tsim,tper);
```

Plot the output response and the periodic input signal.

```
vin = repmat(signalValue,1,3);
tin = [signalTime,signalTime+tper,signalTime+2*tper];
figure
plot(tin*1e9,vin,t*1e9,tran,'LineWidth',2)
axis([0 4.5 -2 6.2]);
xlabel('Time (ns)');
ylabel('Input Signal and Response (v)');
legend('Input','Resp');
```



Input Arguments

h — Rational fit object

`rfmodel.rational` object | `rational` object

Rational fit object, specified as either an `rfmodel.rational` or a `rational` object.

`signalTime` — Input signal time
vector of positive integers

Input signal time, specified as a vector of positive integers in seconds.

`signalValue` — Input signal amplitude
vector of integers

Input signal amplitude, specified as a vector of integers. The signal amplitude corresponds to the input signal time specified in `signalTime`.

Note For a periodic input signal, specify the `signalValue` over only to the first period of the signal.

`tsim` — Simulation time
vector of positive integers

Simulation time, specified as a vector of positive integers in seconds.

`tper` — Period of input signal
positive scalar

Period of the input signal, specified as a positive scalar in seconds.

`flag` — Flag to speed up time response computation
`'Rapid'`

Flag to speed up the time response computation, specified as `'Rapid'`. Use the `flag` argument for the function to rapidly compute the time response.

Note When you specify `flag` as an input argument to this function, the time vector, `t` corresponding to the output response ,`tran`, might be different from the simulation time `tsim` specified.

Output Arguments

`tran` — Output response of piecewise linear input signal
nonnegative vector

Output response of a piecewise linear input signal, returned as a nonnegative vector.

`t` — Time vector
nonnegative vector

Time vector corresponding to the output response, `tran`, returned as a nonnegative vector.

Version History

Introduced in R2021a

See Also

`rationalfit` | `freqresp` | `stepresp` | `timeresp`

Functions

rftool

(To be removed) Open RF Analysis Tool (RF Tool)

Note `rftool` is not recommended. Use the **RF Budget Analyzer** app instead. For more information, see “Compatibility Considerations”.

Syntax

```
rftool
```

Description

`rftool` opens the RF Tool interface. Use this tool to:

- Create circuit components and set their parameters.
- Analyze components over a specified frequency range and step size.
- Plot the analysis results.
- Import component objects to and export them from the MATLAB workspace.
- Save RF Tool sessions for later use.

For more information, see “The RF Design and Analysis Tool”.

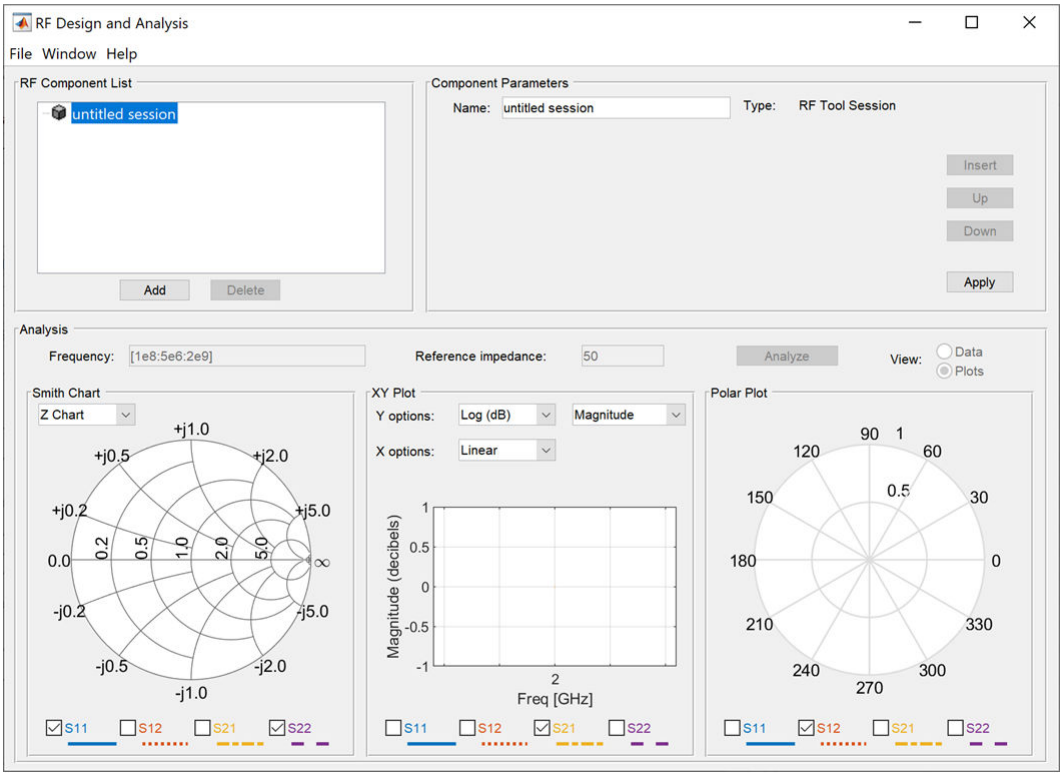
Examples

Open RF Analysis Tool

Open RF Analysis Tool using the `rftool` function.

```
rftool
```

```
Warning: rftool will be removed in a future release. Use rfBudgetAnalyzer instead.
> In rftool>rftool_OpeningFcn (line 54)
In rftool>gui_mainfcn (line 3867)
In rftool (line 44)
```



Version History

Introduced before R2006a

`rftool` function will be removed

Warns starting in R2022a

`rftool` is not recommended. Use the **RF Budget Analyzer** app instead.

To update your code, change instances of the function name `rftool` to `rfBudgetAnalyzer`.

Unlike the `rftool` function, the **RF Budget Analyzer** app supports gain, noise figure, IP2, and IP3 analysis of cascaded RF elements and export to RF Blockset.

See Also

RF Budget Analyzer

Topics

“The RF Design and Analysis Tool”

add

Add additional data to existing Smith chart

Syntax

```
add(plot,data)
add(plot,frequency,data)
```

Description

`add(plot,data)` adds data to an existing Smith chart.

`add(plot,frequency,data)` adds data to an existing Smith chart based on multiple data sets containing frequencies corresponding to columns of data matrix.

Examples

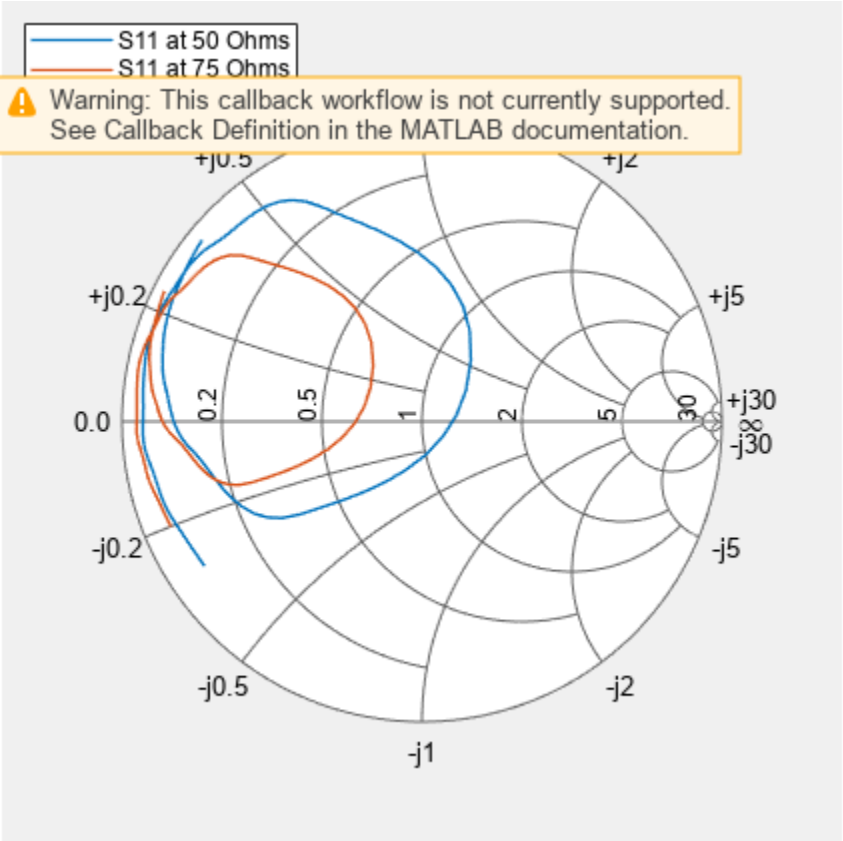
Add S-Parameter Data to Existing Smith Plot

Read S-parameter data.

```
amp = read(rfckt.amplifier,'default.s2p');
Sa = sparameters(amp);
figure
smithplot(Sa,[1,1])
```

Plot S-parameter object with new impedance of Z0 = 75 Ohms.

```
Sa = sparameters(Sa,75);
S11 = rfparam(Sa,1,1);
Freq = Sa.Frequencies;
s = smithplot('gco');
add(s, Freq, S11);
s.LegendLabels = {'S11 at 50 Ohms', 'S11 at 75 Ohms'};
```



Input Arguments

`plot` — Smith chart

function handle

Smith chart handle, specified as a function handle. If the handle of the Smith chart is not retained during creation, it is obtained by using the command `p = smithplot('gco')`.

Data Types: `double`

`data` — Input data

complex vector | complex matrix

Input data, specified as a complex vector or complex matrix.

For a matrix *D*, the columns of *D* are independent data sets. For *N*-by-*D* arrays, dimensions 2 and greater are independent data sets.

Data Types: `double`
Complex Number Support: Yes

`frequency` — Frequency data

real vector

Frequency data, specified as a real vector.

Data Types: `double`

Version History

Introduced in R2017b

See Also

`smithplot` | `replace`

replace

Remove current data and add new data to Smith chart

Syntax

```
replace(plot,data)
replace(plot,frequency,data)
```

Description

`replace(plot,data)` removes all current data from a Smith chart, `plot`, and adds new data to the Smith chart.

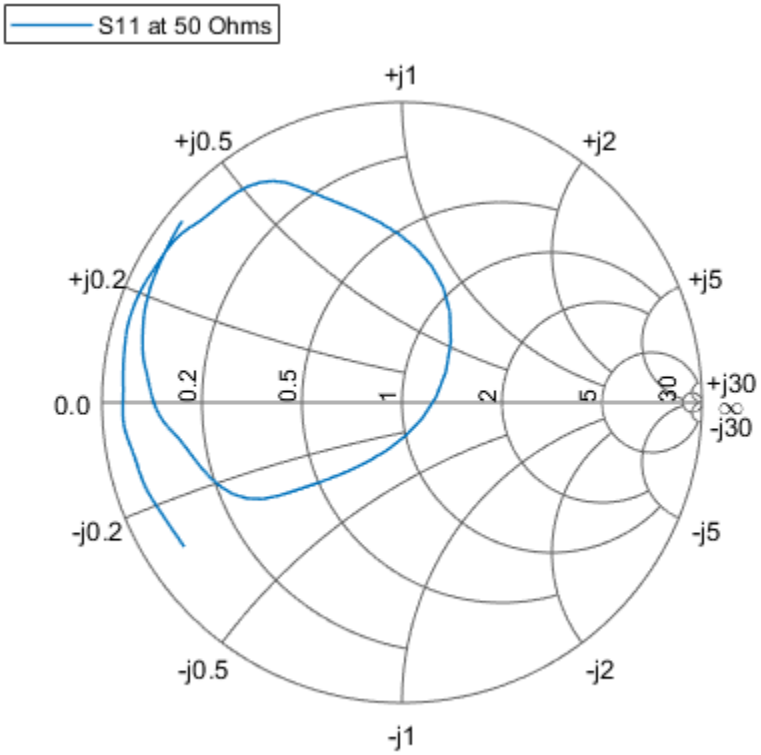
`replace(plot,frequency,data)` removes all current data and adds new data to the Smith chart based on multiple data sets containing frequencies corresponding to columns of the data matrix.

Examples

Replace S-Parameter Data on an existing Smith Plot

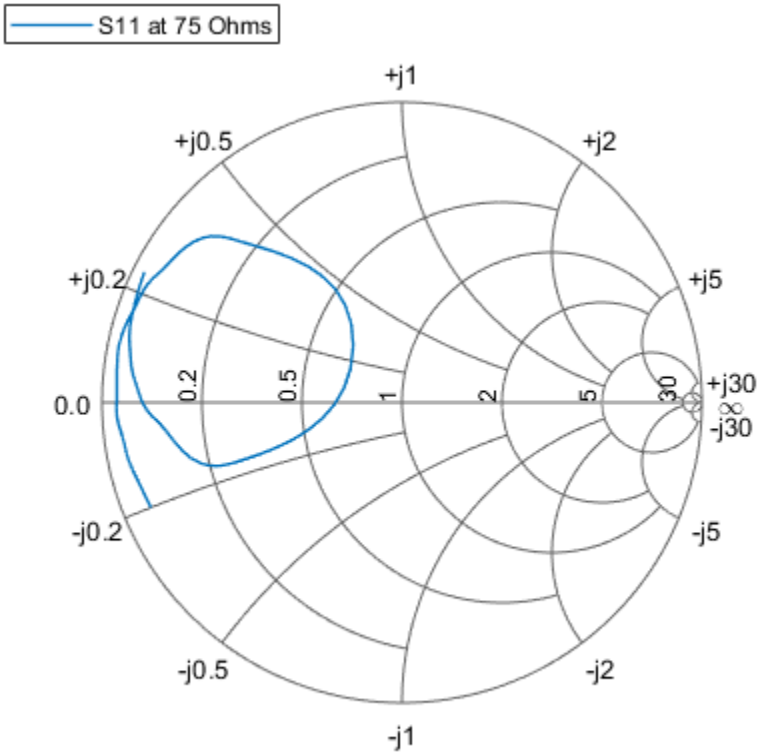
Read S-parameter data.

```
amp = read(rfckt.amplifier,'default.s2p');
Sa = sparameters(amp);
smithplot(Sa,[1,1],'LegendLabels','S11 at 50 Ohms');
```



Plot S-parameter object with a new impedance of Z0 = 75 Ohms.

```
Sa = sparameters(Sa,75);
S11 = rfparam(Sa,1,1);
Freq = Sa.Frequencies;
s = smithplot('gco');
replace(s, Freq, S11);
s.LegendLabels = 'S11 at 75 Ohms';
```



Input Arguments

`plot` — Smith plot

plot handle

Smith chart handle, specified as a plot handle. If the handle of the Smith chart is not retained during creation, use `p = smithplot('gco')`.

`data` — Input data

complex vector | complex matrix

Input data, specified as a complex vector or complex matrix.

For a matrix *D*, the columns of *D* are independent datasets. For *N*-by-*D* arrays, dimensions 2 and greater are independent datasets.

Data Types: `double`
Complex Number Support: Yes

`frequency` — Frequency data

real vector

Frequency data, specified as a real vector.

Data Types: `double`

Version History

Introduced in R2017b

See Also

`add` | `smithplot`

smithplot

Plot measurement data on Smith chart

Syntax

```
smithplot(data)
smithplot(frequency,data)
smithplot(ax, ___ )
smithplot(hnet)
smithplot(hnet,i,j)
smithplot(hnet,[i1,j1;i2,j2;....,in,jn])
smithplot(rfbudgetobj,i,i)
s = smithplot( ___ )
s = smithplot('gco')
smithplot( ___ ,Name,Value)
```

Description

`smithplot(data)` creates a Smith chart based on input data values.

Note The Smith chart is commonly used to display the relationship between a reflection coefficient, typically given as S11 or S22, and a normalized impedance.

`smithplot(frequency,data)` creates a Smith chart based on frequency and data values.

`smithplot(ax, ___ )` creates a Smith chart with a user defined axes handle, `ax`, instead of the current axes handle. Axes handles are not supported for network parameter objects. This parameter can be used with either of the two previous syntaxes.

`smithplot(hnet)` plots all the network parameter objects in `hnet`.

`smithplot(hnet,i,j)` plots the (i,j)th parameter of `hnet`. `hnet` can be a network parameter, an rfckt, an rfdata,or an nport object.

`smithplot(hnet,[i1,j1;i2,j2;....,in,jn])` plots multiple parameters ($i_1, j_1, i_2, j_2, \ldots, i_n, j_n$) of `hnet`. `hnet` can be a network parameter, an rfckt, an rfdata, or an nport object.

`smithplot(rfbudgetobj,i,i)` plots the reflection coefficient of an `rfbudget` object.

Note For rfbudget objects, smith plot is restricted to reflection coefficients.

`s = smithplot( ___ )` returns a Smith chart object handle so you can customize the plot and add measurements.

`s = smithplot('gco')` returns a Smith chart object handle of the current plot. This syntax is useful when the function handle, `p` was not returned or retained.

`smithplot(___,Name,Value)` creates a Smith chart with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding property value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Properties not specified retain their default values.

For list of properties, see SmithPlot Properties.

Note The property `'Parent'` might be used to control the location where Smith chart gets plotted. Target can be figure, UI figure, UI panel, etc.

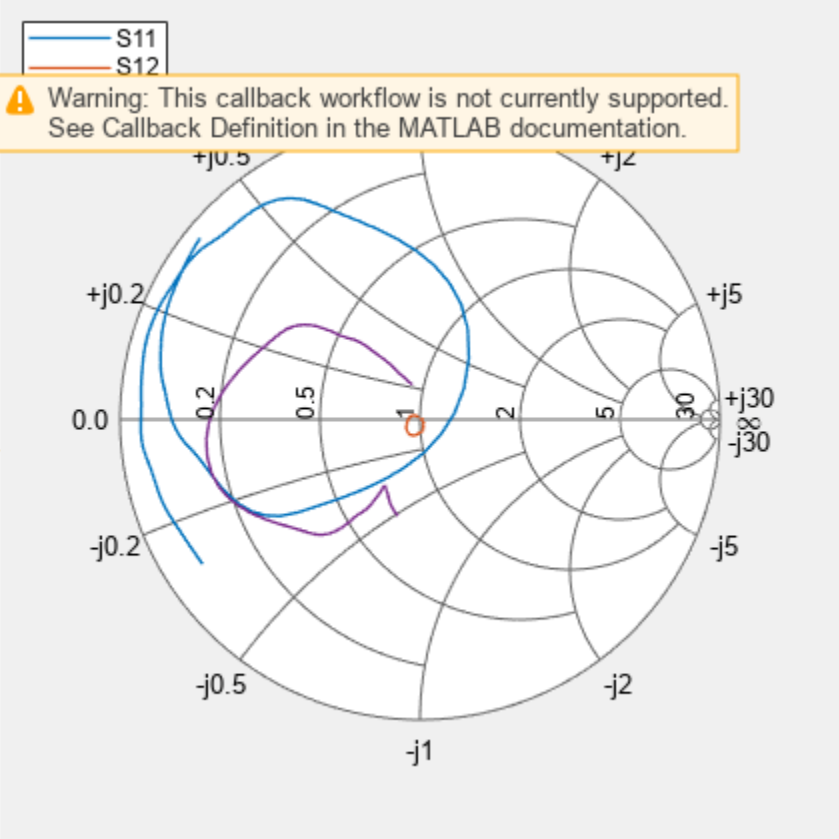
Examples

Smith Plot of S-Parameter from n-Port Circuit Object with Interactive Menu

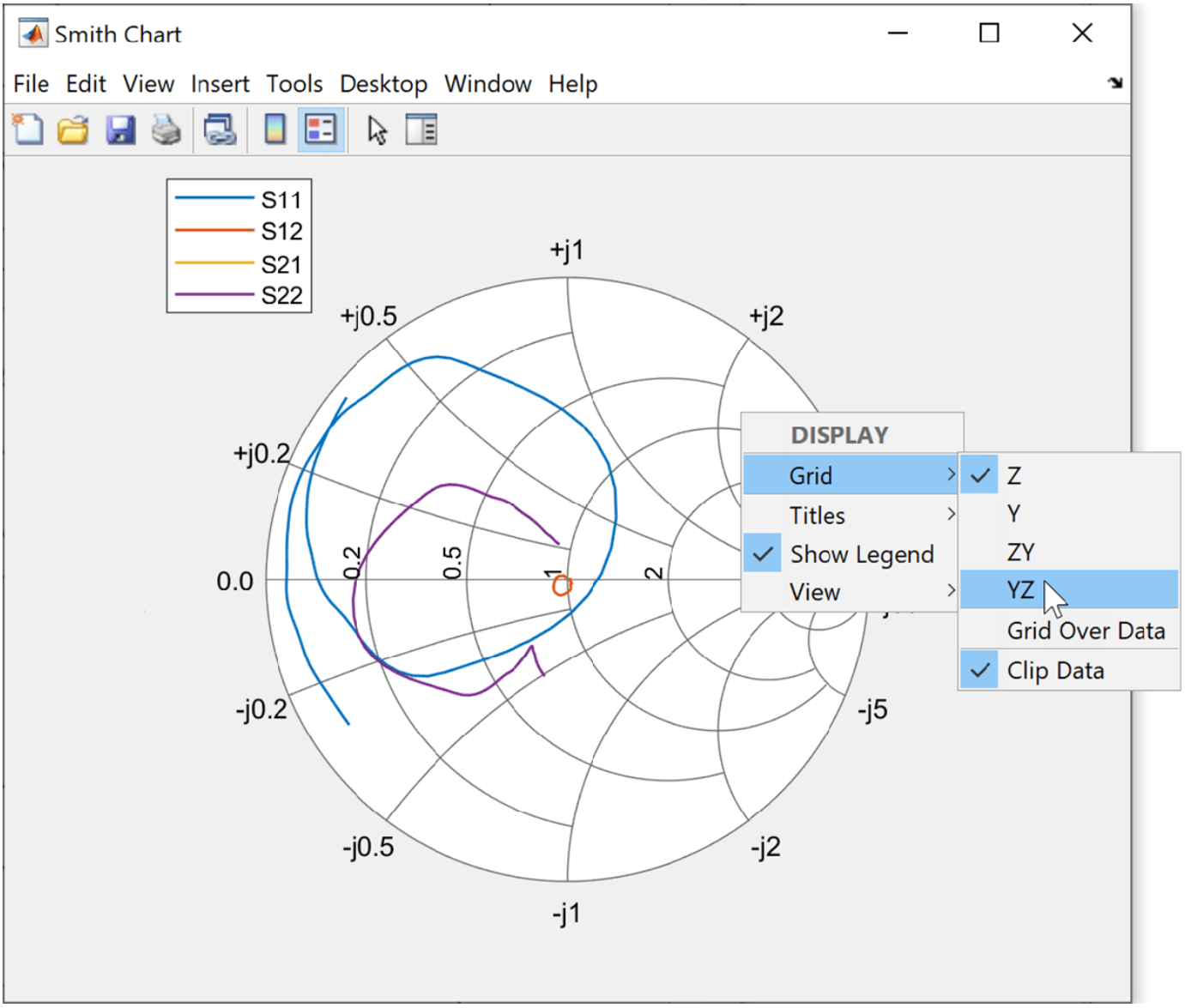
Use the Smith plot interactive menu to change grid type of the Smith plot.

Plot the Smith plot of S-parameters data file, `default.s2p`.

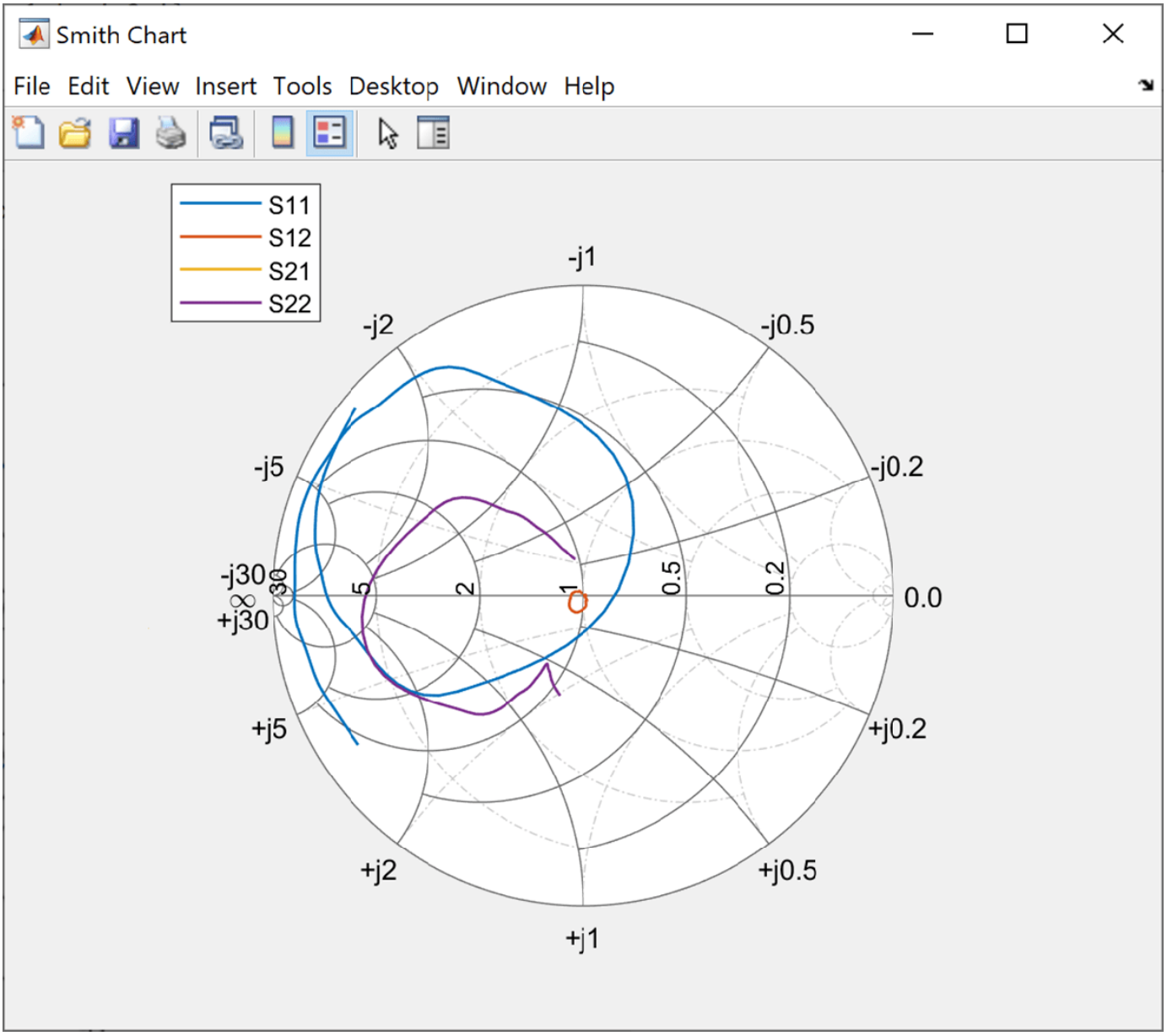
```
data = nport('default.s2p');
smithplot(data)
```



Right click on the S11 line to reveal interactive menu, **DISPLAY**. Use `Grid` to change the grid to `YZ` on the Smith plot.



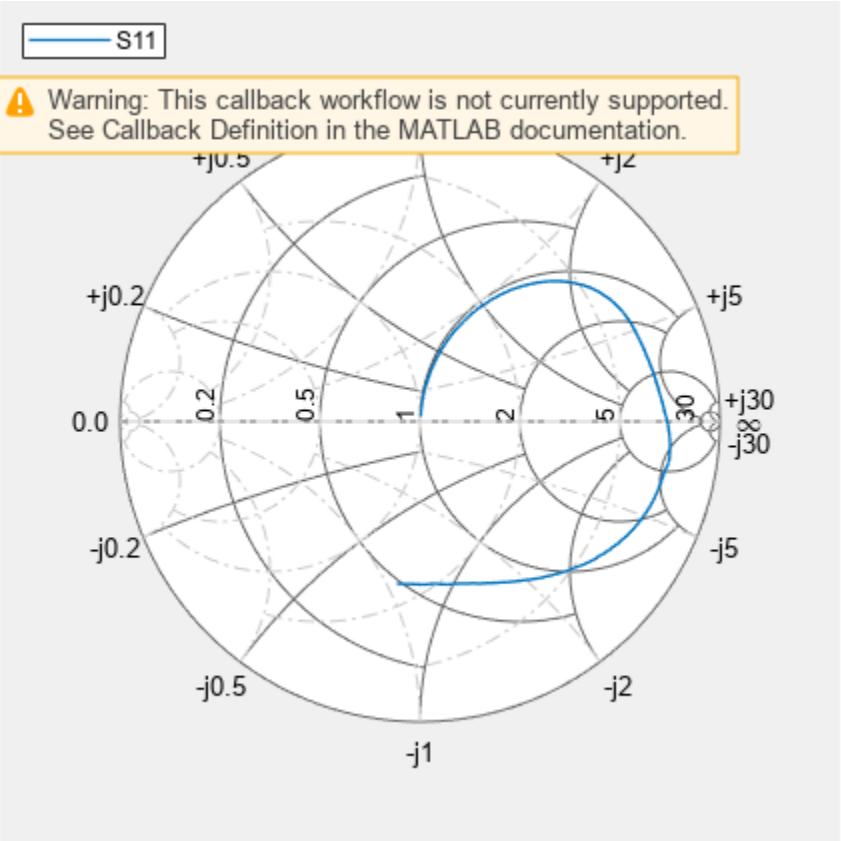
YZ- grid Smith plot is displayed below.



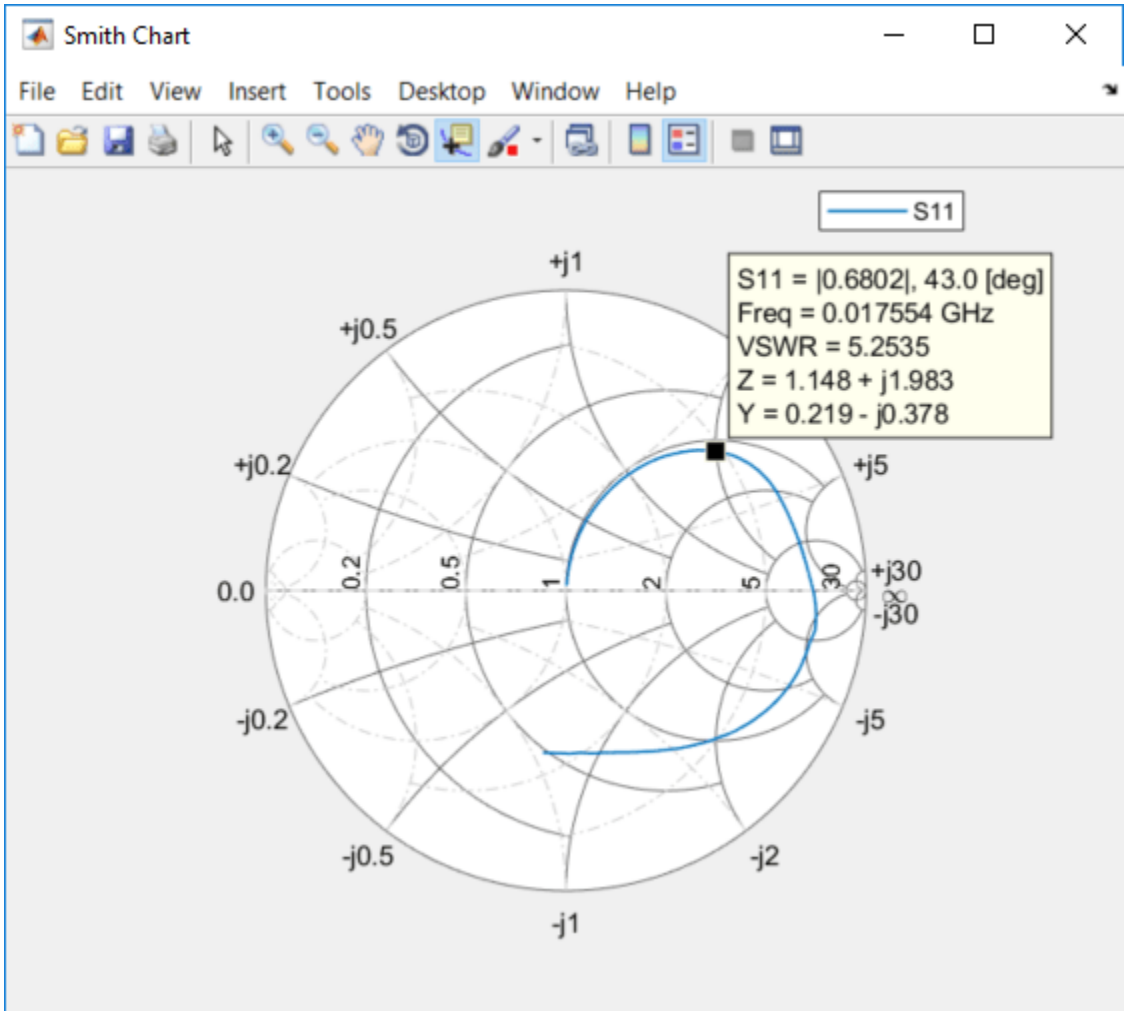
Smith Plot of (i,j)th Parameter of S-Parameter Data

Plot the Smith plot of S11 of s-parameter data file using an impedance of 75 ohms.

```
data = sparameters('passive.s2p' );
s = sparameters(data,75);
p = smithplot(s,1,1, 'GridType','ZY');
```



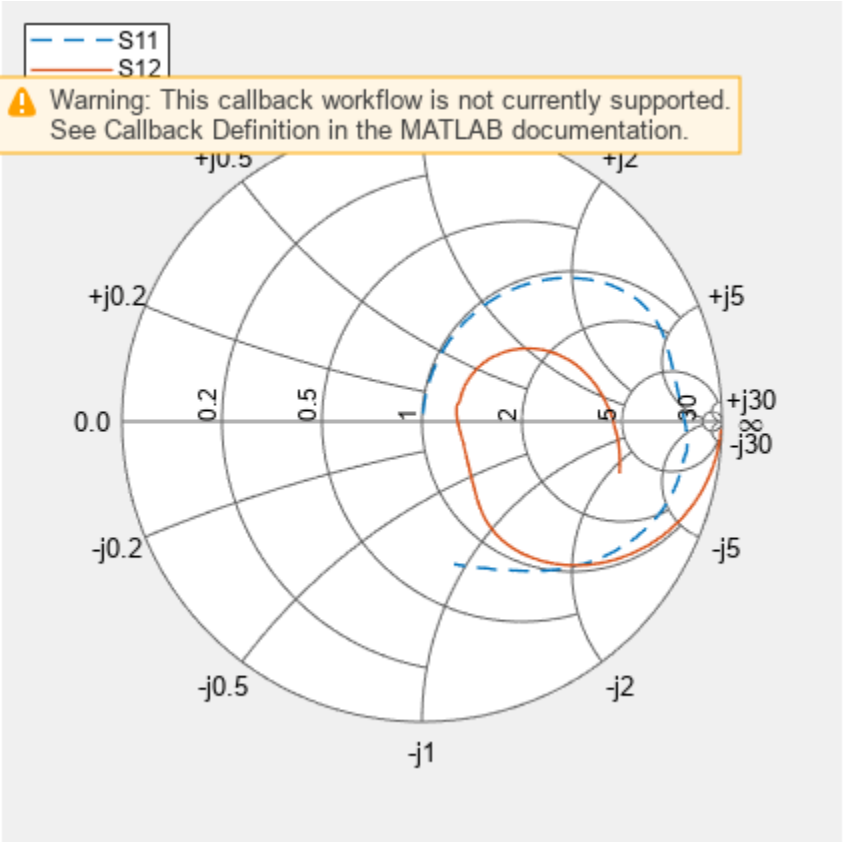
Use the data cursor icon in the toolbar to insert a cursor on your smith plot chart. You now know the S11, VSWR, Impedance, and frequency values at that cursor. For admittance value, change the Grid Type.



Smith Plot of rfckt Object

Plot the Smith chart of an rfckt.amplifier object.

```
S = read(rfckt.amplifier,'passive.s2p');
ports = [1,1;1,2];
s = smithplot(S,ports);
s.LineStyle = {'--', '-'};
```



Plot Impedance Data On Smith Plot

This example shows how to plot impedance data on a smithplot

Define impedance data

```
z1 = 0.1*50 + 1j*(0:2:50);
z2 = (0:2:50) - 0.6*50j;
```

Characteristic Impedance

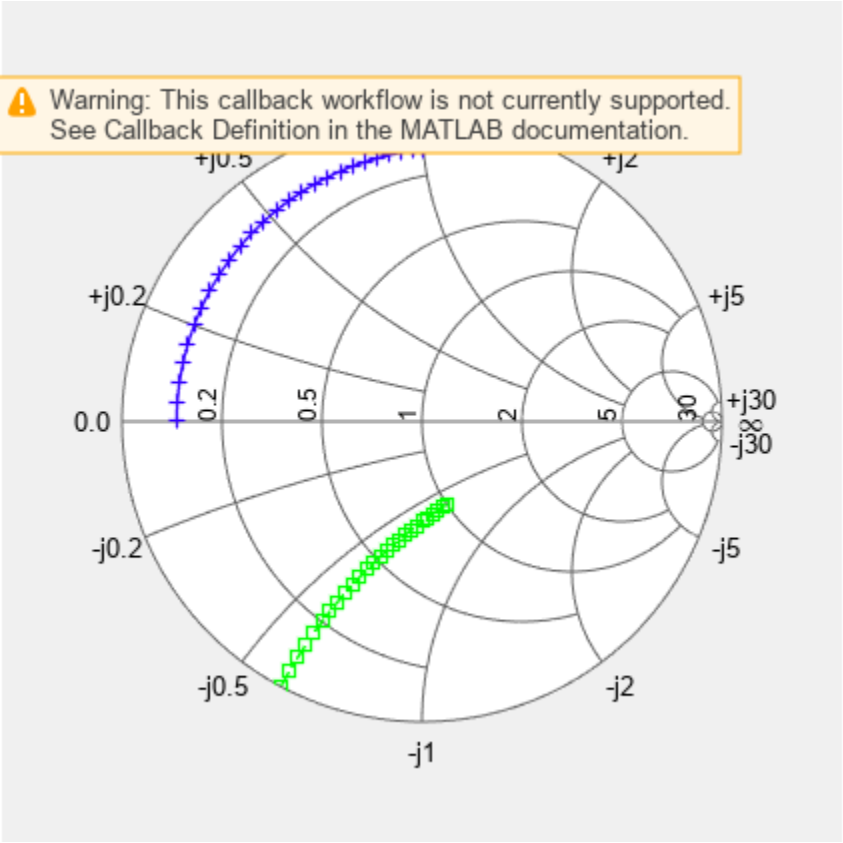
```
z0 = 50;
```

Convert impedance data to reflection coefficient

```
gamma1 = z2gamma(z1,z0);
gamma2 = z2gamma(z2,z0);
```

Plot the impedance data on the smithplot

```
s = smithplot(gamma1,'Color',[0.2 0 1],'GridType',"Z");
hold on;
s = smithplot(gamma2,'Color','g','LineStyle','-.','LineWidth',1);
s.Marker = {'+','s'}
```




```
s =
  smithplot with properties:

         Data: {[26x1 double]  [26x1 double]}
    Frequency: {[]  []}

   Show all properties, methods
```

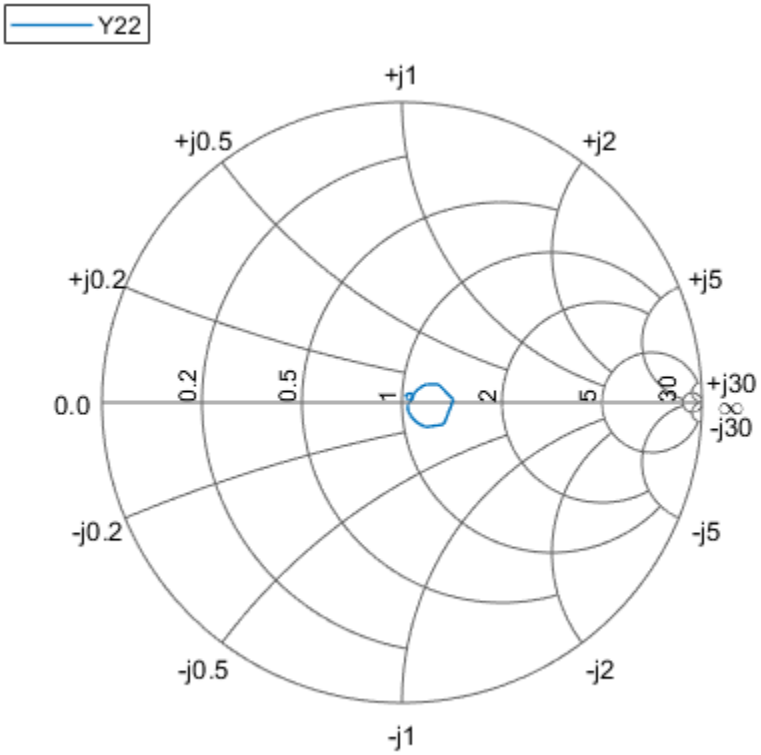
Add Z-Parameter Data to Existing Smith Plot

Read the S-parameter data.

```
amp = read(rfckt.amplifier,'default.s2p');
Sa = sparameters(amp);
```

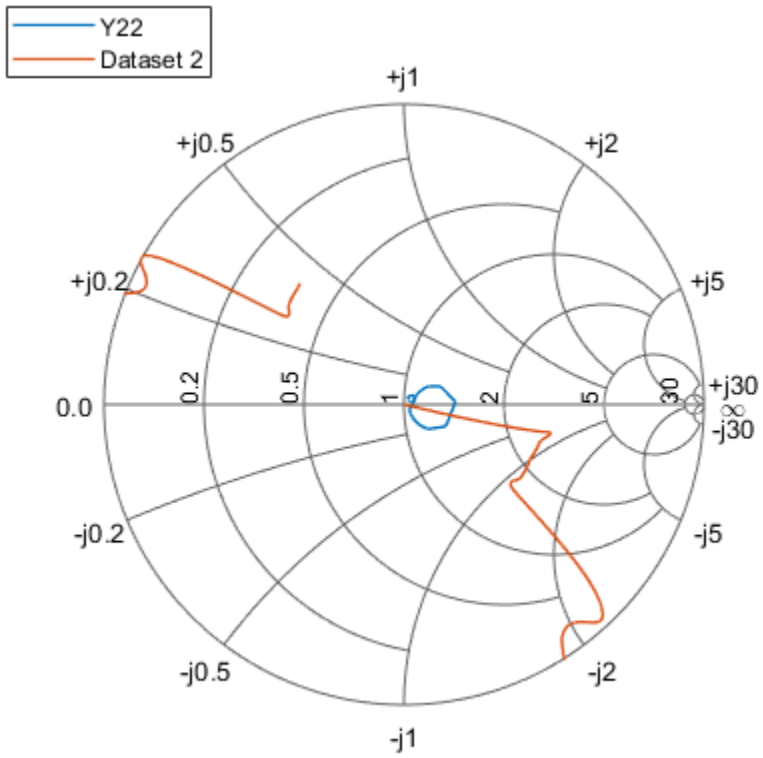
Convert the S-parameter data to Y-parameter and plot it on a Smith plot.

```
Ya = yparameters(Sa);
smithplot(Ya,2,2)
```



Add Z-parameter data to the same plot.

```
Za = zparameters(Sa);
Z12 = rfparam(Za,1,2);
Freq = Za.Frequencies;
s = smithplot('gco');
add(s, Freq, Z12);
```



Input Arguments

`data` — Input data

complex vector | complex matrix

Input data, specified as a complex vector or complex matrix.

For a matrix *D*, the columns of *D* are independent data sets. For *N*-by-*D* arrays, dimensions 2 and greater are independent data sets.

Data Types: `double`
Complex Number Support: Yes

`frequency` — Frequency data

real vector

Frequency data, specified as a real vector.

Data Types: `double`

`hnet` — Input objects

RF Toolbox network parameter object | `rfckt` object | `rfdata` object | `nport` object

Input objects, specified as one of the following:

- RF Toolbox network parameter object. For more information, see “Data Import and Network Parameters”.

- `rfckt` object. For more information, see “RF Circuit Objects”.
- `rfdata` object. For more information, see “RF Data Objects”.
- `nport` object

Data Types: `object`

`rfbudgetobj` — RF budget object
`rfbudget` object

RF budget object, specified as a `rfbudget` object.

Output Arguments

`s` — Smith chart object handle
object

Smith chart object handle. You can use the handle to customize the plot and add measurements using MATLAB commands.

Tips

- To list all the property `Name,Value` pairs in `smithplot`, use `details(s)`. You can use the properties to extract any data from the Smith chart. For example, `s = smithplot(data,'GridType','Z')` displays the impedance data grid from the Smith chart.
- For a list of properties of `smithplot`, see SmithPlot Properties.
- You can use the `smithplot` interactive menu to change the line and marker styles.

Version History

Introduced in R2017b

See Also

`add` | `replace`

Apps

RF Budget Analyzer

Analyze gain, noise figure, IP2, and IP3 of cascaded RF elements and export to RF Blockset

Description

The **RF Budget Analyzer** app analyzes the gain, noise figure, and nonlinearity of proposed RF system architecture.

Using this 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.
- Compute nonlinear effects such as output power, IP2, NF, and SNR using harmonic balance analysis.

  Note SNR calculation of an RF chain in **RF Budget Analyzer** app uses 290K as a reference temperature.
- Plot `rfbudget` results across bandwidths and over stages.
- Plot S-parameters of the RF System on a Smith chart and a polar plot.
- Plot magnitude, phase and real, and imaginary parts of S-parameters of the RF System and over stages.
- Export per-stage and cascade values to the MATLAB workspace.
- Export the system design to RF Blockset for simulation.

  Note If you use a stripline element in your system, then the app does not support exporting your system to RF Blockset.
- Export the system design to the RF Blockset Testbench as a device under test (DUT) subsystem and verify the results using simulation.

  Note If you use an antenna element, the app does not support exporting to a testbench in the RF Blockset using the **Measurement Testbench** option.
- Export the system design to `rfsystem` system object.
- Visualize budget results and S-parameters over stages and frequencies.
- Compare Friis and harmonic balance budget results.

Available Blocks

The app toolstrip contains these nonlinear elements that you can use to create an RF system:

- Amplifier
- Modulator
- Demodulator
- Generic

- Mixer IMT

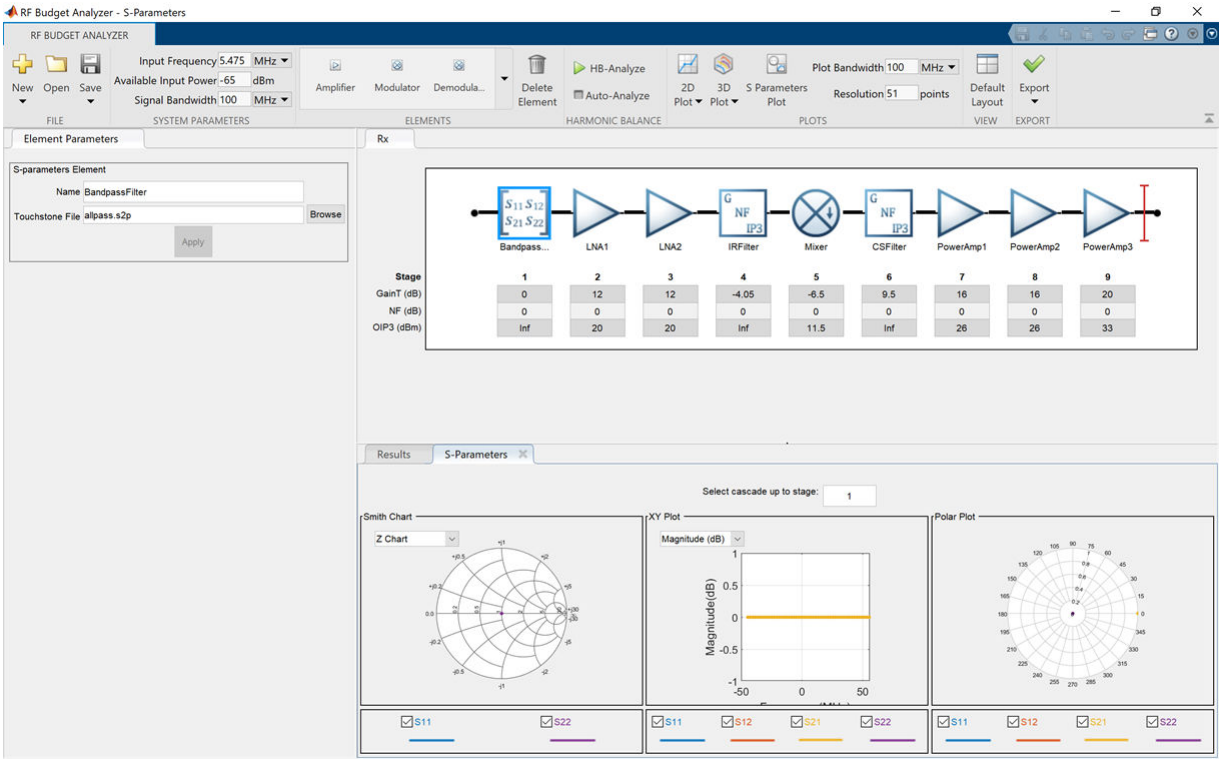
The app toolstrip contains these linear elements that you can use to create an RF system:

- S-Parameters
- Filter
- Transmission Line
- Series RLC
- Shunt RLC
- Attenuator
- Transmitter Antenna
- Receiver Antenna
- Transmit-Receive Antenna
- LC Ladder
- Phase Shift

Available Templates

The app toolstrip contains these templates that you can use to design a transmitter or a receiver system:

- Receiver
- Transmitter



Open the RF Budget Analyzer App

- MATLAB Toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, click the app icon.
- MATLAB command prompt: Enter `rfBudgetAnalyzer`.

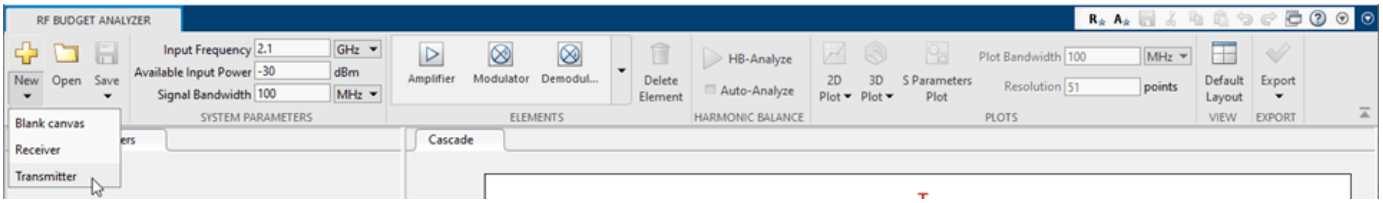
Examples

RF Transmitter System Analysis

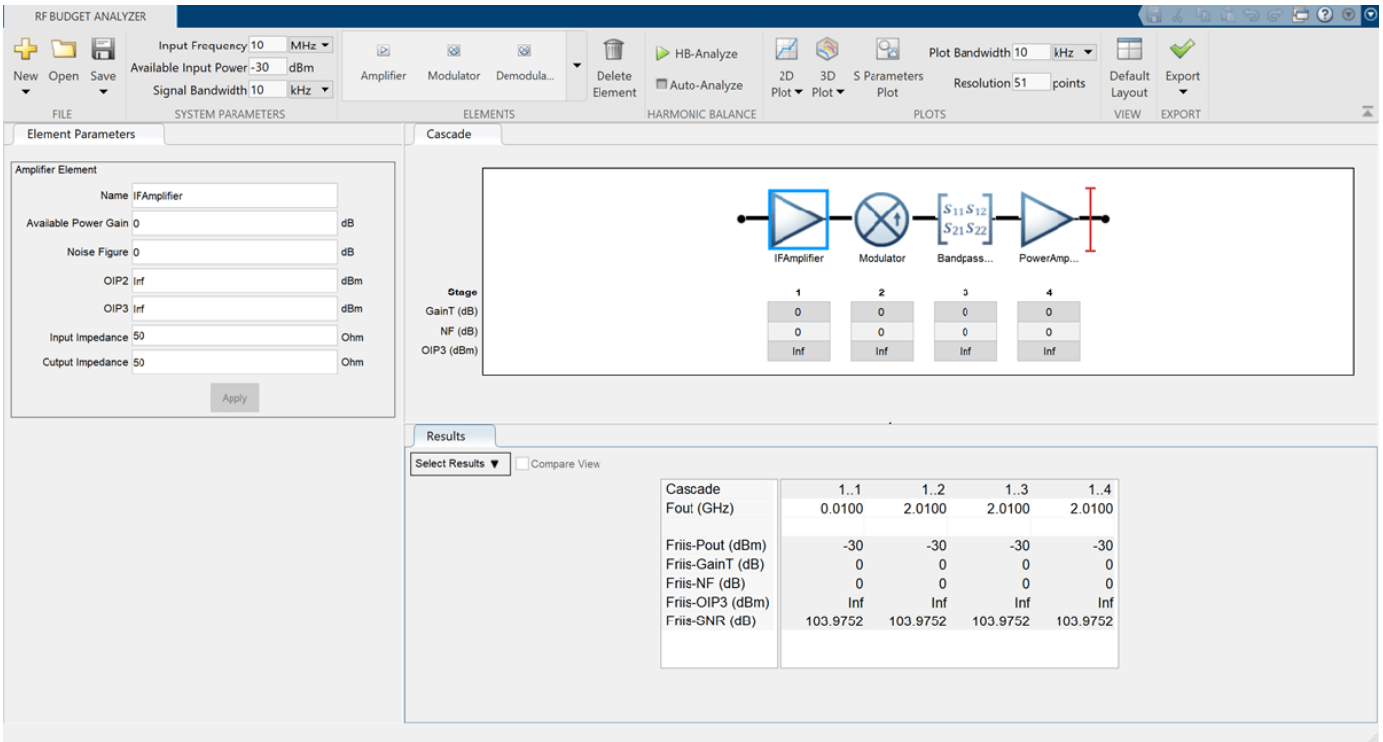
Design and analyze an RF transmitter using the **RF Budget Analyzer** app.

Enter `rfBudgetAnalyzer` to open the app.

Use the **Transmitter** template to create a basic transmitter.

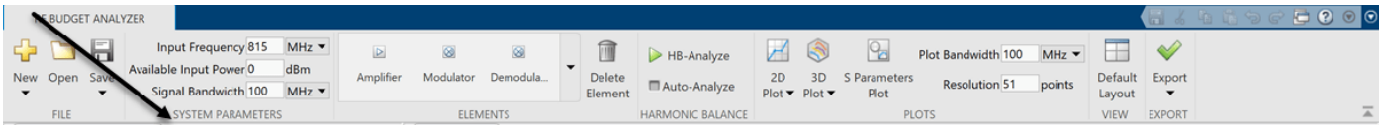


The transmitter template is displayed as follows.



In **System Parameters**, specify the RF transmitter requirements:

- **Input Frequency** — `815` MHz
- **Available Input Power** — `0` dBm
- **Signal Bandwidth** — `100` MHz



Click the `IFAmplifier` in the design canvas and delete the element using the **Delete Element** button on the toolstrip.

Add a **Generic** element in the place of the `IFAmplifier` using the toolstrip. In the **Element Parameters** pane, specify:

- **Name** — `IFFilter`
- **Available Power Gain** — `-3.6` dB
- Select **Apply**.

Click the **Modulator** element. In the **Element Parameters** pane, specify:

- **Name** — `Mixer`
- **Available Power Gain** — `-6.5` dB
- **OIP3** — `11.5` dBm
- **LO Frequency** — `4.97` GHz
- **Converter Type** — `Up`
- Select **Apply**.

Delete the **S-Parameters** element named `BandpassFilter`. Add a **Generic** element. In the **Element Parameters** pane, specify:

- **Name** — `RFFilter1`
- **Available Power Gain** — `-1.4` dB
- Select **Apply**.

Select the `PowerAmplifier` element and in the **Element Parameters** pane, specify:

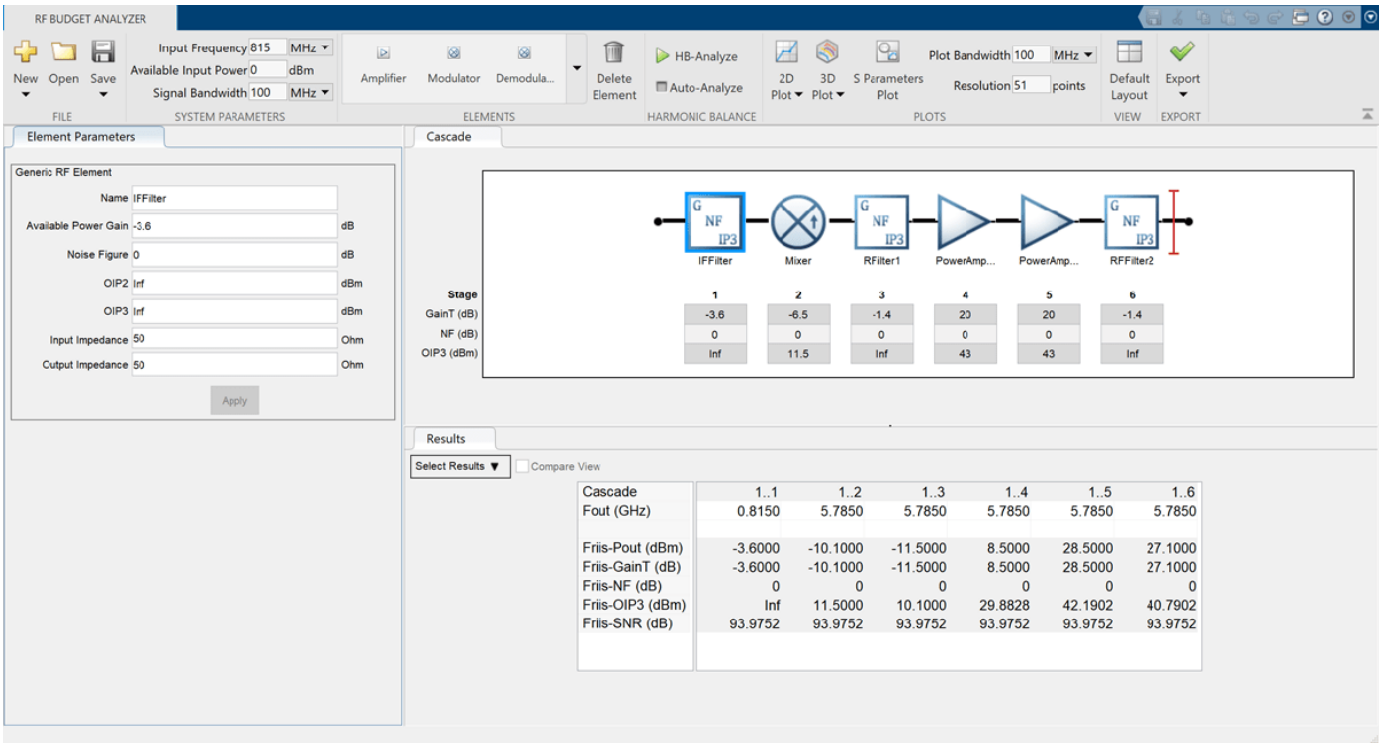
- **Name** — `PowerAmplifier1`
- **Available Power Gain** — `20` dB
- **OIP3** — `43` dBm
- Select **Apply**.

Add another **Amplifier** element using the toolstrip. In the **Element Parameters** pane, specify:

- **Name** — `PowerAmplifier2`
- **Available Power Gain** — `20` dB
- **OIP3** — `43` dBm
- Select **Apply**.

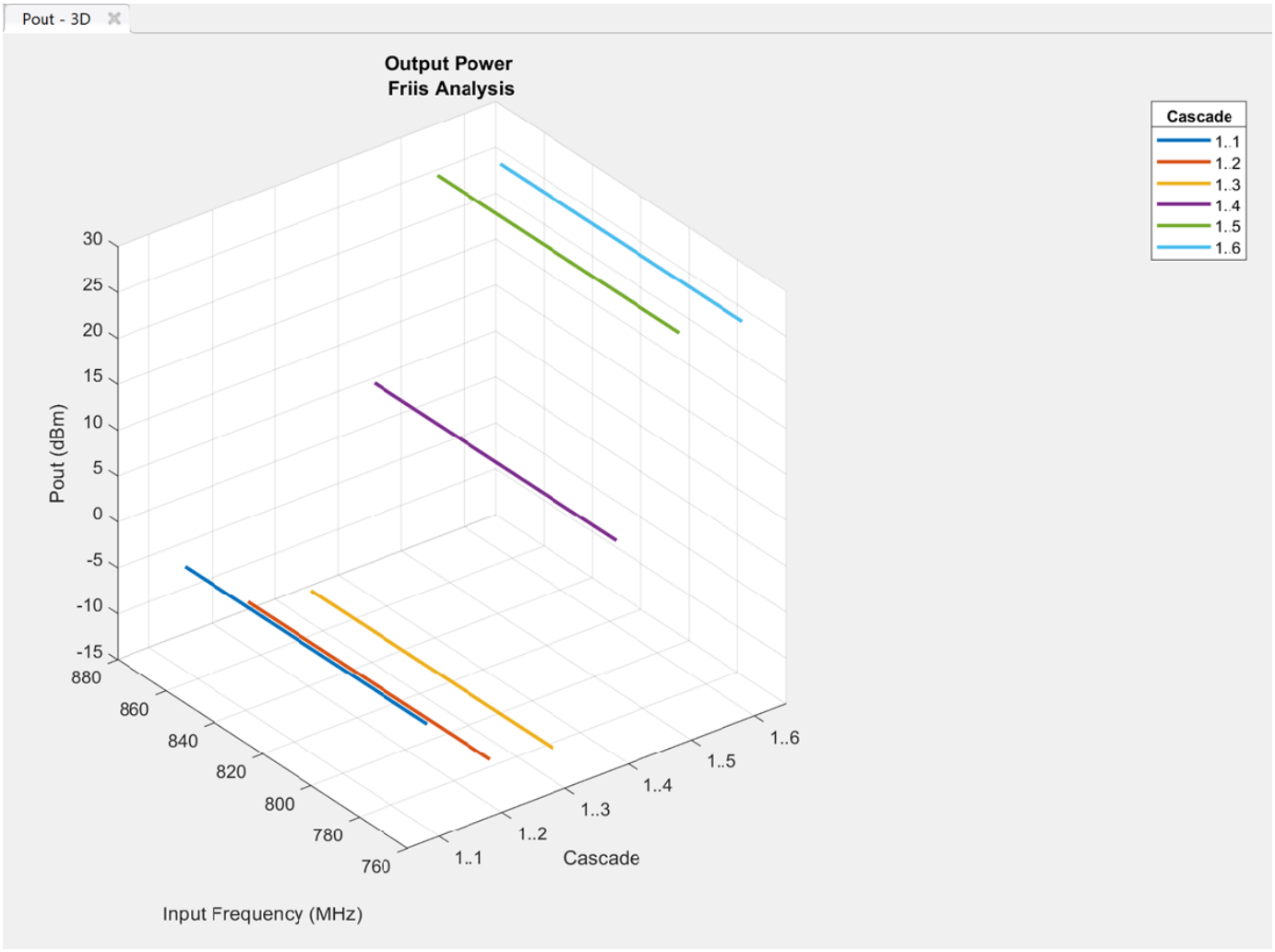
Add another **Generic** element. In the **Element Parameters** pane, specify:

- **Name** — `RFFilter2`
- **Available Power Gain** — `-1.4` dB
- Select **Apply**.



Save the system. The app saves the system in a MAT file.

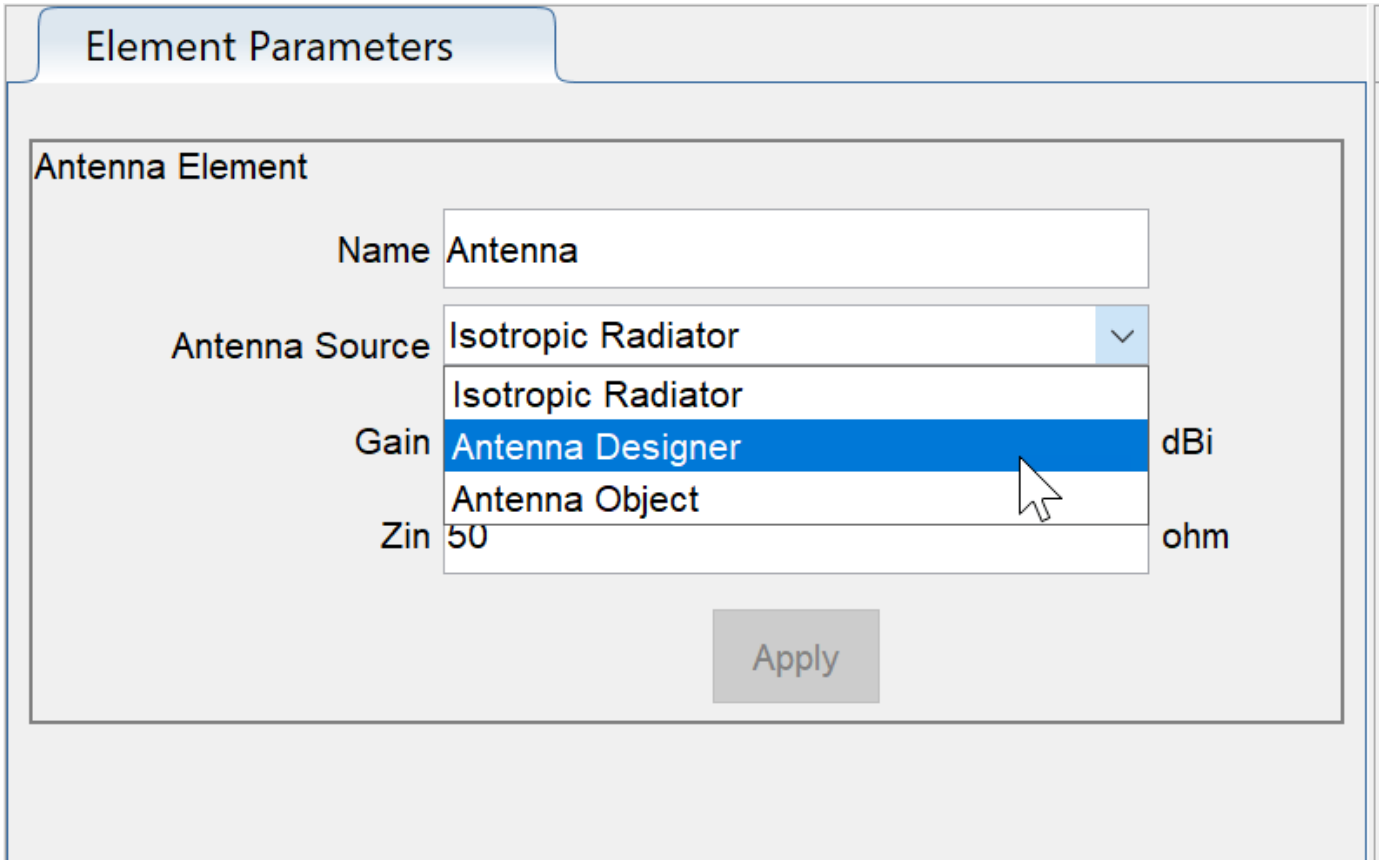
Plot the output power of the transmitter using the **3D Plot** button. Select **3D Plot** and choose `Output Power - Pout`.



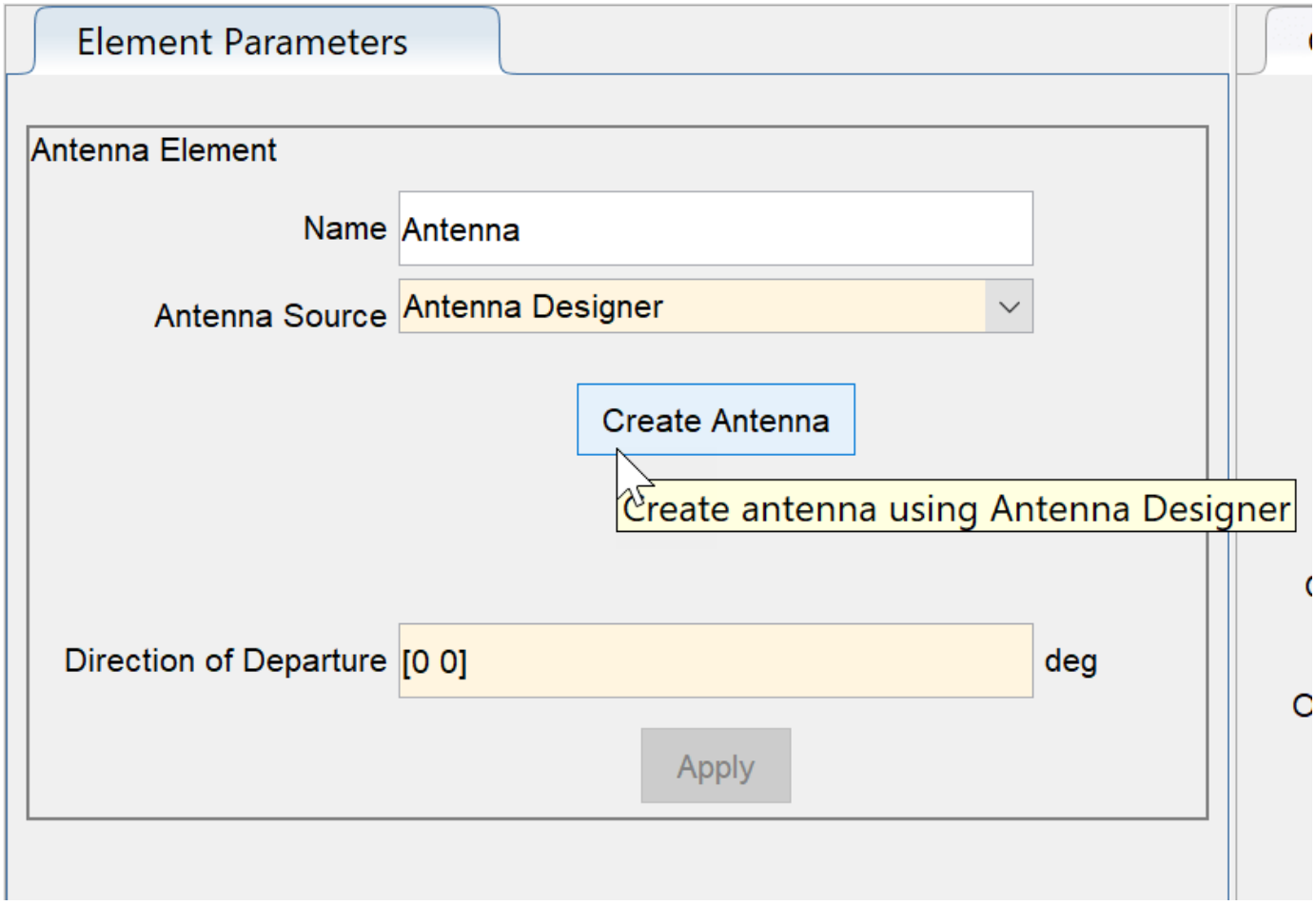
Use Transmit Antenna in your RF Transmitter Design

This example uses an RF transmitter design from the “RF Transmitter System Analysis” on page 5-4 example.

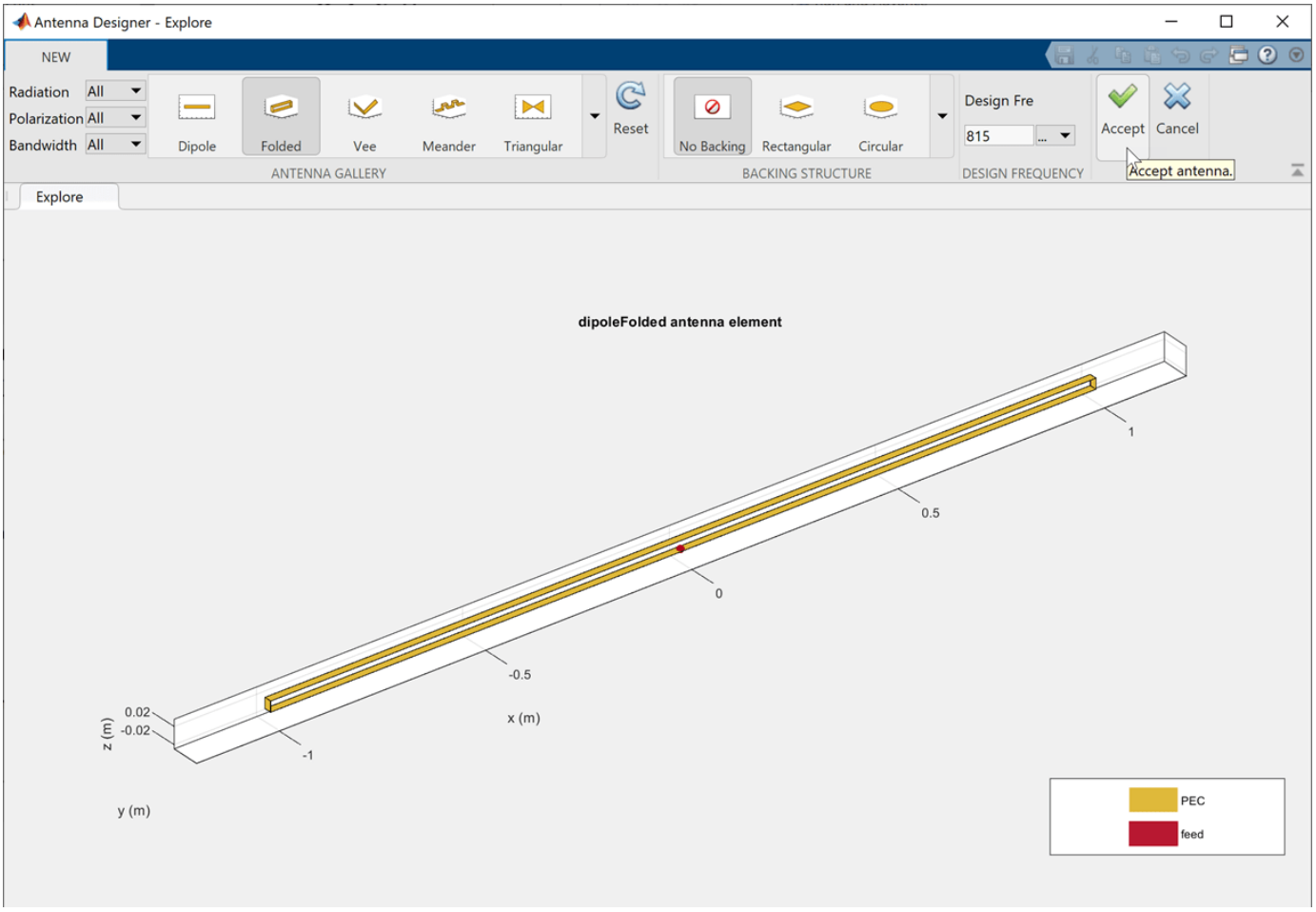
Follow the RF Transmitter System Analysis example to design an RF transmitter. Select the antenna element from the **Elements** section and add the element at end of this RF transmitter. In the **Element Parameters** pane, select `Antenna Designer` from the **Antenna Source** drop-down list.



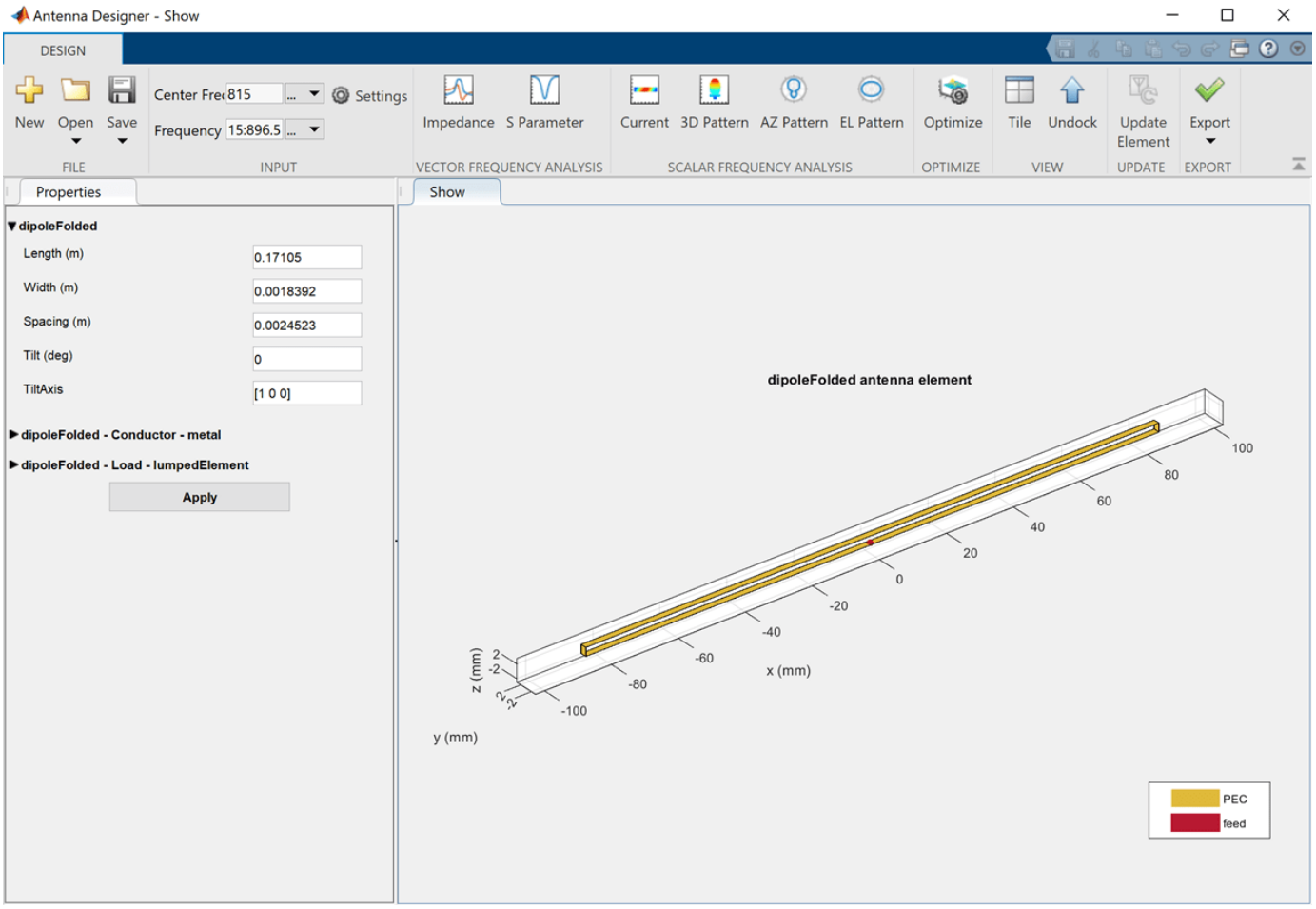
Click **Create Antenna** in the **Element Parameters** pane.



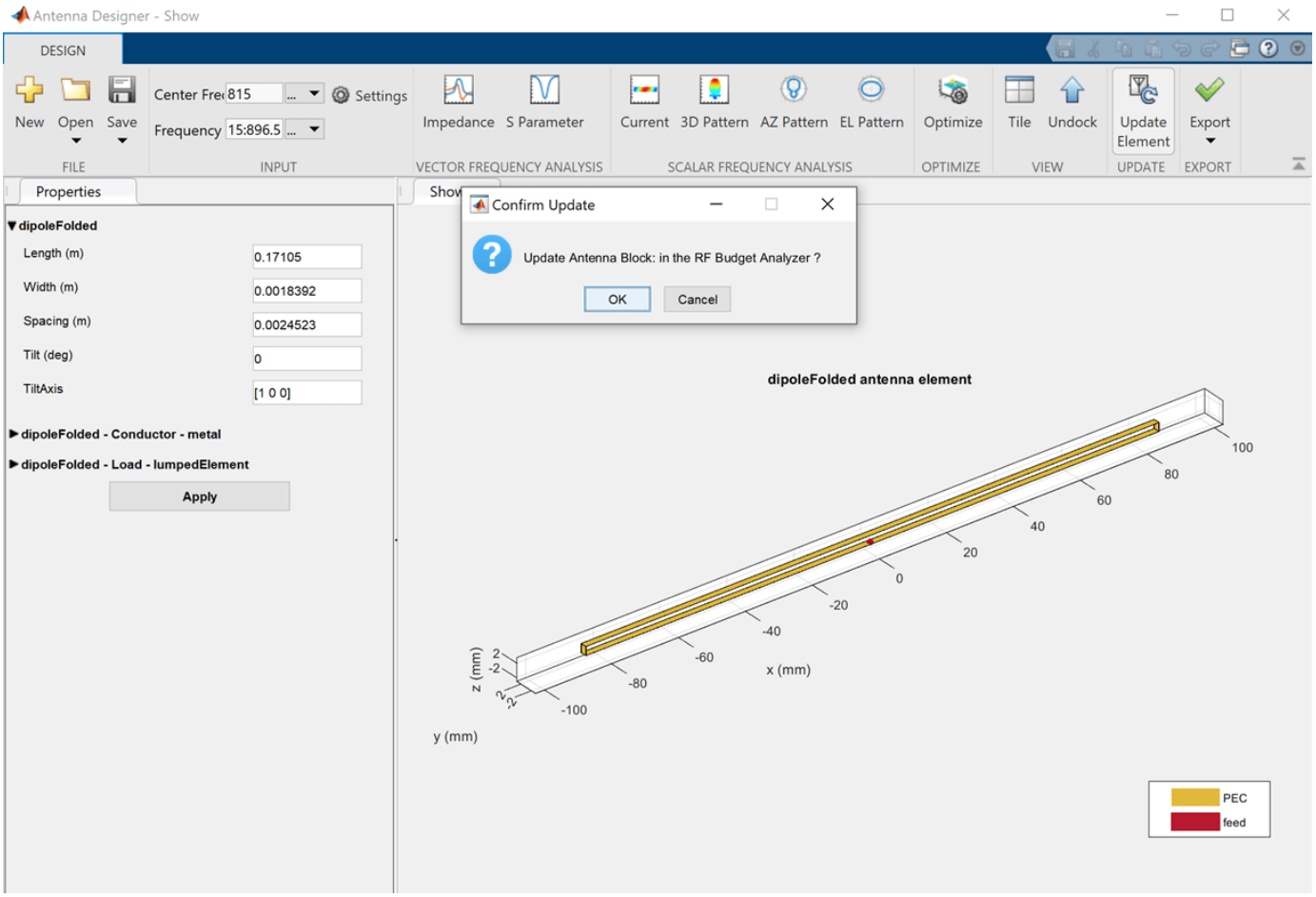
The **Antenna Designer** app opens. Click **New** to explore the antenna library. This example uses a `dipoleFolded` antenna element with a center frequency of `815` MHz. To do this, select `Folded` element from the **Antenna Gallery**, set **Design Frequency** to `815` MHz, and click **Accept**.



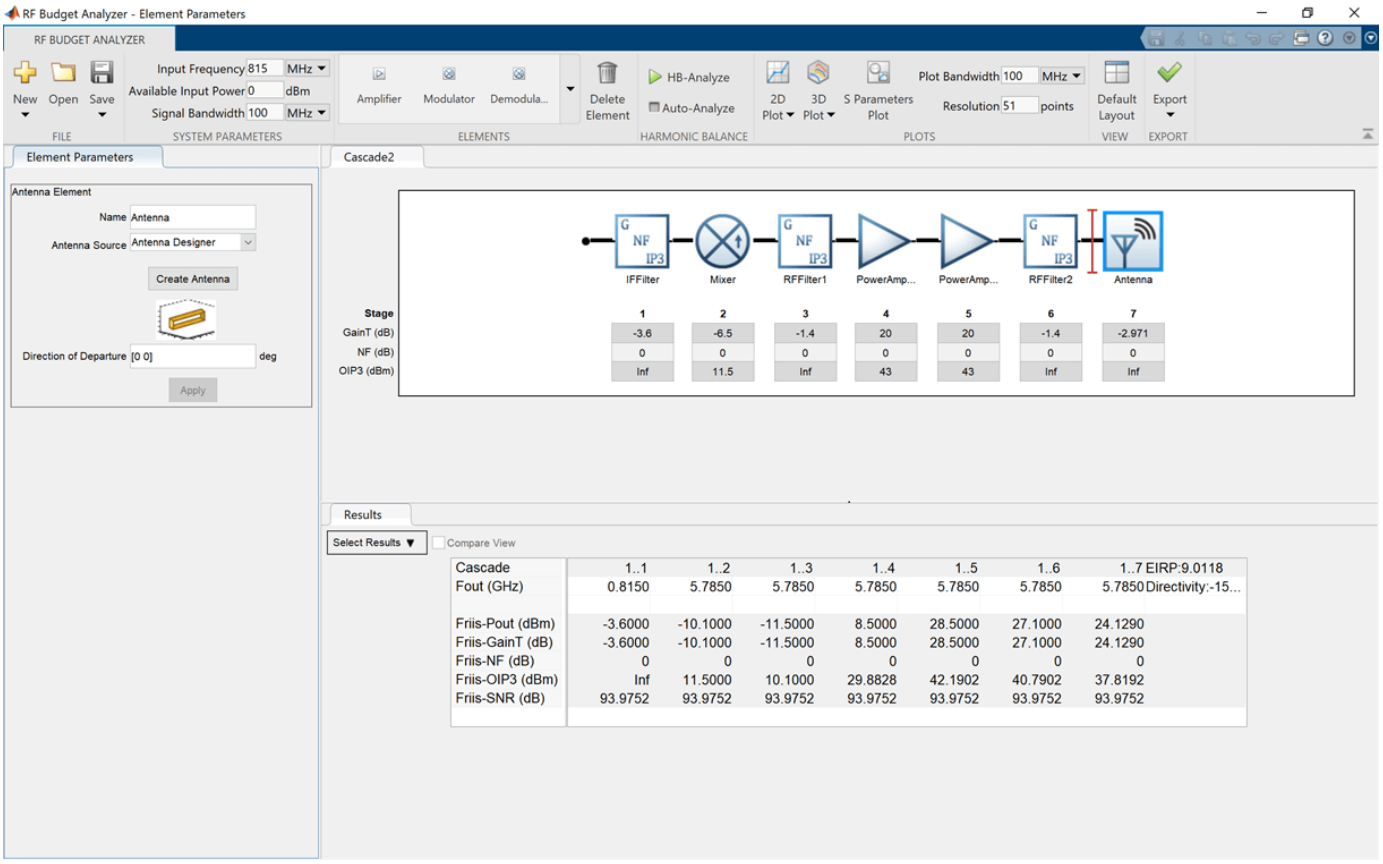
The updated antenna shows in the window.



Click **Update Element** to update the `Antenna` element in the **RF Budget Analyzer** app. Click **OK** in the Confirm Update dialog box.



The **Antenna Designer** app window closes and the `Antenna` element is updated in the **RF Budget Analyzer** app. The **Results** pane is automatically updated for Friis analysis with `EIRP` and `Directivity` of the `Antenna` element.

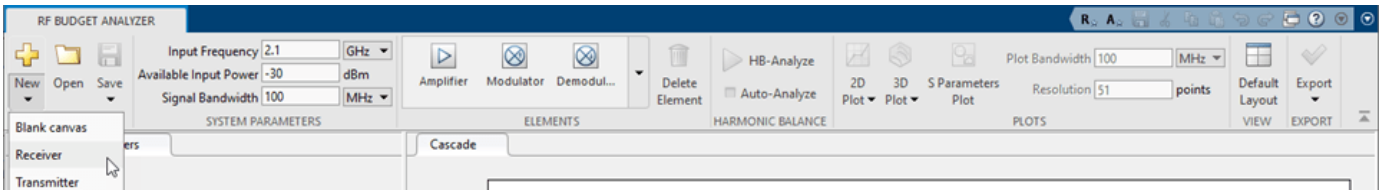


RF Receiver System Analysis

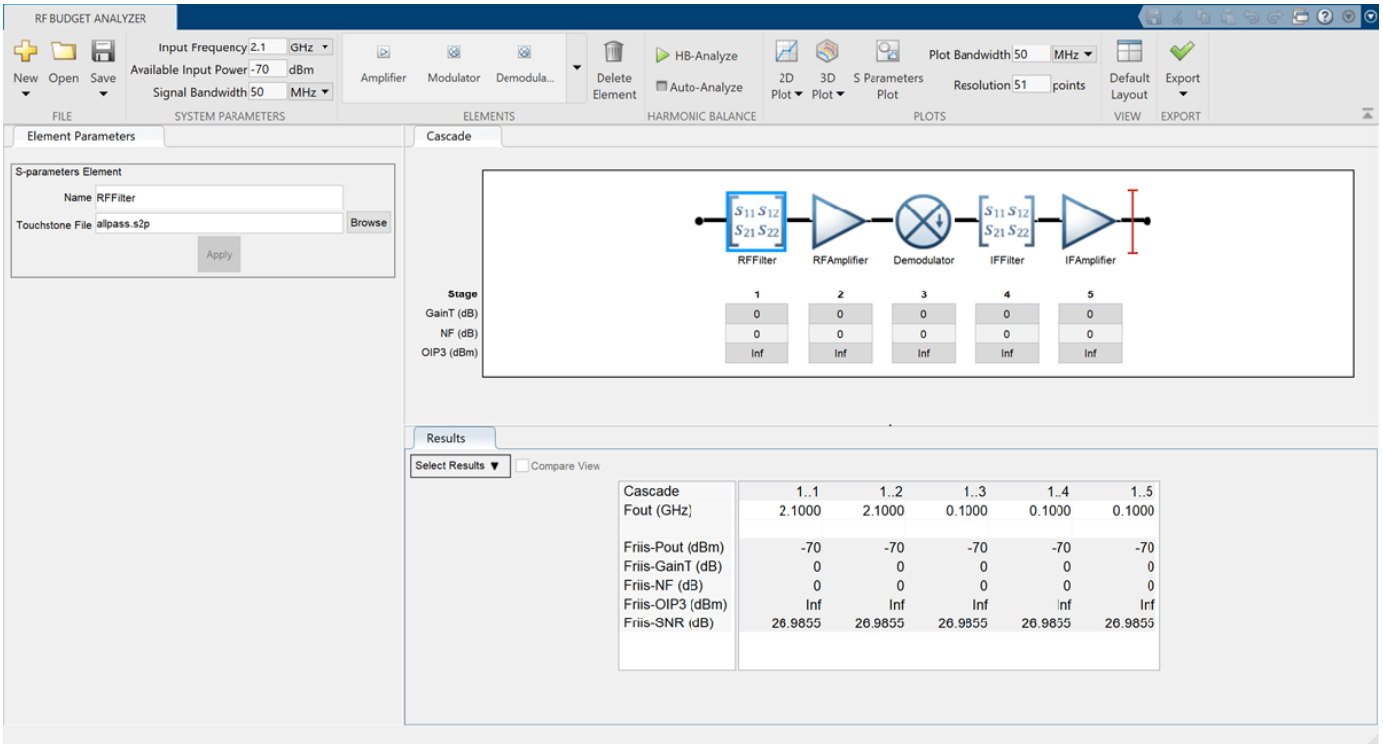
Design and analyze an RF receiver using the **RF Budget Analyzer** app.

Enter `rfBudgetAnalyzer` to open the app.

Use the **Receiver** template option to create a basic receiver.

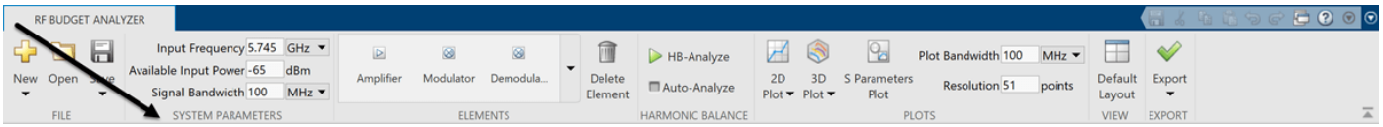


The receiver template is displayed as follows:



In **System Parameters**, specify the RF receiver requirements:

- **Input Frequency** — `5.745` MHz
- **Available Input Power** — `-65` dBm
- **Signal Bandwidth** — `100` MHz



Click `RFFilter` in the design canvas. This `RFFilter` is an **S-parameters** element. It accepts a Touchstone® file in the S2P file type. Update the **Element parameters** pane as follows:

- **Name**: `BandpassFilter`
- **S2P file:** Choose an S2P file by clicking the **Browse** button.
- Select **Apply**.

Click the `RFAmplifier` element. In the **Element Parameters** pane, specify the element requirements:

- **Name** — `LNA1`
- **Available Power Gain** — `12` dB
- **OIP3** — `20` dBm
- Select **Apply**.

Add another **Amplifier** element using the toolstrip. In the **Element Parameters** pane, specify the element requirements:

- **Name** — `LNA2`
- **Available Power Gain** — `12` dB
- **OIP3** — `20` dBm
- Select **Apply**.

Add a **Generic** element. In the **Element Parameters** pane, specify the element requirements:

- **Name** — `IRFilter`
- **Available Power Gain** — `-4.05` dB
- Select **Apply**.

Click the `Demodulator` element. In the **Element Parameters** pane, specify the element requirements:

- **Name** — `Mixer`
- **Available Power Gain** — `-6.5` dB
- **OIP3** — `11.5` dBm
- **LO Frequency** — `4.93` GHz
- **Converter Type** — `Down`
- Select **Apply**.

Delete the `IFFilter`, **S-parameters** element. Add a **Generic** element in its place. In the **Element Parameters** pane, specify the element requirements:

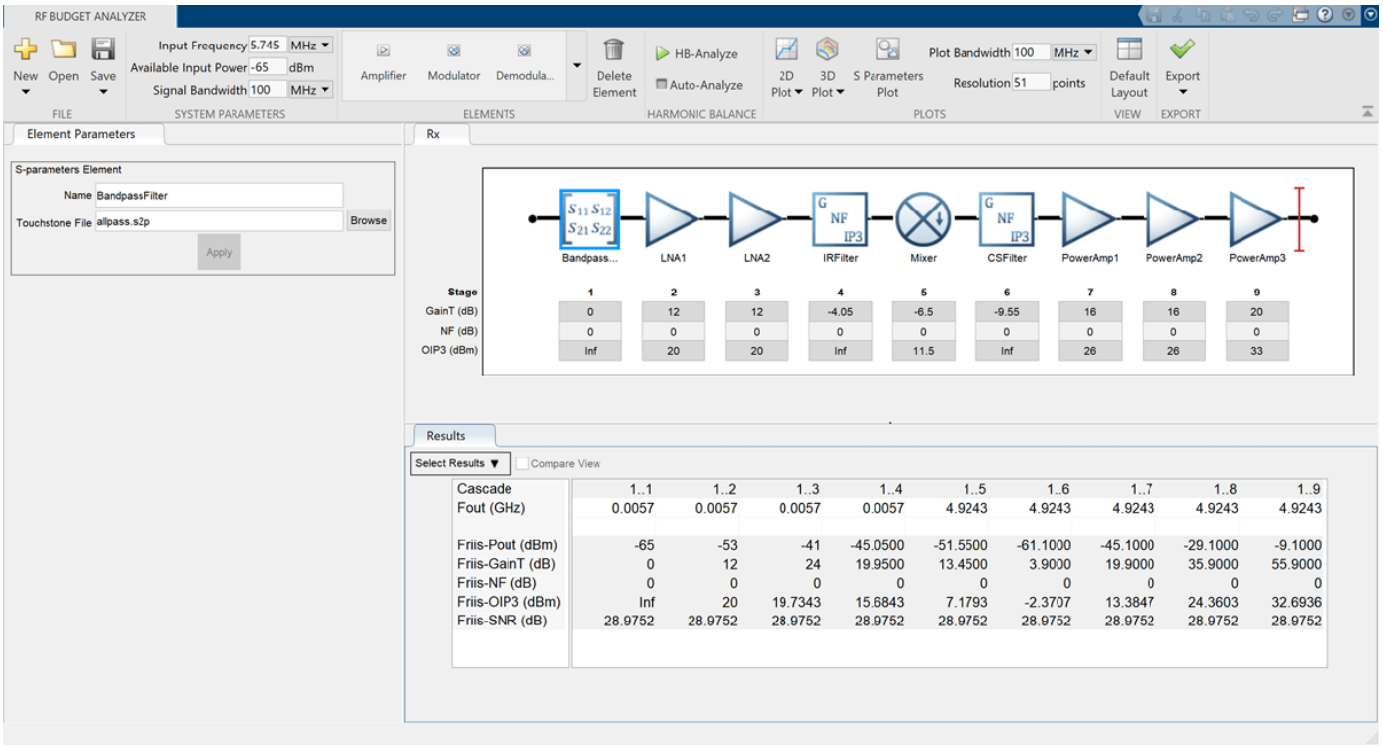
- **Name** — `CSFilter`
- **Available Power Gain** — `-9.55` dB
- Select **Apply**.

Click the `IFAmplifier` element. In the **Element Parameters** pane, specify the element requirements:

- **Name** — `PowerAmp1`
- **Available Power Gain** — `16` dB
- **OIP3** — `26` dBm
- Select **Apply**.

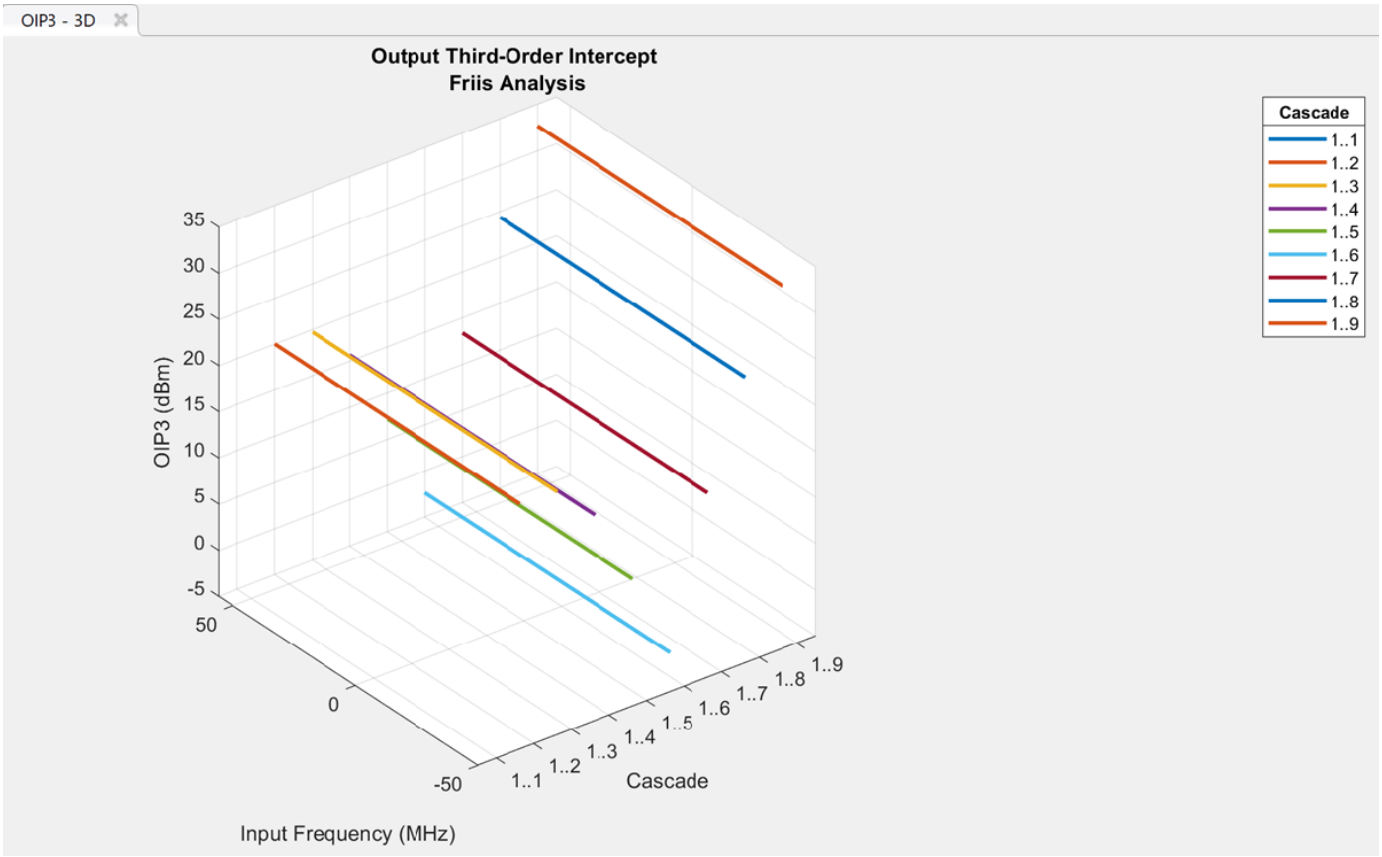
Add two more **Amplifier** elements. For each element, in the **Element Parameters** panes specify the element requirements:

- **Name** — `PowerAmp2` | `PowerAmp3`
- **Available Power Gain** — `16` dB | `20` dB
- **OIP3** — `26` dBm | `33` dBm
- Select **Apply**.



Save the system. The app saves the system in a MAT file.

Plot the output OIP3 of the receiver using the **3D Plot** button. Select the **3D Plot** button and choose `Output Third-Order Intercept Point - OIP3`.



Compare Friis and Harmonic Balance Solver

Create an amplifier with a gain of 4 dB.

```
a = amplifier('Gain',4);
```

Create a modulator with an OIP3 of 13 dBm.

```
m = modulator('OIP3',13);
```

Create an `nport` using `passive.s2p`.

```
n = nport('passive.s2p');
```

Create an RF element with a gain of 10 dB.

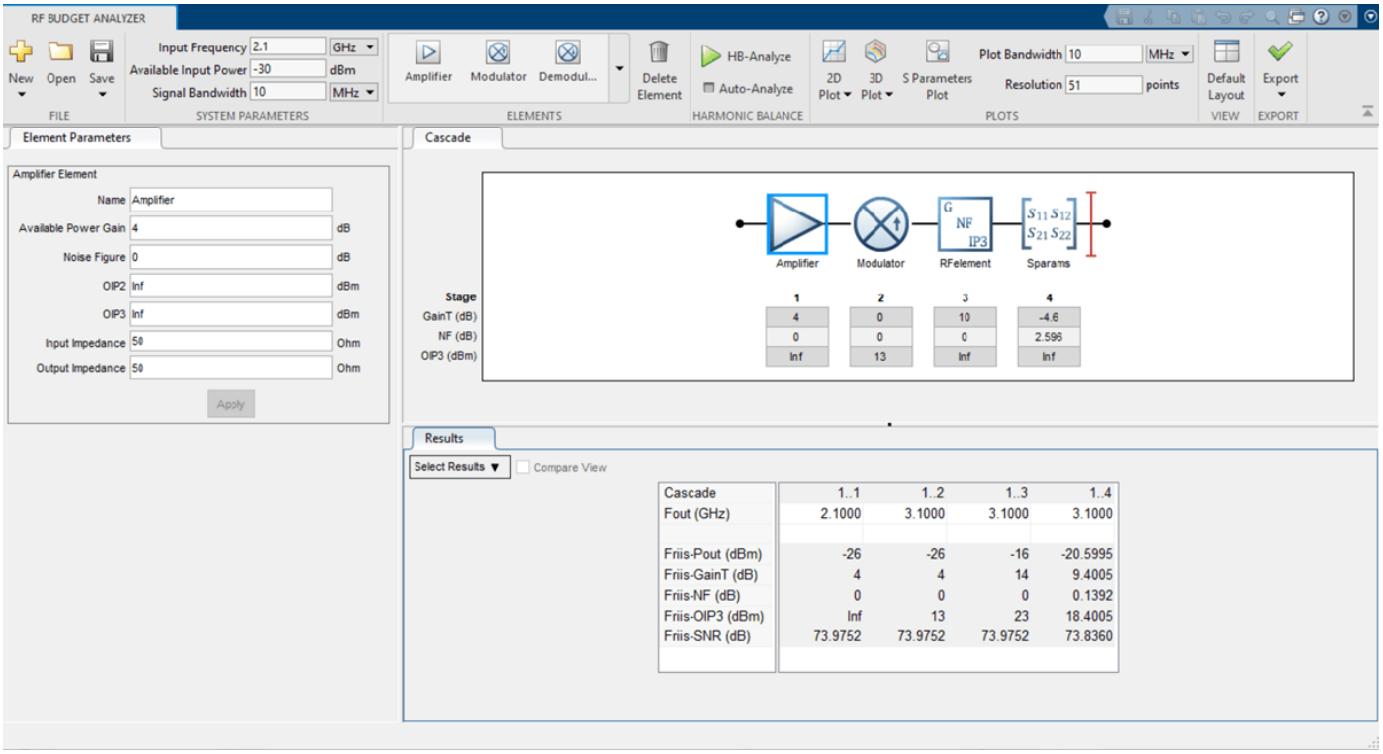
```
r = rfelement('Gain',10);
```

Calculate the `rfbudget` of a series of RF elements at an input frequency of 2.1 GHz, an available input power of -30 dBm, and a bandwidth of 10 MHz.

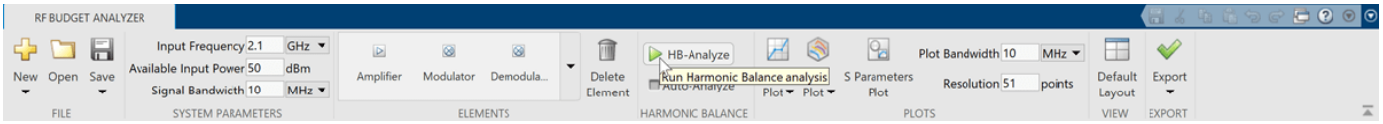
```
b = rfbudget([a m r n],2.1e9,-30,10e6);
```

Run this command in the command window, to open the system in **RF Budget Analyzer** app.

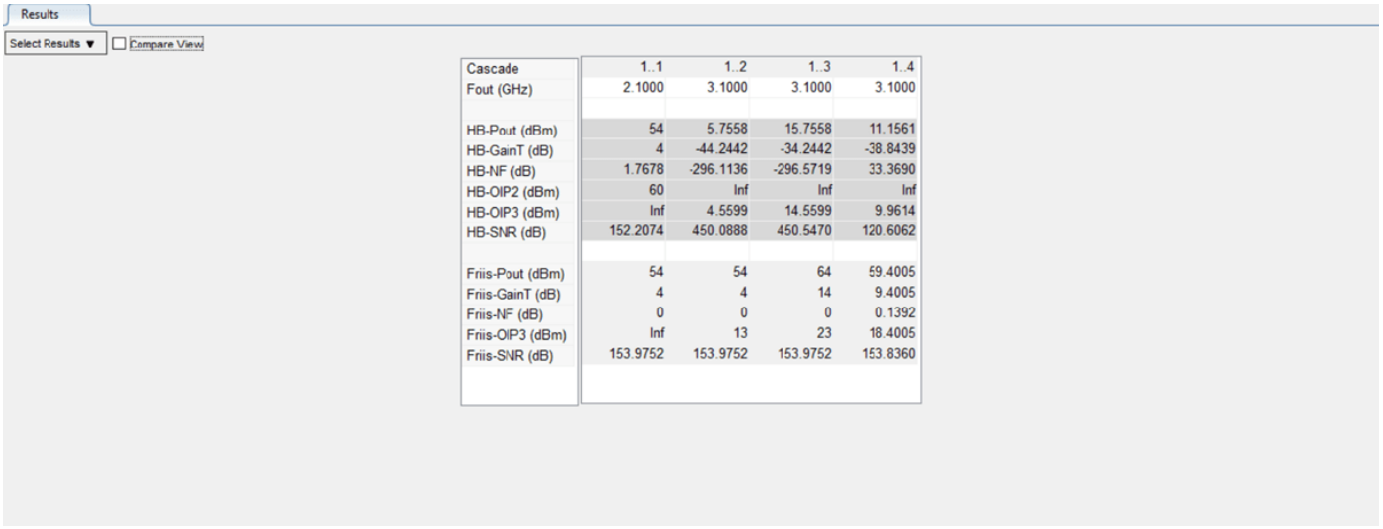
```
show(b)
```



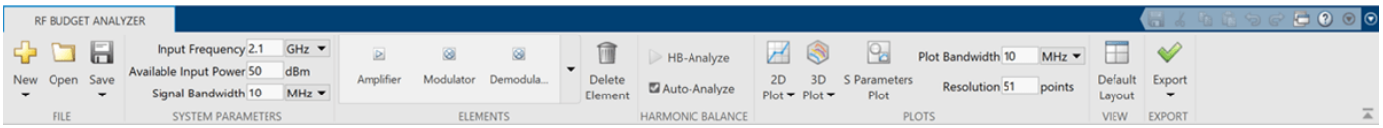
Set **OIP2** value of `Amplifier` to `60` dBm using **Elements Parameters** pane and select **Apply**. In **System Parameters** section, set the **Available Input Power** to `50` dBm and run harmonic balance analysis using **HB-Analyze** button.



The results are displayed as shown below.

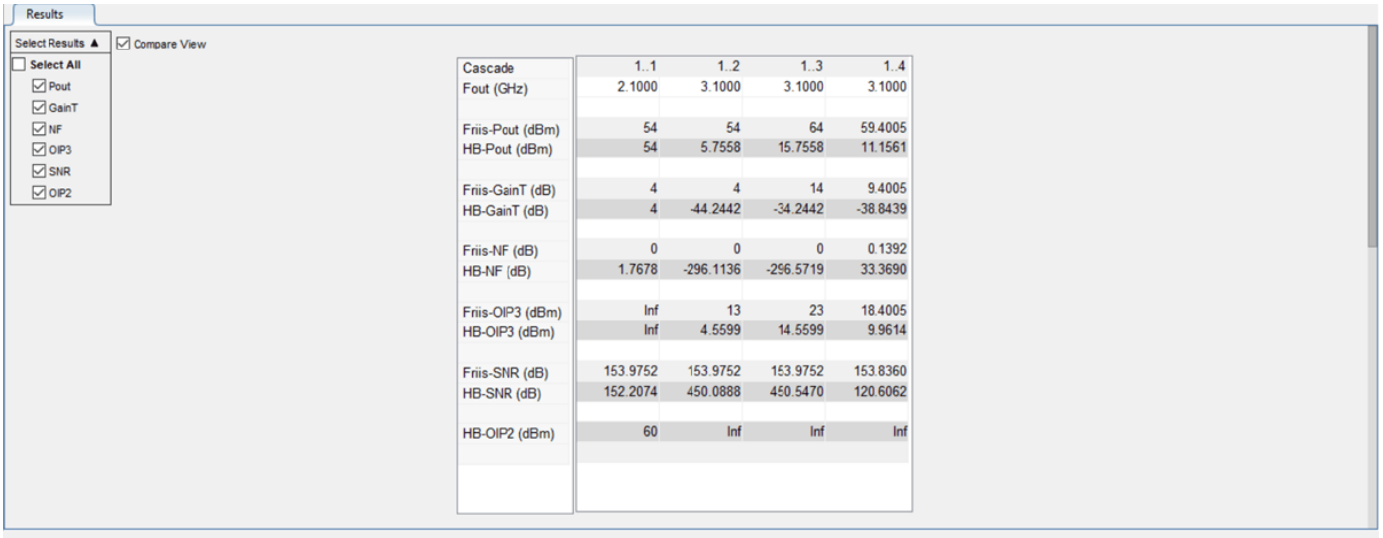


Select **Auto-Analyze** checkbox to automatically recompute the harmonic balance analysis calculations.



Set **OIP2** value of `RFelement` as `50` dBm using **Elements Parameters** pane and select **Apply.**

Select **Compare View** checkbox in the **Results** pane to compare the calculated Friis and harmonic balance solver results. You can use **Select Results** drop-down from the **Results** pane to filter the results and to compare between Friis and harmonic balance solver.



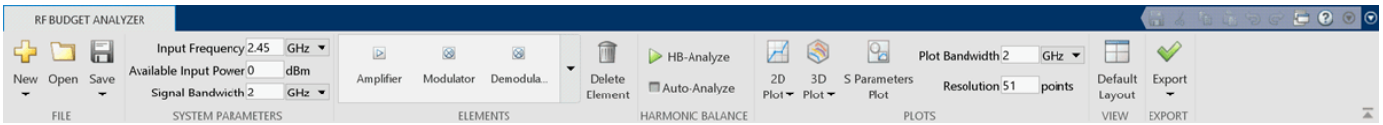
Design Input Matching Network Using Transmission Line Element

Design an input matching network for a two-stage amplifier using the **Transmission Line** element in the **RF Budget Analyzer** app.

Enter `rfBudgetAnalyzer` to open the app.

In **System Parameters**, specify the requirements:

- **Input Frequency** — `2.45` GHz
- **Available Input Power** — `0` dBm
- **Signal Bandwidth** — 2 GHz



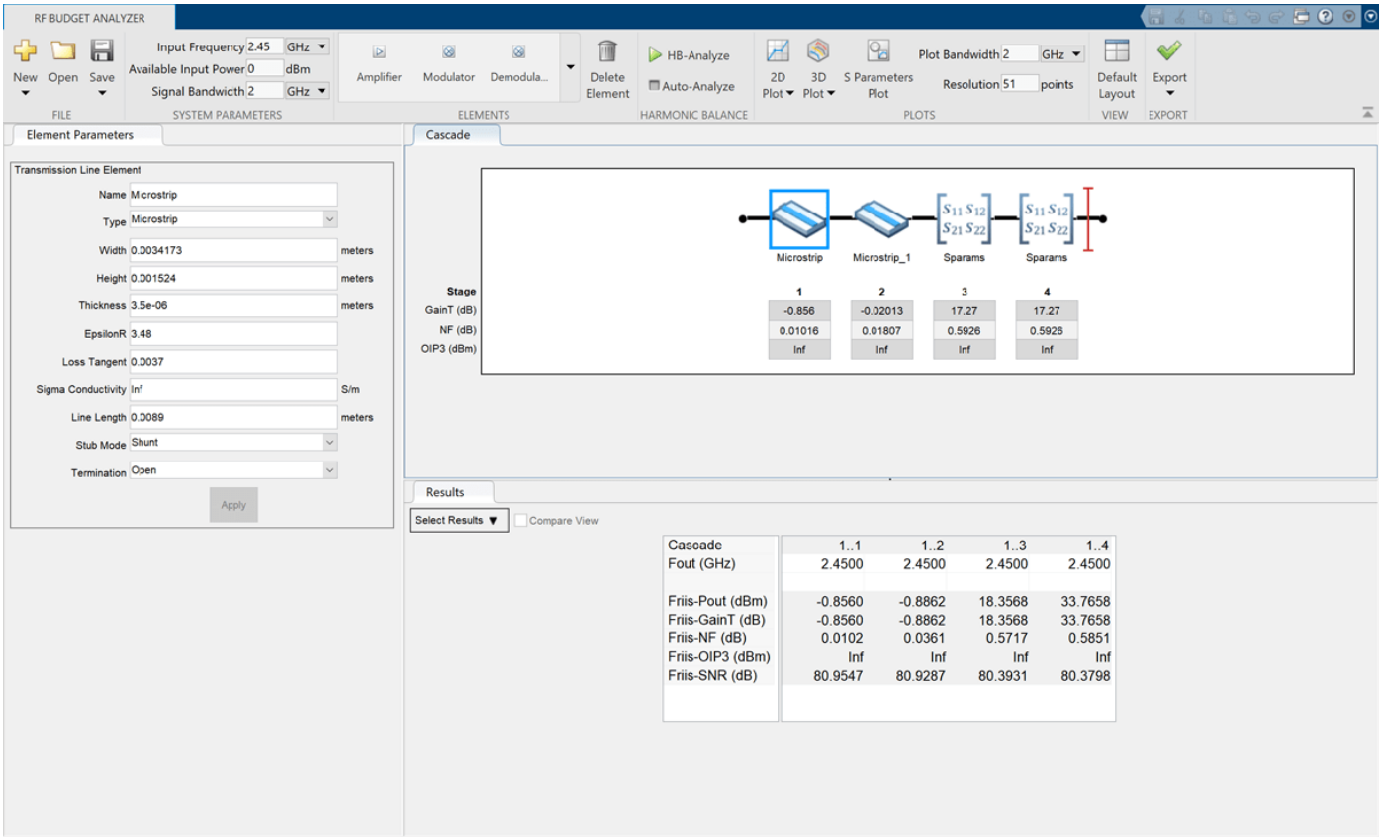
Add two **Transmission Line** elements. In the **Element Parameters** pane, specify:

- **Name** — `Microstrip1` | `Microstrip2`
- **Type** — `microstrip` | `microstrip`
- **Width** — `0.0034173` | `0.0034173` meters
- **Height** — `0.001524` | `0.001524` meters
- **Thickness** — `3.5e-06` | `3.5e-06` meters
- **EpsilonR** — `3.48` | `3.48`
- **LossTangent** — `0.0037` | `0.0037` meters
- **SigmaCond** — `Inf` | `Inf` S/m
- **LineLength** — `0.0089` | `0.0147` meters
- **StubMode** — `Shunt` | `NotAStub`
- **Termination** — `Open`
- Select **Apply**.

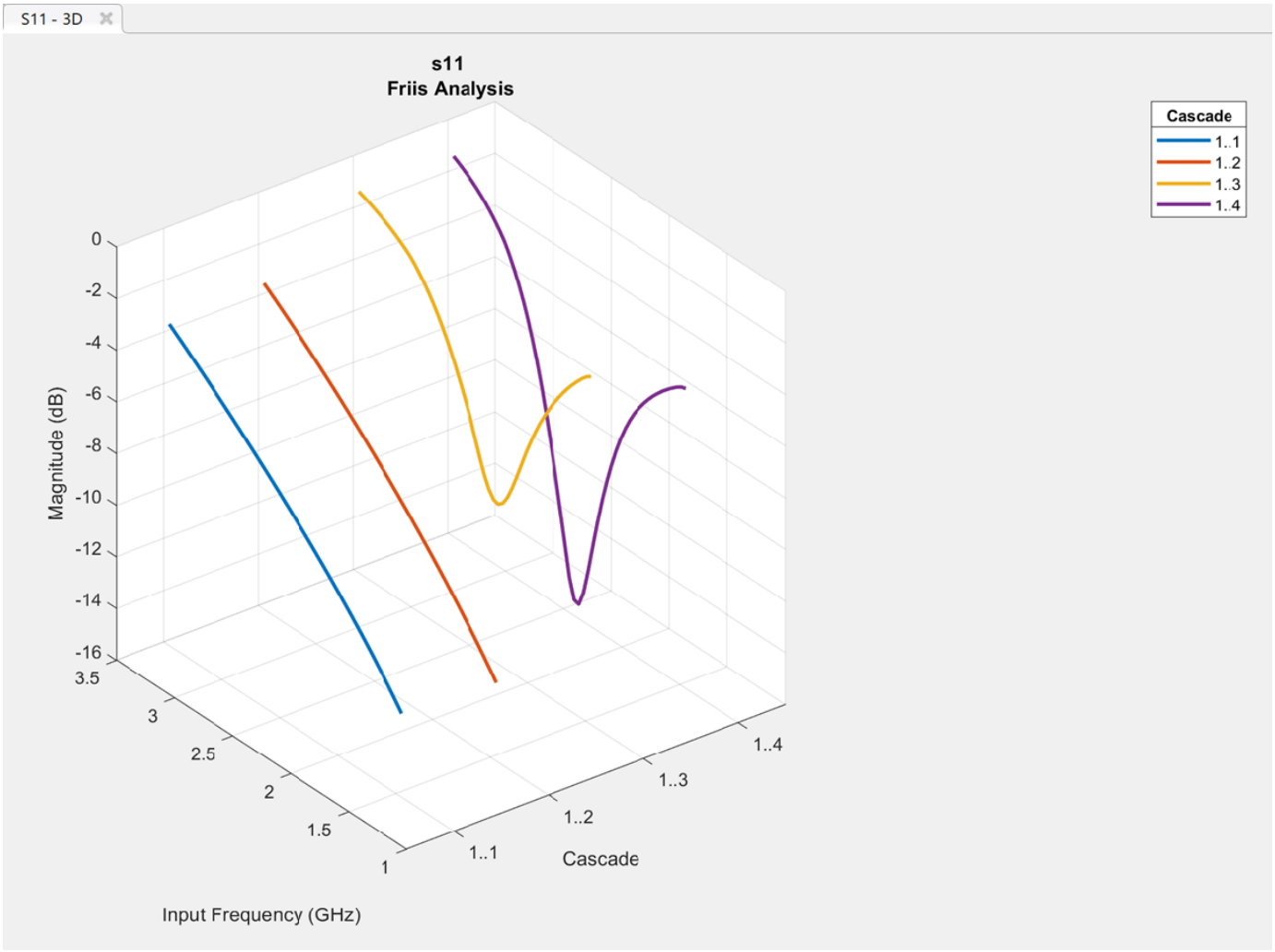
Add two **S-Parameters** elements. In the **Element Parameters** pane, specify:

- **Name** — `Sparams1` | `Sparams2`

Load the Touchstone® file (`f551432p.s2p`) to the **S-Parameters** elements provided in this example and select **Apply**.



Plot the input reflection coefficient of the system using the **3D Plot** button. Select the **3D Plot** button, choose `S-Parameters` and select `S11`.



Plot S-Parameters, Output Power, and Transducer Gain of RF System

Design an RF system and plot S-parameters, output power, and transducer gain using **RF Budget Analyzer** app.

Enter `rfBudgetAnalyzer` to open the app.

In **System Parameters**, specify the requirements:

- **Input Frequency** — `2.1` GHz
- **Available Input Power** — `-30` dBm
- **Signal Bandwidth** — 45 MHz

Add a **S-Parameters** element. In the **Element Parameters**, specify:

- **Name** — `RFBandpassFilter`

Load the Touchstone® file (`RFBudget_RF.s2p`) to the **S-Parameters** element provided in this example and select **Apply**.

Add an **Amplifier** element. In the **Element Parameters**, specify:

- **Name** — `RFAmplifier`
- **Available Power Gain** — `11.53` dB
- **NF** — `1.53` dB
- **OIP3** — `35` dBm
- Select **Apply**.

Add the **Demodulator** element. In the **Element Parameters**, specify:

- **Name** — `Demodulator`
- **Available Power Gain** — `-6` dB
- NF — `4` dB
- **OIP3** — `50` dBm
- **LO Frequency** — `2.03` GHz
- **Converter Type** — `Down`
- Select **Apply**.

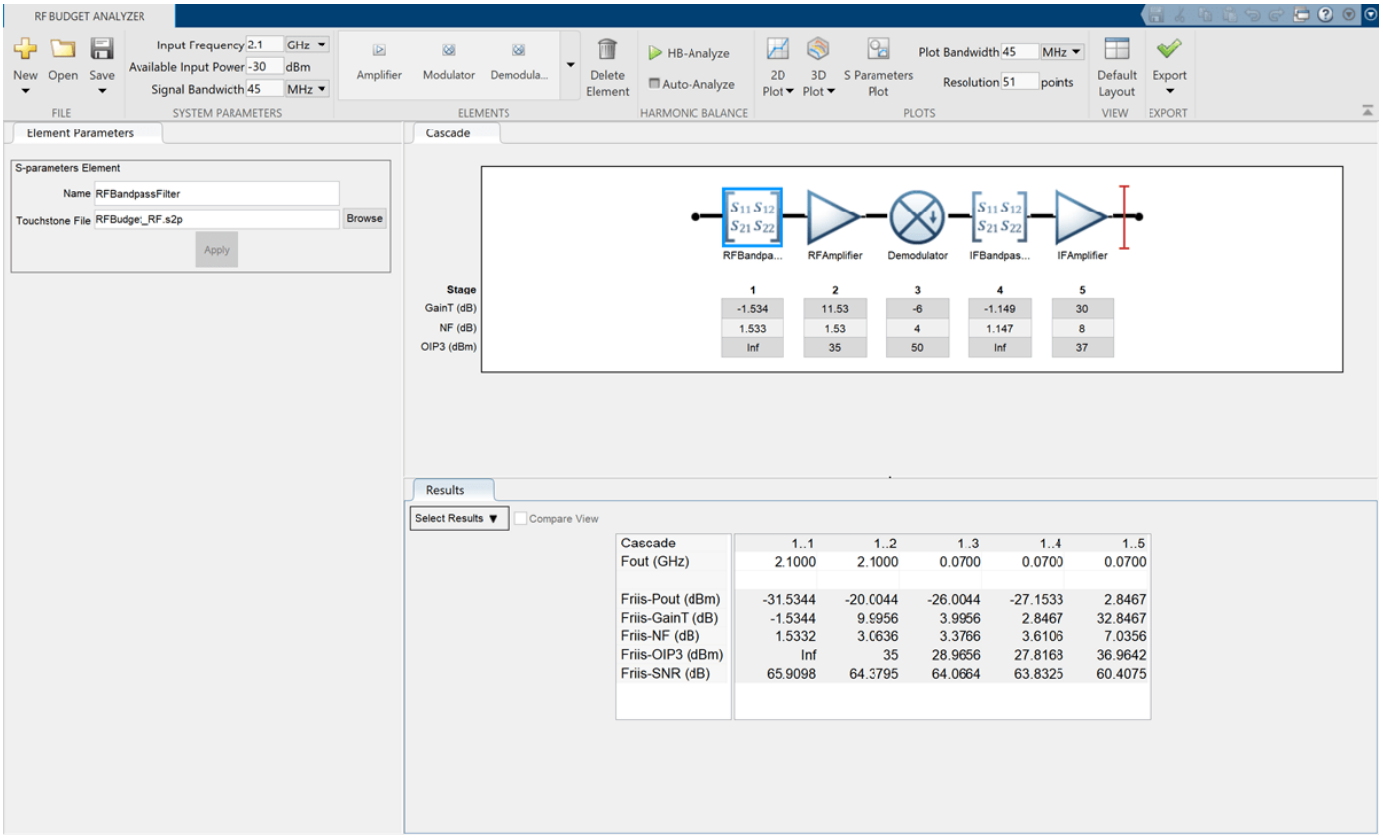
Add another **S-Parameters** element. In the **Element Parameters**, specify:

- **Name** — `IFBandpassFilter`

Load the Touchstone file (`RFBudget_IF.s2p`) to the **S-Parameters** element provided in this example and select **Apply**.

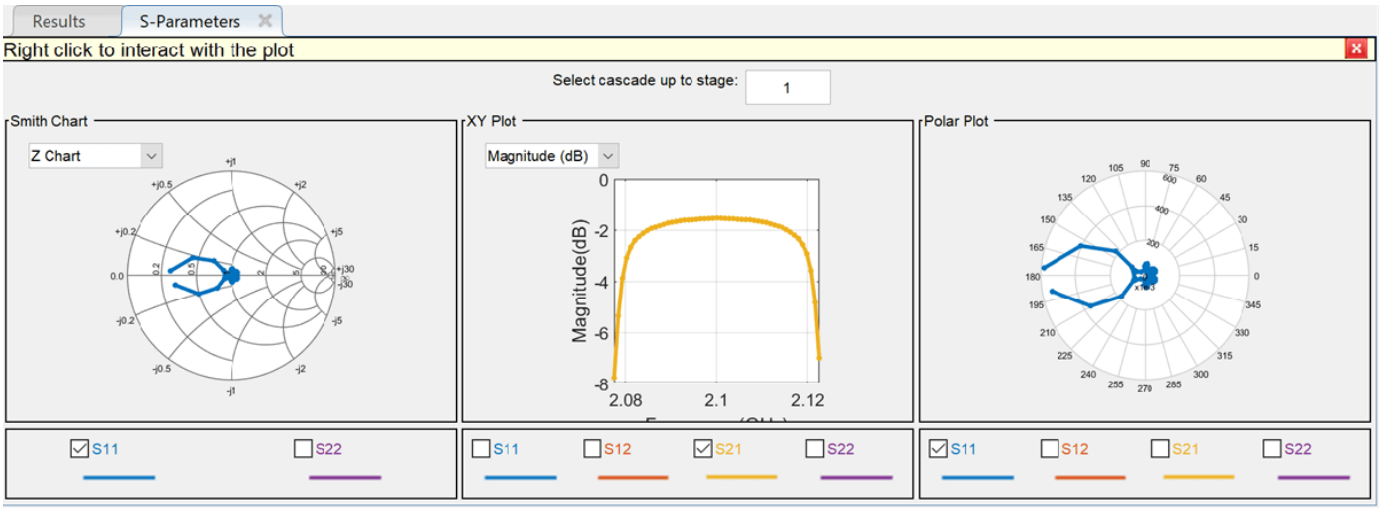
Add another **Amplifier** element. In the **Element Parameters**, specify:

- **Name** — `IFAmplifier`
- **Available Power Gain** — `30` dB
- **NF** — `8` dB
- **OIP3** — `37` dBm
- Select **Apply**.

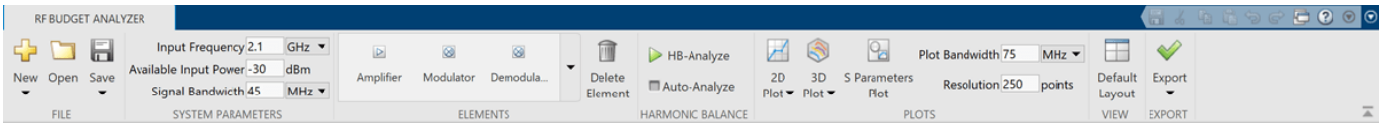


Save the system. The app saves the system in a MAT file.

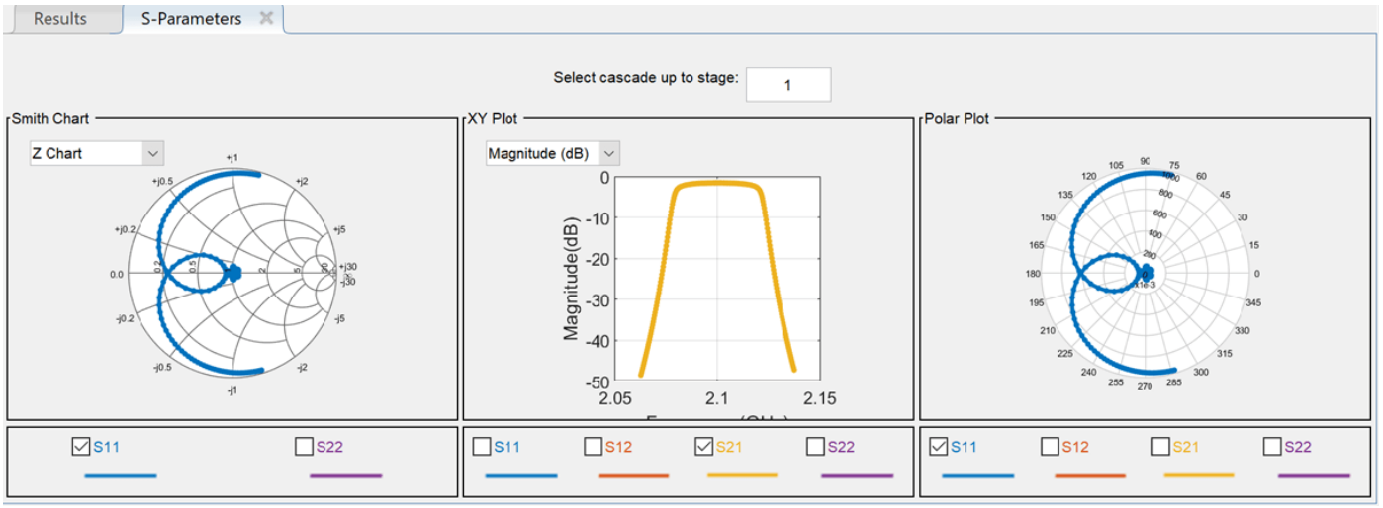
Select **S-Parameters Plot** button. This allows you to plot Smith® chart, polar plot, magnitude, phase and real, and imaginary parts of S-parameters of the RF System and over stages.



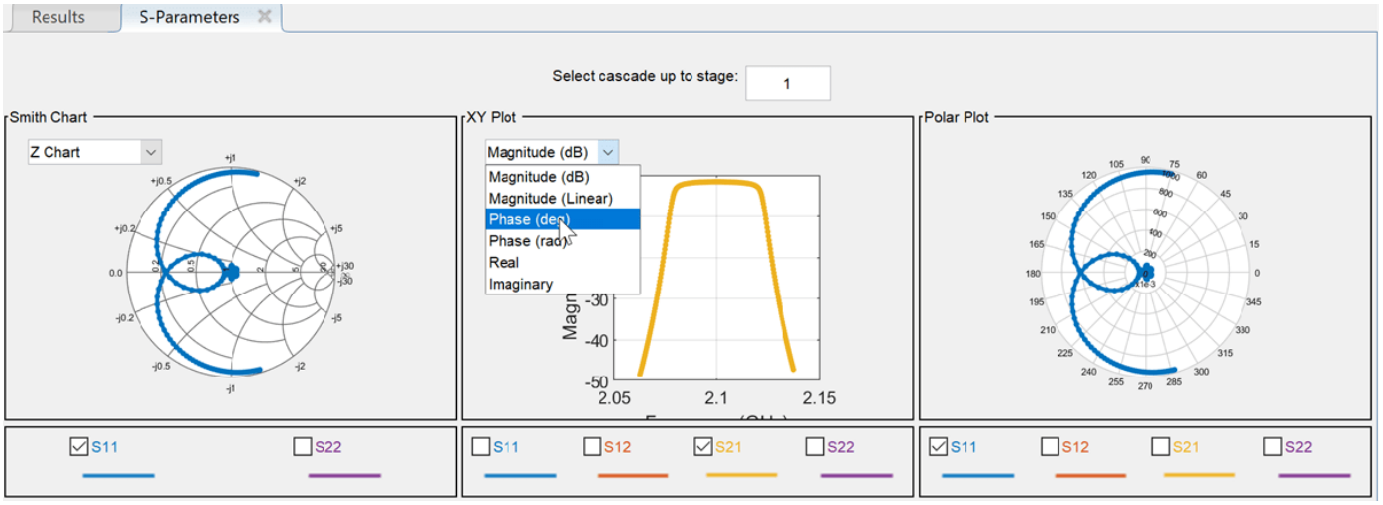
Set the **Plot Bandwidth** to 75 and **Resolution** to 250 under **Plots** section.



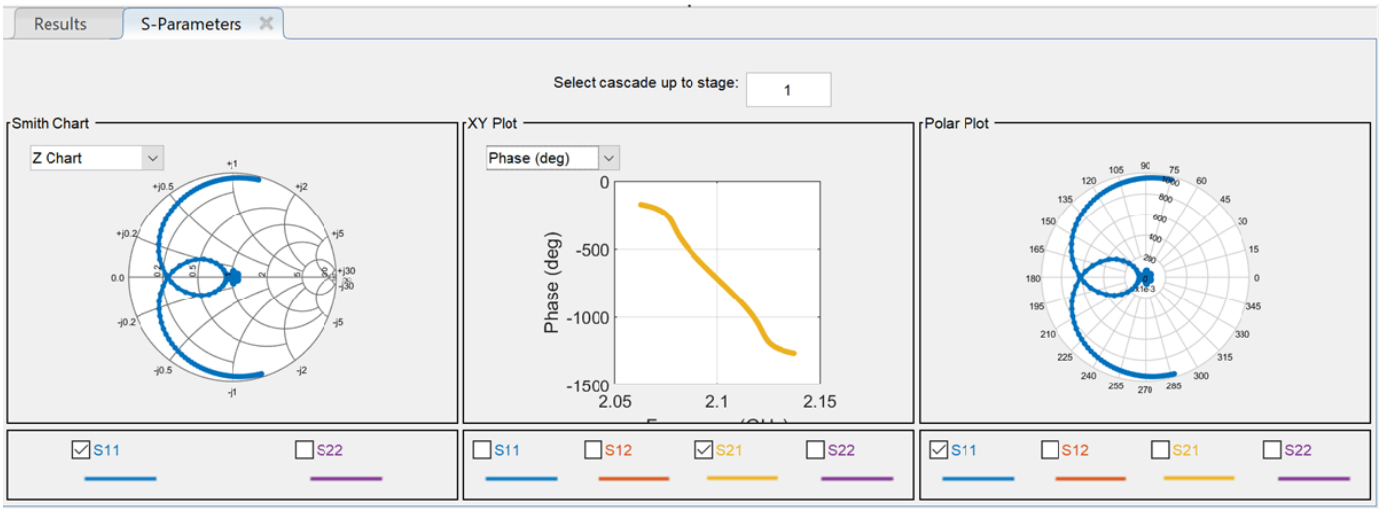
The S-parameters data is displayed as follows.



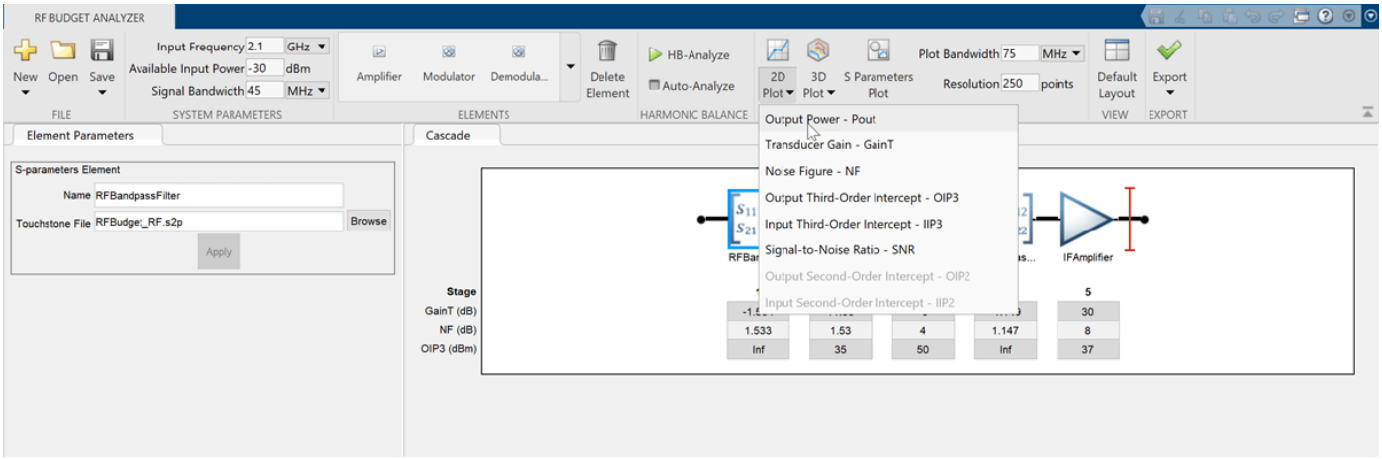
Select `Phase (deg)` from the drop-down menu of XY Plot in **S-Parameters** pane to plot the phase of the S21.



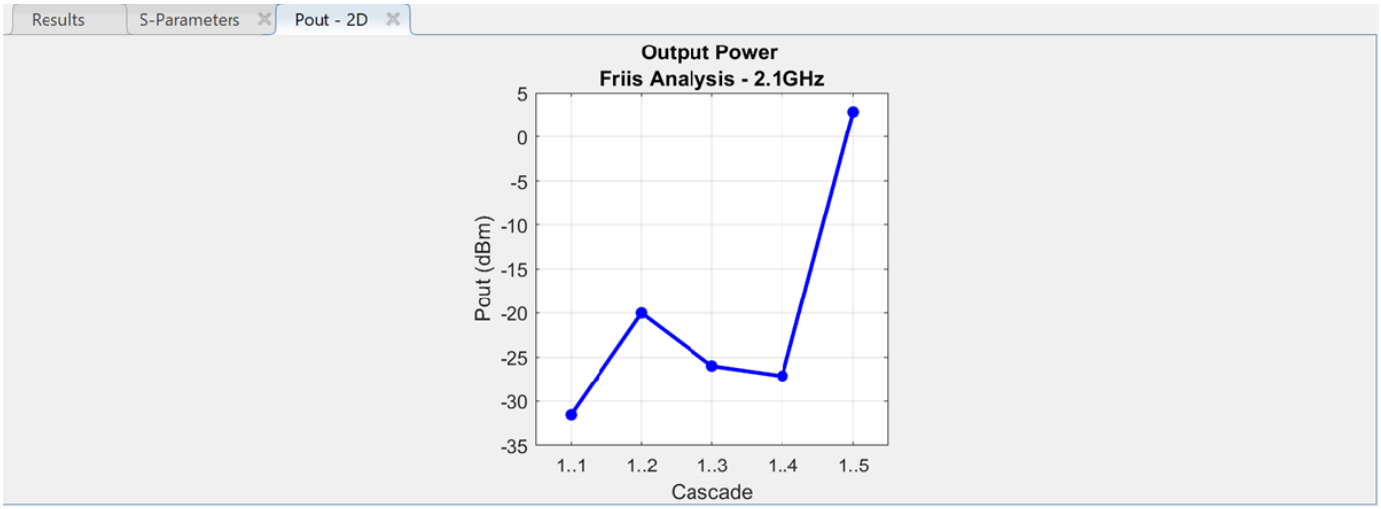
The phase plot is displayed as shown.



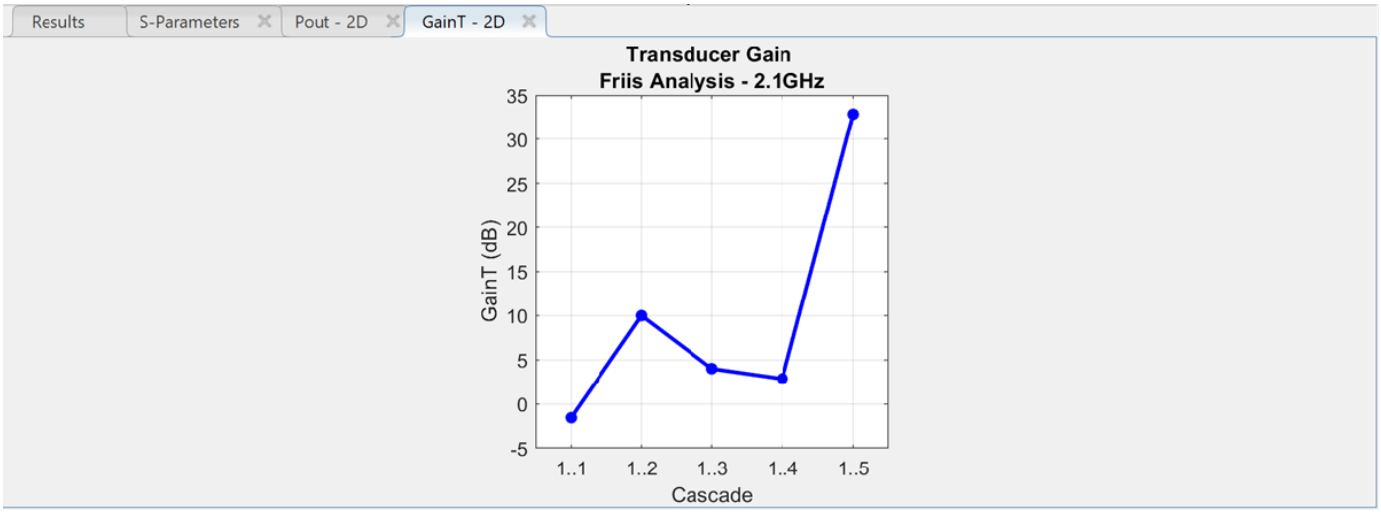
Plot the output power of the RF system using the **2D Plot** button. Select **2D Plot** button and choose `Output Power - Pout`.



2-D output power is displayed.



Plot the transducer gain of the RF system using the **2D Plot** button. Select **2D Plot** button and choose `Transducer Gain - GainT`.



Perform HB Analysis on RF Receiver Designed Using `rfsystem`

Design an RF receiver using the `rfsystem` system object. View the object in the the **RF Budget Analyzer** app to perform harmonic balance (HB) analysis.

Create fifth- and seventh-order bandpass RF filters.

```
f1 = rffilter('ResponseType','Bandpass','FilterOrder',5, ...
        'PassbandFrequency',[4.85 5.15]*1e9);
f2 = rffilter('ResponseType','Bandpass','FilterOrder',7, ...
        'PassbandFrequency',[10 130]*1e6);
```

Create two amplifier objects with 3 dB and 5 dB gain, respectively.

```
a1 = amplifier('Gain',3,'NF',1.53,'OIP3',35);
a2 = amplifier('Gain',5,'NF',8,'OIP3',37);
```

Create a modulator with a local frequency of `4.93` GHz.

```
d = modulator('Gain',0,'NF',4,'OIP3',50,'LO',4.93e9, ...
        'ConverterType','Down');
```

Design an RF receiver with the budget elements at an input frequency of `5` GHz, an available input power of – `30` dBm, and a bandwidth of `10` MHz.

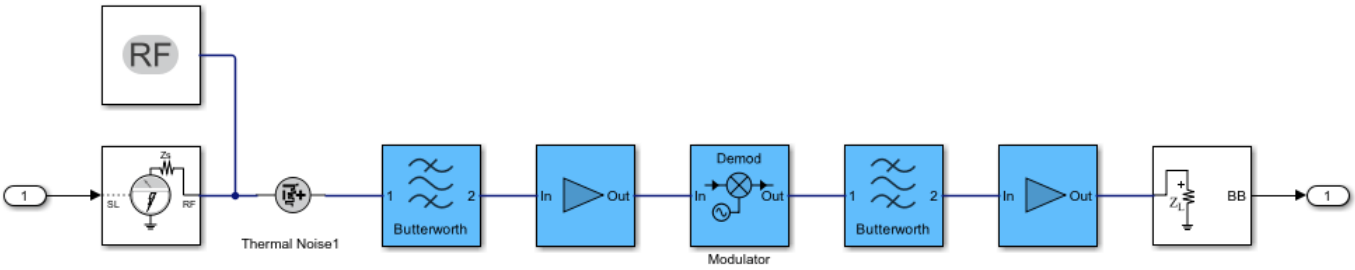
```
rfb = rfbudget([f1 a1 d f2 a2],5e9,-30,10e6);
```

Create an RF system for the RF receiver using the `rfbudget` object.

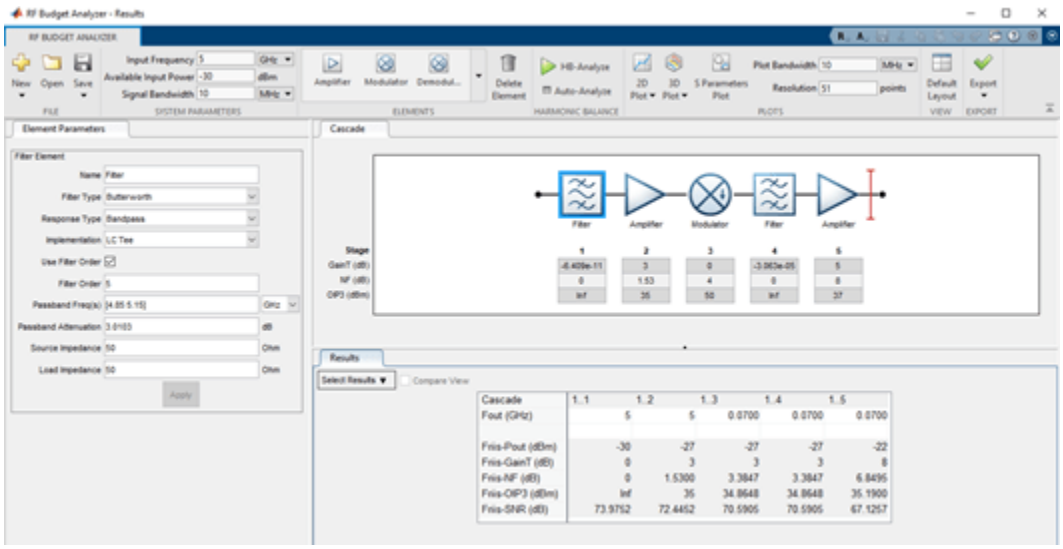
```
rfs = rfsystem(rfb);
```

Open an RF Blockset model of the designed RF system using the `open_system` object function.

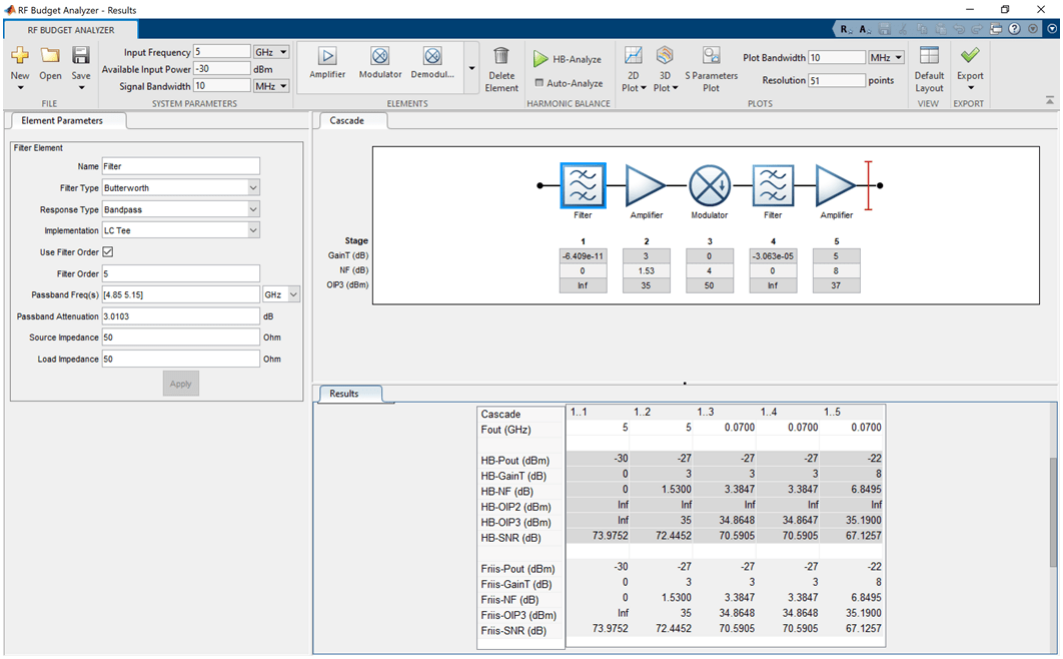
```
open_system(rfs)
```



Type rfBudgetAnalyzer(rfs) command at the MATLAB® command line to open this RF system in the **RF Budget Analyzer** app.



To conduct HB analysis in the app, click the **HB-Analyze** button.



- Superheterodyne Receiver Using RF Budget Analyzer App
- “Design RF Chain Using RF Antenna Object”

Programmatic Use

`rfBudgetAnalyzer` opens the **RF Budget Analyzer** app to analyze the stage-wise and total gain, noise figure, and nonlinearity (IP3) of an RF system.

`rfBudgetAnalyzer(rfbmat)` opens an RF system saved using the **RF Budget Analyzer** app. `rfbmat` is a MAT file.

`rfBudgetAnalyzer(rfb)` opens an RF system saved using the `rfbudget` object `rfb` in the **RF Budget Analyzer** app.

`rfBudgetAnalyzer(rfs)` opens an RF system saved using the `rfsystem` system object `rfs` in the **RF Budget Analyzer** app.

Tips

- The **RF Budget Analyzer** app accepts `0` Hz as **Input Frequency** for a system. You can set the **Input Frequency** in the **System Parameters** section.
- The **RF Budget Analyzer** app does not accept `0` Hz as **LO Frequency**. This is applicable to Modulator and Demodulator elements.
- The output frequencies from the **RF Budget Analyzer** app are always positive.
- The Filter element allows you to use only the `'Transfer function'` implementation when you set the **Filter Type** to `'InverseChebyshev'` in the **Element Parameters** pane.
- To design an antenna element using the **RF Budget Analyzer** app, in the Antenna Element pane, set **Antenna Source** to `Isotropic radiator`.You can also design an antenna element using the

Antenna Designer app or an antenna object. To use the **Antenna Designer** app or the antenna object, you need Antenna Toolbox license.

- Antenna elements designed using a default antenna object require larger memory. To speed up the simulation, design your antenna element at a high frequency, 2 GHz or more.

Version History

Introduced in R2016a

Model stripline element and use network parameter object to model S-parameter

Use the **RF Budget Analyzer** app to:

- Model a stripline element to your RF system by selecting **Transmission line** from the **Elements** tab and setting **Type** to `Stripline`.
- Model an S-parameters element with a network parameter object created at the MATLAB command line. To design the S-parameter element, first select **S-parameters** from the **Elements** tab, and then set **Data Source** to `Object`.

Model IMT mixer and transmit-receive elements and model amplifier element using network parameter object

Use the **RF Budget Analyzer** app to:

- Model a IMT mixer by selecting **Mixer IMT** element from the element pane .
- Model an antenna in transmit-receive configuration by selecting **TxRxAntenna** element from the element pane.
- Model an amplifier element with a network parameter object created at the MATLAB command line. To design an amplifier element, first select **Amplifier** from the **Elements** tab and set **Input Method** to `Network Parameters`. Then, set **Data Source** to `Object`.

Export RF system to `rfsystem` object

Use the **RF Budget Analyzer** app to export your RF system to `rfsystem` object. To export, select **Export**, and click **Export to rfsystem**.

References

[1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.

See Also

Apps
Matching Network Designer

Topics
Superheterodyne Receiver Using RF Budget Analyzer App
“Design RF Chain Using RF Antenna Object”

“Using RF Measurement Testbench” (RF Blockset)

Matching Network Designer

Design, visualize, and compare matching networks for one-port load

Description

The **Matching Network Designer** app lets you design, visualize, and compare matching networks for one-port load.

Using this app, you can:

- 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.

  Note

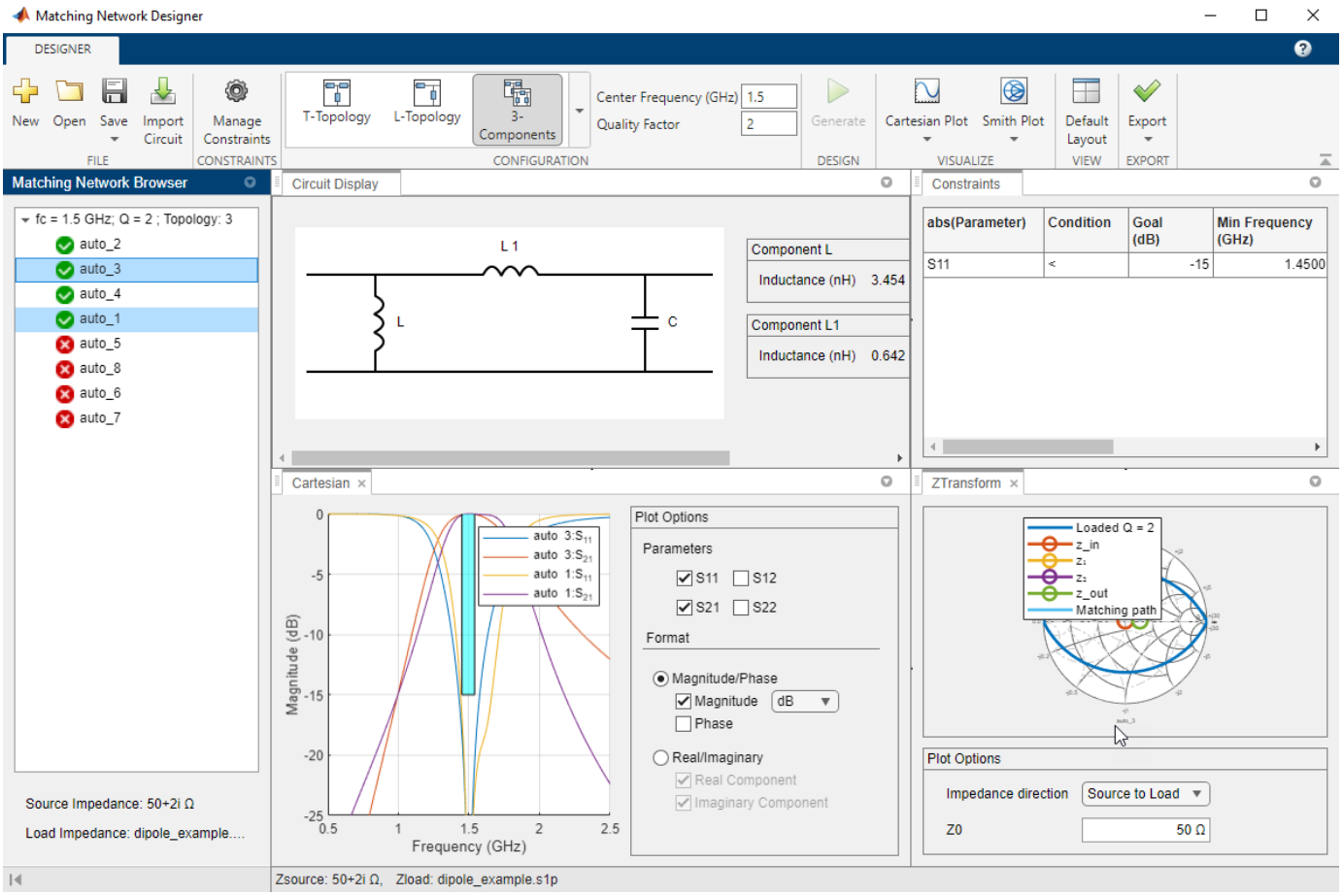
  - To load one-port circuit object to the app, you must set ports to your circuit object using the `setports` function.
  - To use an Antenna Toolbox object, you must have an Antenna Toolbox license.
  - One-port Touchstone files include S1P, Z1P, and Y1P file types.

- Sort the 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 Configurations

The app toolstrip contains these network configurations that you can use to design matching networks:

- Pi-Topology
- T-Topology
- L-Topology
- 3-Components



Open the Matching Network Designer App

- MATLAB Toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, click the **Matching Network Designer** app icon.
- MATLAB command prompt: Enter `matchingNetworkDesigner`.

Examples

Design, Visualize, and Compare Three-Component Matching Networks

Type this command at the command line to open the **Matching Network Designer** app.

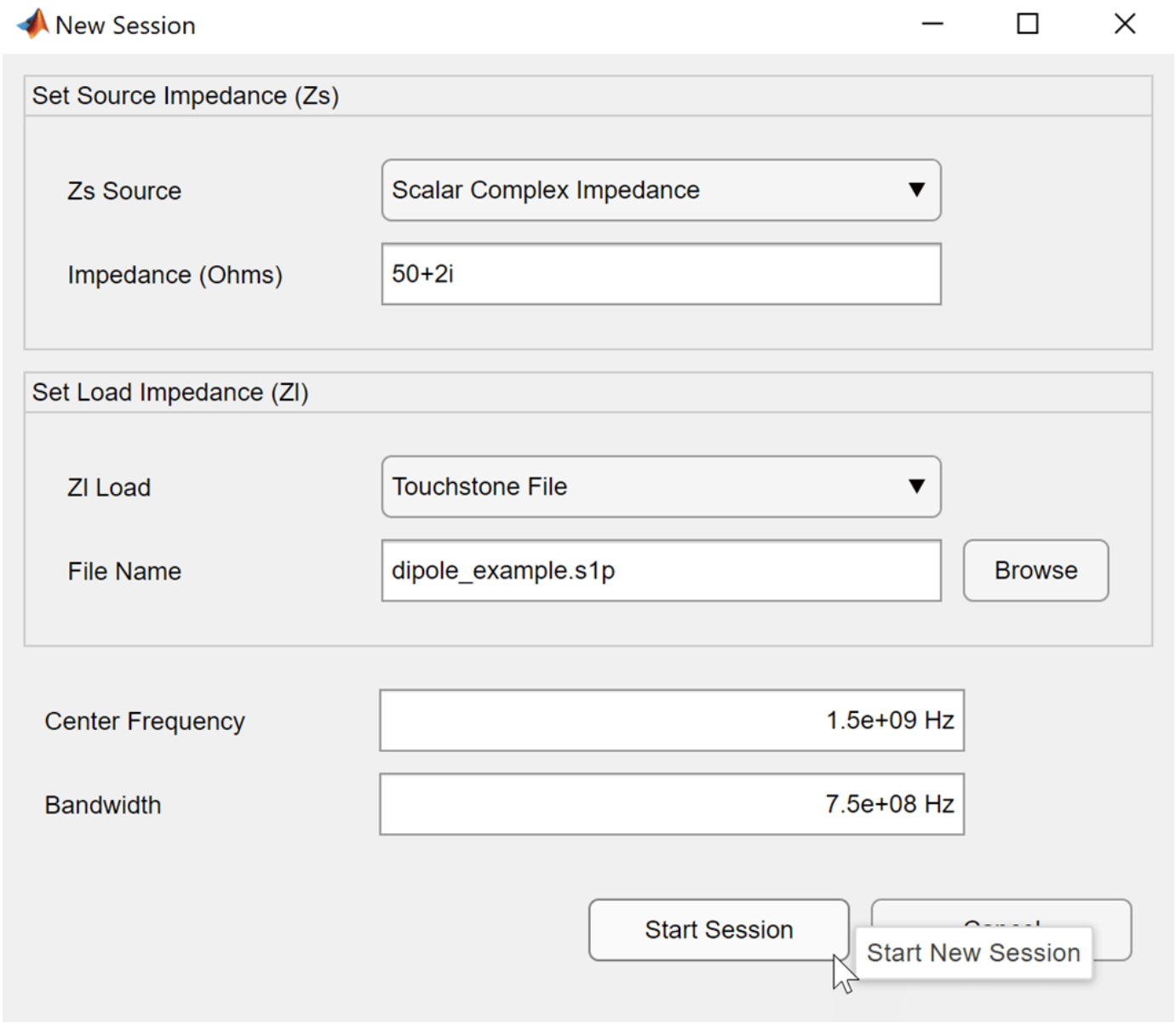
```
matchingNetworkDesigner
```

Select **New** under `File` section to start a new session. In the **New Session** window, specify the design requirements:

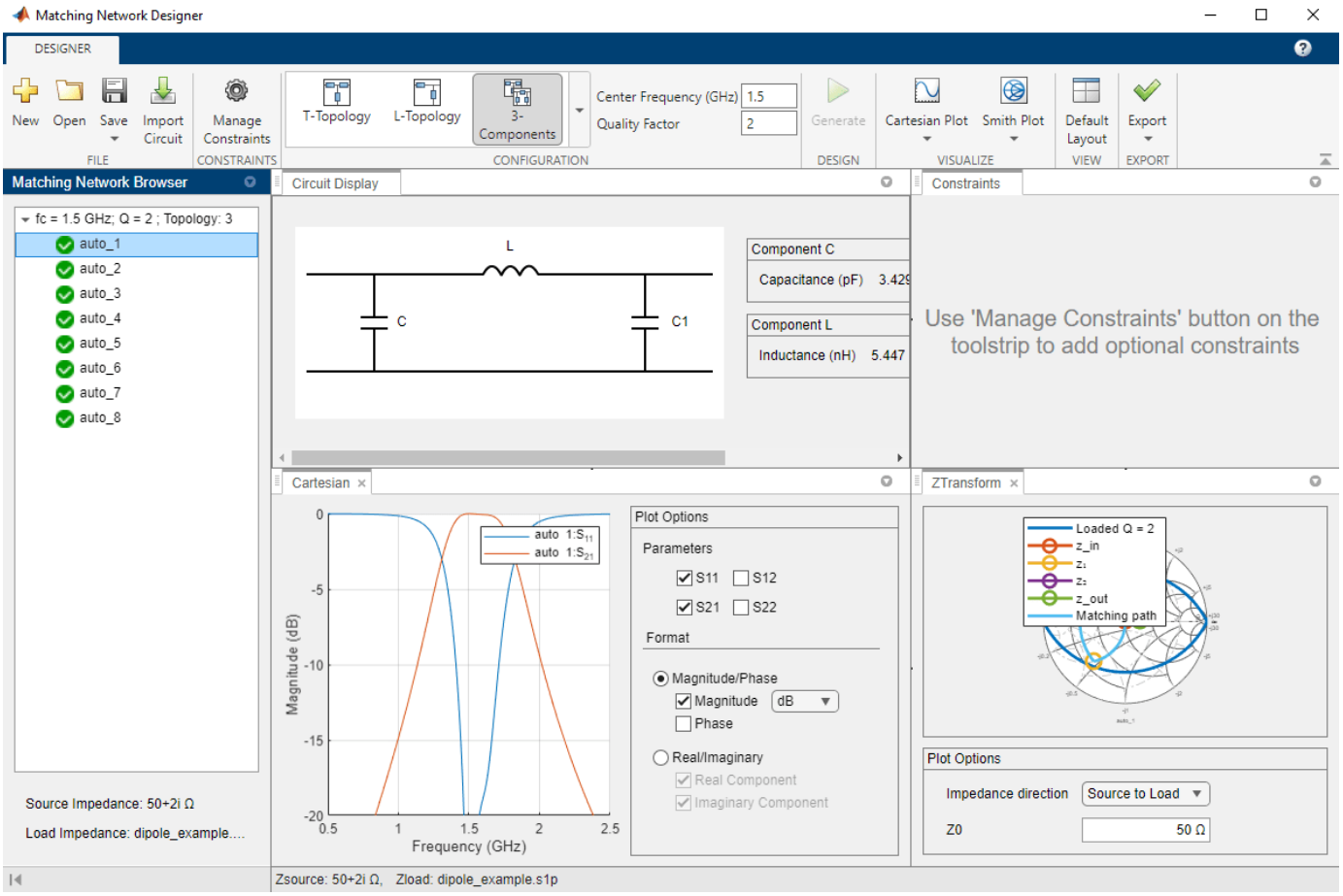
- **Zs Source** — `Scalar Complex Impedance`
- **Impedance (Ohms)** — `50+2i`.

- **Zl Source** — `Touchstone File`
- **File Name** — `dipole_example.s1p`
- **Center Frequency** — `1.5e9` and
- **Bandwidth** — `750e6`.

The app only recognizes one-port Touchstone files and converts the center frequency and bandwidth to Hz.



Select **Start session**. In the toolstrip of the app window, select **3-Components** under the `Configuration` section and select **Generate** to generate the matching network. From the **Matching Network Browser** pane, select the nodes. For the purpose of this example, select `auto_1`. The **Quality Factor** is populated based on the data entered in the **New Session** window.



Set constraints to sort the three-component matching networks. To do this, click **Manage Constraints**. In the **Design Constraints** window, click  button and add the constraints. Set the constraint to:

abs(Parameter) — `S11`

Condition — `<`

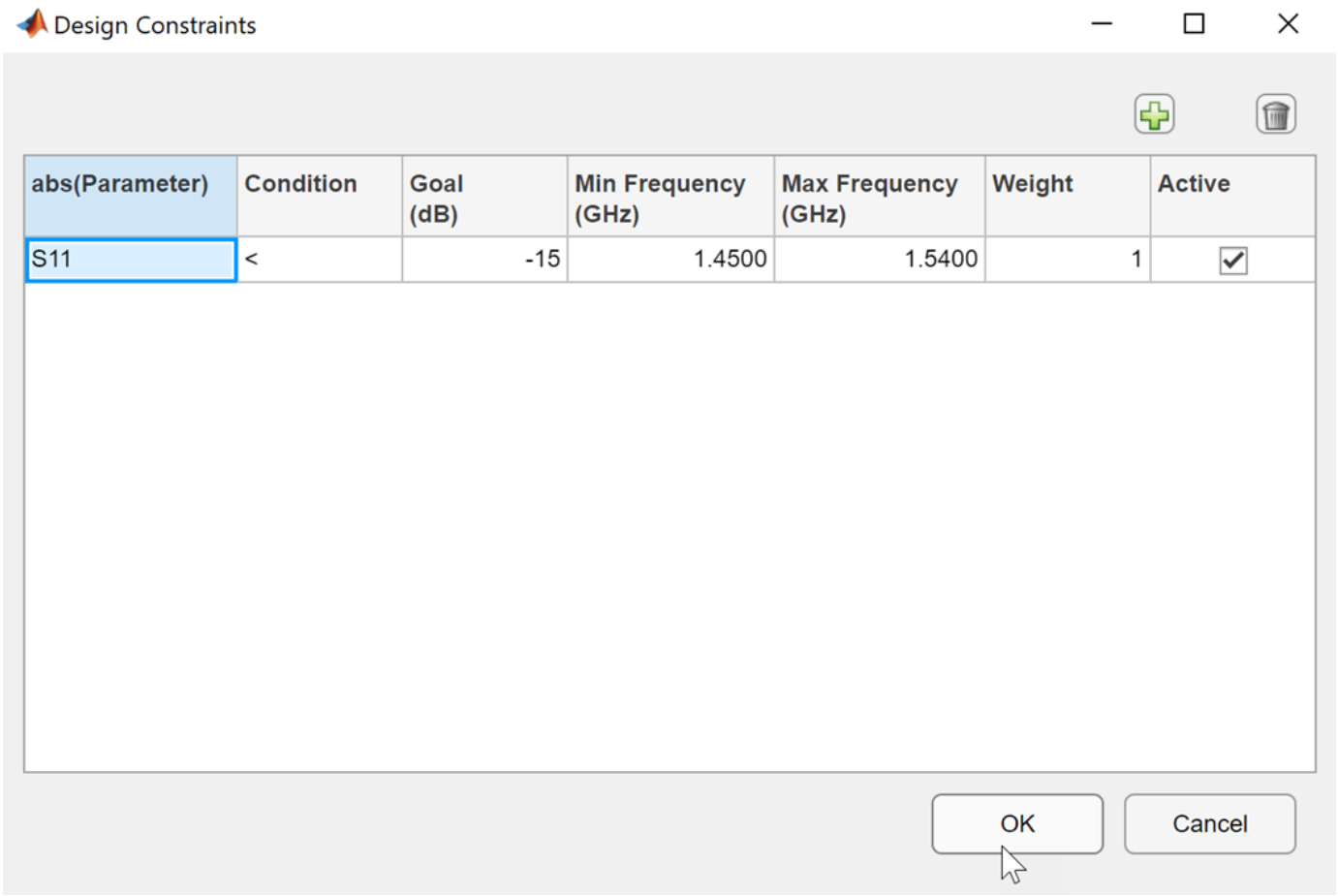
Goal (dB) — `-15`

Min Frequency (GHz) — `1.4500`

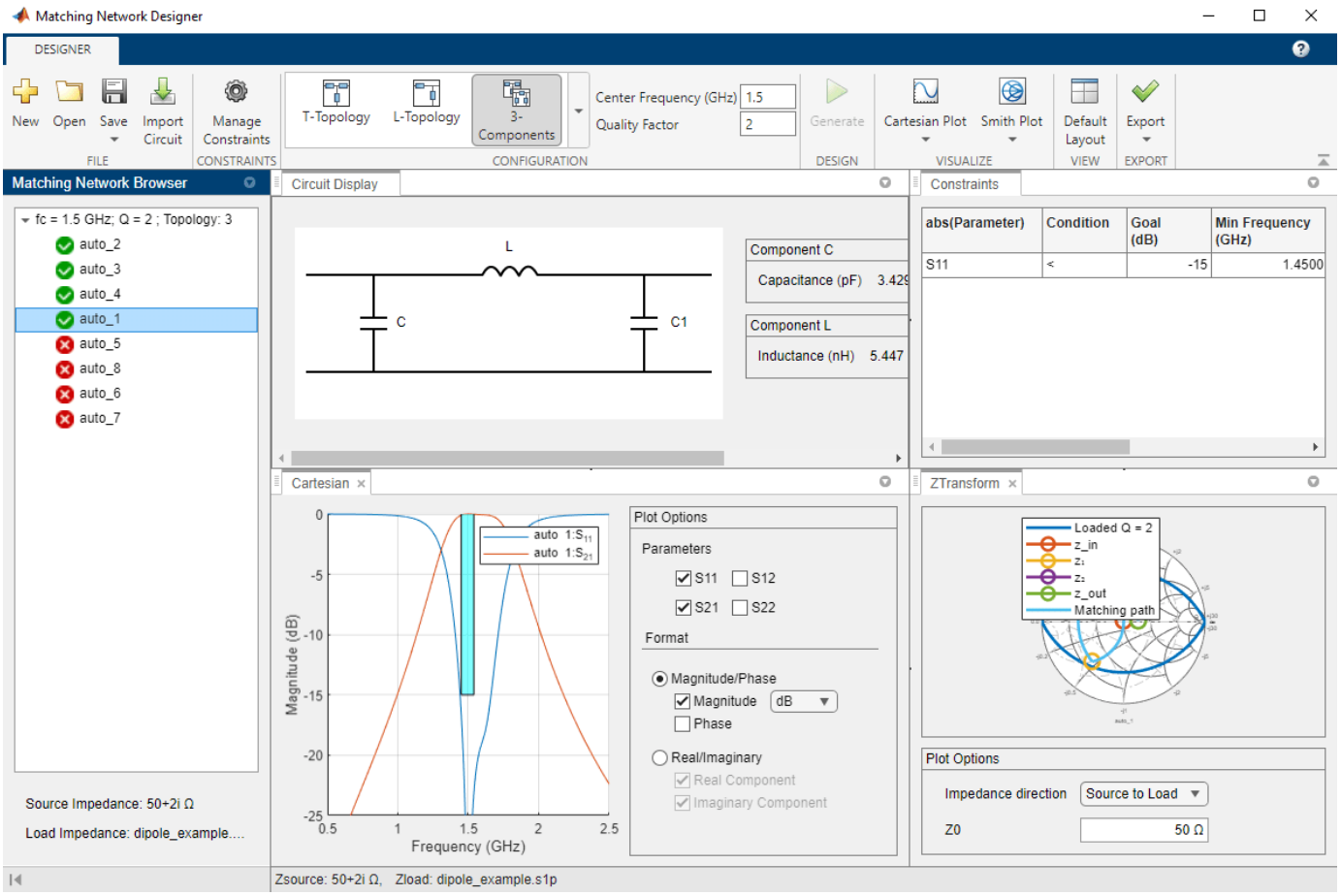
Max Frequency (GHz) — `1.5400`

Weight — `1`

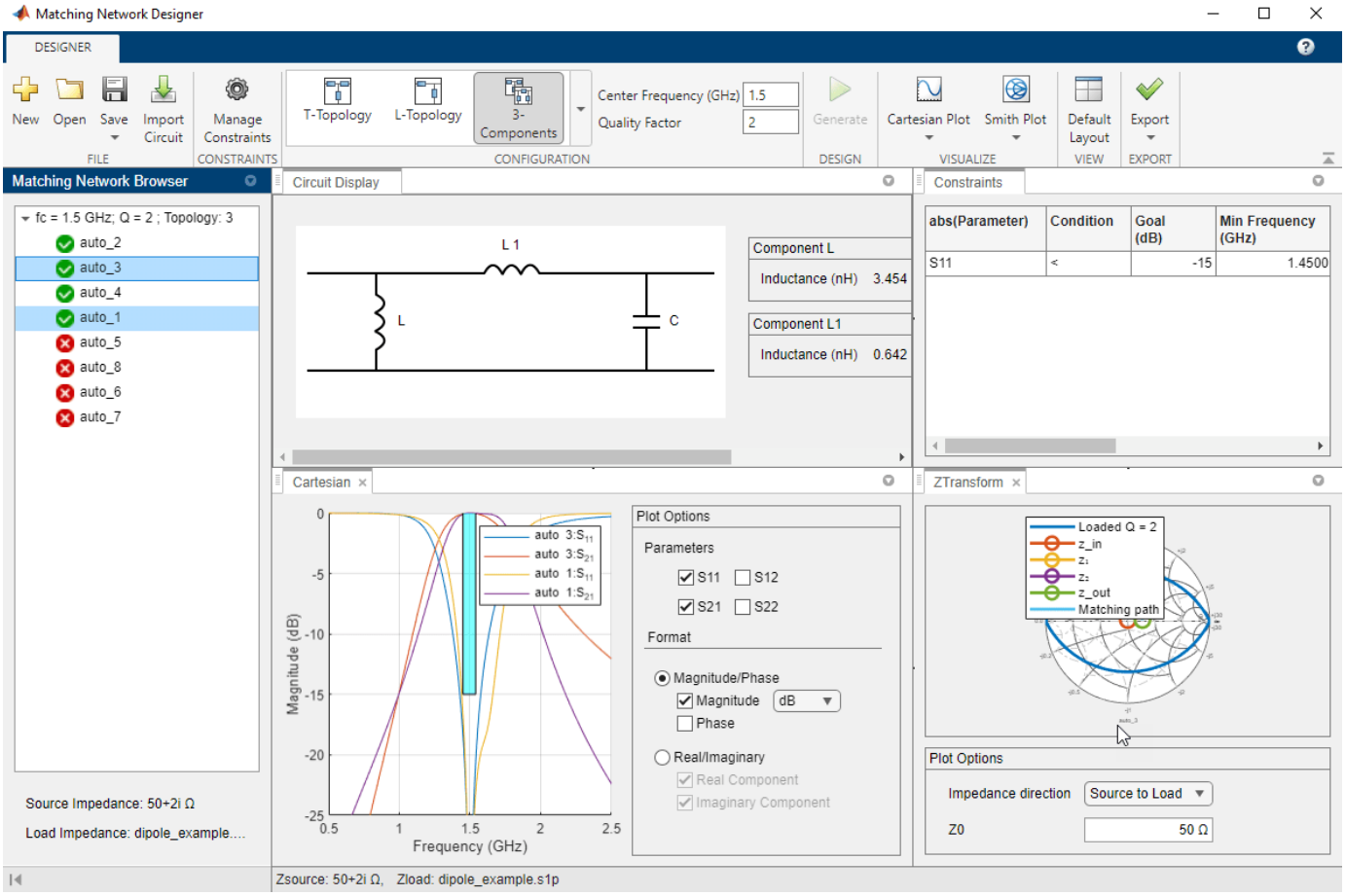
Select **Active** and click **OK**.



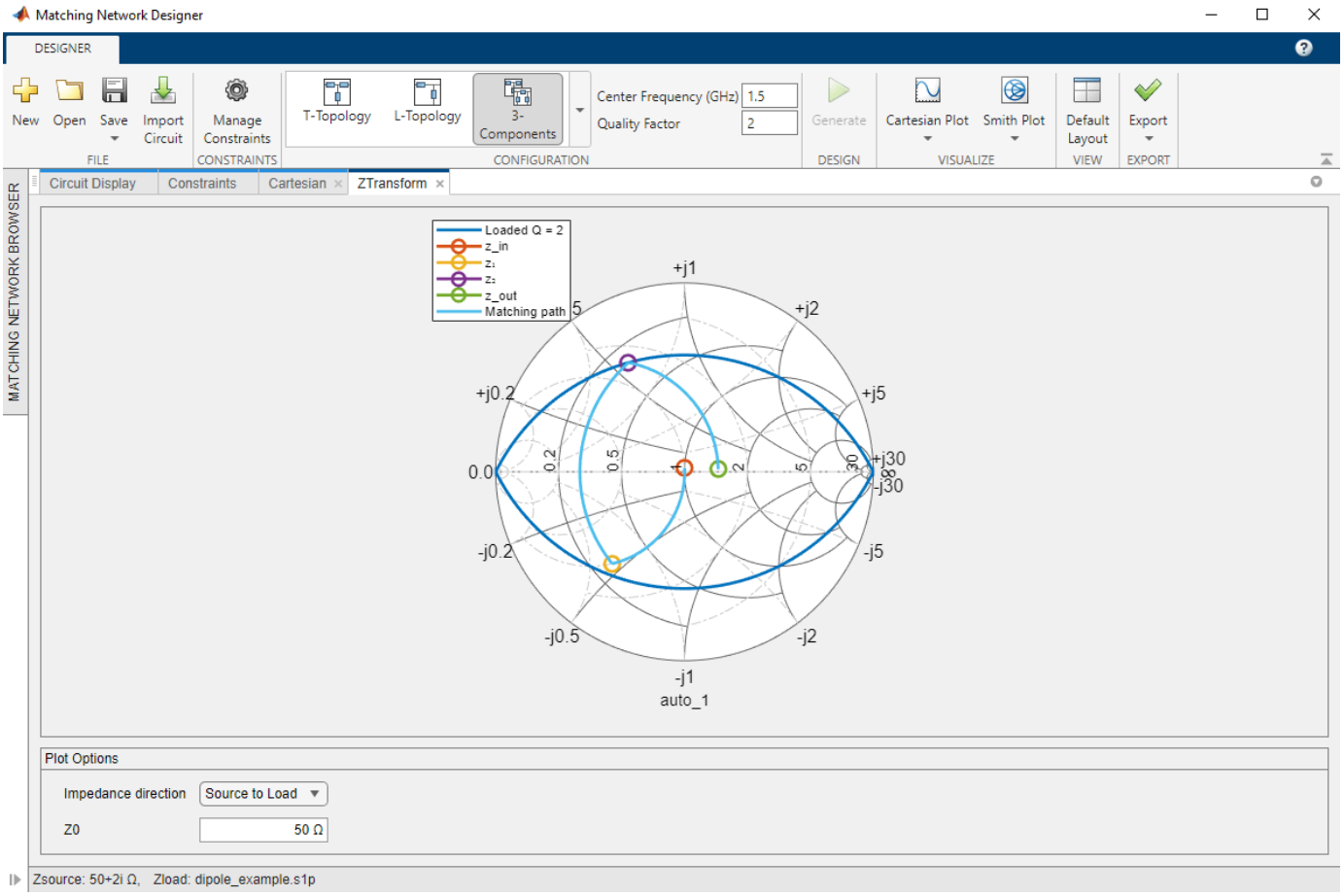
The matching networks are sorted based on the constraints and the nodes are rearranged under the **Matching Network Browser** pane.



Compare the power wave S-parameter results between the nodes. For the purpose of this example, compare the power wave S-parameter results between `auto_1` and `auto_3` nodes. To do this, select the `auto_1` and `auto_3` nodes using the **Ctrl** key. The results are displayed in the Cartesian and Smith plot.



Deselect the `auto_3` node. To visualize the impedance transformation of the `auto_1` node, select **Impedance Transformation** under **Smith Plot** or select the `ZTransform` window on the right hand side of the app.



Design Narrow-Band Double Tuning L-Section Matching Network for Monopole Antenna

Design a narrow-band double tuning L-section matching network between a resistive source and a capacitive load in the form of a small monopole. This example designs an L-section matching network consisting of two inductors. The equivalent source impedance is `50` ohms and the load is a monopole with resonant frequency of around `1` GHz. The load (antenna) impedance is at `500` MHz, which is half the resonant frequency.

```
load_antenna = design(monopole,1e9);
sparams_load = sparameters(load_antenna,linspace(0.45e9,0.55e9,101));
```

To open the **Matching Network Designer** app, type this command at the command line.

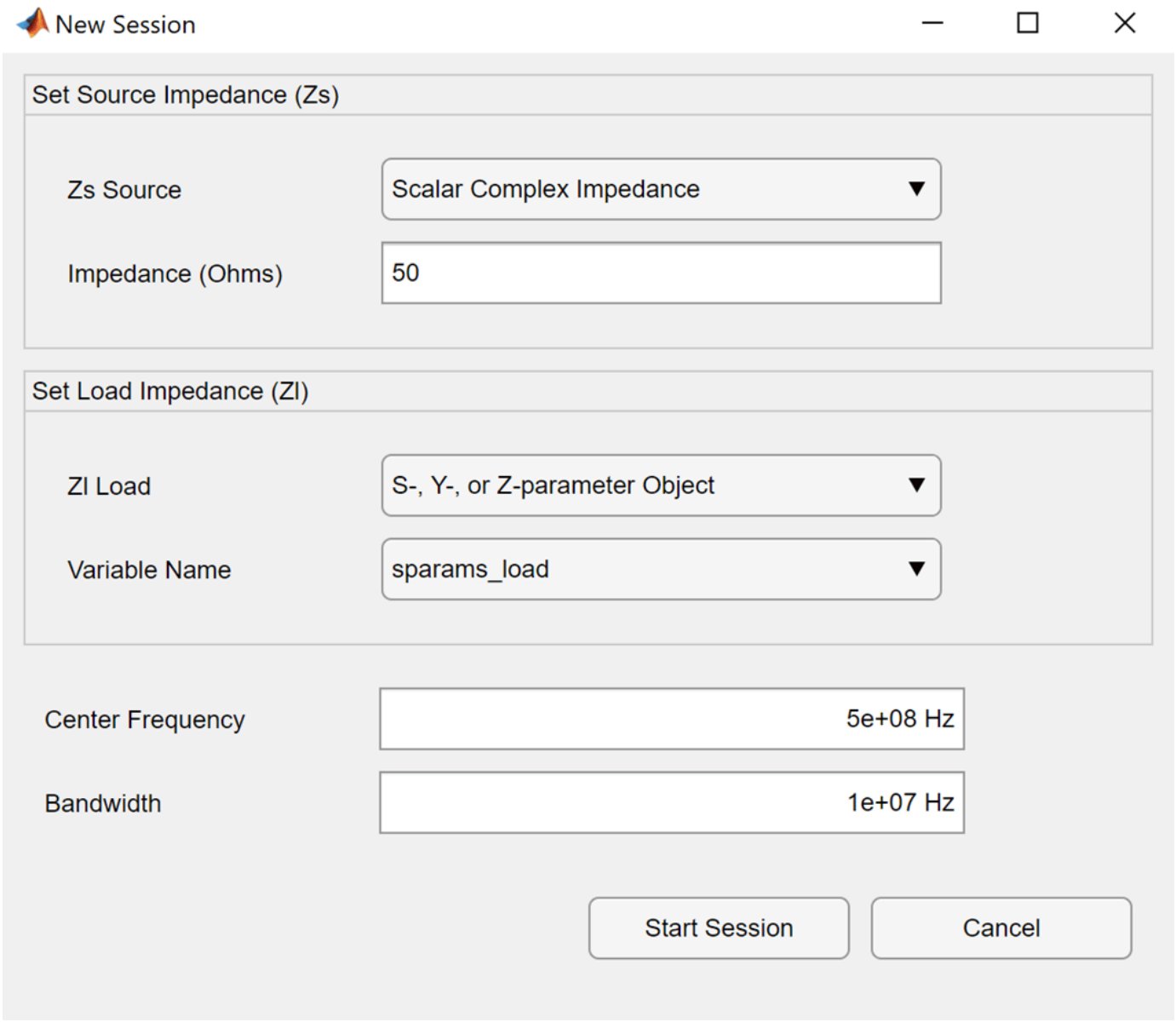
```
matchingNetworkDesigner
```

Select **New** under `File` section to start a new session. In the **New Session** window, specify the requirements:

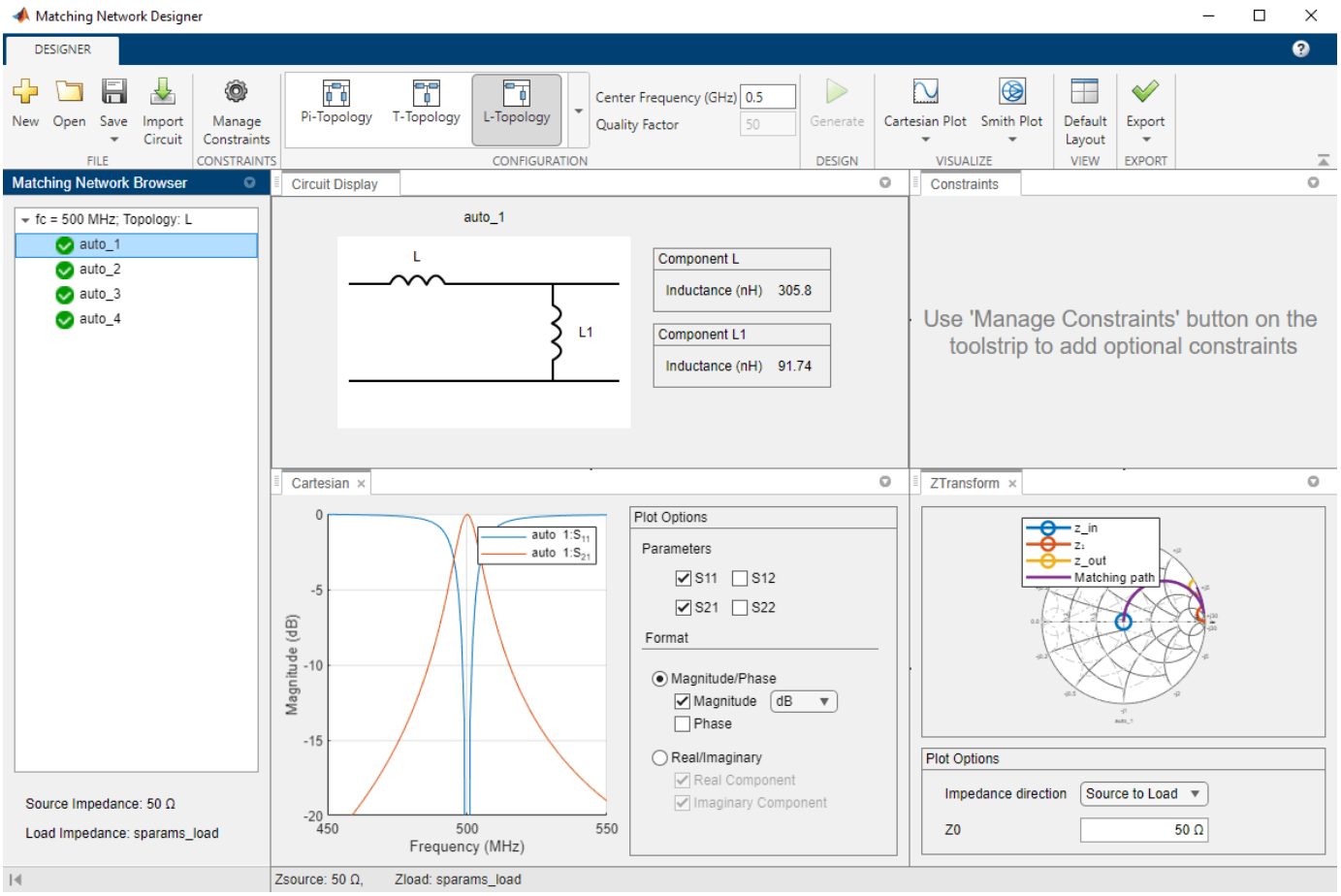
- **Zs Source** — `Scalar Complex Impedance`
- **Impedance (Ohms)** — `50`

- **Zl Source** — `S-,Y-, or Z-parameter Object`
- **Variable Name** — `sparams_load`
- **Center Frequency** — `500e6` and
- **Bandwidth** — `10e6`.

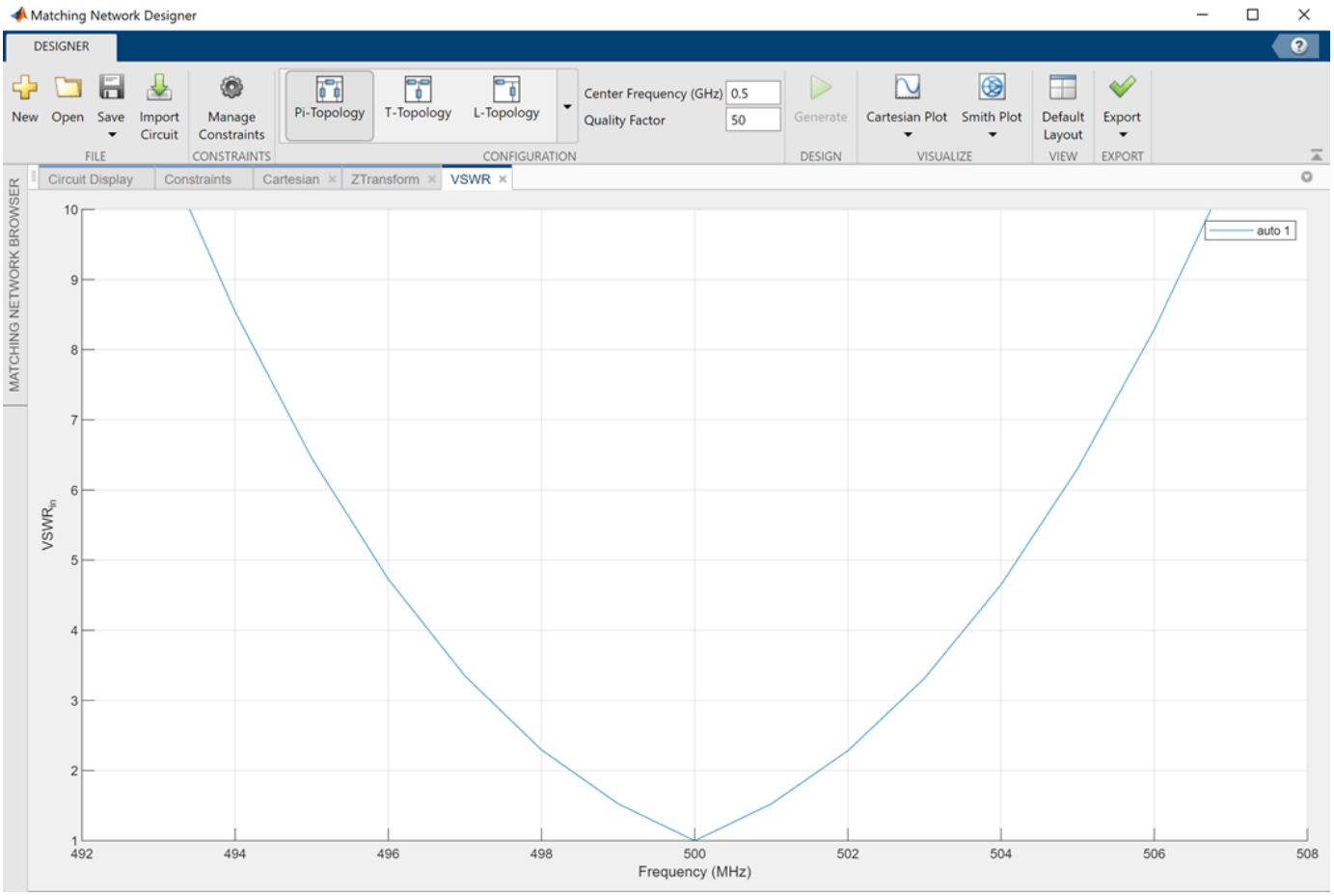
The app converts the center frequency and bandwidth to Hz.



Select **Start Session**. In the toolstrip of the app window, select **L-Topology** under `Configuration` section and select **Generate** to generate the matching network. From the **Matching Network Browser** pane, select the nodes. For the purpose of this example, select `auto_1`.



To plot the VSWR, select `VSWR` under **Cartesian plot**.



Import Custom Matching Network

Design a pi-matching network with circuit objects. For the purpose of this example, the custom pi-matching network consists of two capacitors and an inductor.

Create a `circuit` object.

```
ckt = circuit('test_ckt2');
```

Create two capacitors, `C1` and `C2` with the capacitance of `3.35` pF and `2.917` pF.

```
c1 = capacitor(3.35e-12,'C1');
c2 = capacitor(2.917e-12,'C2');
```

Create a `5.44` nH inductor.

```
l = inductor(5.44e-9,'L');
```

Add `C1` to the node `[1,0]` of the circuit object.

```
add(ckt,[1,0],c1);
```

Add `L` to the node `[1,2]` of the circuit object.

```
add(ckt,[1,2],l);
```

Add `C2` to the node `[2,0]` of the circuit object.

```
add(ckt,[2,0],c2);
```

Save the circuit object.

```
save('test_file2.mat','ckt');
```

Set ports to the circuit object and resave the circuit object in MAT file type.

```
setports(ckt,[1 0],[2 0]);
save('test_file2.mat','ckt');
```

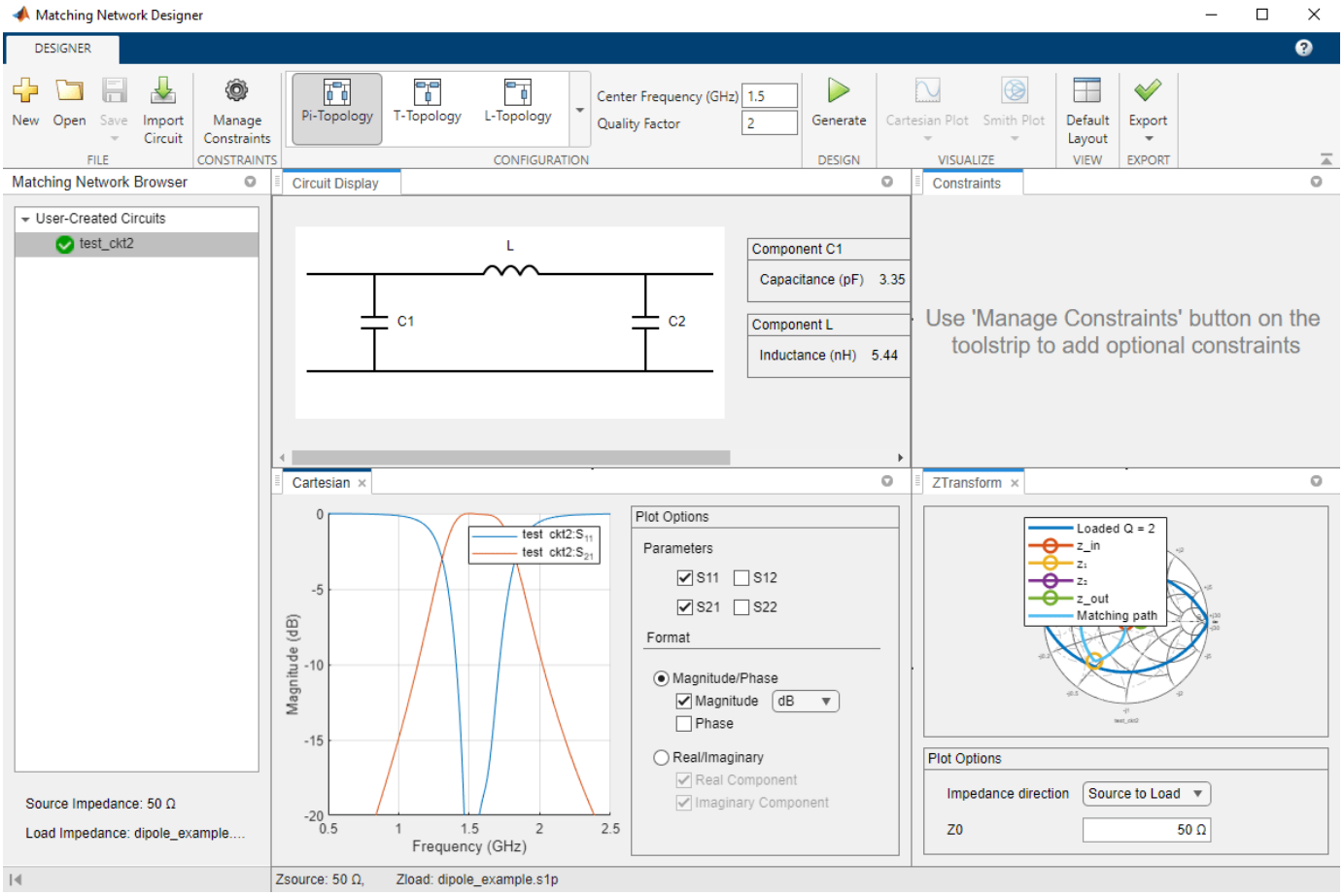
Type this command at the command line to open the **Matching Network Designer** app.

```
matchingNetworkDesigner
```

Select **New** under `File` section to start a new session. In the **New Session** window, specify the design requirements:

- **Zs Source** — `Scalar Complex Impedance`
- **Impedance (Ohms)** — `50`
- **Zl Source** — `Touchstone File`
- **File Name** — `dipole_example.s1p`
- **Center Frequency** — `1.5e9` and
- **Bandwidth** — `750e6`

Select **Import Circuit** to import the custom pi-matching network designed in this example. Select node `test_ckt2` under **Matching Network Browser** pane.



S11 and S21 plot of the custom pi-matching network is displayed under Cartesian plot.

- “Impedance Matching of Small Monopole Antenna”

Programmatic Use

`matchingNetworkDesigner` opens the **Matching Network Designer** app to design, visualize, and compare one-port narrowband matching networks.

`matchingNetworkDesigner(mnnetwork)` opens a matching network saved using the **Matching Network Designer** app. `mnnetwork` is a MAT file.

Tips

- Use this expression to set design constraints: |***Parameter***| ***Condition*** *10^(**Goal (dB)**/20)*. For example, |S21| > 10^(-3/20) implies that matching network circuits are sorted with S21 greater than -3 dB as a design constraint. In linear scale the expression can be rewritten as |S21| > 0.7079.

Algorithms

Color Codes in Cartesian Plot

You can use the color codes provided in this table to analyze the Cartesian plot generated in the app. By doing this, you can determine whether your matching network has satisfied the conditions you specified using the **Manage Constraints** button.

Criteria	S11 and S22	S12 and S21
Pass	Green	Cyan
Fail	Orange	Red

Note Overlapping regions will result in color mixing.

Version History

Introduced in R2021a

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

Apps
RF Budget Analyzer

Objects
`matchingnetwork`

Topics
“Impedance Matching of Small Monopole Antenna”

Properties

Polar Properties

Control appearance and behavior of polar plot

Description

Polar properties control the appearance and behavior of the polar plot function object. By changing property values, you can modify certain aspects of the polar plot. To change the default properties use:

```
p = polar(____,Name,Value)
```

To view all the properties of the polar plot function object use:

```
details(p)
```

Properties

Angle Properties

`'AngleAtTop'` — Angle at top of polar plot
`90` (default) | scalar in degrees

Angle at the top of the polar plot, specified as a comma-separated pair consisting of `'AngleAtTop'` and a scalar in degrees.

Data Types: `double`

`'AngleLim'` — Visible polar angle span
`[0 360]` (default) | 1-by-2 vector of real values

Visible polar angle span, specified as a comma-separated pair consisting of `'AngleLim'` and a 1-by-2 vector of real values.

Data Types: `double`

`'AngleLimVisible'` — Show interactive angle limit cursors
`0` (default) | `1`

Show interactive angle limit cursors, specified as a comma-separated pair consisting of `'AngleLimVisible'` and `0` or `1`.

Data Types: `logical`

`'AngleDirection'` — Direction of increasing angle
`'ccw'` (default) | `'cw'`

Direction of increasing angle, specified as a comma-separated pair consisting of `'AngleDirection'` and `'ccw'` (counterclockwise) or `'cw'` (clockwise).

Data Types: `char`

`'AngleResolution'` — Number of degrees between radial lines
`15` (default) | scalar in degrees

Number of degrees between radial lines depicting angles in the polar plot, specified as a comma-separated pair consisting of `'AngleResolution'` and a scalar in degrees.

Data Types: `double`

`'AngleTickLabelRotation'` — Rotate angle tick labels
`0` (default) | `1`

Rotate angle tick labels, specified as a comma-separated pair consisting of `'AngleTickLabelRotation'` and `0` or `1`.

Data Types: `logical`

`'AngleTickLabelVisible'` — Show angle tick labels
`1` (default) | `0`

Show angle tick labels, specified as a comma-separated pair consisting of `'AngleTickLabelVisible'` and `0` or `1`.

Data Types: `logical`

`'AngleTickLabelFormat'` — Format for angle tick labels
`360` (default) | `180`

Format for angle tick labels, specified as a comma-separated pair consisting of `'AngleTickLabelFormat'` and `360` degrees or `180` degrees.

Data Types: `double`

`'AngleFontSizeMultiplier'` — Scale factor of angle tick font
`1` (default) | numeric value greater than zero

Scale factor of angle tick font, specified as a comma-separated pair consisting of `'AngleFontSizeMultiplier'` and a numeric value greater than zero.

Data Types: `double`

`'Span'` — Show angle span measurement
`0` (default) | `1`

Show angle span measurement, specified as a comma-separated pair consisting of `'Span'` and `0` or `1`.

Data Types: `logical`

`'ZeroAngleLine'` — Highlight radial line at zero degrees
`0` (default) | `1`

Highlight radial line at zero degrees, specified as a comma-separated pair consisting of `'ZeroAngleLine'` and `0` or `1`.

Data Types: `logical`

`'DisconnectAngleGaps'` — Show gaps in line plots with nonuniform angle spacing
`1` (default) | `0`

Show gaps in line plots with nonuniform angle spacing, specified as a comma-separated pair consisting of `'DisconnectAngleGaps'` and `0` or `1`.

Data Types: `logical`

Magnitude Properties

`'MagnitudeAxisAngle'` — Angle of magnitude tick label radial line
`75` (default) | real scalar in degrees

Angle of magnitude tick label radial line, specified as a comma-separated pair consisting of `'MagnitudeAxisAngle'` and real scalar in degrees.

Data Types: `double`

`'MagnitudeTick'` — Magnitude ticks
`[0 0.2 0.4 0.6 0.8]` (default) | 1-by-N vector

Magnitude ticks, specified as a comma-separated pair consisting of `'MagnitudeTick'` and a 1-by-N vector, where N is the number of magnitude ticks.

Data Types: `double`

`'MagnitudeTickLabelVisible'` — Show magnitude tick labels
`1` (default) | `0`

Show magnitude tick labels, specified as a comma-separated pair consisting of `'MagnitudeTickLabelVisible'` and `0` or `1`.

Data Types: `logical`

`'MagnitudeLim'` — Minimum and maximum magnitude limits
`[0 1]` (default) | two-element vector of real values

Minimum and maximum magnitude limits, specified as a comma-separated pair consisting of `'MagnitudeLim'` and a two-element vector of real values.

Data Types: `double`

`'MagnitudeLimMode'` — Determine magnitude dynamic range
`'auto'` (default) | `'manual'`

Determine magnitude dynamic range, specified as a comma-separated pair consisting of `'MagnitudeLimMode'` and `'auto'` or `'manual'`.

Data Types: `char`

`'MagnitudeAxisAngleMode'` — Determine angle for magnitude tick labels
`'auto'` (default) | `'manual'`

Determine angle for magnitude tick labels, specified as a comma-separated pair consisting of `'MagnitudeAxisAngleMode'` and `'auto'` or `'manual'`.

Data Types: `char`

`'MagnitudeTickMode'` — Determine magnitude tick locations
`'auto'` (default) | `'manual'`

Determine magnitude tick locations, specified as a comma-separated pair consisting of `'MagnitudeTickMode'` and `'auto'` or `'manual'`.

Data Types: `char`

`'MagnitudeUnits'` — Magnitude units
`'dB'` | `'dBLoss'`

Magnitude units, specified as a comma-separated pair consisting of `'MagnitudeUnits'` and `'db'` or `'dBLoss'`.

Data Types: `char`

`'MagnitudeFontSizeMultiplier'` — Scale factor of magnitude tick font
`0.9000` (default) | numeric value greater than zero

Scale factor of magnitude tick font, specified as a comma-separated pair consisting of `'MagnitudeFontSizeMultiplier'` and a numeric value greater than zero.

Data Types: `double`

Miscellaneous Properties

`'NormalizeData'` — Normalize each data trace to maximum value
`0` (default) | `1`

Normalize each data trace to maximum value, specified as a comma-separated pair consisting of `'NormalizeData'` and `0` or `1`.

Data Types: `logical`

`'ConnectEndpoints'` — Connect first and last angles
`0` (default) | `1`

Connect first and last angles, specified as a comma-separated pair consisting of `'ConnectEndpoints'` and `0` or `1`.

Data Types: `logical`

`'Style'` — Style of polar plot display
`'line'` (default) | `'filled'`

Style of polar plot display, specified as a comma-separated pair consisting of `'Style'` and `'line'` or `'filled'`.

Data Types: `char`

`'TemporaryCursor'` — Create temporary cursor
`0` (default) | `1`

Create a temporary cursor, specified as a comma-separated pair consisting of `'TemporaryCursor'` and `0` or `1`.

Data Types: `logical`

`'ToolTips'` — Show tool tips
`1` (default) | `0`

Show tool tips when you hover over a polar plot element, specified as a comma-separated pair consisting of `'ToolTips'` and `0` or `1`.

Data Types: `logical`

'ClipData' — Clip data to outer circle

0 (default) | 1

Clip data to outer circle, specified as a comma-separated pair consisting of 'ClipData' and 0 or 1.

Data Types: logical

'NextPlot' — Directive on how to add next plot

'replace' (default) | 'replacechildren' | 'add'

Directive on how to add next plot, specified as a comma-separated pair consisting of 'NextPlot' and one of the values in the table:

Property Value	Effect
'replace'	Removes all axes objects and resets figure properties to their defaults before adding new graphics objects.
'replacechildren'	Removes all prior plot but preserves all axes settings.
'add'	Adds new graphics objects without clearing or resetting the current figure.

Legend and Title Properties

'LegendLabels' — Data tables for legend annotation

character vector | cell array of character vectors

Data tables for legend annotation, specified as a comma-separated pair consisting of 'LegendLabels' and a character vector or cell array of character vectors. Ⓐ denotes the active line for interactive operation.

Data Types: char

'LegendVisible' — Show legend label

0 (default) | 1

Show legend label, specified as a comma-separated pair consisting of 'LegendVisible' and 0 or 1.

Data Types: logical

'TitleTop' — Title to display above the polar plot

character vector

Title to display above the polar plot, specified as a comma-separated pair consisting of 'TitleTop' and a character vector.

Data Types: char

'TitleBottom' — Title to display below the polar plot

character vector

Title to display below the polar plot, specified as a comma-separated pair consisting of 'TitleBottom' and a character vector.

Data Types: char

`'TitleTopOffset'` — Offset between top title and angle ticks
`0.1500` (default) | scalar

Offset between top title and angle ticks, specified as a comma-separated pair consisting of `'TitleTopOffset'` and a scalar. The value must be in the range `[-0.5,0.5]`.

Data Types: `double`

`'TitleBottomOffset'` — Offset between bottom title and angle ticks
`0.1500` (default) | scalar

Offset between bottom title and angle ticks, specified as a comma-separated pair consisting of `'TitleBottomOffset'` and a scalar. The value must be in the range `[-0.5,0.5]`.

Data Types: `double`

`'TitleTopFontSizeMultiplier'` — Scale factor of top title font
`1.1000` (default) | numeric value greater than zero

Scale factor of top title font, specified as a comma-separated pair consisting of `'TitleTopFontSizeMultiplier'` and a numeric value greater than zero.

Data Types: `double`

`'TitleBottomFontSizeMultiplier'` — Scale factor of bottom title font
`0.9000` (default) | numeric value greater than zero

Scale factor of bottom title font, specified as a comma-separated pair consisting of `'TitleBottomFontSizeMultiplier'` and a numeric value greater than zero.

Data Types: `double`

`'TitleTopFontWeight'` — Thickness of top title font
`'bold'` (default) | `'normal'`

Thickness of top title font, specified as a comma-separated pair consisting of `'TitleTopFontWeight'` and `'bold'` or `'normal`.

Data Types: `char`

`'TitleBottomFontWeight'` — Thickness of bottom title font
`'normal'` (default) | `'bold'`

Thickness of bottom title font, specified as a comma-separated pair consisting of `'TitleBottomFontWeight'` and `'bold'` or `'normal`.

Data Types: `char`

`'TitleTopTextInterpreter'` — Interpretation of top title characters
`'none'` (default) | `'tex'` | `'latex'`

Interpretation of top title characters, specified as a comma-separated pair consisting of `'TitleTopTextInterpreter'` and:

- `'tex'` — Interpret using a subset of TeX markup
- `'latex'` — Interpret using LaTeX markup

- `'none'` — Display literal characters

TeX Markup

By default, MATLAB supports a subset of TeX markup. Use TeX markup to add superscripts and subscripts, modify the text type and color, and include special characters in the text.

This table lists the supported modifiers when the `TickLabelInterpreter` property is set to `'tex'`, which is the default value. Modifiers remain in effect until the end of the text, except for superscripts and subscripts which only modify the next character or the text within the curly braces `{}`.

Modifier	Description	Example
`^{ }`	Superscript	`'text^{superscript}'`
`_{ }`	Subscript	`'text_{subscript}'`
`\bf`	Bold font	`'\bf text'`
`\it`	Italic font	`'\it text'`
`\sl`	Oblique font (rarely available)	`'\sl text'`
`\rm`	Normal font	`'\rm text'`
`\fontname{specifier}`	Set `specifier` as the name of a font family to change the font style. You can use this with other modifiers.	`'\fontname{Courier} text'`
`\fontsize{specifier}`	Set `specifier` as a scalar numeric value to change the font size.	`'\fontsize{15} text'`
`\color{specifier}`	Set `specifer` as one of these colors: `red`, `green`, `yellow`, `magenta`, `blue`, `black`, `white`, `gray`, `darkGreen`, `orange`, or `lightBlue`.	`'\color{magenta} text'`
`\color[rgb]{specifier}`	Set `specifier` as a three-element RGB triplet to change the font color.	`'\color[rgb]{0,0.5,0.5} text'`

LaTeX Markup

To use LaTeX markup, set the `TickLabelInterpreter` property to `'latex'`. The displayed text uses the default LaTeX font style. The `FontName`, `FontWeight`, and `FontAngle` properties do not have an effect. To change the font style, use LaTeX markup within the text.

The maximum size of the text that you can use with the LaTeX interpreter is 1200 characters. For multiline text, the maximum size reduces by about 10 characters per line.

Data Types: `char`

`'TitleBottomTextInterpreter'` — Interpretation of bottom title characters

`'none'` (default) | `'tex'` | `'latex'`

Interpretation of bottom title characters, specified as a comma-separated pair consisting of `'TitleBottomTextInterpreter'` and:

- `'tex'` — Interpret using a subset of TeX markup
- `'latex'` — Interpret using LaTeX markup
- `'none'` — Display literal characters

TeX Markup

By default, MATLAB supports a subset of TeX markup. Use TeX markup to add superscripts and subscripts, modify the text type and color, and include special characters in the text.

This table lists the supported modifiers when the `TickLabelInterpreter` property is set to `'tex'`, which is the default value. Modifiers remain in effect until the end of the text, except for superscripts and subscripts which only modify the next character or the text within the curly braces `{}`.

Modifier	Description	Example
`^{ }`	Superscript	`'text^{superscript}'`
`_{ }`	Subscript	`'text_{subscript}'`
`\bf`	Bold font	`'\bf text'`
`\it`	Italic font	`'\it text'`
`\sl`	Oblique font (rarely available)	`'\sl text'`
`\rm`	Normal font	`'\rm text'`
`\fontname{specifier}`	Set `specifier` as the name of a font family to change the font style. You can use this with other modifiers.	`'\fontname{Courier} text'`
`\fontsize{specifier}`	Set `specifier` as a scalar numeric value to change the font size.	`'\fontsize{15} text'`
`\color{specifier}`	Set `specifer` as one of these colors: `red`, `green`, `yellow`, `magenta`, `blue`, `black`, `white`, `gray`, `darkGreen`, `orange`, or `lightBlue`.	`'\color{magenta} text'`
`\color[rgb]{specifier}`	Set `specifier` as a three-element RGB triplet to change the font color.	`'\color[rgb]{0,0.5,0.5} text'`

LaTeX Markup

To use LaTeX markup, set the `TickLabelInterpreter` property to `'latex'`. The displayed text uses the default LaTeX font style. The `FontName`, `FontWeight`, and `FontAngle` properties do not have an effect. To change the font style, use LaTeX markup within the text.

The maximum size of the text that you can use with the LaTeX interpreter is 1200 characters. For multiline text, the maximum size reduces by about 10 characters per line.

Data Types: `char`

Grid Properties

`'GridOverData'` — Draw grid over data plots
`0` (default) | `1`

Draw grid over data plots, specified as a comma-separated pair consisting of `'GridOverData'` and `0` or `1`.

Data Types: `logical`

`'DrawGridToOrigin'` — Draw radial lines within innermost circle
`0` (default) | `1`

Draw radial lines within innermost circle of the polar plot, specified as a comma-separated pair consisting of `'DrawGridToOrigin'` and `0` or `1`.

Data Types: `logical`

`'GridAutoRefinement'` — Increase angle resolution
`0` (default) | `1`

Increase angle resolution in the polar plot, specified as a comma-separated pair consisting of `'GridAutoRefinement'` and `0` or `1`. This property increases angle resolution by doubling the number of radial lines outside each magnitude.

Data Types: `logical`

`'GridWidth'` — Width of grid lines
`0.5000` (default) | positive scalar

Width of grid lines, specified as a comma-separated pair consisting of `'GridWidth'` and a positive scalar.

Data Types: `double`

`'GridVisible'` — Show grid lines
`1` (default) | `0`

Show grid lines, including magnitude circles and angle radii, specified as a comma-separated pair consisting of `'GridVisible'` and `0` or `1`.

Data Types: `logical`

`'GridForeGroundColor'` — Color of foreground grid lines
`[0.8000 0.8000 0.8000]` (default) | `'none'` | character vector of color names

Color of foreground grid lines, specified as a comma-separated pair consisting of `'GridForeGroundColor'` and an RGB triplet, character vector of color names, or `'none'`.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
"red"	"r"	[1 0 0]	"#FF0000"	
"green"	"g"	[0 1 0]	"#00FF00"	
"blue"	"b"	[0 0 1]	"#0000FF"	
"cyan"	"c"	[0 1 1]	"#00FFFF"	
"magenta"	"m"	[1 0 1]	"#FF00FF"	
"yellow"	"y"	[1 1 0]	"#FFFF00"	
"black"	"k"	[0 0 0]	"#000000"	
"white"	"w"	[1 1 1]	"#FFFFFF"	

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	"#0072BD"	
[0.8500 0.3250 0.0980]	"#D95319"	
[0.9290 0.6940 0.1250]	"#EDB120"	
[0.4940 0.1840 0.5560]	"#7E2F8E"	
[0.4660 0.6740 0.1880]	"#77AC30"	
[0.3010 0.7450 0.9330]	"#4DBEEE"	
[0.6350 0.0780 0.1840]	"#A2142F"	

Data Types: `double` | `char`

`'GridBackGroundColor'` — Color of background grid lines

`'w'` (default) | character vector of color names | `'none'`

Color of background grid lines, specified as a comma-separated pair consisting of `'GridBackGroundColor'` and an RGB triplet, character vector of color names, or `'none'`.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
"red"	"r"	[1 0 0]	"#FF0000"	

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
"green"	"g"	[0 1 0]	"#00FF00"	
"blue"	"b"	[0 0 1]	"#0000FF"	
"cyan"	"c"	[0 1 1]	"#00FFFF"	
"magenta"	"m"	[1 0 1]	"#FF00FF"	
"yellow"	"y"	[1 1 0]	"#FFFF00"	
"black"	"k"	[0 0 0]	"#000000"	
"white"	"w"	[1 1 1]	"#FFFFFF"	

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	"#0072BD"	
[0.8500 0.3250 0.0980]	"#D95319"	
[0.9290 0.6940 0.1250]	"#EDB120"	
[0.4940 0.1840 0.5560]	"#7E2F8E"	
[0.4660 0.6740 0.1880]	"#77AC30"	
[0.3010 0.7450 0.9330]	"#4DBEEE"	
[0.6350 0.0780 0.1840]	"#A2142F"	

Data Types: `double` | `char`

Marker, Color, Line, and Font Properties

`'Marker'` — Marker symbol

`'none'` (default) | character vector of symbols

Marker symbol, specified as a comma-separated pair consisting of `'Marker'` and either `'none'` or one of the symbols in this table. By default, a line does not have markers. Add markers at selected points along the line by specifying a marker.

Marker	Description	Resulting Marker
"o"	Circle	○
"+"	Plus sign	+
"*"	Asterisk	✳
"."	Point	•
"x"	Cross	×
"_"	Horizontal line	—

Marker	Description	Resulting Marker
`"|"`	Vertical line	
`"square"`	Square	
`"diamond"`	Diamond	
`"^"`	Upward-pointing triangle	
`"v"`	Downward-pointing triangle	
`">"`	Right-pointing triangle	
`"<"`	Left-pointing triangle	
`"pentagram"`	Pentagram	
`"hexagram"`	Hexagram	
`"none"`	No markers	Not applicable

`'MarkerSize'` — Marker size
6 (default) | positive value

Marker size, specified as a comma-separated pair consisting of `'MarkerSize'` and a positive value in point units.

Data Types: `double`

`'ColorOrder'` — Colors to use for multiline plots
seven predefined colors (default) | three-column matrix of RGB triplets

Colors to use for multiline plots, specified as a comma-separated pair consisting of `'ColorOrder'` and a three-column matrix of RGB triplets. Each row of the matrix defines one color in the color order.

Data Types: `double`

`'ColorOrderIndex'` — Next color to use in color order
1 (default) | positive integer

Next color to use in color order, specified as a comma-separated pair consisting of `'ColorOrderIndex'` and a positive integer. New plots added to the axes use colors based on the current value of the color order index.

Data Types: `double`

`'EdgeColor'` — Color of data lines
`'k'` (default) | RGB triplet vector

Color of data lines, specified as a comma-separated pair consisting of `'EdgeColor'` and a character vector of color names or RGB triplet vector.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`"red"`	`"r"`	`[1 0 0]`	`"#FF0000"`	
`"green"`	`"g"`	`[0 1 0]`	`"#00FF00"`	
`"blue"`	`"b"`	`[0 0 1]`	`"#0000FF"`	
`"cyan"`	`"c"`	`[0 1 1]`	`"#00FFFF"`	
`"magenta"`	`"m"`	`[1 0 1]`	`"#FF00FF"`	
`"yellow"`	`"y"`	`[1 1 0]`	`"#FFFF00"`	
`"black"`	`"k"`	`[0 0 0]`	`"#000000"`	
`"white"`	`"w"`	`[1 1 1]`	`"#FFFFFF"`	

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`"#0072BD"`	
`[0.8500 0.3250 0.0980]`	`"#D95319"`	
`[0.9290 0.6940 0.1250]`	`"#EDB120"`	
`[0.4940 0.1840 0.5560]`	`"#7E2F8E"`	
`[0.4660 0.6740 0.1880]`	`"#77AC30"`	
`[0.3010 0.7450 0.9330]`	`"#4DBEEE"`	
`[0.6350 0.0780 0.1840]`	`"#A2142F"`	

Data Types: `double` | `char`

`'LineStyle'` — Line style of the plot

`'-'` (default) | `'--'` | `':'` | `'-.'` | `'none'`

Line style of the plot, specified as a comma-separated pair consisting of `'LineStyle'` and one of the symbols in the table:

Symbol	Line Style	Resulting Line
`'-'`	Solid line	
`'--'`	Dashed line	

Symbol	Line Style	Resulting Line
`':'`	Dotted line	.....
`'-.'`	Dash-dotted line	_._._._.
`'none'`	No line	No line

`'LineWidth'` — Line width of plot
`2` (default) | positive scalar | positive vector

Line width of the plot, specified as a comma-separated pair consisting of `'LineWidth'` and a positive scalar or vector.

`'FontSize'` — Font size of text in plot
`10` (default) | positive scalar

Font size of text in the plot, specified as a comma-separated pair consisting of `'FontSize'` and a positive scalar.

`'FontSizeAutoMode'` — Set font size
`'auto'` (default) | `'manual'`

Set font size, specified as a comma-separated pair consisting of `'FontSizeAutoMode'` and `'auto'` or `'manual'`.

Data Types: `char`

See Also

SmithPlot Properties

Control appearance and behavior of Smith chart

Description

Smith chart properties control the appearance and behavior of the Smith plot object. By changing property values, you can modify certain aspects of the Smith chart. To change the default properties use: .

```
s = smithplot(____,Name,Value)
```

To view all the properties of the Smith plot object use:

```
details(s)
```

Properties

Display

`ClipData` — Clip data to outer circle
`1` (default) | `0`

Clip data to outer circle, specified as `0` or `1`.

Data Types: `logical`

`ColorOrder` — Colors to use for multiline plots
seven predefined colors (default) | three-column matrix of RGB triplets

Colors to use for multi-line plots, specified as three-column matrix of RGB triplets. Each row of the matrix defines one color in the color order.

For more information, see `ColorOrder` in the Axes.

Data Types: `double`

`ColorOrderIndex` — Next color to use in color order
`1` (default) | positive integer

Next color to use in color order, specified as a positive integer. New plots added to the axes use colors based on the current value of the color order index.

For more information, see `ColorOrderIndex` in the Axes.

Data Types: `double`

`FontName` — Font name
`'Helvetica'` (default) | character vector

Font name, specified as a character vector.

Note

- To display and print text properly, you must choose a font that your system supports. The default font depends on your operating system and locale.
- To use a fixed-width font that looks good in any locale, use `FixedWidth`. The fixed-width font relies on the root `FixedWidthFontName` property.
- The `listfonts` function generates list of available font names.

Data Types: `char`

`FontSize` — Font size

`10` (default) | positive integer

Font size, specified as a positive integer.

Data Types: `double`

`FontSizeMode` — Changes the font size based on window size

`'auto'` (default) | 'manual'

Font size mode, specified as `'auto'`. Changes the font size based on window size.

Data Types: `char`

`GridBackgroundColor` — Background grid line color

`'w'` (default) | character vector of color names | `'none'`

Background gird line color, specified as an RGB triplet, or as a character vector of color names, or `'none'`. Using `'none'` turns off the grid completely.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (#) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`"red"`	`"r"`	`[1 0 0]`	`"#FF0000"`	
`"green"`	`"g"`	`[0 1 0]`	`"#00FF00"`	
`"blue"`	`"b"`	`[0 0 1]`	`"#0000FF"`	
`"cyan"`	`"c"`	`[0 1 1]`	`"#00FFFF"`	
`"magenta"`	`"m"`	`[1 0 1]`	`"#FF00FF"`	
`"yellow"`	`"y"`	`[1 1 0]`	`"#FFFF00"`	

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
"black"	"k"	[0 0 0]	"#000000"	
"white"	"w"	[1 1 1]	"#FFFFFF"	

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	"#0072BD"	
[0.8500 0.3250 0.0980]	"#D95319"	
[0.9290 0.6940 0.1250]	"#EDB120"	
[0.4940 0.1840 0.5560]	"#7E2F8E"	
[0.4660 0.6740 0.1880]	"#77AC30"	
[0.3010 0.7450 0.9330]	"#4DBEEE"	
[0.6350 0.0780 0.1840]	"#A2142F"	

Data Types: `char` | `double`

`GridForegroundColor` — Foreground grid line color
`[0.4000 0.4000 0.4000]` (default) | `'none'` | character vector of color names

Foreground grid line color, specified as RGB triplet, or as a character vector of color names, or `'none'`.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
"red"	"r"	[1 0 0]	"#FF0000"	
"green"	"g"	[0 1 0]	"#00FF00"	
"blue"	"b"	[0 0 1]	"#0000FF"	
"cyan"	"c"	[0 1 1]	"#00FFFF"	
"magenta"	"m"	[1 0 1]	"#FF00FF"	
"yellow"	"y"	[1 1 0]	"#FFFF00"	
"black"	"k"	[0 0 0]	"#000000"	

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
"white"	"w"	[1 1 1]	"#FFFFFF"	

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	"#0072BD"	
[0.8500 0.3250 0.0980]	"#D95319"	
[0.9290 0.6940 0.1250]	"#EDB120"	
[0.4940 0.1840 0.5560]	"#7E2F8E"	
[0.4660 0.6740 0.1880]	"#77AC30"	
[0.3010 0.7450 0.9330]	"#4DBEEE"	
[0.6350 0.0780 0.1840]	"#A2142F"	

Data Types: `double` | `char`

`GridLineStyle` — Grid line style
`'-'` (default) | `'--'` | `':'` | `'-.'` | `'none'`

Grid line style, specified as one of the following:

Line Style	Description	Resulting Line
'-'	Solid line	–––––––
'- -'	Dashed line	- - - - - - - - -
':'	Dotted line	.....
'- .'	Dash-dotted line	- . - . - .
'none'	No line	No line

Data Types: `char`

`GridLineWidth` — Grid line width
`'0.5000'` (default) | positive scalar

Grid line width, specified as positive scalar.

Data Types: `double`

`GridOverData` — Draw grid over data plots
`0` (default) | `1`

Draw grid over data plots, specified as `0` or `1`.

Data Types: `logical`

`GridSubForegroundColor` — Sub-foreground grid lines color
`[0.8000 0.8000 0.8000]` (default) | `'none'` | character vector of color names

Sub foreground grid lines color, specified as an RGB triplet, character vector of color names, or `'none'`.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`"red"`	`"r"`	`[1 0 0]`	`"#FF0000"`	
`"green"`	`"g"`	`[0 1 0]`	`"#00FF00"`	
`"blue"`	`"b"`	`[0 0 1]`	`"#0000FF"`	
`"cyan"`	`"c"`	`[0 1 1]`	`"#00FFFF"`	
`"magenta"`	`"m"`	`[1 0 1]`	`"#FF00FF"`	
`"yellow"`	`"y"`	`[1 1 0]`	`"#FFFF00"`	
`"black"`	`"k"`	`[0 0 0]`	`"#000000"`	
`"white"`	`"w"`	`[1 1 1]`	`"#FFFFFF"`	

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`"#0072BD"`	
`[0.8500 0.3250 0.0980]`	`"#D95319"`	
`[0.9290 0.6940 0.1250]`	`"#EDB120"`	
`[0.4940 0.1840 0.5560]`	`"#7E2F8E"`	
`[0.4660 0.6740 0.1880]`	`"#77AC30"`	
`[0.3010 0.7450 0.9330]`	`"#4DBEEE"`	
`[0.6350 0.0780 0.1840]`	`"#A2142F"`	

Data Types: `char` | `double`

`GridSubLineStyle` — Subgrid line style

`'- .'` (default) | `'--'` | `':'` | `'-'` | `'none'`

Subgrids line style, specified as one of the following:

Line Style	Description	Resulting Line
'-'	Solid line	–––––––
'- -'	Dashed line	- - - - - - - - -
':'	Dotted line	.....
'- .'	Dash-dotted line	- . - . - .
'none'	No line	No line

Data Types: `char`

`GridSubLineWidth` — Subgrid line width
`'0.5000'` (default) | positive scalar

Subgrid line width, specified as positive scalar.

Data Types: `double`

`GridType` — Grid type
`'Z'` (default) | `'Y'` | `'ZY'` | `'YZ'`

Grid type, specified as `'Z'`, `'Y'`, `'ZY'`, `'YZ'`. Grid type specifies if the plot is an admittance plot, impedance plot, or both.

Data Types: `char`

`GridValue` — Defines constant resistance circles and constant reactance arcs
`[30.0 5.0 2.0 1.0 0.5 0.2; Inf 30.0 5.0 5.0 2.0 1.0]` (default)

Two-row matrix. Row 1 specifies the values of the constant resistance circles and constant reactance arcs in the chart. Row 2 specifies the value at which the corresponding arcs and circles defined in Row 1 end.

Data Types: `double`

`GridVisible` — Show grid on Smith chart
`'1'` (default) | `'0'`

Show grid on Smith chart, specified as `'1'` or `'0'`.

Data Types: `logical`

`NextPlot` — Directive on how to add next plot
`'replace'` (default) | `'replacechildren'` | `'add'`

Directive on how to add next plot, specified as a comma-separated pair consisting of `'NextPlot'` and one of the values in the table:

Property Value	Effect
`'replace'`	Removes all axes objects and resets figure properties to their defaults before adding new graphics objects.
`'replacechildren'`	Removes all prior plot but preserves all axes settings.

Property Value	Effect
`'add'`	Adds new graphics objects without clearing or resetting the current figure.

`Parent` — Figure parent
root object

Figure parent, returned as a root object.

`TitleBottom` — Title to display below Smith chart
character vector

Title to display below the Smith chart, specified as a character vector.

Data Types: `char`

`TitleBottomFontSizeMultiplier` — Bottom title font scale factor
`0.9000` (default) | numeric value greater than zero

Bottom title font scale factor, specified as a numeric value greater than zero.

Data Types: `double`

`TitleBottomFontWeight` — Bottom title font thickness
`'normal'` (default) | `'bold'`

Bottom title font thickness, specified as `'bold'` or `'normal`.

Data Types: `char`

`TitleBottomOffset` — Offset between bottom title and arc ticks
`0.1500` (default) | scalar

Offset between bottom title and angle ticks,specified as a scalar. The value must be in the range `[-0.5,0.5]`.

Data Types: `double`

`TitleBottomTextInterpreter` — Interpretation of bottom title characters
`'none'` (default) | `'tex'` | `'latex'`

Interpretation of bottom title characters, specified one of the following:

- `'tex'` — Interpret using a subset of TeX markup
- `'latex'` — Interpret using LaTeX markup
- `'none'` — Display literal characters

TeX Markup

By default, MATLAB supports a subset of TeX markup. Use TeX markup to add superscripts and subscripts, modify the text type and color, and include special characters in the text.

This table lists the supported modifiers when the `TickLabelInterpreter` property is set to `'tex'`, which is the default value. Modifiers remain in effect until the end of the text, except for superscripts and subscripts which only modify the next character or the text within the curly braces `{}`.

Modifier	Description	Example
`^{ }`	Superscript	`'text^{superscript}'`
`_{ }`	Subscript	`'text_{subscript}'`
`\bf`	Bold font	`'\bf text'`
`\it`	Italic font	`'\it text'`
`\sl`	Oblique font (rarely available)	`'\sl text'`
`\rm`	Normal font	`'\rm text'`
`\fontname{specifier}`	Set `specifier` as the name of a font family to change the font style. You can use this markup with other modifiers.	`'\fontname{Courier} text'`
`\fontsize{specifier}`	Set `specifier` as a scalar numeric value to change the font size.	`'\fontsize{15} text'`
`\color{specifier}`	Set `specifier` as one of these colors: `red`, `green`, `yellow`, `magenta`, `blue`, `black`, `white`, `gray`, `darkGreen`, `orange`, or `lightBlue`.	`'\color{magenta} text'`
`\color[rgb]{specifier}`	Set `specifier` as a three-element RGB triplet to change the font color.	`'\color[rgb]{0,0.5,0.5} text'`

LaTeX Markup

To use LaTeX markup, set the `TickLabelInterpreter` property to `'latex'`. The displayed text uses the default LaTeX font style. To change the font style, use LaTeX markup within the text.

The maximum size of the text that you can use with the LaTeX interpreter is 1200 characters. For multiline text, the maximum size reduces by about 10 characters per line.

Data Types: `char`

`TitleTop` — Title to display above the Smith chart

character vector

Title to display above the Smith chart, specified as a character vector.

Data Types: `char`

`TitleTopFontSizeMultiplier` — Top title font scale factor

`1.1000` (default) | numeric value greater than zero

Top title font scale factor, specified as a numeric value greater than zero.

Data Types: `double`

`TitleTopFontWeight` — Top title font thickness

`'bold'` (default) | `'normal'`

Top title font thickness, specified as `'bold'` or `'normal`.

Data Types: `char`

`TitleTopOffset` — Offset between top title and arc ticks

`0.1500` (default) | scalar

Offset between top title and angle ticks, specified as a scalar. The value must be in the range `[-0.5,0.5]`.

Data Types: `double`

`TitleTopTextInterpreter` — Interpretor of top title characters

`'none'` (default) | `'tex'` | `'latex'`

Interpretation of top title characters, specified one of the following:

- `'tex'` — Interpret using a subset of TeX markup
- `'latex'` — Interpret using LaTeX markup
- `'none'` — Display literal characters

TeX Markup

By default, MATLAB supports a subset of TeX markup. Use TeX markup to add superscripts and subscripts, modify the text type and color, and include special characters in the text.

This table lists the supported modifiers when the `TickLabelInterpreter` property is set to `'tex'`, which is the default value. Modifiers remain in effect until the end of the text, except for superscripts and subscripts which only modify the next character or the text within the curly braces `{}`.

Modifier	Description	Example
`^{ }`	Superscript	`'text^{superscript}'`
`_{ }`	Subscript	`'text_{subscript}'`
`\bf`	Bold font	`'\bf text'`
`\it`	Italic font	`'\it text'`
`\sl`	Oblique font (rarely available)	`'\sl text'`
`\rm`	Normal font	`'\rm text'`
`\fontname{specifier}`	Set `specifier` as the name of a font family to change the font style. You can use this markup with other modifiers.	`'\fontname{Courier} text'`
`\fontsize{specifier}`	Set `specifier` as a scalar numeric value to change the font size.	`'\fontsize{15} text'`
`\color{specifier}`	Set `specifer` as one of these colors: `red`, `green`, `yellow`, `magenta`, `blue`, `black`, `white`, `gray`, `darkGreen`, `orange`, or `lightBlue`.	`'\color{magenta} text'`

Modifier	Description	Example
`\color[rgb]{specifier}`	Set `specifier` as a three-element RGB triplet to change the font color.	`'\color[rgb]{0,0.5,0.5} text'`

LaTeX Markup

To use LaTeX markup, set the `TickLabelInterpreter` property to `'latex'`. The displayed text uses the default LaTeX font style. To change the font style, use LaTeX markup within the text.

The maximum size of the text that you can use with the LaTeX interpreter is 1200 characters. For multi-line text, the maximum size reduces by about 10 characters per line.

Data Types: `char`

`View` — View section of Smith plot

`'full'` (default) | `'top'` | `'bottom'` | `'left'` | `'right'` | `'top-left'` | `'top-right'` | `'bottom-left'` | `'bottom-right'`

View the full Smith plot or a section of the plot by selecting one of the values in this table.

Value	View
`'full'`	Full Smith plot
`'top'`	Top half of the Smith plot
`'bottom'`	Bottom half of the Smith plot
`'left'`	Left half of the Smith plot
`'right'`	Right half of the Smith plot
`'top-left'`	Top left of the Smith plot
`'top-right'`	Top right of the Smith plot
`'bottom-left'`	Bottom left of the Smith plot
`'bottom-right'`	Bottom right of the Smith plot

Example: `s.View = 'top-left'`

Data Types: `char` | `string`

Datasets

`EdgeColor` — Data line color

`'k'` (default) | RGB triplet vector

Data line color, specified as a character vector of color names or as an RGB triplet vector.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
"red"	"r"	[1 0 0]	"#FF0000"	
"green"	"g"	[0 1 0]	"#00FF00"	
"blue"	"b"	[0 0 1]	"#0000FF"	
"cyan"	"c"	[0 1 1]	"#00FFFF"	
"magenta"	"m"	[1 0 1]	"#FF00FF"	
"yellow"	"y"	[1 1 0]	"#FFFF00"	
"black"	"k"	[0 0 0]	"#000000"	
"white"	"w"	[1 1 1]	"#FFFFFF"	

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	"#0072BD"	
[0.8500 0.3250 0.0980]	"#D95319"	
[0.9290 0.6940 0.1250]	"#EDB120"	
[0.4940 0.1840 0.5560]	"#7E2F8E"	
[0.4660 0.6740 0.1880]	"#77AC30"	
[0.3010 0.7450 0.9330]	"#4DBEEE"	
[0.6350 0.0780 0.1840]	"#A2142F"	

Data Types: `double` | `char`

`LegendLabels` — Data tables for legend annotation

character vector | cell array of character vectors

Data tables for legend annotation, specified as a character vector or as a cell array of character vectors.

Data Types: `char`

`LegendVisible` — Show legend label

`1` (default) | `0`

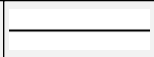
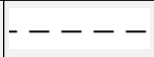
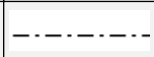
Show legend label, specified as `0` or `1`.

Data Types: `logical`

`LineStyle` — Plot line style

`'-'` (default) | `'--'` | `':'` | `'-.'` | `'none'`

Plot line style, specified as one of the symbols in the table:

Symbol	Line Style	Resulting Line
'-'	Solid line	
'--'	Dashed line	
':'	Dotted line	
'-.'	Dash-dotted line	
'none'	No line	No line

`LineWidth` — Plot line width
1 (default) | positive scalar | vector

Plot line width, specified as a positive scalar or vector.

`Marker` — Marker symbol
`'none'` (default) | character vector of symbols

Marker symbol, specified as `'none'` or one of the symbols in this table. By default, a line does not have markers. Add markers at selected points along the line by specifying a marker.

Marker	Description	Resulting Marker
"o"	Circle	
"+"	Plus sign	
"*"	Asterisk	
"."	Point	
"x"	Cross	
"_"	Horizontal line	
"|"	Vertical line	
"square"	Square	
"diamond"	Diamond	
"^"	Upward-pointing triangle	
"v"	Downward-pointing triangle	
">"	Right-pointing triangle	
"<"	Left-pointing triangle	
"pentagram"	Pentagram	

Marker	Description	Resulting Marker
"hexagram"	Hexagram	
"none"	No markers	Not applicable

MarkerSize — Marker size

6 (default) | positive value

Marker size, specified as a positive value in points.

Data Types: double

Arcs

ArcFontSizeMultiplier — Arc tick font scale factor

1 (default) | numeric value greater than zero

Arc tick font scale factor, specified as a numeric value greater than zero.

Data Types: double

ArcTickLabelColor — Arc tick labels

'k' (default) | RGB triplet vector

Arc tick labels, specified as a character vector of color names or as an RGB triplet vector.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1]; for example, [0.4 0.6 0.7].
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (#) followed by three or six hexadecimal digits, which can range from 0 to F. The values are not case sensitive. Thus, the color codes '#FF8800', '#ff8800', '#F80', and '#f80' are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
"red"	"r"	[1 0 0]	"#FF0000"	
"green"	"g"	[0 1 0]	"#00FF00"	
"blue"	"b"	[0 0 1]	"#0000FF"	
"cyan"	"c"	[0 1 1]	"#00FFFF"	
"magenta"	"m"	[1 0 1]	"#FF00FF"	
"yellow"	"y"	[1 1 0]	"#FFFF00"	
"black"	"k"	[0 0 0]	"#000000"	
"white"	"w"	[1 1 1]	"#FFFFFF"	

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	"#0072BD"	
[0.8500 0.3250 0.0980]	"#D95319"	
[0.9290 0.6940 0.1250]	"#EDB120"	
[0.4940 0.1840 0.5560]	"#7E2F8E"	
[0.4660 0.6740 0.1880]	"#77AC30"	
[0.3010 0.7450 0.9330]	"#4DBEEE"	
[0.6350 0.0780 0.1840]	"#A2142F"	

Data Types: `char` | `double`

`ArcTickLabelVisible` — Show arc tick labels

`1` (default) | `0`

Show arc tick labels, specified as `0` or `1`.

Data Types: `logical`

Circles

`CircleFontSizeMultiplier` — Circle tick font scale factor

`0.9000` (default) | numeric value greater than zero

Circle tick font scale factor, specified as a numeric value greater than zero.

Data Types: `double`

`CircleTickLabelColor` — Circle tick label color

`'k'` (default) | RGB triplet vector

Circle tick labels color, specified as a character vector of color names or as an RGB triplet vector.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
"red"	"r"	[1 0 0]	"#FF0000"	
"green"	"g"	[0 1 0]	"#00FF00"	
"blue"	"b"	[0 0 1]	"#0000FF"	
"cyan"	"c"	[0 1 1]	"#00FFFF"	

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
"magenta"	"m"	[1 0 1]	"#FF00FF"	
"yellow"	"y"	[1 1 0]	"#FFFF00"	
"black"	"k"	[0 0 0]	"#000000"	
"white"	"w"	[1 1 1]	"#FFFFFF"	

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	"#0072BD"	
[0.8500 0.3250 0.0980]	"#D95319"	
[0.9290 0.6940 0.1250]	"#EDB120"	
[0.4940 0.1840 0.5560]	"#7E2F8E"	
[0.4660 0.6740 0.1880]	"#77AC30"	
[0.3010 0.7450 0.9330]	"#4DBEEE"	
[0.6350 0.0780 0.1840]	"#A2142F"	

Data Types: `double` | `char`

`CircleTickLabelVisible` — Show circle tick labels

1 (default) | `0`

Show arc tick labels, specified as `0` or `1`.

Data Types: `logical`

See Also

`smithplot`

System Objects

rfsystem

Perform circuit envelope simulation of RF system designed using `rfbudget`

Description

Use the `rfsystem` System object™ to perform circuit envelope simulation of an RF system designed using an `rfbudget` object. You can use the `rfsystem` object to generate an RF Blockset™ model. This object supports vector inputs and has no frame-size limits.

Note

- You can add or delete RF Blockset blocks to this model but you cannot modify the parameters of Inport and Outport blocks. After this update, the input `rfbudget` object to the `rfsystem` will be preserved and you can inspect this `rfbudget` object using the **RF Budget Analyzer** app.

To perform circuit envelope simulation of an RF system:

1. Create the `rfsystem` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
rfs = rfsystem(rfb)
rfs = rfsystem(rfb,Name=Value)
```

Description

`rfs = rfsystem(rfb)` creates a System object from the RF system `rfb`. Use the System object, `rfs`, to perform circuit envelope simulation.

The System object generates an untitled RF Blockset model of the RF system. Use “Object Functions” on page 7-5 to open, save, close, or hide the RF Blockset model.

`rfs = rfsystem(rfb,Name=Value)` sets “Properties” on page 7-2 using name-value arguments. For example, `rfsystem(rfb,'ModelName'='rfmodel')` sets the name of the RF Blockset model to `rfmodel`.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`'ModelName'` — Name of RF Blockset model
`'untitled'` (default) | string scalar | character vector

Name of the RF Blockset model, specified as a string scalar or character vector.

Example: `'ModelName','RFModel'`

`'InputFrequency'` — Input frequency
`rfb.InputFrequency'` (default) | row vector

Input frequency applied to each input of the RF system, specified as a row vector in Hz. This input frequency is derived from the RF system designed using an `rfbudget` object. Your RF system will have two inputs, I and Q, at DC if the `rfb.InputFrequency` is set to `0`.

Note You can set `InputFrequency` to be a vector if you want to investigate the intermodulation of nearby signals going through the same architecture after construction of the `rfsystem` system object.

`'OutputFrequency'` — Output frequency
`rfb.OutputFrequency(:,end)'` (default) | row vector

Output frequency to be computed at the output of the RF system, specified as a row vector in Hz. This output frequency is derived from the RF system designed using an `rfbudget` object. Your RF system will have two outputs, I and Q, at DC if the value of the `rfb.OutputFrequency` is `0`.

`'SampleTime'` — Time step for circuit envelope simulation
`1/rfb.SignalBandwidth/8` (default) | positive scalar

Time step for circuit envelope simulation, specified as a positive scalar. This property sets the step size between simulations.

`'Rx'` — Number of input chains
`1` (default) | positive scalar

Number of input chains in a multiple-input single-output receiver system, specified as a positive scalar.

Note You can set:

- Either the `'Rx'` or the `'Tx'` property when creating the System object.
- `'Rx'` property only when creating the System object.

`'Tx'` — Number of output chains
`1` (default) | positive scalar

Number of output chains in a single-input multiple-output transmitter system, specified as a positive scalar.

Note You can set:

- Either the `'Rx'` or the `'Tx'` property when creating the System object.
- `'Rx'` property only when creating the System object.

`'SLInputs'` — Number of optional Simulink inputs
`0` (default) | positive scalar

Number of optional Simulink® inputs, specified as a positive scalar. The Simulink inputs are added as Inport block in the model.

By default, the **Signal Type** parameter in the Simulink Inport blocks is set to `complex`. To change this complex signal to real, either add a Complex to Real-Imag block to your model, or in the Simulink Inport block, on the **Signal Attributes** tab set the **Signal Type** parameter `real`.

`'SLOutputs'` — Number of optional Simulink outputs
`0` (default) | positive scalar

Number of optional Simulink outputs, specified as a positive scalar. The Simulink outputs are added as Outport blocks in the model.

By default, the **Signal Type** parameter in the Simulink Outport block is set to `complex`. To change this complex signal to real, either add a Complex to Real-Imag block to your model, or in the Simulink Inport block, on the **Signal Attributes** tab set the **Signal Type** parameter `real`.

`'RFInputs'` — Number of RF inputs
nonnegative scalar

This property is read-only.

Number of RF inputs implemented as RF Blockset Inport blocks in the generated model, returned as a nonnegative scalar. The value of this property depends on the 'Rx' property and the first value in the 'InputFrequency' row vector.

`'RFOutputs'` — Number of RF outputs
nonnegative scalar

This property is read-only.

Number of RF outputs implemented as RF Blockset Outport blocks in the generated model, returned as a nonnegative scalar. The value of this property depends on the 'Tx' property and the last value in the 'OutputFrequency' row vector.

Usage

Syntax

```
out = rfs(in)
```

Description

`out = rfs(in)` creates an RF Blockset circuit envelope simulation output `out` using input signal values `in`. Pass `in` as an input argument to an automatically-generated RF Blockset model.

You can design four architectures, RF to RF, DC to RF, RF to DC, and DC to DC, using the `rfsystem` object. For more information, see “Design RF-RF, IQ-RF, RF-IQ, and IQ-IQ Architectures” on page 7-7.

Note Passing multiple input vectors and concatenating the output vectors is equivalent to performing one long simulation with a vertically-concatenated input.

Input Arguments

`in` — Time-domain input signal

column vectors

Time-domain input signal, specified as column vectors. Specify `in` as an array of *N* column vectors when there are *N* 'InputFrequency' values.

If 'RFInputs' > 1 , 'SLInputs' > 1, and both, then the inputs are passed to the `step` method as arguments `out = rfs(RFin1,RFin2,...,SLin1,SLin2,...)`.

Output Arguments

`out` — Time-domain output signal

column vectors

Time-domain output signal, returned as column vectors. `out` is returned as an array of *N* column vectors when there are *N* 'OutputFrequency' values to be computed.

If 'RFOutputs' > 1, 'SLOutputs' > 1,and both, then the outputs are returned by the `step` method is `[RFout1,RFout2,...,SLout1,SLout2,...] = rfs(in)`.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to `rfsystem`

open_system	Open RF Blockset model created using rfsystem
save_system	Save RF Blockset model created using rfsystem
close_system	Close RF Blockset model window created using rfsystem
hide_system	Hide RF Blockset model window created using rfsystem
load_system	Load RF Blockset model to memory

Common to All System Objects

step	Run System object algorithm

release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Perform Circuit Envelope Simulation on RF Receiver

Design an RF receiver to perform circuit envelope simulation.

Create fifth- and seventh-order bandpass RF filters.

```
f1 = rffilter('ResponseType','Bandpass','FilterOrder',5, ...
        'PassbandFrequency',[4.85 5.15]*1e9);
f2 = rffilter('ResponseType','Bandpass','FilterOrder',7, ...
        'PassbandFrequency',[10 130]*1e6);
```

Create two amplifier objects with `3` dB and `5` dB gain, respectively.

```
a1 = amplifier('Gain',3,'NF',1.53,'OIP3',35);
a2 = amplifier('Gain',5,'NF',8,'OIP3',37);
```

Create a modulator with a local frequency of `4.93` GHz.

```
d = modulator('Gain',0,'NF',4,'OIP3',50,'LO',4.93e9, ...
        'ConverterType','Down');
```

Design an RF receiver with the budget elements at an input frequency of `5` GHz, an available input power of `-30` dBm, and a bandwidth of `200` MHz.

```
rfb = rfbudget([f1 a1 d f2 a2],5e9,-30,200e6);
```

Create an RF system for the RF receiver using the `rfbudget` object.

```
rfs = rfsystem(rfb);
```

Specify 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.

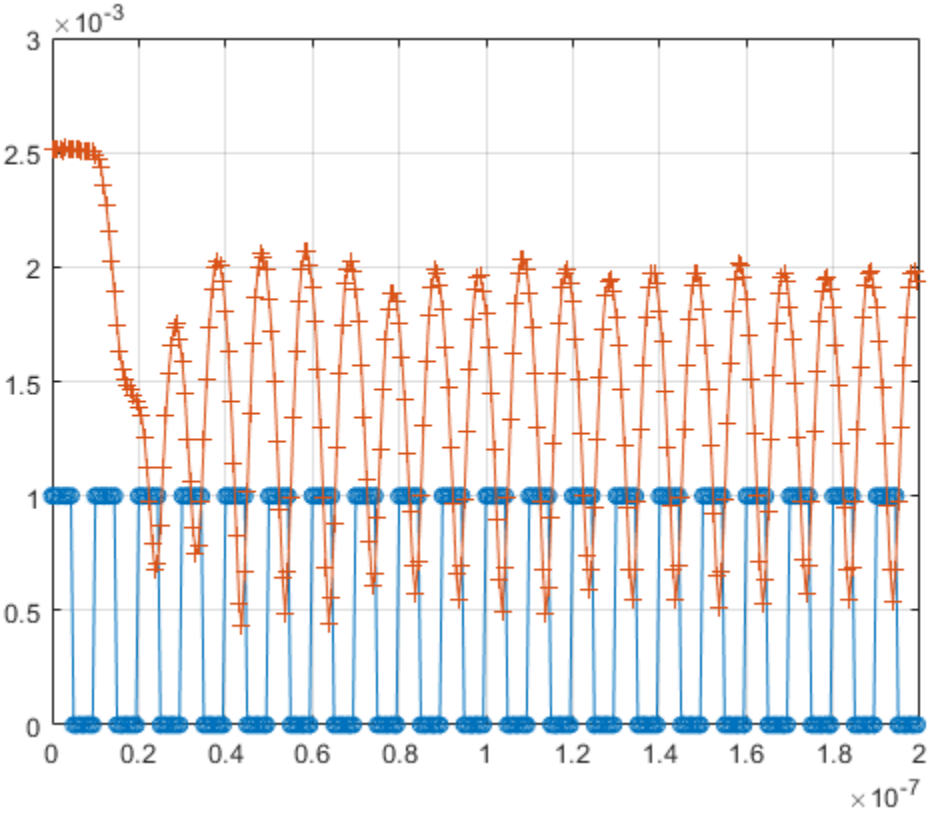
```
out = rfs(in);
out = [out; rfs(in)];
```

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
```

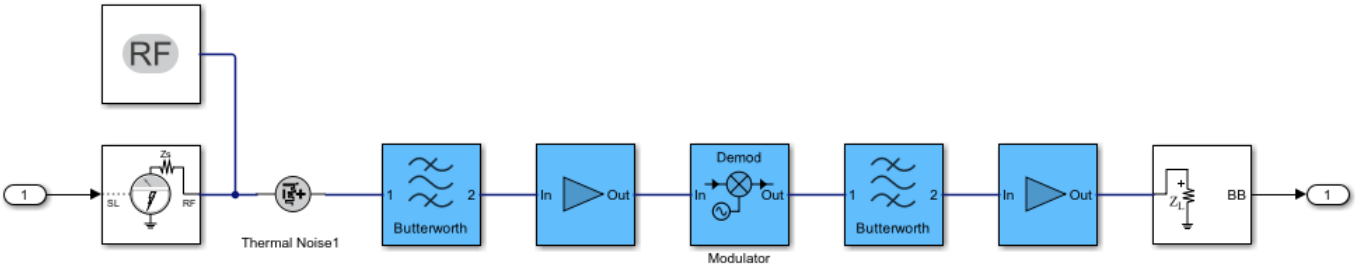


Release system resources and turn off fast restart.

```
release(rfs)
```

Open an RF Blockset model of the designed RF system using the `open_system` object function.

```
open_system(rfs)
```



Design RF-RF, IQ-RF, RF-IQ, and IQ-IQ Architectures

Design four different chain architectures using an RF System object.

Create an input column vector.

```
in = (1:8)';
```

Design RF-RF Architecture

Create an `rfbudget` object using an `amplifier` object.

```
a = amplifier;
```

Calculate the RF budget of the amplifier at an input frequency of `5` GHz, an available input power of `−30` dBm, and a bandwidth of `10` KHz.

```
rfb = rfbudget(a,5e9,-30,10e3);
```

Create an RF system using the `rfbudget` object.

```
rfs = rfsystem(rfb);
```

Create an RF-RF architecture using the input column vector.

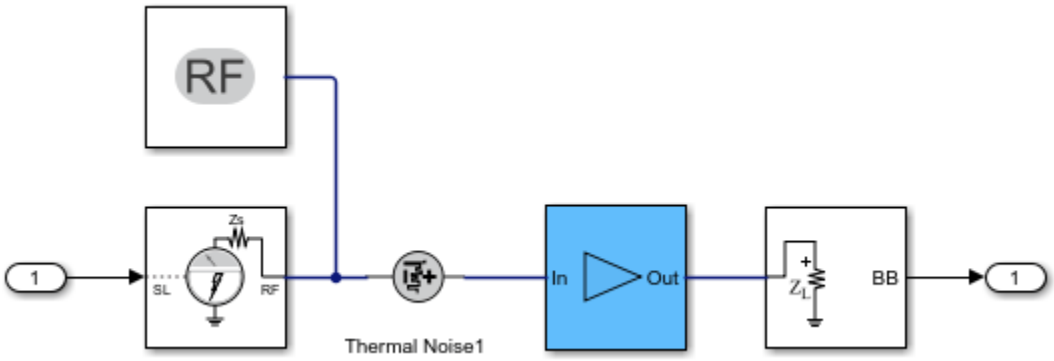
```
out0 = rfs(in);
```

Release system resources and turn off fast restart.

```
release(rfs)
```

Open an RF Blockset model of the RF system.

```
open_system(rfs)
```



Design IQ-RF Architecture

Use a `modulator` object with an up converter to create an `rfbudget` object.

```
u = modulator('ConverterType','Up','LO',1e9);
```

Calculate the RF budget of the modulator at an input frequency of `0` GHz, an available input power of `−30` dBm, and a bandwidth of `10` KHz.

```
rfb2 = rfbudget(u,0,-30,10e3);
```

Create an RF system using the `rfbudget` object.

```
rfs2 = rfsystem(rfb2);
```

Create an IQ-RF architecture using the input column vector.

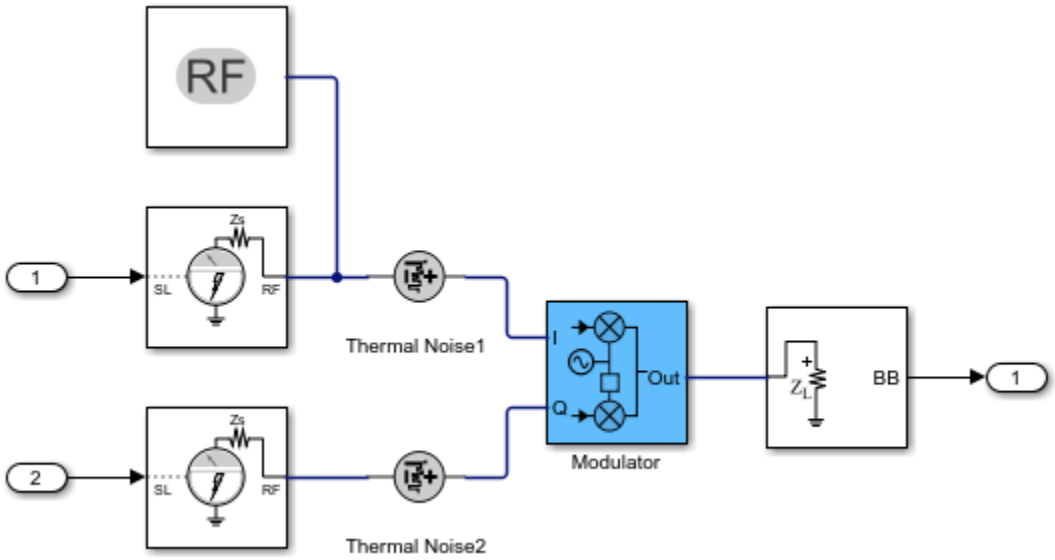
```
inI = in;
inQ = in;
out = rfs2(inI,inQ);
```

Release system resources and turn off fast restart.

```
release(rfs2)
```

Open an RF Blockset model of the RF system.

```
open_system(rfs2)
```



Design RF-IQ Architecture

Use a `modulator` object with a down converter to create an `rfbudget` object.

```
d = modulator('ConverterType','Down','LO',1e9);
```

Calculate the RF budget of the modulator at an input frequency of `1` GHz, an available input power of `–30` dBm, and a bandwidth of `10` KHz.

```
rfb3 = rfbudget(d,1e9,-30,10e3);
```

Create an RF system using the `rfbudget` object.

```
rfs3 = rfsystem(rfb3);
```

Create an RF-IQ architecture using the input column vector.

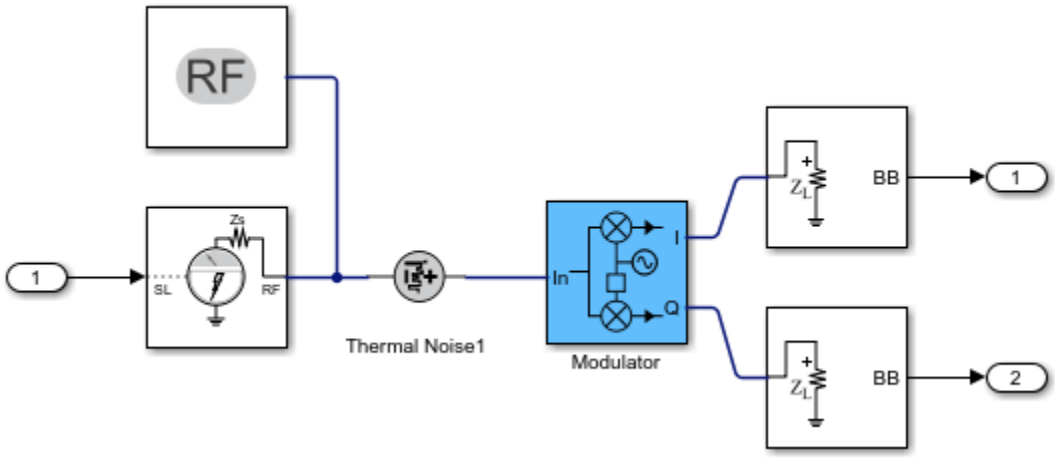
```
[outI,outQ] = rfs3(in);
```

Release system resources and turn off fast restart.

```
release(rfs3)
```

Open an RF Blockset model of the RF system.

```
open_system(rfs3)
```



Design IQ-IQ Architecture

Create an `rfbudget` object using an `amplifier` object.

```
a1 = amplifier;
```

Calculate the RF budget of the amplifier at an input frequency of `0` GHz, an available input power of `-30` dBm, and a bandwidth of `10` KHz.

```
rfb4 = rfbudget(a1,0,-30,10e3);
```

Create an RF system using the `rfbudget` object.

```
rfs4 = rfsystem(rfb4);
```

Create an IQ-IQ architecture using the input column vector.

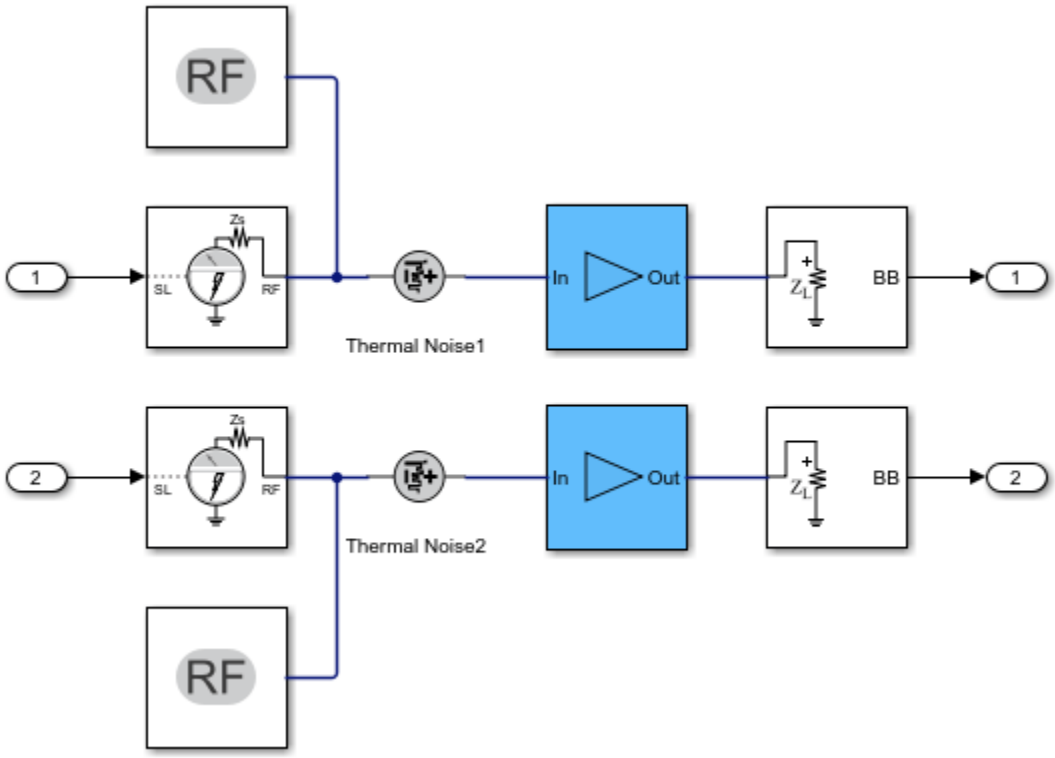
```
[outI2,outQ2] = rfs4(inI,inQ);
```

Release system resources and turn off fast restart.

```
release(rfs4)
```

Open an RF Blockset model of the RF system.

```
open_system(rfs4)
```



Model MISO Receiver and SIMO Transmitter Systems

Create a fifth-order bandpass RF filter.

```
f1 = rffilter('ResponseType','Bandpass','FilterOrder',5, ...
   'PassbandFrequency',[4.85 5.15]*1e9);
```

Create an amplifier with 3 dB gain.

```
a1 = amplifier('Gain',3,'NF',1.53,'OIP3',35);
```

Create a modulator with a local frequency of 4.93 GHz.

```
d = modulator('Gain',0,'NF',4,'OIP3',50,'LO',4.93e9, ...
   'ConverterType','Down');
```

Create a seventh-order bandpass RF filter.

```
f2 = rffilter('ResponseType','Bandpass','FilterOrder',7, ...
   'PassbandFrequency',[10 130]*1e6);
```

Create another amplifier with a 3 dB gain.

```
a2 = amplifier('Gain',5,'NF',8,'OIP3',37);
```

Design an RF receiver with the budget elements at an input frequency of 5 GHz, an available input power of –30 dBm, and a bandwidth of 200 MHz.

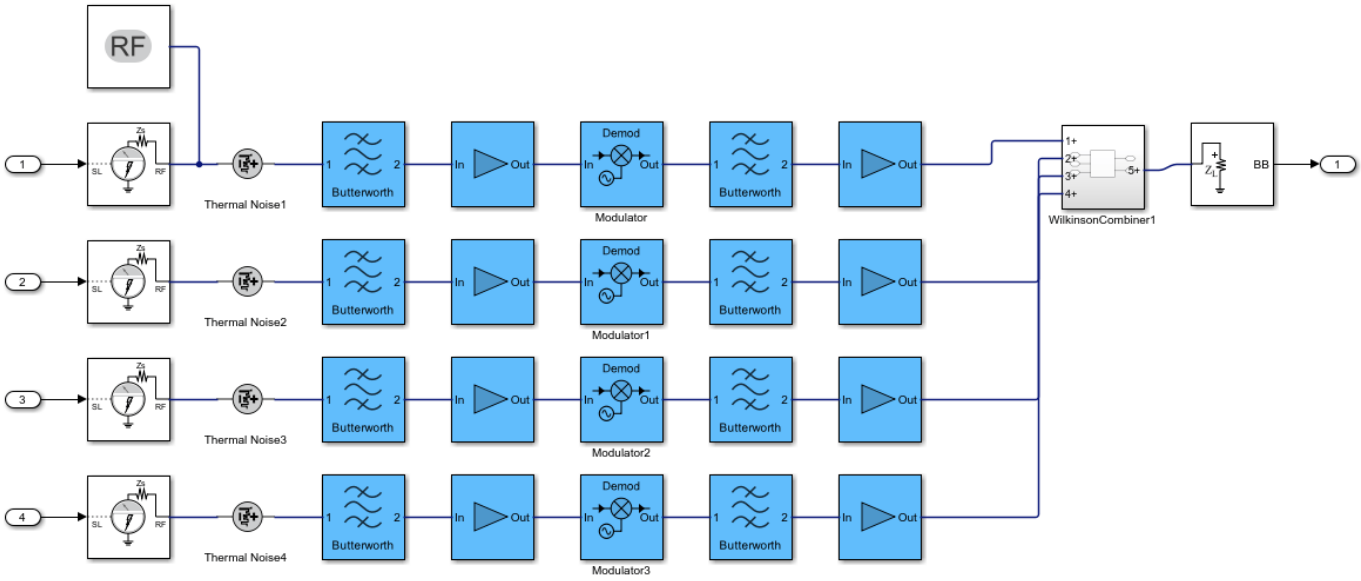
```
b = rfbudget([f1 a1 d f2 a2],5e9,-30,200e6);
```

Duplicate the budget chain into a multiple-input single-output receiver (MISO) system with four branches.

```
rfs = rfsystem(b,Rx=4);
```

Open the underlying RF system model to inspect the MISO receiver

```
open_system(rfs)
```




```
set_param(bdroot,'ZoomFactor','FitSystem')
```

Create four inputs to the MISO receiver. The pulsed carrier square wave input is time-sliced into four pieces.

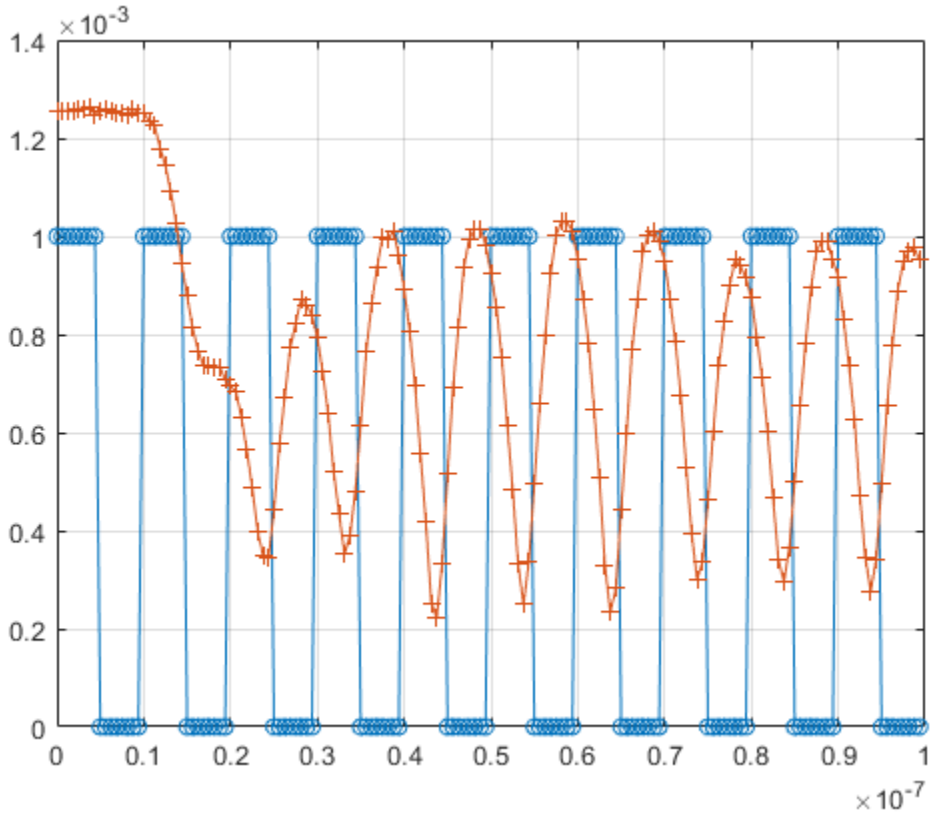
```
in1 = [1e-3*[1;1;0;0;0;0;0;0]; zeros(8,1)] .* ones(1,10);
in1 = in1(:);
in2 = [1e-3*[0;0;1;1;0;0;0;0]; zeros(8,1)] .* ones(1,10);
in2 = in2(:);
in3 = [1e-3*[0;0;0;0;1;1;0;0]; zeros(8,1)] .* ones(1,10);
in3 = in3(:);
in4 = [1e-3*[0;0;0;0;0;0;1;1]; zeros(8,1)] .* ones(1,10);
in4 = in4(:);
out = rfs(in1,in2,in3,in4);
```

Release system resources and turn off fast restart.

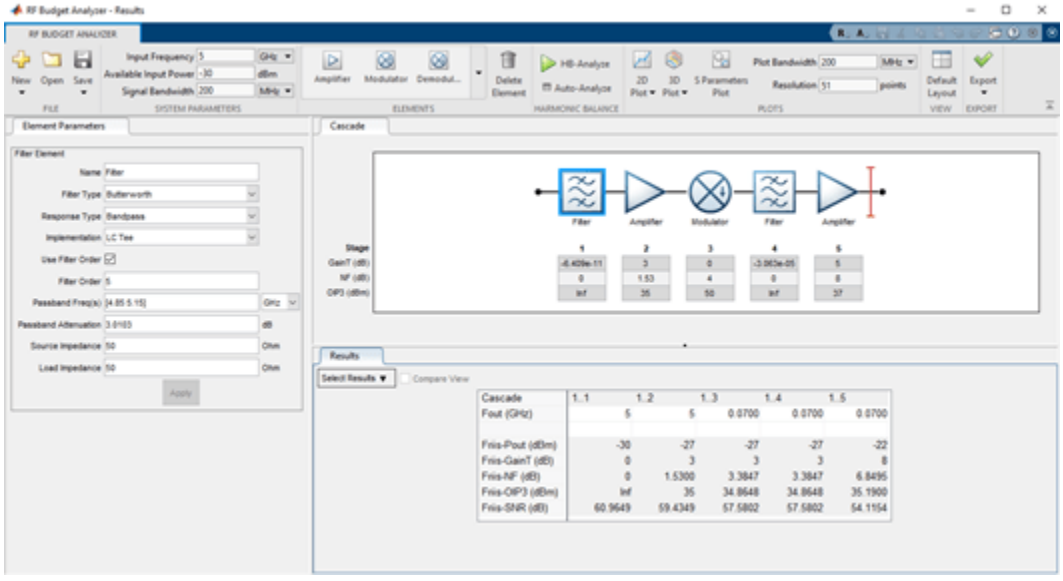
```
release(rfs)
reset(rfs)
```

Plot the circuit envelope simulated result.

```
t = rfs.SampleTime*(0:length(out)-1);
plot(t,in1+in2+in3+in4,'-o',t,abs(out),'-+')
grid on
```



Type `rfBudgetAnalyzer(rfs)` command at the command line to open the MISO receiver in the **RF Budget Analyzer** app to visualize the initial budget chain `b`.

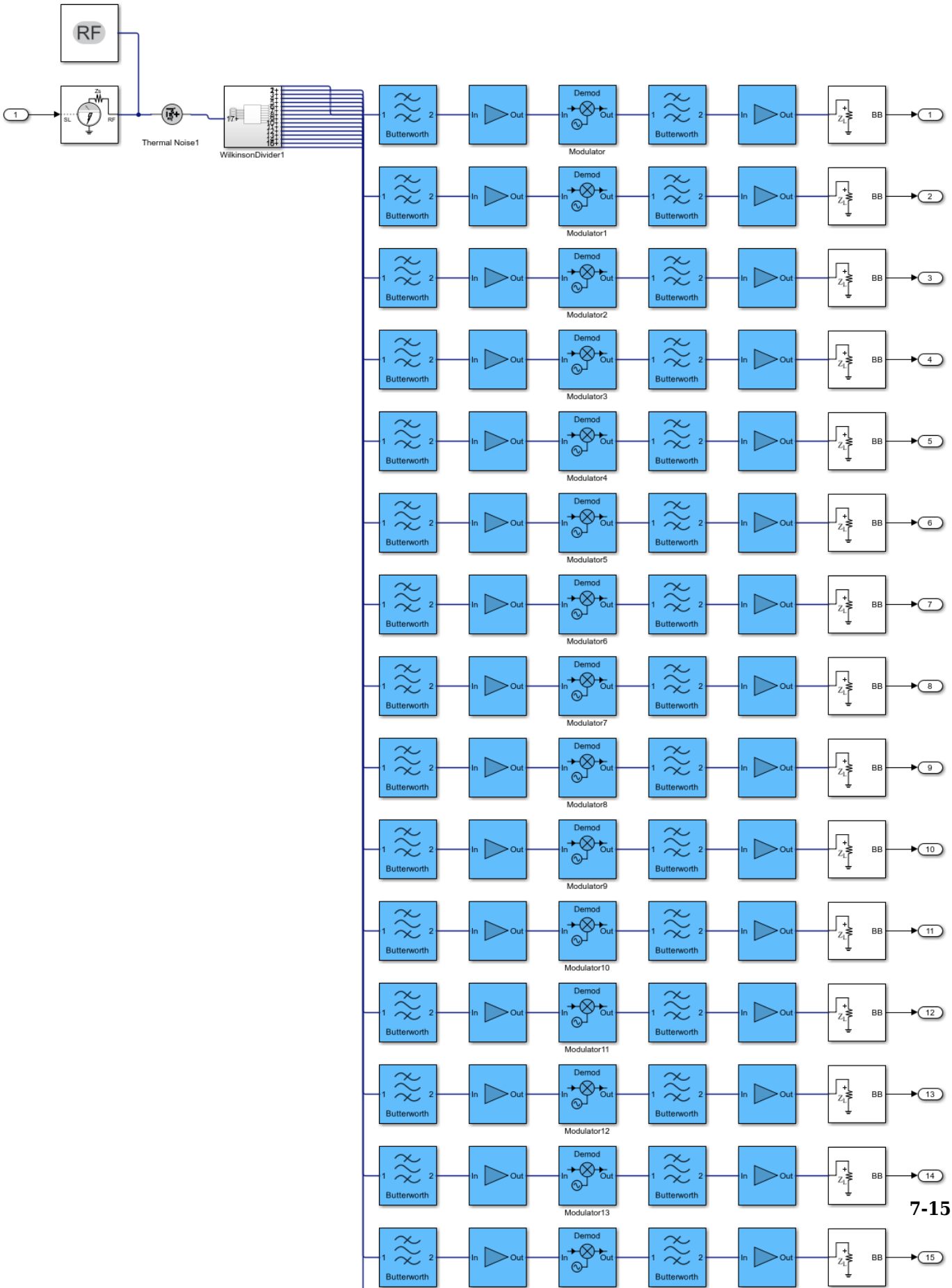


Duplicate the budget chain into a single-input multiple-output (SIMO) array system with sixteen branches.

```
rfs2 = rfsystem(b,Tx=16);
```

Open the underlying RF system model to inspect the SIMO receiver

```
open_system(rfs2)
```


RF
1
SL
RF
Thermal Noise1
WilkinsonDivider1
Butterworth
In
Out
Demod
Modulator
Modulator1
Modulator2
Modulator3
Modulator4
Modulator5
Modulator6
Modulator7
Modulator8
Modulator9
Modulator10
Modulator11
Modulator12
Modulator13
BB
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Name Your RF Blockset Model

Create a fifth-order bandpass RF filter.

```
f1 = rffilter('ResponseType','Bandpass','FilterOrder',5,'PassbandFrequency',[4.85 5.15]*1e9);
```

Create an amplifier with the gain of 3 dB, noise figure of 1.53 dB, and OIP3 of 35 dBm.

```
a1 = amplifier('Gain',3,'NF',1.53,'OIP3',35);
```

Create an `rfbudget` object using these elements at an input frequency of 5 GHz, an available input power of -30 dBm, and a bandwidth of 200 MHz.

```
rfb = rfbudget([f1 a1],5e9,-30,200e6);
```

Create an RF system using the rfbdget object. Name the model and save the RF Blockset model.

```
rfs = rfsystem(rfb,'ModelName','myRFSystem_Model')
save_system(rfs);
```

```
rfs =

  rfsystem with properties:

          ModelName: 'myRFSystem_Model'
         SampleTime: 6.2500e-10
    InputFrequency: 5.0000e+09
   OutputFrequency: 5.0000e+09
           RFInputs: 1
          RFOutputs: 1
```

Perform HB Analysis on RF Receiver Designed Using `rfsystem`

Design an RF receiver using the `rfsystem` system object. View the object in the the **RF Budget Analyzer** app to perform harmonic balance (HB) analysis.

Create fifth- and seventh-order bandpass RF filters.

```
f1 = rffilter('ResponseType','Bandpass','FilterOrder',5, ...
        'PassbandFrequency',[4.85 5.15]*1e9);
f2 = rffilter('ResponseType','Bandpass','FilterOrder',7, ...
        'PassbandFrequency',[10 130]*1e6);
```

Create two amplifier objects with 3 dB and 5 dB gain, respectively.

```
a1 = amplifier('Gain',3,'NF',1.53,'OIP3',35);
a2 = amplifier('Gain',5,'NF',8,'OIP3',37);
```

Create a modulator with a local frequency of 4.93 GHz.

```
d = modulator('Gain',0,'NF',4,'OIP3',50,'LO',4.93e9, ...
        'ConverterType','Down');
```

Design an RF receiver with the budget elements at an input frequency of 5 GHz, an available input power of – 30 dBm, and a bandwidth of 10 MHz.

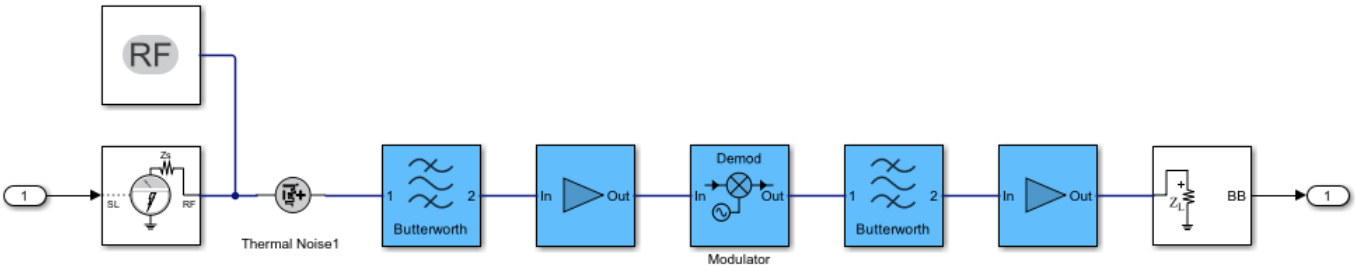
```
rfb = rfbudget([f1 a1 d f2 a2],5e9,-30,10e6);
```

Create an RF system for the RF receiver using the `rfbudget` object.

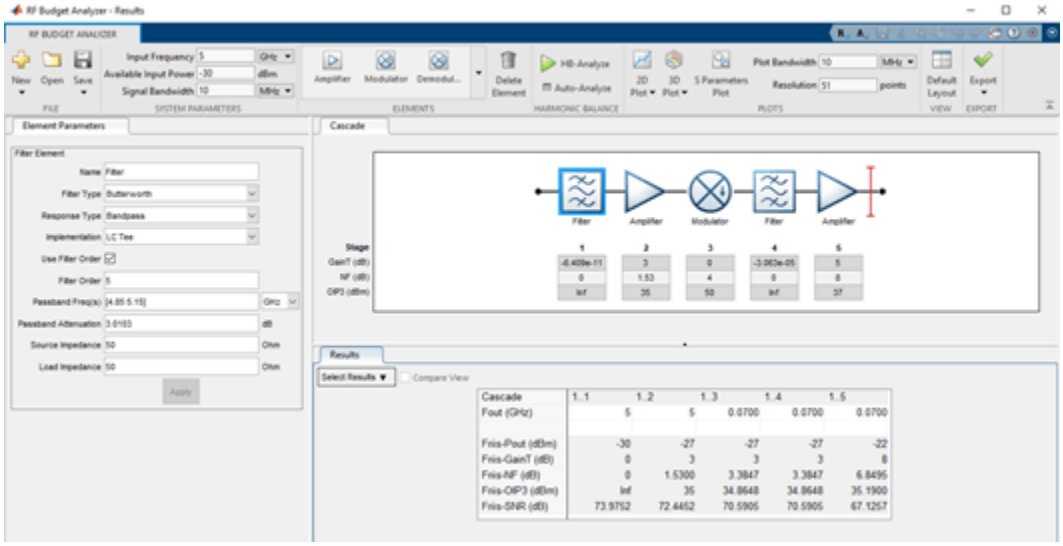
```
rfs = rfsystem(rfb);
```

Open an RF Blockset model of the designed RF system using the `open_system` object function.

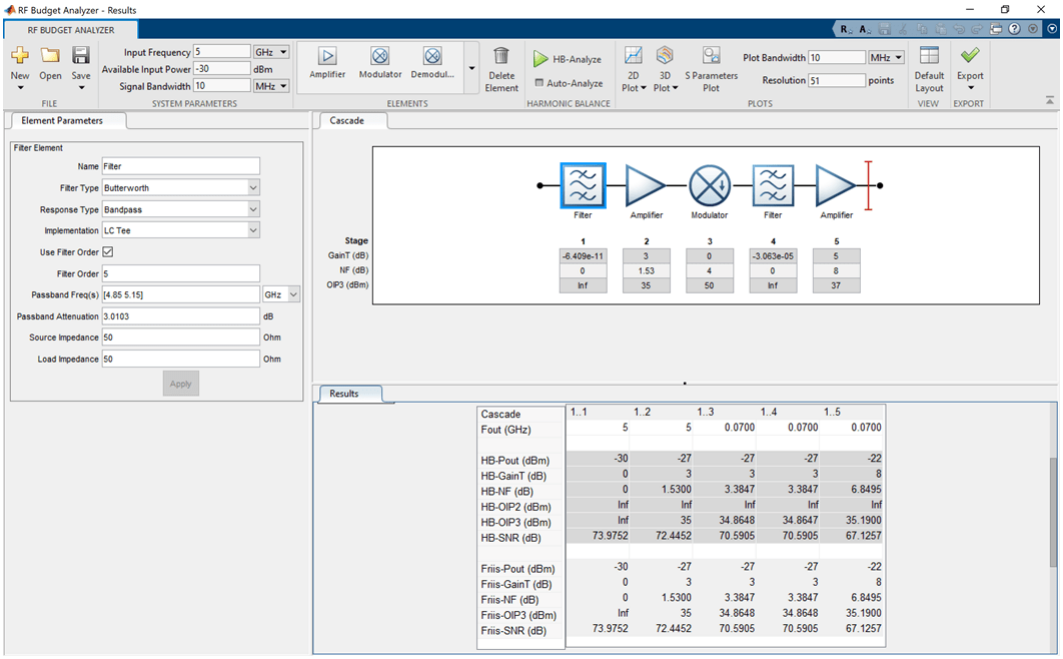
```
open_system(rfs)
```



Type rfBudgetAnalyzer(rfs) command at the MATLAB® command line to open this RF system in the **RF Budget Analyzer** app.



To conduct HB analysis in the app, click the **HB-Analyze** button.



Version History

Introduced in R2021a

Enhancements to `rfsystem`

Updates to the `rfsystem` System object now enable you to:

- Model multiple-input and single-output (MISO) and single-input and single-output (SIMO) systems by specifying number of input and output RF chains using the 'Rx' and 'Tx' properties.
- Add Simulink inports and outports to your RF Blockset model by specifying number of Simulink input and output ports using 'SLInputs' and 'SLOutputs' properties.
- Export your RF system to the **RF Budget Analyzer** app, type `rfBudgetAnalyzer(rfs)` at the MATLAB command line.
- Use the `load_system` function specific to the System object to add the system model to memory.
- Use the `close_system` function specific to the System object to close the system model window.

See Also

`rfbudget` | `nport` | `modulator` | `amplifier`

Topics

“RF Receiver Modeling for LTE Reception” (RF Blockset)
“Circuit Envelope Simulation at MATLAB Command Line”